A centralized and distributed multi-robot system for 3D surface reconstruction of unknown environments

Guillaume Hardouin

UNIVERSITÉ de Picardie Jules Verne

Thèse de Doctorat

Mention Sciences pour l'Ingénieur

Spécialité Robotique

présentée à l'École Doctorale en Sciences, Technologie, Santé (ED 585)

de l'Université de Picardie Jules Verne

par

Guillaume HARDOUIN

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

A centralized and distributed multi-robot system for 3D surface reconstruction of unknown environments

Soutenue le 22 mars 2022, après avis des rapporteurs, devant le jury d'examen :

Paolo Robuffo Giordano	Directeur de Recherche CNRS, IRISA, Rennes	Président
Ouiddad Labbani-Igbida	Professeure des Universités ENSIL-ENSCI, Limoges	Rapporteur
Rudolph Triebel	Professeur des Universités DLR, TUM, Munich	Rapporteur
Isabelle Fantoni	Directrice de Recherche CNRS, LS2N, Nantes	Examinateur
Julien Marzat	Ingénieur de Recherche ONERA-DTIS, Palaiseau	Examinateur
El Mustapha Mouaddib	Professeur des Universités UPJV MIS, Amiens	Directeur de thèse
Fabio Morbidi	Maître de Conférences UPJV MIS, Amiens	Co-encadrant
Julien Moras	Ingénieur de Recherche ONERA-DTIS, Palaiseau	Co-encadrant

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

ONERA
THE FRENCH AEROSPACE LAB

Région Hauts-de-France

Remerciements

Je souhaite tout d'abord remercier les professeurs Ouiddad Labbani-Igbida et Rudolf Triebel d'avoir accepté de rapporter ce manuscrit de thèse. Je remercie également Isabelle Fantoni d'avoir accepté de faire partie de mon jury de thèse en tant qu'examinatrice. Enfin, je remercie Paolo Robuffo Giordano d'avoir accepté de présider le jury de thèse.

Je remercie la région Hauts-de-France et l'ONERA d'avoir financé ma thèse.

Je remercie Hélène Piet-Lahanier pour son soutien, ses conseils, d'avoir cru en moi et permis de rejoindre l'ONERA en stage, et pour le suivi de ma thèse. Je remercie également Claude Pégard pour ses conseils et le suivi du bon déroulement de ma thèse.

Ensuite, je souhaite remercier mes encadrants. Merci à El Mustapha Mouaddib d'avoir dirigé ma thèse. Sa passion pour le patrimoine et la robotique, son souhait de transmission et ses qualités humaines m'ont permis de réaliser ma thèse dans un cadre idéal. Merci Fabio Morbidi pour son soutien indeffectible, sa rigueur scientifique, ses longues relectures et sa bonne humeur. Merci à Julien Moras pour son soutien, sa grande disponibilité et nos nombreuses discussions. Enfin, je remercie Julien Marzat pour ses conseils avisés, sa polyvalence et sa réactivité. Merci pour nos discussions scientifiques, techniques ou personnelles, votre engagement constant, les opportunités d'expérimentations qui ont constitué un facteur extrêmement motivant : j'ai énormement appris à vos côtés !

Je souhaite remercier l'équipe DTIS/IVA, dans laquelle j'ai pu réaliser mes trois années de thèse. Merci aux membres de l'équipe, Guy, Martial, Anthelme pour nos discussions photo, Fred pour la plongée, Philippe pour les randonnées, Patrick les debriefs vélo, Stéphane, Pierre, Adrien, Fabrice S., Fabrice J., Benjamin, Alexandre E., Alexandre A., Alexandre B. pour nos discussions du matin, Elise, Flora, Aurélien, Annie, Alain, Gilles. Je remercie particulièrement Anthelme pour sa disponibilité et son travail sur les plateformes robotiques, Alexandre E. pour les debugs informatiques et subtilités ROS et UNIX et Martial pour sa réactivité à régler les problèmes matériels et sa gentillesse. Merci également à Sylvain Bertrand de m'avoir initié à la recherche et à l'enseignement. Merci à Fabrice S. pour son bagout si agréable et son extrême efficacité pour résoudre des situations administratives complexes. Merci à Florence Marie pour sa patience, son implication et sa réactivité. Ce fut un plaisir de travailler avec vous !

Merci aussi à tous les doctorants, stagiaires et apprentis que j'ai pu côtoyer. Merci à Rodolphe, Marcela, Maxime F., Rodrigo, Soufiane, Pierre, Vincent, Sergio, Camille S., Baptiste, pour les bons moments passés ensemble. Merci à Jordan Caracotte pour ses conseils avisés lors de mes venues ponctuelles à Amiens. Merci à la promotion 2021 : Guillaume V.R., Javiera, Alexis, Louis, Benjamin, Camille P., Esteban et Enzo. Merci

aux doctorants arrivés plus récemment, Rémy, Gaston, Laurane, Maxime C., Nathan et Thomas, Marius et Quentin. Je me souviendrai de nos parties de tarot et tamalou endiablées, nos discussions et rigolades autour d'un café en salle de pause et des occasionnelles blagues. Je vous souhaite le meilleur pour la suite !

Je remercie chaleureusement mes niçois : Gildas, Pierre-Michel, Mathilde, Damien, Bastien, Marie, Adrien, Kévin et Aurore pour leurs grands soutiens. Vous êtes loin des yeux mais tellement près du cœur !

Je remercie ma famille pour m'avoir toujours encouragé dans mes choix et sans faillir, en particulier ma mère Frédérique, mon père Philippe et mon frère Maxime.

Enfin, je remercie infiniment Margot de m'avoir soutenu pendant ces trois années et pour ses encouragements quotidiens.

Encore merci à tous ! Cette thèse, c'est aussi grâce à vous !

Contents

Acronyms **xi**

Introduction **1**

1 State of the art **7**
1.1 Introduction . . . 8
1.2 Autonomous robots: basic notions . . . 8
1.2.1 Guidance, navigation and control . . . 9
1.2.2 Pose estimation via visual-inertial odometry . . . 9
1.2.3 Cooperation among multiple robots . . . 11
1.2.4 Centralized, decentralized and distributed architectures . . . 11
1.3 3D mapping . . . 12
1.3.1 Surface shape representation . . . 13
1.3.2 Volumetric representations . . . 15
1.3.3 TSDF mapping . . . 17
1.4 Path planning methods . . . 20
1.4.1 Graph-search algorithms . . . 20
1.4.2 Potential fields method . . . 22
1.4.3 Sampling-based planners . . . 23
1.5 Informative planning for 3D reconstruction . . . 27
1.5.1 Next-Best-View planning . . . 27
1.5.2 Surface inspection . . . 29
1.5.3 Volume exploration . . . 30
1.5.4 Mixed approaches . . . 32
1.5.5 Multi-robot 3D modelling . . . 33
1.6 Conclusion . . . 33

2 Surface reconstruction of unknown environments with a single robot **35**
2.1 Introduction . . . 35
2.2 Problem formulation . . . 36
2.3 System overview . . . 38
2.4 Perception . . . 39
2.4.1 Surface-based mapping . . . 39
2.4.2 ISE extractor and viewpoint generation . . . 41

2.4.3 Cluster of configurations . . . 41
2.5 Single-robot planning . . . 42
2.5.1 Graph formulation . . . 42
2.5.2 Inspection problem . . . 44
2.5.3 Low-level planner . . . 45
2.6 Numerical experiments . . . 47
2.6.1 Simulation setup . . . 47
2.6.2 Metrics . . . 50
2.6.3 Choice of penalty terms . . . 50
2.6.4 Results and discussion . . . 51
2.7 Conclusion . . . 53

3 Surface reconstruction: centralized multi-robot architecture 55
3.1 Introduction . . . 55
3.2 Problem formulation . . . 56
3.3 System overview . . . 57
3.4 Perception . . . 59
3.4.1 Surface-based mapping . . . 59
3.4.2 ISE extractor, viewpoint generation and clustering . . . 60
3.5 Centralized Planning . . . 60
3.5.1 Inspection problem . . . 60
3.5.2 Low-level planner . . . 64
3.6 Numerical experiments . . . 65
3.6.1 Multi-robot nearest neighbour planner . . . 66
3.6.2 Simulation setup . . . 66
3.6.3 Metrics . . . 67
3.6.4 Results and discussion . . . 68
3.7 Conclusion . . . 71

4 Surface reconstruction: distributed multi-robot architecture 73
4.1 Introduction . . . 73
4.2 Problem formulation . . . 74
4.3 System overview . . . 75
4.4 Perception . . . 78
4.4.1 Surface-based mapping . . . 78
4.4.2 ISE extractor, viewpoint generation and clustering . . . 80
4.5 Distributed Planning . . . 80
4.5.1 Inspection problem . . . 80

4.5.2 Low-level planner . . . 83
4.6 Numerical experiments . . . 86
4.6.1 Simulation setup . . . 86
4.6.2 Metrics . . . 87
4.6.3 Results and discussion . . . 88
4.7 Conclusion . . . 91

5 Real-world experiments **93**
5.1 Introduction . . . 93
5.2 Robots . . . 94
5.2.1 Hardware . . . 94
5.2.2 Software . . . 95
5.3 Environments . . . 96
5.4 Metrics . . . 98
5.5 Results . . . 100
5.6 Conclusion . . . 103

Conclusion **105**

Résumé français **113**

6 Un système multi-robots déployable en centralisé ou distribué pour la reconstruction 3D d'environnements inconnus à large échelle **115**
6.1 Introduction . . . 115
6.2 État de l'art . . . 116
6.2.1 Cartographie volumétrique . . . 116
6.2.2 Planification mono-robot . . . 117
6.2.3 Planification multi-robots . . . 118
6.3 Formulation du problème . . . 119
6.4 Aperçu des contributions . . . 121
6.5 Simulations . . . 122
6.6 Expérimentations en environnements réels . . . 124
6.7 Conclusion . . . 125

Appendices **127**

A Kinematics and robotics **129**

A.1 Pose of a rigid body . . . 129
A.1.1 Rotations matrices . . . 129
A.1.2 Rigid-body motion . . . 130
A.2 Mobile robotic platform . . . 131
A.2.1 Robot parametrization . . . 131
A.2.2 Stereo vision . . . 131
A.2.3 Camera and IMU calibration . . . 135
A.2.4 ROS: the Robot Operating System . . . 136

B Graph theory **139**
B.1 Definitions . . . 139
B.1.1 Adjacency relation . . . 139
B.1.2 Special graphs . . . 140
B.2 Traveling Salesman Problem and its variants . . . 141
B.2.1 Formulation . . . 141
B.2.2 Numerical solvers . . . 142

Bibliography **145**

List of Figures

1 Amiens cathedral: example of 3D reconstruction 1

1.1 Surface shape representation 14
1.2 Distance computation in the TSDF 18
1.3 MarchingCubes: Triangulated cubes 19
1.4 Reconstructed mesh from a depth map using a TSDF volume 19
1.5 Behaviour of Dijkstra's algorithm 21
1.6 Examples of potential fields for path planning with obstacles 22
1.7 Illustration of the PRM planner 24
1.8 Illustration of the LazyPRM* planner 25
1.9 Illustration of the RRT algorithm 26
1.10 2D illustration of frontier-based NBV planning for surface inspection . . 29
1.11 2D illustration of NBV planning for volume exploration 31
1.12 2D illustration of NBV planning for surface inspection using volumetric mapping 34

2.1 General flowchart of the single-robot architecture 38
2.2 ISEs and viewpoint configurations 40
2.3 Two-dimensional representation of a weighted directed graph with three clusters 43
2.4 Simulation stack for a single robot 48
2.5 Gazebo simulation environments 48
2.6 Numerical experiments for the single-robot system 52

3.1 General flowchart of the centralized multi-robot architecture 58
3.2 Flowchart of centralized mapping module for a team of 3 robots 59
3.3 Greedy allocation planner 63
3.4 Centralized simulation stack for a fleet of 3 UAVs 67
3.5 Numerical experiments for the centralized architecture 70

4.1 General flowchart of the distributed multi-robot architecture 76
4.2 Examples of communications graphs for the distributed multi-robot system 77
4.3 Flowchart of distributed mapping module for a team of 3 robots 79
4.4 RedundancyCheck function 85
4.5 Distributed simulation stack for a fleet of 3 UAVs 86
4.6 Numerical experiments for the distributed architecure 89

4.7 Top views of reconstructed meshes for Powerplant 90

5.1 Hardware architecture 94
5.2 Real-world environments 97
5.3 Real-world experiments 99
5.4 Some causes and consequences of the odometry drift 101
5.5 3D reconstruction of the Test arena with two robots 102
5.6 3D reconstruction of the Parking lot with two robots 102
5.7 Ground truth versus offline reconstruction of Parking lot 104

6.1 Extended astract: Schémas des architectures multi-robots 121
6.2 Extended astract: Résultats des simulations 123
6.3 Extended astract: Progression de la reconstruction 3D du Parking 124
6.4 Extended astract: Reconstruction 3D du Parking 125

A.1 Scheme of a camera, mounted on a robot 132
A.2 Scheme of a stereo-rig (rectified configuration) 134
A.3 The Robot Operating System, ROS 137

B.1 Examples of graphs 140
B.2 Steps of the 2-opt heuristic for the TSP 143

List of Tables

2.1 Parameters used in the numerical experiments 49
2.2 Tuning of the penalty terms: Powerplant 51
2.3 Tuning of the penalty terms: SoL 51
2.4 Results of the single-robot numerical experiments 52

3.1 Parameters used in the TSGA numerical experiments 68
3.2 Results of the TSGA numerical experiments on Powerplant 69
3.3 Results of the TSGA numerical experiments on SoL 69
3.4 Comparison between the TSGA and NNB planner 71

4.1 Parameters used in the dist-TSGA numerical experiments 88
4.2 Results of the dist-TSGA numerical experiments on Powerplant 89

5.1 Software modules of centralized architecture 96
5.2 Software modules of distributed architecture 96
5.3 Parameters used in the real-world experiments 97
5.4 Results of the real-world experiments 100

Acronyms

ATSP Asymmetric Traveling Salesman Problem 45

maxATSP maximum Open Asymmetric Traveling Salesman Problem 44

CAD Computer Aided Design 13

CPU Central Processing Unit 15

CSLAM Collaborative-SLAM 109

DCNNs Deep Convolutional Neural Networks 110

DFJ Dantzig–Fulkerson–Johnson 141

EDT Euclidean Distance Transform 15

ESDF Euclidian Signed Distance Field 16

FOV Field-of-View 37

GNC Guidance, Navigation and Control 8

GNSS Global Navigation Satellite Systems 108

GPs Gaussian Processes 16

GPU Graphics Processing Unit 15

GS Ground Station 95

ICP Iterative Closest Point 28

ILP Integer Linear Program 141

IMU Inertial Measurement Unit 135

ISE Incomplete Surface Element 37

LDIs Layered Depth Images 13

LiDAR Light Detection And Ranging 1

LKH Lin-Kernighan-Helsgaun 45

OG Occupancy Grid 15

OMPL Open Motion Planning Library 47

MAVs Micro Aerial Vehicles 16

MPC Model Predictive Control 48

MCTS Monte-Carlo Tree Search 33

MTZ Miller–Tucker–Zemlin 141

M3C2 Multiscale Model to Model Cloud Comparison 50

NBV Next-Best-View 3

NNB Nearest Neighbor 65

NTP Network Time Protocol 95

NURBS Non-Uniform Rational Basis Splines 14

PRM Probabilistic RoadMap 8

RMSE Root-Mean-Square Error 50

ROS the Robot Operating System 5

RRT Rapidly-exploring Random Tree 8

SLAM Simultaneous Localization And Mapping 14

SRM Semantic Road Map 31

TSDF Truncated Signed Distance Function 8

TSGA TSP-Greedy Allocation 34

dist-TSGA Distributed TSP-Greedy Allocation 80

TSP Traveling Salesman Problem 4

UAVs Unmanned Aerial Vehicles 2

VO Visual Odometry 9

Introduction

As research in multi-agent systems is developing at a rapid pace, autonomous co-operative robots have been increasingly used in time-sensitive applications, such as ocean sampling [Leonard *et al.* 2007], cinematography [Alcántara *et al.* 2021], wildlife survey [Shah *et al.* 2020], logistics automation [Digani *et al.* 2015], and mine tunnel exploration [Miller *et al.* 2020], just to name some relevant examples. However, in spite of these recent achievements, we are still a long way from coordinated online exploration and 3D reconstruction of vast, complex, unknown environments (industrial plants, caves, archaeological sites, battlegrounds, etc.), for which only a few solutions are currently available in the literature. Digital technologies provide new means to preserve the cultural heritage and widen the access to knowledge. In particular, the scans realized by high-resolutions lasers (Light Detection And Ranging (LiDAR)) allow to analyze the surface of a building to gather precise information on its geometry and appearance (texture, albedo, etc.): the laser ray sent by the scanner measures the distance to the first object cutting the beam. The collected data, represented as a 3D point-cloud can be used to build digital 3D models for the preservation of the cultural heritage. Another popular technique, photogrammetry, uses multiple images of the same scene from different viewpoints, to generate a 3D model of it. However the scanning process, using laser sensors or cameras remains slow, complex and expensive. In practice, with limited-range sensors, tens or hundreds of measurements are typically needed to ensure that the accuracy of the final 3D model, is spatially homogeneous. Therefore, completing the scan of large-scale buildings containing fine details, as for example the rich Gothic sculptures of Amiens

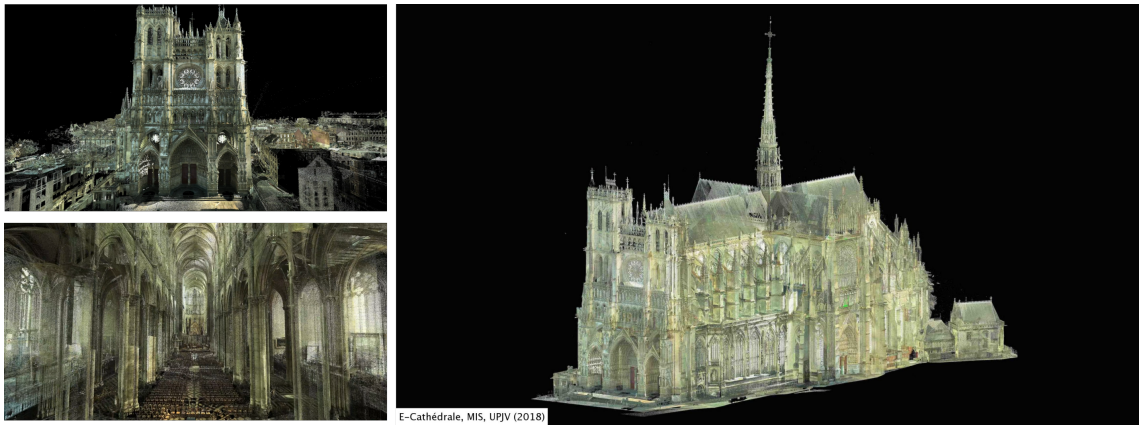


Figure 1: *Amiens cathedral:* example of 3D reconstruction using LiDAR; [top-left] front view of the 3D point-cloud model; [bottom-left] elevated view of the nave; [right] side view.

Cathedral[1] (see Fig 1, [Mouaddib *et al.* 2019]) requires the involvement of a team of specialized technicians working in parallel, sometimes in dangerous conditions, for days or weeks. Moreoever, the offline data fusion of multiple point-clouds can take long hours and it is subject to human failure. Then, a new generation of scanning systems based on mobile robots, may address these challenges, improving efficiency, flexibility and responsiveness.

Scanbot: 3D modelling of cultural heritage with autonomous robots

This PhD thesis has been conducted within the framework of the *ScanBot project*[2] (funded by the Hauts-de-France region and by ONERA DTIS), which aims to develop new functional modules for digital modeling of cultural heritage, in order to make the overall process faster, cheaper, more automatic and accurate, more robust to uncertainty, and safer for the human operators. The idea of the project is to embed sensors, such as cameras or LiDARs on mobile robotic platforms. These robots can be terrestrial (wheeled robots having long-term autonomy and large payloads), and aerial (agile Unmanned Aerial Vehicles (UAVs), able to cover those areas which are inaccessible to the operators). Each robot, dubbed "ScanBot", can team up with the others and shares its workload to speed up and automate the scanning, data fusion and 3D reconstruction processes, for digital cultural heritage.

The ScanBot project aims at both scientific and technological research. In spite of some recent important advancements of 3D scanners, the 3D reconstruction of large-scale environments using high-resolution range sensors is still mostly realized empirically, based on the experience of specialized technicians. Moreover, the whole scanning process (from the data acquisition to the final 3D model of a building) presents some inherent weaknesses, and it is far from being optimized in terms of human, financial and technical resources. The problem needs to be rigorously formalized, for a group of cooperative robots to be able to perform the scanning process autonomously and faster than a single robot, thus reducing human intervention in fastidious and dangerous tasks (transport of heavy and cumbersome material, sensor positioning). The scientifical and technical challenges are numerous:

1. Online fusion of sensor data into a low-quality 3D model, estimating the uncertainties and matching the local models of each robot.

2. Trajectory planning of the robots, in order to minimize the 3D model uncertainty and guarantee uniform accuracy, minimize the completion time of full scans, and

[1] https://home.mis.u-picardie.fr/~ecathedrale/

[2] ScanBot: Scanners Robotisés pour la Numérisation Automatique du Patrimoine (2018-2021)

ensure safe navigation.

3. Completeness of the 3D model, with an automatic detection of the missing parts and the definition of a stopping condition.

4. Coordination of the robot team by local communication and control protocols, managing the computation and network resources. Optimal task assignment within the team according to the robot type (terrestrial or aerial), autonomy and payload.

5. Determination of the minimal number of robots and scans to perform, to ensure the full coverage of a building, according to the prior knowledge of a ground-truth model.

6. Generation of a final complete 3D model.

Objective

Within the framework of the ScanBot project, this PhD thesis covers a large spectrum of topics going from computer vision to motion planning. We aim to coordinate multiple mobile robots equipped with a couple of forward-facing cameras, to perform a complete reconstruction of the scene, while ensuring their safe navigation. More specifically, it consists of designing an end-to-end algorithm and of implementing and testing software and hardware stacks which allow to couple perception with planning. Some modules have been re-adapted from the literature, such as odometry-estimation or trajectory-tracking software.

To do so, the robots in the team use the local measurements from their on-board sensors, and share them to determine the next region to visit and pursue the reconstruction, completing holes or occluded areas until a complete model of the environment is obtained. The proposed online modeling technique, allows to fuse the sensor data into a single 3D map, and to specify the level of completeness of 3D reconstruction (addresses item n°3.). This mapping technique should be fast, accurate and should be able to distinguish the occupied and free space for navigation purposes, which is particularly challenging, in the application context of this PhD thesis, the scene is assumed to be a large, indoor (such as a cathedral's nave, a warehouse or a parking lot), or outdoor (such as a historical building) environment (addresses item n°1.). It is also assumed that there is no prior knowledge of the scene. In the literature, the Next-Best-View (NBV) methods aims to use the knowledge of the environment to drive the reconstruction seeking either for exploration of a predefined volume with volume exploration methods or for the surface completeness of an object with surface inspection methods (addresses item n°2.). The last is a surface-based method and consists in reconstructing an object by generating sensor configura-

tions, directly from the knowledge of the current incomplete surface, represented with point clouds or meshes constituted of facets. The obtained 3D model is accurate but the process relies on strong prior hypothesis to define the sensor configuration space which can be limiting for embedded mobile robot applications. The volume exploration techniques aim to scan a whole volume represented as occupancy cells containing attributes which define if it is known or unknown. The scan of a cell, located in the 3D space, will determine if its space belongs to an object, or if it is empty. The collection of these cells constitutes a volumetric map which models the space. The volume exploration methods provide fast reconstruction of unknown environments, while ensuring safe navigation, but they do not always account for surface completeness of the model. A mixed approach has been developed in the thesis, by coupling the advantages of the volumetric mapping and surface-based planning for 3D reconstruction using a team of robots. The robots are coordinated using a two-level algorithm which assigns sub-areas to visit to each robot of the team, and then computes local paths in the free space using a graph-based planner. More precisely, new 3D reconstruction algorithms are proposed for multiple cooperative robots, that address the *surface inspection problem* in an unknown environment, via a NBV frontier-based planner. A single objective function, which takes the surface representation explicitly into account, is used to assign and plan collision-free paths for the robots (addresses item n°4.). The method is divided into two architectures, a centralized one where all computation are performed on a central unit, and a distributed one where computation are split among the agents. Numerical simulations and real-world experiments show that the proposed architectures are robust against uncertain measurements, and that provide accurate, complete and time-efficient 3D reconstructions by referring to accurate offline reconstructed models (addresses item n°6). In summary, the original contributions of this PhD thesis are the following:

- A unique online volumetric mapping is used for both navigation and mapping.
- A new formulation which directly leverages the 3D surface representation, where the completeness of the model is defined as a stopping criterion for exploration.
- A multi-robot NBV planner to visit viewpoint configurations for surface reconstruction is introduced. In particular, we perform a greedy allocation of configurations and successively solve a Traveling Salesman Problem (TSP), to ensure the completeness of the reconstructed surface mesh. The planner can be easily adapted to any type of robot, and to within homogeneous or heterogeneous teams.
- Implementation of a centralized and a distributed multi-robot architecture for 3D reconstruction.

- The proposed methods have been validated through a wide variety of numerical experiments with UAVs and real-world experiments with ground robots.

Chapter 1 reviews some basic notions of mobile robotics, surface representation and mapping, path planning, and informative methods applied to online volume exploration or surface-inspection for reconstruction of unknown environments.

Chapter 2 presents some preliminary results of this thesis where the reconstruction task is allocated to a single robot. A volumetric mapping of the environment is used to represent the space and implicitly its surface. The completeness of the model is defined according to this surface representation. Thus, incomplete surface areas are identified, viewpoint configurations are generated to complete them and are clustered according to their locations in the 3D space. The visit of the viewpoints is planned by solving a variant of TSP, and free-space navigation and collision checking is realized via a sampling-based planner based on the volumetric map. Numerical experiments are performed in two realistic large-scale environments with fleets of UAVs using the Robot Operating System (ROS) and Gazebo simulator. This work is compiled in a conference paper [Hardouin *et al.* 2020b]. The algorithm shows good reconstruction performance even if the completion time remains long for the current autonomy of UAVs.

Chapter 3 presents an extension of the single-robot architecture for a multi-robot centralized systems, where the agents collaboratively reconstruct the whole 3D scene. The information sensed by the team is sent to the ground station which centralizes the computation and sends back the optimal trajectories to the robots. The allocation of the viewpoint clusters is performed using a greedy heuristic. Numerical experiments are performed in two realistic large-scale environments with fleets of UAVs. The centralized algorithm allows to drastically diminish the completion time. The content of the chapter has been presented at IROS conference [Hardouin *et al.* 2020a].

Chapter 4 presents a distributed version of the centralized multi-robot architecture described in Chapter 3, to increase the number of points of failure, where each robot computes its path according to the local and shared available information. A distributed algorithm is used by each robot to obtain an estimation of the map where the incomplete surface is extracted and perform a new scan. At the team level, each robot performs its own planning, but a low-level planner manages possible assignment redundancies. Numerical experiments are performed with a fleet of UAVs on a large-scale environment.

The three methods have been tested in numerical experiments. Chapter 5 presents real-world experiments performed with teams of wheeled robots in two indoor environments, a test arena and a parking lot. The work related in the thesis has been compiled in a journal paper.

Finally, the Conclusion ends the thesis with a summary of the main contributions, and

a list of possible subjects for future research.

Videos

Videos have been realized to illustrate the results of this thesis:

- Numerical simulations for the centralized multi-robot architecture (see Chapter 3): https://youtu.be/ce3XI2CSeh4
- Real-world experiments (see Chapter 5): https://youtu.be/yJ4lEyKrPGA

Publications

The contributions presented in this thesis have been published in peer-reviewed international conferences. Below is a list of the publications and submissions relative to these works at the time of writing.

Journal Paper

- G. Hardouin, J. Moras, F. Morbidi, J. Marzat, E. Mouaddib - *A multi-robot system for 3D surface reconstruction with centralized and distributed architectures* - Submitted to the IEEE Transaction on Robotics (T-RO).

International Conference Papers

- G. Hardouin, F. Morbidi, J. Moras, J. Marzat, E. Mouaddib - *Surface-driven Next-Best-View planning for exploration of large-scale 3D environments*. In Proc. 21st IFAC World Congress, pages 15501–15507, 2020.
- G. Hardouin, J. Moras, F. Morbidi, J. Marzat, E. Mouaddib - *Next-Best-View planning for surface reconstruction of large-scale 3D environments with multiple UAVs*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 1567–1574, 2020.

National Conference Paper

- G. Hardouin, F. Morbidi, J. Moras, J. Marzat, E. Mouaddib - *Surface-driven Next-Best-View planning for exploration of large-scale 3D environments* - Oral presentation at Reconnaissance des Formes, Images, Apprentissage et Perception (RFIAP), 2020[3].

[3] https://cap-rfiap2020.sciencesconf.org/resource/page/id/9

Chapter 1

State of the art

Chapter outline

1.1 Introduction 8

1.2 Autonomous robots: basic notions 8

1.2.1 Guidance, navigation and control 9

1.2.2 Pose estimation via visual-inertial odometry 9

1.2.3 Cooperation among multiple robots 11

1.2.4 Centralized, decentralized and distributed architectures 11

1.3 3D mapping 12

1.3.1 Surface shape representation 13

1.3.2 Volumetric representations 15

1.3.3 TSDF mapping 17

1.4 Path planning methods 20

1.4.1 Graph-search algorithms 20

1.4.2 Potential fields method 22

1.4.3 Sampling-based planners 23

1.5 Informative planning for 3D reconstruction 27

1.5.1 Next-Best-View planning 27

1.5.2 Surface inspection 29

1.5.3 Volume exploration 30

1.5.4 Mixed approaches 32

1.5.5 Multi-robot 3D modelling 33

1.6 Conclusion 33

1.1 Introduction

This chapter presents a review of the literature on 3D reconstruction with autonomous robots. Section 1.2 recalls some basic notions of mobile robotics, and positions the thesis within the existing Guidance, Navigation and Control (GNC) and multi-robot systems literature. Section 1.3 presents the state of the art on 3D cartography, for general applications but especially for mobile robotics, where reliable mapping for navigation, and an accurate representation of the surface of a 3D scene, are needed. The most popular 3D surface representations are presented, and the volumetric mapping literature is reviewed as well. A detailed description of the Truncated Signed Distance Function (TSDF) method and of the MarchingCubes algorithm concludes the section. Section 1.4 presents well-known motion planning methods in robotics, from the classical graph-based planners, to the widely-popular sampling-based planners such as Rapidly-exploring Random Tree (RRT) or Probabilistic RoadMap (PRM). Section 1.5 widens the path planning scope to informative planning and its application to 3D reconstruction with NBV methods. Our review ranges from object reconstruction techniques to the latest volume-exploration methods, and the general informative approaches applied to multi-robot systems are discussed in detail.

1.2 Autonomous robots: basic notions

An autonomous robot is a robot that performs behaviors or tasks with a high degree of autonomy (i.e without or with very limited external influence). A fully autonomous robot can:

- Gain information about the environment.
- Perform one or multiple tasks for an extended period without human intervention.
- Move either all or part of itself throughout its operating environment without human assistance, by knowing its location, where and how it needs to go.
- Avoid situations that are harmful to people, property, or itself unless those are part of its design specifications.

An autonomous robot may also learn or gain new knowledge to adapt itself to new tasks, algorithms or changing environments [Bekey 2005]. These requirements are generally fulfilled by realizing guidance, navigation and control system study. In a multi-robot context, the autonomous robots can interact with each other to various degrees, to perform a mission faster and more accurately, and to improve localization and coverage by an optimal task allocation.

1.2.1 Guidance, navigation and control

The design of an autonomous robot amount to design a robotic Guidance, Navigation and Control (GNC) system. The GNC capability is a critical enabler of every aeronautical or robotic system. It consists in designing innovative, robust, responsive complete or partially autonomous systems through development, and test capabilities. It goes from the initial concept through detailed mission analysis and design, hardware development and test, verification and validation, and real-experiment operations. The general GNC capabilities provided are:

- *Guidance*: algorithm and software development for path and/or trajectory computation according to the mission and safety requirements for all the mission phases, modelling and simulation.
- *Navigation*: system architecture design, sensor selection and modelling, perception, pose estimation and software (filter) development.
- *Control*: algorithms and software controller design, vehicle and control system requirements and specifications, stability analysis, modelling and simulation.
- *Hardware study*: selection, closed-loop testing and qualification; design, build, and calibrate specialized sensors.

In the context of this thesis, all these topics have been covered to some extent, and some basic elements have been given in Appendix A. However, the main research focus has been on the interaction between perception such as 3D mapping and reconstruction (*Navigation*) and path planning (*Guidance*), but also on hardware selection and unit- and functional-testing for the real-world experiments. The goal of this chapter is to present the state-the-art algorithms related to these topics. The next section details a *Navigation* related topics: the visual-inertial odometry algorithm used for estimating the robot pose.

1.2.2 Pose estimation via visual-inertial odometry

The Visual Odometry (VO) algorithms use images captured by a camera (see Appendix A.2) to estimate its motion [Scaramuzza & Fraundorfer 2011, Fraundorfer & Scaramuzza 2012]. The VO algorithms can be classified according to the type of visual information used. Thus, the methods which use all pixels are said to be *dense* and those which uses only few pixels are called *sparse*. There exist hybrid (or mixed) methods as well. Among them, we can also distinguish between the *direct* methods, which use the photometric information by operating directly on pixels, such as DTAM [Newcombe *et al.* 2011b] and LSD-SLAM [Engel *et al.* 2014],

and the *indirect* methods which estimate the motion from primitives extracted from the images, which condense the geometric information. Numerous keypoint detectors have been developed in the last two decade, such as, HARRIS [Harris & Stephens 1988], SIFT [Lowe 2004], FAST [Rosten & Drummond 2006], SURF [Bay *et al.* 2006], STAR [Agrawal *et al.* 2008]. Once detected, the primitives are then tracked by matching the correspondences between consecutive images, directly with the Kanade-Lucas-Tomasi (KLT) [Tomasi & Kanade 1991] algorithm which tracks Harris corners, or indirectly, using numerical descriptors such as SIFT, SURF or binary descriptors such as BRIEF [Calonder *et al.* 2010] and ORB [Rublee *et al.* 2011]. The last step consists in using of the detected matches to estimate the relative pose between two successive images while minimizing the associated geometric error. This Perspective-N-Points (PnP) [Hartley & Zisserman 2004] problem is generally solved using RANSAC algorithm [Fischler & Bolles 1981].

The visual-odometry process can be coupled with the inertial measurements to gather complementary information. In the literature, we can identify two types of interactions. The *loose coupling* approaches first try to estimate the motion separately, with the visual measurements one side, and the inertial measurements on the other, and they eventually fuse the two estimates. [Lynen *et al.* 2013] have proposed a generic method to fuse loose measurements from various types of sensors, where the inertial measurements are used in the prediction step of the Kalman filter while the visual information is exploited in the correction step. The second type of methods, the *tight-coupling* methods, are generally more precise - at the price of sensitive tuning - since they provide false-positive filtering during the visual matching phase, see for example e.g. OKVIS [Leutenegger *et al.* 2015].

In this thesis, a loose-coupling visual-inertial odometry algorithm called eVO [Sanfourche *et al.* 2013], has been used. This algorithm allows to estimate the trajectory of a mobile robot by using stereo image pairs. First, a 3D point-cloud is triangulated from a stereo image pair and is vertically aligned with the inertial measurement available. Harris corners [Shi *et al.* 1994] are extracted on the left image and are tracked with the Lucas-Kanade method [Baker & Matthews 2004] in the subsequent stereo image pairs. A set of 2D-3D matches is obtained, which allows to compute the relative pose from the last keyframe, solving a P3P problem [Fischler & Bolles 1981, Quan & Lan 1999]. Once the number of tracked points decreases below a certain threshold, the current stereo pair is selected to triangulate a next point-cloud reference. eVO provides a reliable state estimation that is used to locate a mobile robot and to plan its trajectories.

1.2.3 Cooperation among multiple robots

A team of autonomous vehicles can be designed to replicate one of the numerous coordination behaviors occurring in the natural world, such as in swarms of bees, mammal herds, schools of fishes or flocks of birds. The notion of multi-agent system has played an important role in robotic research, in the last decade, with applications to wheeled robots, unmanned air vehicles (UAVs), autonomous cars and satellite constellations. These systems can be used for maintenance in areas of difficult access, surveillance, exploration of unknown areas, among others. For these applications, the measurements collected by different robots are combined to improve the cooperative perception, localization, planning and control. Cooperation is justified, when the performance of a multi-robot system exceeds the sum of the performances of the single robots. A multi-robot system is also more robust to failures, when compared to a single-robot system. In fact, the probability of global mission failure decreases as the number of robots increases [Avizienis *et al.* 2004]. However, an efficient cooperation among multiple vehicles, also entails a number of new issues, such as collision avoidance and limited exchanges due to communication bandwidth constraints.

Regardless of the application, the robots should comply with a set of constraints. Each robot must be autonomous, meaning that it must be able to compute and control its own trajectory to achieve its goal. To this end, it must have access to a computing unit and local information coming from its own embedded sensors (proprioceptive and exteroceptive, see Sect. A.2) or by communicating with its nearest neighbors in the team. Finally, the types of pose estimation, trajectory planning, control and communication laws for the multi-robot systems play an important role. Depending on whether they are computed locally or via centralized or distributed fashion, these techniques and specific mission's requirements for each robot will widely differ.

1.2.4 Centralized, decentralized and distributed architectures

Various architectures can be considered for multi-robot systems, such as centralized, decentralized or distributed. Each one of them has its pros and cons, which will be briefly discussed below.

In a centralized architecture, all information required to perform a mission is broadcast and used by a central unit. This central unit can be a robot with special capabilities or a ground station. It collects the global information from the multi-robot system, performs back-end task computations, and broadcasts the results to each robot. It has a global knowledge of the behavior of the team which can be useful in many applications. However, the communication from/to the central unit is crucial, and it is the main drawback of

this architecture. In fact, this multi-robot system is highly dependent on the central unit, and the success of a mission could be compromised in the presence of communication losses or sensor failures. Dealing with a large number of robots could increase the computation time on the central unit and have a negative impact on the real-time constraints on which the system depends, leading to a catastrophic failure, in the worst case. Decentralized and distributed architectures have been developed to overcome these drawbacks.

In a decentralized system, each robot behaves independently and does not communicate with the other robots. In terms of robustness, this architecture is more reliable than the centralized one since a single and critical point of failure no longer exists. However, robot independence limits the achievable performances and precludes cooperative missions. It also becomes difficult to fulfill local objectives in the neighborhood of a robot, such as maintaining relative inter-distances or modifying a trajectory to compensate for unplanned behaviors of the teammates.

A distributed architecture follows the same principles of a decentralized one but it allows for (local) communication and cooperation among the robots, which use local information to perform their mission and exchange information among them. Recently, the advantages of distributed architectures have attracted increasing attention, and numerous applications have been developed including mapping, planning or control.

Information exchanges between robots are important and the nature of information transmitted might depend on the identity or role of the sender. Indeed, in some cases one robot can be a decision-maker for the whole multi-robot system. A team of robots can have a complex hierarchy and it might take advantage of a leader to centralize some network information, and thus increase system performance. The leader can be used to allocate different sub-missions to the vehicles, as well. Similarly, sub-groups can be specified to centralize the information in a neighborhood, and select which critical data should be transmitted to avoid network congestion and packet loss.

1.3 3D mapping

The process of 3D mapping aims at profiling of objects in three dimensions. There exist several 3D mapping methods in the literature, depending on the type of sensor used (stereo cameras, RGB-D camera, LiDAR). 3D mapping provides rich information about the observed environment that can be used for visualization purposes or in mobile-robotics applications. In civil engineering and architecture, a 3D map allows to visualize floor plans, walls and identify possible obstructions which may occur during the construction process. In digital heritage, accurate 3D maps can be used for teaching purposes, for elucidating ancient construction techniques, or as a backup in the case of natural disasters. The on-

line mapping techniques usually provides meshes in soft real-time, what is well-suited for robotic navigation. In the following, we present some of these methods.

1.3.1 Surface shape representation

The most popular mapping methods represent the shape of an object or an environment by means its surface. One common representation is the *point cloud*, which is a collection of data points defined by a given coordinates system. In a 3D Cartesian coordinates system, a point cloud may define the shape of some real or virtual object. Point clouds are generally produced by 3D scanners, such as LiDARs, which measure multiple points on the surface of an opaque object. Cameras are also amenable to produce point clouds via photogrammetry [Albertz 2002], which is often used in geology or cultural heritage preservation. In photogrammetry, we typically try to minimize the sum of the errors squared over the coordinates and relative displacements of corresponding points contained in successive images. This minimization is known as bundle adjustment [Hartley & Zisserman 2004] and is often performed with the Levenberg-Marquardt algorithm [Levenberg 1944]. An example of point cloud is depicted in Fig. 1.1-[left]. As the output of a 3D scanner, point clouds are used in many areas: to create 3D Computer Aided Design (CAD) models for manufactured parts, digital twins of assets for metrology and quality inspection and in a multitude of visualization, animation, rendering, and mass customization applications.

Point clouds are often aligned with 3D models or other point clouds, a process known as point-set registration. For metrology or surface inspection using computed tomography, the point cloud of a manufactured part can be aligned with an existing model and compared to check for differences. Geometric dimensions and tolerances can also be extracted directly from the point cloud, using, for example, the CloudCompare software [Girardeau-Montaut 2016].

Another popular approach is to represent the scene with *surfels*, (i.e., oriented discs) an abbreviation for surface element or surface voxel in the volume rendering and discrete topology literature [Pfister *et al.* 2000], see Fig. 1.1-[middle]. Surfels are well suited to modelling dynamic geometry, because there is no need to compute topology information such as adjacency lists. The mapping process consists of two main steps: sampling and surfel rendering. The sampling process converts geometric objects and their textures to surfels, using ray casting to create three orthogonal Layered Depth Images (LDIs) [Shade *et al.* 1998]. The LDIs store multiple surfels along each ray, one for each ray-surface intersection point. Each surfel stores both shape, such as surface position and orientation, and shade, such as multiple levels of prefiltered texture colors. The rendering pipeline hierarchically projects blocks to screen space using perspective projection. Com-

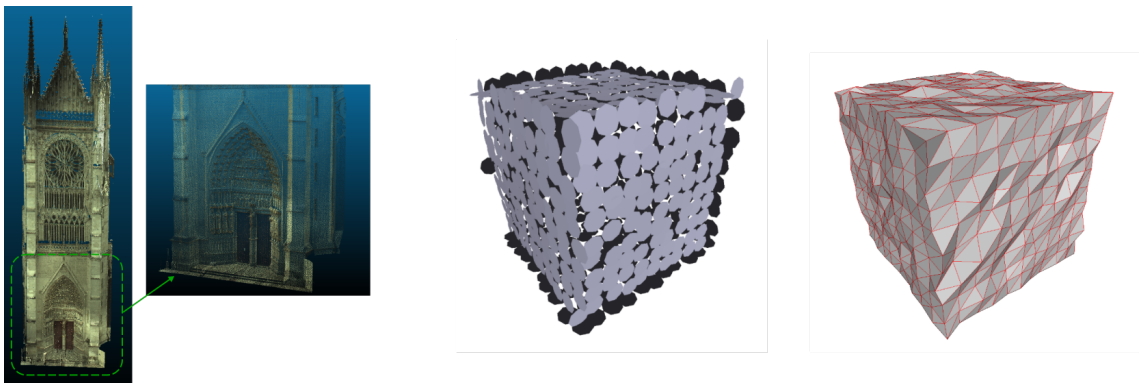


Figure 1.1: *Surface shape representation*: [left] Point cloud of the South portal of Amiens Cathedral; [middle] Examples of a surfel model and [right] a triangular surface mesh of a cube from [Andersen *et al.* 2010].

mon applications are medical scanner data representation, real time rendering of particle systems, and more generally, rendering surfaces of volumetric data by first extracting the isosurface.

These representations of the scene are flexible, since point or surfel coordinates can be updated very efficiently for the whole reconstruction. They are also highly adaptive as measurements at a higher resolution lead to denser point or surfel clouds, and they easily handle thin objects. However their main drawback is their discrete nature, which can be resolved by meshing. The point clouds can be often converted to polygon mesh or triangle *surface mesh* models, Non-Uniform Rational Basis Splines (NURBS) surface models [Piegl & Tiller 1996, Xie *et al.* 2012], or CAD models through a process commonly referred to as surface reconstruction. The classical meshing process consists in approximating the surface of an object by connecting the triangular facets by their edges. The mesh can be modeled as a connected graph, with a collection of sub-graphs constituted of exactly three connected vertices. An example is given in Fig. 1.1-[right]. There are many techniques for converting a point cloud to a 3D surface. Some approaches, like Delaunay triangulation [Delaunay 1934], Poisson surface reconstruction [Kazhdan *et al.* 2006], alpha shapes [Edelsbrunner *et al.* 1983, Edelsbrunner 1995], and ball pivoting [Bernardini *et al.* 1999], build a network of triangles over the existing vertices of the point cloud. These methods are usually too computationally expensive to run in real-time on the dense point clouds generated by RGB-D sensors. Other methods such as [Schöps *et al.* 2019] reconstruct online mesh based on surfel mapping. It asynchronously triangulate the smoothed surfels to reconstruct a dense surface mesh of the scene during a Simultaneous Localization And Mapping (SLAM) process.

While meshes allow to precisely represent the surface shape of an object, the informa-

tion about the occupied and free space is not available, thus limiting their online usage in mobile robotics applications. Surface meshes can be also extracted from volumetric representations which can be better suited for mobile robotic application, using for instance [Lorensen & Cline 1987], see next Sect. 1.3.3.

1.3.2 Volumetric representations

Volumetric mapping consists in discretizing the 3D space into small cells: the cells can be unknown, empty or occupied, and they can be used to represent a 3D scene of interest. This approach has been initially used for mapping the environment in robot navigation, and in the last decades several paradigms have appeared in the literature, thanks to the simultaneous rapid progress of computational resources and availability of inexpensive and accurate 3D sensors. Inspired by the classical 2D Occupancy Grid (OG) maps [Elfes 1989], the first significant 3D modelling approach, called *OctoMap*, was proposed in [Hornung *et al.* 2013]. It relies on a 3D occupancy grid with an internal octree model. This representation adapts the level of detail of the map to the environment, which reduces memory usage, and it has been implemented as an open-source library, which is particularly optimized to run in real time on Central Processing Unit (CPU). Since the OG representation is not very suited for path planning, an Euclidean Distance Transform (EDT) module has been added by the authors in [Lau *et al.* 2013]. It allows to compute the distance to the closest obstacles with a refresh rate of 1 to 2 Hz on a standard embedded CPU.

Newcombe *et al.* [Newcombe *et al.* 2011a] proposed another 3D reconstruction method, called *Kinect Fusion*. This approach was not directly intended for robotic navigation. It uses measurements from a RGB-D sensor to calculate the TSDF (Truncated Signed Distance Function, see Sect. 1.3.3, for more details) [Curless & Levoy 1996] over a grid. The main limitation of this method comes from the need to run on a Graphics Processing Unit (GPU): the mapped volume is thus limited to a few meters by the available memory resources. In [Whelan *et al.* 2012], the method was improved to map larger areas thanks to a moving TSDF volume. Areas that fall outside of the volume are exported into a mesh representation and stored in the system memory. Since the TSDF map is mostly empty (within usual truncation distances), the authors proposed to allocate voxels on the fly and to access them using a spatially-hashed index [Nießner *et al.* 2013]. This method allows to drastically reduce the memory size occupied by the TSDF and the access time, with limited effects on performances. This idea was taken up for the development of the open-source *Chisel* library[1], which provides a TSDF approach for 3D reconstruction on-

[1] https://github.com/personalrobotics/OpenChisel

board of mobile devices [Klingensmith *et al.* 2015]. [Zeng *et al.* 2017] proposed a GPU-based implementation for TSDF volume computation. In [Oleynikova *et al.* 2017], the authors proposed *Voxblox*, a 3D mapping method for robot navigation. This method was directly derived from *Chisel*, with the addition of an Euclidian Signed Distance Field (ESDF) estimation to produce a similar output as *OctoMap/EDT*. An octree-based mapping stack that is able to build either a TSDF or a probabilistic occupancy map has also been presented in [Vespa *et al.* 2018]. Recently in [Funk *et al.* 2021], the authors proposed a volumetric adaptive-resolution dense mapping framework that supports multi-resolution queries and data integration using occupancy mapping. The method maintains a high resolution 3D octree representation of observed occupied and free space in real-time. The *C-Blox* method proposed in [Millane *et al.* 2018] adapted the concept of manifold mapping from [Howard 2004] to the TSDF framework, yielding a mapping approach robust to localization drift. The idea consists in replacing a monolithic map linked to a single fixed frame, with a map subdivided into multiple *patches* (i.e. local sub-maps). Recently [Grinvald *et al.* 2021] introduced *TSDF++*, a novel multi-object TSDF formulation that can encode multiple object surfaces at any given location in the map. The representation allows to maintain accurate reconstruction of surfaces, in a multiple dynamic object tracking and reconstruction scenario, even while they become temporarily occluded by other objects moving in their proximity.

Some mapping methods aim to identify the data contained in the volume using classification methods [Kundu *et al.* 2014, Sengupta & Sturgess 2015, Vineet *et al.* 2015, Pham *et al.* 2019]. For instance in [Jadidi *et al.* 2017] Gaussian Processes (GPs) multi-class classification are used to identify the object contained in the scene and uses the continuous property of GP to extend it to multi-resolution occupancy mapping methods. Their extension of [Hornung *et al.* 2013], *Semantic OctoMap*, integrates into voxels, a semantic labeling and color information for visualization. The semantic data can be reused to guide the exploration of a volume using informative planning methods, see Sect. 1.5.

Compared to the single robot case, the multi-robot case has been rarely studied, especially for 3D mapping. Previous work has mainly focused on matching and merging mono-robot maps. Such methods have been proposed in [Birk & Carpin 2006, Liu *et al.* 2013] using OG maps. In [Forster *et al.* 2013], a collaborative mapping approach has been presented for multiple Micro Aerial Vehicles (MAVs) performing a matching and merging tasks, based on a point cloud representation. An important step forward was taken in [Howard 2004] where the concept of Manifold Mapping was introduced in the 2D space. This approach aims at enhancing map consistency using additional dimensions (e.g. *time*). The proposed implementation subdivides the map into a set of *patches* (corresponding to the new dimension(s)). Using this method, the au-

thors were able to successfully build an OG to map a large environment (up to 600 m^2) using four robots equipped with laser sensors. A 2D multi-robot SLAM algorithm based on laser measurements and signed distance functions, has been also introduced in [Koch *et al.* 2016]. Recently, [Duhautbout *et al.* 2019] proposed a distributed mapping approach based on a data structure similar to the one in *C-Blox* [Millane *et al.* 2018] with a 3D extension of the manifold concept from [Howard 2004]. The main objective is to allow several robots to share *patches*, so that they can plan their own trajectories using the information from the others (and correct the alignment of the patches, if necessary).

