ONLINE ROBOT EXPLORATION AND PERCEPTUAL COVERAGE OF LARGE-SCALE UNKNOWN ENVIRONMENTS: SCALABLE OPTIMIZATION AND GENERALIZED SOFTWARE FRAMEWORK

by

David Vutetakis

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2023

ABSTRACT

DAVID VUTETAKIS. Online Robot Exploration and Perceptual Coverage of Large-Scale Unknown Environments: Scalable Optimization and Generalized Software Framework. (Under the direction of DR. JING XIAO)

This dissertation addresses the problem of non-myopic online exploration and visual sensor coverage of large-scale unknown environments using an autonomous robot. We introduce a novel perception roadmap, referred to as the Active Perception Network (APN), that represents a connected configuration space over a concurrently built spatial map. The APN is modeled by a hierarchical topological hypergraph that equips a robot with an understanding of how to traverse throughout a concurrently built spatial map, and facilitates predictive reasoning on the expected visible information of the environment from untraversed regions of the map.

As new information is added to the map during exploration, the APN is iteratively updated by an adaptive algorithm entitled Differential Regulation (DFR), which applies difference-aware strategies to constrain the complexity of each update relative to the size of changed map information, independent of its total size. DFR employs a view sampling-based strategy to expand and refine traversability knowledge as map knowledge increases, using a novel frontier-based approach to evaluate information gain and guide the sampling and pruning of views within the APN. The APN serves as a knowledge model which can be applied for graph-based exploration planning. An evolutionary planner, designated as APN-P, leverages the hierarchical representation of the APN to perform non-myopic exploration planning that dynamically adapts to the changing map and APN states.

This dissertation further presents a software development framework, Active Perception for Exploration, Mapping, and Planning (APEXMAP), that addresses the unique and non-trivial software engineering challenges inherent to online exploration and active perception tasks. APEXMAP provides a generalized modular framework for these challenges, which is made open source for the benefit of the research community.

ACKNOWLEDGMENTS

This dissertation was made possible only through the support, guidance, and inspiration I received from a vast network of individuals. I would first like to underscore my sincere gratitude to my advisor, Prof. Jing Xiao, and her unwavering support and commitment to my success. Her guidance and direction has been a cornerstone to my growth as a researcher and as an individual, and I am grateful to have had the privilege of her mentorship.

I would also like to recognize my father, Dr. Dave Vutetakis, for his dedicated support and encouragement. He has been an inspiration to me from a young age and his guidance has been invaluable to my growth and success. I also extend my great appreciation to my mother, Elizabeth Vutetakis, who has unconditionally supported and encouraged me. I am further thankful for my lab-mates Saurav Agarwal, Sayantan Datta, Sterling McLeod, Huitan Mao, and many others for their support and the friendships that have developed throughout my academic journey.

I extend my gratitude and appreciation to my committee members, Prof. Min Shin, Prof. Srinivas Akella, and Prof. Aidan Browne for their contributions to my research and personal growth. I also acknowledge the extensive support I have received from many other faculty members and peers at the University of North Carolina at Charlotte, and the support and funding provided by the Department of Computer Science and the Graduate School.

Finally, and most importantly, I thank the Lord God and His unconditional love, for He is the true source of my success and prosperity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AoV	Angle of View.
APEXMAP	Active Perception for Exploration, Mapping, and Planning.
APN	Active Perception Network.
APN-P	APN Planner.
AUV	Autonomous Underwater Vehicle.
BA	Bundle Adjustment.
BRIEF	Binary Robust Independent Elementary Features.
CF	Closest Frontier.
DFR	Differential Regulation.
FAST	Features from Accelerated Segment Test.
FCL	Flexible Collision Library.
FEOTSP	Fixed-Ended Open Traveling Salesman Problem.
FoV	Field of View.
IG	Information Gain.
IMU	Inertial Measurement Unit.
INS	Inertial Navigation System.
IPP	Informative Path Planning.
KF	Kalman Filter.
LoD	Level of Detail.

MAV	Micro Aerial Vehicle.
MLESAC	Maximum Likelihood Estimation SAmple and Consensus.
MRPT	Mobile Robot Programming Toolkit.
NBV	Next-Best-View.
OBB	Oriented Bounding Box.
OMPL	Open Motion Planning Library.
ORB	Oriented FAST and Rotated BRIEF.
OSCP	Online Sensor Coverage Planning.
PF	Particle Filter.
PMX	Partially Mapped Crossover.
POMDP	Partially Observable Markov Decision Process.
PRM	Probabilistic Roadmap.
RANSAC	RANdom Sample Consensus.
ROS	Robot Operating System.
RRG	Rapidly-exploring Random Graphs.
RRT	Rapidly-exploring Random Tree.
SDE	Stochastic Differential Equation.
SDF	Signed Distance Field.
SE	Software Engineering.
SIFT	Scale-Invariant Feature Transform.

SLAM	Simultaneous Localization and Mapping.
SRT	Sensor-based Random Tree.
SURF	Speeded Up Robust Features.
ToF	Time of Flight.
TSP	Traveling Salesman Problem.
UAV	Unmanned Aerial Vehicle.
UGV	Unmanned Ground Vehicle.
UML	Unified Modeling Language.

CHAPTER 1: INTRODUCTION

For mobile robots, such as micro aerial vehicles (MAV) or unmanned ground vehicles (UGV), a model of the *operational environment* is essential to the navigation and planning tasks involved in nearly any autonomous application. However, for many practical applications, the robot must operate in an unknown environment, for which an *a priori* model of the environment is unavailable or unreliable. This includes applications related to search and rescue, 2D terrain surveying [1, 2], 3D object modelling [3, 4], infrastructure modeling and inspection [5, 6, 7], and many others [8].

When an environment model is unavailable before operation, it must be built from sensor information acquired online by the process referred to as *map learning*, or simply *mapping*. This is an incremental task in which onboard sensors are used to perceive parts of the environment surrounding the robot, using the robot pose to geometrically align and integrate these measurements into a consistent global map model. Environment mapping is considered a passive process which is only concerned about how to build the map given the immediate input sensor data, but does not consider the navigation and control of the robot that dictates the information content of the particular inputs.

In contrast, autonomous mapping requires the integration of navigation and control to the map building process. This is formulated as a decision-making process which attempts to predict the new information expected to be added to the map upon execution of a particular control action. The optimal action is determined according to criteria related to the cost of the action (e.g. time or distance traveled), the expected increase of total map knowledge from the sensor measurements produced by the action, and potentially subject to additional application-specific objectives or constraints. This task has been described in the literature by many names including active exploration [9], adaptive exploration [10, 11], active mapping [12], informative path planning (IPP) [13, 14], or simply robot exploration [15] in addition to many others. For clarity, this dissertation will primarily adhere to the term *online exploration* in reference the general process where autonomous decision-making is applied for the purpose of online environment exploration and map learning.

Most early research for online exploration was applied to Unmanned Ground Vehicles (UGV), but recent advancements of lightweight and low-cost Micro Aerial Vehicles (MAV) has made aerial systems increasingly practical for these purposes. However, MAV systems also introduce a number of additional challenges, such as limited power and payload capacity that leads to restrictive flight time and constrained computational and sensing resources. They are also inherently unstable and require robust control systems for safe and accurate operation. Methodologies designed for ground vehicles often cannot be effectively adapted or extended to aerial vehicles, given the increased problem dimensionality and complexity combined with reduced onboard computation resources. While significant research progress has been made using MAVs for online exploration in a short timeframe, less works have concentrated on applications for MAVs compared to ground vehicles, and there are a variety of unsolved or understudied challenges that remain in general.

1.1 Online Sensor Coverage Planning (OSCP) Problems

In its general form, the goal of online exploration is to maximize some measure of environment knowledge, or *information gain*, using a minimal amount of time or energy. However, different tasks or applications may impose additional objectives, constraints, or termination criteria.

For example, subterranean exploration tasks may seek to maximize knowledge of all traversable free-space of an underground cave or tunnel system. A search and rescue application seeks the same information, but may allow termination once a predetermined target is located. In contrast, a structural modeling application seeks maximum information regarding the visible surface geometries or features to support 3D modeling tasks, terminating once maximum surface coverage is reached without regard to the free-space coverage. In general, the differences between formulations can be classified according to the particular features of interest (e.g. free space or surfaces), and the termination conditions that define successful completion.

This dissertation specifically focuses online exploration applications for the purpose of dense 3D modeling of an *a priori* unknown structure or environment using a resource-constrained MAV or other mobile robot. This problem will be referred to as *online sensor coverage planning* (OSCP). The robot is assumed to be equipped with a spatial depth sensor able to perceive the 3D surface geometry of the environment, where a surface point is considered *covered* once it has been observed by the sensor. The OSCP objective is to achieve maximum visual surface coverage of a bounded volume, subject to time or energy constraints. Thus, the features of interest are the visible surfaces, and the termination conditions specify maximum observation of these features.

Naturally, OSCP cannot be solved directly or offline due to the lack of a priori knowledge, and can only be solved online in an incremental fashion. This leads to the distinction between the *global coverage problem* and the *incremental exploration problem*, where the global problem corresponds to the discrete goal state where maximum coverage is achieved, and the incremental problem corresponds to the means of achieving this. The incremental problem represents an iterative action selection problem: given the current incomplete knowledge, determine the optimal control action expected to most efficiently lead to completion of the global objective. Hence, the global coverage state is determined by the cumulative sensor observations acquired over the traversed path, and a path that maximizes the global coverage state represents a solution to the global coverage problem.

A general problem model can be formulated using a partially observable Markov decision process (POMDP), which are known to be highly intractable other than for very small problem sizes [16]. Consequentially, approaches seek practical solutions with reasonable quality sufficient for the targeted application. Still, most existing approaches rely on problem simplifications, operational assumptions, and greedy planning strategies, and are unable to achieve acceptable performance for many applications or conditions. Further research is needed for scalable and efficient planning methods that are practical for a broader range of applications, particularly in large-scale and complex environments.

1.2 Dissertation Overview

The remaining chapters of this dissertation are organized as follows: Chapter 2 provides a literature review of the various components involved in an online exploration system, as well as key approaches towards the problem. A formalized description and formulation of the problem addressed by this dissertation is contained in Chapter 3, and an overview of the presented approach in Chapter 4. Chapters 5, 6, and 7 present the primary aspects of the methodology, and a detailed performance analysis and comparison is provided in Chapter 8. Finally, a software development framework, APEXMAP, is described in Chapter 9, with details regarding its role in the implementation of our methodology. Conclusions and contributions are given Chapter 10, with a description of potential future work.

CHAPTER 2: LITERATURE REVIEW

Online exploration belongs to the more general class of problems referred to as active perception, where intelligent control strategies are applied to the process of remote sensing and data acquisition [17]. The intuitive motivation for active perception is derived from the limited capabilities and noisy measurements inherent to physical sensors. Remote sensing technologies such as depth cameras or LiDAR can only capture a limited amount of information about the environment from a static pose, due to intrinsic properties of the sensor like limited field-of-view and range of measurement, or perspective occlusions by obstacles in the environment. The purpose of an actuated robotic system is to overcome sensor limitations and occlusions by allowing the position and orientation of the sensor to be dynamically moved throughout the environment. Thus, online exploration problems must be addressed in the context of both the sensing modality and the robot platform on which they are mounted, including all other aspects needed to for robot motion planning and control.

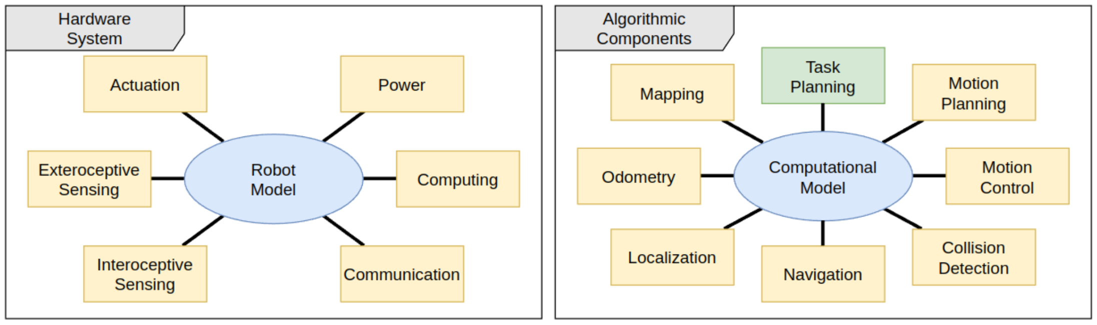


Figure 2.1: Primary hardware and computational components comprised by online exploration systems.

The following sections of this chapter will provide a review of the relevant background theory related to the functional components needed for active perception and online exploration, which are summarized by the depiction in Figure 2.1. This includes a review of the different hardware components necessary for a robotic system to perform active perception tasks, provided in Section 2.1 and 2.2, respectively. A review of mapping methodologies and other approaches for storage of perception knowledge is contained in Section 2.3, and relevant theory related to localization and odometry in Section 2.4. A review of key works and the current state-of-the-art for online exploration tasks is provided in Section 2.5, followed by a summary of current limitations and challenges that warrant further research.

2.1 Autonomous Agent

The autonomous agent describes the integrated robotic system which is deployed in the operational environment for an exploration application. It is comprised of the electromechanical components for sensing and locomotion in the environment, and computing resources for data processing and autonomous operation. In the context of this research, typical classifications of the agent include unmanned ground vehicles (UGV) [18], unmanned aerial vehicles (UAV) [19], unmanned surface vehicle (USV) [20, 21], and autonomous underwater vehicles (AUV) [22], while heterogeneous systems have also been considered [23]. In this work we focus on applications involving aerial vehicles, but the underlying principles and methodologies can be effectively generalized to other vehicular systems.

2.1.1 Micro Aerial Vehicle (MAV)

A Micro Aerial Vehicle (MAV) is class of small and lightweight UAV, typically made to be man-portable. These include commonly consumer available multirotor configurations like quad-rotors, colloquially referred to as *drones.* These are versatile class of robots capable of precise maneuvering in 3D space and deployment in a wide range of environments. These characteristics make them a particularly attractive solution for exploration and volumetric reconstruction, but are also characterized by many unique constraints. These constraints often introduce unique challenges not experienced by other robotic systems such as mobile ground vehicles.

The nature of airborne operation results in highly restricted payload capacities and limited flight times. This drastically reduces the available onboard computation and sensing hardware. Sensors must be both lightweight and low-power. The quality of these sensors is significantly reduced compared to their ground-based counterparts. The reduction of computing resources further limits the ability to process large amounts of sensing data in real-time. One approach is to wirelessly transmit data to a ground station with more computing power, but communication bandwidth and latency can introduce further challenges. This option also may not always be available if prior access to the environment is not permissive.

2.2 Sensing Modalities

Sensing technologies provide the ability of the agent to measure and interpret physical stimuli from the environment or regarding the agents own dynamic state. Sensors can be classified as either interoceptive or exteroceptive, according to the source of their stimuli [24]. Interoceptive sensors measure parameters of the agents inertial state, such as linear acceleration or angular velocity. Exteroceptive sensors measure parameters related to the external environment. The following subsections will provide a brief overview of these classes of sensing and their usage for exploration.

2.2.1 Interoceptive Sensing

Interoceptive (also referred as proprioceptive) sensors provide information regarding the vehicles inertial state which is essential for tasks like motion control and state estimation. Multiple sensor modalities are typically packaged together in an integrated sensor suite known as an Inertial Measurement Unit (IMU), or Inertial Navigation System (INS). These include a multi-axis accelerometer which measures linear acceleration along each orthogonal axis of the agents body frame of reference.

A multi-axis gyroscope similarly measures angular orientation and angular rate of each axis. These measurements can be mathematically integrated over time to resolve additional measurement units like velocity and displacement, which can be used to approximate the position by a technique known as dead-reckoning. However, small integration errors quickly accumulate over time which makes this approach ineffective for accurate positioning over extended time periods.

2.2.2 Exteroceptive Sensing

Exteroceptive sensing receives stimulus from the external environment such as visible light or sound, and allow perception of the scenes visual or geometric characteristics. The most common exteroceptive sensors employed on a lightweight MAV include cameras, laser scanners (LiDAR), RGB-D, ultrasonic, and optical flow [24]. Cameras may be either monocular or a stereo pair. These are passive sensors, meaning they only measure preexisting stimuli from the environment (i.e. light). They are generally lightweight, consume low power, and are able to discern textures and other visual features from the environment. Similarly, optical flow sensors measure emitted light from the environment, but rather than store images, they use the relative change in visual data over time to resolve image velocities or displacement.

LiDAR, RGB-D, and ultrasonic sensors are considered active devices - they introduce stimuli to the environment to measure its interaction [24]. LiDAR emits a focused a laser in a scanning motion, measuring the time of flight (ToF) to resolve distance. These sensors generally have the highest weight and consume the most power, but obtain very dense and accurate measurements. Similarly, RGB-D sensors emit infrared lasers on the scene, but do not directly measure distance. Instead, a structured light pattern is produced, which is observed by a camera sensor. The relative dispersion of the pattern can be used to resolve depth, which is combined with the RGB image data. Lastly, ultrasonic rangefinders emit sound waves, measuring the distance to obstacles using the ToF technique. These generally consume the lowest power and are very lightweight. A major disadvantage results from the dispersion of the sound wave, which is unable to resolve accurate point measurements - the distance reading could come from anywhere within its cone of reception.

For the purpose of collecting data to be used for volumetric exploration, the most useful sensors are cameras, LiDAR, and RGB-D [24]. Cameras can provide rich color and texture information about the environment which is often desired for realistic scene modeling, making them a popular option for data collection. However, they often have limited resolution and high computational loads associated with image processing. LiDAR data does not contain visual information like color or intensity, but may be used if only the geometric structure of the scene is required. In such cases, the weight and power consumption remains a drawback compared to camera-based methods. RGB-D sensors can capture both visual and structural information independently, but have significantly reduced measurement range compared to both LiDAR and camera-based implementations. As a result, sensor implementations are often designed to minimize these trade-off characteristics, dependent on the specific conditions of a particular application. This often results in solutions that do not robustly generalize to different conditions.

2.3 Spatial Mapping

A spatial map model refers to any means of representing the environments spatial and geometric characteristics. For online tasks, these are dynamic data structures that store an organized memory representation of spatial information collected by the agents sensors over time, allowing efficient memory and recall of prior observations. This is essential for state estimation, motion planning and control, and enables further reasoning and inference needed for informed decision making. Key challenges for this problem relate to efficient data representation and storage, updatability, scaling complexity, and uncertainty handling [25]. An IEEE standard for 2D map representation which indicates the maturity of 2D mapping approaches [26]. A similar standard for 3D map representations has yet to be established due to its increased difficulty and lack of generalized solutions.

A variety of map models have been presented in the literature, most of which can be classified as metric maps, topological maps, or semantic maps [27]. A tradeoff exists in each representation between its informativeness and space-complexity, introducing both advantages and disadvantages of each type depending on the task they are used for. In general, no map representation has been developed which is optimal for all functionality requirements, resulting in performance tradeoffs or the adoption of multiple map representations for a single application [27]. The following subsections will provide an overview of each map category and their tradeoffs. Additionally, a comparative analysis of several mapping techniques can be found in [28], and a survey on 3D maps in the context of SLAM can be found in [27].

2.3.1 Metric Map

Metric maps model the geometric space of the environment and are considered the most informative map representation. However, the increased informativeness comes at the cost of higher storage requirements and time complexity. Some of the most commonly used metric maps include occupancy grid maps, signed distance fields (SDF), mesh models, and feature maps. Multiresolution occupancy maps have been developed which contain variably-sizes voxels rather than a single fixed size, which can greatly reduce memory requirements and complexity.

Occupancy grid maps densely partition all space within a bounded volume into a discrete set of cubic volumes known as voxels. Each voxel can then be used to store useful properties of their contained volume, such as its occupancy state as either unknown, free, or occupied. The occupancy state can be represented as a discrete value, or as a probabilistic value which can account for sensor uncertainty. Given a map $\mathbf{M}$, discretized into a set of voxels $\mathbf{v} \in \mathbf{M}$ modeled as a binary random variable, the map is initialized by assinging each voxel a prior probability value (a value of 0.5

is common practice). When a new sensor measurement $z_{1:t}$ is received, the probability $P(v|z_{1:t})$ of a voxel v to be occupied given the sensor measurement is computed as shown below [29].

$$P(v|z_{1:t}) = \left[1 + \frac{1 - P(v|z_t)}{P(v|z_t)} \frac{1 - P(v|z_{1:t-1})}{P(v|z_{1:t-1})} \frac{P(v)}{1 - P(v)}\right]^{-1} \tag{2.1}$$

Conceptually, this formulation can be understood simply that as the occupancy probability of a voxel v given a new measurement z_t is dependant on the prior probability $P(v)$ and the previous estimate $P(v|z_{1:t-1})$. Each voxel is classified as one of three possible values according to:

- **occupied**: occupancy probability > prior probability
- **unknown**: occupancy probability = prior probability
- **free**: occupancy probability < prior probability

An obvious drawback to fixed-resolution occupancy maps is the large memory requirement to explicitly store each voxel, especially as the environment scale increases. An attractive alternative is through multi-resolution maps which allow the size of each voxel to vary when all of its neighbors share the same occupancy state, replacing them with a larger single voxel. It is generally true that the majority environments consists mostly of free space which can be more compactly represented using this structure. In practice, this can be implemented using efficient data structures such as quadtree [30, 31], R-tree [32], or octree [33, 34, 35].

Two particularly noteworthy open-source implementations of octree maps is through OctoMap [29] and Point Cloud Library (PCL) [36]. A brief comparison of the two frameworks is provided in [37], showing that each offer $O(1)$ complexity for random access and are capable of multiresulution queries. However, a key difference is that PCL focuses efficiency on compression needed for streaming, while OctoMap is optimized for robot navigation and exploration. This is clearly evidenced in the literature as an increasing number of robotics applications choose to incorporate OctoMap.

SDF maps are conceptually similar to occupancy maps in that they densely partition volumetric space. The key difference is that each voxel in a SDF map stores a distance metric corresponding to the distance to the nearest obstacle. This metric can be much more computationally expensive to update, but can also significantly decrease the complexity of subsequent queries operations like collision checking.

Feature maps utilize a sparse representation of geometric space, in contrast to occupancy grids or SDF maps. They store a sparse set of salient geometric features extracted from the sensor data. Intuitively, the features should be invariant to changes in viewing perspective, illumination, and other disturbances, allowing them to be uniquely queried. Some of the more commonly used features descriptors are shown below, while many others exist.

- Scale-Invariant Feature Transform (SIFT)
- Speeded Up Robust Features (SURF)
- Features from Accelerated Segment Test (FAST)
- Binary Robust Independent Elementary Features (BRIEF)
- Oriented FAST and Rotated BRIEF (ORB)

A representation of a feature map is through the use of vocabulary trees [38], which efficiently query very large datasets. The motivation behind this approach is to help reduce the memory demands for mapping large-scale environments by only storing the most significant and useful information about the scene. In general, selection of an appropriate feature descriptor is a trade-off between invariance and robustness to computational cost.

2.3.2 Topological Map

Topological maps can be conceptualized as a graph structure where the nodes store some high level semantic understanding of the world, and edges represent relative relationships between these. These maps are useful for identifying the connectivity of different regions, but do not explicitly model the geometry. Semantic maps store a hierarchical structure of semantic elements such as objects or rooms, and also omit explicit geometric modeling. These can be difficult to construct and often require large computational resources that are restrictive for online applications.

2.3.3 Semantic Map

Semantic maps provide the ability to store qualitative or quantitative knowledge and relationships about the environment, often in a more human-understandable form. A formalized model was presented in [39], and further refined in [40]. Given a mathematical description of the environment E and task domain D, a semantic map can be defined by the tuple $M_{sem} = \langle \mathcal{M}, \mathcal{L}, \mathcal{A} \rangle$, where $\mathcal{M}$ is a set of maps for E, and $\mathcal{L}$ is a set of links. $\mathcal{A}$ is a structure representing the knowledge about D, which allows for inference. In general, $\mathcal{A}$ can be arbitrarily defined with respect to a particular applications, and is generally represented using a graph-based structure, or an ontology [39]. Semantic maps are generally referenced, or anchored, to a geometric map, overlaying conceptual or abstracted knowledge such as objects or rooms [40].

2.4 Localization and Odometry

Localization is the general problem of determining the pose of the robotic agent with respect to a fixed reference frame of its operational environment. When operating in an unknown environment, particularly indoors, external systems like GPS that provide absolute pose estimation are often unavailable or unreliable. In these settings, localization must be performed relatively from onboard sensor data using the process of Simultaneous Localization and Mapping (SLAM). Conceptually, this involves using relative sensor measurements Z to build a geometrically consistent map M while concurrently computing the agents pose x with respect to the map [41]. Due to various sources of noise and uncertainty, this is modeled probabilistically as the joint posterior of the agent state x and map M at time instance k according to:

$$p(x_k, M|Z_{0:k}, U_{0:k}, x_0) \tag{2.2}$$

where x_0 is the initial known state, $Z_{0:k} = \{z_0, z_1, z_2, ..., z_k\}$ is the set of prior measurements, and $U_{0:k} = \{u_1, u_2, u_3, ..., u_k\}$ is the set of prior state transitions according to some odometry model [42]. The solution to this problem requires a motion model $p(x_k|x_{k-1}, u_k)$ which represents the probability distribution of the agents pose x_k given the previous pose estimate x_{k-1} and the previous state transition u_k. An observation model $p(z_k|x_k, m)$ is also necessary which describes the probability distribution of a measurement z_k given the agents pose x_k and the measurements location in M.

Indirect methods, also known as feature-based methods, extract visual or geometric features (also known as landmarks) from the raw sensor measurements. Direct methods have also been proposed which use the raw sensor measurements directly, but these are often too computationally expensive for online use and are less effective than feature-based methods [41, 43]. Solutions are typically derived using the recursive Bayes rule [44], with the most common approaches based on Kalman Filters (KF), Particle Filters (PF), and graph-based frameworks. Detailed surveys for SLAM based on filtering approach can be found in [45, 46].

One of the most difficult aspects of SLAM is the correspondence problem, or data association problem [41]. This is the problem of identifying corresponding features between different observations. Incorrect data association can corrupt the map and often results in catastrophic localization failure [47], and is particularly difficult for ambiguous or non-unique features, and for repeating feature patterns [44]. Without absolute reference information, finding true correspondences is not always feasible and introduces large uncertainties. Data ambiguities and outliers are often handled using techniques such as Random Sample Consensus (RANSAC) and Maximum Likelihood Estimation SAmple and Consensus (MLESAC) [48].

Noise and uncertainty inevitably accumulate with each update, causing unbounded increases of uncertainty and errors to the map and localization estimates over time. However, the relative error between features is often highly correlated, and this correlation has been shown to increase with repeated observations [42]. This correlation is a fundamental principle to formulating SLAM as a single joint estimation problem. It can be used to infer that despite high uncertainty in the absolute location of a feature, the relative relationship between features can often be known with significantly greater accuracy. Further, relative uncertainty between features can be monotonically reduced through increasing *reobservation* of known features, allowing non-divergent probability estimation over time [42].

To reduce absolute estimation errors, loop closure is a well known and effective technique. A loop is a sequence of states such that the beginning and end of the loop contain a corresponding set of feature observations. Errors accumulated over the loop result in inconsistent estimates of the features between the two terminal states. The relative correlation principle allows this error to be measured and corrected using bundle adjustment (BA), which jointly refines the map and localization state estimates within the loop [49]. However, since the uncertainty growth is unknown, robust *loop detection* or *place detection* approaches are needed for data association, which can be quite challenging. Accurate loop detection is critical, as a false positive can completely corrupt the integrity of the map [50]. Loop closures and reobservations are both critical components to bound the growth of error over time [51, 52], but various unsolved challenges remain in the literature.

2.5 Active Perception for Online Exploration

A taxonomy of existing approaches to online exploration can be determined according to several key facets of their formulation:

- Action representation: how actions are mathematically represented
- Search space: how candidate actions are generated
- Decision metrics: how candidate actions are evaluated and optimized
- Geometric horizon: geometric extent of available knowledge considered for decision making
- Temporal horizon: forward-time depth of decision evaluation
- Temporal coherence: dynamic maintenance and reuse of the search space over time
- Action commitment: whether actions can be abandoned before completion if a better one is found, or are required to be fully executed before selecting a new one

2.5.1 Frontier-based Exploration

Frontier-based exploration was first introduced by B. Yamauchi [53], where the exploration problem was formulated as: "Given what you know about the world, where should you move to gain as much new information as possible?" [53]. To solve this problem, Yamauchi introduced the concept of frontiers, defined as the boundary regions between free space and unknown space within the partially built map. Frontiers help to identify specific locations where the adjacent free-space can be safely traversed to observe the adjacent unknown space, which in turn will increase perception knowledge and extend the respective frontier boundaries. This is repeated in an iterative fashion until no frontier regions remain, implying the environment has been fully observed and mapped.

Early implementations of classical frontier exploration were presented mostly for 2D exploration tasks involving UGVs [54, 55, 56, 57, 11]. Given a 2D occupancy grid map, a greedy closest frontier (CF) search strategy was applied which computes the nearest frontier to the robots current position as the navigation goal. A path is then planned that leads towards the frontier, while monitoring for potential collisions as the path is executed.

Extensions from 2D to 3D exploration for aerial vehicles introduced significant challenges due the increased computational complexity and memory demand for building and planning on 3D maps, and given the constrained onboard resources. This forced early approaches to make a variety of simplifying assumptions. A technique was used in [58] that reduced each iteration of the exploration from 3D to 2D by using a fixed-altitude for exploration and motion planning, which was perhaps the first demonstration of an autonomous aerial system capable of exploring and navigating a completely unknown environment [59]. The vehicle was equipped with a laser scanner, stereo camera, and IMU which were used in an EKF sensor-fusion approach to perform keyframe SLAM. The map was reduced to a 2D floorplan representation, allowing the direct use of the frontier-based approach in [53]. Experimental results demonstrated the capability of the system to perform localization and stable flight, autonomously exploring various unknown environments for distances of $44.6m$ and flight times of around 6 min. These results were a milestone in the development of autonomous MAVs, but were far from solving the general problem. Decoupled vertical and horizontal motions prevents optimal path planning, and assumes a building layout with consistent floor and ceiling geometries. A relatively low-clutter environment was used which allowed for relaxed planning and control techniques. Further, complete coverage was not achieved, as only the 2D floor plan layout was of interest.

It was discovered that frontier-based exploration suffered in large open areas due to the lack of adequate surface information. One mitigating approach was to use wall-following strategies in such conditions, which helped ensure the vehicle remained near the target surfaces to be mapped. In the approach of [33], nearby walls were detected using RANSAC to perform geometric plane fitting. The point on the wall closest to the current position is selected as p_1, and a point p_2 is computed at the same altitude as the MAV at a distance d from p_1 along the wall. These points are then used to compute the next waypoint p_3, which is perpendicular to the wall plane with a distance equal to the distance between p_1 and the current position of the MAV. Although experimental results indicated an improvement over the frontier-based strategy, a user command was required to determine when to switch to the wall-following strategy. Further, though mapping was performed in 3D, exploration was limited to two-dimensional planes, resulting in over-simplified exploration paths.

To compensate for limited onboard computation, a compact frontier exploration strategy was implemented in [59]. This strategy was based on the Sensor-Based Random Tree (SRT) frontier approach, SRT-Star, developed in [60]. The approach utilizes the characteristics of a 2D laser range scanner to significantly reduce the computational demand. The underlying idea was that a command velocity ($\overrightarrow{v}_{cmd}$) was composed of contributions from frontier velocity ($\overrightarrow{v}_{fr}$) and a wall-following velocity ($\overrightarrow{v}_{wf}$) according to:

$$\overrightarrow{v}_{cmd} = \overrightarrow{v}_{wf} + \overrightarrow{v}_{fr} \tag{2.3}$$

The SRT-Star approach operates by dividing a laser scan beam into sectors containing a left-point, right-point, and mid-point. These sectors are used to identify the location of frontiers. If all sectors are completely free, the mid-point is declared the frontier. If large discontinuities exist between adjacent sectors, then the corresponding left-point or right-point are declared frontiers. If any number of frontiers are detected, a waypoint is generated within the sector that produced the frontier. Otherwise, the vehicle is commanded to the previous waypoint visited. Although autonomous operation was demonstrated, the results were limited and suffered from similar drawbacks as [59].

The work of [34] proposed a more efficient 3D extension of frontier exploration. The efficient OctoMap framework was incorporated, reducing the complexity involved with extension to 3D. A cost function was implemented to evaluate and select the best frontiers, rather than arbitrarily selecting the closest for exploration, similar to the method proposed in [61]. A novel contribution was a more efficient frontier extraction approach that only evaluated frontiers each iteration for state-changed voxels, rather than naively evaluating the entire map each iteration. To compute the optimal frontier, they introduced the utility function:

$$c_f = w_1 \times \frac{N_{unknown}}{N_{all}} - w_2 \times |P_f - P_O| \tag{2.4}$$

where c_f is the cost of the frontier cell candidate, P_f is the frontier cell coordinate, P_O corresponds to the current position, and w_1 and w_2 are relative weights to balance the evaluation towards either maximizing coverage, or maximizing exploration. The terms $N_{unknown}$ and N_{all} correspond to the number of unknown and all cells around the frontier within a spherical radius R.

Experimental results were provided, primarily evaluated in terms of processing time. A test environment of size $12.6m \times 7.8m$ required less than 5 minutes to completely explore. The integration of new observations to the OctoMap required an average of 0.5s process, while frontier clustering and evaluation was in the order of milliseconds. These results clearly demonstrated the computational feasibility of their approach using limited computational resources. However, a measure of coverage completeness, accuracy, or quality was not provided in the results.

A variant of the classical frontier strategy was presented in [62], motivated by the high demands on computation and memory for dense representation of occupied, free, and unknown space. This approach did not require explicit, dense representation of free space in an occupancy map. Using the assumption that unstructured or uncluttered regions of the map generally correlate with unexplored regions, a Newtonian dynamics-based particle model was implemented in which the particles are dispersed based only on interaction of occupied space in the spatial map. The motion of these particles is governed by a stochastic differential equation (SDE), where obstacles induce forces on the particles, causing them to disperse until coming to rest in unknown areas (free of obstacle-generated forces). The final resting positions in unknown space are then defined as frontier regions, providing candidate locations for further exploration. As a result, motion planning is performed in 3D and is efficient enough to operate using only onboard processing.

A significant drawback of classical frontier exploration is that frontiers indicate only the *existence* of adjacent unknown space, but not the quantity or quality. Using a frontier location directly as the navigation goal ignores sensor's measurement range, thus causing inefficient and wasteful motions. Furthermore, a frontier location near surfaces generally do not represent a feasible goal for a robot due to collision with the surface obstacle, making their direct use in this way ineffective for surface coverage tasks.

2.5.2 Sampling-based and Information-theoretic Methods

Exploration can be effectively modeled as an extension of the Next-Best-View (NBV) problem introduced by [63], which can overcome several of the drawbacks associated with classical frontier-based approaches. Here, a *view* refers to a hypothetical pose of the sensor apparatus used to predict and analyze the spatial information expected to be visible if the real sensor were to be placed at this pose. The expected visible information is then said to be *covered* by the view.

The classical NBV problem assumed full prior knowledge of the target object is given to facilitate the search and evaluation of NBVs, where the objective was to find a minimum set of views that maximizes coverage of the known surfaces of the object model. This premise can be adapted for online exploration tasks by instead evaluating views according to currently unknown parts of the environment model, rather than the known parts.

NBV-based exploration methods typically utilize a generate-and-test paradigm which apply sampling techniques necessary to discretize the continuous configuration space into a finite set of candidate views for analysis [64]. The quality of a view is evaluated according to some measure of its *information gain* (IG), which quantifies the new spatial information potentially observable from the view [65, 66, 67, 68]. A cost metric is additionally used to evaluate the expected effort for the robot to visit the view (e.g. time or energy). Most critical differences between existing NBV approaches occur within the sampling strategy for generating view candidates, and the formulation of metrics for analyzing and comparing candidates for goal selection.

Information gain is commonly computed volumetrically by finding the expected amount of unknown space visible from a view [69, 70, 71]. This necessarily involves checking for occlusions within the known space using techniques like raycasting, which incurs high computational complexity that can rapidly increase with various factors like map resolution, sensor field of view, and sensing range. This limits the number of distinct views that can be practically evaluated within a given time period. The high complexity also make it difficult to analyze overlapping or mutual information between views, such that most approaches treat the gain as an independent value that prevents an understanding of the unique gain contributions of each view within a group.

An early formulation of such methods can be found in [72]. In this approach, flight paths are computed by simultaneously considering navigation integrity and exploration gain. They reason that undesirable performance may occur as a result of inaccurate state estimates caused by low information, or from filter instability from incorrect data association. The accuracy of the navigation estimate was represented by the amount of entropic information in the probability distribution of these estimates. The entropy $H(x)$ of a multivariate Gaussian probability distribution of navigation estimates, using covariance matrix $\mathbf{P}$, can be represented as:

$$H(x) = \frac{1}{2} log[(2\pi e)^n |\mathbf{P}|] \tag{2.5}$$

This represents the compactness of the distribution, i.e. the information quantity in vehicle state estimation. The mutual information $I[x, z]$ was defined as the difference between the entropy of the distributions before and after making an observation. They used this principle to formulate a utility function U representing the total utility for traveling to a viewpoint as a weighted sum of the vehicle pose estimate and information gain:

$$\mathbf{U}_{dest} = w_v I[x_v, z] + w_m I[x_m, z] \tag{2.6}$$

where z is the observation, and w_v and w_m are pose and map information weights. The first half of this equation causes the algorithm to prefer vehicle states which minimize state estimation uncertainty by maximizing the number of previously observed features in view. The second half favors exploration of new, unobserved portions of the map. The weights w_v and w_m can be tuned to balance these effects. This formulation contrasts the utility functions described in Section 2.5.1, which generally balance exploration gain with distance traveled. However, its performance is subject to tuning of these parameters.

Simulation results indicated the approach was effective at selecting paths with high information gain and achieved good localization over the course of a 200s flight. Several loop closures were observed during this flight, directly attributed to the ability of the utility function to favor paths with lower vehicle state uncertainty. However, no baseline or comparison to other approaches was provided, making the results difficult to interpret. As the results were performed only in simulation under controlled conditions, the real-world performance is difficult to assess. Further, the proposed solution demonstrated only a planning approach, and not a complete system.

A hybrid approach was presented in [61] attempts to simultaneously plan for both rapid exploration and detailed coverage. This method aimed to explore as much of the environment as possible in a minimum amount of time, while observing the complete surface of an environment, given viewing angle and distance constraints, such that the reconstructed model is complete and distortion-free. The proposed algorithm first searches for goals located near frontier boundaries. These views are then assigned a cost corresponding to the expected information gain weighed exponentially by the cost to reach the view, selecting the single lowest-cost view as the goal. A path is then planned to reach the goal which also attempts to maximize sensor coverage during execution of the path.

Using a vehicle configuration of $[x, y, z, \phi]$, assuming zero roll and pitch, a state lattice $\mathcal{L}$ discretized the 4D state space into a regular 3D grid pattern with yaw angles discretized non-uniformly. A set of candidate goals $\mathbf{G}$ were constructed by including all of the states located on frontiers. For a candidate goal $s_g \in \mathbf{G}$, the information gain $I(s_g)$ was defined as the number of unexplored, non-occluded voxels in the viewing frustum of s_g. Given the current MAV state $a \in \mathcal{L}$, the candidate goal is selected which maximizes the utility function:

$$U_1(s_g) = I(s_g)e^{-\lambda l_{min}(a,s_g)} \tag{2.7}$$

where λ is a parameter to balance between rapid-exploration and detailed coverage, and l_{min} is the cost of the shortest path from a to s_g. Simulation results for an apartment-sized world were provided with a comparison to traditional frontier-based planning. The proposed method observed around 90% of the voxels with a path length of 136.1m, while the frontier approach observed around 74% of the voxel surfaces and produced a total path of length 98.4m.

One simplifying assumption of this work to reduce computation was through pre-computed motion primitives. This restricted the available motions of the vehicle, reducing optimality. It also introduces potentially unsafe maneuvers not handled by the limited set of motion primitives, and does not effectively handle vehicle dynamics. A remaining challenge for the work was performing loop closures for SLAM, to which they argued that no real-time CPU-based methods existed at the time.

To improve the efficiency of frontier exploration with the presence of large open space, some approaches used the concept of surface-frontiers. These are defined as the location that occurs at the intersection of occupied, free, and unknown space. This formulation is useful for large-scale scenes, where the majority of space is usually represented as unoccupied, leading to inefficient exploration, or wasteful computation in the evaluation of many views with little information. Surface frontiers ensure that the candidate exploration points are near the surfaces of interest for the reconstruction, rather than the free-space.

Surface frontiers were first introduced in [73], which were combined with an NBV-based approach. Each planning iteration, a region of interest is defined as $R \subseteq W$, where $W \subseteq \mathbb{R}^3$. Surface frontiers were detected within this region, which were evaluated using a similar utility function to [34] and [61]. This utility function evaluated a view's information gain and the cost to reach the view. A tuning parameter was applied to balance the score between these two terms.

An experimental evaluation was performed on a 50m bridge structure. The exploration was completed in approximately 6 min, and resulted in a 5 million point model of the structure. These results were compared to the performance of a skilled human pilot, qualitatively performing as good or better than the pilot. The results of the planning algorithm were a milestone in terms of scale and efficiency. Although the process was performed completely autonomous, it relied on GPS for localization and did not require any SLAM techniques. The performance of the approach would likely suffer significantly if required to perform SLAM, reducing the available computational resources. The effect of vehicle and map uncertainty on the overall performance would also need to be further analyzed.

A more recent approach is the Receding Horizon Next-Best-View planner presented in [74]. This approach utilizes a sampling-based path planning strategy to generate admissible paths that maximize exploration.

Assume a map $\mathcal{M}$, vehicle configuration $\xi = (x, y, z, \phi)^\top$ constrained by maximum velocity v_{max} and maximum yaw rate $\dot{phi}_{max}$, and a target configuration for motion tracking $\sigma_{k-1}^{k} = s\xi_k + (s-1)\xi_{k-1}$, with $s \in [0,1]$. A finite-iteration random tree is grown within known free space using an RRT-based approach [75]. Admissible samples ξ are connected which satisfy v_{max} and $\dot{phi}_{max}$, and assigned a connection cost according to the Euclidean distance between the configurations. Each branch is also evaluated for information gain in terms of the number of unmapped voxels able to be observed by the sampled views. For a node k, the information gain $\mathbf{Gain}(n)$ is described as:

$$\mathbf{Gain}(n_k) = \mathbf{Gain}(n_{k-1}) + \mathbf{Visible}(\mathcal{M}, \xi_k) e^{-\lambda c(\sigma_{k-1}^{k})} \tag{2.8}$$

Visible where λ is a tuning factor designed to penalize high path costs. The term $\mathbf{Visible}(\mathcal{M}, \xi_k)$ determines the amount of new information observable from configuration ξ_k, using λ to penalize high path costs. The information gain of each view is propagated through the branch. After a defined number of iterations, the branch with the highest information gain is selected as the motion path, but only the first edge of the branch is executed. This process is iteratively repeated until no positive gain can be found. Conceptually, this results in execution of local paths determined from global exploration gain estimates. This allows paths which may not have the most immediate exploration gain, but are predicted to eventually lead towards areas of maximum exploration.

Simulation results were obtained and compared to a frontier-based approach. For an apartment-scale exploration scenario, the frontier planner was able to complete the exploration around 32s faster, but did not observe as many voxels. In a larger scale exploration of a bridge, the proposed planner required 43.8min, where the frontier planner was manually terminated after 1670.1min. These results indicate the receding horizon planner scales well, and suggests the evaluated frontier-based implementation is not suited for large-scale, more complex scenes.

The approach in [76] was recently proposed as follow-up work to the method described in [74]. This approach built on the Receding Horizon Exploration planner, adding uncertainty-aware constraints in a two-step planning paradigm. The first layer of the planner functioned similar to the base approach of [74]. This generates a waypoint to use as the next target for exploration. The novel contribution is the addition of a second planning layer, which attempts to plan a new trajectory to reach the waypoint while attempting to minimize uncertainty in localization. This is achieved through a modification of the utility function of (2.8). Assume a map $\mathcal{M}$, vehicle configuration ξ, admissible path σ, and occupancy probability $P(m)$ of a voxel $m \in \mathcal{M}$. The updated exploration gain $\mathbf{ExplorationGain}(n^E)$ of tree node n^E can then be formulated as:

$$\begin{aligned}\mathbf{ExplorationGain}(n^E_k) = \mathbf{ExplorationGain}(n^E_{k-1}) +& \\ \mathbf{VisibleVolume}(\mathcal{M}, \xi_k) e^{-\lambda c(\sigma^k_{k-1})} +& \\ \mathbf{ReobservationGain}(\mathcal{M}, \mathcal{P}, \xi_k) e^{-\lambda c(\sigma^k_{k-1})}&\end{aligned} \tag{2.9}$$

The addition of the term $\mathbf{ReobservationGain}(\mathcal{M}, \mathcal{P}, \xi_k)$ serves to increase reobservation of previous map data integrating occupancy map probabilities P at a view

ξ_k. Conceptually, this increases the score of views with larger numbers of voxels with a occupancy probability.

An experimental evaluation was performed in an environment with dimensions 12 x 6.5 x 2m, using an arrangement of small boxes to represent geometric structures within the space. The planner was able to completely explore the target environment and produce a visually appealing 3D model. By observation, the paths selected were characterized by low navigation uncertainty. Several iterations of the experiment indicated repeatable results. However, no quantitative analysis was provided with regard to the exploration time, or indication of optimality. More evaluation under different conditions would be needed to validate the effectiveness. Additionally, no comparison to alternative approaches was provided, making it difficult assess any performance improvements over other existing techniques in terms of exploration efficiency, accuracy, or robustness to different environment conditions.

2.5.3 Tree-based Exploration

Tree-based methods organize sampled views as vertices in a geometric tree where directed edges between vertices represent feasible paths between views. The RH-NBVP approach of [77, 78] applies rapidly-exploring random tree (RRT) to grow a tree rooted at the robots current position. Each node in the tree is weighted according to their predicted information gain based on how much unknown space lies within the view. Cost weights are aggregated along each branch, and the leaf node with the highest value is used to identify the best branch to explore, iteratively repeating the process in a receding horizon fashion. This has become a well-known approach and is often used as a baseline for comparative analysis [79, 80, 81].

A hybrid approach that combines both frontier-based and NBV-based techniques was introduced by [79], referred to as AEP. It combines the RH-NBVP strategy for local planning, while switching to frontier-based planning for global search when local planning fails to find informative views. FFI [80] is also a hybrid approach that uses an efficient frontier clustering strategy to guide view sampling.

A significant drawback of tree-based planning is the difficulty in preserving the previously computed tree structure as the robot navigates to each goal. The RH-NBVP approach builds a new tree each iteration, discarding the previously built structure that may still contain useful knowledge. Other approaches attempt to transfer as much of the previous tree structure as possible by rewiring its edges to initialize the construction of a new tree. Since tree-based methods are rooted at the robots position, they tend to become increasingly inefficient over larger distances, making it difficult to handle dead-end or backtracking cases.

2.5.4 Graph-based Exploration

Various approaches have utilized graph structures that can overcome some of the limitations and drawbacks of trees. The approach of [82] builds a history graph that stores previously visited positions and their edge connections. These are used as potential seed points for RRT, which allows a tree to be grown from different positions across the map, rather then just from the robot position. An approach using Rapidly-Exploring Random Graphs (RRG) was presented in [70] for exploration of subterranean environments. A Probabilistic Roadmap (PRM) strategy was used by [83] to build a graph of feasible configurations and paths over the map as it is explored.

2.5.5 Topological Map-based Exploration

Topological maps have been applied by recent works which aim to reduce the planning complexity through the compact representation provided by a topological map. Topological maps can be considered as an extension to graph-based methods, where vertices represent some volumetric sub-map, or *place*, and edges represent the adjacency or reachability between places. This coarse and abstracted representation is more efficient for handling large-scale environments, which can become intractable to explore online using alternative approaches. However, they usually lack sufficient metric knowledge for direct use in navigation.

[84] used a topological map for exploration of underground mines using a ground robot. The regions of intersection between passageways were represented as nodes, and exploration was planned along the edges between nodes. A more recent approach proposed by [85] also uses a topological map for subterranean exploration. Convex polyhedrons are used to estimate distinctive exploration regions (DER-s) which are added as graph nodes to the map. Each DER represents an enclosed 3D volume of the map like an enclosed room or corridor, providing the planner with knowledge of high-level intent such as moving between distinctive rooms or regions. Other approaches have applied segmentation algorithms to identify the separation of distinct exploration regions like rooms of a building [86].

2.5.6 Myopic greedy planning

The majority of existing methods compute navigation goals using myopic planning strategies that greedily optimize the cost of the next single planning decision [87, 80], or within a limited planning horizon [77, 79]. Some works allow planning over the full map, but still use greedy search for the decision making. These are sometimes referred to as global planning methods, but we clarify they are still considered myopic.

Myopic strategies bias exploration toward regions with high information gain, while ignoring small gains even if they are closer. This bias can frequently create regions of incomplete coverage when a high gain goal leads the exploration away from the current region before it is fully mapped. This can also result in frequent back-and-forth oscillation between goals, or require re-visitation of these regions after the robot has traveled a significant distance, backtracking over potentially large distance. This greatly reduces efficiency, and can result in sparse coverage gaps or failure to fully explore an environment within an allowed time limit, especially over large-scales.

A relatively small number of works have recently attempted to overcome the drawbacks of greedy planning using non-myopic planning strategies. This has been formulated using the Traveling Salesman Problem (TSP) [88, 89], but often relies on prior map knowledge [90, 91].

A sector decomposition approach was presented by [89], which partitions the map into a set of convex sectors used to compute a TSP sequence. However, the sector decomposition method is computationally expensive, especially for finer map resolutions, which can greatly decrease the update rate of the map and planning. Additionally, sectors form an exact partitioning of the space, which can make the geometric properties of the resulting sectors difficult to control. This can result in too many or too few number sectors that may not be effective for large-scale and complex environments.

2.5.7 Environment and task-specific approaches

Simplifying or restrictive assumptions are sometimes made on the operational environment. This can include indoor operation, or reliance on certain regular geometric features, e.g. room structures used for segmentation. Some applications are intended to operate in relatively obstacle-free environments, such as outdoors or underwater [92], which contain an abundance of free-space that greatly simplifies collision checking and other sub-tasks. Assumptions can significantly restrict the practicality of many approaches for general use, or require fine tuning of parameters between different environments to achieve their rated performance.

2.6 Critical Analysis

This section provided a review of the two main categories of exploration planning: frontier-based and information-theoretic methods. Frontier-based methods initially struggled to overcome the computational complexity for 3D planning and SLAM given computational constraints, resorting to simplifying assumptions such as fixed-altitude planning, reactive planning based on wall-following, and simplistic, small-scale environments. The introduction of OctoMap as an efficient mapping framework was a critical turning point that reduced the computational load and facilitated new, more sophisticated approaches to become computationally feasible with less software engineering effort.

State-of-the-art information-theoretic approaches are generally formulated using a utility function to evaluate a set of candidate views according to an information gain metric, path cost metric, and any other sub-utilities needed. Different methods may introduce minor contributions and optimizations, but the overall planning search strategy of many recent approaches do not drastically differ. Many approaches also remain dependent on restrictive assumptions that are often violated in different conditions, e.g. clutter-free environments. They are also dependent on tuning of parameters, which can significantly impact performance, and may vary between environmental conditions.

Some of the key factors contributing to limitations of existing approaches are outlined in Figure 2.2, and a summary of significant limitations is as follows:

- greedy and myopic planning strategies that focus on the incremental exploration objective, but fail to consider the global one,
- non-generalized approaches that are limited to small-scale environments, or specialized for specific environments or conditions (e.g. subterranean or building-like structures),
- most approaches succumb to high computational costs:
  - they do not scale well with respect to environment size or map resolution,
  - the ability to quickly replan on added knowledge diminishes, where a sub-optimal plan is fully executed before replanning,
  - reduced velocities are often required to compensate for low planning rates,

– frequent stop-and-go motions can occur.

We observe that there is not sufficient attention to the underlying data management issues of the general OSCP problem in the robotics research community, which could be due to limited research funding and development cycle where data infrastructure was not a focus. There is also a lack of open-source software to help reducing efforts that researchers have to put into developing a good data management system. The aforementioned limitations of existing approaches are the results of that. However, for many realistically large-scale OSCP tasks, it is critical to have smart and sophisticated data management systems, requiring careful conceptual, algorithmic, and data structure designs and efficient software engineering solutions.

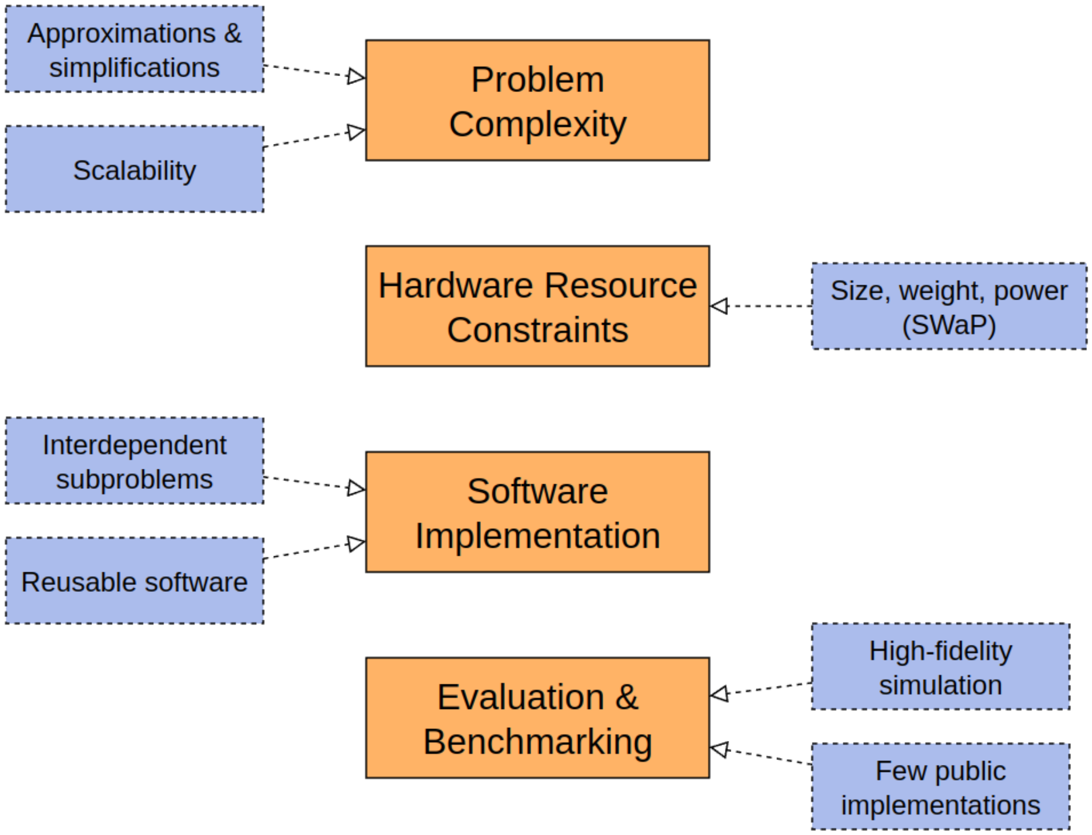

Approximations & simplifications
Scalability
Problem Complexity
Hardware Resource Constraints
Size, weight, power (SWaP)
Interdependent subproblems
Reusable software
Software Implementation
Evaluation & Benchmarking
High-fidelity simulation
Few public implementations

Figure 2.2: Summary of key problem challenges.

CHAPTER 3: PROBLEM FORMULATION

We assume exploration is performed using an MAV equipped with an onboard depth sensor (e.g. stereo-visual, RGB-D, or LiDAR) to perceive 3D space, noting that other systems such as mobile ground robots could also be utilized without loss of generality. The following section will formulate the key constituents of the OSCP problem we address.