1.3.3 TSDF mapping

TSDF volume is a volumetric representation of the environment where the space is subdivided into a regular 3D grid of voxels. Each voxel $\mathbf{v} \in \mathbb{R}^3$ holds a sampled value $\phi(\mathbf{v})$ of the Truncated Signed Distance Function, $\phi(\mathbf{v}) : \mathbb{R}^3 \rightarrow \mathbb{R}$, which presents the distance to the nearest surface, clamped between a maximum and a minimum value. This distance is generally provided by depth maps. By convention, the TSDF is positive in empty space and negative in occupied space. The surface is then implicitly represented by the sign change of TSDF value for consecutive voxels. Moreover, each voxel contains a weight w, initialized to 0, which corresponds to the number of times a voxel has been observed, up to a maximum amount. The TSDF value ϕ and the weight w allow to distinguish between *empty*, *occupied* and *unknown* voxels :

$$\begin{cases} w = 0 & \rightarrow \textit{unknown voxels} \\ w > 0 \begin{cases} \phi \leq 0 & \rightarrow \textit{occupied voxel} \\ \phi > 0 & \rightarrow \textit{empty voxel.} \end{cases} \end{cases} \tag{1.1}$$

However, rarely observed voxels have small weights and unknown voxels have zero weights. Hence, voxels are considered unknown when the space is not explored or deep inside the surface of an object. This volumetric representation allows to clearly identify areas already visited by a robot.

The map is incrementally built as new depth values are collected. At each new depth map k, the integration, for a given voxel $\mathbf{v}$, consists in a recursive weighted mean of the the distance:

$$\begin{cases} \phi_k(\mathbf{v}) & \leftarrow \dfrac{\phi_{k-1}(\mathbf{v}) \cdot w_{k-1}(\mathbf{v}) + \phi_c(\mathbf{v}) \cdot w_c(\mathbf{v})}{w_{k-1}(\mathbf{v}) + w_c(\mathbf{v})} \\ w_k(\mathbf{v}) & \leftarrow w_{k-1}(\mathbf{v}) + w_c(\mathbf{v}) \end{cases} \tag{1.2}$$

where $\phi_c(\mathbf{v})$ and $w_c(\mathbf{v})$ correspond respectively to the computed TSDF distance of the voxel $\mathbf{v}$ to the nearest surface and its associated weight. Following

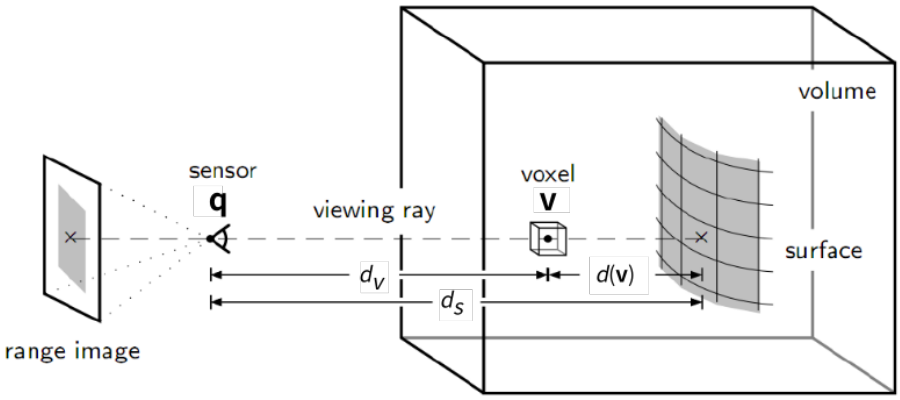


Figure 1.2: *Distance computation in the TSDF.*

[Newcombe *et al.* 2011a], ϕ_c is determined as follows:

$$\phi_c(\mathbf{v}) = \begin{cases} \min\left(1, \dfrac{d(\mathbf{v})}{\mu}\right) \operatorname{sign}\left(d(\mathbf{v})\right) & \text{if } d(\mathbf{v}) \geq -\mu \\ \textit{null} & \text{otherwise} \end{cases} \tag{1.3}$$

where μ represents a truncation threshold on the admissible depth value subject to measurement uncertainty. The depth value is integrated in the TSDF map, following (1.3), if it lies within the interval $\left[-\mu, \mu\right]$. A simple way to compute $d(\mathbf{v})$ consists in subtracting the distance of the sensor from $\mathbf{v}$ along the viewing ray, d_v, to the depth value d_s, as shown in Fig 1.2. A more precise method is proposed in [Newcombe *et al.* 2011a] to avoid uncertainties induced by depth map discontinuities. Another solution which relies on the Kalman filter for measurement integration, to protect the reconstructed model from poor-quality data, has been presented in [Trifonov 2013] and guarantees enhanced reconstruction quality.

The MarchingCubes algorithm [Lorensen & Cline 1987] can reconstruct a 3D mesh from a TSDF volume. Here, a discretized 3D volume consisting of small cubes, is directly based on the TSDF map. More specifically, each cube's vertex corresponds to a voxel of the TSDF map with its signed distance value. The algorithm uses a divide-and-conquer approach to locate the surface in a cube as a collection of triangular facets. It determines how the surface intersects this cube, then moves (or "marches") to the next cube. To find the surface intersection in a cube, the state value 1 is assigned to the vertex of a cube (corner) if the signed distance value at that vertex is less than or equal to 0 meaning that the vertex is on or below the surface (inside the object). On the other hand, cube vertices with distance values larger than 0 set their state value to 0 and are outside the surface. The surface intersects the edge of a cube when one of its vertices is outside and the others are inside the object. By construction, the topology of the surface is determined within

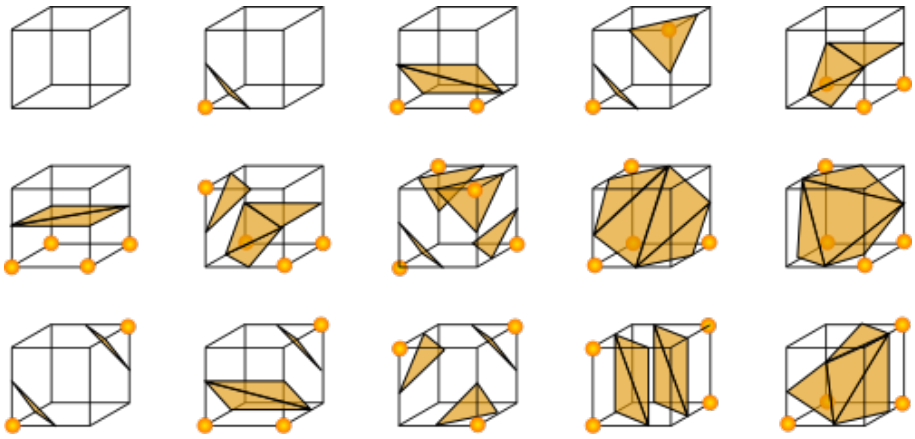
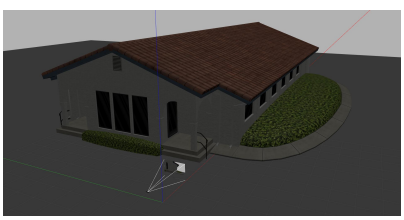


Figure 1.3: *Triangulated cubes*: the cube with no intersection (first top left) and the 14 surface patterns.

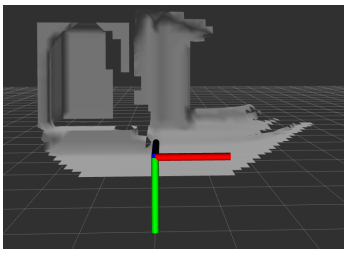
the cubes. Since there are eight vertices in each cube and two possible states (inside and outside), there are only $2^8 = 256$ ways a surface can intersect the cube. By enumerating these 256 cases, we can create a lookup table for surface-edge intersection, given the labeling of cubes' vertices. Two different symmetries of the cube can be used to reduce the number of possible cases: Fig. 1.3 shows these 14 admissible triangulation patterns. An 8-bit index is associated with each cube, based on the state of its vertices (1 bit per vertex), and serves as a pointer to an edge table which gives all edge intersections for a given cube configuration. Using the index to tell which edge the surface intersects, the surface intersection along the edge can be interpolated, according to the vertices' TSDF values. The final step of MarchingCubes consists in computing a unit normal for each triangle vertex.

TSDF mapping and the MarchingCubes algorithm can be coupled for 3D reconstruction in mobile robotics. A ROS-Gazebo simulation example is given in Fig. 1.4, where



((a)) Gazebo environment: the RGB-D sensor (bottom, center) and the building to scan

((b)) Depth map



((c)) 3D reconstructed mesh

Figure 1.4: *Reconstructed mesh from a depth map using a TSDF volume via OpenChisel and MarchingCubes.*

the reconstructed mesh is generated from a depth map.

1.4 Path planning methods

From an estimation of its pose and an a priori knowledge of the environment, an autonomous robot should be able to take decisions and execute complex tasks without any human intervention. A classical problem, for which numerous solutions exist in the literature (see [Goerzen *et al.* 2010]), is to find a collision-free path in a an environment populated with either static or dynamic obstacles. The most popular ones are reviewed here, such as graph search algorithms, potential field methods, and sampled-based planners.

In general terms, the path planning aims to find for the robot a feasible path in the free space of the environment, starting from an initial position and ending to a final one. The trajectory planning couples a schedule to the path computation, where the time or speed is considered.

Let $\mathbf{q} \in \mathrm{SE}(3)$ denote the configuration of the robot. Let $\mathcal{Q} \subset \mathrm{SE}(3)$ be the set of all configurations of the robot, $\mathcal{Q}_{free}$ the set of configurations in the free space, and $\mathcal{Q}_{occ}$ the set of occupied states, such that $\mathcal{Q} = \mathcal{Q}_{free} \cup \mathcal{Q}_{occ}$. The objective is then to find a path between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$, with $\mathbf{q}_{start}$, $\mathbf{q}_{goal} \in \mathcal{Q}_{free}$, which minimizes or maximizes an objective function (for example, find the shortest or minimum-time path, or the path with the maximum curvature radius).

1.4.1 Graph-search algorithms

These methods model the state space as a weighted graph, or a roadmap, and use a graph search algorithm to find a solution. Starting from an initial configuration (node) of the graph, the objective is to find a path to the final configuration, which minimizes a given cost function (distance travelled, time elapsed, etc.). This is done by generating a tree of paths emanating from the start node and extending the paths one edge at a time until all nodes have been covered, or the final configuration has been reached. Here, robot configurations are considered as nodes of the graph and the edges are part of the path linking one configuration to another.

Dijkstra's algorithm

Dijkstra's algorithm [Dijkstra 1959] iteratively searches for the shortest path in a weighted graph between the initial configuration $\mathbf{q}_{start}$ and the goal configuration $\mathbf{q}_{goal}$. The weights on the arcs are used to represent distances between two configurations (two nodes of

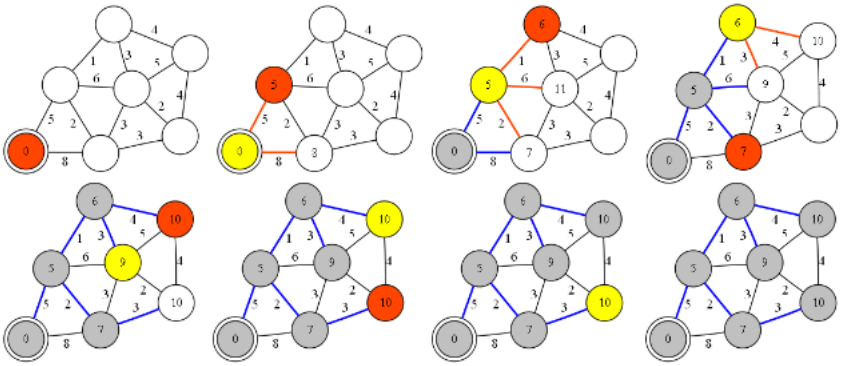


Figure 1.5: *Behaviour of Dijkstra's algorithm.* The search starts in the top-left corner and ends in the bottom-right corner. The yellow and red nodes represent the newly and previously-added configurations to the sub-graph, respectively. Blue edges represent tree edges. Red edges connect a newly added configuration to its adjacent configurations. https://math.cornell.edu/

the graph). Dijkstra's algorithm assumes that a graph has been already created and it builds a sub-graph and iteratively expands it. First, the sub-graph is initialized with the configuration $\mathbf{q}_{start}$. The sub-graph is then expanded by adding the adjacent configuration $\mathbf{q}_{adj}$ with the shortest distance from $\mathbf{q}_{start}$, forming a tree. Once the nearest configuration has been added, the distances from the new adjacent configurations and $\mathbf{q}_{start}$ are updated. If two paths exist between two configurations in the sub-graph, the shortest path is selected to pursue the tree-search. Hence, a newly added configuration is not necessarily adjacent to the previously added configuration (but it is adjacent to a node of the tree). The process is repeated until the generated tree becomes a *spanning tree* (every configuration has been visited) or until the configuration $\mathbf{q}_{goal}$ has been reached. The behavior of the algorithm is illustrated in Fig. 1.5.

Note that Dijkstra's algorithm makes no attempt of direct "exploration" towards the destination. Rather, the sole consideration in determining the next configuration is its distance from $\mathbf{q}_{start}$. It therefore expands outward from the starting point, interactively considering every configuration that is closer in terms of path distance until it reaches the destination, and thus finding the shortest path. However, the performance of the algorithm decreases as the graph size increases and its topology becomes more involved.

A* algorithm

The A* algorithm [Hart *et al.* 1968] is a variant of Dijkstra's algorithm which allows to find a solution in shorter time. The major improvement is an heuristic which computes for each adjacent configuration $\mathbf{q}_{adj}$, the cost required to extend the path all the way through

to the goal $\mathbf{q}_{goal}$. Specifically, A* selects the path that minimizes the function,

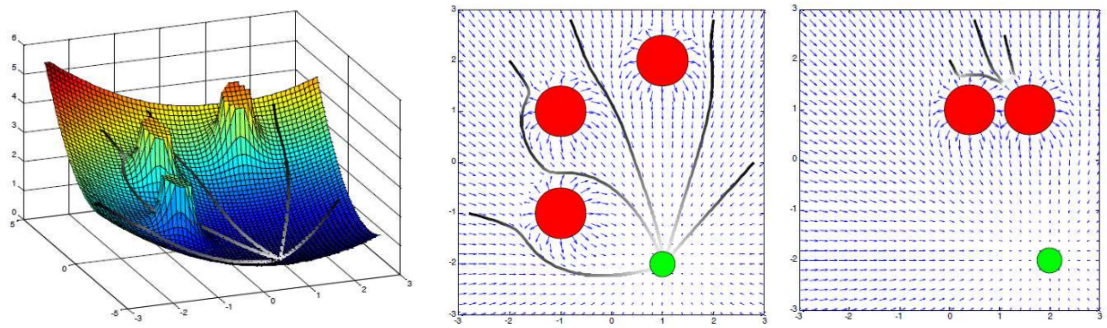
$$f(\mathbf{q}_{adj}) = g(\mathbf{q}_{adj}) + h(\mathbf{q}_{adj})$$

where $g(\mathbf{q}_{adj})$ is the cost of the path from the start configuration $\mathbf{q}_{start}$ to $\mathbf{q}_{adj}$, and $h(\mathbf{q}_{adj})$ is a heuristic function that estimates the cost of the cheapest path from $\mathbf{q}_{adj}$ to $\mathbf{q}_{goal}$. The heuristics allows to limit the number of configurations to add and reduce the computation time, but the optimality is not ensured.

1.4.2 Potential fields method

The potential field method was first introduced in [Khatib 1986]. It considers the robot as a particle subject to the influence of a potential field which guides it towards the goal configuration. Let $\mathbf{q}$ be the configuration be state of the robot, and $\mathcal{Q}$ the configuration space, such that $\mathbf{q} \in \mathcal{Q}$. In practice, the space is partitioned into a grid of cells with obstacles and free space. The algorithm assigns an artificial potential field to each cell which can attract U_{att} or repulse U_{rep} the robot. There are various expressions for these distance-based fields, depending on the type of application and environment considered [Hellström 2011]. The potential field functions is defined as:

$$U(\mathbf{q}) = U_{att}(\mathbf{q}) + \sum_{i=1}^{m} U^i_{rep}(\mathbf{q}) \tag{1.4}$$



((a)) 3D illustration of the total potential field $U(\mathbf{q})$.

((b)) Vector field and trajectories of the robot towards the goal.

((c)) No matter the initial condition, the robot is stuck in a local minimum.

Figure 1.6: *Examples of potential fields for path planning with obstacles, from* [Hellström 2011]: The black curves represent the paths of the robot for different initial conditions. The grey scale is proportional to the speeds of the robot. In (a), the color gradient shows the different magnitude of the fields. In (b) and (c), the red circles represent repulsive obstacles and the green circle the attractive goal.

where $U^i_{rep}(\mathbf{q})$ is the repulsive field associated to the ith obstacle, with $i \in \{1, 2, \ldots, m\}$. The force $\mathbf{F}(\mathbf{q})$ applied to the robot at the configuration $\mathbf{q}$, is computed as the negative gradient of the total potential field,

$$\mathbf{F}(\mathbf{q}) = -\nabla U(\mathbf{q}) \tag{1.5}$$

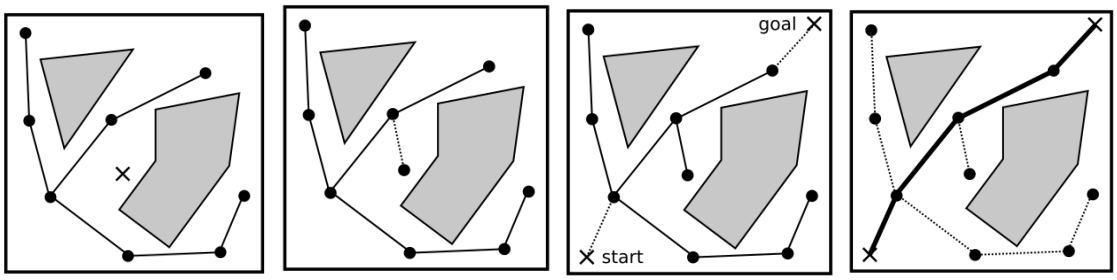
The vector field defined by (1.5), provides, for every configuration $\mathbf{q}$, the direction of steepest descent towards the goal. The potential field method is conceptually simple but it suffers from some drawbacks: a robot can exhibit an oscillatory behavior around an obstacle and it can be trapped in local minima (see Fig. 1.6-(c)). The oscillatory behavior can suppressed by considering distance-based potential fields and a limited range of influence around an obstacle. Moreover, several methods have been proposed to deal with the problem of local minima. For example, when the robot is stuck in a local minimum, to escape from it, a random movement can be triggered [Barraquand & Latombe 1990, Barraquand & Latombe 1991]. In [Akishita *et al.* 1990], potential fields that are solutions to the Laplace equation (harmonic functions), and which do not have local minima, are used. Harmonic functions generate smoother paths, which are more suited to aircraft-like vehicles, than the previous methods. Moreover, these functions can be applied not only to fixed obstacles but also to moving obstacles. An improved version of the potential fields, the navigation functions, can overcome the local minima problem as well [Filippidis *et al.* 2012, Filippidis & Kyriakopoulos 2012, Loizou 2017]. However, the harmonic and navigation functions are more demanding than the standard potential functions, in terms of computational resources.

1.4.3 Sampling-based planners

While classical graph search methods (cf. Sect. 1.4.1) rely on a predefined, static configuration space, sampling-based planners continuously expand it by sampling new configurations. For difficult problems, random sampling provides fast solutions. Moreover, the sampling-based methods are known to have excellent performances in high-dimensional configuration spaces, and they handle nonholonomic constraints. The two most famous sampling-based motion planning algorithms, are presented below.

Probabilistic Roadmap - PRM

The probabilistic roadmap planner [Kavraki *et al.* 1996] allows to determine a path between a starting and final configuration of the robot, while avoiding collisions with the obstacles. The idea is to generate random samples from the configuration space of the



((a)) The construction phase: a random sample, denoted by × is generated.

((b)) A local planner is used to connect the new sample to nearby roadmap vertices

((c)) The query phase: the start and goal configurations are added to the roadmap.

((d)) A graph search algorithm is used to connect the start and goal through the roadmap.

Figure 1.7: *Illustration of the PRM planner, from* [Short *et al.* 2016].

robot, to verify whether they belong to the free space, and to use a local planner to connect these configurations to other nearby configurations. The start and goal configurations are added to the roadmap, and a graph search algorithm is applied to the resulting graph to determine a path between them (see the illustration in Fig. 1.7). More precisely, the procedure consists of two phases:

- In the construction phase, a graph or roadmap is built, approximating the motion of the robot in the environment. First, a random configuration is generated. Then, it is connected to some neighbors, typically either the k-nearest neighbors or all neighbors within some predetermined distance (a disk of radius r). New configurations and arcs are added to the graph and we check if the edge connecting two configurations does not involve any collision. The process continues until the roadmap is dense enough;
- In the query phase, the start and goal configurations $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ are connected to the graph, and the final collision-free path is obtained by applying Dijkstra's or A* algorithm.

Multiple queries can be answered at the same time and the construction and the query phases do not need to be executed sequentially. If PRM experiences some difficulties during the query phase, the algorithm can switch back to the construction phase to adapt the size of the roadmap. This makes PRM approach suitable for trajectory planning with multiple robots, since they can simultaneously look for a solution. However, the PRM planner performs poorly in problems where the optimal path passes through narrow passages in the free space. Under some relatively weak conditions on the shape of the free space, PRM is provably probabilistically complete, i.e. as the number of sampled configurations increases, the probability that the algorithm will not find a path, if one exists, approaches zero.

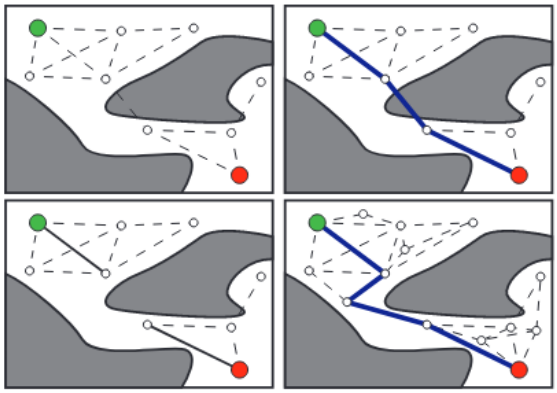


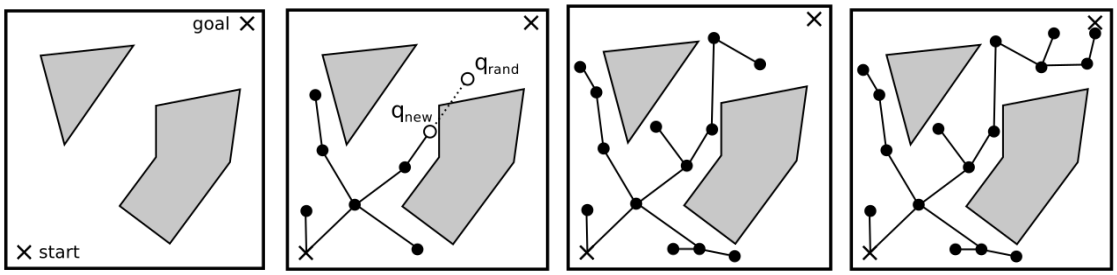
Figure 1.8: *Illustration of the LazyPRM⋆ planner, from* [Hauser 2015]. The edges in the roadmap are not checked for collision (dotted lines) until a candidate path to the goal is found. The feasible edges are marked (solid lines) and the infeasible ones are deleted. The process repeats until a feasible path to the goal is found.

An improvement on PRM, the PRM⋆ algorithm [Karaman & Frazzoli 2011] is a provably asymptotically optimal version of the classical PRM planner (i.e. such that the cost of the returned solution converges almost surely to the optimum). In the construction phase, the connections between the vertices of the roadmap are not established by considering a fixed radius, but r decreases with the number of samples.

An other improvement, LazyPRM⋆ [Hauser 2015] aims to reduce the resource-intensive collision-checking process, by adopting a "lazy strategy". Similarly to the standard PRM, a set of feasible vertices connected by edges is expanded. However, the edges are not immediately checked for collision, but only when a better path to the goal is found, cf. Fig. 1.8. In this way, the vast majority of edges that have no chance of belonging to the optimal path, are not checked. Smoother paths and paths through narrow passages can be found, thus overcoming the major drawbacks of the classical PRM formulation. LazyPRM⋆ converges toward the optimum substantially faster than the existing sampling-based planners in real problems, involving multiple rigid bodies and robot arms.

Rapidly-exploring Random Tree - RRT

While PRM is a two-queries path planning method, Rapidly-exploring Random Tree (RRT) [LaValle 1998] is a single-query path planning method, i.e. it builds a search graph and finds a solution at the same time. The algorithm grows a tree from a starting configuration $\mathbf{q}_{start}$ by iteratively sampling random configurations $\mathbf{q}_{rand}$ in the search space. More precisely, the idea is to bias the exploration toward unexplored areas by dividing the space into partitions of a Voronoi diagram, by sampling one configuration only in each partition and then extending incrementally the exploration tree toward them. This allows RRT to



((a)) A tree is grown from the start configuration towards the goal.

((b)) The planner generates a configuration $\mathbf{q}_{rand}$ the nearest configuration, $\mathbf{q}_{new}$, in the tree is determined.

((c)) The tree rapidly explores the free space.

((d)) The planner terminates when a configuration is close to the goal.

Figure 1.9: *Illustration of the RRT algorithm, from* [Short *et al.* 2016].

explore the entire space rapidly. As a new random configuration is generated, we try to connect it to the nearest configuration in the tree. If the connection is feasible (i.e. it lies entirely in the free space and satisfies other additional constraints), this results in the addition of the new configuration $\mathbf{q}_{new}$ is added to the tree (see Fig. 1.9). If the expansion of the graph is uniform in every direction of the free space, the convergence time could be long. RRT has no need of a local planner because the global planner is capable of finding a feasible path between two states on its own. Moreover, RRT has been specifically designed to handle nonholonomic constraints (including dynamics) and systems with multiple degrees of freedom.

Similarly to PRM*, [Karaman & Frazzoli 2011] proposed an improvement of RRT called RRT*. The graph is modified locally, and the shortest path between the start and goal configuration is found. The optimal path is found but RRT* is typically slower than the original RRT.

Among the methods reviewed in this section, the graph-search algorithms are usually very effective find local paths in the free space between two configurations. The potential field method is resource intensive and it needs a fixed environment to work properly, which limits its usage for incremental online 3D reconstruction. Finally, PRM and RRT are both used in state-of-the-art robotic applications because of their rapidity, and high flexibility to variations in the environment. Notably, as shown in the next section, they are frequently used for informative planning tasks.

1.5 Informative planning for 3D reconstruction

Planning informative paths for autonomous robots, has been the subject of active research in recent years, and it has been used in numerous applications, e.g. for collecting geological samples, weather forecasting, or for real-time 3D modelling of unknown environments. In all cases, the robot needs to travel along a path in order to maximize the collected information, it should often satisfy additional constraints such as energy, time or traveled distance budgets. More precisely, the informative path planning problem aims maximizing the knowledge of a certain environment M from in situ measurements collected by a mobile robot, equipped with a set of sensors. This can be formalized as follows:

$$\begin{aligned} \mathbf{p}^\star &= \arg\max_{\mathbf{p}\in\mathbf{\Psi}_M} u(\text{sample}(\mathbf{p})) \\ &\text{s.t. } \text{cost}(\mathbf{p}) < B \end{aligned} \tag{1.6}$$

where $\mathbf{p}^\star$ is the optimal path, $\mathbf{p}$ belongs to the space of all possible paths $\mathbf{\Psi}_M$ within the environment M, the function sample$(\cdot)$ provides a finite set of measurement locations along the path $\mathbf{p}$, and $u(\cdot)$ denotes the utility (informativeness) of these measurements. The cost of the path is given by the cost$(\cdot)$ function, which cannot exceed a predefined budget B. In contrast to metric path planning, informative path planning usually produces paths which are much longer than the shortest path from the start to the goal: in fact, the robot aims at exploring the entirety of a given environment. To counter this effect, the utility function can be formulated as a single-objective trade-off between utility and cost [González-Banos & Latombe 2002], where the cost serves as a penalizer in the maximization process.

Informative planning methods for 3D reconstruction, is often related to Next-Best-View planning in the literature, as discussed in the next section.

1.5.1 Next-Best-View planning

Typically used for object reconstruction [Connolly 1985], Next-Best-View (NBV) planning methods allow to determine an optimal sequence of viewpoints for a moving sensor. The environment can be partially, or completely unknown and at least one acquisition has been done. NBV planning is a two-step process: it first generates the sensor configurations, and then evaluates them according to the planning objective. The candidate evaluation is performed according to the following criteria [Chen *et al.* 2008]:

- *Positioning*. The view must be reachable by the robot without collisions and near to the previous one to limit motion.

- *Information gain*. The next view must sense unknown surfaces in order to completely scan the target. The information is quantifiable according to the type of representation.
- *Registration*. To merge the next scan with the previous ones, there must be some overlapping. The Iterative Closest Point (ICP) algorithm [Besl & McKay 1992], among others, can be used to register the scans or to update the camera's pose and decrease the positioning error.
- *Sensing*. The surface to be scanned must be within the camera's field of view and ; in addition, for a high-quality reconstruction, the angle between the sensor and the surface normal should be small enough [Chen *et al.* 2008]-p.77.

Obviously, these requirements should be compatible with the mission objective, hardware, mapping method and computational resources. Most of the recent NBV approaches are informative planning methods, searching for the must useful path for the camera-robot. Within the large NBV literature we can identify two types of approaches to generate new viewpoint candidates:

- *Occlusion-based* methods use the occlusion boundaries within the range image of the current view, or a partial map to generate the viewpoint candidates.
- *Sampling-based* methods sample random sensor configurations in the free space and evaluate them as candidate viewpoints.

The sampling-based methods came to prominence with the emergence of sampling-based planners and their usage for mapping large volumes. They are often related to volume-exploration methods, where a robot must rapidly build a map coverage of an unknown environment [Bircher *et al.* 2016]. On the other hand, occlusion-based methods are more related to object reconstruction with 3D surface representations, such as point clouds or 3D meshes. When the 3D representation of a surface is used for viewpoint generation, they are called *surface-based* [Kriegel *et al.* 2011, Border *et al.* 2018]. With the advent of volumetric mapping, these methods are less and less used in mobile robotics.

Volumetric mapping methods represent the free and occupied space, and provide accurate surface estimations for reconstruction, both for the single- and multi-robot cases. In parallel with the emergence of these mapping methods, volume exploration planners have been developed to rapidly explore unknown volumes and provide coarse 3D reconstructions. The main focus of early work on surface-based methods has been on accuracy, and the cost function optimized, incorporates one or multiple surface criteria. Recently,

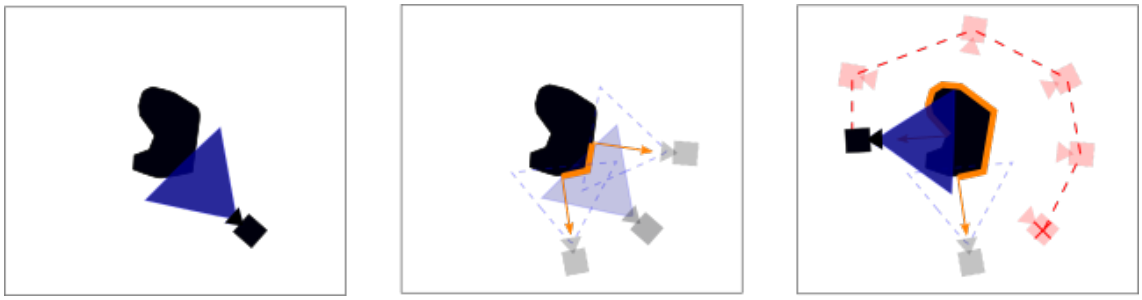


Figure 1.10: *2D illustration of frontier-based NBV planning for surface inspection.* From a first scan (left image), sensor configurations are extracted from the surface representation, and the planning in the free space is performed to pursue the 3D surface reconstruction (center and right image). The sensor configurations are depicted in red, the surface in orange, and the scanned space in blue.

the volumetric and surface-based approaches have been combined to take advantage of their unique properties, but almost only the single-robot case has been considered.

1.5.2 Surface inspection

Surface inspection methods use the current surface to generate candidate viewpoints which ensure a complete and accurate 3D reconstruction. NBV methods determine the next best viewpoints to visit, depending on the mission of the robot. Among them, *frontier-based methods* yield viewpoint configurations pointing towards the frontier of the known surface, represented as a mesh, according to some orientation, position or sensing constraint. By visiting these configurations, the robot gathers new surface information with some overlapping, to ensure continuity [Pito 1996, Vasquez-Gomez *et al.* 2014b, Kriegel *et al.* 2015, Border *et al.* 2018]. The general process is illustrated in Fig. 1.10.

[Border *et al.* 2018] use a density representation to define the a frontier in a point cloud. To ensure that the final point-cloud model has a uniform resolution, its density is computed by counting the number of neighbors within a fixed radius. Areas whose resolution is below a given threshold are considered as frontier regions. The surface around a frontier point is then approximated as locally planar via the eigendecomposition of a matrix representation of its neighborhood. The set of eigenvectors form an orthonormal basis and each eigenvector describes a component of the observed geometry such as, normal, frontier and boundary. The normal is then used to compute a new viewpoint. NBV's is based on a trade-off between the distance between two consecutive views, and the distance between the next and the initial view.

Other methods estimate the shape and orientation of the hidden surfaces-mesh, from the boundaries of a first mesh acquisition. [Pito 1996] creates "void" patches at the fron-

tier of the known surface, directly from the topology of the mesh. The configurations of the scanner are then chosen within its spherical configuration space to cover the maximum number of void patches while ensuring a sufficient overlap with the existing meshes. In [Kriegel *et al.* 2011], region-growing algorithms are used. These algorithms estimate the shape of a surface, as a quadratic patch, from the boundary information. The normal is then computed, far enough from the boundaries, and a new viewpoint is generated along the normal at given distance from the surface, to guarantee sufficient overlap with the existing mesh. Finally, the nearest NBV is chosen to complete the reconstruction. A scan is then performed and discarded if it does not provide a certain amount of new points.

Most of these approaches deal with small objects reconstructed by cameras mounted on the end-effector of robot manipulators, they rely on strong assumptions on the navigable free space and plan the visit towards only one new view. NBV inspection methods have been extended to mobile robots by using an occupancy grid to identify the frontier of a surface [Yoder & Scherer 2016]. The new information provided by a viewpoints is estimated by counting the number unknown cells contained in the sensor range [Kriegel *et al.* 2015]. Similarly, the overlap is quantified by counting occupied cells [Vasquez-Gomez *et al.* 2014a]. More recently, TSDF mapping has been used to generate candidate viewpoints from an implicit surface representation [Monica & Aleotti 2018].

1.5.3 Volume exploration

Given a volumetric mapping, a robot can try to explore an unknown volume containing objects of interest. The volume-exploration methods usually leverage a volumetric representation (cf. Sect. 1.3.2), to identify known, unknown and occupied areas. Similarly to the surface-based methods, the formulation allows to identify the *frontier* between the known and unknown volume and to detect areas of interest to explore [Yamauchi 1997]. The sampling-based planners (cf. Sect. 1.4.3), such as RRT, or RRT*, are used for trajectory generation, by incrementally expanding a tree constituted of randomly-sampled sensor configurations in the free space. On the other hand, PRM and LazyPRM* planners extract a path from a graph formed by randomly-sampled poses between a start and a goal configuration, according to an objective function. In an NBV-planning framework, the volumetric-exploration methods expand trees with randomly-sampled configurations and select the NBV trajectory that guarantees the maximum coverage of the volume, see Fig. 1.11. Some methods use only single-query path planners, such as RRT, to generate the viewpoints in the free space and select them according to the coverage of unknown voxels they may provide [Bircher *et al.* 2016, Papachristos *et al.* 2019, Selin *et al.* 2019, Batinovic *et al.* 2021, Respall *et al.* 2021]. Viewpoint coverage of the trajectory is evalu-

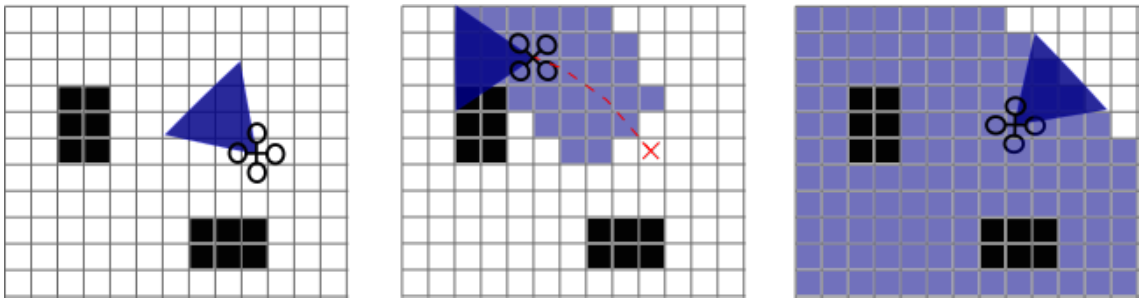


Figure 1.11: *2D illustration of NBV planning for volume exploration.* From a first scan (left image), the robot generates randomly sampled viewpoints to explore the unknown space (white cells) and distinguish between the free and occupied space (black cells). The scanned cells are represented in blue (center and right image).

ated via ray-casting [Bresenham 1965] over the volumetric map, and as the robot follows the assigned path, the volume is automatically explored. Ray-casting can be resource intensive in the presence of a large number of viewpoint configurations. To reduce expensive computation of querying over parts of an OctoMap volume, [Selin *et al.* 2019], it is possible to calculate the viewpoint queries over a continuous utility function (provided by Gaussian processes, [Williams & Rasmussen 2006]), varies smoothly across space. [Batinovic *et al.* 2021] propose frontier-based method which uses the octree properties of OctoMap to perform a multi-resolution frontier extraction which accelerates the volume-exploration planning. [Xu *et al.* 2021] have recently proposed an incremental PRM planner which "saturates" the volume of randomly-sampled viewpoint configurations to ensure volume coverage. An entropy criterion is used to assess whether the roadmap is dense enough. The path is found among the viewpoints, by estimating the unknown voxel of the ESDF map [Oleynikova *et al.* 2017] they cover. Each unknown voxel is classified as "normal", "surface" or "frontier", and a hierarchy is established to define the gain of visiting the viewpoint. Some other methods couple planning to SLAM as in [Papachristos *et al.* 2017], where the NBV planning ensures re-observation of reliable landmarks to refine the odometry estimation.

In [Wang *et al.* 2019], the authors use a topological map, named Semantic Road Map (SRM), to represent the environment during the exploration. It is graph where the nodes correspond to the exploration states and the edges represent the collision-free constraints. Notably, the SRM integrates both the semantic and structure information of the environment, The SRM is incrementally built during the exploration process, thereby avoiding the unnecessary reconsideration of the explored areas when constructing the topological map. The NBV is determined using the semantic information and the resulting path is typically shorter than the one determined by volume exploration methods.

The fast and efficient method reviewed in this section rely on a coarse volumetric map of the environment for navigation purposes. The viewpoint configurations are sampled randomly to maximize space coverage, but they are not generated to account for the completeness and accuracy of the reconstructed surface.

1.5.4 Mixed approaches

Recently, hybrid or mixed methods have been proposed to improve reconstruction accuracy and minimize completion time, by taking advantage of surface-inspection and volumetric-exploration approaches. These methods use volumetric mapping to perform safe navigation, and different strategies to ensure surface coverage and quality. In [Bircher *et al.* 2018], the reconstruction process includes two identical steps applied to different types of 3D model. First, a coarse volumetric map of the environment is built by means of a classical volume-exploration approach [Bircher *et al.* 2016], and its associated 3D mesh is generated. Then, in a second step, the process is repeated, but this time information mapping is performed on surface elements, and the coarse surface mesh is refined until a certain resolution. To evaluate the utility, ray-casting is performed for the surface element, by estimating if it is observable by the camera and pointing to the camera. Reasoning on the occupancy map, the algorithm in [Song & Jo 2017] allows a robot to cover the whole surface model: the viewpoints are sampled with a RRT* planner and pruned to keep only the configurations which cover the maximum number of surface cells. Surface-mesh coverage is assessed by computing the percentage of observed surface cells over the total number of surface cells of the original model. The authors have extended their approach in [Song & Jo 2018], where the path is further refined by taking the completeness of the environment surface into account. This is made possible by coupling volumetric and point cloud representations. More recently, the method has been improved and validated with extensive numerical and real-world experiments in [Song *et al.* 2021]. [Schmid *et al.* 2020] have proposed an RRT*-inspired online informative path planning algorithm which uses a TSDF surface criterion for planning. The accuracy of the obtained mesh is also measured.

However, the aforementioned mixed approaches rely on multiple volumetric and surface maps to achieve their task, which are costly to generate, and only a few of them solve the surface inspection problem directly (as in this thesis). Another exception is [Kompis *et al.* 2021]. Here, the authors propose a surface-based planning algorithm which generates viewpoint configurations directly from the ESDF frontier voxels, and path planning is performed with A* based on motion primitives, followed by a B-Spline trajectory optimization. The algorithm has been validated with large numerical experi-

ments on indoor and outdoor environments using a MAV.

1.5.5 Multi-robot 3D modelling

As already mentioned above, the existing research on incremental online reconstruction, generally features a single robot, and multi-robot cooperative scanning still remains an open problem in the literature [Amigoni & Gallo 2005]. By considering robots equipped with laser scanners in a 2D environment, the authors in [Burgard *et al.* 2005] were the first to propose a method to compute frontier cells, and to define the trade-off between utility and traveled distance for a multi-robot exploration task. Recently, in [Mannucci *et al.* 2017], a team of aerial robots should explore an uncluttered outdoor environment, and the authors propose one of the first cooperative frontier-based methods which relies on 3D space modelling for multi-robot exploration. The centralized map (OctoMap) and the (RRT*-based) coordinated motion planning of the UAVs are computed on a ground station, and the algorithm is evaluated by using realistic numerical experiments in ROS-Gazebo with emulated stereo sensors. Exploration methods based on probabilistic occupancy maps with entropy reduction, such as decMCTS [Best *et al.* 2016] or SGA [Atanasov *et al.* 2015], stem from the notion of mutual information of range sensors [Charrow *et al.* 2015]. In [Corah *et al.* 2019, Corah & Michael 2019], the authors have proposed a finite-horizon decentralized planner, called DSGA ("Distributed Sequential Greedy Assignment"), which relies on sampling-based Monte-Carlo Tree Search (MCTS) [Chaslot 2010]. Robots' paths are allocated by solving a submodular maximization problem over matroid constraints with greedy assignment heuristics. Finally, more recently, in [Corah & Michael 2021], the method has been adapted to exploration, by maximizing a volumetric objective function. This matroid-constrained submodular maximization approach has also been used for NBV planning with a camera-equipped robot arm in [Lauri *et al.* 2020].

1.6 Conclusion

From the previous literature review, we can notice that a large body of research has leveraged volumetric exploration for 3D reconstruction, but without taking the surface-completeness and the problem of occlusions, explicitly into account. Moreover, even if the surface inspection problem is being increasingly addressed with mobile robots, to the best of our knowledge, no multi-robot formulation has been proposed in the literature.

In this thesis, we introduce a generic NBV planning strategy inspired by the mixed approaches, to solve the surface inspection problem and cooperatively reconstruct large-

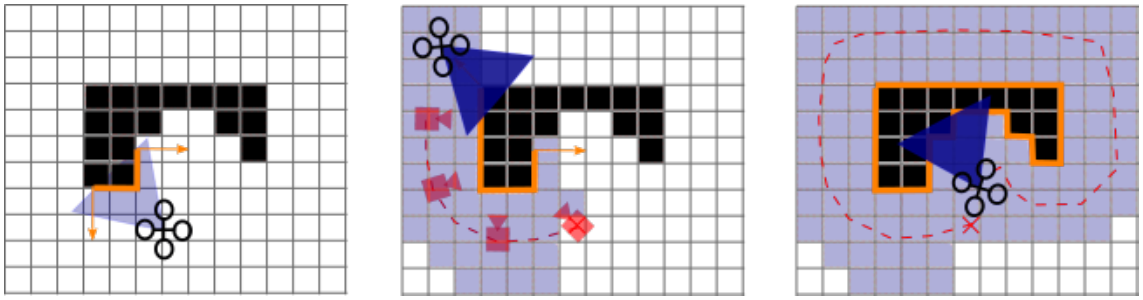


Figure 1.12: *2D illustration of NBV planning for surface inspection using volumetric mapping.* From a first scan (left image), sensor configurations are extracted from the volumetric representation of the surface, and the planning in the free space (white cells) is performed to pursue the 3D surface reconstruction (center and right image). Occupied voxels are depicted in black, sensor configurations in red, the surface in orange, and the scanned space in blue.

scale environments with a team of mobile robots. Our frontier-based method relies on a volumetric representation of the surface, which allows to identify areas of interest to scan. Candidate viewpoint configurations are generated from these areas depending on robots' and sensors' specifications, and are clustered according to their location in space. In order to find the best path for each robot, the interest of visiting a specific cluster is evaluated. The assignment problem is formalized using non-decreasing set function, as a maximization problem over disjoint set. In order to allocate the sensor configurations, a TSP-based greedy local search is used [Fisher *et al.* 1978, Bian *et al.* 2017]. Successive approximate resolutions allow to determine paths that cover the unknown environment, while ensuring short traveled distance with a reduced computational cost. Another attractive property is that the completeness of surface reconstruction is maximized, see Fig. 1.12. The proposed strategy, called TSP-Greedy Allocation (TSGA), is divided into three architectures, single-robot, centralized and distributed (referred to as dist-TSGA), which have been validated with synthetic data and via real-world experiments with multiple multirotor UAVs and wheeled robots.

CHAPTER 2

Surface reconstruction of unknown environments with a single robot

Chapter outline

2.1 Introduction . . . 35
2.2 Problem formulation . . . 36
2.3 System overview . . . 38
2.4 Perception . . . 39
2.4.1 Surface-based mapping . . . 39
2.4.2 ISE extractor and viewpoint generation . . . 41
2.4.3 Cluster of configurations . . . 41
2.5 Single-robot planning . . . 42
2.5.1 Graph formulation . . . 42
2.5.2 Inspection problem . . . 44
2.5.3 Low-level planner . . . 45
2.6 Numerical experiments . . . 47
2.6.1 Simulation setup . . . 47
2.6.2 Metrics . . . 50
2.6.3 Choice of penalty terms . . . 50
2.6.4 Results and discussion . . . 51
2.7 Conclusion . . . 53

2.1 Introduction

In this chapter, we study the problem of incremental exploration and surface reconstruction of an unknown 3D environment for inspection purposes, with a single-robot. To

address this problem, a 3D mapping method is required for collision avoidance and to evaluate the completeness of the reconstruction. Thanks to the volumetric representation, it is possible to identify missing parts of the reconstructed surface and generate a list of candidate sensor configurations to cover them in the identified free space. This surface-based approach tackles directly the unknown surface, drastically reducing the number of configurations to evaluate for the information planning, compared to classical sampling based methods (cf. Sect. 1.4.3). We propose an Next-Best-View (NBV) planning method which allows to visit all these configurations and which solves a Travelling Salesman Problem (TSP) based on the known map: an optimal path visiting the viewpoint configurations is computed by taking the surface coverage they provide and their location in space, into account. As the robot follows the path, it performs successive scans and completes the map. The initial path is updated by leveraging the gathered information, until the whole unknown surface is covered. In summary, the original contributions of this chapter are:

- A pure volumetric representation to measure the quality of reconstruction and to identify the unknown or incomplete areas from which new viewpoint configurations, can be generated.
- A novel objective function for NBV planning, which incorporates the surface-based information gain at each robot configuration.
- A new algorithm, based on the computation of successive (approximate) solutions to the TSP in conjunction with a Probabilistic Roadmap planner (LazyPRM*, [Hauser 2015]), to visit the clusters of viewpoint configurations for surface reconstruction.
- A validation of the new algorithm with extensive numerical experiments over two realistic 3D environments using ROS-Gazebo, in the presence of depth-map uncertainties, and a comparison with state-of-the-art approaches.

Real-world experiments have also been performed to validate the approach but they will be presented in a dedicated chapter (Chapter 5). The material of this chapter has been presented at the IFAC World Congress in 2020 [Hardouin *et al.* 2020b].

2.2 Problem formulation

In this chapter, we consider a single mobile agent. We will use $\mathbf{q} \in \mathrm{SE}(m)$ to denote the pose of the agent, and $\mathbf{q}_0$ its initial configuration: $m = 2$ in the case of ground vehicles, and

$m = 3$ in the case of aerial vehicles. We assume that the agent is equipped with an accurate localization system which allows to estimate its pose with respect to a global reference frame, and that a robust low-level trajectory-tracking algorithm is available. The agent is equipped with a forward-facing depth sensor with limited Field-of-View (FOV) and sensing range, extrinsically calibrated with respect to its body frame. The objective of the agent is to scan an unknown 3D environment (for instance, a building), characterized by its surface. A mapping algorithm is required to build a representation of the reconstructed surface and identify the free space for navigation. We consider a volumetric mapping (Sect. 1.3.2), which allows to build a map M as a 3D grid, i.e. a collection of discretized space elements arranged along a 3D regular lattice. These elements, referred to as *voxels* $\mathbf{v} \in M$, represent *unknown*, *occupied* and *empty* space. Let $X \subset M$ be the set of unknown voxels and $A \subset M$ the set of known voxels such that $X \cap A = \emptyset$. Moreover, let $O \subset A$ be the set of occupied voxels, and $E \subset A$ the set of *empty* voxels. The goal of the scanning process is to discover unknown voxels: in particular, a voxel is said to be *scanned* when a scan changes its state to occupied or empty. The incompleteness of the surface model is defined as follows:

Definition 1 (**Incomplete Surface Element (ISE)**). *We call* Incomplete Surface Element*, or ISE for short, a voxel* $\mathbf{v} \in M$ *lying on the surface at a frontier, near both the unknown and empty space. Let* C *be the set of all ISEs. A voxel* $\mathbf{v} \in C$ *if it respects all the following propositions:*

$$
\begin{aligned}
&a)\ \mathbf{v} \in E, && (\textit{empty})\\
&b)\ \exists\, \mathbf{u} \in \mathcal{N}_{\mathbf{v}}^{6}\ \ s.t.\ \ \mathbf{u} \in X, && (\textit{unknown})\\
&c)\ \exists\, \mathbf{o} \in \mathcal{N}_{\mathbf{v}}^{18}\ \ s.t.\ \ \mathbf{o} \in O, && (\textit{occupied})
\end{aligned}
$$

where $\mathcal{N}_{\mathbf{v}}^{6}$ *and* $\mathcal{N}_{\mathbf{v}}^{18}$ *denote the 6- and 18-connected voxel neighborhoods of* $\mathbf{v}$*, respectively.*

Definition 2 (**Remaining incomplete surface**). *Let* Q *be the set of all collision-free configurations* $\mathbf{q}$ *of an agent, and let* $Q_c(\mathbf{v}) \subseteq Q$ *be the set of all configurations from which an ISE* $\mathbf{v} \in C$ *can be scanned. The* remaining incomplete surface *is then defined as*

$$
C_{\text{rem}} = \bigcup_{\mathbf{v} \in C} \left\{ \mathbf{v} \mid Q_c(\mathbf{v}) = \emptyset \right\}.
$$

We will use the function $\mathbf{p}_{j,k}(s) : [0, 1] \rightarrow \mathrm{SE}(m)$, $m \in \{2, 3\}$, to define the path of the agent from configuration j to configuration k, where $\mathbf{p}_{j,k}(0) = \mathbf{q}_j$ and $\mathbf{p}_{j,k}(1) = \mathbf{q}_k$. We assume that $\mathbf{p}_{j,k}(s)$ is collision-free and feasible for the agent (i.e. the kinematic/dynamic constraints of the vehicle are satisfied along the path). The problem studied in this paper can be then formalized as follows.

Problem 1 (**Inspection problem**). Consider an agent with initial configuration $\mathbf{q}_0 \in Q$. The inspection problem is to find the path sequence $\mathbf{P}_{0,f} = \{\mathbf{p}_{0,1}, \ldots, \mathbf{p}_{f-1,f}\}$, formed of collision-free paths $\mathbf{p}_{k-1,k}(s)$ visiting the poses $\mathbf{q}_k$, $k \in \{1, 2, \ldots, f\}$, *which allows the agent to scan the set* $C_{\text{ins}} = C \setminus C_{\text{rem}}$ *of all ISEs contained in the current reconstructed map* M.

Progressing along its path sequence $\mathbf{P}_{0,f}$, the agent is able to disclose the unknown space, discover new ISEs, and iteratively solve the inspection problem until $C_{\text{ins}} = \emptyset$. Based on these premises, in the next section, we will provide a general overview of the approach proposed in this work to solve Problem 1.

2.3 System overview

The single-robot system for the reconstruction of the 3D surface of an unknown environment is designed to accommodate any type of robot (ground or aerial) as shown in Fig. 2.1, where computation can be centralized on a ground station or on robot's embedded computer. This architecture is constituted of three complementary modules: a *perception module* (green block), which extracts the ISEs (cf. Sect. 2.2, Definition 1) from a volumetric map reconstructed online, a *planning module* (blue block) that is in charge of computing the path of the robot, and a *low-level-planner* (orange block) which locally manages the visit of clusters and ensures navigation safety.

In the *perception module*, the mapping module takes as input the depth map sensed by the robot paired with their associated sensor pose and integrate them into a 3D volumetric map. The TSDF mapping method (cf. Sect. 1.3.3) has been chosen since its allows to implicitly represent a surface in a volumetric map. Thus, it can be used for both surface completeness evaluation, i.e. ISE extraction, and for navigation, i.e. collision-free path planning and tracking. During the exploration, the map is incrementally built and the robot detects sets of incomplete surfaces or contours by extracting ISEs. The configurations which allow to complete them are generated in the free space, depending on the

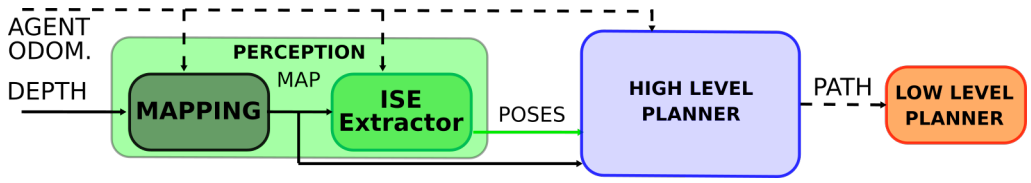


Figure 2.1: *General flowchart of the single-robot surface-reconstruction architecture.* The internal structure of the perception and planning modules is shown inside the green-, and blue-shaded boxes, and the intra- and inter-block connections are represented with solid and dashed lines, respectively.

type of environment and camera properties (range, resolution), and sent to the *planning module*.

The *planning module*, or *high-level planner*, aggregates these configurations depending on their location in the 3D space. Each cluster is evaluated by estimating the information gain of each of its poses using a ray-casting method. A weighted directed graph is created and continuously expanded where the weight of the arc represents the travel utility between clusters in the free space. This utility, used in a single objective function, trades off travel cost and information gain of visiting a new cluster. The path which maximizes the utility and ensures collision-free navigation is extracted solving a variant of Traveling Salesman Problem. As the map evolves, new ISEs are discovered, and the path to complete them is updated. The 3D model is considered complete when there are no more ISEs.

The *low-level-planner* generates a low-level path between a starting and an ending pose in the free space using a probabilistic planner and sends it to controller. It also monitors the exploration, asks for replanning and commands emergency stops in case of obstacle detection. The modules are described in more detail in the next two sections.

2.4 Perception

This section details the perception module. The surface-based mapping used for both mapping and navigation is presented, and the definition of ISE related to this representation, is introduced. We also present the mechanism to generate sensor viewpoints and how they are grouped into clusters for planning.

2.4.1 Surface-based mapping

A volumetric map M, here based on a TSDF representation, is used to detect unknown areas: the ISEs (see Sect. 2.4.2). The TSDF map [Curless & Levoy 1996, Newcombe *et al.* 2011a] consists of a voxel grid, where each voxel contains a truncated signed distance value $\phi \in \mathbb{R}$ and a weight $w \in \mathbb{R}^+$. It implicitly represents surfaces, which correspond to the zero level set of the distance field: hence, the TSDF volume is a *volumetric representation of a surface*. Algorithms such as MarchingCubes [Lorensen & Cline 1987] can be used to extract a triangular mesh which is an explicit representation of those surfaces, e.g. for visualization. This map is built in an incremental fashion by sequentially integrating depth measurements, see Sect. 1.3.3 for more details. In order to keep the map consistent, the pose $\mathbf{q}$ of the sensor must be used to express the depth measurements in the reference frame of the map. This

pose is supposed to be provided by a localization system, e.g. a visual SLAM algorithm [Engel *et al.* 2015, Mur-Artal & Tardós 2017], with sufficient accuracy. In order to take sensor noise into account [Nguyen *et al.* 2012, Oleynikova *et al.* 2020], the new measurements are weighted with $1/z_{\mathbf{q}}^2(\mathbf{v})$, where $z_{\mathbf{q}}(\mathbf{v})$ is the distance between voxel $\mathbf{v}$ and the current pose $\mathbf{q}$. The state of a voxel $\mathbf{v}$ is set to *known* (either *occupied* or *empty*) if $w(\mathbf{v}) \geq W_{\text{th}}$ and to *unknown* if $w(\mathbf{v}) < W_{\text{th}}$, where the threshold W_{th} depends on the sensing range of the depth sensor.

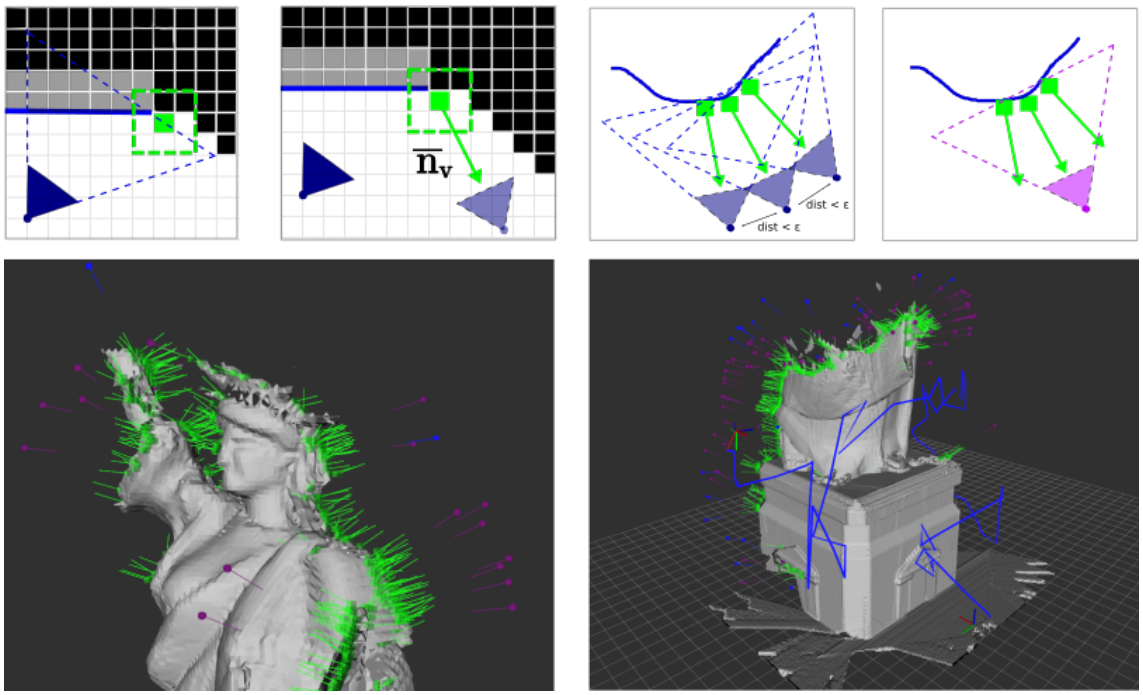


Figure 2.2: *ISEs and viewpoint configurations:* [top-left] Two-dimensional example of ISE $\mathbf{v}$ (filled green square). Its 2D neighborhood is represented by a dashed green square. Unknown voxels are black, occupied are gray, and empty voxels are white. The reconstructed surface is depicted as a blue segment, and the sensor configuration and its frustum as dark blue triangle. Direction from the contour, $\overline{\mathbf{n}}_{\mathbf{v}}$, and corresponding viewpoint configuration at distance δ_{pose} from $\mathbf{v}$ (light blue triangle); [top-right] Graphical illustration of merging process where the three blue configurations are fused into the purple one; [bottom] Snapshots of a simulated reconstruction in progress, with ISEs and their direction from the contour (green arrows) and sensor configurations (blue/purple dots and arrows). The blue line represents the path traveled by the aerial robot.

2.4.2 ISE extractor and viewpoint generation

Inspired by [Monica & Aleotti 2018], a voxel $\mathbf{v} \in M$ is considered as an ISE, i.e. $\mathbf{v} \in C$ as stated in Definition 1, if it verifies the following conditions:

$$
\begin{aligned}
&a)\ w(\mathbf{v}) \geq W_{\text{th}} \ \wedge\ \phi(\mathbf{v}) > 0, && \text{(empty)}\\
&b)\ \exists\, \mathbf{u} \in \mathcal{N}_{\mathbf{v}}^{6} \ \text{ s.t. } \ w(\mathbf{u}) < W_{\text{th}}, && \text{(unknown)}\\
&c)\ \exists\, \mathbf{o} \in \mathcal{N}_{\mathbf{v}}^{18} \ \text{ s.t. } \ w(\mathbf{o}) \geq W_{\text{th}} \ \wedge\ \phi(\mathbf{o}) \leq 0. && \text{(occupied)}
\end{aligned}
$$

Definition 3 (Scanned element). *A voxel* $\mathbf{v} \in M$ *which satisfies* $w(\mathbf{u}) \geq W_{\text{th}},\ \forall\, \mathbf{u} \in \mathcal{N}_{\mathbf{v}}^{6}$, *is called a* scanned element.

The direction $\overline{\mathbf{n}}_{\mathbf{v}}$ to observe the ISE $\mathbf{v}$, is determined from the gradient of the weight function $\nabla w(x, y, z)$, which can be computed as

$$
\mathbf{n}_{\mathbf{v}} = \sum_{\mathbf{c} \in \mathcal{N}_{\mathbf{v}}^{26}} w'(\mathbf{c})\, \frac{\mathbf{c} - \mathbf{v}}{\|\mathbf{c} - \mathbf{v}\|}\,, \qquad \overline{\mathbf{n}}_{\mathbf{v}} = \frac{\mathbf{n}_{\mathbf{v}}}{\|\mathbf{n}_{\mathbf{v}}\|}\,,
$$

where $\mathcal{N}_{\mathbf{v}}^{26}$ is the 26-connected neighborhood of $\mathbf{v}$ and the weight function

$$
w'(\mathbf{c}) = \begin{cases} -W_{\text{th}} & \text{if voxel } \mathbf{c} \text{ is occupied,} \\ W_{\text{th}} & \text{otherwise.} \end{cases}
$$

A sensor configuration is generated along the direction $\overline{\mathbf{n}}_{\mathbf{v}}$ at a distance δ_{pose} from the corresponding voxel $\mathbf{v}$ (see Fig. 2.2-[top-left]). The sensor is oriented towards $\mathbf{v}$ along $-\overline{\mathbf{n}}_{\mathbf{v}}$, and the value of δ_{pose} should be chosen to scan the voxel neighborhood of the ISE, it depends on the sensing range of the depth sensor. If two poses $\mathbf{q}_j, \mathbf{q}_k \in Q$ generated from the ISEs $\mathbf{v}_j, \mathbf{v}_k \in C$, respectively, turn out to be located within a short distance (i.e. $\text{dist}(\mathbf{q}_j, \mathbf{q}_k) < \varepsilon$, for a small $\varepsilon > 0$), and with their viewing directions $\overline{\mathbf{n}}_{\mathbf{v}_j}$, $\overline{\mathbf{n}}_{\mathbf{v}_k}$ almost parallel (i.e. $|\overline{\mathbf{n}}_{\mathbf{v}_j} \cdot \overline{\mathbf{n}}_{\mathbf{v}_k}| \simeq 1$, where "$\cdot$" denotes the dot product), these configurations are merged into a single viewpoint by averaging their positions and orientations. This allows to cover multiple ISEs with fewer scans, reducing the overall number of poses to visit without losing key information, see Fig. 2.2-[top-right].

2.4.3 Cluster of configurations

Even after the merging step described in the previous section, a huge number of candidate poses for observing the ISEs may be generated in large-scale environments, which is not compatible with the planning objective. To overcome this issue, neighbour viewpoints are grouped into *clusters* U_j, $j \in \{1, 2, \ldots, N_c\}$, which can be assigned to an agent more

efficiently. The set of all such clusters is denoted by $\mathcal{U} = \{U_1, U_2, \ldots, U_{N_c}\}$. A configuration $\mathbf{q}_l$ belongs to a generic cluster U if $\exists\, \mathbf{q}_j \in U$ such that $\mathrm{d}(\tau_l^j) < d_v$, where $\mathrm{d}(\tau_l^j)$ denotes the length of the path τ_l^j between $\mathbf{q}_l$ and $\mathbf{q}_j$ on a directed graph (as defined next in Sect. 2.5.1), and d_v is an upper bound on the distance. If no neighbors are found, d_v is increased up to a maximum value $d_v^{\max}$. Once the clusters have been defined, their respective level of informativeness has to be quantified since each cluster, in general, will not contain the same number of viewpoints. To evaluate a configuration, we use the ray-tracing method [Bresenham 1965]. The ray tracing consists of tracing a set of rays inside the voxel map. The ray tracing is configured according to the sensor parameter, such as the field of view, the resolution (rows × columns), the sensor view angle and the depth of view. When a ray "touches" a voxel (as distinct from empty voxels), the attributes from this voxel are stored. After ray tracing, we have all the information about the voxels that are visible from the candidate view. From a frontier-based perspective, i.e. the ISEs that can be seen are counted. Let $C_{\mathbf{q}}$ be the set of all ISEs seen from viewpoint $\mathbf{q}$ and let $C_U = \bigcup_{\mathbf{q} \in U} C_{\mathbf{q}}$. The *gain* $g(U)$ of the cluster U is defined as:

$$g(U) = \frac{|C_U|}{|C|}, \tag{2.1}$$

where $|C_U|$ is the cardinality of the set C_U.

2.5 Single-robot planning

In this section, the NBV planning approach is expressed. We present our two-level planning algorithm for addressing the inspection problem with a single robot. A high-level graph-based planner is used to schedule the visit of clusters. It relies on an approximate solution of a variant of Traveling Salesman Problem. The planned tour of the clusters is sent to the low-level planner which ensures safe navigation and monitors path update depending on map evolution.

2.5.1 Graph formulation

To rigorously state the problem, the high-level planner schedules the visit of clusters according to the TSDF map. The weighted directed graph $\mathcal{G} = (\mathcal{U}, \mathcal{E}, \{a_{UV}\}_{(U,V)\in\mathcal{E}})$ is defined, where $\mathcal{U}$ is the set of clusters, $\mathcal{E}$ is the set of edges, and $\{a_{UV}\}_{(U,V)\in\mathcal{E}}$ is the collection of weights associated with the edges. Each directed edge $e_{UV} \in \mathcal{E}$ connects cluster U to cluster V, with $U, V \in \mathcal{U}$. It is assumed that the initial configuration of the agent belongs to one of the clusters of $\mathcal{G}$, i.e. $\{\mathbf{q}_0\} \in \mathcal{U}$. Let $\mathbf{q}_k$ be a configuration of cluster

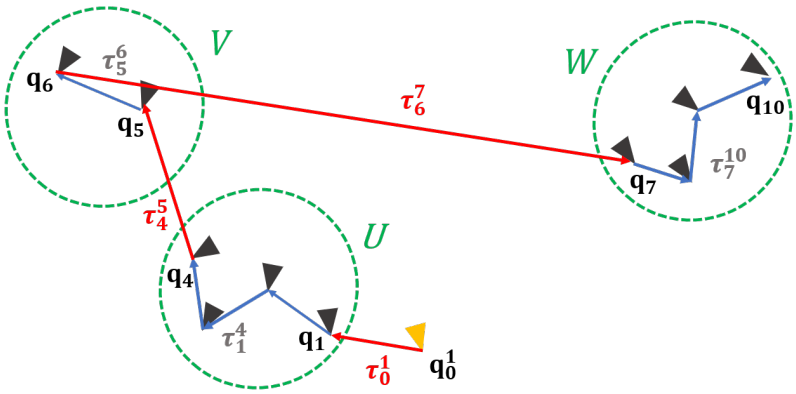


Figure 2.3: *Two-dimensional representation of a weighted directed graph $\mathcal{G}$ with three clusters U, V, $W \in \mathcal{U}$ (dashed green circles).* The first and third cluster contain four configurations, and the second cluster, two configurations (blue triangles). The computed path $\mathbf{p}_0^{10}$ from the initial configuration $\mathbf{q}_0$ (yellow triangle) to the final configuration $\mathbf{q}_{10}$, is red between the clusters, and blue inside them.

U, and $\mathbf{q}_l$, $\mathbf{q}_m$ two configurations of cluster V. Then, the weight a_{UV} between cluster U and V is the 6-tuple defined as,

$$a_{UV} = \left\{\tau_k^l,\ \tau_l^m,\ g(V),\ \mathrm{d}(\tau_k^l),\ \mathrm{d}(\tau_l^m),\ f_{UV}\right\}, \tag{2.2}$$

where

- τ_k^l denotes the path from $\mathbf{q}_k \in U$ to $\mathbf{q}_l \in V$, i.e. the path between cluster U and cluster V (see the red paths in Fig. 2.3),
- τ_l^m denotes the shortest Hamiltonian path [Godsil & Royle 2001] including configurations of V, which starts at $\mathbf{q}_l$ and ends at $\mathbf{q}_m$, computed with a 3-opt heuristic (see the blue paths in Fig. 2.3),
- $g(V)$ is the gain of cluster V, as defined in (2.1),
- $\mathrm{d}(\tau_k^l)$ is the cost associated with the inter-cluster path τ_k^l, i.e. the length of τ_k^l,
- $\mathrm{d}(\tau_l^m)$ is the cost associated with the intra-cluster path τ_l^m, i.e. the length of τ_l^m,
- f_{UV} is the *utility function* defined as,

$$f_{UV} \;=\; g(V)\exp\left(-\lambda_{\mathrm{tc}}\,\mathrm{d}(\tau_k^l) - \lambda_{\mathrm{ic}}\,\mathrm{d}(\tau_l^m)\right), \tag{2.3}$$

  where λ_{tc} and λ_{ic} are positive penalty terms for the inter-cluster and intra-cluster costs, respectively, which can be used to promote the visit of clusters far apart or large clusters. Their value depends on the capabilities of the agents (i.e.

ground robot vs. aerial robot). A similar utility function was originally proposed in [González-Banos & Latombe 2002].

The paths τ_k^l and τ_l^m are computed using the Euclidian distance between two configurations. However, the shortest path in the free space may be alternatively computed with a graph-based planner to ensure a more accurate distance estimation. The weights on the directed graph $\mathcal{G}$ in (2.2), quantify the potential benefit of choosing a path, to pursue the 3D reconstruction: in fact, the higher the value of the function f_{UV}, the more advantageous is the path. Note that $f_{UV} > 0$, since $g(V) > 0$. A path can be extracted from this graph as shown in Fig. 2.3.

2.5.2 Inspection problem

The inspection problem for a single robot can be stated as a maximum Open Asymmetric Traveling Salesman Problem (maxATSP). It consists in finding a maximum-utility Hamiltonian path on $\mathcal{G}$, i.e. a path which visits every configuration exactly once, maximizing the sum of the weights on the arcs f_{UV}, $U \neq V$, starting from the initial configuration $\mathbf{q}_0$ and ending to $\mathbf{q}_f$, with $\mathbf{q}_f \neq \mathbf{q}_0$. In what follows, we denote by **maxATSP**($\mathcal{U}$), the set function that takes the set of clusters $\mathcal{U}$ as input and outputs its utility value $p \in \mathbb{R}^+$, from which the path sequence $\mathbf{P}_{0,f}$, e.g. the viewpoint sequence to be visited, can be computed. A linear programming formulation of the **maxATSP**($\mathcal{U}$) is:

$$\textbf{Maximize} \quad \sum_{U \in \mathcal{U}} \sum_{V \in \mathcal{U}} f_{UV}\, x_{UV} \tag{2.4a}$$

$$\textbf{subject to} \tag{2.4b}$$

$$x_{UV} \in \{0,1\}, \qquad U, V \in \mathcal{U},\ U \neq V, \tag{2.4c}$$

$$\sum_{U \in \mathcal{U},\, U \neq V} x_{UV} = 1, \qquad V \in \mathcal{U} \setminus \{\mathbf{q}_0\}, \tag{2.4d}$$

$$\sum_{V \in \mathcal{U} \setminus \{\mathbf{q}_0\},\, V \neq U} x_{UV} \leq 1, \qquad U \in \mathcal{U}, \tag{2.4e}$$

$$\sum_{V \in \mathcal{U} \setminus \{\mathbf{q}_0\},} x_{\{\mathbf{q}_0\}V} = 1, \qquad \{\mathbf{q}_0\} \in \mathcal{U}, \tag{2.4f}$$

$$\sum_{U \in Q} \sum_{V \in Q,\, V \neq U} x_{UV} \leq |Q| - 1, \qquad \forall\, Q \subsetneq \mathcal{U},\ |Q| > 2\,, \tag{2.4g}$$

where $\{\mathbf{q}_0\}$ is the cluster which only contains the starting configuration, Q is a proper subset of $\mathcal{U}$, and

$$x_{UV} = \begin{cases} 1 & \text{if the arc belongs to the optimal path,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

The maximization of the objective function (2.4a) is subject to multiple constraints:

- (2.4d) forces the agent to visit all clusters,
- (2.4e) ensures that the agent stays on the last visited cluster, and does not come back to $\{\mathbf{q}_0\}$.
- (2.4f) ensures that the agent starts its tour exactly once,
- (2.4g) avoids the presence of subtours.

However, solving Problem (2.4a) using a classical LP solver is computationally expensive and time consuming. For real-time solutions, we then opted for the "V-opt" Lin-Kernighan heuristic [Lin & Kernighan 1973] via the Lin-Kernighan-Helsgaun (LKH) implementation of [Helsgaun 2000] which provides optimal solutions to various TSP variation within $O(n^{2.2})$ time, with n the number of nodes, even for non-metric problem as considered here (see Appendix B.2). The resolution is performed after having converted the Asymmetric Traveling Salesman Problem (ATSP) into a symmetric TSP via [Jonker & Volgenant 1983].

Successive scans along the path allow to generate new viewpoint configurations, and the information gain associated to the visit of clusters, changes as the map grows. As ISEs are updated, the clusters are sent by the perception module, a path is replanned and sent to the low-level planner. The reconstruction process stops when no more ISEs remain.

2.5.3 Low-level planner

The low-level planner receives the sequence of clusters to visit $\mathbf{P}_{0,f}$, or $\mathbf{P}$ for short, and the coordinates of their respective configurations, computed by solving (2.4a) for a given update of the TSDF map M. Its main purpose is to compute the local path in the free-space of the agent using the Lazy PRM* algorithm (cf. Sect. 1.4.3) and send it to the robot's Controller. **Algorithm 1** describes the procedure. The low-level planner consists of multiple subfunctions:

1. Given a start and an end pose, $\mathbf{q}_{curr}$ and $\mathbf{q}$ respectively, it computes the robot *path* $\mathbf{p}^i_{curr,q}$ (i.e. the sampled set of consecutive robot configurations) in the free space given by the TSDF volume. According to the type of robot, the paths are computed in the 3D space (aerial vehicles), or in the 2D space (ground vehicles). If the goal configuration can be reached by the robot, the path $\mathbf{p}^i_{curr,q}$ is sent to the controller (*feasible* is set to *True* by **LazyPRM***).

2. The process is repeated and a new path is put on hold until the current one has been fully traveled (*reached* set to *True* by **GoalReached**), as new ISEs are uncovered.

3. As the robot visits viewpoints and completes the ISEs, the remaining utility is computed to evaluate the interest of pursuing the current path (*update* set to *True* by **UtilityCheck**). The utility check consists in comparing the number of ISEs left C, with the initial set of ISEs C_{curr} covered by the given path. It stops and waits for a new path, if a certain amount (c_{thres}) of initial ISEs has been completed.

4. Finally, the low-level planner ensures an emergency *stop* via a distance threshold to the surface (obtained from the TSDF), if a new obstacle is detected along the path (*obstacle* set to *True* by **ObstacleCheck**). The robot's safe space is modelled, around its position, as a sphere if it is an aerial vehicle, or an ellipsoid if it is a ground robot (where the greater semi-axis is parallel to the orientation of the robot). As the robot navigates, the distances from its position to near occupied voxels (i.e. objects) are computed and the emergency stop is triggered if one voxel reaches the ellipsoid or the sphere. For this, we consider a collision distances which correspond to the radius of the sphere for an aerial vehicle and the semi-major and semi-minor axis of the ellipsoid for a ground robot.

Algorithm 1: Single-robot low-level planner

```
1  Inputs: q_curr, Q, P, M, C ;
2  Set reached = False; obstacle = False; update = False ;
3  C_start ← C ;
4  foreach q ∈ Q , following P, do
5      update ← UtilityCheck(C_start, C) ;
6      if update then
7          wait for high-level re-planning ;
8          break ;
9      {feasible, p^i_{curr,q}} ← LazyPRM*(q_curr, q, M) ;
10     if feasible then
11         send p^i_{curr,q} to Controller ;
12         while (not reached) and (not obstacle) do
13             reached ← GoalReached(q_curr, q) ;
14             obstacle ← ObstacleCheck(q_curr, M) ;
15         if obstacle then
16             send stop to Controller ;
17             wait for high-level re-planning ;
18             break ;
```

2.6 Numerical experiments

The proposed surface-driven method has been validated via realistic numerical experiments. For these simulations, we considered multirotor UAV with 4 degrees of freedom: its 3D position $[x, y, z]^T$ and its yaw angle ψ. Two large-scale environments have been selected. We chose a benchmark industrial plant, the **Powerplant** model, which is widely used in the volumetric exploration literature. In order to study the impact of the penalty terms in (2.3), on the reconstruction accuracy/completeness, we also considered a monumental model of the famous **Statue of Liberty** (**SoL**). The majority of existing methods in the literature come with complete robotic setups for 3D reconstruction but all the implementation details and the code of the authors are rarely available. On the other hand, a paper-based reproduction of these methods is generally very time consuming and it may lead to unintentional discrepancies which might bias the comparative analysis. For a fair evaluation, we then decided to compare our approach with pertinent state-of-the-art methods based on numerical results directly given in their papers.

2.6.1 Simulation setup

To simulate the aerial vehicle and the environment, we used the ROS middleware coupled with the Gazebo simulator[1], see Sect. A.2.4 in Appendix A. For the hexarotor UAV with an on-board stereo camera, we leveraged the model provided by the RotorS simulator [Furrer *et al.* 2016]. The experiments have been conducted with and without depth-map uncertainty. A Gaussian-noise model is used here to represent this uncertainty. The standard deviation associated with a pixel corresponds to the depth-value sensing error of the corresponding point located at a distance z, i.e.

$$\sigma(z) = \frac{|e_d|}{fB} z^2 \,,$$

where $|e_d|$ is the magnitude of the disparity error, f the focal length in pixels, and B the baseline of the stereo camera in meters. Following [Nguyen *et al.* 2012, Keselman *et al.* 2017], the raw depth map was smoothed out by using a 3×3 Gaussian kernel. The TSDF volume was generated with the algorithm proposed in [Zeng *et al.* 2017], where the reconstruction is performed with MarchingCubes [Lorensen & Cline 1987] and the weight increment has been modified as detailed in Sect. 2.4.1. The path of the UAV is computed with the LazyPRM* planner from the Open Motion Planning Library (OMPL) [Şucan *et al.* 2012], which finds the shortest path between two configurations by sampling random robot configurations in the free space and taking the structure

[1] **ROS**: https://ros.org/ , **Gazebo**: http://gazebosim.org/

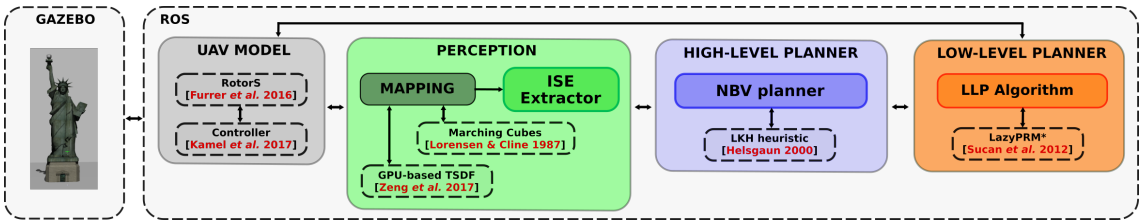


Figure 2.4: *Simulation stack*: Gazebo simulator interacts with the ROS architecture. Components described in the literature (dash-dotted bubble) are adapted and integrated into our software nodes: **Mapping**, **ISE Extractor**, **NBV planner** and **Low-level planner algorithm** (solid-line colored bubble).

of the TSDF map into account (the collision radius is 1 m). In terms of performance, LazyPRM* allows multi-query path planning to all destination points, which is useful for reachable-path checking and distance evaluation, because of the reduced computational complexity with respect to RRT (the average runtime is below 1 s). Model Predictive Control (MPC) [Kamel *et al.* 2017] has been used to track the generated paths, the reference translational velocity being fixed at 0.6 m/s.

The code used to model the aerial vehicle, includes open-source implementations of the algorithms mentioned above, which are integrated into our ROS software architecture as depicted in Fig. 2.4. The simulation stack interacts with Gazebo which simulates the 3D model of the environment to reconstruct, and the robot behaviour based from the model provided by RotorS. The ROS-nodes are implemented in Python and C++, following the architecture presented in the system overview (Sect. 2.3) with three main functional modules: *Perception module*, *High-level planner* and *Low-level planner*. This modular architecture allows the nodes to be easily modified if needed. All nodes are designed to run on CPU, except for the TSDF mapping which is GPU-based for fast depth map integration. We ran our algorithm on a Dell Precision 7520 with 2.90 GHz Intel Core i7 processor, 16 GB RAM and Quadro M2200 graphics card. The virtual environments considered in the

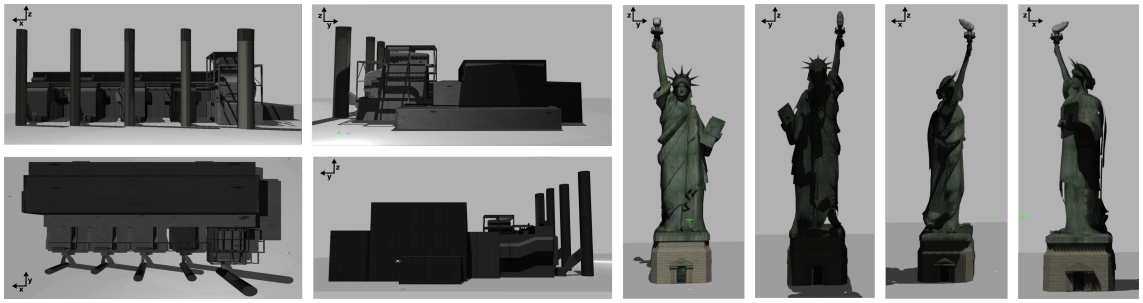


Figure 2.5: *Gazebo simulation environments*: [left] **Powerplant**, Powerplant; [right] **SoL**, the Statue of Liberty.

experiments, **Powerplant** and **SoL**[2] are shown in Fig. 2.5. The **Powerplant** model has been scaled to fit in a box of size $65 \times 42 \times 15\,\mathrm{m}^3$ (as a result, the five flues have the same height). Because of its narrow passages, high walls and roof, large flues and thin gantries, **Powerplant** is challenging for both navigation and reconstruction (occlusion problem). In **SoL**, we considered a model of the Statue of Liberty ($20 \times 20 \times 60\,\mathrm{m}^3$), which contains multiple sharp edges and fine details (diadem and gown). The simulation parameters used in the two scenarios are reported in Table 2.1.

Table 2.1: Parameters used in the numerical experiments.

Parameter	**Powerplant**	**SoL**
Voxel resolution $r_\mathbf{v}$ [m]	0.3	0.15
Threshold W_{th}	0.3	0.3
$e_{\max}$ [m]	0.2598	0.1299
Camera range [m]	[1.6, 8]	[1, 5]
Camera FOV [deg.] (H, V)	90×60	90×60
e_d [pixels]	0.1	0.1
f [pixels]	376	376
B [m]	0.11	0.11
Collision radius [m]	1	1
UAV nominal speed [m/s]	0.6	0.6
δ_{pose} [m]	4.7	3.6
d_v [m]	2.0	2.5
$d_v^{\max}$ [m]	5	5
Penalty term λ_{tc}	0.3	0.17
Penalty term λ_{ic}	0.03	0.15
c_{thres} [%]	80	50

The voxel resolution $r_\mathbf{v}$ depends on the volume of the environment to scan and on the presence or absence of mesh details. Scanning large volumes is computationally intensive, and the depth map integration into the TSDF is directly related to the voxel resolution. A low resolution leads to faster volume computation, but large voxels are unsuitable to model fine details, which negatively impacts the mesh quality: a trade-off between voxel resolution and computational cost should be then found empirically, and the camera range should be fixed accordingly. The camera range is adapted from this resolution. The upper bound $d_v^{\max}$ on the distance between cluster configurations d_v, changes during the reconstruction, and its default value has been determined empirically by considering the spatial distribution of viewpoint configurations.

[2]**Powerplant**: http://models.gazebosim.org/, **SoL**: https://free3D.com/

2.6.2 Metrics

In order to obtain statistically-significant values, 10 trials per scenario have been carried out with an identical experiment setup e.g. the same initial pose for the robot. As in the exploration methods (cf. Sect. 1.5.3), we evaluated the total length of the path of the UAV, and the completion time, i.e. the time necessary to cover the 3D environments. The reconstructed 3D surface has been evaluated with CloudCompare[3] using the Multiscale Model to Model Cloud Comparison (M3C2) algorithm [Lague *et al.* 2013]. To quantify how well the surface has been recovered, dense point clouds were sampled on the reconstructed and ground truth (GT) meshes (10000 pt/m^2), and their deviation was measured by performing a cloud-to-cloud comparison (see Fig. 2.6). For a fair evaluation, we pruned all the invisible surfaces of the GT mesh beforehand (e.g. the interior floor and walls), and we restricted our analysis to the exterior surface mesh only. We consider that a point belonging to the GT point cloud has been covered by a corresponding one in the reconstructed cloud, if their absolute distance is less than the length of the half diagonal of a voxel, i.e. less than $e_{\max} = r_{\mathrm{v}}\sqrt{3}/2$, where r_{v} is the voxel resolution. Note that $2\,e_{\max}$ is the maximum reconstruction error provided by the MarchingCubes algorithm by default [Lorensen & Cline 1987]. The quality of the recovered surface is evaluated by reporting the average and standard deviation of the signed distance error with respect to the GT point cloud, and the Root-Mean-Square Error (RMSE).

2.6.3 Choice of penalty terms

The choice of the inter-cluster penalty term λ_{tc} and the intra-cluster penalty term λ_{ic} appearing in the utility function (2.3), depends on the nature of the 3D environment where the UAV evolves and should be tuned accordingly. Multiple reconstructions of **Powerplant** and **SoL** have been carried out with different combinations of λ_{tc} and λ_{ic}, to find the best suited for each type of environment according to the traveled path. The results are compiled in Tables 2.2 and 2.3, and show that the shape of the environment has an impact on the optimality of the penalty terms. More precisely, in wide box-like environments as **Powerplant**, the ISEs tend to appear in the proximity of occluded regions and sharp edges, and large extents of known surface may separate these zones. To minimize the total distance traveled, inter-cluster utility should then take priority over intra-cluster utility, i.e. $\lambda_{tc} \gg \lambda_{ic}$. On the other hand, the pedestal of the statue excluded, **SoL** predominantly consists of round surfaces and the average distance between two clusters is much smaller than in **Powerplant**. As a consequence, similar penalty terms are preferable (i.e. $\lambda_{tc} \simeq \lambda_{ic}$).

[3] https://www.danielgm.net/cc/

Table 2.2: Tuning of the penalty terms: **Powerplant**

	Powerplant				
	$\lambda_{tc} \gg \lambda_{ic}$		$\lambda_{tc} \approx \lambda_{ic}$	$\lambda_{tc} \ll \lambda_{ic}$	
λ_{tc}	0.35	0.3	0.15	0.03	0.01
λ_{ic}	0.01	0.03	0.15	0.3	0.35
Path length [m]	787	**780**	795	814	822

Table 2.3: Tuning of the penalty terms: **SoL**

	SoL				
	$\lambda_{tc} \gg \lambda_{ic}$		$\lambda_{tc} \approx \lambda_{ic}$	$\lambda_{tc} \ll \lambda_{ic}$	
λ_{tc}	0.35	0.3	0.17	0.03	0.01
λ_{ic}	0.01	0.03	0.15	0.3	0.35
Path length [m]	578	559	**550**	587	601

2.6.4 Results and discussion

The results of the numerical experiments are reported in Table 2.4 and a snapshot of 3D reconstruction is shown in Fig. 2.6. In what follows, we will refer to the results of the single-UAV planner with perfect measurements as *SR* and to those with noisy depth measurements as SR^*.

Compared to the noise-free case, noisy depth measurements have a negligible effect on the trajectory of the UAV and on the completion time, but as expected, a substantial degradation of reconstruction quality and coverage is observed. Compared to the state-of-the-art, the algorithm proposed by [Song & Jo 2018] exhibits similar completion times (around 35 min) to ours (32 min for *SR*, and 33 min 10 s for SR^*) in the **Powerplant** scenario. The gap is larger with **SoL**: in fact, our algorithm took 36 min for *SR* and 37 min for SR^*, while that of [Song & Jo 2017], around 53 min. However, in our case, the trajectory of the quadrotor UAV is longer (780 m for *SR* / 785 m for SR^* vs 324 m, in **Powerplant**), and more jagged. This is not surprising, since the viewpoint configurations have been generated from the unknown surface and not for navigation purposes as in [Song & Jo 2018]. Moreover, no trajectory refinement (e.g. smoothing) is performed. Nevertheless, our method guarantees that all the regions that are accessible to the UAV are covered. In addition, in keeping with the recent analysis in [Schmid *et al.* 2020], it turns out to be competitive with the state-of-the-art approaches in terms of overall 3D reconstruction quality ([Schmid *et al.* 2020] report an RMSE of 6.4 ± 0.8 cm for a voxel resolution of 0.1 m). However, further work would be needed to perform a comparative study under identical simulation conditions. It is finally worth pointing out here, that the

Table 2.4: Numerical results for the single-UAV system (averages over 10 trials).

Environment	**Powerplant**		**SoL**	
Algorithm	*SR*	*SR**	*SR*	*SR**
Path length [m]	780	785	547	550
Completion time [min.]	32′	33′10″	36′	37′
$e_{\max}$ [cm]	25.98	25.98	12.99	12.99
Surface coverage [%]	91.5	90.4	92.3	91.2
M3C2 Avg. error [cm]	0.14	−0.13	0.29	−0.02
M3C2 Std. dev. error [cm]	5.85	7.51	3.41	3.67
RMSE [cm]	5.86	7.51	3.43	3.67

quality of 3D reconstruction is resolution dependent: in fact, it is inversely proportional to the size of TSDF voxels. A small resolution amounts to a large number of voxels to be integrated in the TSDF map, which is a resource-intensive process. Therefore, if the quadrotor UAV explores a large-scale environment using only on-board sensing and processing, a compromise between reconstruction quality and computational efficiency should be found.

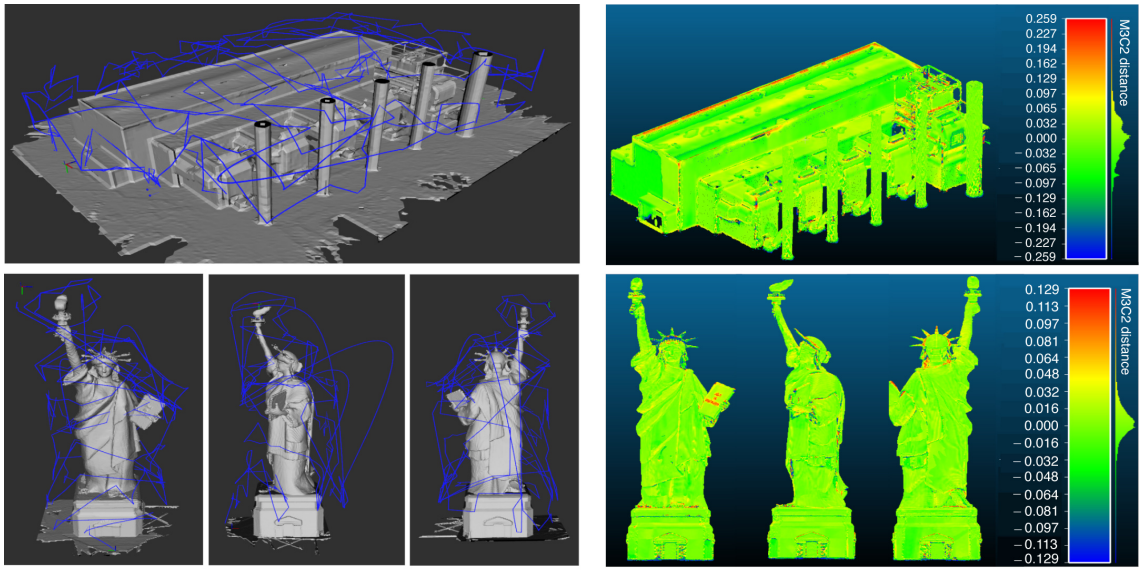


Figure 2.6: *Numerical experiments:* [top] **Powerplant** and [bottom] **SoL**: [left] Reconstructed mesh and 3D exploration path $\mathbf{P}_{0,f}$ (blue) of the quadrotor UAV (*SR*); [right] Signed distance error. The colour coding shows the error in meters with respect to the ground truth, computed with CloudCompare's M3C2 algorithm.

2.7 Conclusion

In this chapter, we have presented a new method which combines the advantages of both volumetric and surface-based planning, for the reconstruction of large scale environments with a single-robot. In particular, a single 3D representation of the environment has been chosen for both surface reconstruction and navigation. Moreover, a two-level path planning strategy with a novel cluster-based 3D reconstruction gain and cost-utility formulation has been proposed. Realistic numerical experiments with ROS and Gazebo have validated the proposed method in two challenging outdoor environments, in the presence of uncertain depth measurements. The results indicate that the algorithm is on a par with or superior to the state-of-the-art methods. However, the completion time and distance traveled remain fairly large, as also our hardware experiments (to be presented in Sect. 5) have confirmed. Considering the current flight autonomy of aerial vehicles, the usage of a single-robot for scanning large-scale real environments may be problematic. Based on these conclusions, the single-robot architecture studied in this chapter will be extended to a multi-robot setting with a centralized coordination, in the next chapter.

Chapter 3

Surface reconstruction: centralized multi-robot architecture

Chapter outline

3.1 Introduction . . . 55
3.2 Problem formulation . . . 56
3.3 System overview . . . 57
3.4 Perception . . . 59
3.4.1 Surface-based mapping . . . 59
3.4.2 ISE extractor, viewpoint generation and clustering . . . 60
3.5 Centralized Planning . . . 60
3.5.1 Inspection problem . . . 60
3.5.2 Low-level planner . . . 64
3.6 Numerical experiments . . . 65
3.6.1 Multi-robot nearest neighbour planner . . . 66
3.6.2 Simulation setup . . . 66
3.6.3 Metrics . . . 67
3.6.4 Results and discussion . . . 68
3.7 Conclusion . . . 71

3.1 Introduction

The numerical experiments presented in the previous chapter, showed that a single robot might take a long time to perform a complete scan, and this motivated us to improve our robotic system. In this chapter, we extend the problem of incremental exploration and surface reconstruction of an unknown 3D environment for inspection purposes, for teams of multiple robots. A centralized architecture will be presented here, and it will be further

extended as a distributed architecture in Chapter 4. Depth maps provided by the agents are gathered in a centralized volumetric mapping module, which identifies occupied and free space. It allows to extract the missing parts of surface, and to generate viewpoints to visit, to complete the scan. The proposed NBV planning method greedily assigns these configurations to the robots, according to the "utility" (i.e. information gain) they locally provide to the team and known map. Optimal paths are computed by solving a variant of the Travelling Salesman Problem (TSP) [Punnen 2007], and updated when unknown areas are completed. The original contributions of this chapter are threefold:

- The single-robot mapping module is adapted to the multi-robot setting using a volumetric representation of surfaces, to cautiously navigate, identify incomplete areas, and evaluate the information gain and the quality of 3D reconstruction.
- A multi-robot surface-based planner is designed to visit viewpoint configurations for surface reconstruction. It relies on a greedy allocation of configurations and on the successive resolution of the TSP. Navigation in free space is ensured by a local planning module based on the probabilistic planner (LazyPRM*, [Hauser 2015]).
- The proposed multi-robot system has been extensively validated in two realistic simulation environments, in the presence of depth-map uncertainties.

Real-world experiments have also been performed to validate the approach and are presented in Chapter 5. The material of this chapter has been presented at the IEEE/RSJ IROS conference in 2020 [Hardouin *et al.* 2020a].

3.2 Problem formulation

We consider a team of N cooperative mobile agents. We will use $\mathbf{q}^i \in \mathrm{SE}(m)$ to denote the pose of agent i, and $\mathbf{q}^i_0$ its initial configuration, $i \in \{1, 2, \ldots, N\}$: $m = 2$ in the case of ground vehicles, and $m = 3$ in the case of aerial vehicles. In a similar way than Chapter 2, we assume that all agents are equipped with an accurate localization system which allows to estimate their pose with respect to a global reference frame, and that a robust low-level trajectory-tracking algorithm is available. Each agent is equipped with a forward-facing depth sensor with limited field-of-view (FOV) and sensing range, extrinsically calibrated with respect to its body frame. The agents should cooperatively scan an unknown 3D environment (for instance, a building), characterized by its surface. A mapping algorithm is required to build a representation of the reconstructed surface and identify the free space for navigation. We consider a volumetric mapping (cf. Sect. 1.3.2), which allows to build a map M as a collection of discretized 3D space elements.

From **Definition 1** of the *incomplete surface element*, we recall that C is the set of ISE contained in M. Thus, we extend the **Definition 2** of the *remaining incomplete surface* to the multi-agent case:

Definition 4 (**Remaining incomplete surface (multi-agent)**). *Let Q be the set of all collision-free configurations $\mathbf{q}^i$ of the agent $i \in \{1, 2, \ldots, N\}$, and let $Q_c(\mathbf{v}) \subseteq Q$ be the set of all configurations from which an ISE $\mathbf{v} \in C$ can be scanned. The* remaining incomplete surface *is then defined as*

$$C_{\text{rem}} = \bigcup_{\mathbf{v} \in C} \left\{ \mathbf{v} \mid Q_c(\mathbf{v}) = \emptyset \right\}.$$

We will use the function $\mathbf{p}^i_{j,k}(s) : [0, 1] \rightarrow \text{SE}(m)$, $m \in \{2, 3\}$, to define the path of agent i from configuration j to configuration k, where $\mathbf{p}^i_{j,k}(0) = \mathbf{q}^i_j$ and $\mathbf{p}^i_{j,k}(1) = \mathbf{q}^i_k$, $i \in \{1, 2, \ldots, N\}$. We assume that $\mathbf{p}^i_{j,k}(s)$ is collision-free and feasible for agent i (i.e. the kinematic/dynamic constraints of the vehicle are satisfied along the path). The problem studied in this chapter can then be stated as follows.

Problem 2 (**Multi-agent inspection problem**). Consider a team of N agents with initial configurations $\mathbf{q}^i_0 \in Q$, $i \in \{1, 2, \ldots, N\}$. The multi-agent inspection problem is to find for each agent i, the path sequence $\mathbf{P}^i_{0,f_i} = \{\mathbf{p}^i_{0,1}, \ldots, \mathbf{p}^i_{f_i-1,f_i}\}$ formed of collision-free paths $\mathbf{p}^i_{k-1,k}(s)$ visiting the poses $\mathbf{q}^i_k$, $k \in \{1, 2, \ldots, f_i\}$, so that the team scans the set $C_{\text{ins}} = C \backslash C_{\text{rem}}$ of all ISEs contained in the current reconstructed map M.

Progressing along their paths $\mathbf{P}^i_{0,f_i}$, $i \in \{1, 2, \ldots, N\}$, the agents are able to disclose the unknown space, discover new ISEs, and iteratively solve the inspection problem until $C_{\text{ins}} = \emptyset$. Based on these premises, in the next section, we will provide a general overview of the approach proposed to solve Problem 2.