3.1 Operational Environment

Let $\mathcal{W} \subset \mathbb{R}^3$ represent the bounded 3D space of the operational environment, referred to as the *world*. The material structures and objects of the world represent *occupied* space $\mathcal{W}_{occ} \subset \mathcal{W}$, while the remaining volume is defined as *free-space* $\mathcal{W}_{free} \subset \mathcal{W}$, such that $\mathcal{W} \equiv \mathcal{W}_{free} \cup \mathcal{W}_{occ}$. Occupied space is assumed to be rigid such that contact by the agent would cause a collision, and free-space is considered to be collision-free and visually transparent.

The intersection boundaries between occupied and free-space define the surface manifolds, $\mathcal{S} \subset \mathbb{R}^2$. Surface manifolds are assumed to be visually opaque, and a surface point is considered optically visible from a point $\boldsymbol{x} \in \mathcal{W}_{free}$ only if no occupied space lies between the surface and $\boldsymbol{x}$. Otherwise, the surface is considered to be occluded from $\boldsymbol{x}$.

3.2 Environment Map Model

A spatial occupancy map $\mathcal{M}$ is used to store environment knowledge as it is discovered from sensing. We assume the use of a 3D grid-based occupancy map $\mathcal{M} = \{\boldsymbol{m}_0, \ldots, \boldsymbol{m}_m\}$, though other map models could also be used without loss of generality (e.g. Signed Distance Field (SDF) [93]). The map partitions $\mathcal{W}$ by a set of non-overlapping cubic volumes $\boldsymbol{m} \in \mathbb{R}^3$, known as voxels. The minimum edge length of a voxel dictates the map resolution, $r_{\mathcal{M}}$.

To account for measurement uncertainty, each voxel has an associated occupancy probability $\boldsymbol{m} \mapsto [0, 1]$ defined over its volume that is updated from sensor observations. Higher probability values indicate higher confidence that some occupied space lies within the voxel volume, and lower values indicate the voxel volume contains only free-space. An occupancy state classifier $\mathcal{O}(\boldsymbol{m}) \in \{o_{unk}, o_{occ}, o_{free}\}$ is used to evaluate the discrete probability state according to an upper and lower probability threshold, where o_{unk} indicates the state is *unknown* when its probability does not exceed the thresholds. As sensor measurements are integrated the state is classified as either o_{occ} or o_{free} to indicate, respectively, whether the voxel is believed to contain occupied or free space. The set of occupied voxels are given as $\mathcal{M}^{occ} = \{\boldsymbol{m} \mid \mathcal{O}(\boldsymbol{m}) = o_{occ}\}$, the set of free voxels is given by $\mathcal{M}^{free} = \{\boldsymbol{m} \mid \mathcal{O}(\boldsymbol{m}) = o_{free}\}$, and the set of unknown voxels is given by $\mathcal{M}^{unk} = \{\boldsymbol{m} \mid \mathcal{O}(\boldsymbol{m}) = o_{unk}\}$, with the initial map state given as $\mathcal{M} \overset{\text{init}}{=} \mathcal{M}^{unk}$.

Spatial frontier boundaries occur within $\mathcal{M}$ wherever unknown voxel and a free voxel share a common border, as shown in Figure 3.1. The location of a frontier feature $f \in \mathcal{M}$ corresponds to the unknown voxel of the frontier boundary, with the set of frontier features defined by:

$$\mathcal{F} = \left\{ \boldsymbol{m} \in \mathcal{M}^{unk} \mid \sum_{\boldsymbol{m}_k \in \boldsymbol{m} \oplus \hat{\mathbf{K}}} \left[\boldsymbol{m} \in \mathcal{M}^{free} \right] > 0 \right\} \tag{3.1}$$

where $[\cdot]$ is the Iverson bracket. $\boldsymbol{m} \oplus \hat{\mathbf{K}}$ represents the adjacent voxel neighbors about $\boldsymbol{m}$, where $\hat{\mathbf{K}} = \{\hat{\boldsymbol{m}}_i\}$ is a structuring kernel that defines the connected voxel neighborhood basis and $\oplus$ is the Minkowski sum operator. $\hat{\mathbf{K}}$ is structured from the set or subset of the normalized 27-connected voxel neighbors, taken with respect to $\boldsymbol{m}$ by the Minkowski sum operation. Each neighboring voxel is checked to determine

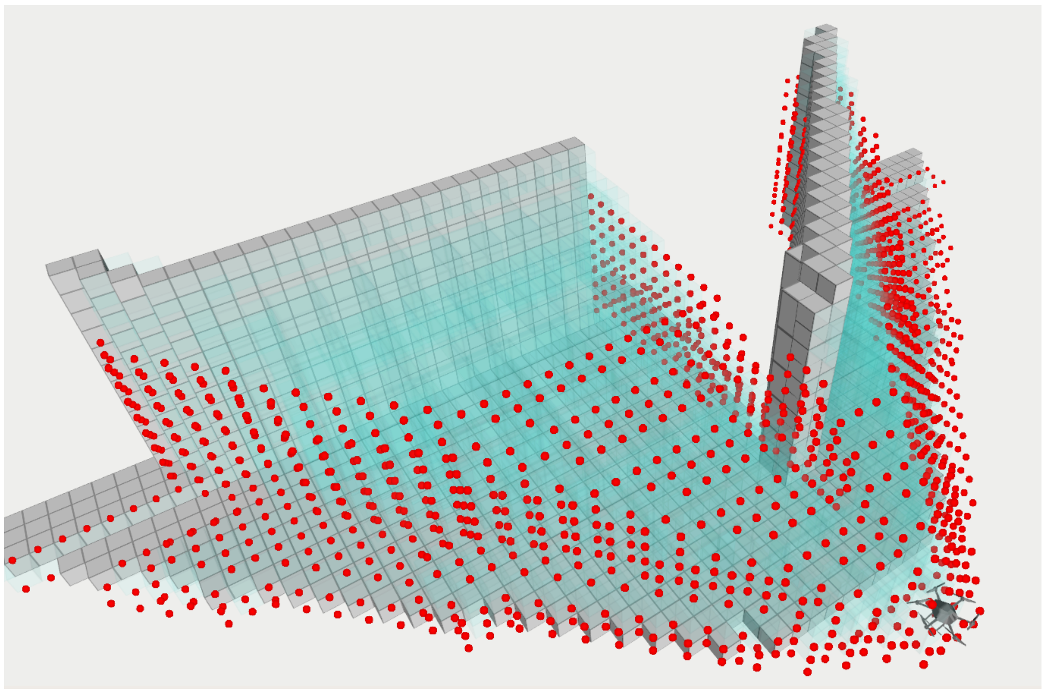


Figure 3.1: Visual depiction of frontier boundaries within the occpupancy map. Free voxels are shown by the semi-transparent blue cubes, and occupied voxels are shown by the gray cubes. Frontier features are overlayed in red at the boundaries between unknown space.

if the frontier condition is met, where $\hat{\mathbf{K}}$ can be differently structured to mask some of the neighbors if desired, for example to exclude corners from being evaluated.

3.3 Robot Model

The robot agent is modeled by a rigid body with pose configuration $\boldsymbol{q}^{agent}(t) = (\boldsymbol{x}, \boldsymbol{a}),\ \boldsymbol{q} \in SE(3)$ at time t, where $\boldsymbol{x} \in \mathbb{R}^3$ is the position vector and $\boldsymbol{a} = \{\varphi, \vartheta, \psi\}$ is the orientation vector represented by roll, pitch, and yaw Euler angles, respectively. Additional parameters $\boldsymbol{v}_{max}$ and $\dot{\psi}_{max}$ are used to specify the maximum allowable velocity and yaw rate, respectively.

A spherical volume B^{safe} centered at $\boldsymbol{x}$ with radius d_{safe} is defined, where d_{safe} specifies the minimum obstacle separation distance for safe operation. A point within the map is considered collision-free, or feasible, if the volume B^{safe} centered at the point contains only free space, otherwise it is considered to be in collision (i.e. infeasible).

3.4 Sensor Model

The robot's depth sensor is modeled by the parameter vector $[R_s, \alpha_s, d_{max}^{sense}]$. $\alpha_s = [\alpha_h, \alpha_v] \in (0, 2\pi]$ is the maximum angular field of view (FoV) on the horizontal and vertical dimensions of the sensor, and $R_s = [R_{sx}, R_{sy}]$ is the maximum spatial resolution. $d_{max}^{sense} \in \mathbb{R}$ is the maximum effective sensing range that surface points can be accurately detected by the sensor. This value corresponds to the physical limitations of the sensor, where distances greater than d_{max}^{sense} either cannot be measured, or are rejected due to loss of accuracy.

The sensor parameters can be combined with a pose $\boldsymbol{q}$ to form a projection model $\lambda \in \Lambda$, referred to as a *viewpose*. The projected space from λ is described by the subset of rays that pass through the view's origin $\boldsymbol{x}$, constrained by the intervals $[\vartheta \pm \alpha_v/2]$ and $[\psi \pm \alpha_h/2]$ of the unit-sphere. The length of each ray is constrained by d_{max}^{sense}. The projected space defines the view volume of a viewpose, and a location within the view volume is considered visible if there are no occlusions between it and the origin. This provides the basis for making visibility queries and predictions on the expected information gain.

3.5 Reachable Configuration Space

The *reachable configuration space*, $\mathcal{X} \subset \mathbb{R}^3$, is a metric space defined by all admissible configurations path-connected to the robot's initial position, $\boldsymbol{x}_0^{agent}$. As a precondition, a configuration is considered *admissible* if it does not intersect any occupied space within distance d_{safe}. It is then considered *reachable* if there exists a simply-connected path of admissible configurations from $\boldsymbol{x}_0^{agent}$. The distance between two reachable points is quantified by a metric value $L \in \mathbb{R}$.

3.6 Goal Space

The surfaces that can possibly be covered at any point during exploration is inherently restricted to a subset $\mathcal{S}_{vis} \subseteq \mathcal{S}$ for which some viewpose λ able to observe the surface exists within the reachable configuration space. The *goal space* $\Lambda^G \subset \Lambda$ is then defined as the set of all such feasible configurations that contribute some amount of coverage of $\mathcal{S}_{vis}$, quantified by a gain metric, $\gamma \in \mathbb{R}$.

3.7 Information space

The *information space*, Ω, refers to the time-varying knowledge of the environment, together with any additional knowledge derived from the map through inference methods. In this way, the information space includes the spatial map, $\mathcal{M}$, together with any additional knowledge derived from $\mathcal{M}$, such as spatial frontiers $\mathcal{F}$, the reachable C-Space $\mathcal{X}$, and goal space Λ^G.

3.8 Non-myopic planning

A planning strategy operates by searching the information space for the optimal goal $\boldsymbol{q}^g \in \Lambda^G$ for navigation, where the myopicity corresponds to the extent of its planning horizon. A myopic strategy typically uses greedy search techniques which treats each goal or action as independent of the others, greedily selecting the best one. They may also constrain the search to only some local sub-region of the map, rather than considering its full extent. Myopic strategies use greedy search to find the an action that provides the best balances the immediate information gain with the cost to acquire it. This often biases actions towards the largest regions of information, while leading the robot away from nearby smaller regions of incomplete coverage that can be much more costly to revisited. Further, if the search strategy only considers local neighborhoods near the robot, rather than searching globally, these sparse coverage gaps may never be revisited at all, making these approaches incomplete.

In contrast, a non-myopic strategy searches globally over most or all of the available map, and additionally considers the long-term cost dependencies between selected actions. In general, this involves searching for ordered sequences of actions, rather individual actions, to evaluate and compare their long-term quality. While the search is performed over sequences of actions, typically the output of the planner is only a single action, corresponding to the first action of the optimal sequence. This is because the involved evaluation metrics are state-dependent with the robot pose and information space, both of which will changes once the first action is executed and necessitate replanning.

An important consideration is that the individual actions of a globally optimal plan can be locally suboptimal, such that the robot may need to execute suboptimal actions expected to lead to increased global optimality. However, since these metrics are state-dependent, the frequency of replanning becomes critical to prevent continued execution of a suboptimal action once it is no longer globally optimal with respect to the dynamically changing state information. In this way, the computational efficiency and frequency of each replanning iteration can directly impact the long-term costs of the exploration path that is executed.

CHAPTER 4: APPROACH OVERVIEW

This dissertation addresses the OSCP problem by first recognizing the representation and construction information space serves as a key computational bottleneck to any planning approach. Thus, we factorize our approach into three distinct subproblems, outlined by the iterative control flow model depicted in Figure 4.1. These include the subproblems regarding how to model the information space, how to update the information space, and how to apply non-myopic planning using the information space [94].

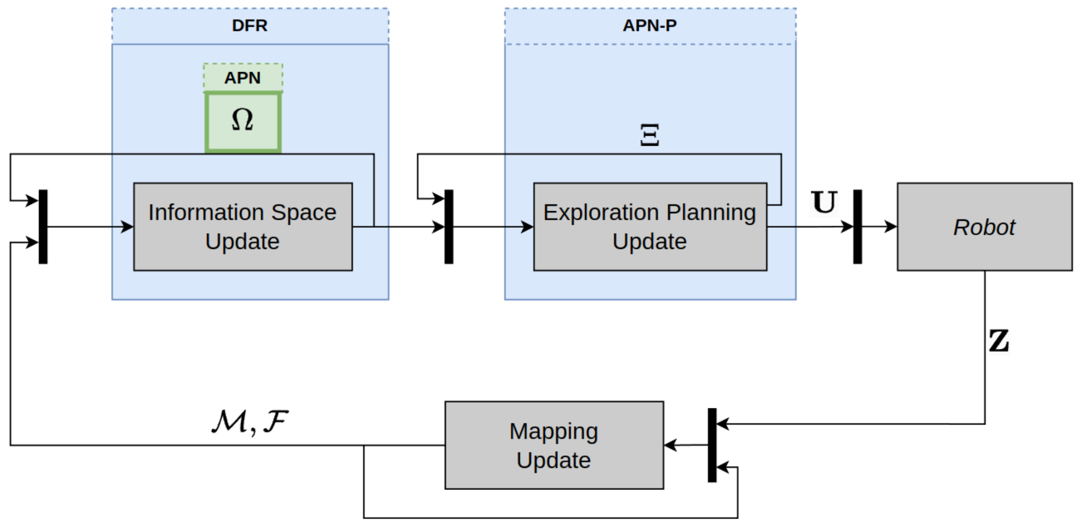


Figure 4.1: Approach execution control flow model.

The goals of the approach are such that the model of the information space, Ω, should be generalizable to different environments with varying sizes, complexities, and geometric characteristics. Dynamically updating Ω should be efficient to ensure its latest state accurately represents the latest map state for planning. Updates must also be scalable such that Ω can be maintained over the full extent of the spatial map as it is built, a necessary condition for globally informed planning. Finally, non-myopic planning operates on the latest state of the information space and should also be efficient and scalable to remain tractable over its increasing scale and complexity.

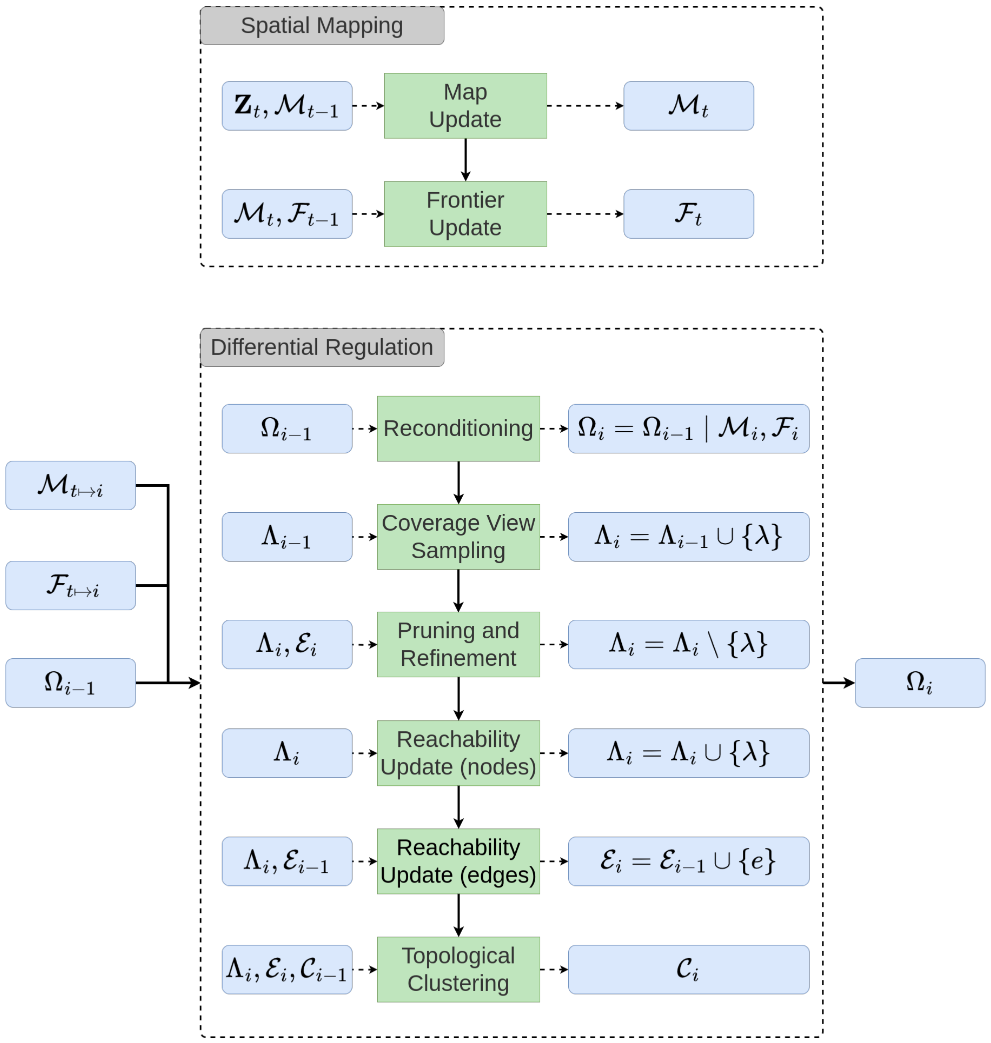
To achieve these goals, we first introduce a novel graph-theoretic information structure named the *Active Perception Network* (APN) to model the exploration state space data, detailed in Chapter 5. The APN is highlighted in green in Figure 4.1 as the representation of the information space. A key feature of the APN is a hierarchical representation over its configurations that helps to reduce its size complexity and enables variable-resolution planning as the map increases in scale. Another focus of the APN is the storage and organization of the contained data, such that dynamic changes can be efficiently made to any of its contents as its size increases, while also maximizing the low-level efficiency for search and query operations. Some of these details are related to software, data structures, and other implementation challenges, which discussed later in Chapter 9.

We additionally introduce the process of *Differential Regulation* (DFR) in Chapter 6, which operates on the APN to modulate its state with respect to the increasing map knowledge, as indicated in Figure 4.1. DFR consists of sampling-based methods for increasing knowledge of the goal space and reachable space. A novel approach for information gain analysis is utilized that enables the individual and mutual information gain of the APN to be efficiently computed, which is leveraged to accelerate informative view sampling, pruning, and refinement.

DFR exploits the incremental nature of map building where each sequential map update induces changes that occur only within a relatively small local region of bounded volume, independent of the total map size. With this insight, these incremental changes are tracked and cached using difference-awareness and memoization strategies to greatly reduce the computational overhead necessary to update the APN. This allows more discrete updates to be performed in a given time period, increasing the completeness and accuracy of each update. The ability to quickly perform each update is also critical to ensure the size of the map changes remain small, since the complexity of each update scales with the size of the changes.

An anytime exploration planner is presented in Chapter 7, which demonstrates the use of the APN to efficiently compute non-myopic global exploration sequences, and is also outline in Figure 4.1. The hierarchical representation of the APN is leveraged to first compute a global topological exploration plan over the full map. The beginning of the global plan is then locally optimized at a higher-resolution. Similar to the difference-aware approach used by DFR, sequential changes to the APN typically occur within locally bounded regions which are leveraged to initialize new planning instances from previous results. This allows optimizations to achieve faster convergence despite the increasing size of the map and APN.

The iterative update pipeline is illustrated in Figure 4.2, which consists primarily of two asynchronous processing loops. The first loop is dedicated for spatial mapping to allow continuous integration of the sensor measurement data, $\mathbf{Z}_t$, at high frequency. Frontier detection is performed after each map update, which operates only on the state-changed voxels that resulted from the update. This minimizes the complexity required to maintain the global frontier set, and provides a constant upper complexity bound that remains independent of the total map size. The second loop concurrently performs DFR to update the APN, which then serves as the input for replanning the current exploration solution. Further details of each DFR subroutine will be provided in Chapter 6.



Spatial Mapping
$\mathbf{Z}_t, \mathcal{M}_{t-1}$
Map Update
$\mathcal{M}_t$
$\mathcal{M}_t, \mathcal{F}_{t-1}$
Frontier Update
$\mathcal{F}_t$
Differential Regulation
$\mathcal{M}_{t\mapsto i}$
$\mathcal{F}_{t\mapsto i}$
Ω_{i-1}
Ω_{i-1}
Reconditioning
$\Omega_i = \Omega_{i-1} \mid \mathcal{M}_i, \mathcal{F}_i$
Λ_{i-1}
Coverage View Sampling
$\Lambda_i = \Lambda_{i-1} \cup \{\lambda\}$
$\Lambda_i, \mathcal{E}_i$
Pruning and Refinement
$\Lambda_i = \Lambda_i \setminus \{\lambda\}$
Λ_i
Reachability Update (nodes)
$\Lambda_i = \Lambda_i \cup \{\lambda\}$
$\Lambda_i, \mathcal{E}_{i-1}$
Reachability Update (edges)
$\mathcal{E}_i = \mathcal{E}_{i-1} \cup \{e\}$
$\Lambda_i, \mathcal{E}_i, \mathcal{C}_{i-1}$
Topological Clustering
$\mathcal{C}_i$
Ω_i

Figure 4.2: Mapping, frontier detection, and Differential Regulation process pipelines used to update the APN.

CHAPTER 5: ACTIVE PERCEPTION NETWORK

The Active Perception Network (APN) serves as a topological roadmap that stores the unified knowledge of the dynamically changing information state space. Its fundamental structure is represented by a hypergraph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{C}), \tag{5.1}$$

where $\mathcal{V} = \{v_i\}_{i=1,\ldots,n}$ is the set of graph nodes and $\mathcal{E} = \{e_{u,v}\}_{u,v\in[1,n]}$ is the set of traversal edges between nodes. The nodes have a bijective mapping to a codomain of viewposes, $\mathcal{V} \hookrightarrow \Lambda$, where the terms node and viewpose may also be referred to interchangeably.

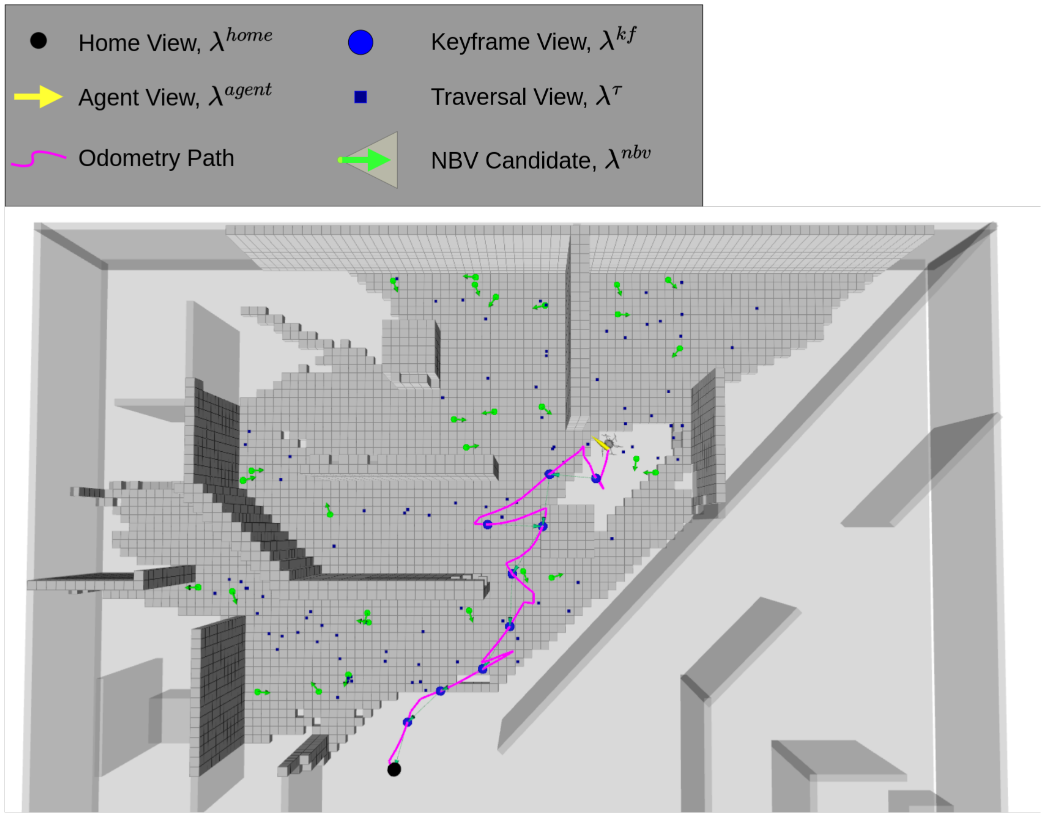
The set of hyperedges $\mathcal{C} = \{\mathcal{H}\} \in \mathcal{P}(\mathcal{V})$, represents a hierarchical decomposition of $\mathcal{V}$, where $\mathcal{P}$ is the power set. Each hyperedge $\mathcal{H} \subseteq \mathcal{V}$ contains a unique subset of $\mathcal{V}$, where the set of hyperedges collectively provide a topological structuring over the underlying nodes.

5.1 Graph Nodes

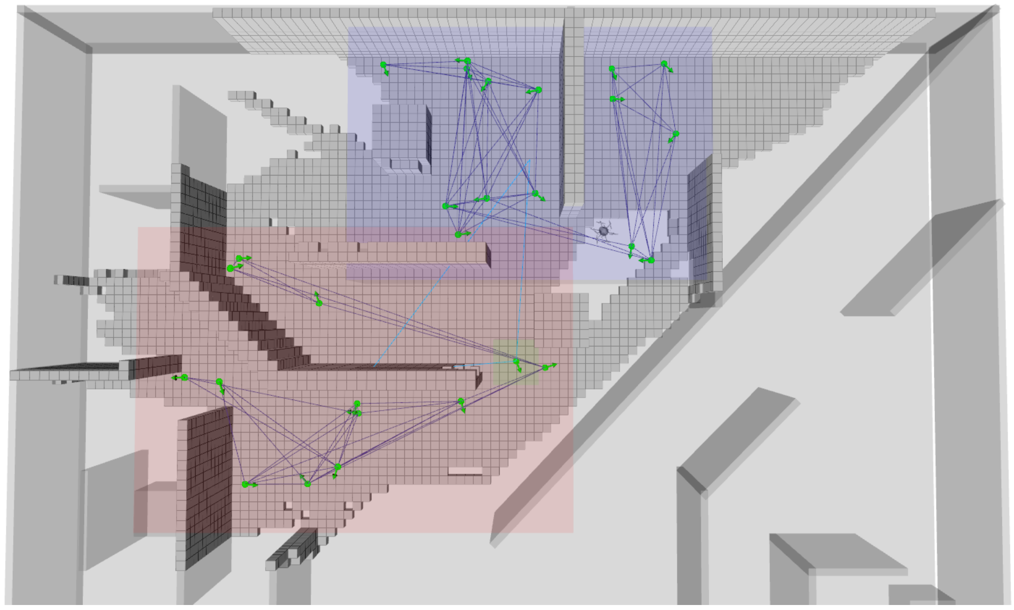
Each node $v_i \in \mathcal{V}$ corresponds to a viewpose information structure that consists of the tuple

$$v_i = \{\boldsymbol{q}_i, \gamma_i, \mathbb{1}_i^{open}\}, \tag{5.2}$$

where $\boldsymbol{q}_i$ is its pose which has an associated viewpose $\boldsymbol{q}_i \mapsto \lambda_i$, and $\gamma_i \in \mathbb{R}$ is a reward metric that quantifies the expected information gain available from λ_i. The node's visitation state is stored by a Boolean indicator $\mathbb{1}_i^{open} : v_i \mapsto \mathbb{B}$, corresponding to whether the robot has visited the pose of v_i. A *true* value indicates the node


Home View, λ^{home}
Keyframe View, λ^{kf}
Agent View, λ^{agent}
Traversal View, λ^{τ}
Odometry Path
NBV Candidate, λ^{nbv}

(a) Visualization of the APN graph in terms of its node classifications (see figure legend) and traversal edges (blue lines).



(b) APN hyperedge clusters, visualized by a bounding box enclosing the contained nodes. Each cluster forms an induced subgraph over the intra-cluster edges (orange lines).

Figure 5.1: Depictions of the APN composition.

is unvisited, also referred to as *open*, and is otherwise referred to as *closed* if it has already been visited. This is used to discriminate between the open set of nodes $\mathcal{V}^{open}$ which can represent goal candidates, and the closed set $\mathcal{V}^{closed}$ of nodes which have already been visited.

Several important classifications are defined over $\mathcal{V}$ based on their properties. These provide an increased understanding of how the network can serve different tasks. These are depicted in Figure 5.1a and are summarized as follows:

- **Terminal nodes**: the robot's initial pose $\boldsymbol{q}_0^{agent}$ is used to define the *home state*, represented by a unique node v^{home} that remains fixed over the lifetime of the APN. This is depicted by the black colored node in Figure 5.1a. A unique node $v^{agent} \in \mathcal{V}$, referred to as the *agent node*, is used to represent the robot and is dynamically updated with the robot pose as it changes over time. This is indicated in yellow in Figure 5.1a. These represent the terminal poses between the start and endpoints of the explored path.

- **Keyframe nodes**: the previously traversed path of the robot is represented by a path-connected set of *keyframe nodes*, $\boldsymbol{q}_{0:t}^{agent} \mapsto \{v_{0:k}^{kf}\} \in \mathcal{V}^{kf}$, rooted at the home state, $v_0^{kf} = v^{home}$. Keyframe nodes are added in intermediate intervals once the robot has traveled a minimum distance from the last keyframe. These encode the intermediate poses between the terminal nodes, and ensure a direct path to the home state is always known.

- **NBV nodes**: unvisited nodes with positive information gain are classified as candidate *NBV nodes*, represented by the set $\mathcal{V}^{nbv} = \{v \in \mathcal{V}^{open} : \gamma(v) > 0\}$. Each NBV node represents a candidate subgoal for for navigation and planning that is expected to increase map knowledge.

- **Traversal nodes**: any node not classified as an NBV node is considered a *traversal node*, $\mathcal{V}^{\mathcal{X}} = \mathcal{V} \setminus \mathcal{V}^{nbv}$. These represent configurations only useful for traversal purposes, but are not expected to increase map knowledge if visited.

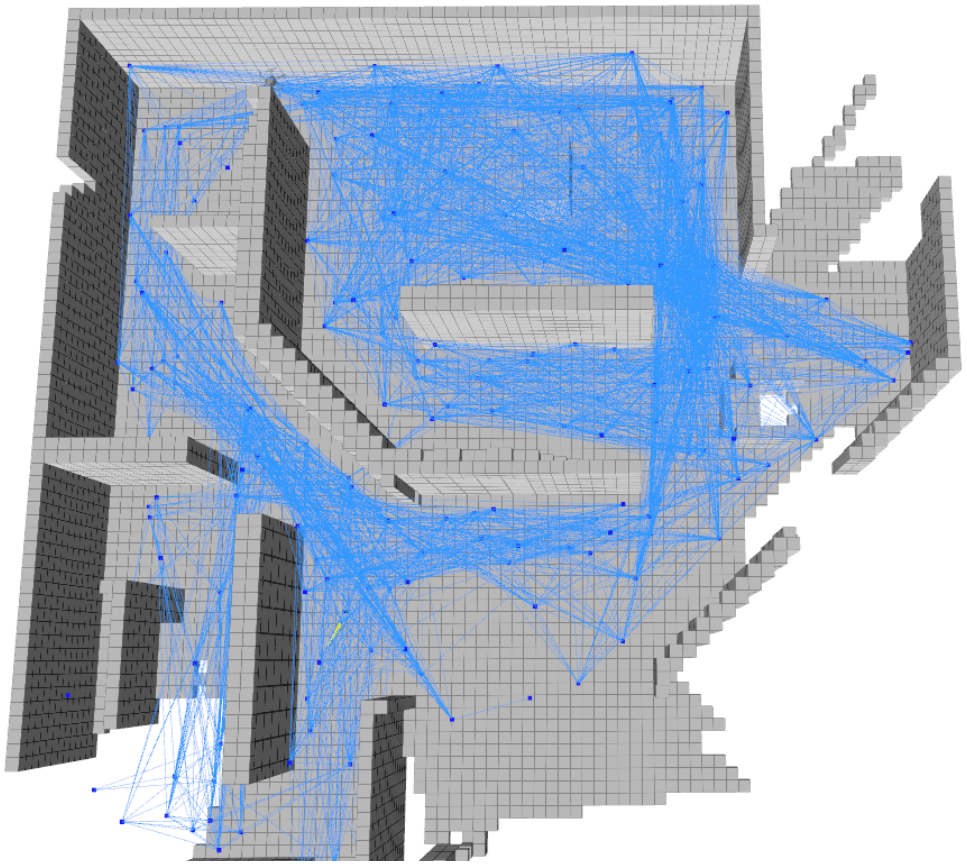


Figure 5.2: Visual depiction of the APN graph edges within a partially built map, where edges are shown as blue lines connecting between nodes to indicate the existence of a collision-free path.

5.2 Graph Edges

Graph edges are depicted in Figure 5.2 within a representative example of a partially built map. Each edge $e_{u,w} \in \mathcal{E}$ corresponds to the pair of nodes $\langle v_u, v_w \rangle$, and stores various analytical information of the traversal space between the pair as follows:

$$e_{u,w} = \{d^{\boldsymbol{x}}, d^{\psi}, L, OBB, l^{\mathcal{O}}, \boldsymbol{p}^{obs}\}, \tag{5.3}$$

where $d^{\boldsymbol{x}}$ and d^{ψ} are the Euclidean distance and the orientation angle distance, respectively, between (v_u, v_w). L is the evaluated cost metric value to traverse the edge given the maximum velocity $\boldsymbol{v}_{max}$ and yaw rate $\dot{\psi}_{max}$, defined by:

$$L(e_{u,w}) = max\left(\frac{d^{\boldsymbol{x}}(e_{u,v})}{\boldsymbol{v}_{max}}, \frac{d^{\psi}(e_{u,v})}{\dot{\psi}_{max}}\right). \tag{5.4}$$

Each edge also stores the Oriented Bounding Box (OBB) enclosing the endpoints, and the collision state of the space contained in the OBB is stored by $l^{\mathcal{O}} : OBB \rightarrow \{free, unk, obs\}$. $\boldsymbol{p}^{obs}$ is used as a memory cache that stores any uncertain voxels found from previous collision checks. This allows for lazy evaluation during future checks by first checking if these discrete voxels have changed, rather than the full OBB volume, to greatly reduce complexity.

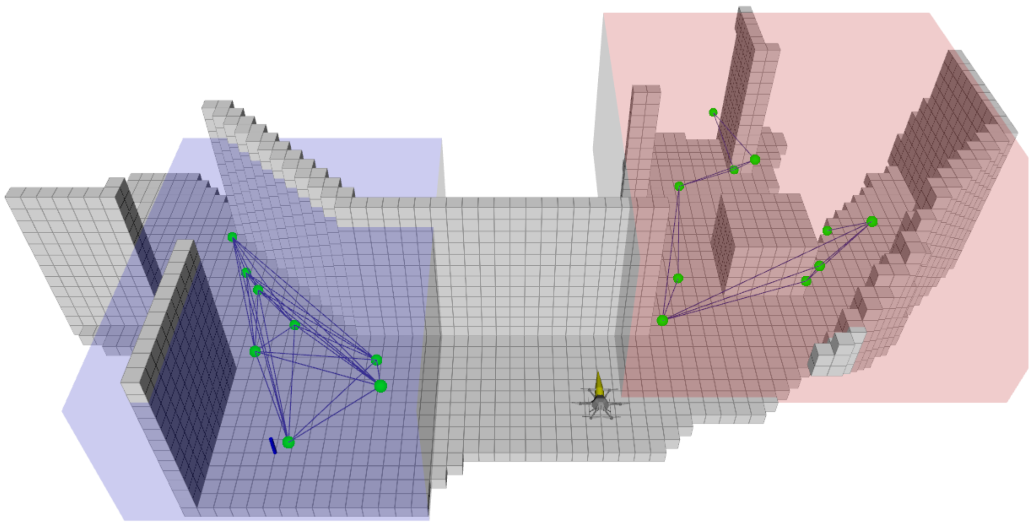
5.3 Graph Hyperedges

The set of hyperedges $\mathcal{C} = \{\mathcal{H}\} \in \mathcal{P}(\mathcal{V})$ are used for the purpose of providing a reduced resolution representation of the underlying graph $\mathcal{G}$ according to a topological decomposition. This allows a cluster of nodes, $\{v\} \subseteq \mathcal{V}$, to be represented by a single hyperedge element, $\mathcal{H} \in \mathcal{C}$, and a single topological edge between different hyperedge clusters can be used to replace the dense pairwise edges between their underlying nodes. This serves as a topological approximation of the underlying graph structure with greatly reduced size and complexity. This allows graph operations like search and traversal to be performed more efficiently at low-resolution at the global scale, while still allowing such operations to be performed at high-resolution locally within each cluster. Each hyperedge is modeled by the following:

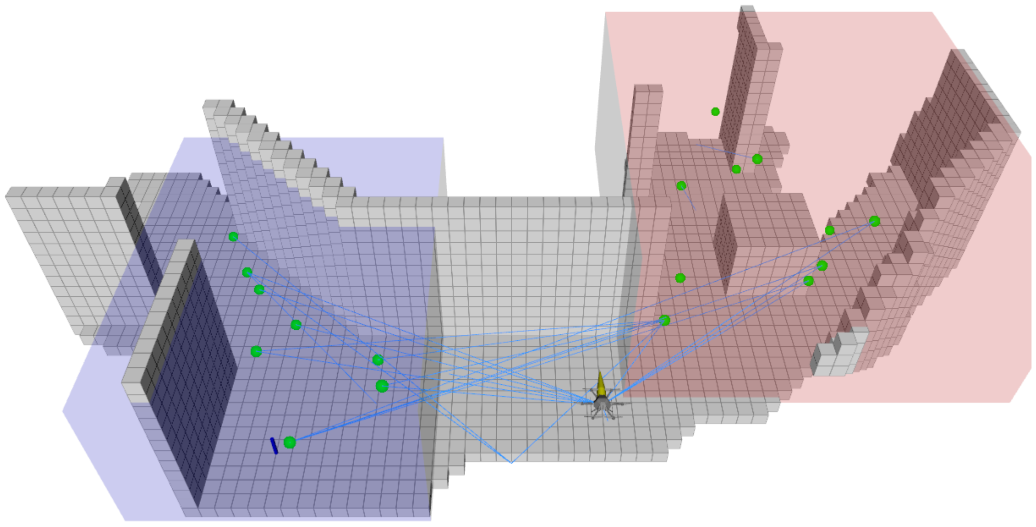
$$\mathcal{H}_i = \{\mathcal{V}_i^{\mathcal{C}}, \mathcal{A}_i, B_i, \boldsymbol{x}_i\}, \tag{5.5}$$

where $\mathcal{V}_i^{\mathcal{C}}$ is the set of nodes belonging to $\mathcal{H}_i$, with the centroid of the contained nodes given by $\boldsymbol{x}_i$ and its bounding volume given as B_i. $\mathcal{A}_i = G[\mathcal{H}_i]$ is the vertex-induced subgraph formed by each cluster containing the clustered nodes $v \in \mathcal{H}$ and the induced edges $(e_{u,w} \in \mathcal{E} : v_u, v_w \in \mathcal{V}_i^{\mathcal{C}})$ with both endpoints belonging to $\mathcal{A}_i$. A basic example of two clusters is illustrated in Figure 5.3.

Induced edges of a cluster $\mathcal{E}[\mathcal{H}_i]$ are referred to as its *interior edges*, shown in Figure 5.3a, while the remaining edges $\mathcal{E} \setminus \mathcal{E}[\mathcal{H}_i]$ that connect different cluster groups are referred to as *exterior edges*, shown in Figure 5.3b. The efficiency of global search queries and traversal through $\mathcal{G}$ can greatly increased by traversing between subraphs using their exterior edges, using the interior edges of the subgraphs to perform local operations as needed.



(a) Interior cluster edges.



(b) Exterior cluster edges.

Figure 5.3: Illustration of hyperedge clusters and their induced subgraphs. Clusters are indicated by the colored bounding boxes enclosing its contained nodes. The interior edges are shown in (a) connecting between nodes of same cluster, and exterior edges are shown in (b) connecting nodes between different clusters.

CHAPTER 6: DIFFERENTIAL REGULATION

The APN is incrementally built by the process of *Differential Regulation* (DFR), which manages how information is added, removed, or modified in the APN with respect to the concurrently built spatial map. DFR evaluates the APN according to a set of objectives and constraints conditioned on the current map, and executes a set of modifying procedures on the APN as needed to ensure they remain satisfied as the map evolves.

A diagram of these procedures is shown in Figure 4.2, and detailed in the following subsections. Their broad purpose is summarized as follows:

1. re-evaluate analytical variables that were computed with respect a prior map state to ensure their continuity (e.g. information gain)

2. add node and edge elements to increase the extent and completeness of the network

3. minimize overall size complexity by pruning unnecessary or redundant nodes and their edges

4. recompute the topological clustering to reflect changes made to the graph composition by the other update procedures

6.1 Reconditioning

Each DFR cycle i begins at a time t with the latest spatial map $\mathcal{M}_{t(i)}$, frontiers $\mathcal{F}_{t(i)}$, and robot pose $\boldsymbol{q}_{t(i)}^{agent}$. The first task is to determine the local differences of these variables to their states from the previous cycle $t(i-1)$. Each incremental map update reports the set of state-changed voxels, which are accumulated in a local cache $\Delta\mathcal{M}$

with its bounding volume ΔB. This is defined as the *local difference neighborhood* and is used to inform various APN update procedures about where state-changes have occurred, described further in the next subsections.

Each regulation cycle then begins by updating the pose of the agent node v^{agent} and its local edges. The length of the local path is then checked and compared against a keyframe threshold distance. If the threshold is exceeded, a new keyframe view v^{kf} is created from v^{agent} and added to the keyframe set $\mathcal{V}^{kf}$, with an edge connection to the previous keyframe to ensure a connected path to the home location is always maintained.

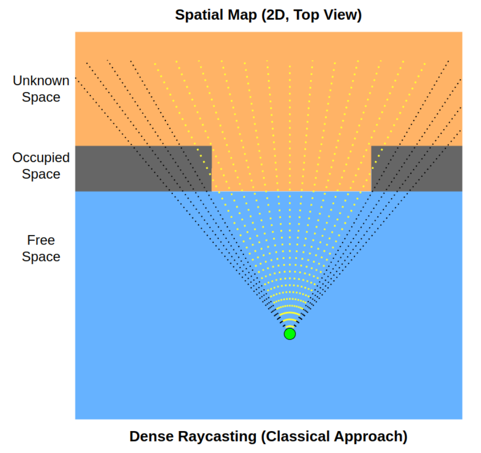
6.2 View Analysis and Coverage Sampling

$\mathcal{V}^{nbv}$ represents the set of NBV subgoal candidates expected to observe currently unknown voxels, such that map coverage will be increased if a subgoal is visited by the robot. To support the purposes of non-myopic planning, $\mathcal{V}^{nbv}$ should be sufficiently distributed to provide maximum coverage of the unknown map space. Additionally, maximum coverage should be achieved using a minimal size of $\mathcal{V}^{nbv}$ to reduce the eventual planning complexity that rapidly increases with the number of views considered.

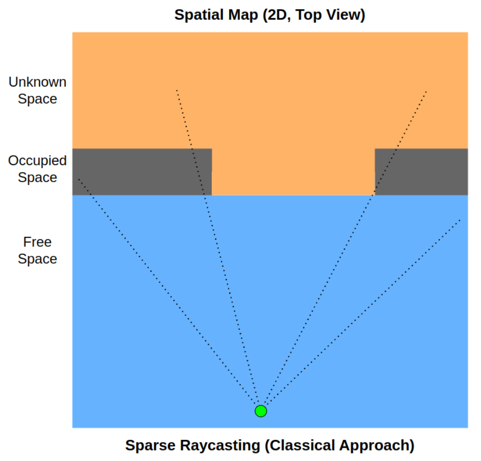
A sampling-based approach is used to incrementally build $\mathcal{V}^{nbv}$ to maintain maximum coverage as the map evolves. To efficiently and scalably achieve the aforementioned characteristics desired of $\mathcal{V}^{nbv}$, we introduce an approach using a frontier-based heuristic to evaluate information gain and also guide the sampling of additional views.

6.2.1 Frontier-based Information Gain

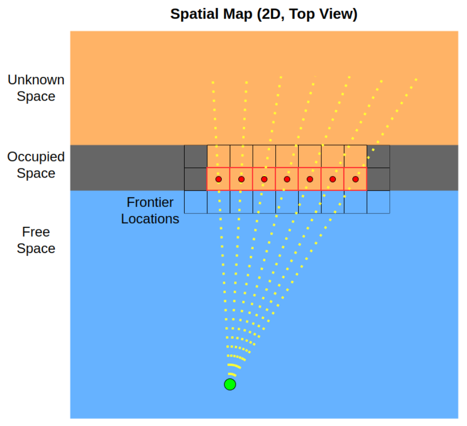
A classical approach in the literature to evaluate the expected information gain of a viewpose is by projecting a dense set of raycasts from the origin of a view, tracing the voxels intersected by each ray within its FoV, as depicted in Figure 6.1a. This has a high computational cost that can become prohibitive when evaluating many views and as the map resolution increases. Sparse raycasting methods have also been used to reduce the complexity, shown in Figure 6.1b, by only considering a reduced subset of raycasts within the FoV. However, this approximation can overlook information that lies in between adjacent rays.


Spatial Map (2D, Top View)
Unknown Space
Occupied Space
Free Space
Dense Raycasting (Classical Approach)

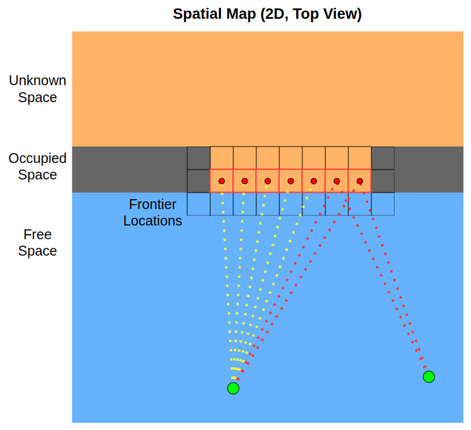
(a) Dense raycasting (classical approach)


Spatial Map (2D, Top View)
Unknown Space
Occupied Space
Free Space
Sparse Raycasting (Classical Approach)

(b) Sparse raycasting (classical approach)



(c) Frontier-based information gain



(d) Mutual information

Figure 6.1: Visualization of frontier-based information gain measurement.

Additionally, the resulting information gain is usually encoded and stored as a single numeric quantity, which excludes the specific voxels involved in its computation. This makes it impossible to determine the overlapping mutual information between different views, such that only the independent information gain of each view can be known. This increases the complexity of non-myopic planning, since the information gain of each view in a sequence can depend on the information gain already visible by the previous views in the sequence, and a separate optimization is needed to handle this.

To mitigate these drawbacks, we directly use the frontier voxels within a view's FoV to constrain the evaluation of information gain, illustrated in Figure 6.1c. Total information gain for an individual raycast corresponds to the number unknown voxels intersecting the ray, conditioned that they are not occluded by any occupied voxels that were previously intersected. To satisfy the visibility conditions, the first unknown voxel visible along a ray must always be preceded by free voxels. Then, any subsequent voxels can only be preceded by either a free voxel, or another unknown voxel, to be considered visible.

The initial unknown voxel visible along a ray naturally corresponds to a frontier boundary since must be preceded by a free voxel. Therefore, any raycast will always cross a frontier boundary before it passes through any additional unknown voxels that represent its information gain. Using this insight, the frontiers within the FoV correspond exactly to the potential "entry points" into larger regions of information gain. In other words, if a dense raycasting operation were performed, only the subset of rays that pass through frontier voxels are capable of discovering information gain. Raycasts can thus be directed towards only these locations, safely ignoring evaluation of other directions without loss of precision.

A *visibility map* $\Gamma : \mathcal{V} \rightarrow \mathcal{F}$ is used to store the visible frontier features of each viewpose, as depicted in Figure 6.2, which is formulated according to:

$$\Gamma(\lambda) = \{f \in \mathcal{F} : Vis(\boldsymbol{m}_f, \lambda)\}, \tag{6.1}$$

where $\boldsymbol{m}_f$ is the voxel associated to f, and Vis is an indicator function returning true if $\boldsymbol{m}_f$ is visible from λ. An *inverse visibility map* $\Upsilon : \mathcal{F} \rightarrow \mathcal{V}$ represents the preimage of Γ storing the viewposes from which each frontier is visible as

$$\Upsilon(f) = \{\lambda \in \Lambda : Vis(\boldsymbol{m}_f, \lambda)\}. \tag{6.2}$$

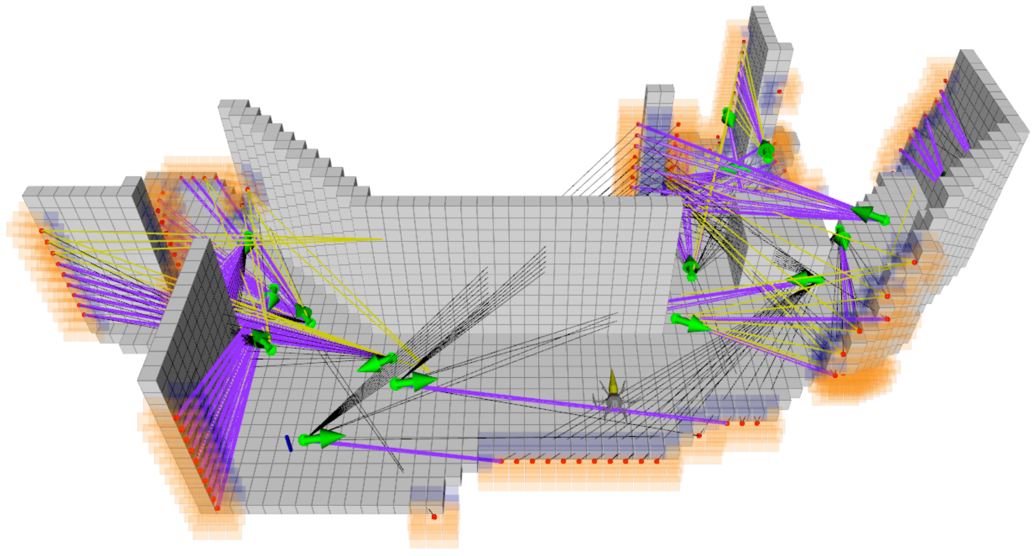


Figure 6.2: Visualization of the frontier visibility map, where green arrows indicate the APN views, and frontiers are indicated by red points. Yellow lines correspond to visible frontiers by a view, purple lines indicate the frontier is visible only from a unique view, and black lines indicate the frontier is occluded.

The *individual gain*, $\mathcal{K}$, of a view λ refers to the independent amount of unknown space visible from the view. This measure can be lower bounded by the number of visible frontiers $\mathcal{K} : \Lambda \mapsto |\Gamma(\lambda)|$, since each frontier corresponds to an unknown voxel location. The *joint gain*, $\mathcal{J}$, refers to the unique information collectively visible from a set of views. These can be respectively formulated as follows:

$$\mathcal{K}(\lambda) = |\Gamma(\lambda)|, \tag{6.3}$$

$$\mathcal{J}(\Lambda) = |\bigcup_{\lambda \in \Lambda} \Gamma(\lambda)|. \tag{6.4}$$

The *exclusive gain*, $\mathcal{I}$, of a view λ refers to its unique contribution to the joint gain, or, in other words, the exclusively information visible by λ that is not visible by any other view in Λ. $\mathcal{I}$ can be determined according to the visible frontiers of $\Gamma(\lambda)$ that are only observed by λ. This can be efficiently computed in linear time on the number of visible frontiers by:

$$\mathcal{I}(\lambda) = |\{f \in \Gamma(\lambda) \ : \ |\Upsilon(f)| = 1\}|. \tag{6.5}$$

6.2.2 View Sampling for Coverage Maximization

An iterative objective of DFR is to ensure maximum coverage of the current unknown space is maintained. Υ supports evaluation of the coverage completeness of the unknown map space by the current views Λ. Let $\mathcal{F}^{cvr}$ represent the set of covered frontiers, where a frontier is considered covered if it has at least one covering view able to observe it according to Υ. The residual set is represented as $\mathcal{F}^{\overline{cvr}} = \mathcal{F} \setminus \mathcal{F}^{cvr}$, and the global coverage completeness is evaluated by the fraction of covered frontiers, $\mathcal{F}^{cvr}/\mathcal{F}$. The iterative coverage maximization objective can be formulated as:

$$\max \frac{|\mathcal{F}^{cvr}|}{|\mathcal{F}|} = \max |\bigcup_{f \in \mathcal{F}} \{f \mid \exists \lambda \in \Lambda, Vis(\boldsymbol{m}_f, \lambda)\}|. \tag{6.6}$$

A frontier-guided sampling strategy is presented to perform the maximization of (6.6) by iteratively sampling viewposes to observe the non-covered frontiers. This effort is concentrated within ΔB which contains the most recent changes to the frontier distribution. Given the high complexity potentially involved in the sampling procedure, a performance tuning parameter $p^{\lambda}_{local} \in (0, 1]$ is provided, representing a probability threshold used to select a random subset of the frontiers in ΔB to be considered for sampling in the current cycle.

A second parameter $p^{\lambda}_{global} \in (0, 1]$ is provided which serves a similar purpose as p^{λ}_{local}, but is applied to any non-covered frontiers that lie outside of ΔB. This is to account for possible frontiers that were not successfully covered in a finite number of attempts during previous DFR cycles, which can result when large amounts of occupied or unknown space exist near a frontier. The difficulty in finding a feasible viewpose can greatly increase for these cases, and in some cases one may not exist with the available map knowledge. Given the increased difficulty, p^{λ}_{global} is given a lesser value than p^{λ}_{local}, allowing the search effort to persist between DFR cycles but with lower priority. In effect, this offers a degree of probabilistic completeness as the likelihood of finding a valid sample, if one exists, can continually increase over time while reducing the individual search effort per DFR cycle.

The sampling procedure is given in Alg. 1, which begins by calling *recondition-Visibility* to update the visible information of existing views withing the changed volume. Between cycles, the frontier boundaries are often pushed back by only a small amount, but remain visible within the many of the same view as the previous cycle. This step ensures these differences are updated, so sampling is only needed when frontiers are pushed beyond visibility of all existing views.

Next, a frontier queue $\widehat{\mathcal{F}}$ is initialized containing the selected subsets from $\mathcal{F}^{\overline{cvr}}$. For each $f_i \in \widehat{\mathcal{F}}$, a sampling subspace B_{f_i} is computed from which f_i can potentially be observed given the sensing parameters. For a maximum of $N^{attempt}_{nbv}$ attempts, viewposes are randomly sampled using *getCoverageSample* and checked by *isValid-Sample* to determine if a valid sample has been found. A sample is considered valid only if it is collision-free and successfully observes the current frontier target, f_i.