3.3 System overview

The proposed multi-robot system for the reconstruction of the 3D surface of an unknown environment is designed to accommodate any type of robot (ground or aerial). A centralized architecture (see Fig. 3.1) is considered in this chapter, where computation is supported by a ground station or by the embedded computer of a robot, which is thus promoted to the rank of team "leader". The architecture keeps the same structure than single-robot and consists in three complementary modules: a *perception module* (green block), which extracts the ISEs (cf. Sect. 2.2, Definition 1) from a volumetric map reconstructed online, a *planning module* (blue block) that is in charge of computing the path

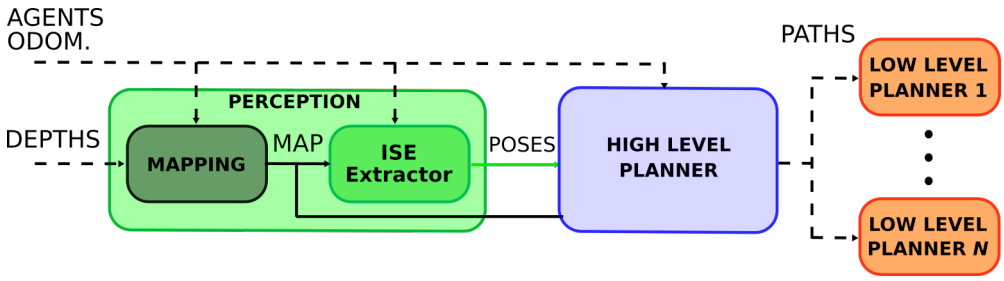


Figure 3.1: *General flowchart of the multi-robot surface-reconstruction architecture.* The internal structure of the perception and planning modules is shown inside the green-, and blue-shaded boxes, and the intra- and inter-block connections are represented with solid and dashed lines, respectively.

of the robots, and a *low-level planner* (orange block) which locally manages the visit of clusters and ensures navigation safety.

Similarly to the single-robot architecture, the *perception module* takes as input the depth maps sensed by each vehicle together with the sensor pose and integrate them into a 3D volumetric map. This time, the volumetric map is updated asynchronously when a new depth map has been sensed by a robot. The TSDF method (cf. Sect. 1.3.3) has been chosen since it allows to implicitly represent a surface in a volumetric map. Thus, it can be used for both surface completeness evaluation, i.e. ISE extraction, and for navigation, i.e. collision-free path planning and tracking. During the exploration, the map is incrementally built and the team of robots detects sets of incomplete surfaces or contours by extracting ISEs. The configurations which allow to complete them are generated in the free space, depending on the type of environment and camera properties (range, resolution), and sent to the *planning module*.

The *planning module*, or *high-level planner*, aggregates these configurations depending on their location in the 3D space. Each cluster is evaluated by estimating the information gain of each of its poses using a ray-casting method. A weighted directed graph is created and continuously expanded where the weights on the arc represent the travel utility between clusters in the free space. This utility, used in a single objective function, trades off travel cost and information gain of visiting a new cluster. The visit of a cluster is assigned greedily to one agent by maximizing the utility of the fleet, and by using the previous single-agent TSP formulation as an insertion method (i.e the TSP solving will place the cluster in the path such that it maximizes the utility). As the map evolves, new ISEs are discovered, and the path to complete them is updated. The 3D model is considered complete when there are no more ISEs.

The *low-level-planner* generates a low-level path between a start and an end pose in the free space using a probabilistic planner and sends it to the controller. It also monitors

the exploration, asks for replanning with a traffic policy and commands emergency stops in case of obstacle detection. Each agent has a dedicated low-level planner. The modules are described in more detail in the next two sections.

3.4 Perception

This section presents the centralized perception module used in our multi-robot system. This is a rather straightforward extension of the single-robot architecture. Please refer to Sect. 2.4 for further details.

3.4.1 Surface-based mapping

A volumetric map M, here based on a TSDF representation, is used to detect non-reconstructed areas as defined by the extraction of ISEs. The TSDF map [Curless & Levoy 1996] consists of a voxel grid, where each voxel contains a truncated signed distance value $\phi \in \mathbb{R}$ and a weight $w \in \mathbb{R}^+$. Algorithms such as MarchingCubes [Lorensen & Cline 1987] can be used to extract a triangular mesh which is an explicit representation of those surfaces, e.g. for visualization. This map is built in an incremental manner by sequentially integrating depth measurements provided by the agents into the centralized mapping module, see Fig. 3.2. In order to keep the map consistent, the pose $\mathbf{q}^i$ of the sensor must be used to express the depth measurements in the reference frame of the map. This pose is supposed to be provided by a localization system, e.g. a visual SLAM algorithm (see for example [Engel *et al.* 2015, Mur-Artal & Tardós 2017]). In order to take the sensor uncertainty into account [Nguyen *et al.* 2012, Oleynikova *et al.* 2020], the new measurements

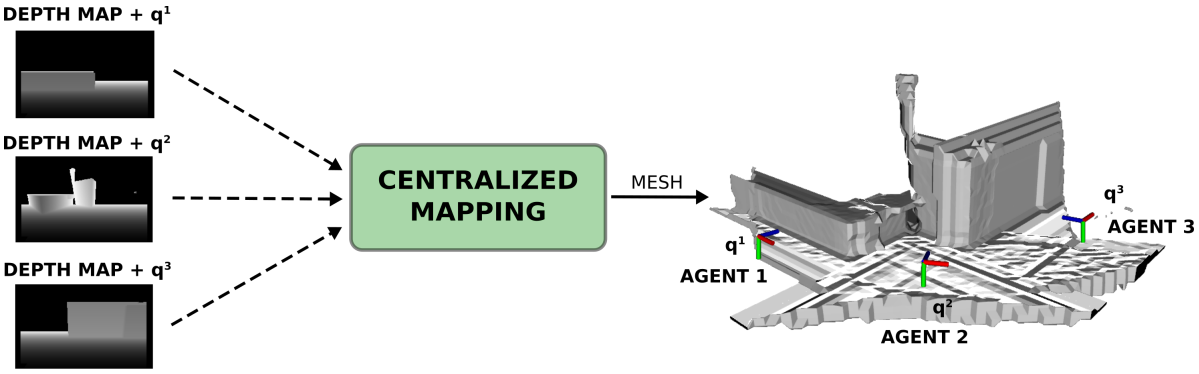


Figure 3.2: *Flowchart of centralized mapping module for a team of 3 robots:* each robot (3D frame on the right snapshot) sends its depth map and pose to the ground station which generates a TSDF volume and a mesh.

are weighted by an inverse-squared distance increment $1/z^2_{\mathbf{q}^i}(\mathbf{v})$, where $z_{\mathbf{q}^i}(\mathbf{v})$ is the distance between voxel $\mathbf{v}$ and the current pose $\mathbf{q}^i$. The state of a voxel $\mathbf{v}$ is set to *known* (either *occupied* or *empty*) if $w(\mathbf{v}) \geq W_{\text{th}}$ and to *unknown* if $w(\mathbf{v}) < W_{\text{th}}$, where the threshold W_{th} depends on the sensing range of the depth sensor.

3.4.2 ISE extractor, viewpoint generation and clustering

The ISE extraction, the viewpoints generation and clustering processes are identical to the single-robot architecture (cf. Sect. 2.4.2 and 2.4.3), as they only rely on the generated TSDF map, irrespective of its source.

3.5 Centralized Planning

In this section, the TSP-Greedy Allocation planner is detailed. We present our two-level planning algorithm for addressing the inspection problem with a multiple robots in a centralized way. Here, since the agents have different initial conditions, they perceived different travel graphs. Based on these graphs, the high-level TSGA planner is used to assign and schedule the visit of clusters. This greedy algorithm allocates the clusters by maximising the utility of the fleet, using successive solution of a variant of Traveling Salesman Problem (similar to Sect. 2.5.2) as a cluster's insertion method. The planned tour of the clusters is sent to the low-level planner which ensures safe navigation and monitors path update depending on map evolution.

3.5.1 Inspection problem

The inspection problem for an agent can be stated as a *maximum Open Asymmetric Traveling Salesman Problem* (**maxATSP**). It consists in finding a maximum-utility Hamiltonian path on $\mathcal{G}$ (cf. the graph formulation in Sect. 2.5.1), i.e. a path which visits every configuration exactly once, maximizing the sum of the weights on the arcs f_{UV}, $U \neq V$, starting from the initial configuration $\mathbf{q}_0$ and ending to $\mathbf{q}_f$, with $\mathbf{q}_f \neq \mathbf{q}_0$. Following Sect. 2.5.2, let us denote by **maxATSP**($\mathcal{U}$), the set function that takes the set of clusters $\mathcal{U}$ as input and outputs its utility value $p \in \mathbb{R}^+$, from which the path sequence $\mathbf{P}_{0,f}$, e.g. the viewpoint sequence to be visited, can be computed. By solving **maxATSP**($\mathcal{U}$), we can determine the optimal path for an agent to visit the clusters in $\mathcal{U}$.

To address the multi-agent inspection problem, we need to split the task among the robots. Let $\mathcal{U}^i$ be the set of clusters assigned to agent $i \in \{1, \ldots, N\}$, such that $\bigcup_{i=1}^{N} \mathcal{U}^i =$

$\mathcal{U}$. Then, the assignment problem can be stated as follows,

$$\max_{\mathcal{U}^1,\dots,\mathcal{U}^N \subset \mathcal{U}} \Big\{ \sum_{i=1}^{N} \text{maxATSP}(\mathcal{U}^i) \;\Big|\; \mathcal{U}^i \cap \mathcal{U}^\ell = \emptyset,\, i \neq \ell,\, \bigcup_{i=1}^{N} \mathcal{U}^i = \mathcal{U} \Big\}, \tag{3.1}$$

where $\sum_{i=1}^{N} \text{maxATSP}(\mathcal{U}^i)$ is a non-decreasing set function, and the space of feasible paths has the structure of a simple partition matroid [Wilson 1973]. The maximisation problem (3.1) can be approximately solved via local greedy heuristics (cf. examples [Bian *et al.* 2017, Nemhauser *et al.* 1978, Fisher *et al.* 1978]), which will seek for the locally maximum utility depending to an initial ranking of the element to assign. The Centralized TSP-Greedy Allocation (TSGA) procedure [Hardouin *et al.* 2020a] reported in **Algorithm 2**. Note that the single-agent algorithm is a special case of the centralized multi-agent algorithm, with $N = 1$.

At each ISE extraction, the clusters are formed and their shortest Euclidian distance to an agent of the fleet is computed, in order to rank them by distance in ascending order for the greedy heuristic. The TSGA planner greedily assigns each cluster to an agent. A cluster is assigned when it locally maximizes the overall utility of the team. The path sequence of agent i which results from the allocated clusters $\mathcal{U}^i$, is denoted by $\mathbf{P}^i_{\mathcal{U}^i}$, and its utility value by p^i. Once computed, the paths $\mathbf{P}^1_{\mathcal{U}^1}, \dots, \mathbf{P}^N_{\mathcal{U}^N}$ are broadcast to all the agents (cf. **Algorithm 2**). Unlike classical greedy insertion methods where a cluster is added to a robot's path [Jawaid & Smith 2015], the maxATSP problem is solved for the extended cluster set $\mathcal{U}^i \cup V$ with $V \in \mathcal{U} \setminus \mathcal{U}^i$. This strategy maximizes the individual utility of the agents over disjoint sets so as to maximize, in turn, team-wise utility. The maximization of the utility function over the long run, pushes the agents towards the most valuable areas in terms of completeness, even if it is not the shortest path. This might prompt an agent to move to configurations that cover an area at the frontier between the known and unknown surface, which contains multiple ISEs and scan them all. However, some ISEs can be completed before the planned visit of an agent. To prevent long back-and-forth travel moves, the low-level planner computes the remaining ISEs of paths since the last planning iteration, and it possibly performs an update (see Sect. 3.5.2). This kind of update rule ensures a reactive visit of uncovered ISEs, as the map grows over time. The scanning procedure stops when no more ISEs remain.

The TSGA planner may generate paths of various length: hence, each agent may finish its tour at different time instants. To reduce the idle time of the agents which have finished first, new paths are asynchronously allocated to them in order to visit clusters which are currently not assigned to any robot (e.g. if agent i has not terminated its exploration round, the set of clusters assigned to the rest of the team becomes $\mathcal{U} \setminus \mathcal{U}^i$).

Algorithm 2: TSP-Greedy Allocation (TSGA)

Set $\mathcal{U}^i = \emptyset$ and $p^i = 0$, $\forall\, i \in \{1, 2, \ldots, N\}$;
foreach *cluster* $V \in \mathcal{U}$ **do**
  $i \leftarrow \underset{k \in \{1,2,\ldots,N\}}{\operatorname{argmax}} \{\text{maxATSP}(\mathcal{U}^k \cup V) - p^k\}$;
  $\mathcal{U}^i \leftarrow \mathcal{U}^i \cup V$;
  $p^i \leftarrow \text{maxATSP}(\mathcal{U}^i)$;
  $\mathbf{P}^i_{\mathcal{U}^i} \leftarrow \{p^i, \mathcal{U}^i\}$;
Send the paths $\mathbf{P}^1_{\mathcal{U}^1}, \ldots, \mathbf{P}^N_{\mathcal{U}^N}$ to the agents (low-level planner);

Note that the algorithm can be easily modified to take advantage of the different capabilities of the robots (different types, speed, cameras). For example, if a team of UAVs and wheeled robots is considered, the set of clusters can be split into two, with the ones that can be observed from the ground assigned to the wheeled robots, and the others to the UAVs. On the other hand, for heterogeneous aerial (or ground) vehicles, the penalty terms should be adjusted according to the energy consumption, the flight (ride) autonomy or the speed of each robot.

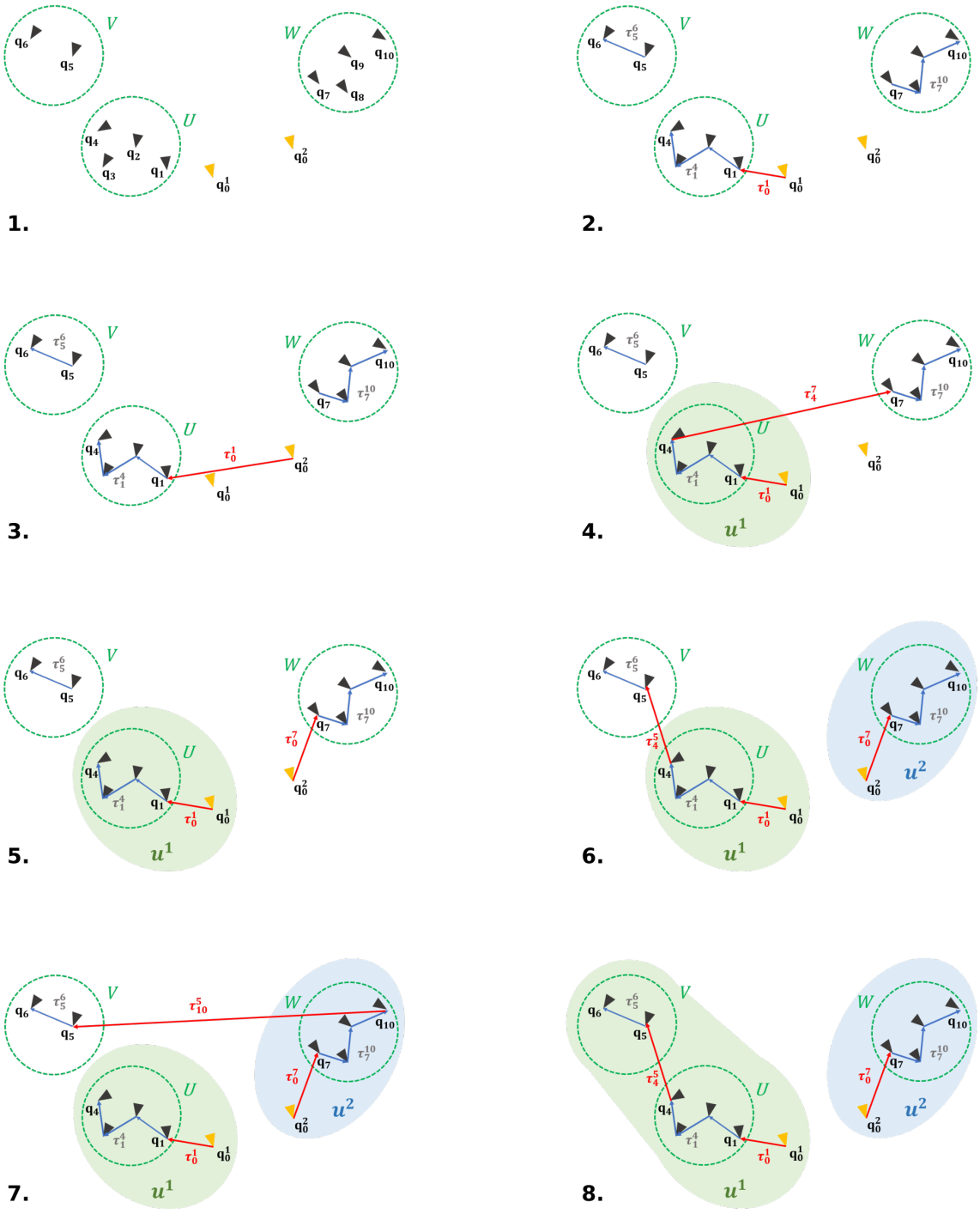


Figure 3.3: *Greedy allocation planner.* **1.** The clusters U, V and W are to be allocated to robot 1 and 2. They are ordered according to their distance to a robot. **2.** Cluster U is the first to be allocated. It is first included into the path of robot 1 by solving **maxATSP**($\mathcal{U}^1 \cup U$) and its utility gain is computed. **3.** Cluster U is inserted into the path of robot 2 by solving **maxATSP**($\mathcal{U}^2 \cup U$) and its utility gain is computed. **4.** Robot 1 gains the most from the TSP insertion: U is assigned to robot 1. The iterative process continues: it is performed for cluster W which is first included into the path of robot 1. **5.** W is inserted into the path of robot 2. **6.** Cluster W is assigned to robot 2. The allocation process continues with cluster V which is added to the path of robot 1. **7.** V is inserted into the path of robot 2. **8.** Eventually, V is allocated to robot 1, bringing the assignment process to an end.

3.5.2 Low-level planner

The low-level planner receives the list of clusters to visit $\mathbf{P}^i$, and the coordinates of their respective configurations, computed by the TSGA planner for a given update of the TSDF map M. **Algorithm 3** describes the procedure. Compared to the single-robot case, it has been modified to manage the traffic of the team (see lines 15-19 in Algorithm 3). Its main purpose is to find the local path in the free-space using the LazyPRM* algorithm (cf. Sect. 1.4.3) and to send it to the robot's controller (see **LazyPRM***). It consists of the following subfunctions:

1. Given a start and an end pose, $\mathbf{q}_{curr}$ and $\mathbf{q}$ respectively, the robot *path* $\mathbf{p}^i_{curr,q}$ (i.e. the sampled set of consecutive robot configurations) in the free space given by the TSDF volume. According to the type of robot, the paths are computed in the 3D space (aerial vehicles), or in the 2D space (ground vehicles). If the goal configuration can be reached by the robot, the path $\mathbf{p}^i_{curr,q}$ is sent to the controller (*feasible* is set to *True* by **LazyPRM***).

2. The function ensuring that the goal has been reached, **GoalReached**, is identical to the one in the single-robot case, see Sect. 2.5.3, Item 2.

3. The utility check function, **UtilityCheck**, is identical to the single-robot case, see Sect. 2.5.3, Item 3.

4. The obstacle check function, **ObstacleCheck**, is identical to the single-robot case, see Sect. 2.5.3, Item 4.

5. By gathering all the odometric measurements, each agent knows the position and orientation of the others and is then able to detect when another robot is near, when it faces it, or when it will cross its path. A traffic policy takes care of collision avoidance: e.g. a robot is asked to step aside (in the free space) to avoid a frontal collision, or to temporarily stop and wait until it exits from its field of view, when a teammate crosses its path (maneuver is set to *stop* or *step aside* by **TrafficPolicy**), waiting for low-level replanning. The traffic policy takes the robot's speed into account and relies on a safety distance threshold for collision avoidance, similarly to **ObstacleCheck**.

Algorithm 3: Centralized multi-robot low-level planner for agent i

Inputs: $\mathbf{q}^i_{curr}$, $\mathbf{P}^i$, Q, M, C;
Set $reached = False$; $obstacle = False$; $update = False$; $maneuver = None$;
$C_{start} \leftarrow C$;
foreach $\mathbf{q} \in Q$, following $\mathbf{P}^i$, **do**
    $update \leftarrow$ **UtilityCheck**(C_{start}, C);
    if *update* **then**
        wait for high-level re-planning;
        break;
    $\{feasible, \mathbf{p}^i_{curr,q}\} \leftarrow$ **LazyPRM**$^\star(\mathbf{q}^i_{curr}, \mathbf{q}, M)$;
    if *feasible* **then**
        send $\mathbf{p}^i_{curr,q}$ to Controller;
        while (not *reached*) **do**
            $reached \leftarrow$ **GoalReached**$(\mathbf{q}^i_{curr}, \mathbf{q})$;
            $obstacle \leftarrow$ **ObstacleCheck**$(\mathbf{q}^i_{curr}, M)$;
            $maneuver \leftarrow$ **TrafficPolicy**$(\mathbf{q}^i_{curr}, \mathbf{q}^k_{curr}, M),\ \forall k \in \{1, 2, \ldots, N\},\ i \neq k$;
            if $maneuver \neq None$ **then**
                send *maneuver* to Controller;
                wait for *maneuver*;
                $maneuver = None$;
            if *obstacle* **then**
                send *stop* to Controller;
                wait for high-level re-planning;
                $reached = True$;

3.6 Numerical experiments

Similarly to the single-robot case, the proposed multi-robot system has been validated via realistic numerical experiments. For these simulations, we considered multirotor UAVs with 4 degrees of freedom: their 3D position $[x, y, z]^T$ and their yaw angle ψ. We considered a fleet of 3 and 5 UAVs. The large-scale environments described in Chapter 2, have been re-used: the industrial benchmark **Powerplant** model and the monumental model of the Statue of Liberty (**SoL**). We compared the TSGA planner with another multi-robot planner, the Nearest Neighbor (NNB) greedy algorithm. In NNB, only one cluster is allocated to each robot by locally computing $\max_{v \in U} f_{uv}$ for the updated map. Replanning is thus very fast compared to TSGA and reactive to map update, but only one cluster at a time is set to be visited. In Chapter 2 (cf. Sect. 2.6.4), the numerical experiments showed that the single-UAV system is competitive with respect to the state-of-the-art methods. In this chapter, we will compare the 3D reconstruction performance of the single-UAV

system with that of the centralized multi-robot architecture.

3.6.1 Multi-robot nearest neighbour planner

The idea of the NNB planner is to simply allocate "the most useful" cluster to a robot, every time the ISE map is updated. This is formalized in **Algorithm 4**, where the same notation of Sect. 3.5 has been used. Since only one motion is planned every update, this fast local approach is very efficient for local reconstruction. However, no paths over long horizons can be planned, and this might lead the robots to turn around the same location, especially if the surface contains fine details and occluded areas. Allocation conflicts are handled by the low-level planner, in a similar fashion to TSGA (cf. Sect. 3.5.2).

Algorithm 4: multi-robot Nearest Neighbour planner (NNB)

Inputs: q_0^i, $\mathcal{U}$, $i \in \{1, 2, \ldots, N\}$;
foreach $i \in \{1, 2, \ldots, N\}$ **do**
    $V \leftarrow \underset{V \in \mathcal{U}}{\operatorname{argmax}} \; f_{\{q_0^i\} V}$;
    $\mathbf{P}^i \leftarrow V$;
Send the paths to the agents (low-level planner);

3.6.2 Simulation setup

We used the RotorS simulator [Furrer *et al.* 2016] to model multirotor UAVs equipped with a stereo camera, in the ROS-Gazebo[1] environment (see Sect. A.2.4, Appendix A). As in Sect. 2.6.1, we followed [Nguyen *et al.* 2012, Keselman *et al.* 2017] and we represented the depth-map uncertainty using a Gaussian-noise model. To generate the TSDF volume, we relied on the GPU-algorithm proposed in [Zeng *et al.* 2017] which has been modified to gather multiple depth maps, and the weight increment has been modified to be quadratic, as reported in Sect. 3.4.1. The mesh is reconstructed with MarchingCubes [Lorensen & Cline 1987]. Collision-free UAV paths have been computed using the LazyPRM* planner from the Open Motion Planning Library [Şucan *et al.* 2012] (the collision radius was set to 1 m). The UAVs track the generated paths using MPC [Kamel *et al.* 2017]: the reference translational velocity is fixed to 0.6 m/s.

The code used to model the aerial vehicles, includes open-source implementations of the algorithms mentioned above, which are integrated into our ROS software architecture as depicted in Fig. 3.4. The simulation stack interacts with Gazebo which simulates the 3D model of the environment to reconstruct, and the robot behaviour via RotorS. The

[1] `https://ros.org/`, `http://gazebosim.org/`

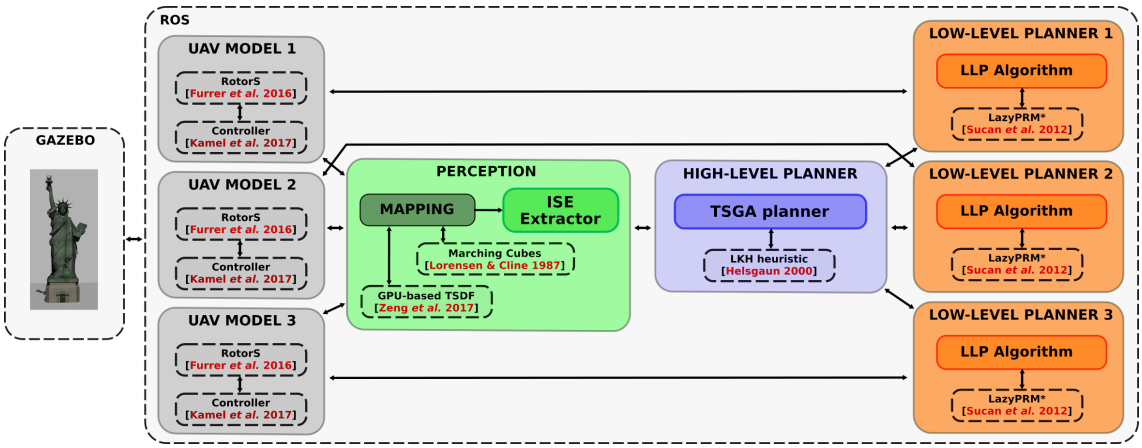


Figure 3.4: *Centralized simulation stack for a fleet of 3 UAVs*: Gazebo interacts with ROS. Some existing software modules (dashed bubbles) have been adapted and integrated into our software nodes: **Mapping**, **ISE Extractor**, **TSGA planner** and **Low-level planners** (colored bubbles).

ROS nodes are implemented in Python (v2.7 and v3) and C++, following the architecture presented in the system overview (Sect. 3.3) with three main functional modules: *Perception module*, *High-level planner* and *Low-level planner*. This modular architecture allows the nodes to be easily modified if needed. All nodes are designed to run on CPU, except for the centralized TSDF mapping which is GPU-based for fast depth map integration. We ran our algorithm on a Dell Precision 7520 with 2.90 GHz Intel Core i7 processor, 16 GB RAM and Quadro M2200 graphics card. The virtual environments, **Powerplant** and **SoL**[2] are also considered in the experiments: these environments are described in Chapter 2 (see Sect. 2.6.1, Fig. 2.5). In both scenarios, the UAVs are initially located in the same area, around a ground station (magenta dots in Fig. 3.5). The simulation parameters used in the two scenarios are reported in Table 3.1 and described in Sect. 2.6.1.

3.6.3 Metrics

The single-UAV planner proposed in Chapter 2 is used as a baseline to evaluate our new multi-robot strategy, in terms of cumulated path length and completion time to cover the entire 3D environments. This includes travel time and sensing time (e.g. one depth map integration and map update). The reconstructed 3D surface has been evaluated with CloudCompare[3] using the M3C2 (Multiscale Model to Model Cloud Comparison) algorithm [Lague *et al.* 2013]. See Sect. 2.6.2 for more details.

[2] **Powerplant**: `http://models.gazebosim.org/`, **SoL**: `https://free3D.com/`
[3] `https://danielgm.net/cc/`

Table 3.1: Parameters used in the TSGA numerical experiments.

Parameter	**Powerplant**	**SoL**
Voxel resolution $r_{\mathbf{v}}$ [m]	0.3	0.15
Threshold W_{th}	0.3	0.3
$e_{\max}$ [m]	0.2598	0.1299
Camera range [m]	[1.6, 8]	[1, 5]
Camera FOV [deg.] (H, V)	90×60	90×60
e_d [pixels]	0.1	0.1
f [pixels]	376	376
B [m]	0.11	0.11
Collision radius [m]	1	1
UAV nominal speed [m/s]	0.6	0.6
δ_{pose} [m]	4.7	3.6
d_v [m]	2.0	2.5
$d_v^{\max}$ [m]	5	5
Penalty term λ_{tc}	0.3	0.17
Penalty term λ_{ic}	0.03	0.15
c_{thres} [%]	80	50

3.6.4 Results and discussion

The results of our numerical experiments are reported in Tables 3.2 and 3.3, for **Powerplant** and **SoL**, respectively. As in Sect. 2.6.4, we will refer to the results with the single-UAV planner and perfect measurements, as *SR*, and to those with noisy depth measurements, as SR^*. We compared *SR* and SR^* with both the TSGA and NNB planners for 3 and 5 UAVs. To obtain statistically-significant values, 10 trials per scenario were carried out.

Multi-UAV vs. single-UAV

As expected, the use of multiple UAVs has a beneficial effect on the completion time. In particular, the TSGA planner significantly reduces it. In **Powerplant** (8*th* row of Tab. 3.2), the fleet of three (five) UAVs performs the 3D reconstruction 68.8% (80.8%) faster than a single UAV. With **SoL** (8*th* row of Tab. 3.3), we had a similar outcome, the gain on the completion time being of 72.2% (81.8%). With a completion time of about 11 minutes in both scenarios for a fleet of 3 UAVs, the proposed multi-robot system is thus amenable to a hardware implementation on real multirotors, whose flight autonomy ranges between 15 and 20 minutes. On the other hand, the cumulated path length (6*th* row) of the fleet is bigger than that of a single UAV in both scenarios, but the individual paths of robots are shorter (1*st* to 5*th* row). The number of multirotors has little impact on the reconstruction

quality (10*th* to 12*th* row) and completeness (9*th* row). A video illustrating this analysis is available at: https://youtu.be/ce3XI2CSeh4

Table 3.2: Results of the TSGA numerical experiments on **Powerplant** (stats. over 10 trials).

Number of UAVs		1		3		5	
#	Algorithm	*SR*	*SR**	NNB	TSGA	NNB	TSGA
1	Path length rob. 1 [m]	–	–	340.2	273.0	219.2	172.2
2	Path length rob. 2 [m]	–	–	353.1	255.6	225.3	178.0
3	Path length rob. 3 [m]	–	–	344.7	261.4	226.5	175.2
4	Path length rob. 4 [m]	–	–	–	–	221.7	173.8
5	Path length rob. 5 [m]	–	–	–	–	220.3	179.8
6	Total path length [m]	780	785	1038	790	1113	879
7	Completion time [min.]	32′	33′10″	11′09″	10′20″	6′51″	6′22″
8	Time gain [%] w.r.t. *SR**	–	–	66.4	68.8	79.4	80.8
9	Surface coverage [%]	91.5	90.4	90.0	91.0	90.5	90.6
10	M3C2 avg. error [cm]	0.14	−0.13	−0.15	−0.26	−0.11	−0.3
11	M3C2 std. error [cm]	5.85	7.51	7.52	7.54	7.50	7.52
12	RMSE [cm]	5.86	7.51	7.52	7.55	7.50	7.52

Table 3.3: Results of the TSGA numerical experiments on **SoL** (stats. over 10 trials).

Number of UAVs		1		3		5	
#	Algorithm	*SR*	*SR**	NNB	TSGA	NNB	TSGA
1	Path length rob. 1 [m]	–	–	245.4	237.8	114.7	115.3
2	Path length rob. 2 [m]	–	–	239.1	240.0	116.5	114.1
3	Path length rob. 3 [m]	–	–	248.5	243.2	118.4	117.5
4	Path length rob. 4 [m]	–	–	–	–	114.9	113.3
5	Path length rob. 5 [m]	–	–	–	–	115.5	114.0
6	Total path length [m]	547.0	550.0	733.0	721.0	580.0	574.2
7	Completion time [min.]	36′	37′	13′10″	10′18″	7′30″	6′45″
8	Time gain [%] w.r.t. *SR**	–	–	64.4	72.2	79.7	81.8
9	Surface coverage [%]	92.3	91.2	91.1	91.0	90.9	91.1
10	M3C2 avg. error [cm]	0.29	−0.02	−0.80	−0.03	0.06	−0.01
11	M3C2 std. error [cm]	3.41	3.67	3.61	3.69	3.65	3.66
12	RMSE [cm]	3.43	3.67	3.69	3.69	3.65	3.66

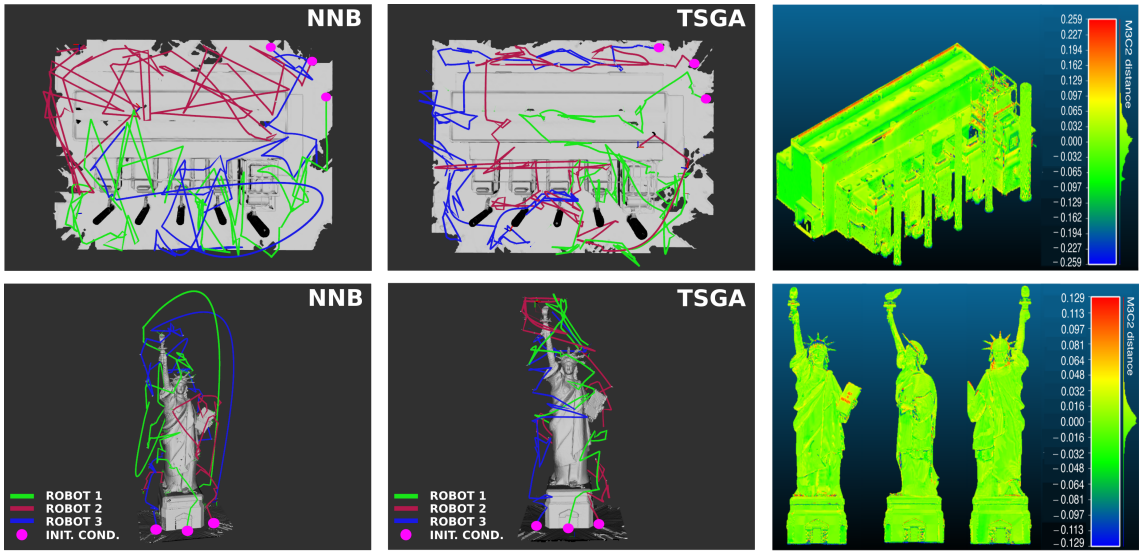


Figure 3.5: *Numerical experiments:* [top] **Powerplant** and [bottom] **SoL**: Reconstructed mesh and 3D exploration paths $\mathbf{p}^1_{0,f}$, $\mathbf{p}^2_{0,f}$, $\mathbf{p}^3_{0,f}$ (green, red, blue) of 3 UAVs (the initial locations are marked with magenta dots); [left] NNB planner; [middle] TSGA planner; [right] Signed distance error: the color coding shows the error in meters with respect to the ground truth, computed with CloudCompare's M3C2 algorithm.

TSGA vs. NNB planner

Overall, the TSGA planner is more effective than the NNB planner for multi-robot navigation (see the comparative results reported in Table 3.4). TSGA outperforms NNB in terms of cumulated path length in wide and large environments (**Powerplant**), but the results are comparable in medium-size structures (**SoL**). Indeed, the surface properties play an important role on the reconstruction, and a planner promoting fast local updates tends to be more successful in small environments containing close occluded areas, sharp edges and fine details, where many ISEs can be revealed after a scan. On the other hand, over a long horizon, a planner is more effective in finding the shortest path in a large planar environment where each viewpoint covers less ISEs. By comparing the completion times for **SoL**, we can notice that TSGA is much faster than NNB, but that the UAVs need to travel longer distances before completing the same number of ISEs. Nevertheless, NNB performs many more scans which are close to each other, and long paths are computed to complete the reconstruction (see Fig. 3.5-[bottom left]).

Our method guarantees that all the regions that are accessible to the UAVs are covered. Moreover, in keeping with the recent qualitative analysis for single-robot exploration of [Yoder & Scherer 2016, Song & Jo 2018] methods in [Schmid *et al.* 2020], it turns out to be competitive with the state-of-the-art approaches in terms of overall 3D quality: with reconstruction error between 5 and 10 cm. Indeed, the quality of 3D reconstruction is

Table 3.4: Comparison between the TSGA and NNB planner.

	Powerplant		**SoL**	
Number of UAVs	3	5	3	5
Path length gain over NNB [%]	23.9	21	1.64	1
Completion-time gain over NNB [%]	7.32	7.06	21.8	10

resolution-dependent: in fact, it is inversely proportional to the size of the TSDF voxels. A small resolution amounts to a large number of voxels to be integrated in the TSDF map, which is a resource-intensive process. Therefore, if multiple UAVs explore a large environment using on-board sensing and processing, a trade-off between reconstruction accuracy and computational efficiency should be found.

3.7 Conclusion

In this chapter, we have presented a centralized multi-robot architecture for the reconstruction of large-scale environments, as a direct extension of our previous single-robot system. A single 3D representation of the environment has been kept for both surface reconstruction and navigation. This time, all depth maps are centralized into a shared mapping module to generate the volumetric map of the environment. The former two-level path planning strategy with surface-based 3D reconstruction gain and cost-utility formulation has been extended to separate the scanning tasks. The proposed TSGA planner addresses the viewpoint allocation problem for the agents of the team, by using a greedy heuristic and a TSP insertion policy. Realistic numerical experiments with ROS and Gazebo have validated the proposed method in the same two challenging environments as in the previous chapter. As expected, the proposed method drastically reduces the scanning time and path traveled per robot compared to the single-robot case. The performances are in line with the flight autonomy of existing battery-powered multi-rotor UAVs. In addition, the TSGA planner outperforms a classical greedy planner (the Nearest Neighbour planner (NNB)), which is fast, local and thus very reactive to map updates. However, the centralized architecture studied in this chapter entails some drawbacks, since all calculations have to be performed on a single computer, a ground station, or a leader agent, which ultimately means an individual point of failure, in case of communication loss or crashes. In addition, the proposed centralized mapping module suffers from scaling issues. Indeed, the depth map integration of the whole robot team in real time can be achieve using GPU-computing for a small amount of robots but may not be performed with larger team. To ensure more robustness and to improve portability on

an embedded computer, in the next chapter, we will present a *distributed* version of our centralized multi-robot architecture.

CHAPTER 4

Surface reconstruction: distributed multi-robot architecture

Chapter outline

4.1 Introduction . . . 73
4.2 Problem formulation . . . 74
4.3 System overview . . . 75
4.4 Perception . . . 78
4.4.1 Surface-based mapping . . . 78
4.4.2 ISE extractor, viewpoint generation and clustering . . . 80
4.5 Distributed Planning . . . 80
4.5.1 Inspection problem . . . 80
4.5.2 Low-level planner . . . 83
4.6 Numerical experiments . . . 86
4.6.1 Simulation setup . . . 86
4.6.2 Metrics . . . 87
4.6.3 Results and discussion . . . 88
4.7 Conclusion . . . 91

4.1 Introduction

In this chapter, we present a the *distributed* 3D reconstruction algorithm for multiple cooperative robots, that addresses the *surface inspection problem* in an unknown environment, via a Next-Best-View (NBV) frontier-based planner. The main goal is to design an algorithm which is more robust and resilient than the centralized one discussed in Chapter 3 by increasing the number points of failure of the system [Avizienis *et al.* 2004]. To this end, we present a distributed manifold mapping module which allows the robots to

obtain a local map of the environment and to share it with their teammates. The ISE extraction process remains the same while the greedy allocation planning becomes distributed. The low-level planner is extended to manage possible redundant assignments of viewpoints. As in the previous chapters, numerical simulations show that the proposed architecture provides accurate, complete and time-efficient 3D reconstructions. To the best of our knowledge (cf. Chapter 2), we are not aware of any other distributed multi-robot algorithm which relies on surface information for planning. In summary, the original contribution of this chapter is threefold:

- A distributed manifold mapping is used to model the surface and free space of environment, useful for reconstruction, planning and navigation.
- A distributed planner is developed to visit viewpoint configurations for surface reconstruction. In particular, the robots perform a greedy allocation of configurations based on their available local information, perceived by the robot or shared by the team, to ensure completeness of reconstructed surface mesh.
- Following the same framework of Chapter 3, the proposed distributed multi-robot system has been validated in two realistic simulation environments, in the presence of depth-map uncertainties.

The distributed architecture has been also validated with real-world experiments and the results will be discussed in Chapter 5.

4.2 Problem formulation

As in Chapter 3, we consider a team of N cooperative mobile agents. We will use $\mathbf{q}^i \in \mathrm{SE}(m)$ to denote the pose of agent i, and $\mathbf{q}^i_0$ its initial configuration, $i \in \{1, 2, \ldots, N\}$: $m = 2$ in the case of ground vehicles, and $m = 3$ in the case of aerial vehicles. We assume that all agents are equipped with an accurate localization system which allows to estimate their pose with respect to a global reference frame, and that a robust low-level trajectory-tracking algorithm is available. Each agent is equipped with a forward-facing depth sensor with limited field-of-view (FOV) and sensing range, extrinsically calibrated with respect to its body frame. The agents should cooperatively scan an unknown 3D environment (for instance, a building), characterized by its surface using distributed information and processing.

We will use the function $\mathbf{p}^i_{j,k}(s) : [0, 1] \rightarrow \mathrm{SE}(m)$, $m \in \{2, 3\}$, to define the path of agent i from configuration j to configuration k, where $\mathbf{p}^i_{j,k}(0) = \mathbf{q}^i_j$ and $\mathbf{p}^i_{j,k}(1) = \mathbf{q}^i_k$, $i \in \{1, 2, \ldots, N\}$. We assume that $\mathbf{p}^i_{j,k}(s)$ is collision-free and feasible for agent i (i.e. the

kinematic/dynamic constraints of the vehicle are satisfied along the path).

Referring to the previous definitions of *incomplete surface element* (**Definition 1**) and *remaining incomplete surface* (**Definition 4**), let $\widehat{C}^i$ be the set of ISEs and $\widehat{C}^i_{\text{rem}}$ the remaining incomplete surface extracted from an estimation $\widehat{M}_i$ of the global map M modeled by an agent i, such that $\bigcup_{i=1}^{N} \widehat{M}_i \simeq M$. Let $\widehat{Q}_i \subset Q$ the set of configurations that agent i should visit. We can then formulate the distributed multi-agent inspection problem as:

Problem 3 (**Distributed multi-agent inspection problem**). *Consider a team of N agents with initial configurations $\mathbf{q}^i_0 \in Q$, $i \in \{1, 2, \ldots, N\}$. The distributed multi-agent inspection problem is to find, for every agent i, the path sequence $\mathbf{P}^i_{0,f_i} = \{\mathbf{p}^i_{0,1}, \ldots, \mathbf{p}^i_{f_i-1,f_i}\}$ formed of collision-free paths $\mathbf{p}^i_{k-1,k}(s)$ visiting the poses $\mathbf{q}^i_k \in \widehat{Q}_i$, $k \in \{1, 2, \ldots, f_i\}$, based on the sensed and shared information locally available. Such a path sequence $\mathbf{P}^i_{0,f_i}$ should allow agent i to scan the set $\widehat{C}^i_{\text{ins}} = \widehat{C}^i \setminus \widehat{C}^i_{\text{rem}}$ of all ISEs contained in its local estimate $\widehat{M}_i$ of the global map M.*

Progressing along their paths $\mathbf{P}^i_{0,f_i}$, $i \in \{1, 2, \ldots, N\}$, the agents are able to disclose the unknown space, discover new ISEs, and iteratively solve the inspection problem until $\widehat{C}^i_{\text{ins}} = \emptyset$. Based on these premises, in the next section, we will provide a general overview of the approach proposed to solve Problem 3.

4.3 System overview

The distributed multi-robot architecture reported in Fig. 4.1 is considered in this chapter. The computational load is now shared among the robots (on their on-board computers), but similarly to the centralized system, the architecture still consists of three complementary modules: a *perception module* (green block), which extracts the ISEs (cf. Sect. 2.2, Definition 1) from a volumetric map reconstructed online, a *planning module* (blue block) which is in charge of computing the path of the robots, and a *low-level planner* (orange block) which locally manages the visit of clusters and ensures navigation safety.

In the *perception module*, each agent computes its own local map based on its own sensing and localization data, and map sharing is performed via a distributed process based on manifold mapping [Howard 2004]. The agents integrate their data in a local map called a *patch*, identified by a unique ID, until a triggering event occurs. This event could either be the integration of a certain number of frames or a total travelled distance during the patch integration. A completed patch is stored into a *private map* (owned by the producer) and shared with the neighbouring agents, and a new local patch is created to continue the mapping task. Each agent maintains a *global map* by collecting and fusing

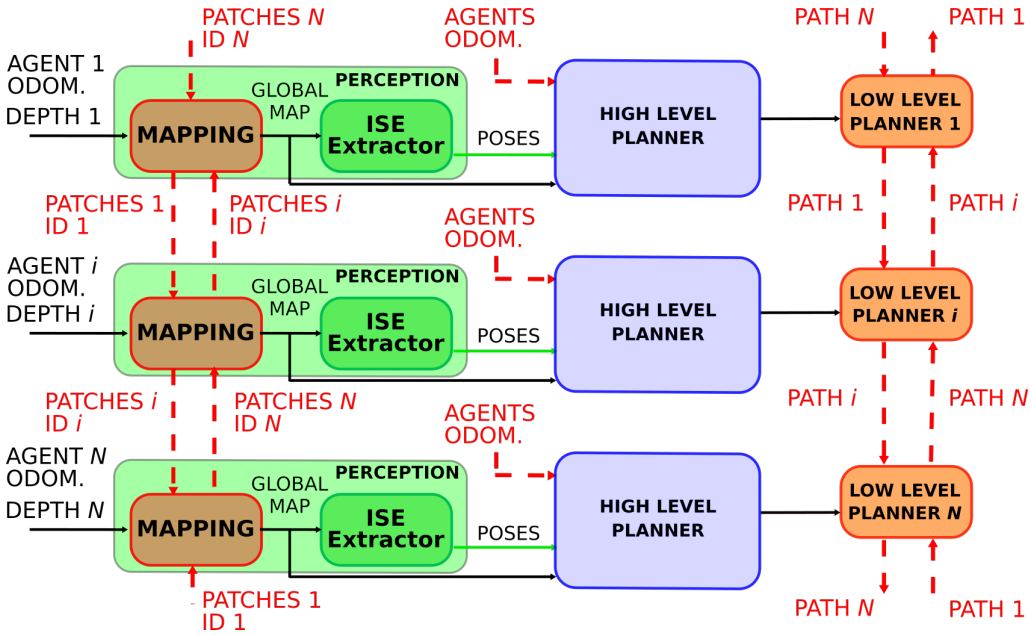


Figure 4.1: *General flowchart of the multi-robot surface-reconstruction architecture.* The internal structure of the perception and planning modules is shown inside the green-, and blue-shaded boxes, and the intra- and inter-block connections are represented with solid black and dashed red lines, respectively.

all received patches: this mapping process allows each agent to extract consistent sets of ISEs from this incremental manifold map.

The *planning module* guarantees that the agents correctly achieve the surface reconstruction. Given the candidate viewpoints provided by the perception module, each agent plans, the visits of each configuration with an appropriate TSP-based path finder. Here, the *dist-TSGA planner* ("TSP-Greedy Allocation") clusters sets of configurations according to their location in the 3D space to identify and rank areas of interest in the incomplete map. It then generates a directed graph which represents the travel utility of visiting a cluster, depending on the type of robot (terrestrial or aerial). In order to maximize the cumulative utility function for the team of robots, each robot computes collision-free paths for the fleet from the digraph and sends its own path to the low-level planner. As the TSDF map is updated, the ISEs are completed, others are uncovered, and the paths are computed. The scanning process stops when no ISEs are left.

The list of viewpoints of each robot is sent to the *low-level planners*, which generate a sampling-based path between two viewpoints for the agents, and gather the odometric and path-allocation information, for update and collision avoidance. In the *distributed architecture*, global-map inconsistencies, due e.g. to patch losses or communication delays, may result in clusters assigned to multiple robots. To overcome this issue, the low-level planner sends the path sequence currently explored to the others, collects the path

sequences of the team in order to check the consistency between the individual and team-wise allocation, and waits for a re-assignment, if needed. Finally, trajectory tracking could be performed with standard controllers (e.g. MPC).

The extension from a centralized to a distributed architecture, makes the multi-robot system more robust: in fact, as each robot is able to independently compute its path based on the local and shared available information. However, the whole system heavily relies on the communication network. The interaction between the robots can be modeled as a communication graph, where the robots are the nodes and communication links are the edges, see Fig. 4.2. In the nominal configuration, the communication graph is complete, and each robot can communicate with the others. If a communication link is lost, the graph is still connected and the robot which maintains the connection (robot 1 in Fig. 4.2) relays the information between the robots. Here the system still behaves normally. When the communication graph is no more connected a performance degradation (in terms of planning) is expected. In that case, the non-trivial connected components of the graph, will behave as multi-robot systems. Finally, when the graph is fully disconnected (e.g. it has n connected components), each robot behaves as a single-robot system. If the communication links are recovered, the distributed mapping algorithm allows each robot to send the private patches to the others, and to return to nominal functioning.

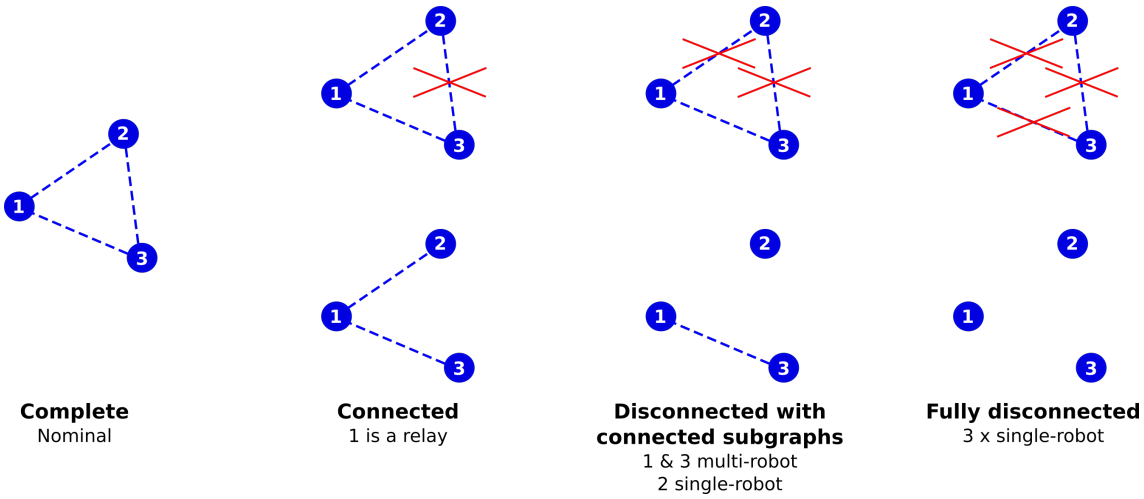


Figure 4.2: *Examples of communications graphs for the distributed multi-robot system.* [left] Complete, [middle-left] connected, [middle-right] disconnected and [right] fully disconnected graphs, and respective team behavior.