Upon finding a valid sample, it is used to add a new node to the network, and all of its visible frontiers are computed to update the visibility map. If any of these frontiers are contained in $\widehat{\mathcal{F}}$, they are removed since they have been already covered by the current sample. This can greatly reduce the number of samples, since in practice a single view will often be able to observe many nearby frontiers.

Algorithm 1: Frontier-guided view sampling for information gain maximization

```
1  $reconditionVisibility(\mathcal{G}, \Delta B)$
2  $\widehat{\mathcal{F}} \leftarrow frontierQueueInit(\mathcal{F}^{cvr}, \Delta B, p^{\lambda}_{local}, p^{\lambda}_{global})$
3  while $\widehat{\mathcal{F}} \neq \emptyset$ do
4      $f_i \leftarrow extractNext(\widehat{\mathcal{F}})$
5      $B_{f_i} \leftarrow getSamplingVol(f_i, d^{sense}_{max}, \alpha_s)$
6      $success \leftarrow false, n \leftarrow 0$
7      while $n < N^{attempt}_{nbv}$ & $\neg success$ do
8          $\bar{\boldsymbol{q}} \leftarrow getCoverageSample(B_{f_i})$
9          if $isValidSample(\bar{\boldsymbol{q}})$ then
10             $\lambda_j = addNode(\bar{\boldsymbol{q}}, \mathcal{G})$
11             $\mathcal{F}^{vis}_{\lambda_j} \leftarrow computeVisible(\lambda_j, f_i)$
12             $updateVisibility(\lambda_j, \mathcal{F}^{vis}_{\lambda_j})$
13             $\widehat{\mathcal{F}} = \widehat{\mathcal{F}} \setminus \mathcal{F}^{vis}_{\lambda_j}$
14             $success \leftarrow true$
15         end
16         $n = n + 1$
17     end
18 end
```

6.3 Pruning and Refinement

The growth rate of the network is reduced by pruning unnecessary views that no longer provide any individual gain contribution, or redundant views with little or no exclusive information gain. These conditions naturally occur as the robot progresses its exploration of the map and observes the previously unknown space within each view. They also occur as a result when new view samples are added to the network which overlap with the pre-existing views, decreasing their exclusive gain.

The goal of pruning is to identify the views that can be removed from the network without loss of the overall joint gain, which is equivalent to finding the minimum subset of views needed to cover the current unknown space. This is formulated as a submodularity maximization problem, supported through the use of the joint gain and exclusive gain measures. Given an initial set of views Λ, pruning can be described as follows:

$$\underset{\Lambda^* \subseteq \Lambda}{\operatorname{argmin}}(\Lambda^*),$$
$$\text{s.t.} \tag{6.7}$$
$$\mathcal{J}(\Lambda) - \mathcal{J}(\Lambda^*) \approx 0.$$

To solve (6.7), a set of pruning candidates is found by searching for views that have negligible individual or exclusive information gain. Given the local difference neighborhood ΔB, the search is restricted to the views located within visible range d_{max}^{sense} of ΔB, corresponding to the views with visibility information that was potentially effected by the map changes. The candidates within this region are further evaluated for their edge connectivity. Any candidate found to have a cut-edge is preserved to maintain the graph connectivity, while the remainder are deleted.

Once the pruning stage is complete, the coverage views of $\mathcal{V}^{nbv}$ represent the supremal set that maximizes map coverage using a minimal number of views. Not only does this help to reduce the total size, but it minimizes redundant coverage to help simplify the planning problem. Since each NBV has some positive amount of exclusive gain after pruning, they represent an exact set of targets that a planner must determine how to optimally visit, without the need to evaluate their redundancy during its search.

6.4 Reachability Update

The reachability knowledge represented by $\mathcal{E}$ is updated each iteration to account for new map knowledge and any state changes in $\mathcal{V}$. Additional nodes are also sampled during this stage to increase the overall node density and uniformity in $\mathcal{V}^{\mathcal{X}}$. This accounts for the non-uniformity of coverage view sampling, which is biased towards the frontier boundaries. Since the purpose of $\mathcal{V}^{\mathcal{X}}$ is primarily to increase the network connectivity, only the position of these samples is needed, while the visible information and pose orientation attributes can be ignored.

Algorithm 2: Reachability expansion algorithm.

```
1  $\widehat{B} \leftarrow getSearchVolume(\Delta B, d^{sample}_{traversal})$
   // Stage 1: increase node density
2  $i, n \leftarrow 0$
3  while $i < N^{attempt}_{traversal}$ & $n < N^{sample}_{traversal}$ do
4      $\boldsymbol{x}_i \leftarrow generateReachabilitySample(\widehat{B})$
5      $\boldsymbol{x}_{near} \leftarrow findNearest(\boldsymbol{x}_i, \Lambda)$
6      if $distance(\boldsymbol{x}_i, \boldsymbol{x}_{near}) > d^{sample}_{traversal}$ then
7          $addNode(\mathcal{G}, \boldsymbol{x}_i)$
8          $n = n + 1$
9      end
10     $i = i + 1$
11 end
   // Stage 2: increase edge density
12 $\mathcal{E}_{local} \leftarrow getUncertainEdgePairs(\widehat{B}, p^{e}_{update})$
13 for $e_i \in \mathcal{E}_{local}$ do
14     $OBB, l^{\mathcal{O}}, \boldsymbol{p}^{obs} \leftarrow computeEdgeState(e_i)$
15     if $l^{\mathcal{O}} = free$ then
16         $addEdge(\mathcal{G}, e_i)$
17     else if $l^{\mathcal{O}} = unk$ then
18         $cacheUncertainEdge(\mathcal{G}, e_i, OBB, l^{\mathcal{O}}, \boldsymbol{p}^{obs})$
19     else if $l^{\mathcal{O}} = occ$ then
20         $cacheCollisionEdge(\mathcal{G}, e_i)$
21     end
22 end
```

The pseudocode for the reachability update procedure is shown in Alg. 2, which contains two primary stages. The first stage samples traversal nodes to increase the distribution density within the graph, and the second stage increases the total edge density.

In the first stage, collision-free positions are uniformly sampled from $\widehat{B}$, for a maximum of $N^{attempt}_{traversal}$ attempts, or until a threshold of $N^{sample}_{traversal}$ samples are accepted. Each sample is evaluated according to the distance of its nearest neighbor in Λ, and compared against a threshold distance, $d^{sample}_{traversal}$. $d^{sample}_{traversal}$ serves as a density constraint to prevent too many samples from being added in close proximity, which would unnecessarily increase the size complexity of the graph while adding little or no additional reachability knowledge. A sample is accepted if its nearest neighbor distance is greater than $d^{sample}_{traversal}$, and a new node is added to the graph using the sampled position.

The second stage begins by extracting the local set of candidate edge pairs $\mathcal{E}_{local}$ using the function *getUnknownEdgePairs*. This procedure searches $\widehat{B}$ to find the set of node pairs (v_u, v_w) such that the collision state of the corresponding edge $e_{u,w}$ is either null or unknown. Here, a null edge indicates the edge does not exist (i.e. has not been evaluated in any DFR cycle), while unknown refers to an edge found with an uncertain collision state from a previous DFR cycle. A parameter $p^{e}_{update} \in [0, 1)$ is used to specify a random probability threshold of whether to evaluate a candidate node pair (v_u, v_w). This helps to limit the number of edge evaluation operations that occur per cycle, similar the parameter p^{λ}_{local} used for coverage view sampling.

Each edge is evaluated by *computeEdgeState* to determine its collision state data, which leverages previously cached results if available. Since edges may be evaluated between any nodes over any distance within $\widehat{B}$, the cached collision data can significantly reduce the update complexity. If an occupied collision is found, the edge is added to the cache of collision edges to prevent future evaluation, as illustrated in Figure 6.3. For unknown voxel collisions, the edge is added to the cache of uncertain edges along with the intermediate collision data results to accelerate future re-evaluation. Otherwise, the edge is added to the graph by *addEdge* which computes and stores its associated cost information according to (5.3) for efficient lookup by other procedures and planning.

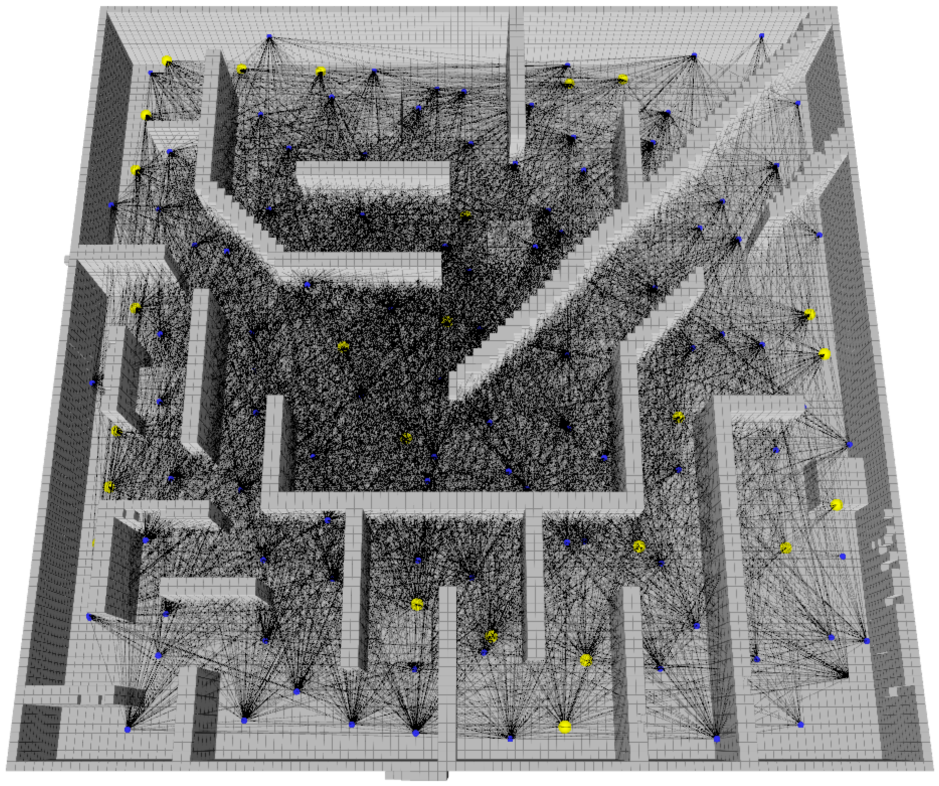


Figure 6.3: Visual depiction of cached collision edges used to reduce redundant computations.

6.5 Topological Clustering

The graph nodes are decomposed into a set of subgraph regions represented by the hyperedges $\mathcal{C}$, as illustrated in Figure 5.1b. $\mathcal{C}$ serves as a topological hierarchy over $\mathcal{G}$ to reduce its size complexity. This representation can be utilized to increase the efficiency for search, traversal, and other operations. A tradeoff occurs where greater reductions in size complexity also result in reduced level of detail (LoD), i.e. resolution.

To compute the hyperedges, we use a density-based clustering approach based on [95, 96], extended to leverage both the geometric and reachability knowledge already present in the APN. The algorithm uses two parameters, D_c and ρ_c, where D_c defines neighborhood distance threshold, and ρ_c defines a density threshold for the neighborhood.

Let a node v_p be defined as a *core node* if it has at least ρ_c edge-connected neighbors within distance D_c. A node v_q is then defined as a *reachable node* from v_p only if there exists an edge connection between v_p and v_q, and v_q is within distance D_c from v_p. Given a core node v_p, a cluster is formed by all nodes reachable from v_p. Any remaining nodes that are neither core nodes nor reachable from a core node are assigned as singleton clusters.

This approach allows clusters to form more naturally by additionally considering the edge connectivity between points. They are also not required to be geometrically convex as with other clustering approaches. This enables fewer clusters to be formed, since they can be better fit to the nodes over arbitrarily shaped space. Explicit constraints on the maximum number of clusters or their size are also not necessary, such that clusters can conform to the map with variable size and density, which can effectively handle environments where different regions may have different geometric characteristics and complexities.

CHAPTER 7: NON-MYOPIC EXPLORATION PLANNING

The iteratively updated APN provides a graph-theoretic model that aggregates the total information space needed for planning. It can then be utilized by any graph-based planning strategy for global and local planning. In this chapter, we present a non-myopic exploration planning approach referred to as the APN Planner (APN-P), which is responsible for planning and replanning the optimal navigation goals for exploration. It receives each updated APN state as its input to perform non-myopic planning based on evolutionary optimization, and outputs the updated navigation plan. Additionally, the output plan is used as feedback to initialize the subsequent replanning iterations, allowing prior optimization efforts to be reused to offset the increasing problem size over time.

The underlying planning objective is to find the optimal sequence of NBVs to maximally observe the current unknown space, which is formulated as a Fixed-Ended Open Traveling Salesman Problem (FEOTSP). The APN and DFR help to simplify this problem by the application of pruning, which ensures that each NBV contributes some amount of unique information. As such, all NBVs are required to achieve maximum coverage, and planning must only consider the motion cost when searching for their optimal sequence order.

The FEOTSP problem is NP-hard, which can be prohibitive for online operation as the number of NBVs increase. To reduce the computational complexity and increase the scalability of planning, the hierarchical decomposition of the APN is leveraged to separate planning into multiple stages that operate at different levels of detail. A visualization of this procedure is displayed in Figure 7.1.

In the first stage, the global plan is coarsely estimated by finding the optimal

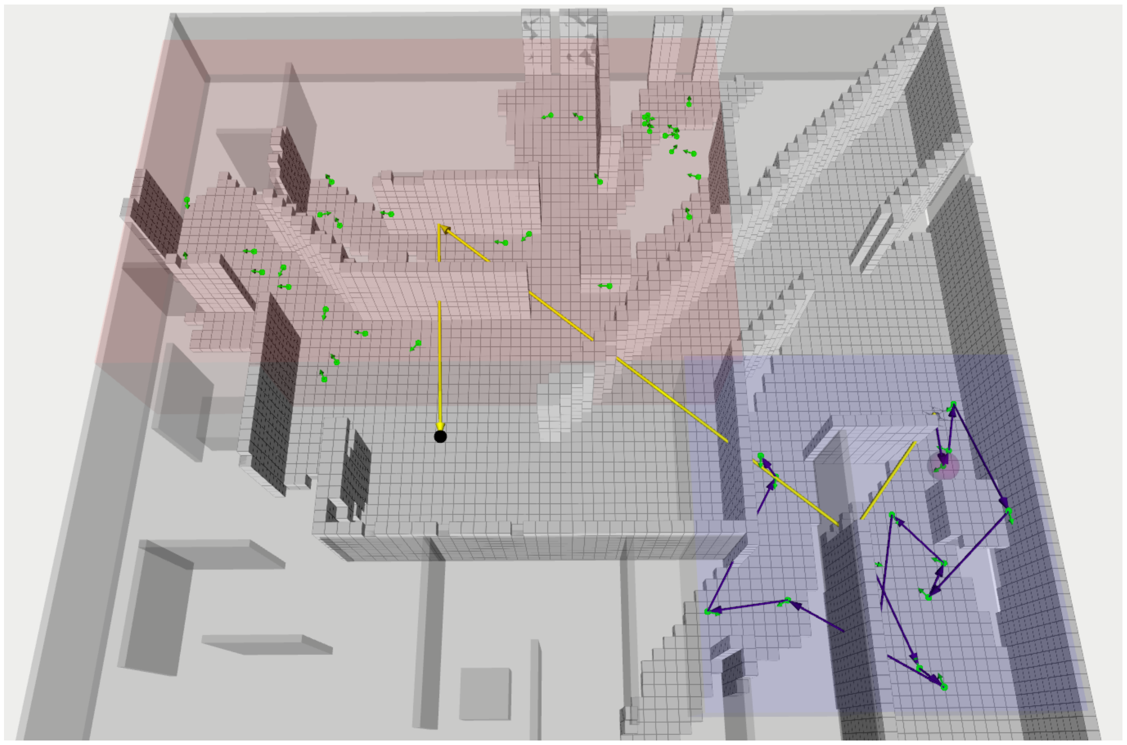


Figure 7.1: Visual depiction of the hierarchical planning strategy. The first stage computes the global path (yellow arrow path) using the hyperedge clusters, with the start fixed to the robot location and the end fixed to the home location. The second stage optimizes the NBV sequence (depicted using blue arrows) within the first cluster of the global sequence (green bounding box).

sequence order to visit each topological subgraph region, shown as the yellow arrow path in Figure 7.1. The solution to this stage effectively finds the relative ordering between each cluster of nodes, without specifying the particular order within each cluster. The second planning stage then operates on the global plan to optimize the local ordering of the views within a cluster. This only needs to be performed for the first cluster of the global sequence in order to determine the next navigation goal.

7.1 Adaptive Evolutionary Path Optimization

Each planning stage models a distinct problem instance of the FEOTSP, both of which are solved using mimetic evolutionary optimization [97]. A population of P_n candidates, or individuals, are initialized by randomized permutations of $\hat{\Pi}$. For a maximum of N_g generations, the population is optimized using a pairwise swap mutation and partially mapped crossover (PMX) [98]. The procedure terminates once N_g generations is exceeded, or an improved solution cannot be found after N_{stall} generations.

The optimized sequences of each stage are preserved by a data cache, allowing them to be reused in subsequent planning cycles. The previous solution is first cleaned to remove any nodes that were removed from the APN after its update, preserving the relative order of the remaining elements, and new APN nodes are inserted using local search optimization to determine their initial positions.

Typically, only a locally bounded region of the solution will be effected by this process, while much of it will stay the same. Over time, the unchanged parts of the solution will approach an optimal convergence, which in effect can allow the optimization efforts to focus on the smaller parts that are frequently changing. This helps constrain the complexity growth despite the increasing number of nodes as the environment is explored.

7.2 Topological Global Planning

Let $\mathbb{J}^{\mathcal{H}} \subset \mathbb{N}$ be an index set that enumerates the clusters $\mathcal{C}$. A cost matrix $\boldsymbol{m}^{\mathcal{H}}$ is computed by finding the shortest path between the centroids of each pair of clusters. Given the pairwise cost $s(u, w) \in \boldsymbol{m}^{\mathcal{H}}$ between cluster indices $u, w \in \mathbb{J}^{\mathcal{H}}$, the global planning objective is to find the minimum cost permutation $\Pi^{\mathcal{H}} \in \boldsymbol{S}_n(\mathbb{J}^{\mathcal{H}})$ of the indices $\mathbb{J}^{\mathcal{H}}$, where $\boldsymbol{S}_n(\mathbb{J}^{\mathcal{H}})$ is the symmetric group of $\mathbb{J}^{\mathcal{H}}$.

7.3 Local View Path Planning

View path planning is similarly formulated to global planning, with only minor changes needed regarding how the input data is initialized. Given the first cluster $\mathcal{V}_0^{\mathcal{C}}$ of $\Pi^{\mathcal{H}}$, its induced subgraph $\mathcal{G}[\mathcal{V}_0^{\mathcal{C}}]$ specifies the NBV targets applied for local planning. Given an index set $\mathbb{J}^{\Lambda} \subset \mathbb{N}$ enumerating the NBVs $\{\lambda\} \in \mathcal{G}[\mathcal{V}_0^{\mathcal{C}}]$, a pairwise cost matrix $\boldsymbol{m}^{\Lambda}$ between index pairs $u, w \in \mathbb{J}^{\Lambda}$ can be obtained directly from the existing edge costs. The view path planning objective is then to find the minimum cost permutation $\Pi^{\lambda} = (v^{agent}, \lambda_{\mathbb{J}^{\Lambda}_0}, \cdots, \lambda_{\mathbb{J}^{\Lambda}_n})$, which begins at the current robot configuration v^{agent} and visits each NBV node of the target cluster.

7.4 Navigation Goal Selection

Once the exploration plan optimization is complete, the first view of the local sequence represents the navigation goal, λ_g. If this goal is different from the previous goal, the cost of their respective sequences is compared to determine whether to accept or reject the new goal, penalizing significant changes in the direction of motion. Once the appropriate goal is selected, its trajectory is computed with a variant of *RRT* [99], using the APN to find the shortest path to initialize the trajectory planner. Exploration terminates once no frontiers remain, or no further feasible views can be found to observe any remaining frontiers.

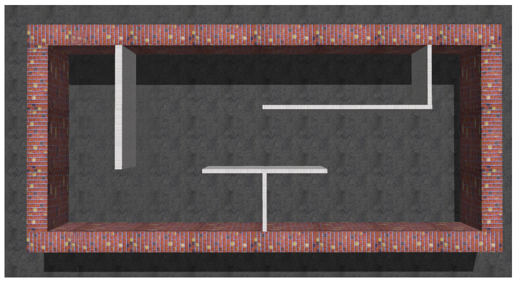
CHAPTER 8: EVALUATION

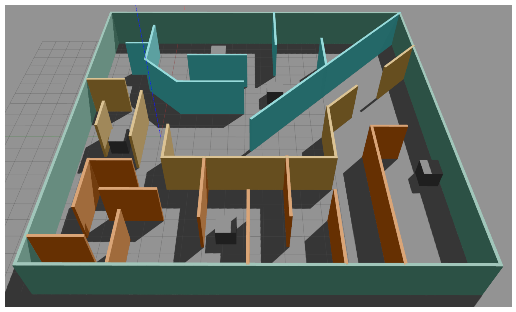
The APN and APN-P were evaluated through ROS-based simulations using Gazebo [100] and the RotorS MAV simulation framework [101]. The AscTec Firefly MAV model provided by RotorS was used to simulate the robot dynamics and control systems, and was equipped with a stereo depth sensor for visual perception. The simulations and all algorithms were executed using a single laptop computer with Intel Core i7 2.6 GHz processor and 16 GB RAM. The test results were used to analyze the computational performance and planning efficiency of the proposed approach.

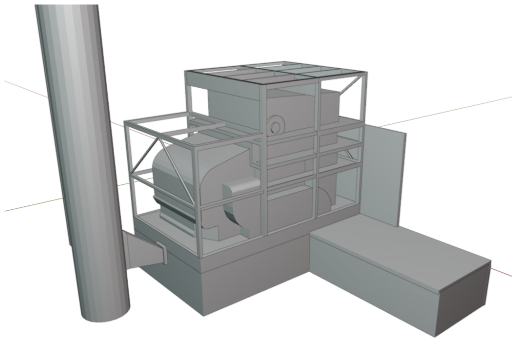
Exploration was tested using several different 3D structure models with various scales as displayed in Figure 8.1, with a visual comparison of their relative scales shown in Figure 8.1e. In addition to varying sizes, each environment provides different characteristics for evaluation, such as obstacle density, narrow spaces opposed to open space, dead-ends, and overall geometric complexity.

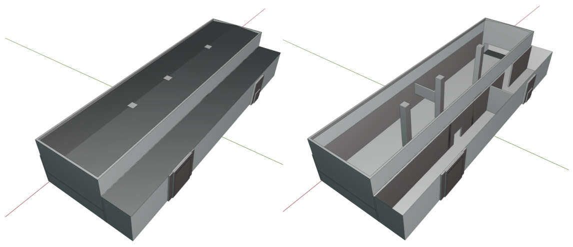
To account for the stochastic nature of the approach, each scenario was run 5 times and statistical analysis was computed over a variety of performance metrics, summarized in Table 8.1. The average total exploration runtime required to complete the exploration task is denoted as $\overline{T}$, and $\overline{t^c}$ refers to the average computational time required per update and planning cycle. A maximum exploration time limit of $T_{max} = 14\text{min}$ (840s) was imposed, which is the maximum rated flight time for the AscTec Firefly. If this threshold is exceeded, exploration immediately terminates and failure is reported.

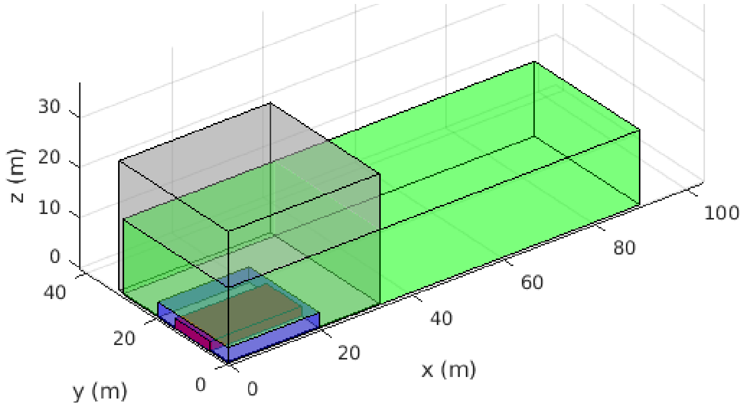
The total map coverage is given as the ratio $\vartheta_{\mathcal{M}}$ of the number of surface voxels $\mathcal{M}^{occ}$ discovered during exploration with respect to a ground truth set $\widehat{\mathcal{M}}^{occ}$ of all visible surface voxels. $\widehat{\mathcal{M}}^{occ}$ was determined by manually guiding the robot through


(a) Apartment


(b) Maze


(c) Industrial Plant


(d) Warehouse



(e) Relative environment scale comparison

Figure 8.1: Visualization of each evaluated world scenario. The relative scale of each scenario is depicted in 8.1e according to their bounding box dimensions, where red represents the Apartment (slightly offset from the origin for visual clarity), blue represents the Maze, grey represents the Industrial Plant, and green represents the Warehouse.

Table 8.1: Summary of performance analysis metrics.

Symbol	Description
$\overline{T}$	average total exploration runtime (s)
$\overline{t^c}$	average computational time per cycle (ms)
$\vartheta_{\mathcal{M}}$	final map surface coverage ratio (%) as $\mathcal{M}^{occ}/\widehat{\mathcal{M}}^{occ}$
$\eta_{\mathcal{M}}$	average voxel discovery rate (m^3/s)
$\eta_{\mathcal{M}^{occ}}$	average surface voxel discovery rate (m^3/s)
$\vartheta_{\mathcal{V}}$	avg. number of nodes per unit of map volume (1/100m^3)
$\vartheta_{\mathcal{E}}$	avg. ratio of known edges $\lvert\mathcal{E}\rvert$ to possible edges $\binom{\lvert\mathcal{V}\rvert}{2}$

each world scenario, carefully ensuring every observable surface was covered by the sensor. The total volumetric exploration rate is given as $\eta_{\mathcal{M}}$, which is the average volume of new information gain per second in m^3/s. Since the objective is to achieve complete surface coverage, a more useful metric is $\eta_{\mathcal{M}^{occ}}$ which refers to the rate of occupied information gain in m^3/s.

The APN is evaluated according to its average node density $\vartheta_{\mathcal{V}}$ and edge density $\vartheta_{\mathcal{E}}$. Here, node density refers to the number of nodes within a standard unit of volume, normalized as the number of nodes per 100m^3 of the mapped free space. Edge density refers to the ratio between the known edges $\lvert\mathcal{E}\rvert$ and the total edge capacity of a complete edge set over the nodes, $\binom{\lvert\mathcal{V}\rvert}{2}$. $\vartheta_{\mathcal{V}}$ and $\vartheta_{\mathcal{E}}$ are given as the average over all cycles of the test scenario.

Table 8.2: Summary of common configuration parameters.

	Scenario			
Param	Apt.	Maze	Ind. Plant	Warehouse
$r_{\mathcal{M}}$	$\{0.1, 0.2, 0.4\}$	$\{0.1, 0.2\}$	$\{0.2\}$	$\{0.4\}$
d_{safe}	0.75m			
$\boldsymbol{v}_{max}$	1.0m/s	2.0m/s	2.5m/s	3.0m/s
$\dot{\psi}_{max}$	0.75rad/s			
d^{sense}_{max}	5m	6m	7m	9m
α_v, α_h	[60°, 90°]	[60°, 90°]	[75°, 115°]	[75°, 115°]

The following baseline approaches were used for comparative analysis with the APN-P:

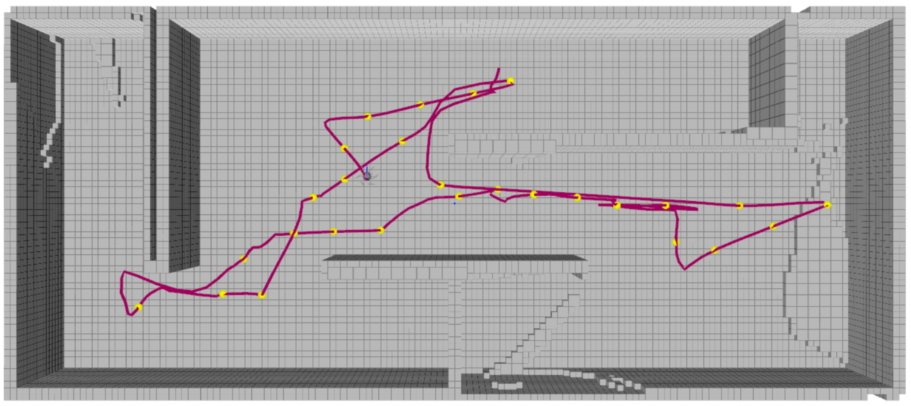
- RH-NBVP [78]: A receding horizon method that finds informative view paths using RRT-based expansion within a local region of the robot.
- AEP [79]: An approach that extends the strategy of RH-NBVP, using RH-NBVP for local planning and frontier-based planning for global search when local planning fails to find informative views.
- FFI [80]: A hybrid frontier-based and sampling-based approach that uses an efficient frontier clustering strategy to guide the sampling of views.
- Rapid [81]: An extension of frontier-based planning designed to maintain the fastest allowable velocity by guiding towards frontiers within the sensors current field of view, and using classical frontier planning when no visible frontiers are available.

A summary of common parameters for the different scenarios is shown in Table 8.2, which were selected as consistently as possible to the baseline approaches. The map resolution $r_{\mathcal{M}}$ was varied between the values $\{0.1, 0.2, 0.4\}$m to analyze its effects on performance scalability. The maximum linear velocity $\boldsymbol{v}_{max}$ and yaw rate $\dot{\psi}_{max}$ were assigned based on the common values used in the comparative approaches, along with the sensing parameters d^{sense}_{max} and (α_v, α_h).

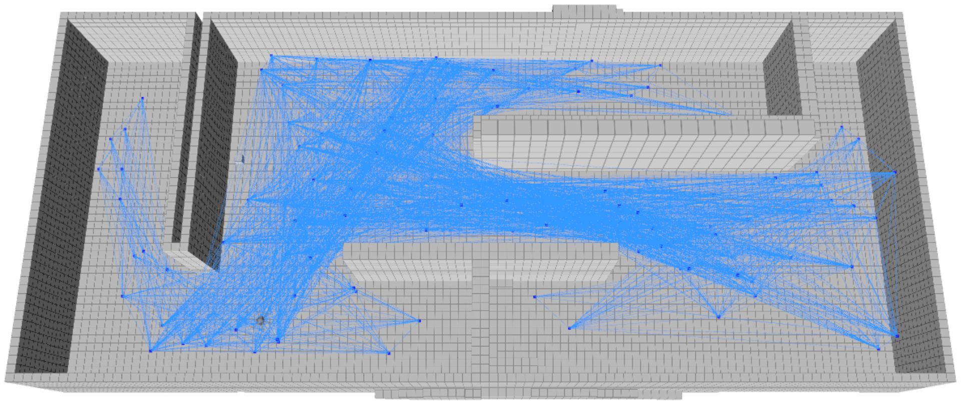
Coverage view sampling parameters related to Alg. 1 were set as $p^{\lambda}_{local} = 0.8$, $p^{\lambda}_{global} = 0.1$, and $N^{attempt}_{nbv} = 30$ for each scenario. The reachability update parameters for Alg. 2 for each scenario were commonly set to $N^{sample}_{traversal} = 3$, $p^{e}_{update} = 0.7$, and $d^{sample}_{traversal} = 2.0$m.

8.1 Apartment Scenario

The apartment scenario in Figure 8.1a is a relatively small scale interior space with the dimensions $20 \times 10 \times 3(\text{m}^3)$, used as a baseline for comparing the larger and more complex scenarios. An example map reconstruction by APN-P is shown in Figure



(a) Reconstructed map and exploration path.

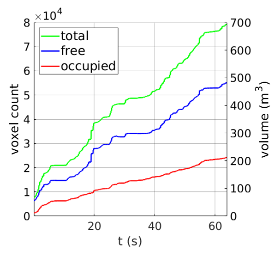


(b) APN reachability roadmap.

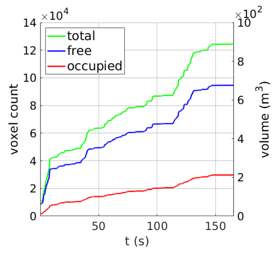
Figure 8.2: Exploration results for the Maze Scenario. (a): The explored path is plotted in red, with intermediate keyframe configurations represented by yellow points. (b): The APN nodes and edges overlayed in blue.

8.2a with the traced exploration path, and the APN roadmap is shown in Figure 8.2b. The average distance traveled was 76.5m, and a surface coverage completeness of $\vartheta_{\mathcal{M}} = 100\%$ was consistently achieved at each evaluated map resolution.

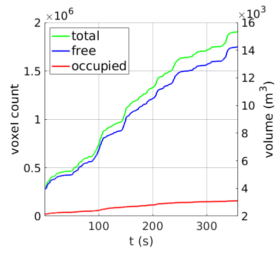
Figure 8.3a shows an example of the explored map volume over time using resolution 0.2m for reference. The surface coverage rate $\eta_{\mathcal{M}^{occ}}$ was 1.5m^3/s and 2.6m^3/s for the respective map resolutions of 0.1m and 0.2m. Since there are multiple dead-end regions for this scenario, some amount of backtracking is unavoidable, where the effects of backtracking correspond to the periods in Figure 8.3a where the map growth briefly stagnates (e.g. around the 30s timestamp).



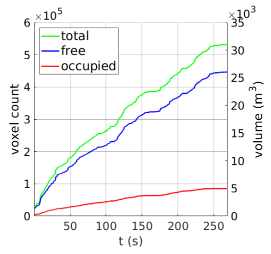
(a) Apt., $r_{\mathcal{M}} = 0.2$m



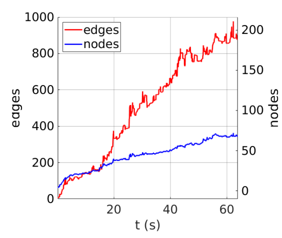
(b) Maze, $r_{\mathcal{M}} = 0.2$m.



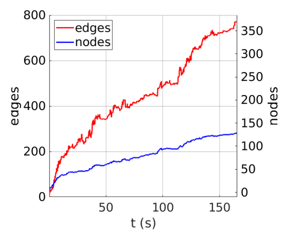
(c) Ind. Plant, $r_{\mathcal{M}} = 0.4$m.



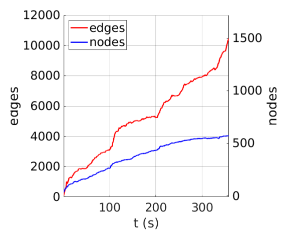
(d) Warehouse, $r_{\mathcal{M}} = 0.4$m.



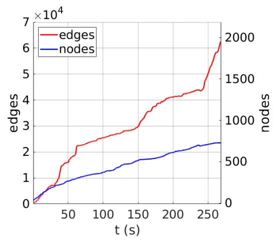
(e) Apt., $\vartheta_{\mathcal{V}} = 17.1$, $\vartheta_{\mathcal{E}} = 0.58$.



(f) Maze, $\vartheta_{\mathcal{V}} = 16.0$, $\vartheta_{\mathcal{E}} = 0.20$.



(g) Ind. Plant, $\vartheta_{\mathcal{V}} = 3.8$, $\vartheta_{\mathcal{E}} = 0.25$.



(h) Warehouse, $\vartheta_{\mathcal{V}} = 2.7$, $\vartheta_{\mathcal{E}} = 0.42$.

Figure 8.3: Representative results of the exploration progress over time. (a) - (d): explored map in terms of total voxels and their volume. (e) - (h): corresponding APN size in terms of its nodes (red) and edges (blue), with the respective node density ($\vartheta_{\mathcal{V}}$) and edge density ($\vartheta_{\mathcal{E}}$).

The size growth of the APN over time shown in Figure 8.3e. Compared to the map scale in Figure 8.3a, the APN is significantly smaller and its growth over time is non-monotonic due to iterative pruning and refinements. The final state of the APN roadmap is shown in Figure 8.2b, which can be seen to expand throughout the reachable free-space at a sufficient density for planning and navigation.

Figure 8.4a shows representative results of the computation times per cycle, using map resolution 0.2m as reference. The time taken for DFR remains fairly consistent over time despite the increasing map size. This demonstrates the effectiveness of the difference-aware update procedures at constraining the complexity as the map grows. A statistical boxplot of the respective procedures executed per cycle is shown in Figure 8.4e. The majority of computation time per cycle was spent on view planning, which had a median value of 13.6ms. The time spent on global cluster planning was negligible due to the relatively small size and complexity of this environment. The

APN contained an average of only 1.2 clusters, resulting in a trivial instance of cluster sequence optimization. The computation times for all differential regulation procedures were minimal compared to planning, given the relatively simple environment.

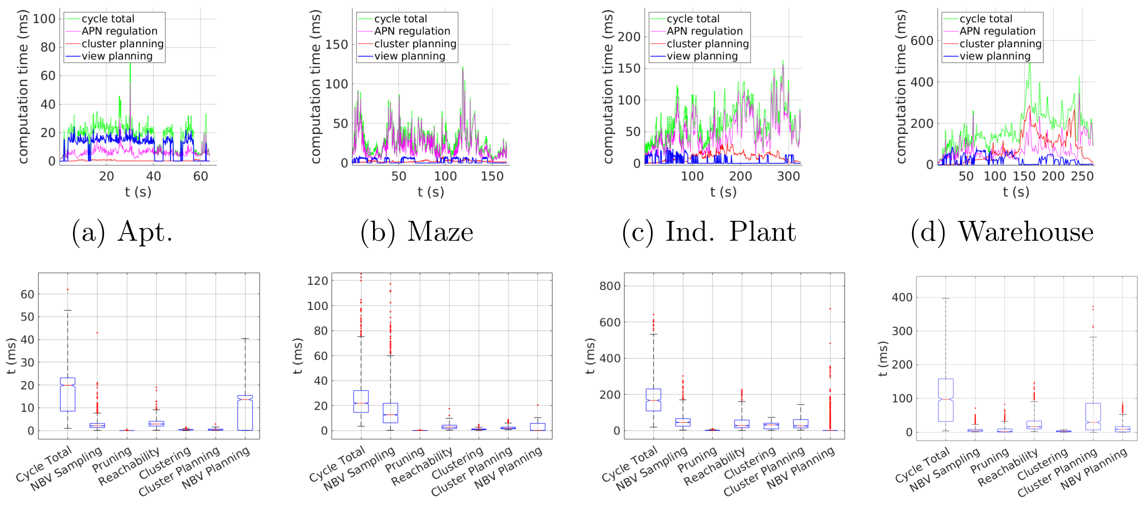
The time performance with the compared methods is summarized in Table 8.3. At the lowest map resolution of 0.4m, the APN-P achieved an average total exploration time of $\overline{T} = 52.9\text{s} \pm 4.3\text{s}$, and average computation time per iteration of $\overline{t^c} = 14.0\text{ms}$. Using a map resolution of 0.2m, the average exploration time was 57.9s with 18.9ms per cycle. At the highest map resolution of 0.1m, the average exploration time was 69.4s with average 28.9ms per cycle.

The RH-NBV approach required the highest total exploration time of 501.9s, with an average computation time per iteration of 153ms. For AEP, the total exploration time for each resolution was reported to take approximately 200s on average (exact quantities were not specified), with an average computation time per iteration of 98ms. FFI reported the fastest exploration time of the compared methods, with a total time of 80s and 151s for the respective map resolutions 0.4m and 0.1m. It should be noted that this approach was terminated once 95% exploration was reached, rather than full coverage.

The APN-P performance demonstrated a significant improvement over the compared state-of-the-art implementations in terms of both total exploration time and per-iteration computation times. Compared to FFI, APN-P achieved complete coverage while the exploration time was reduced by 34% using resolution 0.4m, and 54% using resolution 0.1m. Additionally, the percent improvement between resolutions indicates better scalability to higher resolution mapping.

8.2 Maze-like Scenario

A maze-like environment is presented in Figure 8.1b with the dimensions of $20 \times 20 \times 2.5(\text{m}^3)$. This scenario was tested using map resolutions of 0.1m and 0.2m; higher resolutions were not evaluated since there are narrow passageways that require lower



(a) Apt. (b) Maze (c) Ind. Plant (d) Warehouse

(e) Apt. (f) Maze (g) Ind. Plant (h) Warehouse

Figure 8.4: Timing performance for each exploration scenario. (a)-(d): depict the processing time taken per cycle. (e)-(h): display the median statistical boxplot of the DFR and planning computation times per cycle.

Table 8.3: Time performance comparison in terms of total exploration runtime $\overline{T}$ and computation time per cycle $\overline{t^c}$, averaged over 10 runs.

		APN-P		**FFI**		**AEP**		**NBVP**	**Rapid**
Scenario	$r_{\mathcal{M}}$[m]	$\overline{T}$[s]	$\overline{t^c}$[ms]	$\overline{T}$[s]	$\overline{t^c}$[ms]	$\overline{T}$[s]	$\overline{t^c}$[ms]	$\overline{T}$[s]	$\overline{T}$[s]
Apt.	0.4	52.9	14.0	80	122	200	92	501.9	-
	0.2	57.9	18.9	-	156	200	-	-	-
	0.1	69.4	28.9	151	68	200	129	-	-
Maze	0.2	145.1	26.1	177	155	-	-	-	-
	0.1	212.6	48.0	330	238	-	-	-	-
Ind. Plant	0.2	353.1	186.8	> 1000	152	941	–	2104	582
Warehouse	0.4	268.1	121.3	-	-	-	-	-	-

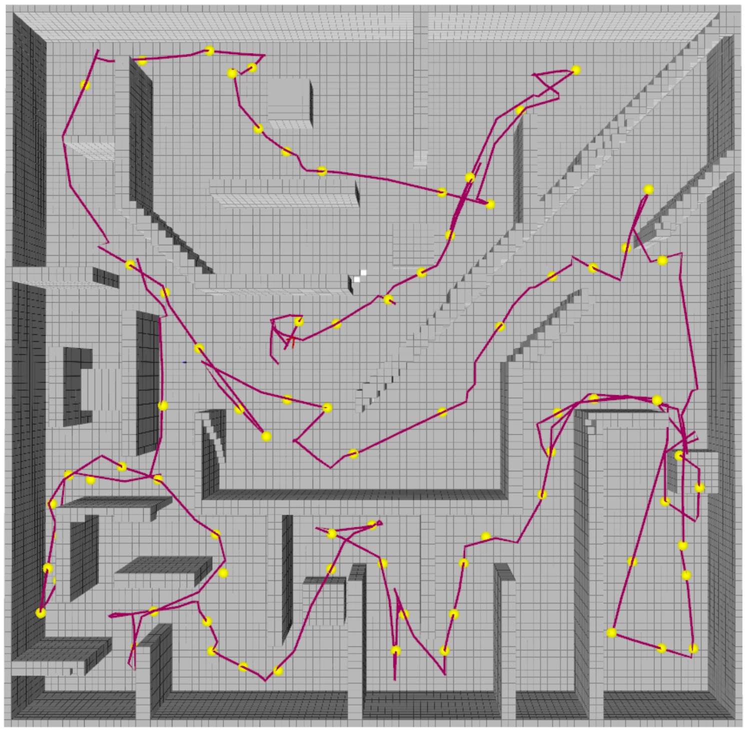
resolutions to admit collision-free paths (as also noted in [80]). This scenario was primarily compared against FFI, as this scenario was not evaluated in the original works of the other approaches.

A representative example of the mapped environment after exploration is shown in Figure 8.5a with the executed exploration path overlayed in red. The path shows that very few redundant motions were executed and progresses smoothly throughout the maze passages, with an average total path length of 208.9m.

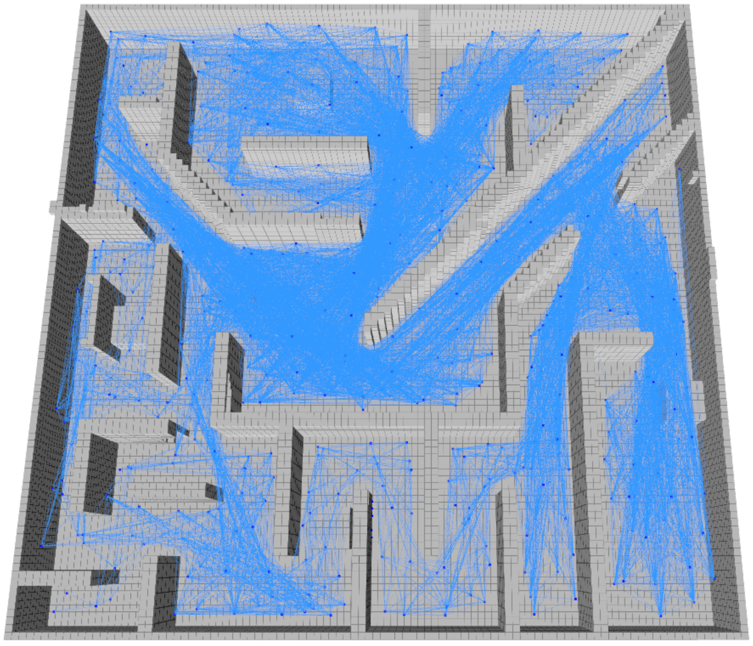
Figure 8.3b shows the map construction over time. An average coverage value of $\vartheta_{\mathcal{M}} = 100\%$ was reached at each map resolution, and the surface coverage rate $\eta_{\mathcal{M}^{occ}}$ was 0.5m^3/s and 1.4m^3/s for the respective map resolutions of 0.1m and 0.2m. The APN size growth over time was plotted in Figure 8.3f, and visualized in Figure 8.5b. The average node density per 100m^3 was $\vartheta_{\mathcal{V}} = 16.0 \pm 1.3$, with an average edge density of $\vartheta_{\mathcal{E}} = 0.20 \pm 0.12$.

The computation times per cycle are plotted in Figure 8.4b with a statistical analysis of the computation time taken per procedure shown in Figure 8.4f. For this scenario, most of the computation time went towards APN regulation, with coverage view sampling requiring the most time of 15.8ms due to the prevalence of obstacles and occlusions. Despite the high obstacle density, the computation times for reachability updates remained relatively small, while still maintaining sufficient node and edge densities to facilitate planning. This demonstrates the effectiveness of the local difference-awareness and efficient data caching strategies that minimize wasteful or redundant processing.

Table 8.3 summarizes the exploration efficiency of the compared approaches with respect to total exploration time and computation time per cycle. Note that as previously mentioned, exploration time for FFI was reported when 95% coverage was achieved, rather than 100%. The APN-P completed the exploration with 100% coverage in an average time of 145.1s and 212.6s for map resolutions 0.2m and 0.1m,



(a) Reconstructed map and exploration path.



(b) APN reachability roadmap.

Figure 8.5: Exploration results for the Maze Scenario. (a): The explored path is plotted in red, with intermediate keyframe configurations represented by yellow points. (b): The APN nodes and edges overlayed in blue.

respectively. These are significant improvements over the results of FFI, while the processing time per cycle was also reduced by around 80% and had much less variability. Additionally, the total exploration time for FFI increased by 86% between the two map resolutions, while the respective increase for the APN-P was 45%. This further demonstrates the performance scalability for higher mapping resolutions using larger and more complex environments.

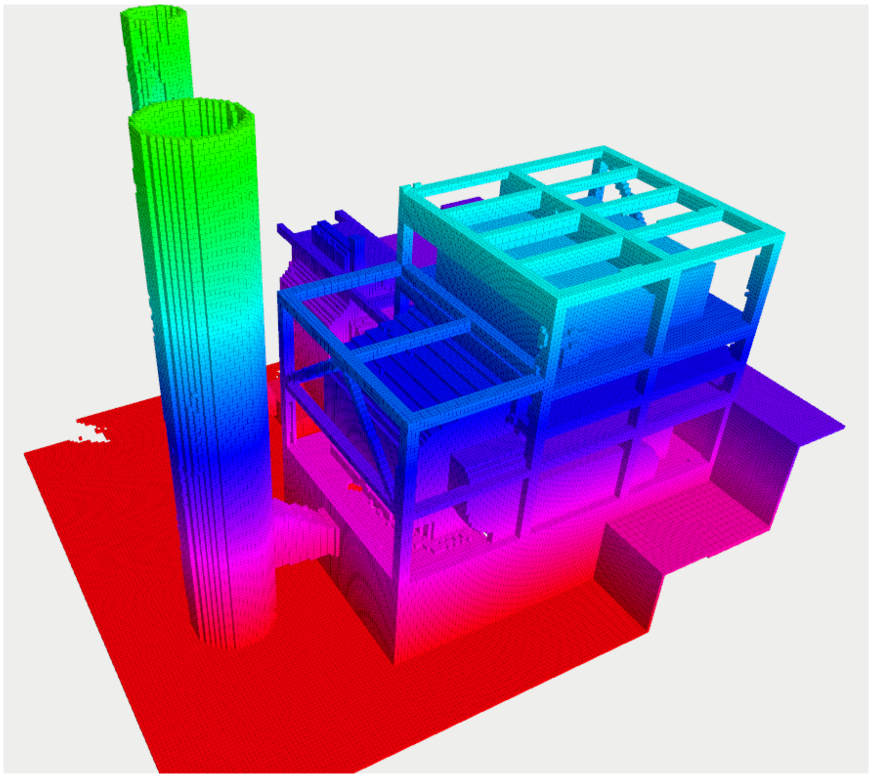
8.3 Industrial Plant Scenario

The Industrial Plant scenario shown in Figure 8.1c is an outdoor environment based on the Gazebo Powerplant model, truncated to the approximate dimensions of $33 \times 31 \times 26(\mathrm{m}^3)$. It represents both a large-scale and complex exploration task due to intricate structural geometries with many auto-occlusions. It was tested using a map resolution of 0.2m and maximum velocity of 2.5m/s, consistent with the compared approaches.

An example of the explored map is shown in Figure 8.6a, with the explored volume over time plotted in Figure 8.3c. A high surface coverage rate of $\eta_{\mathcal{M}^{occ}} = 3.2\mathrm{m}^3/\mathrm{s}$ was achieved, which was consistently maintained as shown in Figure 8.3c. The average total coverage was 98.7%, due to a few small regions with high surrounding occlusions, where coverage sampling failed to find a feasible viewpose. This could be overcome by selecting more aggressive sampling parameters, which was not done for these tests for parameter consistency between scenarios.

The APN size over time is plotted in Figure 8.3g, with the final roadmap structure visualized in Figure 8.6b. The average node and edge density were $\vartheta_{\mathcal{V}} = 3.8$ and $\vartheta_{\mathcal{E}} = 0.25$, respectively. By visual inspection of Figure 8.6b, the extent and density of the network appear to provide good coverage throughout the map.

The processing time per cycle is displayed in Figure 8.4c, with a statistical boxplot of the time taken by each subroutine shown in Figure 8.4g. Traversal edge maximization required the most computation time during differential regulation with an


(a) Reconstructed map (colorized by voxel height).

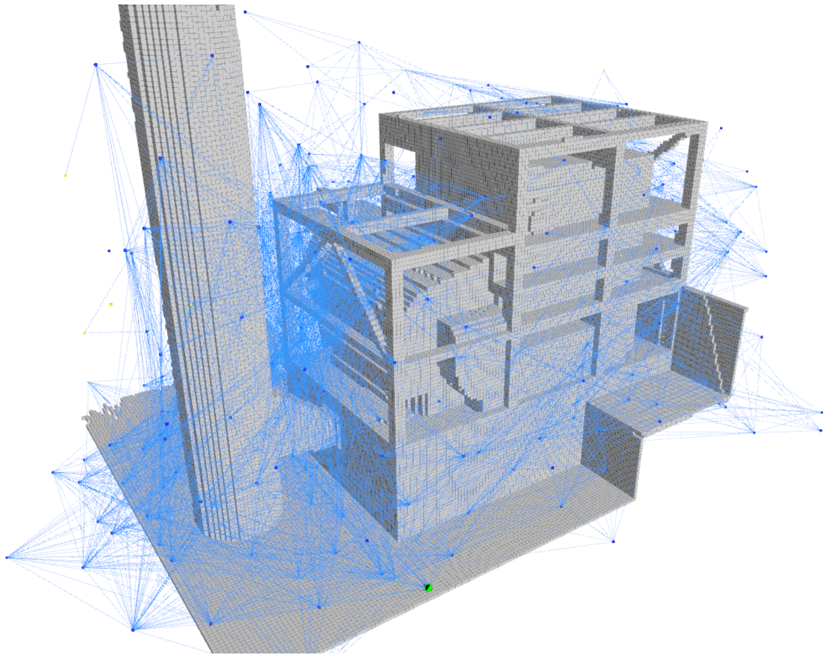

(b) APN reachability roadmap.

Figure 8.6: Exploration results of the Industrial Plant scenario.

average of 67.2ms due to the large scale and the amount of empty-space surrounding the structures which is initially unknown. Unknown edges are repeatedly checked for collision checks until they can be determined as either completely free, or having an occupied collision after which they are suppressed. The processing time spent on planning was well-balanced between the hierarchical layers.

A comparison of the timing results to the baseline approaches is shown in Table 8.3. We note that this environment was not originally tested by the authors of RH-NBVP; instead, the corresponding value of 2104s was obtained from the comparative analysis performed by [81].

Rapid performed with the fastest total exploration time among the compared methods, taking 582s with an average total distance of 728m. This was also the only approach able to finish exploration within the rated time limit T_{max} of 840s. This could be explained because this approach takes advantage of the large amount of free space to maintain high velocity, which helps to offset the diminished efficiency from greedy planning. However, this also has the effect of frequently leaving regions that have only been partially mapped. Coverage gaps can frequently occur that require large redundant paths to revisit, or otherwise reduce the completeness of the final map depending on the specific termination criteria.

Additionally, the authors of Rapid note that their implementation can spend a significant amount of time computing paths over large distances (up to 10 seconds) using Dijkstras algorithm over the map. These computation times were omitted from the reported total exploration time to focus evaluation only on the quality of their flight behavior. Even without this consideration, the APN-P was still able to reach complete exploration around 65% faster on average with a decrease in distance traveled of around 10%. This also highlights the importance of the APN efficiency to prevent such high computation times from occurring in practice.

APN-P exhibited significantly better performance than all compared methods, requiring an average total exploration time of only 353.1s, with each cycle requiring an average of 186.8ms. The average total distance traveled was 406.3m, with mean velocity of 1.9m/s. The MAV was able to maintain higher velocities due to the fast cycle times, which enabled the system to quickly react to the changing spatial map and re-plan its exploration path. Often the information gain of the current NBV goal gets fully observed as the MAV gets closer which can be quickly reflected within the network, allowing it to maintain its momentum by not needing to completely stop at each goal.

To evaluate how the larger size of this scenario correlates to the processing time per cycle, the Ind. Plant was additionally evaluated against the Maze scenario. To enable more consistent comparison, the map resolution was kept at 0.2m, and the maximum velocity and sensor parameters were assigned the values used for the Maze as indicated in Table 8.2. The resulting cycle processing time for the Ind. Plant decreased by around 58%, with each cycle taking an average of $\overline{t^c} = 78.7$ms. Within each cycle, DFR required 46.4ms and planning required 32.3ms.

The effects of map resolution were analyzed by a testing the timing performance using map resolution of 0.4m. This resulted in a significant decrease in the cycle processing time, which was reduced to $\overline{t^c} = 30.9$ms, and the total exploration time was reduced to $\overline{T} = 220.2$s. This indicates the increased cycle processing time at resolution 0.2 were primarily due to the increased resolution, rather than the larger environment size directly.

8.4 Warehouse Scenario

The Warehouse scenario is a large-scale indoor environment with the approximate dimensions $90 \times 30 \times 15$ (m^3), shown in Figure 8.1d with its exterior shown on the left, and the interior structures shown on the right. The models exterior structure was derived from the Powerplant model available from the Gazebo model library, while the interior was modified by adding a various geometric features and structures to create a more intricate environment for exploration. Since this was a custom built model, the APN-P was evaluated independently as comparative results were unavailable.