4.4 Perception

This section describes the perception module considered in our distributed multi-robot system. The distributed surface-based mapping used for both cartography, navigation and the definition of ISEs related to this representation are presented. The mechanisms used for viewpoint and cluster generation, are recalled.

4.4.1 Surface-based mapping

A volumetric map M, based on the TSDF representation, is used to detect non-reconstructed areas as defined by the extraction of ISEs (see Sect. 3.4.2). The TSDF method is presented in Sect. 1.3.3 and their centralized implementations in Sect. 2.4.1 and Sect. 3.4.1. The pose $\mathbf{q}^i$ of agent i must be used to express the depth measurements in the reference frame of the map. Similarly to the centralized architecture, the pose uncertainty is taken into account [Nguyen *et al.* 2012, Oleynikova *et al.* 2020], as the new measurements are weighted with $1/z^2_{\mathbf{q}^i}(\mathbf{v})$, where $z_{\mathbf{q}^i}(\mathbf{v})$ is the distance between voxel $\mathbf{v}$ and the current pose $\mathbf{q}^i$. The state of a voxel $\mathbf{v}$ is set to *known* (either *occupied* or *empty*) if $w(\mathbf{v}) \geq W_{\text{th}}$ and to *unknown* if $w(\mathbf{v}) < W_{\text{th}}$, where the threshold W_{th} depends on the sensing range of the depth sensor.

In the remainder of this section, we will present a CPU-based distributed manifold mapping derived from [Duhautbout *et al.* 2019] which allows each agent to compute TSDF (sub-)maps on its own embedded computer. The general flowchart of the mapping module is depicted in Fig. 4.3. To synchronize the map among the agents, the map is subdivided into different *patches*. Each *patch* is a local TSDF with a unique ID and linked to a local frame (i.e. the frame of the first depth map processed). This frame is used to integrate the depth maps until a certain user-defined event is triggered. We assume that a new *patch* is created when a given number of depth maps has been integrated into the current map. Once this happens, the current *patch* is locally stored into an agent's *private map* and shared with the others, and a new current *patch* is initialized. In addition, once a new patch is created, the producer broadcasts the updated list of IDs of its private *patches*. Thus, the robots can tell if a patch is missing. A client-server protocol is used to request a patch which might have been lost during a communication attempt. Moreover, communications are coordinated so that the request is always handled by the nearest robot even if it is not the patch producer. This mechanism allows to reduce the communication load, possible network saturation and package drop-out. When an agent receives a *patch* sent by another agent, it stores it locally into its *public map*. Note that an approximation of the global map, denoted by $\widehat{M}_i$ can be constructed by agent i by merging its current *patches*, its private map and its public map. To rebuild the TSDF map from the collection

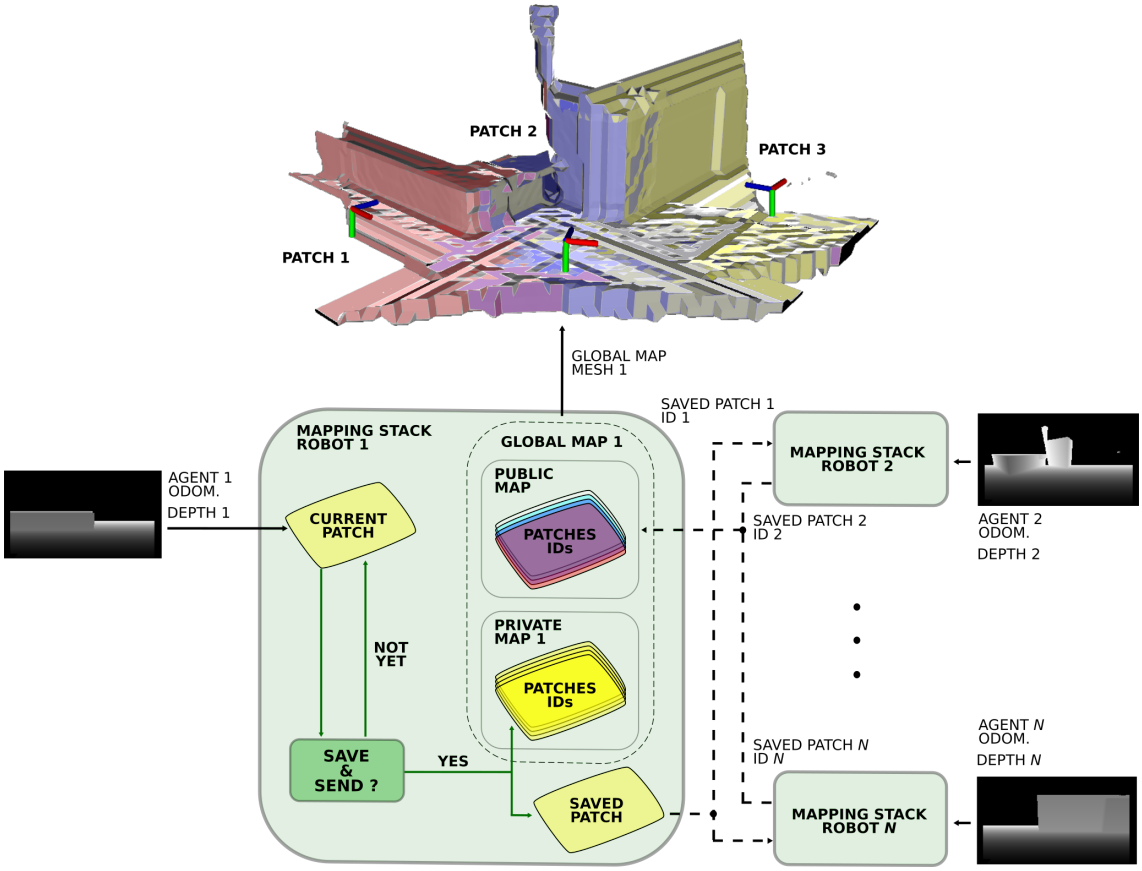


Figure 4.3: *Flowchart of distributed mapping module for a team of 3 robots:* each robot sends its depth map to the mapping module which generates and broadcasts the patches to the team. It also gathers the other patches of the agents to form its own global map. The global map shown at the top of the figure, has been obtained by fusing three patches: patch 1 (red), patch 2 (blue) and patch 3 (yellow). The intra- and inter-block connections are represented with solid and dashed lines, respectively.

of patches, the volumes are aggregated and the overlapping regions between the patches are fused together by summing up the weights and by computing the weighted average of distance values for each TSDF voxel. Again, to avoid the communication network from being overloaded, the current patch of an agent is not accessible to the others until its completion. Therefore, even without any communication loss, the agents do not have access to the global map M, simultaneously. However, it is worth noticing that the list-and-request mechanism to synchronize the older patches between the agents ensures that most of the global map $\widehat{M}_i$ is available them, except for the most recent patches which are currently being built.

4.4.2 ISE extractor, viewpoint generation and clustering

The ISE extraction, viewpoint generation and clustering processes are identical to the single-robot architecture (cf. Sect. 2.4.2 and 2.4.3), as they only rely on the generated TSDF map, irrespective of its source. We denote by $\widehat{\mathcal{U}}_i = \{U_1^i, U_2^i, \ldots, U_{N_C^i}^i\}$ the set of clusters that should be observed by agent i to complete the reconstructed surface.

4.5 Distributed Planning

In this section, the Distributed TSP-Greedy Allocation (dist-TSGA) planner is described. In particular, we present a two-level planning algorithm for addressing the inspection problem with multiple robots in a distributed fashion. The proposed distributed version of the high-level TSGA graph-based planner is such that the paths are autonomously computed by each agent and they depend on their available local information. Each robot assigns and schedules on its side the visit of clusters for the team, sends its path to all low-level planners, and follows it by default. The low-level planner has been improved to manage possible allocation redundancies, ensure safe navigation and monitor path update depending on map evolution.

4.5.1 Inspection problem

The high-level planner of robot i schedules the visit of clusters according to its global TSDF map $\widehat{M}_i$. As in Chapter 2 and 3, let $\mathcal{G}_i = (\widehat{\mathcal{U}}_i, \mathcal{E}_i, \{a_{UV}\}_{(U,V)\in\mathcal{E}_i})$ be a weighted directed graph, where $\widehat{\mathcal{U}}_i$ is the set of clusters uncovered from $\widehat{M}_i$, $\mathcal{E}_i$ is the set of edges, and $\{a_{UV}\}_{(U,V)\in\mathcal{E}}$ is the collection of weights associated with the edges. As seen in Chapter 2, the inspection problem for an agent can be stated as a *maximum Open Asymmetric Traveling Salesman Problem* (**maxATSP**). It consists in finding a maximum-utility Hamiltonian path on $\mathcal{G}_i$, i.e. a path which visits every configuration exactly once, maximizing the sum of the weights on the arcs f_{UV}, $U \neq V$, starting from the initial configuration $\mathbf{q}_0$ and ending to $\mathbf{q}_f$. As in eq. (2.4a) in Sect. 2.5.1, let us denote by **maxATSP**$(\widehat{\mathcal{U}}_i)$, a set function that takes the set of clusters $\widehat{\mathcal{U}}_i$ as input and outputs its utility value $p_i \in \mathbb{R}^+$, from which the path sequence $\mathbf{P}_{0,f}$ can be computed. Solving **maxATSP**$(\widehat{\mathcal{U}}_i)$ allows to find an optimal path for an agent, to visit the clusters of $\widehat{\mathcal{U}}_i$.

Similarly to Chapter 3, the agents should work together and each of them should be assigned to a subtask. Let $\widehat{\mathcal{U}}_i^k$ be the set of clusters assigned to agent k from the perspective of agent i, such that $i, k \in \{1, \ldots, N\}$, such that $\bigcup_{k=1}^{N} \widehat{\mathcal{U}}_i^k = \widehat{\mathcal{U}}_i$. Then, the

assignment problem can be stated as follows,

$$\max_{\widehat{\mathcal{U}}_i^1,\ldots,\widehat{\mathcal{U}}_i^N \subset \widehat{\mathcal{U}}_i} \Big\{ \sum_{k=1}^{N} \text{maxATSP}(\widehat{\mathcal{U}}_i^k) \;\Big|\; \widehat{\mathcal{U}}_i^k \cap \widehat{\mathcal{U}}_i^\ell = \emptyset,\, k \neq \ell,\, \bigcup_{k=1}^{N} \widehat{\mathcal{U}}_i^k = \widehat{\mathcal{U}}_i \Big\}, \tag{4.1}$$

where $\sum_{k=1}^{N} \text{maxATSP}(\widehat{\mathcal{U}}_i^k)$ is a non-decreasing set function, and the space of feasible paths has the structure of a simple partition matroid [Wilson 1973]. The maximisation problem (4.1) can be approximately solved via local greedy heuristics (cf. examples [Bian *et al.* 2017, Nemhauser *et al.* 1978, Fisher *et al.* 1978]), with the Distributed TSP-Greedy Allocation (dist-TSGA) procedure reported in **Algorithm 5**.

At each ISE extraction, the clusters are formed and their shortest Euclidian distance to an agent of the fleet is computed, in order to rank them by distance in ascending order for the greedy heuristic. The dist-TSGA planner greedily assigns each cluster to an agent. A cluster is assigned when it locally maximizes the overall utility of the team. The path of agent i which results from the allocated clusters $\widehat{\mathcal{U}}_i^i$, is denoted by $\mathbf{P}^i_{\widehat{\mathcal{U}}_i^i}$, and its utility value by p_i^i. Differently from the centralized system, agent i sends only its own path $\mathbf{P}^i_{\widehat{\mathcal{U}}_i^i}$ (or $\mathbf{P}^i$, for short, in the following) to its low-level planner (see Sect. 4.5.2) for collision-free trajectory generation, once the assignment has been made. Unlike classical insertion methods where a cluster is added to a robot's path [Jawaid & Smith 2015], the **maxATSP** problem is solved for the extended cluster set $\widehat{\mathcal{U}}_i^k \cup V$ with $V \in \widehat{\mathcal{U}}_i \setminus \widehat{\mathcal{U}}_i^k$. This strategy maximizes the individual utility of the agents over disjoint sets so as to maximize, in turn, team-wise utility, and it is suitable to a distributed implementation in which only local information is used (such as, local free space, ISEs, $\widehat{\mathcal{U}}_i$ related to $\widehat{M}_i$ of agent i). The maximization of the utility function over the long run, pushes the agents towards the most valuable areas in terms of completeness, even if it is not the shortest path. This might prompt an agent to move to configurations that cover an area at the frontier between the

Algorithm 5: dist-TSGA

```
Set Û_i^i = ∅, and p_i^i = 0 for agent i ∈ {1, 2, ..., N};
foreach cluster V ∈ Û_i do
    ℓ ← argmax_{k∈{1,2,...,N}} {maxATSP(Û_i^k ∪ V) − p_i^k};
    if ℓ = i then
        Û_i^i ← Û_i^i ∪ V;
        p_i^i ← maxATSP(Û_i^i);
        P^i_{Û_i^i} ← {p_i^i, Û_i^i};
    Send path P^i_{Û_i^i} to the low-level planner;
```

known and unknown surface, which contains multiple ISEs and scan them all. However, some ISEs can be completed before the planned visit of an agent. To prevent long back-and-forth travel moves, the low-level planner computes the remaining ISEs of paths since the last planning iteration, and it possibly waits for an update (see Sect. 4.5.2). This kind of update rule ensures a reactive visit of uncovered ISEs, as the map grows over time. The scanning procedure stops when no more ISEs remain.

In the distributed architecture, because communication is not instantaneous, the real global map M cannot be considered for planning. Only an estimation of global map $\widehat{M}_i$ is accessible to the agent i depending on its available local information. Moreover, due to material constraints such as network saturation or packet losses, a newly-created patch may not be received by all the agents. The mapping stack allows the agent to recover the public map by requesting the missing patches, and thus, the union of gathered patches $\widehat{M}_i$, converging to M, (see Sect. 4.4.1, Fig. 4.3). However, the recovery process is not instantaneous. Hence, because the agents do not have the same knowledge of the global map, the high-level planners of the team to allocate some clusters multiple times. To overcomes this drawback, the low-level planner has been improved to manage assignment redundancy.

4.5.2 Low-level planner

The low-level planner receives the tour of clusters to visit and the coordinates of their respective configurations, computed by the dist-TSGA planner for a given update of the TSDF map $\widehat{M}_i$. The main purpose of the module remains to find the local path in the free-space using the LazyPRM* algorithm (cf. Sect. 1.4.3) and to send it to the robot's controller. **Algorithm 6** describes the procedure. The major change, compared to the centralized architecture concerns the redundancy check which manages possible assignment conflicts (lines 5-8 in Algorithm 6). To ensure a path synchronization, robot i broadcasts the currently explored path $\mathbf{P}^i$ and the corresponding poses to the other robots. It also gathers their paths $\mathbf{P}^k$, $k \in \{1, \ldots, N\}$, $i \neq k$. The low-level planner consists of multiple subfunctions:

1. Given a start and an end pose, $\mathbf{q}^i_{curr}$ and $\mathbf{q}$ respectively, the *path* $\mathbf{p}^i_{curr,q}$ (i.e. the sampled set of consecutive robot configurations) in the free space is computed, given by the TSDF volume. If the goal configuration can be reached by the robot, the path $\mathbf{p}^i_{curr,q}$ is sent to the controller (*feasible* is set to *True* by **LazyPRM***).

2. The function ensuring that the goal has been reached, **GoalReached**, is identical to the single-robot case, see Sect. 2.5.3, Item 2.

3. The utility check function, **UtilityCheck**, is identical to the single-robot case, see Sect. 2.5.3, Item 3, with $\widehat{C}^i$ the number of ISEs left and $\widehat{C}^i_{start}$ the set of ISEs to scan by travelling $\mathbf{P}^i$.

4. The obstacle check function, **ObstacleCheck**, is identical to the single-robot case, see Sect. 2.5.3, Item 4.

5. The traffic policy function, **TrafficPolicy**, is identical to the one used in the centralized architecture, see Sect. 3.5.2, Item 5.

6. Because of the distributed mapping and the possible inconsistencies in the global maps, multiple agents might be assigned to the same viewpoints of a cluster. If so, to decide which agent should visit a viewpoint, we consider each agent's progress along its current path sequence. It consists in counting the number of paths $\mathbf{p}^i$ followed by each robot ("steps") before reaching a cluster. The low-level planner will then assign a viewpoint to the agent which can reach it first, according to its order in the path and the current location of agents. For the agents which have not been selected, a re-planning is performed (*redundancy* set to *True* by **RedundancyCheck**). The procedure is depicted in Fig. 4.4. In **Situation 1**, the robot which

has priority over the others, to visit a redundantly-assigned cluster may stop its progression along its current path before reaching it, as it asks for an update for any reason (obstacle, low remaining path utility). As a consequence, it will suppress a single redundant allocation, or at least, lower its priority in case of multiple redundant allocations. In the absence of a feedback mechanism, each low-level planner ensures that the robots continue to follow their paths until the priority robot has reached the redundant cluster. If a robot has no longer maximum priority, its newly updated path sequence will be considered, and another robot will take over. This mechanism ensures that the cluster is visited and avoid unnecessary path updates.

Algorithm 6: Distributed multi-robot low-level planner for agent i

Inputs: $\mathbf{q}^i_{curr}$, $\mathbf{P}^i$, $\widehat{Q}_i$, $\widehat{M}_i$, $\widehat{C}^i$, $\mathbf{q}^k_{curr}$, $\mathbf{P}^k$, $\quad \forall k \in \{1, \ldots, N\},\ i \neq k$;
Set $reached = False$; $obstacle = False$; $update = False$; $maneuver = None$; $redundancy = False$;
$\widehat{C}^i_{start} \leftarrow \widehat{C}^i$;
foreach $\mathbf{q} \in \widehat{Q}_i$, following $\mathbf{P}^i$, **do**
  $redundancy \leftarrow$ **RedundancyCheck**($\mathbf{q}^i_{curr}$, $\mathbf{q}^k_{curr}$, $\mathbf{P}^i$, $\mathbf{P}^k$) ;
  if *redundancy* **then**
    wait for high-level re-planning ;
    break ;
  $update \leftarrow$ **UtilityCheck**($\widehat{C}^i_{start}$, $\widehat{C}^i$) ;
  if *update* **then**
    wait for high-level re-planning ;
    break ;
  $\{feasible, \mathbf{p}^i_{curr,q}\} \leftarrow$ **LazyPRM***($\mathbf{q}^i_{curr}$, $\mathbf{q}$, $\widehat{M}_i$) ;
  if *feasible* **then**
    send $\mathbf{p}^i_{curr,q}$ to Controller ;
    while *(*not *reached)* **do**
      $reached \leftarrow$ **GoalReached**($\mathbf{q}^i_{curr}$, $\mathbf{q}$) ;
      $obstacle \leftarrow$ **ObstacleCheck**($\mathbf{q}^i_{curr}$, $\widehat{M}_i$) ;
      $maneuver \leftarrow$ **TrafficPolicy**($\mathbf{q}^i_{curr}, \mathbf{q}^k_{curr}$, $\widehat{M}_i$) ;
      if $maneuver \neq None$ **then**
        send *maneuver* to Controller ;
        wait for *maneuver* ;
        $maneuver = None$;
      if *obstacle* **then**
        send *stop* to Controller ;
        wait for high-level re-planning ;
        $reached = True$;

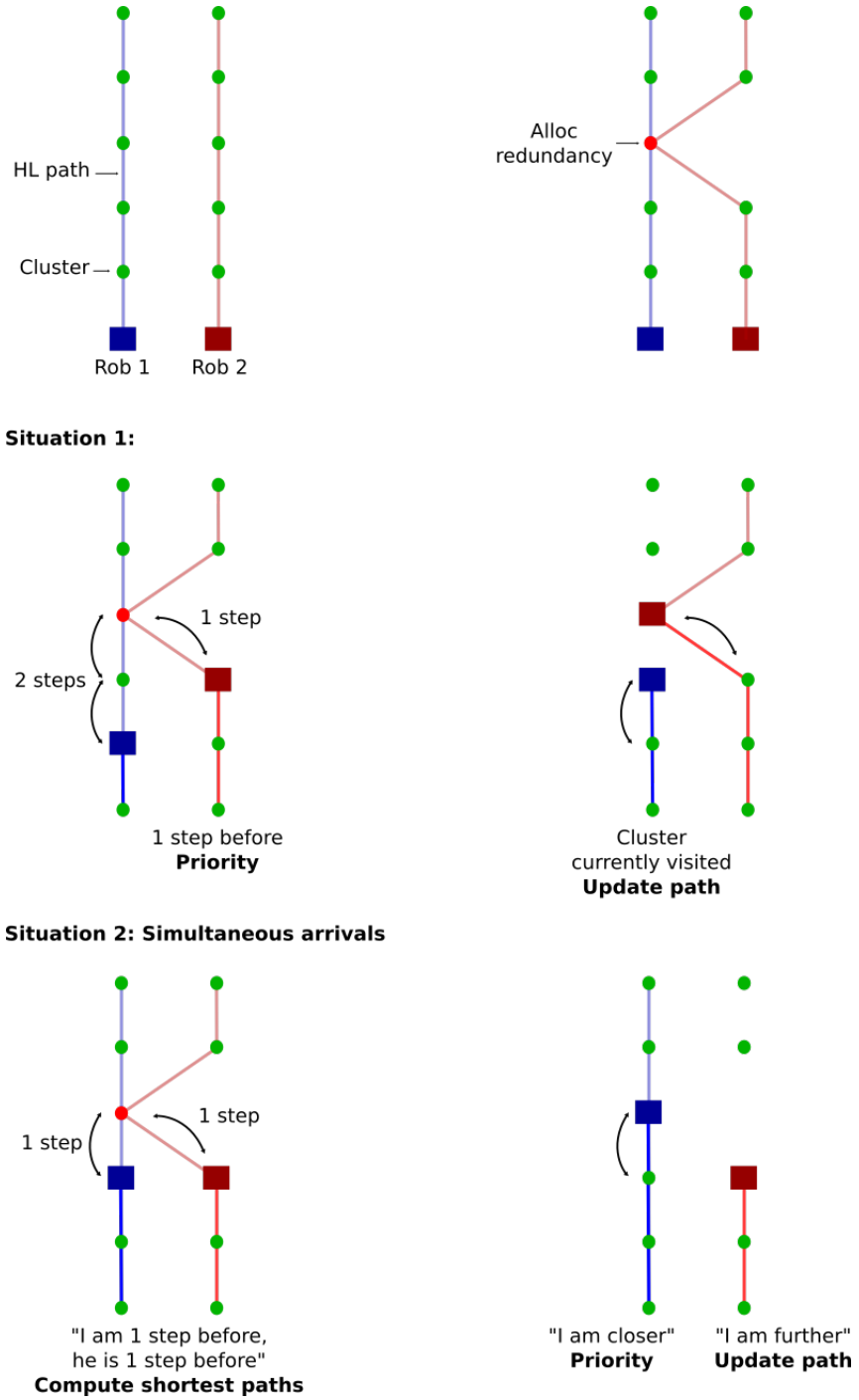


Figure 4.4: ***RedundancyCheck*** *function*: [top-left] Nominal case: Robot 1 (blue square) and 2 (red square) and the paths to follow without redundant allocation (the clusters are the green disks). [top-right] Presence of a redundantly-assigned cluster (red disk) between the paths. [middle] **Situation 1**: given the robot poses and path sequences, the low-level planner identifies the closer robot in terms of number of maneuver or "step" to reach the cluster. The robot that takes the priority continues its path, and the others keep moving until they reach the cluster located just before the redundantly-assigned one, and wait for a path update. [bottom] **Situation 2**: Multiple robots arrive simultaneously. Here the distances to the cluster are computed and the closest robot takes the priority and continues to follow its path. Again, the others wait for an update.

4.6 Numerical experiments

Similarly to the previous single- and multi-robot cases, the proposed distributed multi-robot architecture has been validated via realistic numerical experiments. For these simulations, we considered multirotor UAVs with 4 degrees of freedom: their 3D position $[x, y, z]^T$ and their yaw angle ψ. We considered a fleet of 3 and 5 UAVs. The **Powerplant** model described in Chapter 2 (cf. Sect 2.6.1), have been re-used. In Chapter 3 (cf. Sect. 3.6.4), the numerical experiments showed that the centralized multi-UAV architecture allows to reduce the reconstruction time when compared to the single-UAV case, without sacrificing accuracy and provide better performance than the NNB planner. In this chapter, we will compare the performance of the two multi-robot architectures, to study their pros and cons.

4.6.1 Simulation setup

We used the RotorS simulator [Furrer *et al.* 2016] to model multirotor UAVs equipped with a stereo camera, in the ROS-Gazebo environment. As in Chapter 2 (cf. Sect 2.6.1) and Chapter 3, the depth map has been corrupted using a Gaussian-noise model. Distributed mapping is performed with the method proposed in [Duhautbout *et al.* 2019], which allows each robot to compute its own local volume and send it to the other robots,

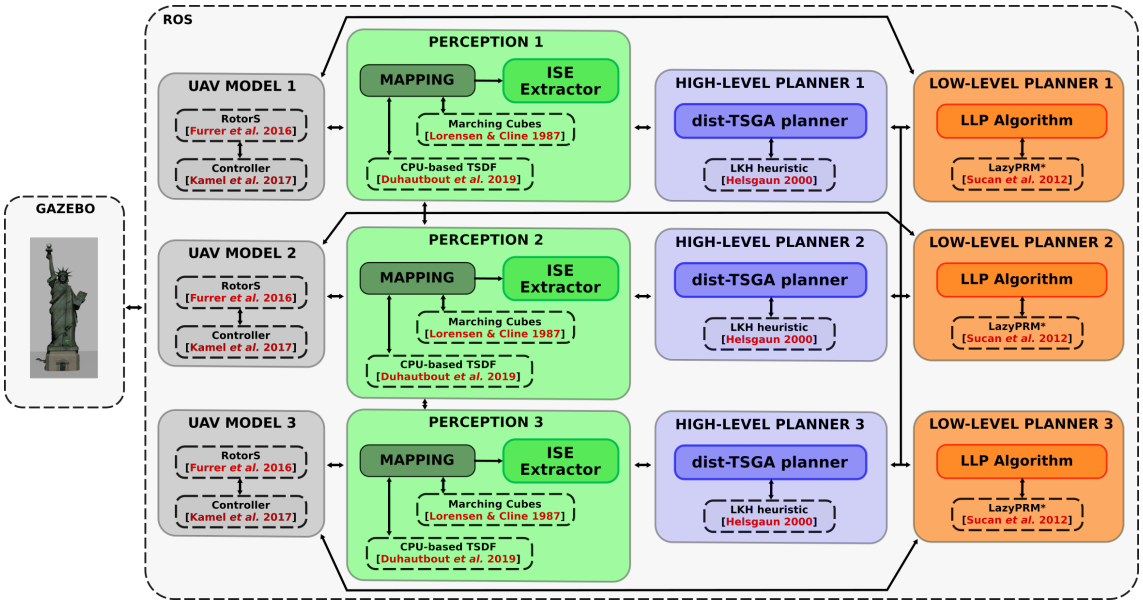


Figure 4.5: *Distributed simulation stack for a fleet of 3 UAVs*: Gazebo interacts with ROS. Some existing software modules (dashed bubbles) have been adapted and integrated into our software nodes: **Mapping**, **ISE Extractor**, **dist-TSGA planner** and **Low-level planner** (colored bubbles).

so that a global map can be reconstructed. The event that triggers the integration and broadcast of a new *patch* to the other agents, is that 5 depth maps have been processed (cf. Sect. 4.4.1). The weight increment has been modified to be quadratic. Unlike the centralized case, the mapping algorithm is now CPU-based and it can be run on an embedded computer with limited resources. The mesh is reconstructed with MarchingCubes [Lorensen & Cline 1987]. Collision-free UAV paths are found by the LazyPRM* planner from the OMPL [Şucan *et al.* 2012]. The UAVs track the generated paths using MPC [Kamel *et al.* 2017]: the reference translational velocity was fixed at 0.6 m/s.

The software used to model the aerial vehicles, includes open-source implementations of the algorithms mentioned above, which are integrated into our ROS software architecture as depicted in Fig. 4.5. The simulation stack interacts with Gazebo which simulates the 3D model of the environment to reconstruct, and the robot's behaviour via RotorS. The ROS nodes are implemented in Python (v2.7 and v3) and C++, and the architecture is composed of three main functional modules: *Perception*, *High-level planner* and *Low-level planner*. This modular architecture allows the nodes to be easily modified if needed. Differently from the centralized architecture, all nodes have been designed to run on CPU, and the multi-robot system was simulated on a high-performance 1U Supermicro SYS-1028GR-TR[1] computer, equipped with 2 Intel Xeon E5-2680v4 CPUs (14 cores/28 threads at 2.4 GHz), 256 GB of RAM and an Nvidia Tesla P100 PciExpress 12 GB VRAM graphics card. This allowed us to overcome the limitations of conventional desktop computers, when simulating CPU-based instances of our algorithms on small and medium-size robot teams. The **Powerplant** virtual environment has been considered (cf. Fig. 2.5), and scaled to fit in a box of size $65 \times 42 \times 15\,\mathrm{m}^3$. The identical UAVs are initially located in the same area, around the ground station (magenta dots in Fig. 4.6), as in the simulations for the centralized architecture in Sect. 3.6. The parameters used in the simulations are reported in Table 4.1.

4.6.2 Metrics

The single-UAV planner proposed in Chapter 2 has been considered as a baseline to evaluate the new distributed architecture, in terms of cumulated path length and completion time (to cover the entire 3D environment). This includes travel time and sensing time (e.g. one depth map integration and map update). As in the previous chapters, the reconstructed 3D surface has been evaluated with CloudCompare using the M3C2 algorithm [Lague *et al.* 2013].

[1] www.supermicro.com/en/products/system/1U/1028/SYS-1028GR-TR.cfm

4.6.3 Results and discussion

The results of our numerical experiments are reported in Table 4.2. As in Chapter 3, we refer to the results of the single-UAV planner with perfect measurements as *SR* (cf. Sect. 2.6.4) and to those with noisy depth measurements, with SR^*. We compared *SR* and SR^* with both the TSGA and dist-TSGA planners for 3 and 5 UAVs. To obtain statistically-significant values, 10 trials per scenario were carried out.

As the number of robots grows, the execution time decreases (see 7th and 8th row of Table 4.2): in fact dist-TSGA (TSGA) with 3 UAVs allows to scan 70.0% (68.8%) faster, compared to the single-UAV case. With 5 UAVs, dist-TSGA (TSGA) is 81.5% (80.8%) faster than a single multirotor. The traveled distance per UAV is shorter than that of a single UAV, but the total path length of any team of UAVs is larger (see rows 1-6 of Table 4.2). The performances of both multi-robot architectures are almost identical in terms of path traveled and scanning time. A scan is performed in about 10 minutes with 3 UAVs and in about 6 minutes with 5 UAVs. The distributed algorithm outperform the centralized one in terms of reconstruction accuracy. Coverage improves as well, reaching 95.9% with 3 robots and 95.6% with 5 robots. It is worth pointing out here that the mapping accuracy strongly depends on the implementation chosen, and for the same resolution, the outcome of two different pieces of code may vary considerably. Since the MarchingCubes algorithm depends on voxel size, the reconstructed mesh will be more accurate

Table 4.1: Parameters used in the multi-UAV numerical experiments.

Parameter	**Powerplant**
Voxel resolution $r_{\mathbf{v}}$ [m]	0.3
Threshold W_{th}	0.3
$e_{\max}$ [m]	0.2598
Camera range [m]	[1.6, 8]
Camera FOV [deg.] (H, V)	90×60
e_d [pixels]	0.1
f [pixels]	376
B [m]	0.11
Collision radius [m]	1.0
UAV nominal speed [m/s]	0.6
δ_{pose} [m]	4.7
d_v [m]	2.0
$d_v^{\max}$ [m]	5.0
Penalty term λ_{tc}	0.3
Penalty term λ_{ic}	0.03
c_{thres} [%]	80

Table 4.2: Results of the multi-UAV numerical experiments on **Powerplant** (stats. over 10 trials).

Number of UAVs		1		3		5	
#	Algorithm	*SR*	*SR**	TSGA	dist-TSGA	TSGA	dist-TSGA
1	Path length rob. 1 [m]	–	–	273.0	260.8	172.2	173.6
2	Path length rob. 2 [m]	–	–	255.6	258.5	178.0	176.0
3	Path length rob. 3 [m]	–	–	261.4	261.7	175.2	175.3
4	Path length rob. 4 [m]	–	–	–	–	173.8	172.8
5	Path length rob. 5 [m]	–	–	–	–	179.8	174.3
6	Total path length [m]	780	785	790	781	879	872
7	Completion time [min.]	32	33′10″	10′20″	9′56″	6′22″	6′09″
8	Time gain [%] w.r.t. *SR**	–	–	68.8	70.0	80.8	81.5
9	Surface coverage [%]	91.5	90.4	91.0	95.9	90.6	95.6
10	M3C2 avg. error [cm]	0.14	−0.13	−0.26	0.62	−0.3	−0.73
11	M3C2 std. error [cm]	5.85	7.51	7.54	3.33	7.52	3.43
12	RMSE [cm]	5.86	7.51	7.55	3.39	7.52	3.51

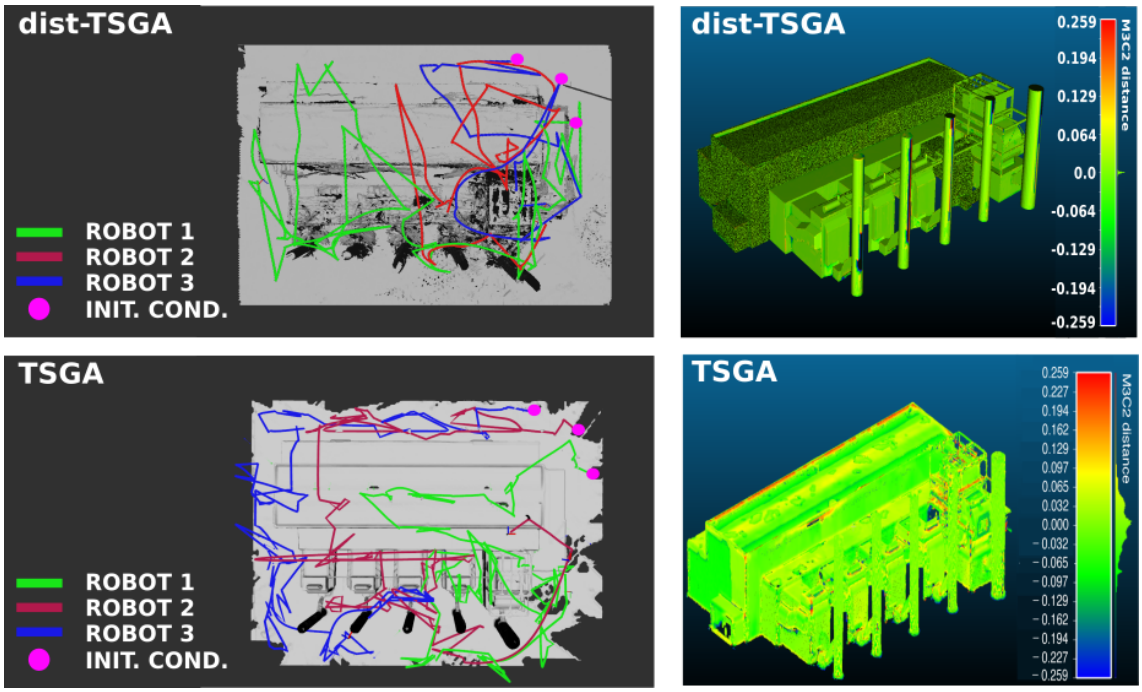


Figure 4.6: *Numerical experiments on* ***Powerplant***: [top] dist-TSGA and [bottom] TSGA; [left] Reconstructed mesh and 3D exploration paths $\mathbf{P}^1_{0,f}$, $\mathbf{P}^2_{0,f}$, $\mathbf{P}^3_{0,f}$ (green, red, blue) of 3 UAVs (the initial locations are marked with magenta dots); [right] Signed distance error: the color coding shows the error in meters with respect to the ground truth, computed with CloudCompare's M3C2 algorithm.

if the resolution is high. However, if the environment to scan is large, a high resolution entails a resource-intensive process for mapping and ISE extraction, which ultimately

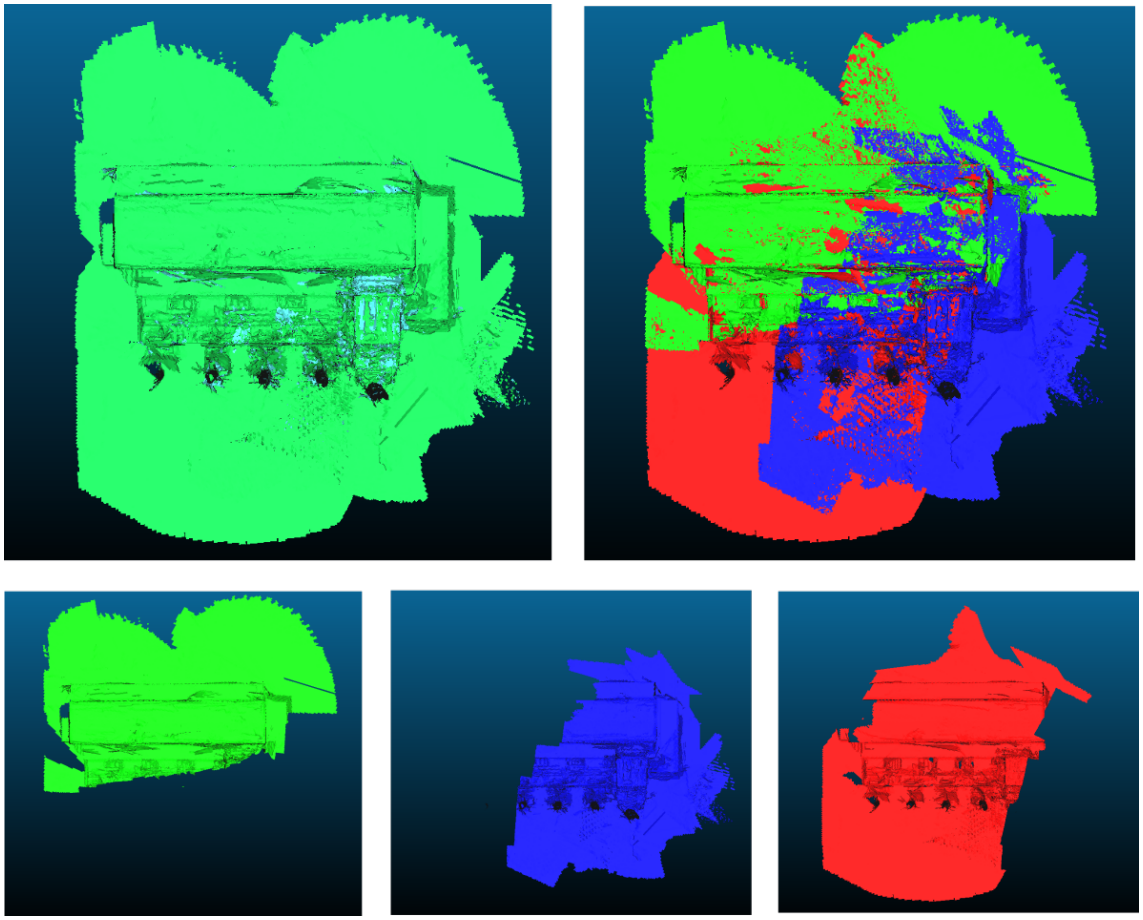


Figure 4.7: *Top views of reconstructed meshes for* ***Powerplant*** *with 3 UAVs*: [top left] full raw mesh of the building, and [top right], colored private maps. [bottom] Corresponding individual private maps.

makes the whole reconstruction chain prohibitively expensive. Therefore, a trade-off between computational efficiency and scan accuracy should be found. Some snapshots of the reconstructed meshes are shown in Fig. 4.7, where each robot reconstructs a portion of the entire volume.

The numerical experiments show that the two multi-robot architectures are successful in scanning the 3D environment, covering more than 90% of their surface, even in the presence of noisy depth measurements. In the next section, we will further extend our analysis, and study the accuracy and robustness of the centralized and distributed architectures deployed on mobile robots in real-life conditions.

However the demanding computational requirements of the distributed architecture, and in particular the need for a supercomputer for the simulations, indicate that the stack proposed in this chapter may not be easily applied to real robots. In the next chapter, we will adapt this implementation for usage of real hardware robots.

4.7 Conclusion

In this chapter, we have presented a distributed multi-robot architecture for the reconstruction of large-scale environments, as a direct extension of our previous centralized algorithm. Here, the mapping of the unknown environment is performed with a CPU-based distributed manifold mapping algorithm, which allows each agent to reconstruct local portions of the model, exchange and fuse them into a global map. The ISEs extraction module is similar to the one considered in the previous chapters, but it only relies on the local and shared information available to each agent. The depth map integration process, an imperfect communication, or the loss of patches may lead the agents to have only an approximate knowledge of the map. Hence, the high-level greedy allocation strategy allows each robot to estimate distributively the behaviour of the fleet, to plan its path and send it to the team. The low-level planner has been improved to manage possible conflicts in viewpoints allocation. Each agent's dist-TSGA planner addresses the viewpoint allocation problem for the robots of the team, by using a greedy heuristic and a TSP insertion policy. Once each agent has carried out its path allocation, its own path is broadcast to all low-level planners which handle possible conflicts and ensure safe navigation. The proposed solution contributes to make the robot team more robust and resilient, by multiplying the possible points of failure. Numerical experiments with ROS and Gazebo have compared the centralized and distributed architectures, showing similar performances, which are, again, in line with the flight autonomy of existing battery-powered multi-rotor UAVs. However the process is resource intensive and real-time computation and communication issues are expected, when the distributed architecture is implemented on hardware platforms. To conclusively validate our approach, the next chapter will then present real-world experiments with the three proposed architectures (single, centralized and distributed) for 3D surface reconstruction of unknown environments.

CHAPTER 5

Real-world experiments

Chapter outline

5.1 Introduction 93

5.2 Robots 94

5.2.1 Hardware 94

5.2.2 Software 95

5.3 Environments 96

5.4 Metrics 98

5.5 Results 100

5.6 Conclusion 103

5.1 Introduction

In the previous chapters, the proposed 3D reconstruction algorithms have been tested with multiple UAVs in realistic ROS-Gazebo simulation environments. While successful, the influence of a number of factors (such as, imperfect communication with packet losses and/or delays, uncertain robot localization, image blur, etc.) could not be precisely assessed during the simulations. To address this issue and conclusively validate the theory, in this chapter we will present the results of extensive real-world experiments. Here, the single-robot, centralized and distributed multi-robot software stacks have been integrated into team of wheeled robot and tested in two indoor environments with complex geometry and of increased complexity. To cope with variable environmental conditions (e.g. change in luminosity during the day), and slight differences in robot configuration (camera calibration, level of charge of the battery, etc.), a statistical analysis over multiple trials has been carried out.

5.2 Robots

The experimental campaign took place at ONERA, where multiple robotic platforms are available. The experiments have been conducted with a team of 4 identical Wifibots[1]. Ground robots are ideal to validate new algorithms, since they are sturdier than aerial robots and require less space and assistance. In addition, the safety requirements are less strict.

To setup the experiments, the simulation stack developed in Chapters 4 and 5 for multiple UAVs, has been adapted to a team of wheeled nonholonomic robots. The different modules of our architectures have been integrated on the ground station or on the robots, and functional tests have been performed to validate the full system or each module taken individually.

5.2.1 Hardware

Each Wifibot is equipped with an Intel NUC 7i7BNH[2] embedded computer and with a stereo-rig consisting of two IDS[3] UI-1241-LE monochromatic cameras (see Fig. 5.1). The cameras have a focal length of 4 mm, a resolution of 640×512 pixels, a frame rate of 5 Hz, and their auto-shutter is enabled. The stereo-rig used by the perception module has a

[1] https://www.wifibot.com/
[2] https://ark.intel.com/
[3] https://fr.ids-imaging.com/

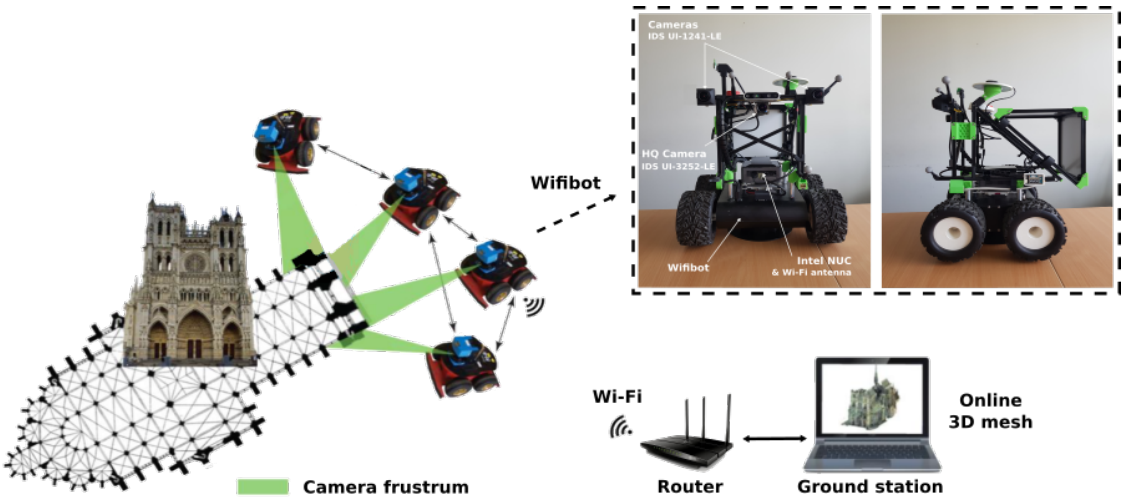


Figure 5.1: *Hardware architecture*: The robots communicate with each other and with the ground station via a router. [Top-right] Front and side view of one of the Wifibots used in the field deployment. All Wifibots are identical, i.e. they have the same software and hardware components.

baseline B of 26 cm. The cameras have been calibrated using Kalibr [Furgale *et al.* 2014], following the procedure presented in Appendix A.2.3. We also equipped the robots with a high-quality IDS UI-3252-LE monochromatic camera (1600×1200 pixels, focal length of 4 mm), to capture images at a frequency of 3 Hz. With these images, we performed an offline 3D reconstruction of the environments by photogrammetry (Agisoft MetaShape[4]), that was used as our ground-truth (GT) map. The ground station is a Dell Precision 5530 laptop with 2.60 GHz Intel Core i7, 16 GB RAM and Quadro P1000 graphics card. The Wifibots and the ground station communicate via Wi-Fi using a 5 GHz band TP-Link Archer-C7 router. The communication graph of the network is a complete graph i.e. each agent can communicate with every other agent. The ROS processes running on every robot are synchronized using a Node Manager[5]. The multi-master architecture allows to run nodes on a networks of multiple computers. Each NUC contains an Intel Dual Band Wireless-AC 8265 network card, that ensures a maximum data-transfer speed of 867 Mb/s. Clock synchronisation is guaranteed by a Network Time Protocol (NTP) reference. The observed autonomy of the Wifibots is between 20 and 30 minutes.

5.2.2 Software

The embedded ROS software and architecture are almost identical to the one used in numerical experiments. The software components of the centralized and distributed architectures, the computers where they are executed (Ground Station (GS), or embedded computer of the Wifibots), and the corresponding references, are summarized in Tables 5.1 and 5.2, respectively. We recall that the single-robot architecture is a special case of the centralized one with a single robot. Only minor changes have been made to the simulation architectures to adapt them to the real sensors and physical constraints of the Wifibots. In particular, the code that generated the emulated depth maps and odometry has been replaced with validated software modules: the depth maps are computed with the ELAS algorithm [Geiger *et al.* 2010] and the odometry is estimated with the vision-based eVO algorithm [Sanfourche *et al.* 2013]. This algorithm is not endowed with a loop closure capability: as a consequence, a localization drift may occur depending on the traveled distance and occurrence of sharp turns. The reconstruction process is initialized by synchronizing each robot with a global reference frame whose origin is defined by an AprilTag calibration cube [Olson 2011]. In the centralized architecture, the map is updated every time a new depth map is sensed (at around 1 Hz), and the computations are all realized on the ground station. In the distributed architecture, instead, a new patch is

[4] https://www.agisoft.com/

[5] http://wiki.ros.org/node_manager_fkie, doc. http://fkie.github.io/multimaster_fkie/node_manager.html

Table 5.1: Software modules of centralized architecture (GS stands for ground station).

Architecture	Centralized		
	Component	Reference	Computer
Localization	eVO	[Sanfourche *et al.* 2013]	Wifibot
Depth map	ELAS	[Geiger *et al.* 2010]	Wifibot
Mapping	Central. TSDF	[Zeng *et al.* 2017, Hardouin *et al.* 2020a]	GS
Completeness	ISE extraction	[Hardouin *et al.* 2020a]	GS
Surface-based plan.	TSGA	[Hardouin *et al.* 2020a, Helsgaun 2000]	GS
Local planning	Low-level planner	[Hardouin *et al.* 2020a, Şucan *et al.* 2012]	Wifibot
Controller	MPC	[Kamel *et al.* 2017]	Wifibot

Table 5.2: Software modules of distributed architecture.

Architecture	Distributed		
	Component	Reference	Computer
Localization	eVO	[Sanfourche *et al.* 2013]	Wifibot
Depth map	ELAS	[Geiger *et al.* 2010]	Wifibot
Mapping	Manifold TSDF	[Duhautbout *et al.* 2019]	Wifibot
Completeness	ISE extraction	Hardouin (T-RO)	Wifibot
Surface-based plan.	dist-TSGA	Hardouin (T-RO), [Helsgaun 2000]	Wifibot
Local planning	Low-level planner	Hardouin (T-RO), [Şucan *et al.* 2012]	Wifibot
Controller	MPC	[Kamel *et al.* 2017]	Wifibot

stored and broadcast every time 5 depth maps have been integrated into the current TSDF patch. We used a depth masker to ignore parts of the depth map, when another robot is present in the field of view. More specifically, a 2D mask of neutral depth values is placed at its location in the depth map: this way, the overall reconstructed volume is not altered by the presence of a moving object. The mapping and planning modules of the centralized and distributed architectures are identical to those presented in Chapter 3 and Chapter 4, respectively. The mapping and planning parameters used in the real-world experiments are summarized in Table 5.3.

5.3 Environments

Two different indoor environments have been considered in our experiments. The maps and photos of these environments are shown in Fig. 5.2. In the top row of the figure, a **Test arena** of $8 \times 7 \times 2$ m^3 consists of an open area with a central obstacle of a volume of $0.8 \times 0.8 \times 2$ m^3, surrounded by four walls covered by mattresses (green in Fig. 5.2). The ground

Table 5.3: Parameters used in the real-world experiments.

Parameter	**Test arena**	**Parking lot**
Voxel resolution r_v [m]	0.1	0.2
Threshold W_{th}	0.3	0.3
e_{max} [m]	0.0866	0.1732
Camera range [m]	[0.3, 5]	[0.3, 5]
Camera FOV [deg.] (H, V)	90×60	90×60
Collision sM-sm axis [m]	0.55 - 0.25	0.55 - 0.25
Robot nominal speed [m/s]	0.5	0.5
δ_{pose} [m]	1.3	1.3
d_v [m]	2.0	2.0
d_v^{max} [m]	3.5	3.5
Penalty term λ_{tc}	0.5	0.7
Penalty term λ_{ic}	0.01	0.01

station which allows to monitor the progress of the experiments is located outside the area of interest (labelled "0" in Fig. 5.2). This small and relatively-simple environment has been used to perform preliminary tests with our algorithms on the embedded computers of the robots. The second environment is an underground $21 \times 14 \times 2$ m^3 **Parking lot**, containing 3 pillars in the middle, and eight rectangular obstacles have been added at the ground level, to make the 3D reconstruction even more challenging:

- 3 (2 grey and 1 red) boxes of $1.5 \times 1.5 \times 1$ m^3,
- 2 (beige) boxes of $3 \times 0.5 \times 0.8$ m^3,
- 2 (black) boxes of $1.5 \times 0.5 \times 0.5$ m^3,
- 1 black box of $2 \times 0.7 \times 0.8$ m^3.

Figure 5.2: *Environments*: [top] **Test arena**, and [bottom] **Parking lot**. [left] Maps, and [center, right] photos of the two environments, including two panoramic views of the **Test arena**. The circled numbers indicate obstacles or areas of interest.

Here, because of several physical constraints (accessibility, power supply, shelter from rain and humidity), the ground station is located inside the scene. This part of scene has been ignored in the final 3D reconstruction of the parking lot. It is worth pointing out here, that differently from **Test arena**, the lighting of the parking a lot has not been designed to be uniform in whole volume, and some areas are darker than others (especially in the corners). This may negatively impact the accuracy of the pose estimated by the visual odometry algorithm and depth estimation as well. Finally, the entrance of the parking has no doors: therefore, the luminosity of the area is subject to large variations during the day. To alleviate this problem, we covered the entrance with a tarpaulin (see Fig. 5.4-[left]).

5.4 Metrics

The same metrics as in the numerical experiments have been considered. The traveled distance and execution time are measured for navigation, and the degree of completeness of reconstruction is evaluated by comparing the mesh obtained during the exploration and the GT map of the two environments obtained by photogrammetry (the two reference point clouds are shown in next section, see Fig. 5.5-[left] and Fig. 5.6-[left]). Again, the error distance between the mesh and the point cloud is computed with the M2C3 plugin of CloudCompare, and points of the GT are considered as outliers and pruned if the error is larger than the length of the half-diagonal of a voxel, i.e. $e_{\max} = r_{\mathbf{v}}\sqrt{3}/2$, where $r_{\mathbf{v}}$ is voxel's resolution. The percentage of surface mesh coverage is obtained by computing the ratio between the number of points of the pruned cloud and the GT. The accuracy of the pruned cloud is represented via the RMSE distance. Data exchange is also monitored: for the single-robot and centralized multi-robot architectures, it corresponds to the depth maps, odometry and paths' messages transmitted between the ground station and the Wifibots, whereas for the distributed algorithm, it corresponds to the patches, odometry and paths' messaged exchanged by the robots.

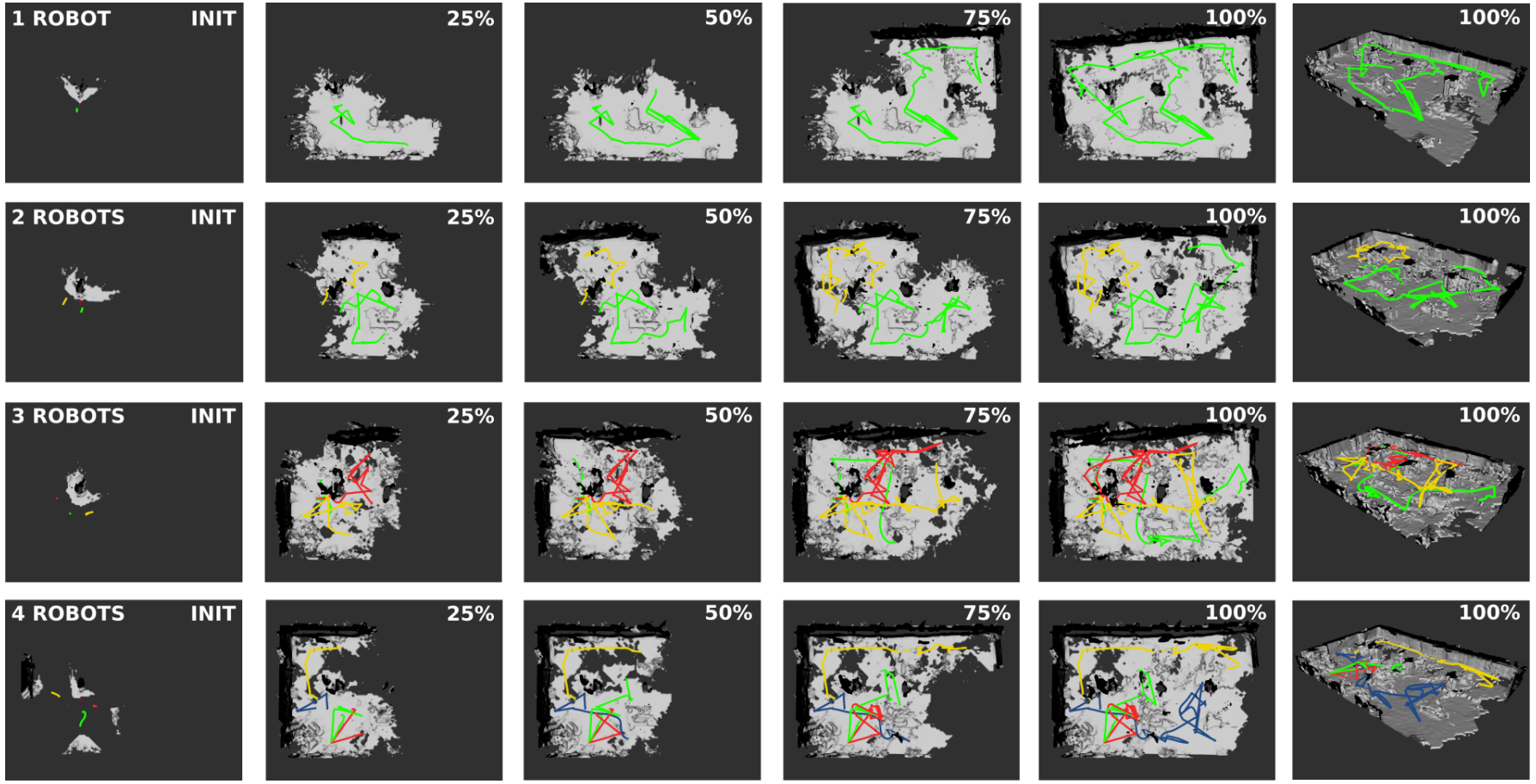


Figure 5.3: *Real-world experiments:* **Parking lot**. From left to right, progression of surface reconstruction in percentage, obtained with the *centralized architecture* (top view). In the rows, the number of robots varies between 1 and 4. The last column reports an isometric view of final reconstructed meshes and robots' paths. The ceiling of the parking lot has been removed to provide visibility of the interior. Video clip: https://youtu.be/yJ4lEyKrPGA

5.5 Results

Table 5.4 compiles our experimental results. For the **Test arena**, the reconstruction has been performed with a single robot, and a team of two robots for the centralized and distributed architectures. On the other hand, for the **Parking lot**, we considered a single robot and teams of 2, 3 and 4 robots for the centralized architecture, and a team of 2 robots for the distributed architecture. For each environment/team, 5 trials have been performed with identical initial positions and orientations for the robots. Fig. 5.3 shows different snapshots of the 3D reconstruction of the **Parking lot**, obtained with the centralized architecture. The single-robot case is considered as a reference, for both environments. From Table 5.4, we can notice that as the number of robots grows, the completion time decreases, but the cumulative path traveled by the team increases. However, the individual traveled distance of each robot, decreases. For example, in the **Parking lot**, a team of 4 robots allows to reduce the reconstruction time by 31.1% compared to the single-robot case. On the other hand, doubling the number of robots, leads to a reduction of completion time of 6.25%, in the **Test Arena**. The distributed architecture outperforms the centralized one, in terms of path traveled and execution time. However, the amount of data exchanged by the robots is significantly larger, and the communication network can be overloaded. In fact, the distributed architecture could not be tested with 3 and 4 robots, since the large number of patches exchanged by the robots saturated the wireless network.

Table 5.4: Results of the real-world experiments (statistics over 5 trials). The symbols "*SR*", "*C*" and "*D*" stand for "single-robot", "centralized" and "distributed", respectively.

	Test arena			**Parking lot**				
Number of robots	1	2		1	2		3	4
Architecture	*SR*	*C*	*D*	*SR*	*C*	*D*	*C*	*C*
Path length rob. 1 [m]	–	12.7	10.2	–	56.7	53.7	48.9	40.6
Path length rob. 2 [m]	–	14.3	14.0	–	56.9	53.4	47.3	41.1
Path length rob. 3 [m]	–	–	–	–	–	–	38.0	43.4
Path length rob. 4 [m]	–	–	–	–	–	–	–	35.9
Total path length [m]	20.6	27.0	24.2	106.0	113.6	107.1	134.2	161.0
Total path std. [m]	2.6	4.3	3.1	12.8	8.9	9.2	16.5	17.6
Completion time [min.]	2′53″	2′45″	2′37″	13′56″	11′06″	10′41″	10′14″	9′36″
Time gain [%] w.r.t. *SR*	–	6.25	9.25	–	20.30	23.30	26.60	31.10
Surface coverage [%]	85.6	80.3	91.0	89.1	81.1	88.4	76.4	73.0
M3C2 avg. error [cm]	0.27	0.40	0.12	0.01	−0.48	0.07	−0.18	0.61
M3C2 std. error [cm]	3.99	4.16	3.33	8.37	8.29	5.85	8.93	8.97
RMSE [cm]	4.00	4.18	3.33	8.37	8.30	5.85	8.94	8.99
Data exchanged [GB]	0.750	0.888	3.981	4.849	6.029	24.111	10.715	13.271

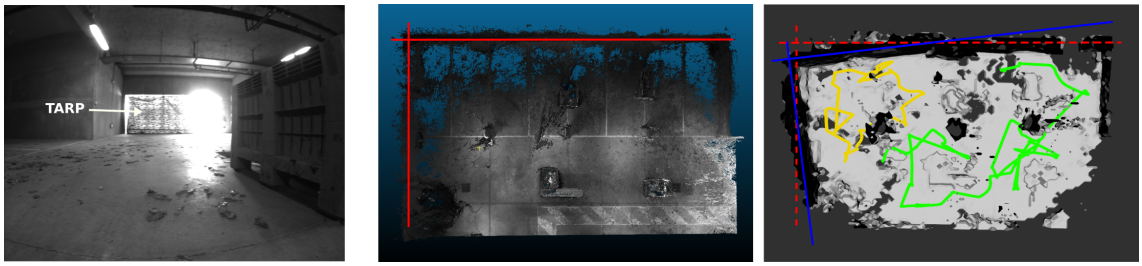


Figure 5.4: *Some causes and consequences of the odometry drift.* [left] Picture from left camera of the stereo-rig with the presence of a light halo at the parking lot entrance, partly covered by a tarp. [center] Top view of the ground truth of the parking lot with right angle corner formed by the walls (red lines). [right] Example of a consequence of the drift on the reconstruction: expected (red) vs. reality (blue).

In the two environments, the TSGA algorithms allowed to cover the entire area. For example, in the single-robot case, the reconstructed mesh covered 89.1% of the surface of the **Parking lot** for a voxel resolution $r_{\mathbf{v}} = 20$ cm and an admissible error $e_{\max} = 17.32$ cm, and covered 85.6% of the surface of the **Test arena** with $r_{\mathbf{v}} = 10$ cm and $e_{\max} = 8.66$ cm. Differently from the numerical experiments (cf. Sect. 3.6.4 and 4.6.3), as the number of robots increases, the surface coverage decreases, until reaching 73% for 4 robots in **Parking lot** and 80.3% for 2 robots in **Test arena**. In fact, to be integrated into the map, a depth map is associated to the pose used to sense it, estimated by a visual odometry algorithm and subject to drift. According to the odometry estimation technique, this can be caused by the environment (see Fig. 5.4-[left]), or the behavior of the robot as it explores the area e.g. doing prompt turns, stops or jagged paths. The estimation error due to the drift, has an impact on the TSDF volume when the depth map is integrated (see Fig. 5.4-[center-right]). Hence, the error accumulates as the number of robots increases, and the overall mesh accuracy decreases, with more outliers to prune and less space covered. The accuracy of the remaining points slightly decreases, with a RMSE between 8.37 cm and 8.99 cm for the **Parking lot** and between 4 cm and 4.18 cm for the **Test Arena**. Distributed mapping allows to cover more points than centralized mapping (88.4% vs 80.3%). The fusion policy in this case (cf. Sect. 4.4.1) overlaps the TSDF patches, which reduces the impact of odometry drift thanks to the better (raw) quality of mapping for the same resolution. The GT, reconstructed meshes and signed distance errors for the both architecture, are depicted in Fig. 5.5 and Fig. 5.6 (left, center and right, respectively). Fig. 5.3, Fig. 5.5 and Fig. 5.6 show that the centralized and distributed algorithms provide accurate 3D reconstruction, with a decent coverage in spite of odometry drift.

The robustness and mapping accuracy of the distributed architecture comes at a cost: in fact, the communication is more intensive. For example, with 2 robots, the total amount

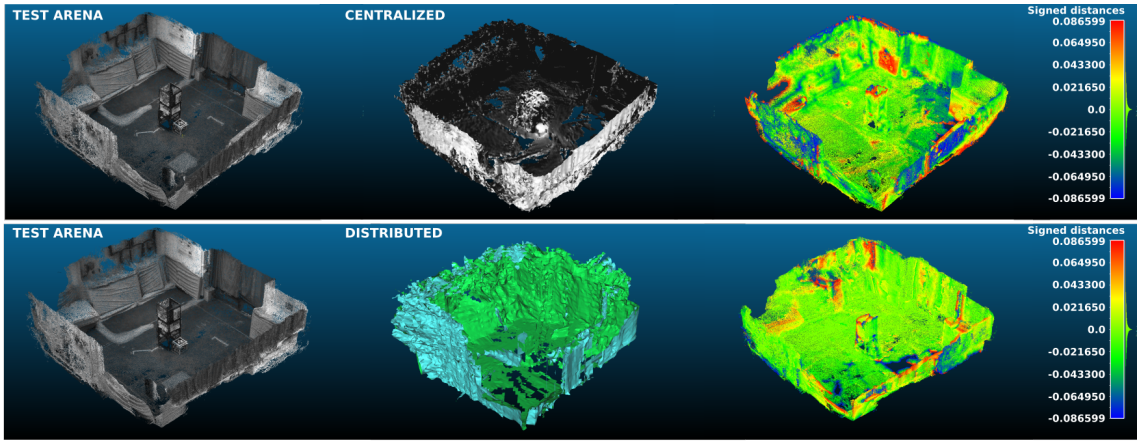


Figure 5.5: *3D reconstruction of the **Test arena** with two robots*, [top] centralized architecture, and [bottom] distributed architecture. [left] Ground-truth point cloud. [center] Reconstructed mesh. [right] Signed distance error: the color scale shows the error in meters with respect to the ground truth, computed with CloudCompare's M3C2 plugin.

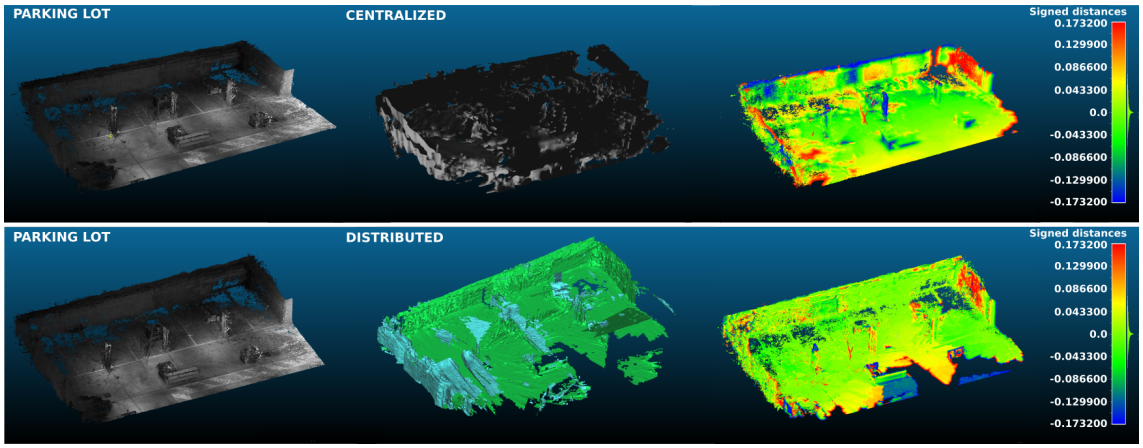


Figure 5.6: *3D reconstruction of the **Parking lot** with two robots*, [top] centralized architecture, and [bottom] distributed architecture. [left] Ground-truth point cloud. [center] Reconstructed mesh. [right] Signed distance error: the color scale shows the error in meters with respect to the ground truth, computed with CloudCompare's M3C2 plugin.

of data exchanged is, on average, 4 to 4.5 times larger than with the centralized architecture. In particular, the amount of data transmitted over the communication network decreases as the number of patches decreases (or equivalently, as the threshold on the maximum number of depth maps which triggers the creation of a new patch patch, increases, cf. Sect. 4.4.1). A larger threshold will result in fewer, larger patches, thus reducing the communication load. However, the loss of a patch might push the team to re-explore a vast area, and execute unnecessary maneuvers: as a consequence, the update rate of the map will decrease, less ISEs will be discovered, and the overall performance of the plan-

ner will be ultimately affected. Therefore, for all these practical reasons, map accuracy should be balanced against communication cost.

The high resolution camera equipped on the robot allowed to reconstruct a ground truth, but also to gather pictures of the scene during the mission, to perform offline reconstruction via photogrammetry with the Metashape software. The results are depicted in Fig. 5.7. Some details of the scene are presented in the figure where obstacles such, as pillars and boxes, are finely modeled. The influence of light halo on the reconstruction is also certified on the offline reconstruction (red shades of the entrance (right part) of the parking on "C3"). The robot mission allows to compute a qualitative offline reconstruction compared to the GT, with an average signed error of 0.01 cm (M3C2 avg. error) and a standard deviation of 5.74 cm (M3C2 std. error) i.e a RMSE of 5.74 cm with 98.3% of surface coverage, considering a truncature distance for outlier points of ±17.3 cm.

5.6 Conclusion

In this chapter, we presented the real-world experiments realized to validate our 3D reconstructions architectures. The multi-robot architectures were initially designed for a fleet of UAVs, and during our experimental campaigns, they have been successfully adapted to ground robots with satisfactory results in terms average mesh error (for a given mapping method and resolution). Moreover, the multi-robot system is able to cover the majority of the scene. However, the surface coverage decreases as the number of robots increases. As the team grows, the sum of each robot's measurement and estimation errors increases. It concerns especially the visual-odometry drift which is intensified by other unmodeled disturbances, such as the poorly textured walls and luminosity variations, which have not been taken into account in the simulation experiments. A possible solution to the drift issue, would be to address the loop-closure problem, which has been largely treated in the SLAM literature. To have an even better coverage of the two 3D environments, a heterogeneous team of robots could have been considered as well, by joining the Wifibots and with multiple UAVs. However, time and safety constraints, and the limited availability of the test spaces and material at ONERA due to tight schedules, prevented us from validating our algorithms with real quadrotor UAVs. This is left as an interesting subject for future research.

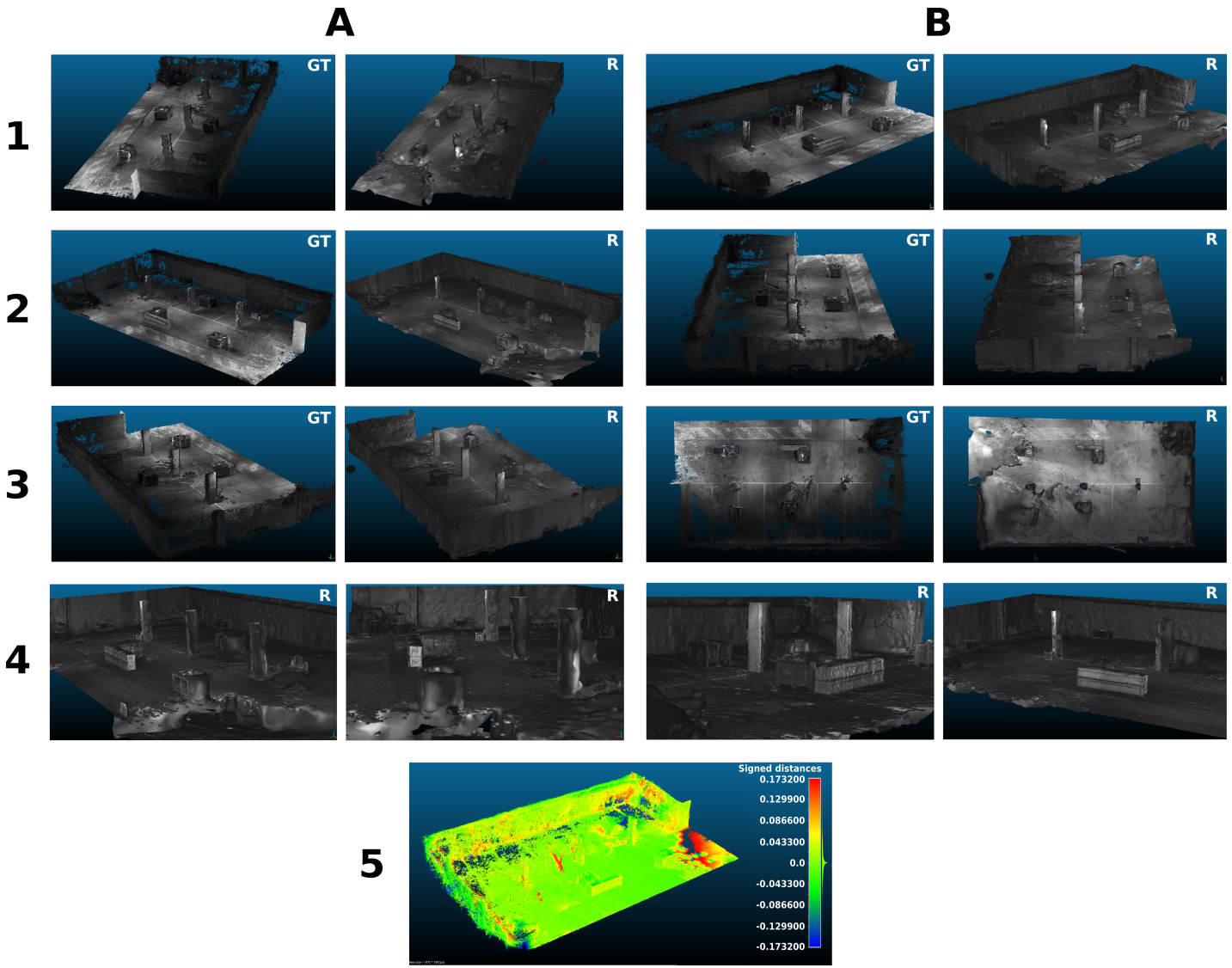


Figure 5.7: *Ground truth versus offline reconstruction of* ***Parking lot***. 6 pairs of views (from "A1" to "B3") of offline 3D reconstructions, performed with photogrammetry via Metashape: [GT] GT point cloud, [R] mesh from reconstruction performed by 2 robots with the distributed architecture. Line "4" is composed of multiple zooms of the mesh. Finally, the results of the computation error in line "5".

Conclusion

This work has been motivated in part by needs in digital cultural heritage, where the existing 3D reconstruction methods rely on fixed laser scanners positioned by hand, or monitored UAV equipped with camera, and on photogrammetry. The process remains slow, complex, expensive and risky for the operators, who have to take thousands of measurements with limited-range sensors, sometimes in dangerous areas, to ensure to completeness and consistency of the 3D model. In this thesis, we have proposed an original solution to the problem of autonomous reconstruction of an unknown environment, by deploying a team of cooperative robots equipped with forward-facing stereo cameras. In the literature, Next-Best-View methods use the knowledge of the environment to drive the reconstruction by either trying to explore a predefined volume (volume-exploration methods) or to fully cover the surface of an object (surface inspection methods). The latter reconstruct an object by generating sensor configurations directly from the knowledge of the current incomplete surface, represented by a point cloud or by meshes with multiple facets. The obtained 3D model is accurate but relies on strong navigation assumptions for the definition of the sensor configuration space limiting a direct exportation to mobile robot. The volume exploration techniques scan a whole environment volume represented by occupancy cells with two possible states: known or unknown. The volumetric mapping allows the robot to identify areas where it can navigate. These methods provide fast reconstruction of unknown environments, while ensuring safe navigation, but they do not always account for surface completeness of the model. A mixed approach has been developed in the thesis, by combining the advantages of volumetric mapping and surface-based planning for 3D reconstruction with multiple mobile robots.

The proposed method guides the robot towards a 3D mesh to reconstruct, based on their knowledge of the environment. From their initial pose, the robots reconstruct a first portion of the map using the current measurements. Volumetric mapping is instrumental in classifying the free and occupied space and in ensuring safe navigation, but it is also used to represent the surface. The implicit representation of the surface allows to identify discrete incomplete surface areas for the robot to visit and pursue the reconstruction.

The first problem considered in the thesis, is 3D reconstruction with a single robot. The embedded 3D sensor provides depth maps which serve as input to volumetric mapping. The viewpoint configurations are generated from the incomplete surface elements located at the frontier between the known and unknown surface. Every configuration is clustered according to its location in the space in order to discover highly-informative areas in the configuration space. The clusters are evaluated according to the information

gain they can provide. A roadmap for the robot is then defined by taking the relative distance between its actual pose and the viewpoints, and the information gain that a cluster can provide (defined as "utility"), into account. This single objective function is used to fix the sequence of viewpoints that allows to pursue the reconstruction. The robot plans its path by solving a variant of TSP over the constructed roadmap using the Lin-Kernighan heuristic. After going through the sequence of viewpoints, the path between clusters in the free space is computed with a sampling-based planner. The path is updated if what is left is no more relevant in terms of incomplete surface, and the reconstruction comes to an end, when all the incomplete surface has been covered. Chapter 2 has detailed the procedure and presented the results of numerical experiments with a UAV, for the reconstruction of two large-scale outdoor environments. A smoothing filter has been considered in the input depth map to simulate measurement noise. Our results indicate that the proposed strategy is competitive with the state-of-the-art methods in terms of completeness and accuracy, but the completion time remains too long for the limited flight endurance of real multi-rotor UAVs.

To address this issue, the single-robot formulation has been extended to a multi-robot setting, by jointly visiting the candidate viewpoints: in this way, the overall reconstruction time is minimized. This approach is described in Chapter 3. A centralized architecture has been designed to collect the data of the team and compute the paths locally before sending them to the robots. For mapping, the robots transmit their depth maps to the central unit, which integrates them incrementally into the occupancy volume with a GPU-based algorithm. The incomplete surface is extracted from the map, and the viewpoints are generated and clustered according to the single-robot procedure. The clusters are ranked according to their distance to a vehicle, and assigned greedily to the robots by solving a maximum assignment problem with a greedy heuristics: the "TSGA planner". Each cluster is tested by iteratively inserting it into the path of a robot and by solving a local TSP (similar to the single-robot planner), called a "TSP-insertion". The cluster is then allocated to the robot which guarantees the maximization of global (team level) utility. Once the paths have been assigned, they are broadcast to the team. Numerical simulations have been carried out in the same previous synthetic environments, and they have showed that the multi-robot system reduced the completion time compared to the single-robot case, without sacrificing accuracy and completeness. The numerical experiments reveal that the magnitude of the reconstruction error is on par with that of state-of-the art methods. However, the centralized approach possesses a single point of failure and a mission will end abruptly if the central unit crashes. Moreover, the GPU-based mapping algorithm is problematic if all computations are performed on the embedded computer with limited resources of a robot.

To increase robustness, a multi-robot architecture has been proposed in Chapter 4, where the robots share the computational load and share the results with the team. In particular, CPU-based distributed manifold mapping has been used and each robot computes a private map using the local information and shares it with its teammates. Each robot is able to estimate the global map, from which it can extract the missing areas and frontier, and generate and cluster the viewpoint configurations. The planning is distributed: in fact, the assignment problem is solved by each robot's "dist-TSGA planner" by using its local knowledge. The assigned paths are sent to the whole team and the low-level planner in charge of local path computation in the free space manages possible allocation conflicts or redundancies. Differently from the centralized architecture, special attention should be paid here to manage information bias among the robots, induced by the distrbuted mapping approach. Each agent reconstructs their own estimate of the map according to the exchanged data among the team, which can be altered by communication issues such as jitter or loss of information. Numerical experiments featuring a fleet of UAVs, have been performed on a supercomputer with multiple CPUs, to validate our approach. The performance of the distributed architecture is similar to that of the centralized one, in terms of completion time and traveled path, and provides also accurate models. Overall the simulation results are promising (although odometry drift has not been considered at this stage), and they encouraged us to implement our algorithms on real robotic platforms.

The three architectures have been adapted to wheeled robots to perform our hardware experiments, as described in Chapter 5. The pose of each robot has been estimated with a visual-inertial odometry algorithm. The algorithms have been tested in two indoor cluttered environments, a test arena and a parking lot. A high-resolution camera and a consumer-grade photogrammetry software, have been used to generate a ground truth of the scenes, which served as a reference to compute the reconstruction error. The algorithms exhibited good flexibility and adaptability: in fact, the kinematic models of the robots and the 3D environments, were significantly different from those considered in simulation. However, the experiments have shown that reconstruction completeness decreases (but still remains within acceptable limits) as the number of robots increases, especially in the centralized architecture for which larger teams have been tested. This behaviour can be explained by the natural odometry drift of the robots, a phenomenon that has been neglected in our numerical simulations.

Multiple objectives among those set by the "Scanbot" project have been treated. The usage of the online TSDF mapping integrates depth maps computed directly from sensor data to form a 3D model. The completeness of this model has been defined from the mapping technique with the ISE and constitute a stop criterion for the reconstruction. The

designed multi-robot systems guide and coordinate the team of robots for the scanning task of the unknown environment, while ensuring safe navigation. They are easily adaptable to any type of robot, for both homogeneous or heterogeneous team. The scanning processes allowed to reconstruct accurate online and offline 3D models. The results of the thesis are very promising and the proposed multi-robot architectures can serve as a basis for future research. In fact, thanks to the modularity of our architectures, each functional block can be replaced with another, implementing more advanced algorithms.

In the next section, we provide some closing remarks and discuss possible directions for future research.

Discussion and directions for future work

In practical terms, our qualitative results could be improved by using more accurate exteroceptive sensors such as LiDARs, and, more importantly, by addressing the odometry drift issue. The odometry drift has multiple causes, especially for an open-loop estimation algorithm which uses no feedback such as loop closure or fusion with an absolute reference. For instance, the (slow) variation of some parameters in the environment, such as luminosity can alter the perceived shades of grey, if the scene is over- or under-exposed and degrade the feature corners extraction or matching and thus shift the pose estimation. Another factor, more controllable, is the sensitiveness of the algorithm to the type of path: in fact, the more accurate the odometry, the smoother the path is. Indeed in our case, the paths are usually jagged and several back-and-forth motions or successive U-turns are likely to have perturbed the efficiency of eVO [Sanfourche *et al.* 2013], especially the Kanade-Lucas-Tomasi (KLT) feature tracker. Indeed, when the spatial location of a point changes abruptly on consecutive images, and to a greater spatial distance, optical flow algorithms would generally not perform well. To limit the drift, a direct solution would be to smooth out the path, or modify the planning part to integrate a single query algorithm such as RRT* where path smoothing is included in the planning process.

Another way to reduce the odometry drift during the exploration of large-scale environments where no global positioning information is available (such as Global Navigation Satellite Systems (GNSS)), would be to treat the loop-closure problem[6] [Williams *et al.* 2009]. This problem is well known in the simultaneous localization and mapping (SLAM) literature [Cadena *et al.* 2016] and consists in detecting when a robot has returned to a past location after having explored the environment for a certain period of time: pose graph optimization can then be applied to correct the past trajectory and the current pose. To limit the drift, the path of each robot should contain multiple sub-tours,

[6] An example of loop closure with LDSO, a monocular sparse visual SLAM [Gao *et al.* 2018] is available at: `https://www.youtube.com/watch?v=lThcTKYE7Es`

but this is not always possible in real missions. Another possibility is to leverage the poses and trajectories of the other robots, to refine the pose estimation, by using a Collaborative-SLAM (CSLAM) algorithms [Dubois *et al.* 2021, Lajoie *et al.* 2021]. CSLAM extends the single-robot SLAM architectures with a task and data allocation scheme (centralized, decentralized or distributed), an adapted communication policy and a matching and merging (or data association and fusion) strategy. In our case, the roadmaps computed by the TSGAs can be stored and adapted to the SLAM formulation, and serve as reference for the planner to force multiple loop closures according to a specific criterion, such as a threshold on the distance from the last closure. In [Vutetakis & Xiao 2019], the authors use the loops generated by solving a classical TSP over a set of viewpoints to form sub-tours. Another option would be to modify the utility function in order to include a weight on pose uncertainty, similarly to [Papachristos *et al.* 2017]. In this paper, informative NBV planning is coupled to a visual-inertial odometry algorithm and guarantees that a robot re-observes reliable landmarks (features which can easily be re-observed and distinguished in the environment, used to localize itself) and follows a trajectory that suitably excite the inertial sensors. The results of such a "closed-loop" method could be confronted to the proposed TSGA algorithms to study the impact on the reconstruction error.

On the robotic-planning side, possible improvements include either the design of a new path-finder algorithm, or the improvement of existing planning method by tailoiring it to meet some specific, application-oriented requirements. The literature on graph-based planning is extremely vast and state-of-the-art algorithms such as RRT or PRM (and their variants) are very efficient and competitive in many applications. On the other hand, NBV planning depends on the nature of the mission, and the planning objective should be designed according to the map representation. A simple solution to improve our method, would be to use ESDF mapping [Oleynikova *et al.* 2017] which does not suffer from the from parallax effect of TSDF mapping, and for which the ISE extraction process could be easily adapted. Some mapping representations generate continuous representation of the occupancy, which can be challenging for planning. With this technique, an incomplete areas could be identified as "hot zones" inside the 3D map and a potential fields planner could define a set preferable directions for the camera-robot. [Hitz *et al.* 2017] have proposed a new informative planning algorithm for online environment monitoring, which relies on splines over a continuous scalar field defined by Gaussian processes [Singh *et al.* 2010, Jadidi *et al.* 2017]. The path is thus optimized according to a continuous map, the steering constraints of the robots are satisfied, and smoother and more informative trajectories are obtained compared to the graph-based planners. Other improvements can be done on the objective function by considering the energy consumption or reconstruction time.

With the rise of *machine learning*, new mapping techniques have recently appeared, which rely on semantic segmentation information provided by state-of-the-art Deep Convolutional Neural Networks (DCNNs), in order to enrich the representation of the environment. The segmentation step subdivides a digital image into multiple segments (sets of pixels, also known as image objects) in order to simplify or make it meaningful for a specific purpose. The semantic representations typically extend metric representations such as images and grids with a semantic layer, and rely on segmentation methods [Crespo *et al.* 2020]. The semantic segmentation methods label each pixel of an image to a value belonging to one of different known classes (e.g. foreground vs background, or road vs sidewalk). State-of-the-art methods for image segmentation rely on deep neural network architectures [Ronneberger *et al.* 2015, He *et al.* 2017] that learn from large annotated datasets to regress from 2D images to object masks that encode the layout of input object's. Although the learning phase remains extremely resource intensive, these methods can be run on embedded systems. As far as RGB-D cameras are concerned, again the images can be semantic-segmented to identify objects in the scene and each pixel label can be projected into its corresponding voxel of the volumetric map using its associated depth value [Jadidi *et al.* 2017]. Given this extended representation of the space, an NBV planning can be performed to guide the exploration, as in [Zheng *et al.* 2019, Koch *et al.* 2019, Bartolomei *et al.* 2020, de Figueiredo *et al.* 2021]. In our digital heritage application, different classes can be defined according to the level of detail (or informativeness): as a result, a higher gain value will be associated with a statue or a bas-relief, than a blank wall.

The proposed distributed architecture could be improved in terms of network communication and data transfer. For example, one might assume that the number of information exchanges among the robots is limited and triggered by a certain a condition to reduce the communication bandwidth, as in the large event-triggered control literature [Heemels *et al.* 2012, Hardouin *et al.* 2019]. At a system level, the performance of the team of robots could be improved by considering heterogeneous vehicles: for example, wheeled robots could scan the ground level, and a fleet of UAVs, instead, is in charge of the fine details at a height. The TSGAs could be adapted to take the specific configuration space of each robot into account by segregating the set of clusters, and perform the greedy assignment. Another option is to relax the partition constraint of the assignment problem, and assign some clusters to multiple robots for different tasks in the same mission, with different utility values. For instance, a fast aerial robot may be asked to visit distant clusters and build a coarse prior navigation map for identifying possible areas of interest, in order to plan smoother paths for slower ground robots equipped with bulky LiDARs, in charge of creating a high-quality 3D model. Based on this synergy of a

robot team and to go further, the whole reconstruction scheme can be improved in order to perform a highly precise scan, by dividing it into multiple scanning steps. Usually, the high accuracy sensors used in these campaigns, such as LiDARs, requires stable structure provided by heavy robots, evolving at low speed, which are often high energy consumers. For instance, the proposed multi-robot systems could be set up for an initial gross reconstruction using small versatile robots and serve as a roadmap for the precise scans. Thus, the produced volumetric model could be kept for the navigation of the heavy robots, some clusters of viewpoints could be selected to ensure the maximum coverage and reused to plan optimal trajectories (in terms of energy consumption, path travelled, etc.) for the whole accurate scan of the building.

Résumé français

Chapter 6

Un système multi-robots déployable en centralisé ou distribué pour la reconstruction 3D d'environnements inconnus à large échelle

---

6.1 Introduction

Alors que la recherche sur les systèmes multi-robots se développe, les robots autonomes qui coopèrent entre eux sont de plus en plus utilisés pour accélérer certaines tâches, comme par exemple, la collecte d'échantillons dans les océans [Leonard *et al.* 2007], la recherche sur la faune [Shah *et al.* 2020], la cinématographie [Alcántara *et al.* 2021], l'automatisation d'opérations logistiques [Digani *et al.* 2015], ou l'exploration de mines souterraines. Ces avancées récentes, mettre en place une mission d'exploration et de reconstruction 3D coordonnées en ligne dans des environnements vastes, encombrés et inconnus (usines, grottes, sites archéologiques, champs de bataille) reste complexe et, à ce jour, peu de solutions existent dans la litterature.

Ce travail de thèse présente une nouvelle solution pour la reconstruction d'environnement 3D à l'aide d'un ou plusieurs robots, et traite le problème d'inspection de surface dans un environnement incconu, via une approche Next-Best-View (NBV) basée frontière. Une fonction-objectif qui considère explicitement la représentation de surface est utilisée pour planifier un chemin dans l'espace libre, dans le cas d'une architecture distribuée et centralisée.

Des simulations et expérimentations réelles ont montré que les architectures prosposées sont robustes aux incertitudes de mesures, produisent des modèles 3D complets, précis et avec des durées de reconstruction courtes.
Pour résumer les principales contributions de la méthode développée dans cette thèse :

- Une seule cartographie volumétrique en ligne est utilisée pour la navigation et la reconstruction.

- Une nouvelle formulation qui prend en compte la représentation de surface 3D, où la complétude du modèle est définie en temps que critère d'arrêt pour la reconstruction.
- Un planificateur NBV multi-robots est utilisé pour parcourir les points de vue. En particulier, l'allocation de configurations est réalisée avec une approche gloutonne et des résolutions successives du problème du voyageur de commerce (TSP), afin d'assurer la complétude du modèle reconstruit. Le planificateur peut être adapté à n'importe quel type de robot, pour des flottes hétérogènes ou homogènes.
- Une implémentation multi-robots pouvant être déployée selon une architecture centralisée ou distribuée pour la reconstruction 3D.
- Les méthodes proposées ont été validées grâce à des simulations pour des flottes de drones et des expérimentations réelles pour des robots roulants.

6.2 État de l'art

La planification de chemin informative en temps réel pour la reconstruction 3D d'environnements inconnus est un important sujet de recherche. La cartographie volumétrique permet de représenter l'espace libre et occupé, et permet d'estimer précisement la surface des objets de l'environnement, pour un ou plusieurs robots. En parallèle des techniques de cartographie volumétrique, de nouveaux planificateurs ont été développés permettant aux robots d'explorer rapidement des volumes inconnus et produire des modèles 3D grossiers. D'autres méthodes de reconstruction plus anciennes, dites "basée-surface", se focalisent sur la précision de reconstruction, où la fonction de coût tient compte d'un ou plusieurs critères surfaciques. Plus récemment, ces deux approches ont été couplées pour garder leurs avantages, mais seulement dans un cas mono-robot.

6.2.1 Cartographie volumétrique

La cartographie volumétrique consiste à discrétiser l'espace 3D en cellules : une cellule peut être inconnue, libre ou occupée. L'ensemble des cellules peut être utilisé afin de représenter la scène en 3D. Une approche en ligne de cartographie 3D, nommée OctoMap, a été introduite dans [Hornung *et al.* 2013]. Elle repose sur une grille d'occupation 3D ayant une structure interne de type "octree". Cette representation adapte le niveau de détail de la carte en fonction de l'environnement, réduit l'utilisation mémoire, permettant ainsi un fonctionnement temps réel sur CPU. Dans [Newcombe *et al.* 2011a], les auteurs proposent *KinectFusion* qui utilise les acquisitions d'une caméra RGB-D pour

calculer la fonction distance signée tronquée (TSDF) [Curless & Levoy 1996] sur une grille, afin de représenter implicitement la surface de l'environnement. Des améliorations successives ont permis de réduire l'utilisation mémoire [Nießner *et al.* 2013], et de rendre possible la reconstruction 3D sur des téléphones mobiles via la solution libre accès CHISEL [Klingensmith *et al.* 2015]. Dans [Oleynikova *et al.* 2017], cette méthode de cartographie a été revisitée en ajoutant une estimation de champs de distance euclidienne, qui améliore la précision globale de la carte. Le concept de carte multiple [Howard 2004] a été adopté dans le cadre de la TSDF, pour limiter les erreurs de cartographie dues à la dérive de la localisation [Millane *et al.* 2018], et la carte monolitique a été remplacée par une collection de sous-cartes locales (ou patchs). Récemment, dans [Duhautbout *et al.* 2019], les auteurs ont adapté cette idée au cas d'une architecture distribuée, pour plusieurs robots roulants.

6.2.2 Planification mono-robot

En se basant sur une carte volumétrique, un robot peut explorer un volume contenant des objets d'interêt. Les méthodes d'*exploration volumique* utilisent une représentation volumétrique comme OctoMap pour identifier les zones inconnues, libres et occupées. Les planificateurs par échantillonnage tels que les Rapidly-exploring Random Trees (RRT [LaValle & Kuffner Jr 2001], RRT* [Karaman & Frazzoli 2011]) sont utilisés pour calculer la trajectoire, en étendant incrémentalement un arbre constitué de poses échantillonées dans l'espace libre pour le capteur visuel. Un autre planificateur, le Pobabilistic RoadMap (PRM [Kavraki *et al.* 1996], LazyPRM* [Hauser 2015]) extrait le chemin à partir d'un graphe formé par des poses échantillonnées aléatoirement entre la configuration de départ et celle d'arrivée, en optimisant une fonction objectif. Dans le cadre de la planification NBV, l'exploration volumique est une méthode de planification informative qui garantit la couverture maximale du volume à cartographier (voir [Bircher *et al.* 2016, Papachristos *et al.* 2019, Selin *et al.* 2019, Batinovic *et al.* 2021, Respall *et al.* 2021] pour les méthodes utilisant un RRT ou [Xu *et al.* 2021] pour le PRM). La couverture de la trajectoire est évaluée par prolongement des rayons émanants du capteur dans les voxels de la carte (le lancer de rayon [Bresenham 1965]). Le volume est exploré au fur et à mesure que le robot suit son chemin. Ces méthodes rapides permettent de reconstruire des modèles suffisants pour la navigation, mais ne prennent pas en compte explicitement la complétude et la qualité de la surface reconstruite.

Les méthodes d'*inspection de surface* utilisent la connaissance partielle de la surface pour générer des poses candidates qui vont permettre de poursuivre la recon-

struction. Les méthodes NBV déterminent la "prochaine meilleure vue" à visiter en fonction de la mission à réaliser. Parmi ces méthodes, les approches basées-frontière génèrent des configurations qui orientent le capteur, en fonction de ses caractéristiques, vers la frontière avec la surface connue, représentée par un maillage. En visitant ces points de vue, le robot collecte de nouvelles informations sur la surface, tout en assurant une certaine couverture avec le modèle existant, et donc la continuité de la surface [Connolly 1985, Vasquez-Gomez *et al.* 2014b, Border *et al.* 2018]. La plupart de ces approches concernent la reconstruction de petits objets par des caméras montées sur des préhenseurs ou bras articulés, et reposent sur de fortes hypothèses sur l'espace atteignable par le capteur. Ces dernières années, ces méthodes NBV ont été étendues au cas des robots mobiles en utilisant des représentations volumétriques [Vasquez-Gomez *et al.* 2014b, Yoder & Scherer 2016] et TSDF [Monica & Aleotti 2018, Schmid *et al.* 2020].

Récemment, des méthodes hybrides ont été prosposées afin d'améliorer la précision des modèles et d'accélérer la reconstruction en prenant les avantages des méthodes d'inspection de surface et d'exploration de volume. Dans [Bircher *et al.* 2018], la procédure est divisée en deux étapes : d'abord, un modèle grossier de l'environnement est reconstruit, ensuite, la surface est affinée en planifiant une trajectoire grâce à l'à priori du modèle grossier. En utilisant des cartes d'occupation, l'algorithme de [Song & Jo 2017] permet au robot de couvrir tout l'espace par une approche volumétrique classique. Dans leur extension [Song & Jo 2018], le chemin est affiné en prenant en compte la complétude de la surface. Plus récemment, la méthode a été améliorée et validée grâce à des expérimentations réelles [Song *et al.* 2021]. Cependant, ces méthodes dépendent de plusieurs représentations, utilisées en simultané, qui peuvent être coûteuses à produire et à exploiter. De plus, seulement quelques unes d'entre elles résolvent le problème d'inspection explicitement (voir [Schmid *et al.* 2020]).

6.2.3 Planification multi-robots

Comme présenté au dessus, la majorité des études sur la reconstruction en ligne concerne un seul robot, et la coopération multi-robots pour ce type de tâche reste un problème ouvert [Amigoni & Gallo 2005]. En considérant des robots équipés de lasers dans un environnement 2D, les auteurs de [Burgard *et al.* 2005] ont été les premiers à extraire les cellules-frontière et à modéliser l'objectif comme un compromis entre l'utilité et la distance à parcourir afin d'accomplir des tâches multi-robots. Dans [Mannucci *et al.* 2017], une flotte de robots aériens est considérée pour l'exploration d'un environnement extérieur, et les auteurs proposent une méthode de reconstruction coopérative basée-frontière. La cartographie OctoMap et la planification de trajec-

toires multi-UAV (basée RRT*) sont calculées de manière centralisée sur la station-sol, et l'approche a été évaluée par des simulations sur ROS-Gazebo. D'autres méthodes d'exploration basées sur des cartes d'occupation probabilistes avec réduction d'entropie, comme decMCTS [Best *et al.* 2016] ou SGA [Atanasov *et al.* 2015], utilisent la notion d'information mutuelle des capteurs de distance [Charrow *et al.* 2015]. Dans [Corah *et al.* 2019, Corah & Michael 2019], les auteurs ont proposé un planificateur décentralisé, Distributed Sequential Greedy Assignment (DSGA), qui repose sur une recherche arborescente Monte-Carlo (MCTS) [Chaslot 2010]. Les chemins des robots sont générés en résolvant un problème de maximisation de fonction sous-modulaire avec des heuristiques d'assignation gloutonnes. Enfin, dans [Corah & Michael 2021] la méthode a été adaptée pour l'exploration, et consiste à maximiser une fonction objectif volumétrique.

À partir de cette revue de l'état de l'art, on peut constater qu'une large majorité des recherches traite un problème d'exploration volumique et ne considère pas explicitement la complétude surfacique dans la planification. De plus, même si le problème d'inspection est de plus en plus traité dans la littérature, d'après nos connaissances, aucune formulation multi-robots n'a été proposée à ce jour. Nos travaux de thèse ont contribué à étendre la recherche dans ce domaine.