Due to the larger scale of this scenario, the mapping resolution was set to 0.4m, and the maximum velocity was increased to 3.0m/s. The sensing parameters were also increased using a maximum range of 9m, with FoV $(75°, 115°)$. The larger sensor view volume results in more information being added to the map per scan and the higher maximum velocity results in more scans being integrated between cycles, both resulting in more changed data to process per cycle. This scenario was also used to analyze variations of the clustering parameters ρ_c and D_c, which are indicated in Table 8.4. Unless otherwise noted, these parameters were set to $\rho_c = 4$ and $D_c = 7.0$, consistent with the previous Industrial Plant evaluation.

A representative example of the reconstructed map results shown in Figure 8.7 and the explored map volume over time is shown in Figure 8.3d. A minimum coverage ratio of $\vartheta_{\mathcal{M}} = 99.98\%$ was achieved for all test configurations. The APN size growth is depicted in Figure 8.3h, which contained an average of 346 nodes and 21494 edges, with an edge density factor of 0.374.

The computation time per cycle is plotted in Figure 8.4d and summarized in Table 8.3. Similar to the Ind. Plant scenario, the time spent on APN regulation remains within a bounded range despite the increasing size of the map and APN. The exploration time performance results are summarized in Table 8.3, requiring an average exploration time of $\overline{T} = 268.1$s and average planning cycle time $\overline{t^c} = 121.3$ms. A more detailed breakdown of the processing times per sub-procedure is shown in Figure 8.4h.

Different clustering parameter variations were applied and the resulting time performance is summarized in Table 8.4. The average exploration time was not significantly changed between parameter variations, indicating the low sensitivity of these parameters. The primary effect of the variations was on the per-cycle computation time, though the differences were relatively minor. Using the values $\rho_c = 4$ and $D_c = 10.0$,

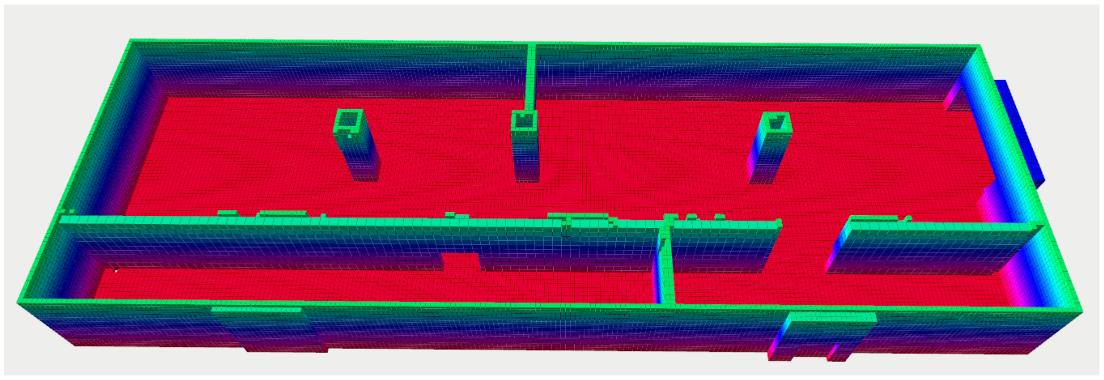


Figure 8.7: The reconstructed map of the Warehouse scenario colorized by voxel height. The maximum height of displayed voxels is truncated for visual clarity.

Table 8.4: Timing performance for the Warehouse Scenario according to variations of the clustering parameters ρ_c and D_c.

ρ_c	D_c[m]	$\overline{T}$[s]	$\overline{t^c}$[ms]	$\overline{t^c}_{DFR}$[ms]	$\overline{t^c}_{plan}$[ms]
4	7.0	268.1	121.3	47.4	78.9
4	10.0	270.0	109.7	55.3	54.4
7	10.0	274.9	128.7	49.1	79.7

the cycle time was nearly evenly distributed between differential regulation, $\overline{t^c}_{DFR}$, and planning, $\overline{t^c}_{plan}$. The other parameter combinations increased the planning time, but only by a small amount.

8.5 Discussions

The experimental results show that our approach has the ability to iteratively update the APN and replan the exploration path at an average rate of at least 20Hz for the two smaller scale scenarios (Apt. and Maze), and at least 5Hz for the larger scales (Industrial Plant and Warehouse). However, the difference between these cycle rates is not primarily due to the larger environment sizes. Instead, the larger sensor view volume and higher maximum velocities are the more significant factors, which result in a larger amount of map data for processing per cycle, but these factors are not directly related to the environment size. This helps to explain the scalability of our approach for larger environments.

For the smaller environments, most of the planning time is spent on local view planning (see Figure 8.4e and 8.4f), This is due to the relatively few clusters needed to partition the nodes, resulting in trivial cluster planning instances. However, planning directly over all NBVs can quickly become intractable as the map size increases, either resulting in unacceptably large processing times, or would otherwise require premature search termination that degrades the planning quality.

The hierarchical planning strategy of APN-P helps to mitigate the complexity by keeping the problem size manageable. Furthermore, planning convergence is further accelerated by initializing each planning cycle from the partially optimized solution of the previous cycle. This reduces the need to introduce further problem simplifications or approximations that would decrease the planning quality. These effects are demonstrated by the results shown in Figure 8.4d and 8.4h. The distributed planning time remains relatively low and does not exhibit continually increasing growth, despite the increasing size of the map and APN as shown in Figure 8.3h, 8.3d.

The frontier-guided information gain and sampling strategy of DFR provides an effective way to avoid the prohibitively high computation costs for analyzing information gain by the existing (compared) approaches and to balance processing time per cycle and update rates. This enables maximized coverage of the unknown map regions to be maintained at high update rates, providing the necessary knowledge needed for non-myopic planning.

CHAPTER 9: APEXMAP SOFTWARE FRAMEWORK

In this chapter, we introduce a software framework designated Active Perception for Exploration, Mapping, and Planning (APEXMAP). It was motivated by the inherent software challenges related to developing complex real-time dynamic systems inherent to online exploration and other active perception problems. Such systems are described as being *complex* as they can be comprised of many subsystems that must interact and communicate with each other. They are additionally described as *dynamic* as they involve time-varying data models built and managed online from sensor inputs, and execute in real-time and at different temporal rates between different subsystems.

Developing software for any experimental robotics application generally involves many non-trivial challenges that can be very time-consuming and may require a high level of programming expertise which not all researchers may possess. For active perception applications, the combination of the underlying problem complexity and real-time operation makes software performance a paramount factor; however, optimizing the performance of software further exacerbates its challenges and increases development time. Consequently, there is a significant entry-barrier for active perception researchers, and the rate of related research innovations and advancement over time can be hindered if the software-related challenges are not effectively addressed.

The essence of online exploration and other active perception problems presents unique challenges towards software engineering and reusability. These challenges primarily derive from several aspects inherent to the problem domain, outlined as follows.

- **Global objectives solved incrementally**: active perception problems generally involve long-term global objectives, which are solved incrementally out of necessity given that the information needed is initially unknown and must be acquired online. Due to this, the behavior of particular solution cannot be meaningfully analyzed from a single iteration, but requires the cumulative long-term analysis over many iterations.

- **Closed-loop feedback operation**: a defining characteristic is the closed-loop process formed by the feedback between sensing, planning, and actions. Since the inputs to a planner depend on its previous output actions, an explicit process model is needed that generates the corresponding inputs from the actions. The dependency between the inputs and outputs precludes the use of offline methods or fixed datasets for analysis purposes. Instead, each robot action must be either executed by a real-world system, or through a realistic simulation. In either case, this necessarily introduces operational dependencies with numerous subsystems for trajectory planning, collision detection, motion tracking and control, and others. Interactions may be needed between these supporting subsystems and other approach-specific subsystems (e.g. trajectory planning and collision detection depend on the state-varying knowledge of the spatial map), which increases software coupling to a particular approach.

- **Environment simulation and interaction**: real-world operation can be impractical or costly, especially for initial development and analysis. Physics-based environment simulation is typically preferred to provide ground truth knowledge for more accurate and thorough analysis. Simulators also eliminate risks of damaged equipment and provide complete control over the environments characteristics, where suitable real-world environments may be difficult to find or create. However, this increases software needed to create the simulation models and interface between the simulation and the application system.

- **Real-time/online operation**: since the robot may be in motion during planning and decision making processed, the online computational performance can have a direct effect on the long-term behavior. This makes performance optimization an important factor early on, limiting the practicality of first analyzing the behavioral aspects then focusing on performance optimization later on, which is a common development approach that helps simplify the initial efforts.

- **Concurrent processes with different rates**: it is generally desired that sensing and map building operate at high-frequency concurrent to planning, rather than sequentially, such that critical sensory information is not "missed" while other processes or executing.

- **Experimental research**: the nature of experimental research naturally involves continuous development and innovation over time, which applies both to the research methodologies and models, and the software requirements specifications to realize the approach by a computational system or to optimize its functional performance. Software should be considered in the long-term context of the ongoing research problems, rather than as singular independent solutions to the immediate problem at hand.

In research contexts, it is common practice to utilize open-source software to accelerate development and mitigate the challenges related to generic software engineering tasks and also to the requirements of a particular problem domain. Open-source software represents a form of mass open collaboration, where software is made publicly available for independent reuse by any number of end-users. The practical reusability of software by end-users can depend on several factors, such as the variability within the problem domains it was design to address. Reusability is also impacted by the modularity and completeness of the software design in addressing the underlying problems. It can also depend on the specific use-case and requirements of the end-user, which can be unpredictable.

An extensive and thorough investigation of available open-source software useful for online exploration and active perception was conducted, finding that the challenges of this problem domain have not been sufficiently addressed relative to other problem domains. Existing open-source software can be reasonably applied to solve some of the subproblems individually, such as generic mathematical operations, motion planning, or spatial map modeling; however, the system-level integration and communication between many components can remain a significant challenge. Furthermore, given the experimental nature of the research field, the approach model and its software requirements specifications are constantly subject to changes over time. If the system-level integration of the various software components is tightly coupled, then even seemingly minor changes to the requirements can cause major software changes to be needed. However, it is difficult to anticipate future changes and their potential effects in advance.

An operational environment must also exist to provide the source of the sensor measurements. From a practical perspective, it is more effective to represent the robot system and environment using high-fidelity simulation which can provide ground truth knowledge that facilitates more meaningful and precise performance analysis, and provides complete control over the test environment and its characteristics. However, this further increases the required software to include the vehicle dynamics model and mathematical models to replicate each the functional hardware components such as sensors and actuators.

In practice, time is a finite and scarce resource that naturally motivates researchers to prioritize reduced development time at the sacrifice of its maintainability, interoperability, and reusability [102]. Given the ongoing nature of research and innovation, these sacrifices tend to increase the long-term cumulative development time and effort to handle future design and requirements changes. The focus is placed primarily on the immediate approach-specific requirements, which causes strong coupling between the approach and its software design. Efforts towards reusability, including documentation, are typically only considered as the research reaches maturity, if ever. In fact, it was discovered by recent studies that less than half of the software from recent AI conferences were able to repreoduce their results, or even to be run at all, even with assistance from the original authors [103, 104, 105]

Non-reusable software precludes open collaboration between independent researchers, such that most or all of the software engineering challenges must be independently resolved by each research group. Thus, each researcher encounters a large-scale development burden and are similarly unlikely to produce reusable software, creating an entry-barrier cycle that can repeat indefinitely.

The APEXMAP framework was designed to facilitate ongoing active perception research using a modular architecture that facilitates reusability between a independent researchers and over wide range of requirements specifications, while also promoting optimized runtime performance. The underlying goal of the framework is to enable end-users to rapidly develop and test software applications without sacrificing their computational efficiency. Additionally, the importance of modularity and extensibility are reflected in the theoretical models of the APN, DFR, and planning methodologies. In this way, these theoretical models can be reformulated to different problem objectives and technical approaches, where these reformulated models can then be efficiently realized using APEXMAP in a complementary fashion.

9.1 Software Engineering

Software engineering (SE) is a fundamental component of any experimental robotics research, which is the human task to design effective and reliable computer software by programming its *source-code.* Robotics programming can be considered a distinct subfield of software engineering, due to a variety of unique requirements and challenging subproblems it necessarily subsumes. As outlined in Chapter 2, there are numerous components and subproblems that comprise a complete mobile robotic system including hardware interfaces, robot and sensor modeling, and task-specific algorithms and mathematical models. It is a non-trivial task both to develop the software for each subsystem, and their integration into a cohesive application system.

Active perception problems further impose a particularly challenging development burden on robotics programming due dependencies between various subsystem from the feedback between sensing, task planning, motion planning, and control, and the influence this has on the behavior of an autonomous system. Consequentially, planning algorithms generally cannot be run or evaluated in isolation, but instead require all components of the full system to be operated together. This is not often encountered for many other robotics problems, which allows them to be studied with much less software effort.

For example, when the inputs to an algorithm are independent from its outputs, fixed datasets or other means of artificially generating the inputs can be utilized for initial development, analysis, or benchmarking. This is often done for motion planning, mapping, or SLAM related research since these are considered passive tasks. However, such approaches are not practical or even feasible for active perception due to the feedback dependencies between its inputs and outputs.

The programming challenges can not only impede research progress, but they can also significantly degrade the runtime performance and task effectiveness of the system. There are typically tradeoffs that arise between software development time and the runtime performance and reusability of a program. Designing a program that optimizes runtime performance consumes more development time, and doing so in such a way that the code can be efficiently reused or extended is even more difficult and requires a higher level of expertise.

These aspects are more often sacrificed in favor of reducing the already high development time. This results in an effect referred to as *technical debt* [106], where suboptimal software design choices that favor expedience tend make future changes increasingly difficult or even impossible. As a result, programs with low modularity or reusability make it difficult and time-consuming to make continued improvements and extensions to an existing approach. Additionally, low-reusability prevents different researchers from sharing the development efforts, where each must resort to developing their own software from scratch and repeating the cycle.

A set of design guidelines has been established as a guide for effective software development [107], describing practices and patterns that facilitate reusability, maintainability, and help simplify the design of complex software systems. These are known by the acronym SOLID, which describes the principles by the following terms:

- **Single-responsibility principle**: classes should be designed to serve singular purposes, and should be subject to changes from only a single source. This encouraged low coupling and high cohesion.

- **Open/closed principle**: classes should be open to extensions which add capabilities, but closed to modifications that alter its original structure or behavior. This encourages the use of abstraction through polymorphism or inheritance techniques.

- **Liskov substitution principle**: also known as behavioral subtyping, this principle states that an object should be transparently substitutable with its derived subtypes.

- **Interface segregation principle**: large class interfaces should be refactored into smaller ones, and should avoid unnecessary dependencies coupling to avoid side effects where a change to one class necessitates changes to potentially many other classes.

- **Dependency inversion principle**: high-level interfaces should not have dependencies on the details of low-level interfaces. Instead, both should depend on abstractions to avoid tight coupling between multiple interfaces.

Asymptotic time complexity, expressed by the well known big O notation, is useful and appropriate in many contexts regarding algorithm analysis in terms of its order of growth, independent from its implement ion. However, since implementation is the focus of software engineering, additional factors can be considered which may be hidden from the asymptotic analysis, such as memory management and runtime execution efficiency. Different implementations of an algorithm can achieve the same semantic behavior, but factors like memory management can greatly influence its efficiency on real hardware. This insights must be considered for program optimization purposes, which help ensure consistency between the theoretical performance and runtime performance.

9.2 Related Work

Open-source software has been developed to address a wide variety of software engineering challenges commonly encountered in robotics research. As mentioned in Section 9.1, active perception is characterized be distinct challenges often not encountered in other robotics subdomains. This section will briefly introduce relevant software designed to address overlapping subproblems, and highlight their limitations to effectively address the domain-specific challenges for active perception.

9.2.1 Generic Software for Robotics

The Robot Operating System (ROS) [108] is a robotics-oriented middleware framework that is perhaps the most widely used and impactful software for robotics. It uses a centralized process to register and manage an arbitrary number of independently running distributed processes, termed *nodes*, and establishes peer-to-peer communications channels that allow any of the registered nodes to communicate via serialized *messages*. A Parameter Server provides a shared database of information that can be directly accessed by nodes. ROS also consists of a variety of tools for a user that simplify configuration, setup, and inspection of running processes.

The design goals of ROS largely focus on handling the variability between the vast number different robot platforms, sensors, and other related hardware components in a generic fashion. A large focus is also placed on standardizing the interactions between processes. ROS provides a means to standardize the overall structure of a robotic system, the build and development environments, and the related setup and configuration tasks, while leaving the programs executed within a ROS system completely open-ended to the user. This is an intentional design to promote a "thin" architecture, where programs can be developed independent from ROS to facilitate reusability. ROS also includes a package management ecosystem, where a primary role of packages is to provide reusable libraries decoupled from the core ROS framework. A broad variety of different library packages are distributed alongside ROS, with many others maintained by third party sources, and users can freely create their own ROS packages as well.

The Mobile Robot Programming Toolkit (MRPT) [109] is a programming library consisting of a collection of programming tools, algorithms, and design patterns that facilitate development of complex systems that can involve many different subproblems. It was motivated by the recognition that many effective libraries exist for specific tasks, such as image processing, linear algebra, or occupancy mapping. However, third party libraries generally posses their own unique data types, syntax, and semantics that are incompatible with others. MRPT intends to serve as the "glue" that allows different domain-specific libraries to be efficiently interconnected, while also providing a variety of additional features on its own. MRPT is actively maintained in present day, and is effectively utilized by many researchers.

9.2.2 Problem Domain-specific Software

The motivation of domain-specific software is to leverage expert knowledge in a particular problem domain to assist in making software that is more reusable, efficient, and understandable [102]. A domain expert can effectively identify and separate the common factors within a domain from the variable factors that can differ between applications [110]. Domain-specific knowledge enables beneficial restrictions on the anticipated requirements and capabilities, helping to simplify their interactions and reduce performance losses that can arise from excessively high abstraction. However, it can be a highly challenging task to design an architectural structure and style that effectively generalizes over the intended domain in a reusable way. For this reason, such software tends to focus on smaller problems that have well-defined models and formulations. Open-source software has been contributed related to various robotics subdomains, which are outlined within this section.

Spatial maps are a common component shared between many different robotics applications, which has motivated many open-source software contributions. For exploration and active perception, it is generally preferable that map models provide discrimination between free space and unknown space. Two conceptual map models are utilized by perhaps the majority of existing works, which consist of voxel-based occupancy grids and signed distance fields (SDF). Many software implementations have been developed to realize these models, with the more popular approaches being OctoMap for occupancy grids [25], and Voxblox for SDF maps [93].

An adjacent task to spatial mapping is that of collision checking, which typically operates in conjunction with a map model. FCL [111] is a popular library that has been used in many works, and is also natively used by various ROS packages. It uses a bounding volume hierarchy with flexible configuration options over the type of collision queries using shape primitives. It is also directly usable with OctoMap to simplify integration tasks. Collision checking is also possible to be performed efficiently through SDF maps, which provide an efficient representation for obstacle distance queries. However, they generally have the tradeoff of being less efficient to incrementally update.

The Open Motion Planning Library (OMPL) [99] is a widely-used library directed at motion planning problems, primarily sampling-based motion planning. It provides a standardized and flexible approach for defining and configuring different forms of motion planning problem, which can be solved using a variety of pre-implemented state-of-the-art planning algorithms.

A significant variety of software has been contributed for problems related to SLAM, such as RTAB-Map [112], ORB-SLAM [113, 114, 115], and LOAM [116]. Most libraries provide only a distinct implementation approach with little intent for modularity of the source code, but often can be flexibly reconfigured for different robots, sensors, or other parameters to facilitate reuse.

9.2.3 Active Perception Software

A relatively small number open-source software have been released for active perceptions tasks like online exploration. A modular framework for volumetric information gain-based exploration using a MAV was presented in [117] by the name *mav_active_3d_planning*. The primary components of its structure are a *Trajectory Generator* module used to expand a tree of trajectories, and a *Trajectory Evaluator* module used to evaluate each branch and select the best trajectory segment in a receding horizon fashion. Here, modularity is achieved by allowing a user to define how trajectories are generated, evaluated, and selected. However, this structure imposes several restrictions on the frameworks usage, including the use of tree-based methods and MAV platforms, and also is tightly coupled with other libraries, such as Voxblox [93] used for map building. An extension was presented in [118] given the name GLocal, which introduces a global search mechanism among other features like state estimation drift correction and sub-mapping.

The approaches of GBPlanner [119] and GBPlanner2 [120] have also been released as open-source software. These frameworks provide a higher level of modularity than the aforementioned, but are still restricted to the general structure of the respective approaches. Other authors have released the source code of their approaches [77, 78, 79, 80, 19, 121], which are intended to allow use of the programs "as-is" without explicit support for modularity or significant functionality changes, limiting their reusability in other contexts.

9.2.4 Limitations and Drawbacks

ROS provides an effective framework for generically handling peer-to-peer communications between separate processes. While this approach provides excellent modularity, it can also introduce significant performance overhead due to the use of message serialization. ROS, by design, leaves most of the task and application-specific programming open-ended for generality. Domain-specific libraries can alleviate the workload for task-specific programming, but are typically designed independent of any framework or other libraries, which can lead to incompatible interfaces or syntax, particularly when many separate libraries are used within a single program. In some cases, integration challenges can even make integration of existing software for a particular task more difficult than implementing it from scratch.

Minimal domain-specific software for active perception has be contributed, none of which has sufficient modularity or extensibility for use significantly beyond their original problem models or approach specifications. While ROS and certain domain-specific libraries can be helpful for certain software engineering tasks, the challenges of large-scale integration, performance optimization, and reusability beyond a restrictive problem model still present a significant impediment to research in this domain as a whole.

9.3 APEXMAP Framework Overview

APEXMAP is a software framework primarily written in C++ which is comprised of a distributed collection of loosely coupled software libraries designed with a focus on the SOLID design principles. Using the separation of concerns principle, each libraries addresses a distinct subproblem domain or group of related subproblems pertaining to the common functionalities needed for active perception system. Each library provides some combination of generic algorithms, data structures, and class models that modularly support tasks related to their subdomain.

Given the lack of standard form and the high variability that can occur between applications and their approach details, library components were designed to provide abstract front-end interfaces to invoke algorithms and behaviors, while hiding how these are achieved. End-users can redefine the hidden interface without causing side-effects in other parts of the source code that use the interface, or derive new interfaces that extend their capabilities for specialized tasks. An end-user can then use these basic components to efficiently design more complex functional subsystems based on their particular requirements specifications, and assemble these into a complete system application.

The basic design goals of APEXMAP consist of the following:

1. rapid development
2. modular and reusable components
3. low coupling, high cohesion
4. efficient component communication
5. dynamic knowledge management
6. benchmarking and performance analysis

The framework structure is composed of several abstract modules that serve as building blocks for the design of any data structures, algorithms, and control flow of a program. Any data structures, objects, or interfaces that may need to be accessed or interact with each other are modeled by the abstract concept *Component Entity* detailed in Section 9.4. Every *Component Entity* instance is uniquely identifiable by an immutable unique resource identifier (URID), or by a reconfigurable name identifier (NameID), and are managed and accessed using the *Component Resource Server* (CRS). They are used to abstractly model arbitrary object-based software components or invokable procedures, which serve as primitives for building larger and more complex systems and manage its runtime control flow behavior.

To minimize coupling between modules, intermediate *concept models* can be designed as resource subtypes. A concept model serves as a behavioral prototype that is to be implemented by an underlying resource. This allows a standardized interface to be designed that defines a set of abstract capabilities of a model, while still hiding the implementation details for better modularity. For example, an occupancy map concept model can be designed which specifies a standardized set of well-defined methods or operations that an implementation is expected to provide, such as returning the occupancy state given an input query point. The concept model thus provides an additional abstraction layer that maintains interchangeability with different implementations. APEXMAP provides a variety of pre-designed concept models for many subtasks like map and sensor modeling, visibility checking, and collision checking, and additional models can be freely designed by a user.

The basic capabilities provided by the component entity patterns meet the requirements defined by the structured program theorem, in which they are sufficient to express any computable algorithm [122]. While this provides a strong theoretical basis for generalizability, in practice certain tasks may be over-complicated or lose efficiency when restricted to conform to only these idioms. This is mitigated through subtype polymorphism, which allows resources to be dynamically cast to their underlying interface models. This allows the custom interfaces normally hidden by the abstraction layers to be directly accessed if needed to directly perform specialized tasks in a more simplified or efficient fashion. However, this also causes a higher level of coupling within the program wherever a custom interface is utilized, which can decrease its modularity and interchangeability for future development. These design decisions and their tradeoffs can be independently determined by end-users.

APEXMAP implements an event-based message communication architecture to further simplify complex interactions between resources and algorithms during execution, detailed in Section 9.5. An event is simply an abstract data type used to communicate information between system components in an indirect manner. These are similar to the concept of messages used by ROS, but are transported using dynamic memory rather than serialization, enabling zero-copy communication and preventing the need for data type conversions. Events can be transmitted via publish-subscribe or client-server models that allow components to communicate indirectly throughout a system. This facilitates reactive programming techniques that can automatically handle dynamic state changes, send status information, or control distributed process executions.

A key capability provided by APEXMAP is directed at dynamic data management, detailed further in Section 9.6. This library augments most of the standard data container types provided by STL to support event-based communication, equipping them with the ability to broadcast and receive state-change events. The response behavior upon receiving a state-change event can be independently defined for each container instance, allowing them to automatically react to state changes according to a user-defined strategy or purpose. A variety of additional container models with specialized functionalities are also provided, such as bidirectional maps and Cartesian product maps, that are also equipped with state-change event communication. This library can greatly reduce the software design efforts for efficiently managing complex systems of dynamic data.

Finally, APEXMAP includes a variety of software library packages which contain many data structures, algorithms, and other programming tools that facilitate rapid development. Similar to ROS, these packages are designed with minimal interdependence to promote a thin coupling for better reusability. These provide pre-implemented and validated software components for many common aspects of active perception, including dynamic data containers and graphs, perception modeling, mathematics, geometric representation, motion planning, visualization, data logging and many others. The support library packages are intended to provide a supportive role during software development, but their use is optional to minimize restrictions a programs syntax and semantics. This is also to better support the integration of existing software into the APEXMAP framework that may without requiring significant changes to the existing source code structure.

9.4 Component Entity Kernel

To simplify the design of large and complex systems, APEXMAP provides a polymorphic object type system that allows systems to be decomposed into smaller reusable components. A component is a generic concept that can be used to represent any functional data type, such as containers for storing state variables, subroutines for invoking algorithms, or complex class interfaces that encapsulate a variety of data and functions related to a subsystem.

Components are represented abstractly by a common superclass, *Component Entity*, detailed in Section 9.4.2. This class provides the base abstraction layer that hides the component-specific details, and serves as a progenitor for all other component subclasses. Component subclasses of any type can be freely defined by extending from the *Component Entity* superclass, adding specialized capabilities as needed within the subclass definition in accordance with the open/closed principle. Object instances of *Component Entity* are created using dynamic polymorphism, where each object is dynamically allocated as a particular component subclass that specifies the underlying type of the object. The collection of component objects that comprise the system are managed by the *Component Resource Server* (CRS) detailed in Section 9.4.1.

The fundamental tasks and subproblems of the problem domain were decomposed into a collection of predefined components referred to as *Concept Models*, detailed further in Section 9.4.3. These allow component subclasses to be defined by deriving from other subclasses, and allows them to be organized into multilayer hierarchies that represent a component interface at variable levels of abstraction. This design pattern facilitates the SOLID principles, encouraging increased modularity and substitutability of components, and minimizing the amount of change dependencies that can propagate throughout the different parts of the system as a result of replacing or modifying its components. One intended use of Concept Models is to develop generic design pattern for abstract modeling of common subsystems like sensor models, spatial maps, collision checkers, and visibility checkers.

A generic Concept Model component, *Subroutine Components*, was designed in this way to provide generic capabilities that simplify the development of modular high-level subroutines and behaviors. These are described further in Section 9.4.4.

9.4.1 Component Resource Server (CRS)

The *Component Resource Server* (CRS) serves as a centralized locale for managing the system components, with its class interface diagram shown in Figure 9.1. Component instances can be dynamically created and added to the CRS, which uses their abstract base type to allow them to be stored as a homogeneous collection. It provides a single location for creating and storing components of any type, which simplifies the abstract composition and assembly of components into a unified system. It also provides a single entry point that allows components to access and interact with other components.