6.3 Formulation du problème

Soit une flotte de N robots mobiles. On definit $\mathbf{q}^i \in \mathrm{SE}(m)$ la pose d'un robot i et $\mathbf{q}_0^i$ sa pose initiale, où $i \in \{1, 2, \ldots, N\}$: $m = 2$ si c'est un robot terrestre et $m = 3$ si c'est un robot aérien. On suppose que tous les robots sont équipés d'un système de localisation pour estimer leur pose par rapport au repère de référence global, et d'un algorithme de suivi de trajectoire robuste. Chaque robot est équipé sur l'avant d'un capteur de profondeur ayant un champs de vue ainsi qu'une portée limités, dont les poses sont connues par rapport au repère du robot. Les robots doivent reconstruire coopérativement un environnement 3D inconnu, comme par exemple un bâtiment, caractérisé par sa surface. Un algorithme de cartographie est requis pour reconstruire la surface et identifier l'espace libre pour la navigation. On considère une méthode de cartographie volumétrique permettant de construire une carte M constituée de plusieurs petits éléments d'espace 3D discrets. Ces éléments appelés voxels $\mathbf{v} \in M$, peuvent représenter un espace inconnu, occupé ou libre. Soit $X \subset M$ l'ensemble des voxels inconnus et $A \subset M$ l'ensemble des voxels connus, tel que $X \cap A = \emptyset$. De plus, soit $O \subset A$ l'ensemble des voxels occupés, et $E \subset A$ l'ensemble des voxels libres. L'objectif du processus de reconstruction est de scanner les voxels inconnus. Un voxel est scanné et devient connu (occupé ou vide) lorsqu'au moins

un agent l'a observé. L'incomplétude surfacique est définie de la façon suivante :

Definition 5 (**Élément de surface incomplet**). *Un Élément de Surface Incomplet (ISE) est un voxel* $\mathbf{v} \in M$ *situé sur la surface, à la frontière entre l'espace inconnu et l'espace libre et* C *l'ensemble des ISEs. Un voxel* $\mathbf{v} \in C$ *si et seulement si*

$$\begin{aligned} &a)\ \mathbf{v} \in E, && (\textit{libre}) \\ &b)\ \exists\, \mathbf{u} \in \mathcal{N}_{\mathbf{v}}^{6}\ \ s.t.\ \ \mathbf{u} \in X, && (\textit{inconnu}) \\ &c)\ \exists\, \mathbf{o} \in \mathcal{N}_{\mathbf{v}}^{18}\ \ s.t.\ \ \mathbf{o} \in O, && (\textit{occupé}) \end{aligned}$$

Où $\mathcal{N}_{\mathbf{v}}^{6}$ *et* $\mathcal{N}_{\mathbf{v}}^{18}$ *représentent le voisinage des 6 et 18 voxels connectés de* $\mathbf{v}$*, respectivement.*

Definition 6 (**Surface incomplète restante**). *Soit* Q *l'ensemble des configurations dans l'espace libre* $\mathbf{q}^i$ *d'un agent* i*, et* $Q_c \subseteq Q$ *l'ensemble des configurations qui permettent de scanner au moins un ISE* $\mathbf{v} \in C$*. On définit la surface incomplète restante :*

$$C_{\text{rem}} = \bigcup_{\mathbf{v} \in C} \left\{ \mathbf{v} \mid Q_c = \emptyset \right\}.$$

On utilise la fonction $\mathbf{p}^i_{j,k}(s) : [0,\, 1] \rightarrow \text{SE}(m)$, avec $m \in \{2,\, 3\}$ pour définir le chemin du robot i, partant de la configuration j jusqu'à la configuration k, où $\mathbf{p}^i_{j,k}(0) = \mathbf{q}^i_j$ et $\mathbf{p}^i_{j,k}(1) = \mathbf{q}^i_k$, $i \in \{1, 2, \ldots, N\}$. On suppose que $\mathbf{p}^i_{j,k}(s)$ est réalisable et ne provoque pas de collision pour l'agent i (i.e. les contraintes cinématiques et dynamiques du véhicule sont satisfaites tout au long du trajet). Le problème d'inspection peut être formalisé comme suit :

Problem 4 (**Multi-agent inspection problem**). *On considère une équipe de* N *robots avec leurs conditions initiales* $\mathbf{q}^i_0 \in Q,\ i \in \{1, 2, \ldots, N\}$*. Le problème d'inspection multi-robots requiert de trouver les chemins successifs à suivre* $\mathbf{P}^i_{0,f_i} = \{\mathbf{p}^i_{0,1}, \ldots, \mathbf{p}^i_{f_i-1,f_i}\}$ *avec les chemins dans l'espace libre* $\mathbf{p}^i_{k-1,\,k}(s)$ *visitant les poses* $\mathbf{q}^i_k$*,* $k \in \{1, 2, \ldots, f_i\}$*, qui permettent aux robots de scanner l'ensemble* $C_{\text{ins}} = C \setminus C_{\text{rem}}$ *de tous les ISEs contenus dans la carte reconstruite courante* M*.*

En progressant le long de leurs chemins $\mathbf{P}^i_{0,f_i},\ i \in \{1, 2, \ldots, N\}$, les robots sont capables de compléter l'espace inconnu, de découvrir de nouveaux ISEs et de résoudre itérativement le problème d'inspection jusqu'à ce que $C_{\text{ins}} = \emptyset$. Basé sur ces notions, un aperçu des contributions de la thèse est présenté.

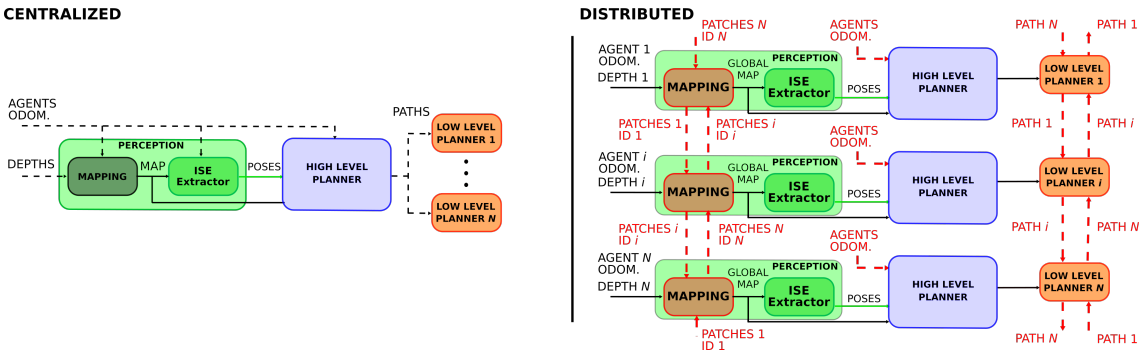


Figure 6.1: *Schémas des architectures multi-robots* : [gauche] centralisée, [droite] distribuée. Les structures internes des modules de perception et planification sont représentées à l'intérieur des bulles vertes et bleues, et les communications intra- et inter- blocs sont représentées en traits pleins et pointillés, respectivement.

6.4 Aperçu des contributions

Dans cette thèse, une nouvelle méthode générique multi-robots pour la reconstruction 3D de surface dans des environnements inconnus est présentée. Elle peut être déclinée selon une architecture centralisée ou distribuée. Ces architectures représentées sur la Fig. 6.1, sont adaptées pour l'utilisation de flottes de robots terrestres et/ou aériens. Ces implémentations peuvent se diviser en deux parties distinctes : un module de perception (blocs verts dans la Fig. 6.1) en charge de l'extraction des ISEs à partir d'une carte volumétrique reconstruite en ligne, et un module de planification (blocs bleus et oranges) pour le calcul des chemins et pour assurer la visite sûre des points-de-vue.

Un algorithme de cartographie constitue l'entrée du module de perception. Il utilise les cartes de profondeur, produites à partir des données des capteurs embarqués (capteurs RGB-D, banc stéréoscopique, etc.), appairées avec leurs poses associées, et les intègre dans une carte 3D volumétrique qui est utilisée, à la fois, pour la reconstruction (i.e. extraction of ISEs) et pour la navigation (i.e. planification de chemin dans l'espace libre). La carte est donc utilisée tout au long de la mission pour extraire les ISEs, élément nécéssaire à la planification. La représentation TSDF a été choisie parce qu'elle permet de représenter implicitement les surfaces. Une fois les ISEs extraits, des prises de vue sont générées pour les scanner efficacement, en fonction de l'environnement et des propriétés du capteur (portée, résolution).

D'autres méthodes de cartographie volumétrique peuvent également être utilisées. Dans l'architecture centralisée, les données d'entrée (cartes de profondeur et poses des robots) sont intégrées directement de façon incrémentale dans une carte commune sur une station-sol. Dans l'architecture distribuée, chaque robot calcule sa propre carte locale basée sur, d'une part, ses propres informations disponibles, et d'autre part, le partage de

carte à travers un processus distribué (similaire à [Howard 2004]). Les robots intègrent les données dans leur carte locale appelée patch, identifié par un index, jusqu'à ce qu'un évènement déclencheur se produise. Cet évènement peut être, l'intégration d'un certain nombre de cartes de profondeur dans le patch, ou la distance totale visitée pendant la construction du patch. Un patch terminé est stocké et diffusé aux autres robots, et un nouveau patch local est créé afin de poursuivre la recontruction. Chaque robot reconstitue une carte globale en collectant et fusionnant tous les patchs reçus.

Le module de planification coordonne les robots afin qu'ils effectuent la reconstruction 3D de la surface. À partir des configurations générées par le module de perception, la visite des configurations est planifiée grâce à une approche basée sur la résolution du TSP. Le planificateur TSP-Greedy Allocation (TSGA) groupe des ensembles de configuration en fonction de leurs localisations dans l'espace, afin d'identifier et classer des zones d'intérêt dans la carte incomplète. Ensuite, un graphe dirigé est généré, et réprésente l'utilité de visiter un de ces groupes, en fonction du type de robot (terrestre ou aérien). Des chemins dans l'espace libre sont extraits à partir de ce graphe, afin de maximiser le cumul d'utilité pour la flotte, et sont assignés aux robots. Quand la carte TSDF est actualisée, certains ISEs sont complétés, d'autres sont découverts. Le chemin est mis à jour si le nombre d'ISEs restants à scanner est faible par rapport au nombre d'ISEs venant d'être découverts. La reconstruction s'arrête quand il ne reste plus d'ISEs. Les chemins "haut-niveau" sont ensuite envoyés aux planificateurs "bas-niveau". Ces derniers génèrent des chemins pour les robots dans l'espace libre (via des planificateurs basés-échantillonnage), et collectent les poses de la flotte afin d'éviter les collisions. Dans l'architecture distribuée, les inconsistances de la carte globale, dues aux éventuelles pertes de patchs ou à des délais de communication, peuvent engendrer des assignations de configurations à plusieurs robots. Dans le but de pallier ce problème, le planificateur bas-niveau vérifie la consistance entre les allocations individuelles et collectives, et si besoin, attend une mise à jour du chemin. Enfin, le suivi de chemin est réalisé à l'aide d'un contrôleur de type Model Predictive Control (MPC).

6.5 Simulations

Les architectures centralisées et distribuées ont été testées et comparées en simulation pour la reconstruction de deux environnements réalistes. Ces environnements sont des modèles à large échelle **Powerplant**[1] ($65 \times 42 \times 15\,\mathrm{m}^3$) et Statue de la Liberté[2] (**SoL**, $20 \times 20 \times 60\,\mathrm{m}^3$). Ils contiennent, à la fois, des plans larges et étendus, des passages étroits

[1] http://models.gazebosim.org/
[2] https://free3D.com/

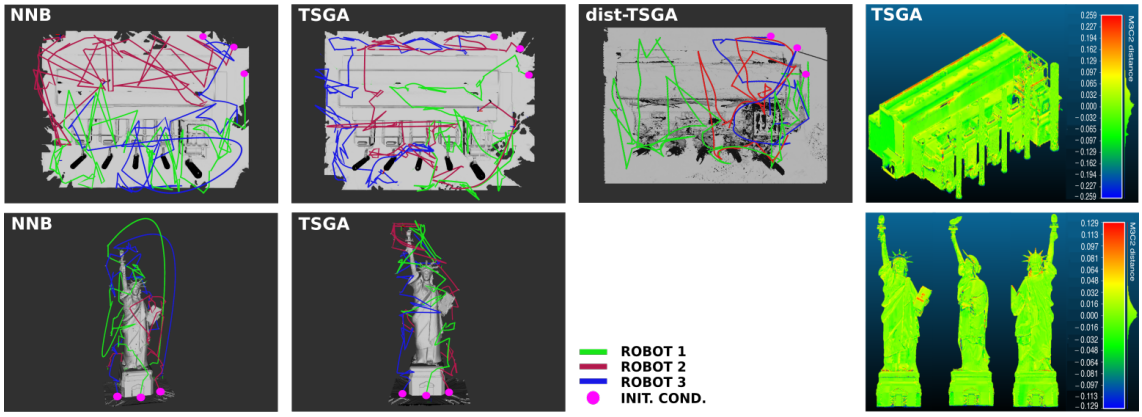


Figure 6.2: *Résultats des simulations* : [haut] **Powerplant** et [bas] **SoL**. Maillages 3D reconstruits et chemins d'exploration $\mathbf{p}^1_{0,f}$, $\mathbf{p}^2_{0,f}$, $\mathbf{p}^3_{0,f}$ (vert, rouge, bleu) des trois drones obtenus avec : [1ère colonne] planificateur NNB ; [2ème colonne] planificateur TSGA ; [3ème colonne] planificateur dist-TSGA. Les poses initiales des drones sont marquées avec des points magenta. [4ème colonne] erreur en distance signée pour le planificateur TSGA : la barre de couleurs représente l'erreur par rapport au modèle de référence, en mètres, calculée avec le plugin M3C2 de CloudCompare.

et des détails fins. Les chaînes logicielles ont été implémentées sur ROS et testées sur Gazebo pour des flottes de 1, 3 et 5 drones équipés de caméras stéréoscopiques. Chaque drone a 4 degrés de liberté : sa position 3D et son angle de lacet. Un aperçu des résultats des simulations est représenté en Fig. 6.2.

L'algorithme a d'abord été testé dans un cas mono-robot de l'architecture centralisée. Celui-ci montre des performances d'un ordre de grandeur similaire aux méthodes de l'état de l'art, en terme de qualité et complétude de reconstruction, de distance parcourue et de temps de reconstruction. Cependant, certaines reconstructions ont des durées très longues pour l'autonomie actuelle des drones. Ainsi, l'utilisation de plusieurs robots simultanément est justifiée dans le but de réduire les temps de reconstruction.

Les tests réalisés pour des flottes de 3 et 5 robots montrent que l'augmentation du nombre de robots permet de réduire drastiquement le temps de reconstruction, et la distance parcourue par chaque robot. Le planificateur TSGA a également été comparé à un algorithme des plus proches voisins (NNB) afin de confronter deux approches gloutonnes. Le TSGA montre de meilleures performances en terme de navigation (temps, distance) que le NNB, malgré sa très grande réactivité aux mises à jour de la carte et sa rapidité de calcul. Cependant, la centralisation des calculs limite la robustesse du système multi-robots.

Enfin, une version distribuée de l'architecture, le dist-TSGA, a été également testée dans l'environnement **Powerplant** afin de comparer les approches multi-robots. Les simulations montrent des résultats similaires en termes de temps de reconstruction, de dis-

tance parcourue et de qualité de reconstruction, par rapport à ceux de l'architecture centralisée. Le système distribué est plus robuste. Cependant, les données échangées entre les robots et la station-sol sont importantes.

6.6 Expérimentations en environnements réels

Des expérimentations réelles ont été réalisées pour poursuivre la validation des algorithmes. Les robots utilisés sont des robots terrestres de type Wifibot, équipés d'un banc stéréoscopique constitué de deux caméras monochromatiques. Deux environnements intérieurs ont été considérés : une **Salle de test** de $8 \times 7 \times 2$ m^3 constituée de 4 murs habillés de matelas et d'un obstacle central, et un **Parking** souterrain de $21 \times 14 \times 2$ m^3 contenant 3 piliers et des obstacles au niveau du sol. Pour quantifier l'erreur et la complétude des résultats de reconstruction, une vérité terrain des environnements a été réalisée par photogrammetrie. Les résultats de reconstruction du **Parking** sont présentés dans la Fig. 6.3 et Fig. 6.4.

En termes logiciels, les architectures multi-robots ont été conservées et adaptées au cas des robots mobiles terrestres. L'estimation de profondeur est réalisée avec l'algorithme ELAS [Geiger *et al.* 2010] et l'estimation de pose est calculée par l'algorithme d'odométrie visuelle eVO [Sanfourche *et al.* 2013]. L'architecture central-

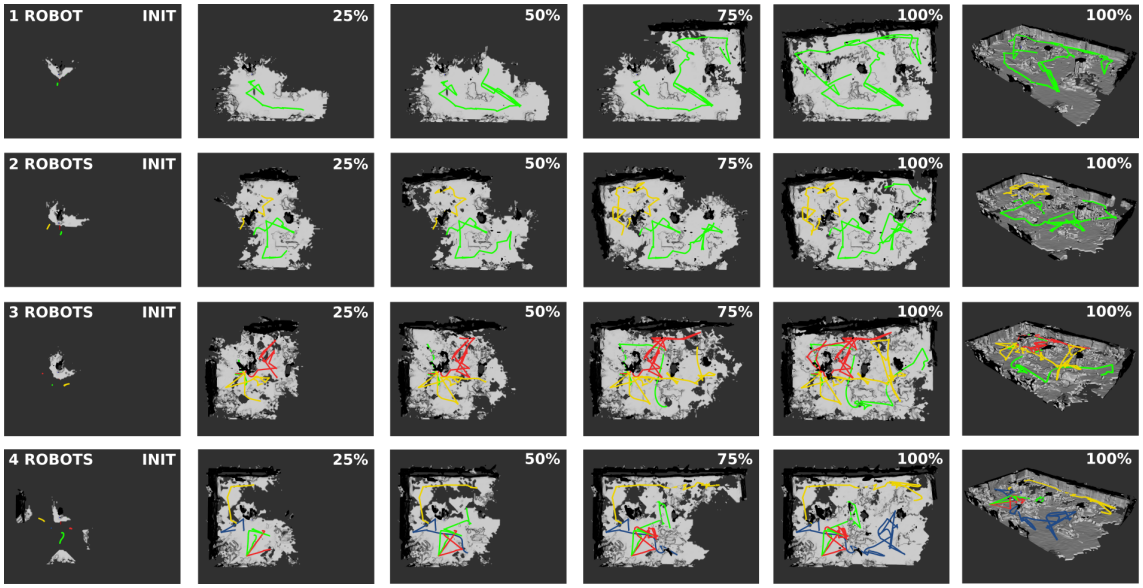


Figure 6.3: *Expérimentations réelles :* progression de la reconstruction du **Parking**. De gauche à droite, la progression de la reconstruction de surface en pourcentage, obtenue avec l'architecture centralisée (vue de dessus). Par ligne, le nombre de robots varie entre 1 et 4. La dernière colonne montre une vue isométrique de la reconstruction finale et des chemins empruntés par les robots. Clip vidéo : `https://youtu.be/yJ4lEyKrPGA`

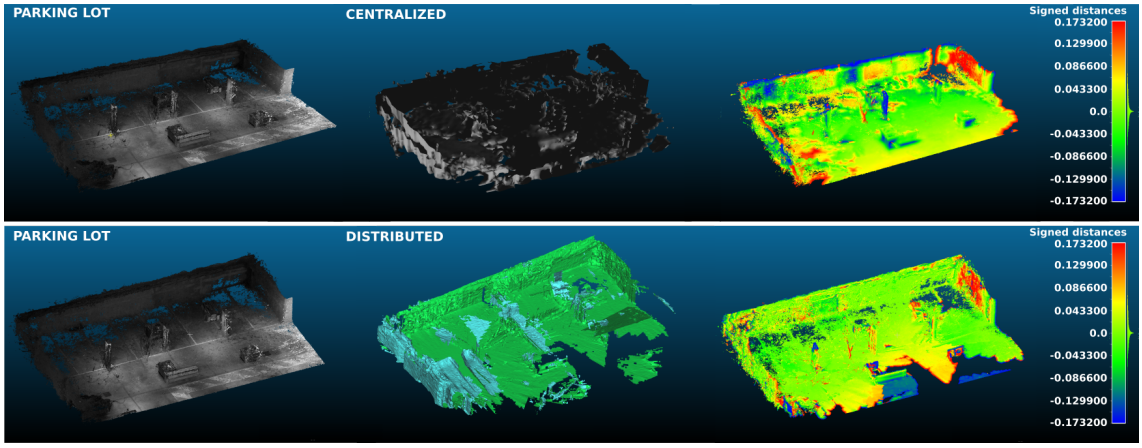


Figure 6.4: *Reconstruction 3D du **Parking** avec deux robots*, [haut] architecture centralisée, et [bas] architecture distribuée. [gauche] nuage de points de la verité terrain. [centre] Maillage reconstruit. [droite] erreur en distance signée du modèle reconstruit : la barre de couleurs représente l'erreur par rapport à la vérité terrain, en mètres, calculée avec le plugin M3C2 de CloudCompare

isée a été testée pour des équipes de 1, 2, 3 et 4 robots et l'architecture distribuée pour une équipe de 2 robots.

Les résultats des expérimentations réelles montrent un comportement similaire à celui observé dans les simulations. Plus le nombre de robots augmente et plus la reconstruction est rapide. Cela, au détriment de la qualité et de la complétude générale du modèle. Puisque qu'aucun recalage de trajectoire n'est réalisé, la dérive naturelle de la pose se propage sur les modules de l'architecture qui ont recours à la pose, notamment la cartographie. Ce phénomène, parmi d'autres, est amplifié par l'augmentation du nombre de robots, qui engendre une accumulation de ces dérives odométriques. Ces expérimentations ont permis de confirmer que ces architectures sont exploitables pour réaliser des reconstructions complètes (entre 73% et 90% de couverture) et relativement précises (environ 9 cm d'erreur moyenne). De plus, ces travaux laissent entrevoir de multiples perspectives de recherche.

6.7 Conclusion

L'objectif de cette thèse est de mettre au point des méthodes de reconstruction 3D d'environnements inconnus, à l'aide de plusieurs robots mobiles terrestre et aériens. Les robots, équipés de caméras stéréoscopiques, explorent l'environnement et révèlent, dans le modèle progressivement reconstruit, des éléments de surface incomplets. Des poses candidates sont alors générées pour les compléter, groupées en zones d'intérêt et sont ensuite assignées aux robots à l'aide d'un algorithme glouton. Cet algorithme est un planifi-

cateur de chemin de type Next-Best-View, basé sur un critère surfacique, où le problème du voyageur de commerce est résolu itérativement. Un planificateur probabiliste permet enfin aux robots de trouver leur chemin dans l'espace libre. La méthode a été déclinée en trois architectures : mono-robot dans un cadre préliminaire, multi-robots centralisée pour réduire les temps de reconstruction, puis distribuée pour en augmenter la robustesse. Une validation par simulation et des tests en environnement réel ont été réalisés, pour des robots aériens et roulants, afin de montrer l'efficacité des trois architectures. Les résultats de simulation indiquent que le système multi-robots est compétitif par rapport aux solutions existantes. Les expérimentations réelles montrent des résultats et des tendances similaires à ceux observés en simulation, et confirment que la méthode est facilement adaptable à différents types de robots ou d'environnements pour reconstruire des modèles 3D complets en temps-réel.

Appendices

Appendix A

Kinematics and robotics

Appendix outline

A.1 Pose of a rigid body 129

A.1.1 Rotations matrices 129

A.1.2 Rigid-body motion 130

A.2 Mobile robotic platform 131

A.2.1 Robot parametrization 131

A.2.2 Stereo vision 131

A.2.3 Camera and IMU calibration 135

A.2.4 ROS: the Robot Operating System 136

This appendix presents the mathematical background and robotic notions used in the thesis. The Section A.1 introduces rotation and rigid-body motion in the 3D space. The Section A.2 sets the robot parametrization, presents briefly the generic functioning of vision systems used for vision-based navigation of a mobile robot and 3D reconstruction, and the ROS middleware used to carry out our experiments.

A.1 Pose of a rigid body

To parametrize the robot and represent the structure that they observe, we use basic geometry tools. In this section, 3D rotation and affine Euclidian transforms are presented.

A.1.1 Rotations matrices

The rotations in the 3D space are defined as orthogonal automorphisms of $\mathbb{R}^3$ of determinant equal to 1. They form the *Special Orthogonal Group* of order 3 or SO(3), for short. A rotation matrix $\boldsymbol{R}$ can be expressed:

$$\boldsymbol{R} = \boldsymbol{R}_z(\psi)\,\boldsymbol{R}_y(\theta)\,\boldsymbol{R}_x(\phi) \tag{A.1}$$

where

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{A.3}$$

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{A.4}$$

are the elementary rotation matrices about the z, y and x axes, respectively. The Euler angles can be defined by elementary geometry or by composition of rotations. The geometrical definition demonstrates that the composition of three elementary rotations is always sufficient to reach any target frame. The three elementary rotations may be extrinsic (rotations about the x, y and z axes of the original coordinate system, which is assumed to remain motionless), or intrinsic (rotations about the axes of the rotating coordinate system, solidary with the moving body, which changes its orientation after each elementary rotation). There exist multiple sets of rotation axes to define the Euler angles, or different names for the same angles. Therefore, any equation employing Euler angles should always be preceded by a clear definition. In this work, we refer to the z-y-x Tait-Bryan convention defined above. Other representations of the 3D orientation of a rigid body exist, such as quaternions which allow to suppress the singularities of the Euler angles. For more details, refer to [Siegwart *et al.* 2011].

A.1.2 Rigid-body motion

The pose of a rigid object, defined by its position and orientation in inertial frame, can be represented by an affine transformation. The set of all these transformations forms the *Special Euclidian Group* or SE(3), for short, which has dimension 6. Unfortunately, the affine property makes difficult the successive compositions of rigid transformations. To address this issue, homogeneous coordinates are used instead. The idea is to represent the rigid motion by means of 4×4 matrices, in order to treat the translation with a linear formalism. Let $\boldsymbol{T}_{AB} \in \mathrm{SE}(3)$ be represented by the following homogeneous matrix,

$$\boldsymbol{T}_{AB} = \begin{bmatrix} \boldsymbol{R}_{AB} & \boldsymbol{t}_{AB} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{A.5}$$

where the vector $\boldsymbol{t}_{AB}$ and the matrix $\boldsymbol{R}_{AB}$ denote the translation and rotation applied to frame B to obtain frame A. This allows to represent the composition and inverse of the affine transform, with their action on the group of 3D vectors, in a linear context with matrix product and inversion. Given the three frames A, B and C, the following equations hold true: $\boldsymbol{T}_{AC} = \boldsymbol{T}_{AB}\boldsymbol{T}_{BC}$ and $\boldsymbol{T}_{BA} = \boldsymbol{T}_{AB}^{-1}$. The explicit formula for the inverse of a homogeneous matrix is:

$$\boldsymbol{T}_{BA} = \boldsymbol{T}_{AB}^{-1} = \begin{bmatrix} \boldsymbol{R}_{AB}^{\intercal} & -\boldsymbol{R}_{AB}^{\intercal} \cdot \boldsymbol{t}_{AB} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{A.6}$$

Let $\widetilde{\mathbf{x}} = (\mathbf{x}, 1)^{\intercal}$ denote the the augmented coordinates of a point $\mathbf{x}$. The application of the affine transform on the vector $\mathbf{x}$, results in the matrix product of their homogeneous representations, that is:

$$\boldsymbol{T}\widetilde{\mathbf{x}} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}\,\mathbf{x} + \boldsymbol{t} \\ 1 \end{bmatrix} \in \mathbb{R}^4 \tag{A.7}$$

A.2 Mobile robotic platform

This section presents robot parametrization, recalls some basic notions on multiple-view geometry and describes the calibration process of cameras and inertial measurement unit (IMU) used in this thesis.

A.2.1 Robot parametrization

The world frame $(x, y, z) \in \mathbb{R}^3$ is defined as reference. By convention, the frame of a robot has its origin at the center of gravity, the x_r-axis is oriented forward, the y_r-axis is oriented on the left and the z_r-axis points upwards. The orientation of the frame is represented by the yaw angle ψ, the roll angle ϕ, and the pitch angle θ defined in previous Sect. A.1.1. In this work we consider vision as principal sensor. For a camera, or an optical system, the body-fixed frame has its center at the optical center, with the z_c-axis pointing forward, the x_c-axis pointing on the right, the y_c-axis pointing downward, by convention. The frames are represented in Fig. A.1.

A.2.2 Stereo vision

Single camera geometry Among exteroceptive sensors, camera is nowadays one of the most used for robotic applications. Although there exist various types of camera, we will focus here on central projection based cameras, also known as “pinhole” cameras. They

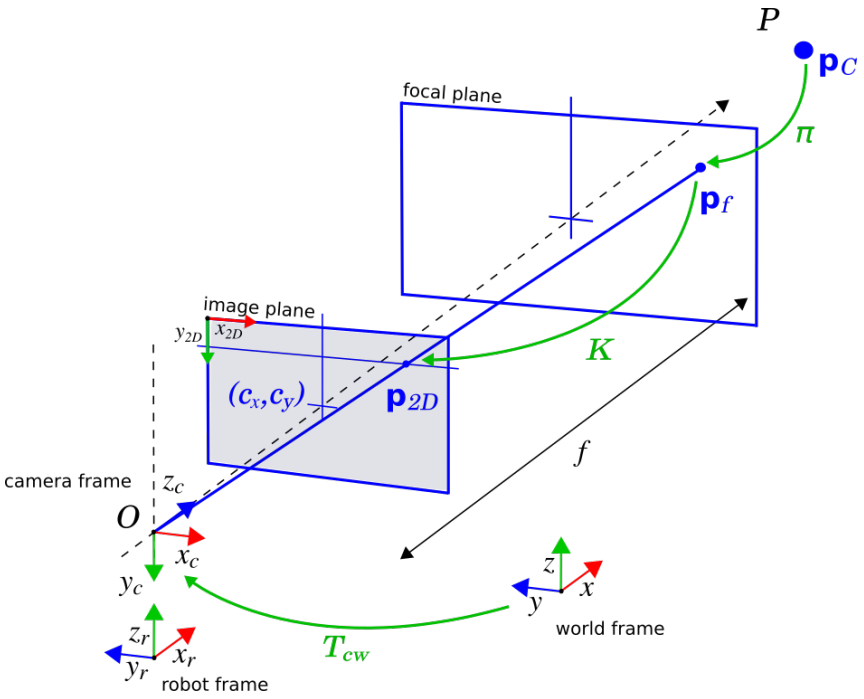


Figure A.1: *Scheme of a camera (mounted on the robot).*

are widely used because of their simplicity and their reduced costs. These sensors produce digital images, i.e. regular matrix of pixels, where each pixel represents the level of light perceived during a small amount of time called exposure duration. This level of light is quantified in order to be represented by fixed precision digital format (8 bits encoding by channel is the most common). In addition, color cameras are constituted of a Bayer filter located in front of the image plane in order to filter the red, green, blue channel. With this filter, each channel observes only one third of the scene but holes are reconstructed by advanced interpolation methods.

For robotic navigation applications, cameras are useful because they provide information on the geometry of the environment. Indeed, a pinhole camera can be modeled as a relationship that link the coordinate of a 3D point P of the space $\mathbf{p}_{3D} = (x, y, z)^\intercal$ to its projection into the image plane $\mathbf{p}_{2D} = (u, v)^\intercal$ that can be seen in the output image. The projections and a camera scheme are depicted in Fig. A.1. To express this projection, let us define the sensor frame, located at the optical center O and aligned in the following way, the z_c axis is co-linear with the optical axis, and axis x_c and y_c are co-planar with the image plane. Let us define $\boldsymbol{T}_{CW}$ the rigid-body motion (extrinsic transform) from the world frame to the sensor frame, such that the P coordinates expressed in the sensor frame are:

$$\widetilde{\mathbf{p}}_C = \boldsymbol{T}_{CW}\, \widetilde{\mathbf{p}}_{3D} \tag{A.8}$$

where $\widetilde{\mathbf{p}}_C = (\mathbf{p}_C,\ 1)^\intercal$ and $\widetilde{\mathbf{p}}_{3D} = (\mathbf{p}_{3D},\ 1)^\intercal$ are the extended coordinate of P, $\mathbf{p}_C$ in the sensor frame and $\mathbf{p}_{3D}$ in the world frame, respectively. The focal plane is orthogonal to the optical axis and is located at coordinate $z_c = -f$ where f is called focal length. By convention the x_f axis pointing to the right and the y_f axis pointing downward. The projection $\pi : \mathbb{R}^3 \mapsto \mathbb{R}^2$ of $\mathbf{p}_C$ of coordinates (x_c, y_c, z_c) into the focal plane is:

$$\mathbf{p}_f = \pi(\mathbf{p}_C) = \left[\frac{x_c}{z_c}, \frac{y_c}{z_c}\right]^\intercal \tag{A.9}$$

Note that this projection model presented here is yet only valid for undistorted images. In reality, because of lens' imperfections, the images produced by a camera are distorted. The distortion effects can be modeled as polynomial functions in order to define a bijective mapping between distorted and undistorted coordinates. The polynomial coefficients of distortion can be estimated during the calibration procedure. More details about distortion models can be found in [Sturm & Ramalingam 2011]. Then, the point on the focal plane is expressed in the image plane, which has by convention the x_{2D} axis pointing to the right and the y_{2D} axis pointing downward. It is common to define the origin of the image frame (in which pixels are indexed) at the top-left of the image. Using homogeneous coordinates, this transformation can be expressed as:

$$\widetilde{\mathbf{p}}_{2D} = (u, v, 1)^\intercal = \boldsymbol{K}\,\widetilde{\mathbf{p}}_f \tag{A.10}$$

with

$$\boldsymbol{K} = \boldsymbol{T}_{2Df} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{A.11}$$

where f_x and f_y represent the focal length (in pixels) of the camera along the axis x_c, y_c, c_x and c_y represent the coordinate of optical axis in the image plane (in pixel), and s represents the skew factor that can be used to model any shear distortion (this term is often omitted for modern cameras). Focal length in pixel can be computed as $f_x = f/\mu_x$ and $f_y = f/\mu_y$ where μ_x and μ_y are the size of pixels following x_c and y_c axis. The matrix $\boldsymbol{K}$ is thus often referred as the intrinsic calibration matrix constituted of intrinsic parameters which represent the coefficients that convert the position of points in the focal plane frame, into their pixel projections on the image frame.

In practice a calibration step must be performed in order to estimate all the intrinsic parameters of the camera (focal length, principal point, lens distortion) used in $\boldsymbol{K}$. This procedure consists in the creation of a dataset of images of a target with a known pattern (chessboard, apriltag board, etc.) are recorded with different poses. From

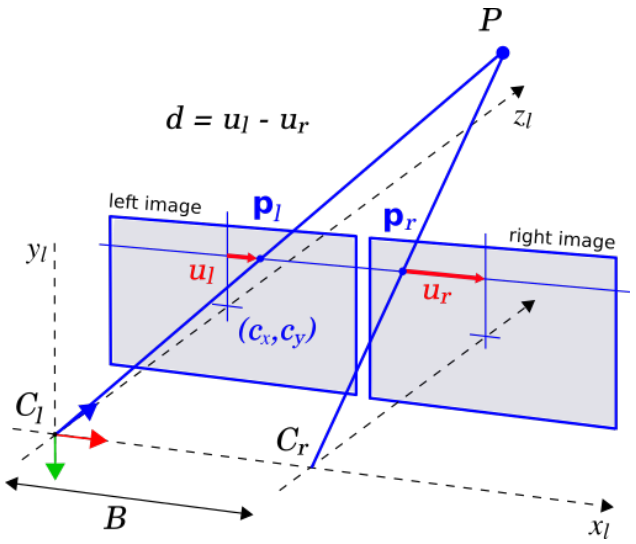


Figure A.2: *Scheme of a stereo-rig (rectified configuration).*

this dataset, non-linear least-square optimisation techniques are used to find the values of parameters that minimise the error of retro-projection of the known pattern, see [Hartley & Zisserman 2004] for more details. The practical procedure is presented in Sect A.2.3.

Stereo-vision Although cameras can describe the geometry of the perceived scene, this description is scaleless. Indeed, because the projection loses the depth information, a camera which observes two similar objects of different scales could produce the same image by adjusting their respective positions from the optical center. A single camera can only observe angles and not distances.

To overcome this, one can use a stereo-rig. A stereo-rig is composed of two synchronized cameras fixed together in a rigid way, such that the transformation from one camera frame to the other remains constant. Even if directly processing the raw stereo images is possible, working in a "rectified" geometry is more straightforward and computationally less intensive. In rectified geometry, both image planes coincide and the both camera frames are oriented in the same way (z_c-axes are parallel), therefore the epipolar lines (lines that connect any 3D point to the optical center of the other camera) are horizontal. For a real stereo-rig, knowing poses of both cameras, a rectifying transformation which defines for each camera a new rectified frame and an associated homography can be computed. In order to keep most of the field of view and to have a sufficient overlap, a common usage is to attach the two cameras on the same axis (x_c-axis), separated by a baseline B. Fig. A.2 presents the geometry of a rectified stereo-rig where the reference frame of the rig is set to the left camera frame with optical center C_l. Considering any

3D point projected on both cameras, because of the rectified geometry, both projections will be located on the same line, and the position difference between two points on the horizontal axis is called *disparity* and is denoted by d. The depth of the point z and the disparity d are inversely proportional:

$$z = \frac{-Bf_x}{d} \tag{A.12}$$

Similarly to cameras, the rigs need to be calibrated to determine each camera intrinsic parameters but also to determine the extrinsic parameters which constitutes the rigid-body motion between the two cameras (cf. Sect. A.2.3).

Depth estimation The obtained stereo-image pairs can be used to estimate depth maps which can be processed to perform mapping, see folowing Sect. 1.3.3. Multiple techniques exist to produce depth maps, and they will not be presented here due to their distance from the subject. However, we used the Efficient Large-Scale Stereo Matching (ELAS) algorithm [Geiger *et al.* 2010] to generate a depth map from stereo-rig in our real-world experiments. From a rectified stereo-image pair, the algorithm first extracts a disparity map which associates to every pixel of the left image with the algebraic distance separating it from its corresponding pixel on the right image, along the epipolar line. The computation of a dense disparity map is time consuming, and in real-time applications, ELAS approximates the disparity map using a Bayesian approach. In fact, it explicits and maximizes in parallel, for every observation, denoted $x_n^{(l)}$ on the left image, a posterior distribution $p(d_n|x_n^{(l)}, \mathcal{X}_n^{(l)}, \mathcal{S})$, where d_n is the disparity, $\mathcal{X}_n^{(l)}$ is the set of observations belonging to the epipolar line induced by $x_n^{(l)}$ on the right image, and $\mathcal{S}$ is a sub-set of support points where the disparity is computed. This set of points is meshed via a 2D Delaunay triangulation [Delaunay 1934] in order to define a prior distribution of the disparity $p(d_n|x_n^{(l)}, \mathcal{S})$ for every pixel on the left image, by linear interpolation. Finally, the depth map is then computed from the disparity map.

A.2.3 Camera and IMU calibration

The accurate calibration of a camera and an Inertial Measurement Unit (IMU) is a crucial stage in the design of visual or visual-inertial odometry algorithms. An IMU typically consists of a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer allows to measure the linear accelerations along the x, y and z axes. On the other hand, the gyroscopes allow to compute the angular velocities about the x, y and z axes. The IMU allows to estimate the current pose of the robot, by using the previous estimates and the current measurements from the accelerometer and the gyroscope.

As presented in Sect. A.2.2, the calibration of a camera consists in estimating the intrinsic parameters (coordinates of the principal point and the focal distances) and the coefficients of the distortion model (equidistant or radial-tangential). Furthermore, the usage of a stereo-rig requires the estimation of some parameters relating the cameras to the IMU. These are the rigid transformation between the IMU and the cameras, and the time delay between the IMU and camera clocks. The estimation of the geometric parameters is called spatial calibration, and the estimation of the time delay is called time calibration [Furgale *et al.* 2013, Rehder *et al.* 2016]. In our real-world experiments, see Chapter 5, the calibration has been performed with Kalibr [Furgale *et al.* 2014]. This toolbox estimates the calibration parameters by observing static patterns such as the AprilTag markers [Olson 2011]. For visual-inertial calibration, the IMU must be stimulated via translational and rotational motions about the three axes, while ensuring that the observed patterns cover the whole image plane, and in particular the image corners for a more accurate estimation of distortion coefficients. Finally, in the case of multiple cameras mounted on the robot, the calibration consists in estimating the rigid-body transforms between the the Apriltag markers and camera frames. To do so, sequences of images are acquired, where the markers are simultaneously visible in multiple cameras. For the IMU, the parameters are estimated by computing the Allan variance [El-Sheimy *et al.* 2007] from measurements captured over several hours on the static IMU.

A.2.4 ROS: the Robot Operating System

The Robot Operating System (ROS) is a set of software libraries and tools to build advanced robot applications (cf. Fig. A.3). From drivers to state-of-the-art open source algorithms, with powerful development tools, ROS allows to easily design interacting modules, implemented in C++ or Python. ROS works as a middleware and links software modules called "nodes" between them, or to simulators (Gazebo, `http://gazebosim.org/`) and hardware components, to design complex robotic systems. The nodes can communicate via standardized *messages*, which can be *published* on a *topic* by a node, by advertising the *master* which manages the communication. Another node can *subscribe* to this topic to have access to the message via a *callback*. Every node can have access to a message which is published on a topic as soon as it decides to listened to it. The publish & subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for Remote Procedure Call request & reply interactions, which are often required in a distributed system. Request & reply is done via a *service*, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the

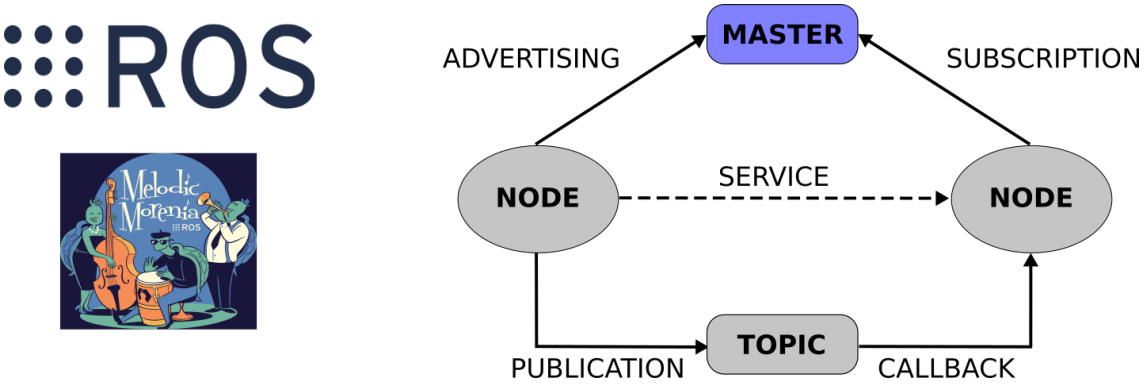


Figure A.3: *The Robot Operating System, ROS*. www.ros.org

service by sending the request message and awaiting the reply.

ROS allows to design highly modular architectures and it is user-friendly since the nodes can be easily incorporated in heterogeneous robotic platforms. Last but not the least, ROS allows to record all the experimental data (in the form of *Rosbags*), to replay them, and to monitor the overall robotic system. The work of this thesis has been realized on the *Melodic Morenia* version, released in 2018.

Appendix B

Graph theory

Appendix outline

B.1 Definitions 139

B.1.1 Adjacency relation 139

B.1.2 Special graphs 140

B.2 Traveling Salesman Problem and its variants 141

B.2.1 Formulation 141

B.2.2 Numerical solvers 142

This appendix presents some basic notions of graph theory [Rigo 2009], and the well known Traveling Salesman Problem, with some of its numerical solvers, used in the thesis.

B.1 Definitions

A graph is a mathematical structure which can be used to model relations between objects. A graph $\mathcal{G}$ is defined by a set of nodes (or vertices), interconnected by edges. A node can represent an object (place, person, agent) and every edge represents a connection between two objects (a distance, a speed, a logical link, a communication channel). The interaction topology of a network of N agents is represented by a graph $\mathcal{G} = (V, E)$ with the set of nodes $V = \{1, 2, ..., N\}$ and edges $E \subseteq V \times V$. E_{ij} denotes the edge between the nodes i and j. A subgraph of $\mathcal{G}$ is another graph formed by a subset of the vertices and edges of $\mathcal{G}$. Two nodes directly linked by an edge in the graph are said to be neighbors. The set of neighbors of a node i is $\mathcal{N}_i = \{ j \in V \mid (i, j) \in E,\ i \neq j \}$. $N_i = |\mathcal{N}_i|$, the cardinality of $\mathcal{N}_i$, is the number of neighbors of node i.

B.1.1 Adjacency relation

The interaction between the nodes of a graph can be represented with the notion of adjacency. The adjacency matrix $A = [a_{ij}]_{N \times N}$ associated with a graph $\mathcal{G}$ is a square matrix of

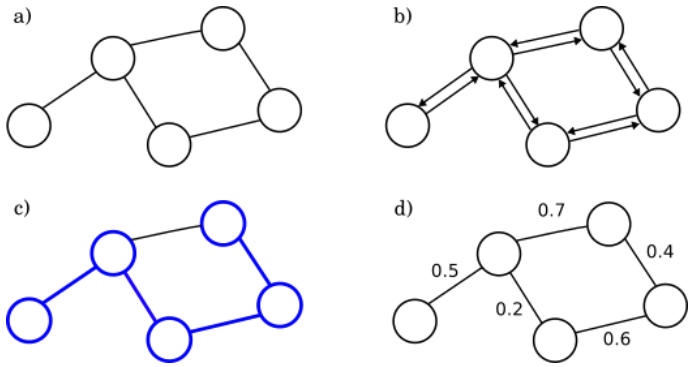


Figure B.1: *Examples of graphs:* a) undirected graph; b) directed graph; c) Hamiltonian path in an undirected graph (depicted in blue); d) weighted graph.

size $N \times N$, where N is the number of nodes. The value of each non-negative element a_{ij} of the matrix is the weight of the edge E_{ij}. Thus, if there is no connection between nodes i and j, $a_{ij} = 0$. On the other hand, if nodes i and j are connected then, $a_{ij} \neq 0$, and they are said to be adjacent. If the graph is undirected, one has $a_{ij} = a_{ji}$ for all (i, j). Moreover, if the graph is unweighted, one has $a_{ij} = 1$ or $a_{ij} = 0$ for all (i, j).

The edges of a graph can also be directed, indicating a direction from a node to another. A directed edge represents a one-way interaction between two nodes, see the example in Fig. B.1-b). A graph with directed edges is called a directed graph (or digraph), and the directed edges are called arcs. An undirected graph is equivalent to a directed graph where all arcs are doubled, see Fig. B.1-a). An edge E_{ij} can be weighted considering its importance compared to other edges. A graph where edges are weighted is called a weighted graph, see Fig. B.1-d). An unweighted graph is a graph where every edge possesses a unit weight.

Graph topology can be time-varying because connections may appear or disappear due, *e.g.* to the influence of the distance between nodes, interferences, material imperfections, or as a result of the selected communication strategy. When edges do not change over time, a graph is said to be *fixed* or *static*. When edges vary over time, the graph is said to be *time-varying* or *dynamic*.

B.1.2 Special graphs

A *path* from i to j is a sequence of distinct nodes, starting from i and ending with j, such that each pair of consecutive nodes is adjacent. The number of edges defines the *length* of the path. In a directed graph, a path is a set of adjacent edges connecting two nodes by respecting the direction of arcs. A directed path is a *chain*. If there is a path from i to j, then i and j are said to be connected. If all pairs of nodes in $\mathcal{G}$ are connected, then

$\mathcal{G}$ is called *connected*. A digraph is *strongly connected* (or *s-connected*) if every node is reachable from every other node. A graph is *fully-connected* or *complete* if every node is connected by a unique edge (one in each direction) with all other nodes. Finally, a graph is *cyclic* if it contains a cycle *i.e.* there is a non-trivial path that starts and ends at the same node. On the contrary, a graph is *acyclic* if it does not contain any cycle. A *ring* is a cyclic graph composed of only one path. A connected acyclic graph with a unique root is called a *tree*. A *spanning tree* is a tree (acyclyic, connected graph) that touches each node. Finally, a *Hamiltonian path* is a path in an undirected or directed graph that visits each vertex exactly once, see Fig. B.1-c). A *Hamiltonian cycle* is a Hamiltonian path that is a cycle.

B.2 Traveling Salesman Problem and its variants

The Travelling Salesman Problem (also called the Travelling Salesperson Problem or TSP) [Lawler 1985, Punnen 2007] is a NP-hard problem which consists, given a list of cities and their pairwise distances, to find the shortest possible path that visits all the cities exactly once and returns to the origin city (or depot).

The TSP has several applications even in its simpliest formulation, such as logistics, planning, microchip design or even DNA sequencing. In this thesis, the TSP has been used for path planning. The roadmap passing through the cities can be represented as a weighted (directed) graph, where the cities are the nodes and the edges (arcs) are the possible paths. The weight associated with each edge, represents the distance between two cities. Finding the shortest path in the graph, visiting each vertex exactly once and starting and coming back to the depot is a minimization problem. The use of a directed graph allows to model paths of different distance between two nodes by setting different weights on the arcs. If the adjacency matrix becomes asymmetric (cf. Sect. B.1.1), the TSP problem is said to be asymmetric (ATSP).

B.2.1 Formulation

The TSP can be formulated as an Integer Linear Program (ILP) [Dantzig 2016]. Several formulations are known. Two notable formulations are the Miller–Tucker–Zemlin (MTZ) formulation and the Dantzig–Fulkerson–Johnson (DFJ) formulation. Let n be the number of cities, and a_{ij} the weight of the arc going from i to j, with $i, j \in \{1, \ldots, n\}$. The DFJ

formulation of the ATSP is then:

$$\textbf{Minimize} \quad \sum_{i=1}^{n} \sum_{i \neq j,\, j=1}^{n} a_{ij}\, x_{ij} \tag{B.1a}$$

subject to (B.1b)

$$x_{ij} \in \{0, 1\}, \qquad i, j \in \{1, \ldots, n\}, \tag{B.1c}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j \in \{1, \ldots, n\}, \tag{B.1d}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i \in \{1, \ldots, n\}, \tag{B.1e}$$

$$\sum_{i \in Q} \sum_{j \in Q,\, j \neq i} x_{ij} \leq |Q| - 1, \qquad \forall\, Q \subsetneq \{1, \ldots, n\},\ |Q| > 2\,, \tag{B.1f}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if the arc from } i \text{ to } j \text{ belongs to the optimal path,} \\ 0 & \text{otherwise.} \end{cases} \tag{B.2}$$

The minimization of the objective function (B.1a) is subject to multiple constraints:

- (B.1d) forces the agent to visit the city,
- (B.1e) forces the agent to quit the city,
- (B.1f) avoids the presence of subtours.

B.2.2 Numerical solvers

There exist several methods to solve problem (B.1) which can be classified as *exact* or *approximate*.Some of them are presented in this section.

Being an NP-hard problem, try a brute-force approach (i.e. try all possible combinations of cities and keep the shortest path) leads to an exact solution within a long time window: in fact, the complexity is $O(n!)$, where n is the number of cities. The Bellman–Held–Karp algorithm [Bellman 1962, Held & Karp 1962] is a dynamic programming algorithm which solves the problem within $O(n^2 2^n)$. Improving this time bound seems to be difficult and, to this day, there is no exact algorithm which allows to solve the problem within $O(1.9999^n)$ [Woeginger 2003].

However, approximation methods and heuristics can yield fast and accurate solutions to the TSP and are more suited to robotic applications. Among the numerous existing methods, there are the constructive heuristics such as the nearest neighbor algorithm (or greedy algorithm). This algorithm lets the salesman choose the nearest city as his next destination and provides a solution. The complexity reduces to $O(n)$, but the gap between

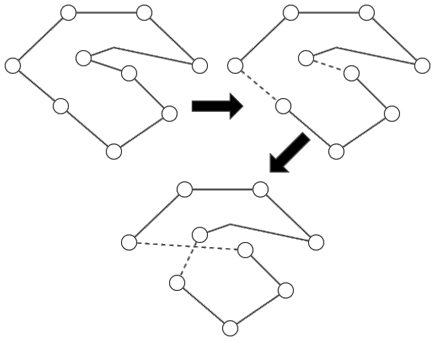


Figure B.2: *Steps of the 2-opt heuristic for the TSP.*

the exact and approximate solution remains relatively large. [Gutin *et al.* 2002] has shown that the nearest-neighbor algorithm always produces a tour which is not worse than at least $n/2 - 1$ other tours, but for some instance it finds a tour which is not worse than at most $n - 2$ other tours, $n \geq 4$.

Other methods, based on local search, perform successive edge permutations to find a local shortest paths and incrementally find the solution. For instance, the pairwise exchange or 2-opt technique [Croes 1958] involves iteratively removing two edges and replacing these with two different edges that reconnect the fragments created by edge removal into a new and shorter tour. Similarly, the 3-opt technique removes 3 edges and reconnects them to form a shorter tour (see Fig. B.2). Unfortunately, the outcome of these algorithms is heavily influenced by the initial path.

A generalization of these methods, the Lin-Kernighan heuristic [Lin & Kernighan 1973] adjusts the number of permutations required locally, while keeping the shortest tour at every iteration. The authors introduced a powerful variable k-opt algorithm. For a given a tour, the algorithm deletes k mutually disjoint edges. Then, it reassembles the remaining fragments into a tour, leaving no disjoint sub-tours (i.e. it does not connect a fragment's endpoints together). This ultimately allows to simplify the TSP into a much simpler problem. In fact, each fragment endpoint can be connected to $2k - 2$ other possibilities: of $2k$ total fragment endpoints available, the two endpoints of the fragment under consideration are disallowed. Such a constrained $2k$-city TSP can then be solved with brute force methods to find the least-cost recombination of the original fragments. Moreover, at each iteration, the algorithm chooses the best value of k for a given node, and selects the one that may result in a shorter tour. The LKH heuristic [Helsgaun 2000] improves upon an earlier implementation which founds optimal solutions to multiple variants of the TSP, such as the Hamiltonian cycle/path

problem or the vehicle routing problem[1]. A wide empirical study has been performed in [Arthur & Frendewey 1988], by testing the LKH over a large set of randomly generated TSPs with known optimal tours. The study shows that the LKH allows to efficiently solve symmetric TSPs and asymmetric TSPs, after converting them into a symmetric problem using the approach proposed in [Jonker & Volgenant 1983]. Moreover, it finds optimal solutions for both metric (i.e. the weights on the edges respect the triangular inequality) and non-metric problems (where optimum was found in one trial) with an average running time of approximately $O(n^{2.2})$.

[1] http://webhotel4.ruc.dk/~keld/research/LKH-3/

Bibliography

[Agrawal *et al.* 2008] M. Agrawal, K. Konolige et M. R. Blas. *Censure: Center surround extremas for realtime feature detection and matching*. In Proc. Eur. Conf. Comp. Vis., pages 102–115. Springer, 2008. (referred on page 10)

[Akishita *et al.* 1990] S. Akishita, S. Kawamura et K.I. Hayashi. *New navigation function utilizing hydrodynamic potential for mobile robot*. In Proceedings of the IEEE International Workshop on Intelligent Motion Control, volume 2, pages 413–417. IEEE, 1990. (referred on page 23)

[Albertz 2002] J. Albertz. *Albrecht Meydenbauer-Pioneer of photogrammetric documentation of the cultural heritage*. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, vol. 34, no. 5/C7, pages 19–25, 2002. (referred on page 13)

[Alcántara *et al.* 2021] A. Alcántara, J. Capitán, R. Cunha et A. Ollero. *Optimal trajectory planning for cinematography with multiple Unmanned Aerial Vehicles*. Robot. Autonom. Syst., vol. 140, page 103778, 2021. (referred on pages 1 and 115)

[Amigoni & Gallo 2005] F. Amigoni et A. Gallo. *A Multi-Objective Exploration Strategy for Mobile Robots*. In Proc. IEEE Int. Conf. Robot. Automat., pages 3850–3855, 2005. (referred on pages 33 and 118)

[Andersen *et al.* 2010] V. Andersen, H. Aanæs et J.A. Bærentzen. *Surfel Based Geometry Reconstruction.* TPCG, vol. 10, pages 39–44, 2010. (referred on page 14)

[Arthur & Frendewey 1988] J.L. Arthur et J.O. Frendewey. *Generating travelling-salesman problems with known optimal tours*. Journal of the Operational Research Society, vol. 39, no. 2, pages 153–159, 1988. (referred on page 144)

[Atanasov *et al.* 2015] N. Atanasov, J. Le Ny, K. Daniilidis et G.J. Pappas. *Decentralized active information acquisition: Theory and application to multi-robot SLAM*. In Proc. IEEE Int. Conf. Robot. Automat., pages 4775–4782, 2015. (referred on pages 33 and 119)

[Avizienis *et al.* 2004] A. Avizienis, J.C. Laprie, B. Randell et C. Landwehr. *Basic concepts and taxonomy of dependable and secure computing*. IEEE Trans. Dependable Secure Comput., vol. 1, no. 1, pages 11–33, 2004. (referred on pages 11 and 73)

[Baker & Matthews 2004] S. Baker et I. Matthews. *Lucas-kanade 20 years on: A unifying framework*. Int. J. Comput. Vision, vol. 56, no. 3, pages 221–255, 2004. (referred on page 10)

[Barraquand & Latombe 1990] J. Barraquand et J.C. Latombe. *A Monte-Carlo algorithm for path planning with many degrees of freedom*. In Proc. IEEE Int. Conf. Robot. Automat., pages 1712–1717. IEEE, 1990. (referred on page 23)

[Barraquand & Latombe 1991] J. Barraquand et J.C. Latombe. *Robot motion planning: A distributed representation approach*. The International Journal of Robotics Research, vol. 10, no. 6, pages 628–649, 1991. (referred on page 23)

[Bartolomei *et al.* 2020] L. Bartolomei, L. Teixeira et M. Chli. *Perception-aware path planning for uavs using semantic segmentation*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 5808–5815. IEEE, 2020. (referred on page 110)

[Batinovic *et al.* 2021] A. Batinovic, T. Petrovic, A. Ivanovic, F. Petric et B. Stjepan. *A Multi-Resolution Frontier-Based Planner for Autonomous 3D Exploration*. IEEE Robot. Autonom. Lett., vol. 6, no. 3, pages 4528–4535, 2021. (referred on pages 30, 31 and 117)

[Bay *et al.* 2006] H. Bay, T. Tuytelaars et L. Van Gool. *Surf: Speeded up robust features*. In Proc. Eur. Conf. Comp. Vis., pages 404–417. Springer, 2006. (referred on page 10)

[Bekey 2005] G.A. Bekey. Autonomous robots: from biological inspiration to implementation and control. MIT press, 2005. (referred on page 8)

[Bellman 1962] R. Bellman. *Dynamic programming treatment of the travelling salesman problem*. Journal of the ACM (JACM), vol. 9, no. 1, pages 61–63, 1962. (referred on page 142)

[Bernardini *et al.* 1999] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva et G. Taubin. *The ball-pivoting algorithm for surface reconstruction*. IEEE Trans. Vis. Comput. Graph., vol. 5, no. 4, pages 349–359, 1999. (referred on page 14)

[Besl & McKay 1992] Paul J Besl et Neil D McKay. *Method for registration of 3-D shapes*. In Sensor Fusion IV: Control Paradigms and Data Structures, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992. (referred on page 28)

[Best *et al.* 2016] G. Best, O. Cliff, T. Patten, R. Mettu et R. Fitch. *Decentralised Monte Carlo Tree Search for Active Perception*. In Proc. Int. Work. Algor. Found. Robot. (WAFR), 2016. Paper 50. (referred on pages 33 and 119)

[Bian *et al.* 2017] A.A. Bian, J.M. Buhmann, A. Krause et S. Tschiatschek. *Guarantees for greedy maximization of non-submodular functions with applications*. In International conference on machine learning, pages 498–507. PMLR, 2017. (referred on pages 34, 61 and 81)

[Bircher *et al.* 2016] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova et R. Siegwart. *Receding Horizon "Next-Best-View" Planner for 3D Exploration*. In Proc. IEEE Int. Conf. Robot. Automat., pages 1462–1468, 2016. (referred on pages 28, 30, 32 and 117)

[Bircher *et al.* 2018] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova et R. Siegwart. *Receding horizon path planning for 3D exploration and surface inspection*. Auton. Robot., vol. 42, no. 2, pages 291–306, 2018. (referred on pages 32 and 118)

[Birk & Carpin 2006] A. Birk et S. Carpin. *Merging Occupancy Grid Maps From Multiple Robots*. Proceedings of the IEEE, vol. 94, no. 7, pages 1384–1397, 2006. (referred on page 16)

[Border *et al.* 2018] R. Border, J.D. Gammell et P. Newman. *Surface Edge Explorer (SEE): Planning Next Best Views Directly from 3D Observations*. In Proc. IEEE Int. Conf. Robot. Automat., pages 6116–6123, 2018. (referred on pages 28, 29 and 118)

[Bresenham 1965] J.E. Bresenham. *Algorithm for computer control of a digital plotter*. IBM Syst. J., vol. 4, no. 1, pages 25–30, 1965. (referred on pages 31, 42 and 117)

[Burgard *et al.* 2005] W. Burgard, M. Moors, C. Stachniss et F.E. Schneider. *Coordinated multi-robot exploration*. IEEE Trans. Robot., vol. 21, no. 3, pages 376–386, 2005. (referred on pages 33 and 118)

[Cadena *et al.* 2016] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid et J.J. Leonard. *Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age*. IEEE Trans. Robot., vol. 32, no. 6, pages 1309–1332, 2016. (referred on page 108)

[Calonder *et al.* 2010] M. Calonder, V. Lepetit, C. Strecha et P. Fua. *Brief: Binary robust independent elementary features*. In Proc. Eur. Conf. Comp. Vis., pages 778–792. Springer, 2010. (referred on page 10)

[Charrow *et al.* 2015] B. Charrow, S. Liu, V. Kumar et N. Michael. *Information-theoretic mapping using Cauchy-Schwarz Quadratic Mutual Information*. In Proc. IEEE Int. Conf. Robot. Automat., pages 4791–4798, 2015. (referred on pages 33 and 119)

[Chaslot 2010] G.M.J.B. Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Maastricht University, 2010. (referred on pages 33 and 119)

[Chen *et al.* 2008] S. Chen, Y.F. Li, W. Wang et J. Zhang. Active Sensor Planning for Multiview Vision Tasks. Springer, 2008. (referred on pages 27 and 28)

[Connolly 1985] C. Connolly. *The Determination of Next Best Views*. In Proc. IEEE Int. Conf. Robot. Automat., volume 2, pages 432–435, 1985. (referred on pages 27 and 118)

[Corah & Michael 2019] M. Corah et N. Michael. *Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice*. Auton. Robot., vol. 43, no. 2, pages 485–501, 2019. (referred on pages 33 and 119)

[Corah & Michael 2021] M. Corah et N. Michael. *Volumetric Objectives for Multi-Robot Exploration of Three-Dimensional Environments*. arXiv preprint arXiv:2103.11625, 2021. (referred on pages 33 and 119)

[Corah *et al.* 2019] M. Corah, C. O'Meadhra, K. Goel et N. Michael. *Communication-Efficient Planning and Mapping for Multi-Robot Exploration in Large Environments*. IEEE Robot. Autonom. Lett., vol. 4, no. 2, pages 1715–1721, 2019. (referred on pages 33 and 119)

[Crespo *et al.* 2020] J. Crespo, J.C. Castillo, O.M. Mozos et R. Barber. *Semantic information for robot navigation: A survey*. Applied Sciences, vol. 10, no. 2, page 497, 2020. (referred on page 110)

[Croes 1958] G.A. Croes. *A method for solving traveling-salesman problems*. Operations research, vol. 6, no. 6, pages 791–812, 1958. (referred on page 143)

[Şucan *et al.* 2012] I.A. Şucan, M. Moll et L.E. Kavraki. *The Open Motion Planning Library*. IEEE Rob. Autom. Mag., vol. 19, no. 4, pages 72–82, 2012. (referred on pages 47, 66, 87 and 96)

[Curless & Levoy 1996] B. Curless et M. Levoy. *A volumetric method for building complex models from range images*. In ACM Trans. Graph., pages 303–312, 1996. (referred on pages 15, 39, 59 and 117)

[Dantzig 2016] G. Dantzig. Linear programming and extensions. Princeton university press, 2016. (referred on page 141)

[de Figueiredo *et al.* 2021] R.P. de Figueiredo, J. Sejersen, J.G. Hansen, M. Brandão et E. Kayacan. *Real-Time Volumetric-Semantic Exploration and Mapping: An Uncertainty-Aware Approach*. arXiv preprint arXiv:2109.01474, 2021. (referred on page 110)

[Delaunay 1934] B. Delaunay. *Sur la sphere vide*. Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, vol. 7, no. 793-800, pages 1–2, 1934. (referred on pages 14 and 135)

[Digani *et al.* 2015] V. Digani, L. Sabattini, C. Secchi et C. Fantuzzi. *Ensemble Coordination Approach in Multi-AGV Systems Applied to Industrial Warehouses*. IEEE Trans. Autom. Sci. Eng., vol. 12, no. 3, pages 922–934, 2015. (referred on pages 1 and 115)

[Dijkstra 1959] E.W. Dijkstra. *A note on two problems in connexion with graphs*. Numerische mathematik, vol. 1, no. 1, pages 269–271, 1959. (referred on page 20)

[Dubois *et al.* 2021] R. Dubois, A. Eudes et V. Frémont. *Sharing visual-inertial data for collaborative decentralized simultaneous localization and mapping*. Robot. Autonom. Syst., page 103933, 2021. (referred on page 109)

[Duhautbout *et al.* 2019] T. Duhautbout, J. Moras et J. Marzat. *Distributed 3D TSDF Manifold Mapping for Multi-Robot Systems*. In Proc. 9th European Conf. Mobile Robots, 2019. (referred on pages 17, 78, 86, 96 and 117)

[Edelsbrunner *et al.* 1983] H. Edelsbrunner, D. Kirkpatrick et R. Seidel. *On the shape of a set of points in the plane*. IEEE Trans. Inf. Theory, vol. 29, no. 4, pages 551–559, 1983. (referred on page 14)

[Edelsbrunner 1995] H. Edelsbrunner. *Smooth surfaces for multi-scale shape representation*. In International conference on foundations of software technology and theoretical computer science, pages 391–412. Springer, 1995. (referred on page 14)

[El-Sheimy *et al.* 2007] N. El-Sheimy, H. Hou et X. Niu. *Analysis and modeling of inertial sensors using Allan variance*. IEEE Trans. Instr. and Meas., vol. 57, no. 1, pages 140–149, 2007. (referred on page 136)

[Elfes 1989] A. Elfes. *Using Occupancy Grids for Mobile Robot Perception and Navigation*. Computer, vol. 22, no. 6, pages 46–57, Juin 1989. (referred on page 15)

[Engel *et al.* 2014] J. Engel, T. Schöps et D. Cremers. *LSD-SLAM: Large-scale direct monocular SLAM*. In Proc. Eur. Conf. Comp. Vis., pages 834–849. Springer, 2014. (referred on page 9)

[Engel *et al.* 2015] J. Engel, J. Stückler et D. Cremers. *Large-scale direct SLAM with stereo cameras*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 1935–1942, 2015. (referred on pages 40 and 59)

[Filippidis & Kyriakopoulos 2012] I.F. Filippidis et K.J. Kyriakopoulos. *Navigation functions for everywhere partially sufficiently curved worlds*. In Proc. IEEE Int. Conf. Robot. Automat., pages 2115–2120. IEEE, 2012. (referred on page 23)

[Filippidis *et al.* 2012] I.F. Filippidis, K.J. Kyriakopoulos et P.K. Artemiadis. *Navigation functions learning from experiments: Application to anthropomorphic grasping*. In Proc. IEEE Int. Conf. Robot. Automat., pages 570–575. IEEE, 2012. (referred on page 23)

[Fischler & Bolles 1981] M.A. Fischler et R.C. Bolles. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. Communications of the ACM, vol. 24, no. 6, pages 381–395, 1981. (referred on page 10)

[Fisher *et al.* 1978] M.L. Fisher, G.L. Nemhauser et L.A. Wolsey. *An Analysis of Approximations for Maximizing Submodular Set Functions — II*. Math. Program. Stud., vol. 8, pages 73–87, 1978. (referred on pages 34, 61 and 81)

[Forster *et al.* 2013] C. Forster, S. Lynen, L. Kneip et D. Scaramuzza. *Collaborative monocular SLAM with multiple micro aerial vehicles*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 3962–3970, 2013. (referred on page 16)

[Fraundorfer & Scaramuzza 2012] F. Fraundorfer et D. Scaramuzza. *Visual odometry: Part ii: Matching, robustness, optimization, and applications*. IEEE Rob. Autom. Mag., vol. 19, no. 2, pages 78–90, 2012. (referred on page 9)

[Funk *et al.* 2021] N. Funk, J. Tarrio, S. Papatheodorou, M. Popović, P.F. Alcantarilla et S. Leutenegger. *Multi-resolution 3D mapping with explicit free space representation for fast and accurate mobile robot motion planning*. IEEE Robot. Autonom. Lett., vol. 6, no. 2, pages 3553–3560, 2021. (referred on page 16)

[Furgale *et al.* 2013] P. Furgale, J. Rehder et R. Siegwart. *Unified temporal and spatial calibration for multi-sensor systems*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 1280–1286. IEEE, 2013. (referred on page 136)

[Furgale *et al.* 2014] P. Furgale, H. Sommer, J. Maye, J. Rehder, T. Schneider et L. Oth. *Kalibr: A unified camera/IMU calibration toolbox*, 2014. (referred on pages 95 and 136)

[Furrer *et al.* 2016] F. Furrer, M. Burri, M. Achtelik et R. Siegwart. *RotorS – A Modular Gazebo MAV Simulator Framework*. In A. Koubaa, editor, Robot Operating System (ROS): The Complete Reference, volume 1, pages 595–625. Springer, 2016. (referred on pages 47, 66 and 86)

[Gao *et al.* 2018] X. Gao, R. Wang, N. Demmel et D. Cremers. *LDSO: Direct sparse odometry with loop closure*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 2198–2204. IEEE, 2018. (referred on page 108)

[Geiger *et al.* 2010] A. Geiger, M. Roser et R. Urtasun. *Efficient large-scale stereo matching*. In Proc. Asian Conf. Comp. Vis., pages 25–38, 2010. (referred on pages 95, 96, 124 and 135)

[Girardeau-Montaut 2016] D. Girardeau-Montaut. *CloudCompare*. France: EDF R&D Telecom ParisTech, 2016. (referred on page 13)

[Godsil & Royle 2001] C. Godsil et G. Royle. Algebraic Graph Theory. Springer, 2001. (referred on page 43)

[Goerzen *et al.* 2010] C. Goerzen, Z. Kong et B. Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance*. J. Intell. Robot. Syst., vol. 57, no. 1, pages 65–100, 2010. (referred on page 20)

[González-Banos & Latombe 2002] H. González-Banos et J.-C. Latombe. *Navigation strategies for exploring indoor environments*. Int. J. Robot. Res., vol. 21, no. 10-11, pages 829–848, 2002. (referred on pages 27 and 44)

[Grinvald *et al.* 2021] M. Grinvald, F. Tombari, R. Siegwart et J. Nieto. *TSDF++: A Multi-Object Formulation for Dynamic Object Tracking and Reconstruction*. arXiv preprint arXiv:2105.07468, 2021. (referred on page 16)

[Gutin *et al.* 2002] G. Gutin, A. Yeo et A. Zverovich. *Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP*. Discrete Applied Mathematics, vol. 117, no. 1-3, pages 81–86, 2002. (referred on page 143)

[Hardouin *et al.* 2019] G. Hardouin, S. Bertrand et H. Piet-Lahanier. *Distributed event-triggered consensus of multi-agent systems with measurement noise and guaran-*

teed interval bounds. IFAC-PapersOnLine, vol. 52, no. 20, pages 7–12, 2019. (referred on page 110)

[Hardouin *et al.* 2020a] G. Hardouin, J. Moras, F. Morbidi, J. Marzat et E. Mouaddib. *Next-Best-View planning for surface reconstruction of large-scale 3D environments with multiple UAVs*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 1567–1574, 2020. (referred on pages 5, 56, 61 and 96)

[Hardouin *et al.* 2020b] G. Hardouin, F. Morbidi, J. Moras, J. Marzat et E. Mouaddib. *Surface-driven Next-Best-View planning for exploration of large-scale 3D environments*. In Proc. 21st IFAC World Congress, pages 15501–15507, 2020. (referred on pages 5 and 36)

[Harris & Stephens 1988] C. Harris et M. Stephens. *A combined corner and edge detector*. In Alvey vision conference, volume 15, pages 10–5244. Citeseer, 1988. (referred on page 10)

[Hart *et al.* 1968] P.E. Hart, N.J. Nilsson et B. Raphael. *A formal basis for the heuristic determination of minimum cost paths*. IEEE Trans. Syst. Man Cybern. Syst., vol. 4, no. 2, pages 100–107, 1968. (referred on page 21)

[Hartley & Zisserman 2004] R.I. Hartley et A. Zisserman. Multiple view geometry in computer vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004. (referred on pages 10, 13 and 134)

[Hauser 2015] K. Hauser. *Lazy Collision Checking in Asymptotically-Optimal Motion Planning*. In Proc. IEEE Int. Conf. Robot. Automat., pages 2951–2957, 2015. (referred on pages 25, 36, 56 and 117)

[He *et al.* 2017] K. He, G. Gkioxari, P. Dollár et R. Girshick. *Mask r-cnn*. In Proc. IEEE Int. Conf. Comp. Vis., pages 2961–2969, 2017. (referred on page 110)

[Heemels *et al.* 2012] W.P.M.H. Heemels, K.H. Johansson et P. Tabuada. *An introduction to event-triggered and self-triggered control*. In Proc. 51st IEEE Conf. Dec. Contr., pages 3270–3285. IEEE, 2012. (referred on page 110)

[Held & Karp 1962] M. Held et R.M. Karp. *A dynamic programming approach to sequencing problems*. Journal of the Society for Industrial and Applied mathematics, vol. 10, no. 1, pages 196–210, 1962. (referred on page 142)

[Hellström 2011] T. Hellström. *Robot navigation with potential fields*. Department of Computer Science, Umea University, 2011. (referred on page 22)

[Helsgaun 2000] K. Helsgaun. *An effective implementation of the Lin–Kernighan traveling salesman heuristic*. Eur. J. Oper. Rer., vol. 126, no. 1, pages 106–130, 2000. (referred on pages 45, 96 and 143)

[Hitz *et al.* 2017] G. Hitz, E. Galceran, M.E Garneau, F. Pomerleau et R. Siegwart. *Adaptive continuous-space informative path planning for online environmental monitoring*. Journal of Field Robotics, vol. 34, no. 8, pages 1427–1449, 2017. (referred on page 109)

[Hornung *et al.* 2013] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss et W. Burgard. *OctoMap: An efficient probabilistic 3D mapping framework based on octrees*. Auton. Robot., vol. 34, no. 3, pages 189–206, 2013. (referred on pages 15, 16 and 116)

[Howard 2004] A. Howard. *Multi-robot mapping using manifold representations*. In Proc. IEEE Int. Conf. Robot. Automat., volume 4, pages 4198–4203, 2004. (referred on pages 16, 17, 75, 117 and 122)

[Jadidi *et al.* 2017] M.G. Jadidi, L. Gan, S.A. Parkison, J. Li et R.M. Eustice. *Gaussian processes semantic map representation*. arXiv preprint arXiv:1707.01532, 2017. (referred on pages 16, 109 and 110)

[Jawaid & Smith 2015] S.T. Jawaid et S.L. Smith. *Informative path planning as a maximum traveling salesman problem with submodular rewards*. Discrete Appl. Math., vol. 186, pages 112–127, 2015. (referred on pages 61 and 81)

[Jonker & Volgenant 1983] R. Jonker et T. Volgenant. *Transforming asymmetric into symmetric traveling salesman problems*. Operations Research Letters, vol. 2, no. 4, pages 161–163, 1983. (referred on pages 45 and 144)

[Kamel *et al.* 2017] M. Kamel, T. Stastny, K. Alexis et R. Siegwart. *Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System*. In A. Koubaa, editor, Robot Operating System (ROS) The Complete Reference, volume 2, pages 3–29. Springer, 2017. (referred on pages 48, 66, 87 and 96)

[Karaman & Frazzoli 2011] S. Karaman et E. Frazzoli. *Sampling-based algorithms for optimal motion planning*. Int. J. Robot. Res., vol. 30, no. 7, pages 846–894, 2011. (referred on pages 25, 26 and 117)

[Kavraki *et al.* 1996] L.E. Kavraki, P. Svestka, J.C. Latombe et M.H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. IEEE Trans. Robot. Autom., vol. 12, no. 4, pages 566–580, 1996. (referred on pages 23 and 117)

[Kazhdan *et al.* 2006] M. Kazhdan, M. Bolitho et H. Hoppe. *Poisson surface reconstruction*. In Proceedings of the fourth Eurographics symposium on Geometry processing, volume 7, 2006. (referred on page 14)

[Keselman *et al.* 2017] L. Keselman, J.I. Woodfill, A. Grunnet-Jepsen et A. Bhowmik. *Intel RealSense Stereoscopic Depth Cameras*. In Proc. IEEE Conf. Comp. Vis. Pattern Recogn. Workshops, pages 1–10, 2017. (referred on pages 47 and 66)

[Khatib 1986] O. Khatib. *Real-time obstacle avoidance for manipulators and mobile robots*. In Autonomous Robot Vehicles, pages 396–404. Springer, 1986. (referred on page 22)

[Klingensmith *et al.* 2015] M. Klingensmith, I. Dryanovski, S. Srinivasa et J. Xiao. *CHISEL: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields*. In Proc. Robotics: Science and Systems XI, volume 4, 2015. (referred on pages 16 and 117)

[Koch *et al.* 2016] P. Koch, S. May, M. Schmidpeter et M. Kühn. *Multi-robot localization and mapping based on signed distance functions*. J. Intell. Robot. Syst., vol. 83, no. 3-4, pages 409–428, 2016. (referred on page 17)

[Koch *et al.* 2019] Tobias Koch, Marco Körner et Friedrich Fraundorfer. *Automatic and semantically-aware 3D UAV flight planning for image-based 3D reconstruction*. Remote Sensing, vol. 11, no. 13, page 1550, 2019. (referred on page 110)

[Kompis *et al.* 2021] Y. Kompis, L. Bartolomei, R. Mascaro, L. Teixeira et M. Chli. *Informed Sampling Exploration Path Planner for 3D Reconstruction of Large Scenes*. IEEE Robot. Autonom. Lett., vol. 6, no. 4, pages 7893–7900, 2021. (referred on page 32)

[Kriegel *et al.* 2011] S. Kriegel, T. Bodenmüller, M. Suppa et G. Hirzinger. *A surface-based next-best-view approach for automated 3D model completion of unknown objects*. In Proc. IEEE Int. Conf. Robot. Automat., pages 4869–4874, 2011. (referred on pages 28 and 30)

[Kriegel *et al.* 2015] S. Kriegel, C. Rink, T. Bodenmüller et M. Suppa. *Efficient next-best-scan planning for autonomous 3D surface reconstruction of unknown objects*. J. Real-Time Image Pr., vol. 10, no. 4, pages 611–631, 2015. (referred on pages 29 and 30)

[Kundu *et al.* 2014] A. Kundu, Y. Li, F. Dellaert, F. Li et J.M. Rehg. *Joint semantic segmentation and 3d reconstruction from monocular video*. In Proc. Eur. Conf. Comp. Vis., pages 703–718. Springer, 2014. (referred on page 16)

[Lague *et al.* 2013] D. Lague, N. Brodu et J. Leroux. *Accurate 3D comparison of complex topography with terrestrial laser scanner: Application to the Rangitikei canyon (N-Z)*. ISPRS J. photogramm., vol. 82, pages 10–26, 2013. (referred on pages 50, 67 and 87)

[Lajoie *et al.* 2021] P.Y. Lajoie, B. Ramtoula, F. Wu et G. Beltrame. *Towards Collaborative Simultaneous Localization and Mapping: a Survey of the Current Research Landscape*. arXiv preprint arXiv:2108.08325, 2021. (referred on page 109)

[Lau *et al.* 2013] B. Lau, C. Sprunk et W. Burgard. *Efficient grid-based spatial representations for robot navigation in dynamic environments*. Robot. Autonom. Syst., vol. 61, no. 10, pages 1116–1130, 2013. (referred on page 15)

[Lauri *et al.* 2020] M. Lauri, J. Pajarinen, J. Peters et S. Frintrop. *Multi-sensor next-best-view planning as matroid-constrained submodular maximization*. IEEE Robot. Autonom. Lett., vol. 5, no. 4, pages 5323–5330, 2020. (referred on page 33)

[LaValle & Kuffner Jr 2001] S.M. LaValle et J.J. Kuffner Jr. *Randomized Kinodynamic Planning*. Int. J. Robot. Res., vol. 20, no. 5, pages 378–400, 2001. (referred on page 117)

[LaValle 1998] S.M. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. 1998. (referred on page 25)

[Lawler 1985] E.L. Lawler. *The traveling salesman problem: a guided tour of combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics, 1985. (referred on page 141)

[Leonard *et al.* 2007] N.E. Leonard, D.A. Paley, F. Lekien, R. Sepulchre, D.M Fratantoni et R.E. Davis. *Collective Motion, Sensor Networks, and Ocean Sampling*. Proc. of the IEEE, vol. 95, no. 1, pages 48–74, 2007. (referred on pages 1 and 115)

[Leutenegger *et al.* 2015] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart et P. Furgale. *Keyframe-based visual–inertial odometry using nonlinear optimization*. Int. J. Robot. Res., vol. 34, no. 3, pages 314–334, 2015. (referred on page 10)

[Levenberg 1944] K. Levenberg. *A method for the solution of certain non-linear problems in least squares*. Quarterly of applied mathematics, vol. 2, no. 2, pages 164–168, 1944. (referred on page 13)

[Lin & Kernighan 1973] S. Lin et B.W. Kernighan. *An effective heuristic algorithm for the traveling-salesman problem*. Operations research, vol. 21, no. 2, pages 498–516, 1973. (referred on pages 45 and 143)

[Liu *et al.* 2013] Y. Liu, X. Fan et H. Zhang. *A Fast Map Merging Algorithm in the Field of Multirobot SLAM*. In The Scientific World Journal, 2013. (referred on page 16)

[Loizou 2017] S.G. Loizou. *The navigation transformation*. IEEE Trans. Robot., vol. 33, no. 6, pages 1516–1523, 2017. (referred on page 23)

[Lorensen & Cline 1987] W.E. Lorensen et H.E. Cline. *Marching cubes: A high resolution 3D surface construction algorithm*. In ACM Trans. Graph., volume 21, pages 163–169, 1987. (referred on pages 15, 18, 39, 47, 50, 59, 66 and 87)

[Lowe 2004] D.G. Lowe. *Distinctive image features from scale-invariant keypoints*. Int. J. Comput. Vision, vol. 60, no. 2, pages 91–110, 2004. (referred on page 10)

[Lynen *et al.* 2013] S. Lynen, M.W. Achtelik, S. Weiss, M. Chli et R. Siegwart. *A robust and modular multi-sensor fusion approach applied to mav navigation*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 3923–3929. IEEE, 2013. (referred on page 10)

[Mannucci *et al.* 2017] A. Mannucci, S. Nardi et L. Pallottino. *Autonomous 3D Exploration of Large Areas: A Cooperative Frontier-Based Approach*. In Proc. Int. Conf. Model. Simul. Auton. Systems, pages 18–39, 2017. (referred on pages 33 and 118)

[Millane *et al.* 2018] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart et C. Cadena. *C-blox: A Scalable and Consistent TSDF-based Dense Mapping Approach*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 995–1002, 2018. (referred on pages 16, 17 and 117)

[Miller *et al.* 2020] I.D. Miller, F. Cladera, A. Cowley, S.S. Shivakumar, E.S. Lee, L. Jarin-Lipschitz, A. Bhat, N. Rodrigues, A. Zhou, A. Cohen, A. Kulkarni,

J. Laney, C.J. Taylor et V. Kumar. *Mine Tunnel Exploration Using Multiple Quadrupedal Robots*. IEEE Robot. Autonom. Lett., vol. 5, no. 2, pages 2840–2847, 2020. (referred on page 1)

[Monica & Aleotti 2018] R. Monica et J. Aleotti. *Contour-based next-best view planning from point cloud segmentation of unknown objects*. Auton. Robot., vol. 42, no. 2, pages 443–458, 2018. (referred on pages 30, 41 and 118)

[Mouaddib *et al.* 2019] E. Mouaddib, G. Caron, D. Groux-Leclet et F. Morbidi. *Le patrimoine «in silico». Exemple de la cathédrale d'Amiens*. In Situ. Revue des patrimoines, no. 39, 2019. (referred on page 2)

[Mur-Artal & Tardós 2017] R. Mur-Artal et J. Tardós. *ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras*. IEEE Trans. Robot., vol. 33, no. 5, pages 1255–1262, 2017. (referred on pages 40 and 59)

[Nemhauser *et al.* 1978] G.L. Nemhauser, L.A. Wolsey et M.A. Fisher. *An Analysis of Approximations for Maximizing Submodular Set Functions — I*. Math. Program., vol. 14, pages 265–294, 1978. (referred on pages 61 and 81)

[Newcombe *et al.* 2011a] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges et A.W. Fitzgibbon. *KinectFusion: Real-time dense surface mapping and tracking*. In IEEE Int. Symp. Mixed Augmen. Reality, volume 11, pages 127–136, 2011. (referred on pages 15, 18, 39 and 116)

[Newcombe *et al.* 2011b] R.A. Newcombe, S.J. Lovegrove et A.J. Davison. *DTAM: Dense tracking and mapping in real-time*. In Proc. IEEE Int. Conf. Comp. Vis., pages 2320–2327. IEEE, 2011. (referred on page 9)

[Nguyen *et al.* 2012] C.V. Nguyen, S. Izadi et D. Lovell. *Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking*. In Proc. 2nd IEEE Int. Conf. 3D Imaging, Model. Proc. Visual. & Transm., pages 524–530, 2012. (referred on pages 40, 47, 59, 66 and 78)

[Nießner *et al.* 2013] M. Nießner, M. Zollhöfer, S. Izadi et M. Stamminger. *Real-time 3D reconstruction at scale using voxel hashing*. ACM Trans. Graphic, vol. 32, no. 6, pages 1–11, 2013. (referred on pages 15 and 117)

[Oleynikova *et al.* 2017] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart et J. Nieto. *Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV*

planning. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 1366–1373, 2017. (referred on pages 16, 31, 109 and 117)

[Oleynikova *et al.* 2020] H. Oleynikova, C. Lanegger, Z. Taylor, M. Pantic, A. Millane, R. Siegwart et J. Nieto. *An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments*. J. Field Robot., vol. 37, no. 4, pages 642–666, 2020. (referred on pages 40, 59 and 78)

[Olson 2011] E. Olson. *AprilTag: A robust and flexible visual fiducial system*. In Proc. IEEE Int. Conf. Robot. Automat., pages 3400–3407. IEEE, 2011. (referred on pages 95 and 136)

[Papachristos *et al.* 2017] C. Papachristos, S. Khattak et K. Alexis. *Uncertainty-aware receding horizon exploration and mapping using aerial robots*. In Proc. IEEE Int. Conf. Robot. Automat., pages 4568–4575. IEEE, 2017. (referred on pages 31 and 109)

[Papachristos *et al.* 2019] C. Papachristos, M. Kamel, M. Popović, S. Khattak, A. Bircher, H. Oleynikova, T. Dang, F. Mascarich, K. Alexis et R. Siegwart. *Autonomous Exploration and Inspection Path Planning for Aerial Robots Using the Robot Operating System*. In A. Koubaa, editor, Robot Operating System (ROS), pages 67–111. Springer, 2019. (referred on pages 30 and 117)

[Pfister *et al.* 2000] H. Pfister, M. Zwicker, J. Van Baar et M. Gross. *Surfels: Surface elements as rendering primitives*. In ACM Trans. Graph., pages 335–342, 2000. (referred on page 13)

[Pham *et al.* 2019] Q.H. Pham, B.S. Hua, T. Nguyen et S.K. Yeung. *Real-time progressive 3D semantic segmentation for indoor scenes*. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1089–1098. IEEE, 2019. (referred on page 16)

[Piegl & Tiller 1996] L. Piegl et W. Tiller. The nurbs book. Springer Science & Business Media, 1996. (referred on page 14)

[Pito 1996] R. Pito. *A Sensor-Based Solution to the "Next Best View" Problem*. In Proc. 13th Int. Conf. Pattern Recogn., volume 1, pages 941–945, 1996. (referred on page 29)

[Punnen 2007] A.P. Punnen. *The Traveling Salesman Problem: Applications, Formulations and Variations*. In G. Gutin et A.P. Punnen, editors, The Traveling Salesman

Problem and Its Variations, volume 12, pages 1–28. Springer, 2007. (referred on pages 56 and 141)

[Quan & Lan 1999] L. Quan et Z. Lan. *Linear n-point camera pose determination*. IEEE Trans. Pattern Anal., vol. 21, no. 8, pages 774–780, 1999. (referred on page 10)

[Rehder *et al.* 2016] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann et R. Siegwart. *Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes*. In Proc. IEEE Int. Conf. Robot. Automat., pages 4304–4311. IEEE, 2016. (referred on page 136)

[Respall *et al.* 2021] V.M. Respall, D. Devitt, R. Fedorenko et A. Klimchik. *Fast Sampling-based Next-Best-View Exploration Algorithm for a MAV*. In Proc. IEEE Int. Conf. Robot. Automat., pages 89–95. IEEE, 2021. (referred on pages 30 and 117)

[Rigo 2009] M. Rigo. *Théorie des graphes*. Université de Liège, 2009. (referred on page 139)

[Ronneberger *et al.* 2015] O. Ronneberger, P. Fischer et T. Brox. *U-net: Convolutional networks for biomedical image segmentation*. In International Conference on Medical Image Computing and Computer-Assisted Intervention, pages 234–241. Springer, 2015. (referred on page 110)

[Rosten & Drummond 2006] Edward Rosten et Tom Drummond. *Machine learning for high-speed corner detection*. In Proc. Eur. Conf. Comp. Vis., pages 430–443. Springer, 2006. (referred on page 10)

[Rublee *et al.* 2011] E. Rublee, V. Rabaud, K. Konolige et G. Bradski. *ORB: An efficient alternative to SIFT or SURF*. In Proc. IEEE Int. Conf. Comp. Vis., pages 2564–2571. IEEE, 2011. (referred on page 10)

[Sanfourche *et al.* 2013] M. Sanfourche, V. Vittori et G. Le Besnerais. *eVO: A realtime embedded stereo odometry for MAV applications*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 2107–2114, 2013. (referred on pages 10, 95, 96, 108 and 124)

[Scaramuzza & Fraundorfer 2011] D. Scaramuzza et F. Fraundorfer. *Visual odometry [tutorial]*. IEEE Rob. Autom. Mag., vol. 18, no. 4, pages 80–92, 2011. (referred on page 9)

[Schmid *et al.* 2020] L.M. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart et J. Nieto. *An Efficient Sampling-based Method for Online Informative Path Planning in Unknown Environments*. IEEE Robot. Autonom. Lett., vol. 5, no. 2, pages 1500–1507, 2020. (referred on pages 32, 51, 70 and 118)

[Schöps *et al.* 2019] T. Schöps, T. Sattler et M. Pollefeys. *Surfelmeshing: Online surfel-based mesh reconstruction*. IEEE Trans. Pattern Anal., vol. 42, no. 10, pages 2494–2507, 2019. (referred on page 14)

[Selin *et al.* 2019] M. Selin, M. Tiger, D. Duberg, F. Heintz et P. Jensfelt. *Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments*. IEEE Robot. Autonom. Lett., vol. 4, no. 2, pages 1699–1706, 2019. (referred on pages 30, 31 and 117)

[Sengupta & Sturgess 2015] S. Sengupta et P. Sturgess. *Semantic octree: Unifying recognition, reconstruction and representation via an octree constrained higher order MRF*. In Proc. IEEE Int. Conf. Robot. Automat., pages 1874–1879. IEEE, 2015. (referred on page 16)

[Shade *et al.* 1998] J. Shade, S. Gortler, L.W. He et R. Szeliski. *Layered depth images*. In ACM Trans. Graph., pages 231–242, 1998. (referred on page 13)

[Shah *et al.* 2020] K. Shah, G. Ballard, A. Schmidt et M. Schwager. *Multidrone aerial surveys of penguin colonies in Antarctica*. Sci. Robot., vol. 5, no. 47, page eabc3000, 2020. (referred on pages 1 and 115)

[Shi *et al.* 1994] J. Shi*et al.* *Good features to track*. In Proc. IEEE Conf. Comp. Vis. Pattern Recogn., pages 593–600. IEEE, 1994. (referred on page 10)

[Short *et al.* 2016] A. Short, Z. Pan, N. Larkin et S. Van Duin. *Recent progress on sampling based dynamic motion planning algorithms*. In IEEE/ASME Int. Conf. Adv. Intell. Mechatron. AIM, pages 1305–1311. IEEE, 2016. (referred on pages 24 and 26)

[Siegwart *et al.* 2011] R. Siegwart, I.R. Nourbakhsh et D. Scaramuzza. Introduction to autonomous mobile robots. MIT press, 2011. (referred on page 130)

[Singh *et al.* 2010] A. Singh, F. Ramos, H.D. Whyte et W.J. Kaiser. *Modeling and decision making in spatio-temporal processes for environmental surveillance*. In Proc. IEEE Int. Conf. Robot. Automat., pages 5490–5497. IEEE, 2010. (referred on page 109)

[Song & Jo 2017] S. Song et S. Jo. *Online Inspection Path Planning for Autonomous 3D Modeling using a Micro-Aerial Vehicle*. In Proc. IEEE Int. Conf. Robot. Automat., pages 6217–6224, 2017. (referred on pages 32, 51 and 118)

[Song & Jo 2018] S. Song et S. Jo. *Surface-based Exploration for Autonomous 3D Modeling*. In Proc. IEEE Int. Conf. Robot. Automat., pages 4319–4326, 2018. (referred on pages 32, 51, 70 and 118)

[Song *et al.* 2021] S. Song, D. Kim et S. Choi. *View Path Planning via Online Multiview Stereo for 3-D Modeling of Large-Scale Structures*. IEEE Trans. Robot., 2021. (referred on pages 32 and 118)

[Sturm & Ramalingam 2011] P. Sturm et S. Ramalingam. Camera models and fundamental concepts used in geometric computer vision. Now Publishers Inc, 2011. (referred on page 133)

[Tomasi & Kanade 1991] C. Tomasi et T. Kanade. *Detection and tracking of point*. Int. J. Comput. Vision, vol. 9, pages 137–154, 1991. (referred on page 10)

[Trifonov 2013] D.S. Trifonov. *Real-time high resolution fusion of depth maps on GPU*. In IEEE/ACM Int. Conf. Comput.-Aided Des. Dig. Tech. Pap., pages 441–442. IEEE, 2013. (referred on page 18)

[Vasquez-Gomez *et al.* 2014a] J.I. Vasquez-Gomez, L.E. Sucar et R. Murrieta-Cid. *View planning for 3D object reconstruction with a mobile manipulator robot*. In Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., pages 4227–4233, 2014. (referred on page 30)

[Vasquez-Gomez *et al.* 2014b] J.I. Vasquez-Gomez, L.E. Sucar, R. Murrieta-Cid et E. Lopez-Damian. *Volumetric Next-best-view Planning for 3D Object Reconstruction with Positioning Error*. Int. J. Adv. Robot. Syst., vol. 11, no. 10, page 159, 2014. (referred on pages 29 and 118)

[Vespa *et al.* 2018] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly et S. Leutenegger. *Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping*. IEEE Robot. Autonom. Lett., vol. 3, no. 2, pages 1144–1151, April 2018. (referred on page 16)

[Vineet *et al.* 2015] V. Vineet, O. Miksik, M. Lidegaard, M. Nießner, S. Golodetz, V.A. Prisacariu, O. Kähler, D.W. Murray, S. Izadi, P. Pérez*et al.* *Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction*. In Proc. IEEE Int. Conf. Robot. Automat., pages 75–82. IEEE, 2015. (referred on page 16)

[Vutetakis & Xiao 2019] D.G. Vutetakis et J. Xiao. *An Autonomous Loop-Closure Approach for Simultaneous Exploration and Coverage of Unknown Infrastructure Using MAVs*. In Proc. IEEE Int. Conf. Robot. Automat., pages 2988–2994, 2019. (referred on page 109)

[Wang *et al.* 2019] C. Wang, D. Zhu, T. Li, M.Q.H. Meng et C.W. de Silva. *Efficient autonomous robotic exploration with semantic road map in indoor environments*. IEEE Robot. Autonom. Lett., vol. 4, no. 3, pages 2989–2996, 2019. (referred on page 31)

[Whelan *et al.* 2012] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard et J. McDonald. *Kintinuous: Spatially Extended KinectFusion*. In RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras, Sydney, Australia, 2012. (referred on page 15)

[Williams & Rasmussen 2006] C.K. Williams et C.E. Rasmussen. Gaussian processes for machine learning, volume 2. MIT press Cambridge, MA, 2006. (referred on page 31)

[Williams *et al.* 2009] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid et J. Tardós. *A comparison of loop closing techniques in monocular SLAM*. Robot. Autonom. Syst., vol. 57, no. 12, pages 1188–1197, 2009. (referred on page 108)

[Wilson 1973] R.J. Wilson. *An introduction to matroid theory*. The American Mathematical Monthly, vol. 80, no. 5, pages 500–525, 1973. (referred on pages 61 and 81)

[Woeginger 2003] G.J. Woeginger. *Exact algorithms for NP-hard problems: A survey*. In Combinatorial optimization—eureka, you shrink!, pages 185–207. Springer, 2003. (referred on page 142)

[Xie *et al.* 2012] W.C. Xie, X.F. Zou, J.D. Yang et J.B. Yang. *Iteration and optimization scheme for the reconstruction of 3D surfaces based on non-uniform rational B-splines*. Computer-Aided Design, vol. 44, no. 11, pages 1127–1140, 2012. (referred on page 14)

[Xu *et al.* 2021] Z. Xu, D. Deng et K. Shimada. *Autonomous UAV Exploration of Dynamic Environments Via Incremental Sampling and Probabilistic Roadmap*. IEEE Robot. Autonom. Lett., vol. 6, no. 2, pages 2729–2736, 2021. (referred on pages 31 and 117)

[Yamauchi 1997] B. Yamauchi. *A frontier-based approach for autonomous exploration*. In IEEE International Symposium on Computational Intelligence in Robotics and Automation, volume 97, page 146, 1997. (referred on page 30)

[Yoder & Scherer 2016] L. Yoder et S. Scherer. *Autonomous Exploration for Infrastructure Modeling with a Micro Aerial Vehicle*. In D.S. Wettergreen et T.D. Barfoot, editors, Field and Service Robotics: Results of the 10th Int. Conf., pages 427–440. Springer, 2016. (referred on pages 30, 70 and 118)

[Zeng *et al.* 2017] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao et T. Funkhouser. *3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions*. In Proc. IEEE Conf. Comp. Vis. Pattern Recogn., pages 1802–1811, 2017. (referred on pages 16, 47, 66 and 96)

[Zheng *et al.* 2019] L. Zheng, C. Zhu, J. Zhang, H. Zhao, H. Huang, M. Niessner et K. Xu. *Active scene understanding via online semantic reconstruction*. In Computer Graphics Forum, volume 38, pages 103–114. Wiley Online Library, 2019. (referred on page 110)

Résumé : Dans un contexte de recherche archéologique ou de préservation du patrimoine, la reconstruction 3D de site à l'aide de capteur de haute qualité, reste un processus lent, complexe et onéreux et chronophage. Avec l'émergence des technologies robotiques, l'utilisation de robots mobiles autonomes permet la production de modèles 3D complets et précis, dans un temps réduit, tout en accédant aux zones les plus difficiles. L'objectif de cette thèse est de mettre au point des méthodes de reconstruction 3D d'environnements inconnus, à l'aide de plusieurs robots mobiles. Les robots, équipés de caméras stéréoscopiques explorent l'environnement et révèlent dans le modèle progressivement reconstruit, des éléments de surface incomplets. Des poses candidates sont alors générées pour les compléter, sont groupées en zones d'intérêt, et sont ensuite assignées aux robots à l'aide d'un algorithme glouton. Cet algorithme est un planificateur de chemin de type Next-Best-View, basé sur un critère surfacique, où le problème du voyageur de commerce est résolu itérativement. Un planificateur probabiliste permet enfin aux robots de trouver leur chemin dans l'espace libre. La méthode a été déclinée en trois architectures : mono-robot dans un cadre préliminaire, multi-robot centralisée pour réduire les temps de reconstruction, puis distribuée pour en augmenter la robustesse. Une validation par simulation et des tests en environnement réel ont été réalisés, pour des robots aériens et roulants, afin de montrer l'efficacité des trois architectures.
Mots clés : Systèmes multi-robots, planification de mouvement, planification Next-Best-View, reconstruction 3D.

Abstract: In archaeology and cultural heritage, the 3D modelling of large-scale structures using high-quality sensors, remains time-consuming, complex, and expansive process. In present age of robotics, a new generation of scanning systems based on mobile robots, could address this challenge, improving efficiency, flexibility and responsiveness. This PhD thesis considers the problem of 3D reconstruction of an unknown environment, with a team of cooperative vehicles. The robots equipped with forward-facing stereo cameras, explore the environment, uncover discrete Incomplete Surface Elements (ISEs) in the volumetric map, and generate candidate viewpoints to scan them. These areas of interest are greedily assigned to the robots using a Next-Best-View approach, where the visit is planned by iteratively solving a Traveling Salesman Problem. Then, a sampling-based planner is used to compute obstacle-free paths using the volumetric map. A single-robot architecture has been first designed, which leverages the 3D surface representation of volumetric map for planning. This architecture has been extended to a multi-robot system with a single base station, in order to accelerate the scanning process. Finally, a distributed architecture has been presented and discussed to increase the robustness of the multi-robot system. Extensive numerical and real-world experiments with multiple aerial and ground robots have been conducted to validate the prosposed architectures in challenging environments.
Keywords: Multi-robot systems, Motion planning, Next-Best-View planning, 3D reconstruction.