An additional functionality of the CRS is to simplify the management of the overall control flow of a program. The execution of most programs can be effectively factorized into several abstract stages of execution, as shown in Figure 9.1, which mainly consist of init, setup, start, and stop. Calling these methods on the CRS causes a corresponding behaviors to be invoked over all the components. Each component can have its own distinct behavior that needs to be executed, which helps simplify the program design and make it easier to manage since these aspect of the code can be self-contained by each object itself, rather than scattered throughout the execution control flow.

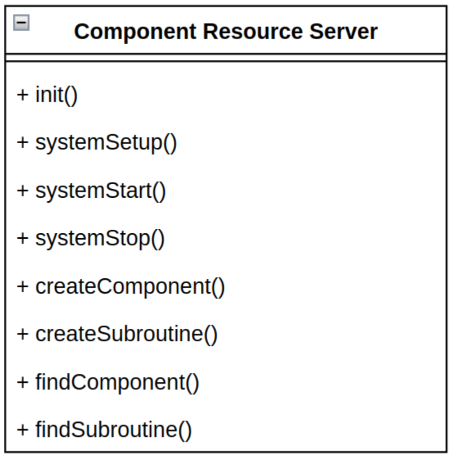


Figure 9.1: UML partial class diagram of the *Component Resource Server* interface.

9.4.2 Component Entities

A *Component Entity*, illustrated in Figure 9.2, represents the highest level of abstraction of a component. This class serves as an opaque data type that hides all aspects of the components subtype, providing a universal basic type for all components.

One advantage of using a universal basic type is to allow the heterogeneous components to be stored as an abstract homogeneous collection by the CRS, which would otherwise become dependent on the explicit types of all managed components. Each *Component Entity* instance is assigned a unique name identifier, which allows them to be efficiently organized and accessed from the CRS. They also provides basic functional operations that simplify their configuration and usage, such as querying the underlying subtypes and performing polymorphic conversions between them.

C++ provides mechanisms for dynamic dispatch using virtual functions, where a

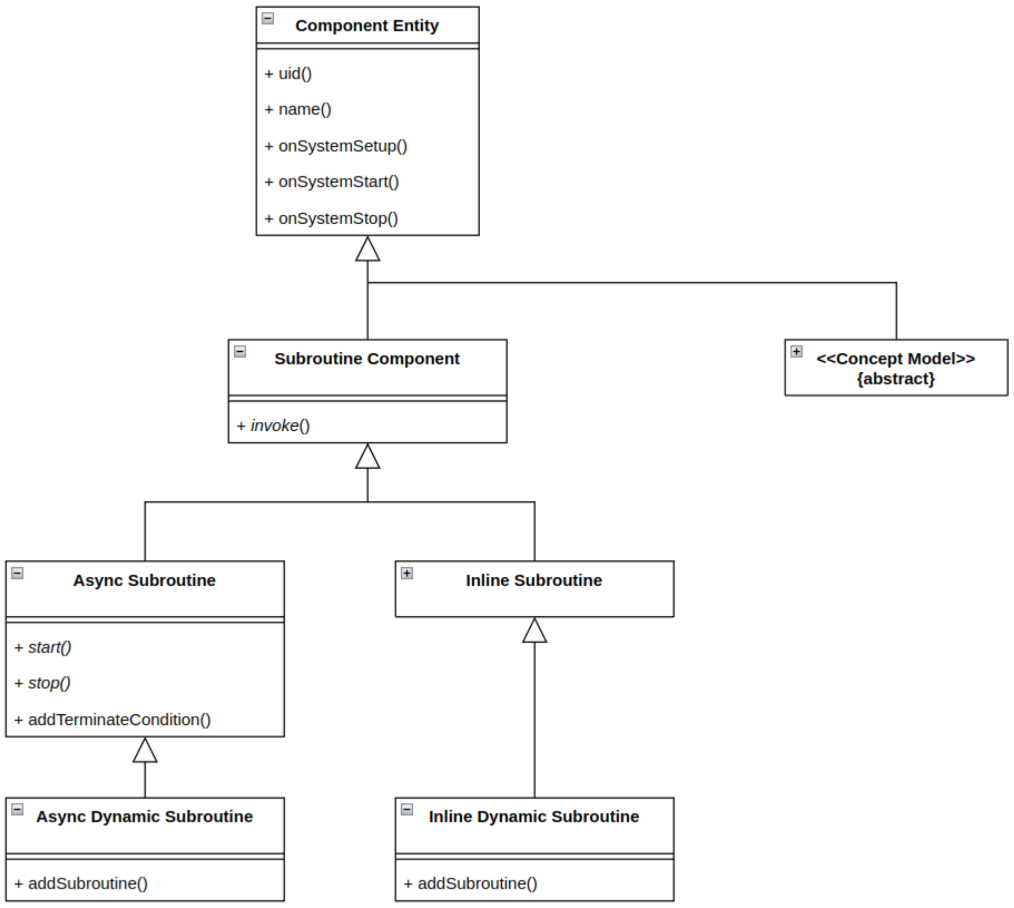


Figure 9.2: UML partial class diagram of the *Component Entities*.

function can be declared in a parent class while allowing the behavioral definition of the function to be overridden by a child class. This allows the behavior of the child class to be transparently invoked without knowing its type information. Dynamic dispatch can be used to allow a parent component to declare an abstract interface, and any child component that satisfies the requirements can be used interchangeably to realize the interface.

9.4.3 Concept Model Components

Components that address domain-specific tasks and subproblems can be abstractly represented by a *Concept Model*. Their intent is to provide a reusable component model that captures the invariant aspects of the concept in a generic fashion, which can then be used to derive more specialized submodels and their orthogonal capabilities. These can be hierarchically organized into multiple abstraction layers, where each layer defines the common interface methods of all its descendants, and each descendant layer can add increasingly specialized interface methods. Dynamic dispatch can also be leverage to allow descendant layers to override the interface methods of parent layers. In this way, other components are only coupled to the particular interface they utilize.

As a basic example, APEXMAP provides a *Spatial Map* concept model capable of representing various maps, which is depicted by the partial class diagram in Figure 9.3. *Spatial Map* provides a virtual method, *getOccupancyState()*, which is a capability required for any derived map model in this context. Two additional concepts, *Occupancy Grid Map* and *SDF Map*, are derived which inherit the common *getOccupancyState()* method, and add their own concept-specific methods of *getProbability()* and *getDistance()*. Any implementation model that conforms to one of the concept model interfaces can then be used to realize the interface, where this example depicts the use of *OctoMap* and *VoxBlox* for the respective implementations.

Using this architectural structure, the abstract data type *Spatial Map* can be used to declare the map object within the application, and at runtime it is dynamically instantiated using the desired implementation type. Any tasks that utilize the *getOccupancyState()*, for example, method can invoke the method directly using the *Spatial Map* type without knowing the underlying implementation type, eliminating type dependencies. If specialized methods are needed, such as *getProbability()*, the map object can be dynamically cast to the corresponding concept to expose this method, while still hiding the lower *OctoMap* implementation layer. In general, any part of the source code that uses methods of a particular layer is immune to side effects from changing the lower implementation layers. This helps greatly reduce coupling and dependencies throughout the system for increased interchangeability and modularity.

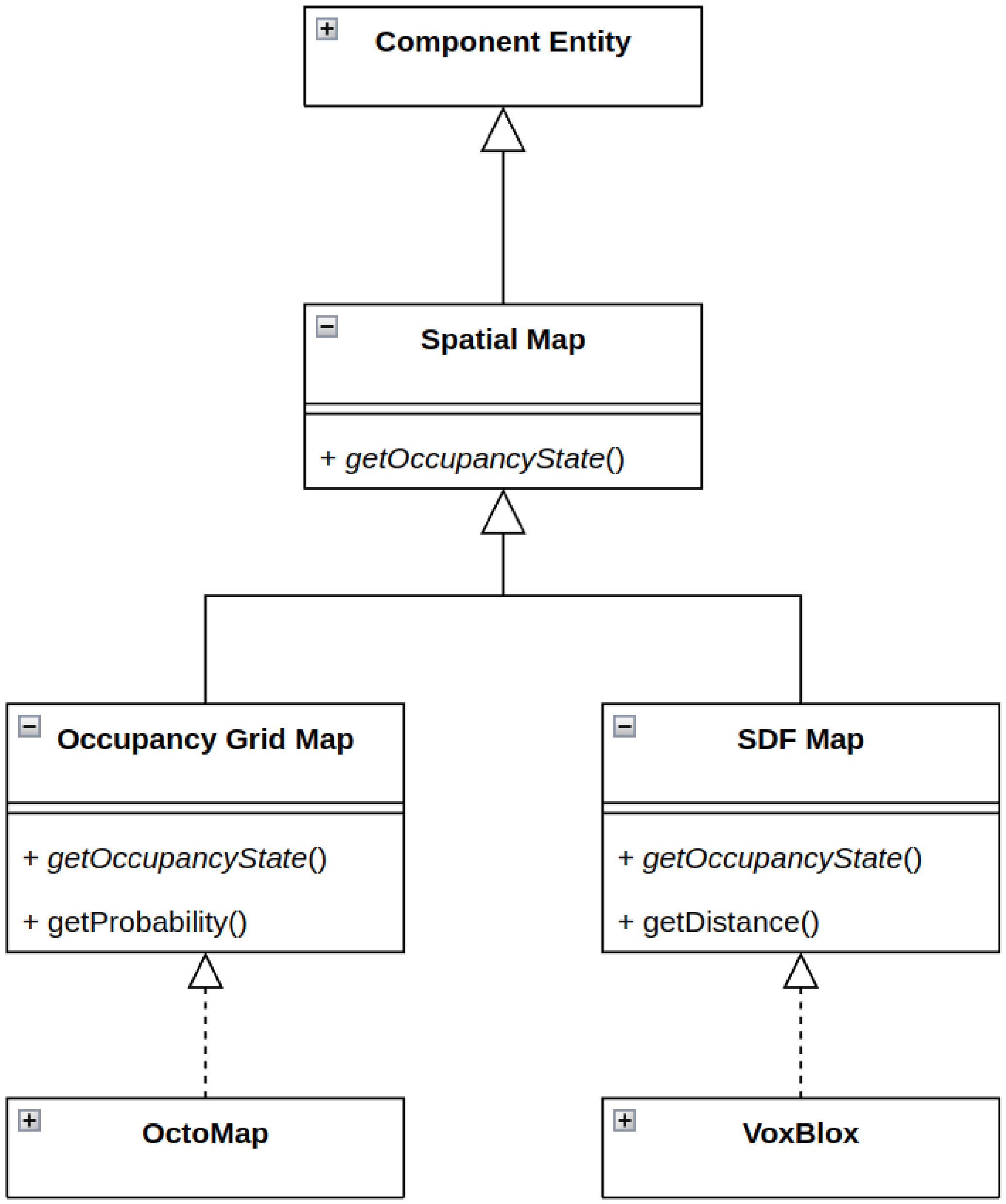


Figure 9.3: Example of Concept Model component hierarchy for spatial maps.

The ideal topology of the decomposition and its granularity is often difficult to normalize, as there may not be cleanly defined separation boundaries between the responsibilities of different subsystems, and high variability can exist within subsystems. Thus, the decomposition of concepts is made modular and open-ended, such that end-user can freely define their own concept models and implementations to meet specific requirements. Still, APEXMAP provides an extensive collection of predeveloped concept models that encompass most of the commonly occurring needs for active perceptions to reduce the development burden for end-users. Additionally, concrete realizations of each abstract model are provides that offer read-to-use capabilities to end-users, or alternatively they can develop their own implementations to conform to any specialized or non-standard requirements.

9.4.4 Subroutine Components

Subroutine Components, as shown in Figure 9.2, are used to model the prototypical concept of an algorithmic procedure, which is used to invoke high-level behaviors. They can be used to encapsulate an algorithmic procedure and any components on which it operates, hiding these details by the abstraction layer. To allow these components to be used generically, their syntax was designed to be free of input arguments or return type to account for the high variability that can occur over these aspects. Instead, any input input arguments are handled internally by extracting the data from the components that provide them, and sending any return data to the components that require it. If unique cases arise where this strategy is undesirable, specialized subtypes can be easily created by the end-user that conforms to a particular function signature as needed.

Subroutine Components are factored into several subtypes that elucidate different orthogonal behaviors. These primarily consist of *Inline Procedure*, *Async Procedure*, and their respective variants class types *Dynamic Inline Procedure* and *Dynamic Async Procedure*. The *Inline Procedure* class is used to perform a procedure that operates inline from the point of execution, blocking the current execution thread until the procedure is complete. These are useful for creating structured sequences of behavior that execute in a specific order.

The *Async Procedure* class is used to invoke a non-blocking procedure that runs asynchronously to the current execution thread. A loop termination condition can be optionally specified which will cause the procedure to repeat its execution at a specified rate until the loop condition is satisfied. This component type is useful for tasks like map building, which often should be performed continuously and in parallel to other parts of a system.

The *Dynamic Inline Procedure* and *Dynamic Async Procedure* classes both mirror the behavior of their respective parent components, with the additional capability to dynamically redefine the subroutine at runtime. One advantage of this capability is the ability to conditionally reconfigure behaviors without needing to alter the source code directly. For example, a *Dynamic Inline Procedure* object could be used to invoke the sampling stage of DFR described in Section 6.2, and a user may want to design and test variations of how this stage is performed, using a configuration file to determine which variation is assigned to the object at runtime.

9.5 Component Communication Kernel

The architecture and style for how components communicate is one of the more critical design aspects for any framework [102]. Components must frequently communicate and interact to transfer information, modify data states, or invoke operations and behaviors. The particular design patterns and protocols that govern such communications unavoidably become integrally coupled throughout many parts of the software, making later design changes to the communication strategies extremely difficult. Furthermore, the communication style can impose constraints that limit flexibility and capabilities available to end-users. The main goal for modularity and reusability is to allow components to efficiently communicate and interact in an *indirect* fashion, where objects do not require direct access to other objects which they communicate with. In this way, dependencies between object types and their interfaces are eliminated and modularity is increased.

Although communication abstraction is a common need among software across many different domains and has been extensively addressed by various existing software implementations, limitations and drawbacks can arise due to differences in the design goals of existing software and APEXMAP. Message passing and similar techniques are most frequently used, with demonstrated effectiveness using ROS messages, for example. However, a design goal for ROS messages is to enable distributed communications, which requires message serialization that introduces computational overhead. Other libraries like Boost MPI are also restricted to serialized message passing.

Other software like QT [123] and GTK [124] transfer information using signals and slots, similar to the publisher-subscriber model used in ROS, but allow communication through dynamic memory, rather than serialization. However, these software architectures are highly coupled with their target domain of User Interface (UI) design. Furthermore, they do not support other communication modalities like the server-client model used in ROS.

Specialized communication tools were developed for APEXMAP to meet its domain-specific requirements and help overcome certain limitations of existing software. Given well-demonstrated effectiveness of the message-based communication methods provided by ROS and its and prevalent usage in research software, the intent was not to develop a replacement for these methods. Instead, the design goal was to develop a unified and cohesive approach that easily supports existing ROS-based communications, while supplementing them with a versatile set of communication methods with augmented capabilities that offer flexibility and extensibility to end-users.

Message Events, described in Section 9.5.1, are used as an abstract data type primitive for information transfer. They utilize dynamic polymorphism in a similar fashion as the components described in Section 9.4 to enable specialized message subtypes to be derived for specific purposes. Message Events can be transferred using several communication modalities, which were designed in part to be supplementary to the related ROS implementations. These include the Publish-Subscribe model described in Section 9.5.3 and the Service Server model described in Section 9.5.3.

9.5.1 Message Events

APEXMAP defines a polymorphic data type primitive, *Message Event*, shown in Figure 9.4, that allows different components to communicate analogously to the message-based communication employed by ROS. A key difference is that *Message Event* types are designed for intra-process communication, whereas ROS messages are generally intended for inter-process communication. Since separate processes cannot directly access each others program memory, or even may not be running on the same physical hardware, ROS uses serialization which converts message into a structured byte stream which is reconstructed upon reception.

Serialization can be very inefficient for large data sizes or messages passed at high-frequency, and can generally cannot directly represent more complex data structures. For example, consider a collection of points stored by a k-d tree in one ROS node, which needs to be accessed by another ROS node. A custom serialization method may need to be defined for the k-d tree structure, and upon receiving this message, the tree would need to be completely reconstructed before queries operations can be called. This would be a very expensive task, especially if only one or a small number of queries are needed. Other approaches could make this more efficient, but generally would need to be customized to the intended use-case and data types, reintroducing coupling between the ROS nodes which messages are intended to eliminate.

In contrast, APEXAMAP allows messages to be transferred using dynamic memory rather than serialization, which does not require any data conversions or copying, and can handle data of arbitrary type and complexity. Additionally, they can be easily adapted to facilitate ROS-style communications by encapsulation of the desired ROS message. This encapsulation is hidden by the abstraction layer, limiting dependencies on the specific ROS message type apart from the small parts of the program responsible for interacting with it.

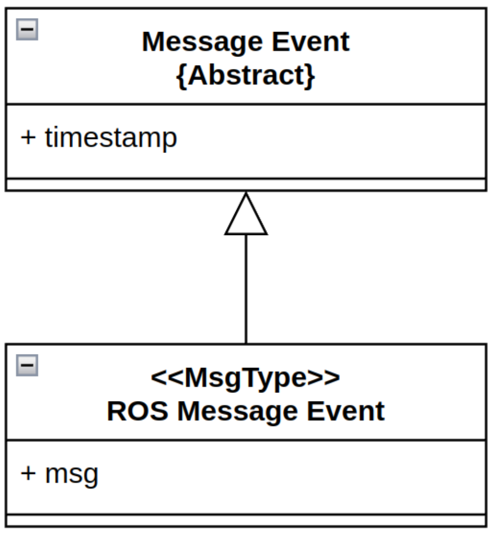


Figure 9.4: Message event model.

9.5.2 Publish-Subscribe Module

The Publish-Subscribe Module, outlined in Figure 9.5, facilitates one-to-many communications where a publisher can transmit information to an arbitrary number of subscribers, similar to the role of publishers and subscribers in ROS. The basic functionalities are provided primarily using two base components, the *Event Publisher* class and *Event Subscriber* class, which are derived from *Component Entity* to equip them with dynamic abstraction. *Event Publisher* allows objects to be created that specify a named communication channel, referred to as its topic, and objects created using *Event Publisher* can be attached to the topic to receive its communications. Unlike ROS, the information transferred on a topic does not require serialization. Instead, the *Message Event* class is used to abstractly encapsulate any type of message information. *Message Event* objects provide no information on their own, but are to be used to derive additional concrete message types for the application.

Using the previous k-d tree example, a *Message Event* could simply store a pointer or reference to the original object. Similarly, they could be used to store ROS message types, and the corresponding *Event Publisher* object could encapsulate a ROS publisher without exposing these implementation details to external objects.

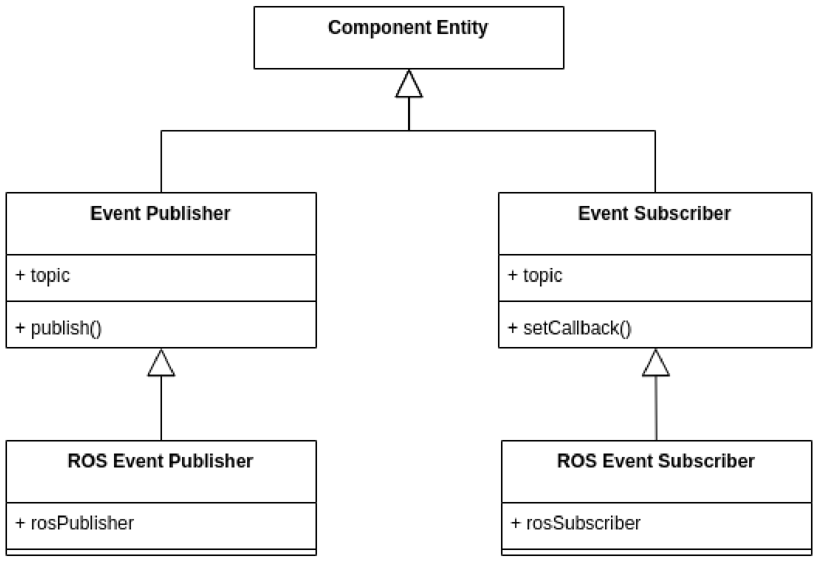


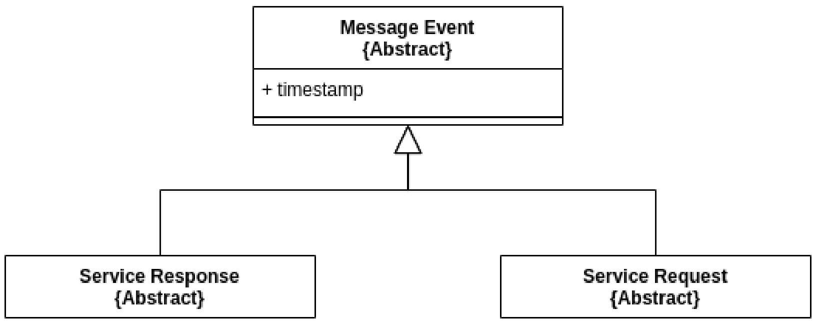
Figure 9.5: Partial class diagram of the Publish-Subscribe Module.

9.5.3 Client-Server Module

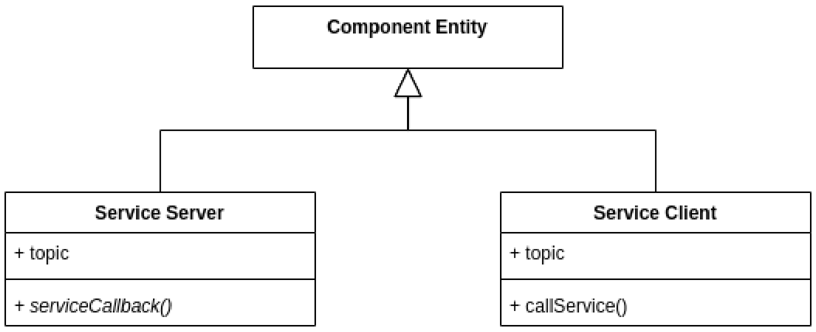
The Service Server Module, illustrated in Figure 9.6, provides communication patterns that allow a behavior, often referred to as a *service* in this context, to be defined by a singular server component, and any number of client components can invoke requests for the service. This capability can be compared to the service-based communication aspects in ROS. Certain communication tasks are better achieved using services when common behavior is needed for multiple components, and each individual component only needs to control when it is invoked. For example, one component may use a service as a trigger to request a planning process defined in another component. The requesting component does not need to be aware of which component provides the service or how it is performed, eliminating coupling between them.

9.6 Dynamic Knowledge Maintenance and Continuity

A defining constituent of active perception that of *dynamic knowledge acquisition.* The environment contains the ground truth, or *axiomatic information*, that is sought



(a) Service message interfaces.



(b) Diagram of the service communication components.

Figure 9.6: Partial class diagram of the Client-Server Module elements.

by the agent. This information is acquired by onboard sensor measurements that are used to build a coherent environment model, generally embodied by the spatial map model, representing the *a posteriori* empirical knowledge of the environments real-world form.

To increase understanding and facilitate planning, inference and reasoning can be applied to existing knowledge to derive additional forms of *analytical knowledge*. Analytical knowledge can be derived directly from the map model, or can be chained to derived from other forms of analytical knowledge. For example, viewpose candidates samples from the map represent analytical knowledge derived directly from the map, but are not considered empirical knowledge since they do not correspond to real features in the environment. Information gain represents an example of chained analytical knowledge, since it determined with respect to an existing viewpose element.

Once knowledge is computed, it is generally desired to store the results in a data structure that accumulates and preserves the state data for efficient use later on. Since analytical knowledge is conditioned on the dynamic state of the empirical knowledge, regular maintenance is required to ensure *data continuity* is sustained as the empirical knowledge dynamically changes.

Analytical knowledge facilitates better understanding and can simplify planning tasks, however, it can have high computational complexity to compute and maintain. The complexity to maintain the data must be balanced with the intended complexity reductions it facilitates for planning or other tasks. Partial reevaluation can be facilitated by storing intermediate information that is otherwise not directly useful for planning, but allows reevaluation to be more efficient, with potential tradeoffs in memory complexity that must also be considered.

For example, information gain is typically stored as a numerical value resulting from a series of raycasting operations which are discarded after the computation. Upon reevaluation, all of the raycasting operations must be fully recomputed, many or all of which may produce the same result as from the initial evaluation. However, it is feasible to preserve some of the intermediate raycasting results to reduce or eliminate wasteful computations, with the tradeoff being increased memory usage to store the intermediate data.

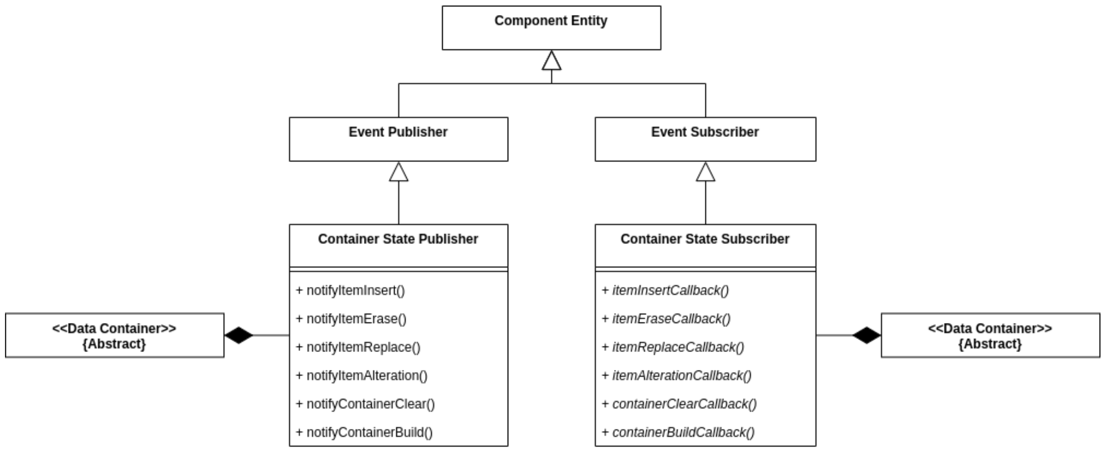


Figure 9.7: Data state change communication model.

From a reusable software engineering perspective, the maintenance of dynamic knowledge presents several challenges. Interface designs should be agnostic to the data types of the information, and should also be decoupled from the other data structures that manage or interact with the information. Additionally, the ability to cache partial computational results and dynamically reapply them is critical to improving performance, but must also be made modular and generalizable.

To simplify the generic design of efficient dynamic data maintenance strategies, APEXMAP provides a collective of generic and reconfigurable tools that serve as building blocks to simplify and expedite the design of more complex models. Firstly, the design patterns related to abstract component entities and indirect communication capabilities provide powerful capabilities for designing *reactive* data structure components. Reactive components can be designed by utilizing event messages to communicate any type of dynamically changing information from anywhere within the system, and any other component can subscribe to this data and implement is own customized reactive policy.

To further facilitate such designs, an extensive data container library was developed which provides variants of most standard data container types that have been adapted to natively provide interfaces to communicate their state changes. A component diagram of these design patterns is shown in Figure 9.7, which extends the *Event Publisher* and *Event Subscriber* interfaces to derive the respective interfaces of *Container State Publisher* and *Container State Subscriber*.

These derived communication models provide a standard generic form, where the class *Container State Publisher* is used to transmit the state changes induced whenever a modifying container operation is performed. *Container State Publisher* is then encapsulated in the implementation of a specific container type, and can be subscribed to by any other container that contains the corresponding *Container State Subscriber* interface. The *Container State Subscriber* interface can also be freely used in any other type of component, rather than only container model types. The methods of *Container State Subscriber* can be independently reconfigured to specialize the response behavior when state change information is received.

9.7 APN-P System Implementation

This section provides a description of how APEXMAP was used to implement the APN, DFR, and APN-P approaches presented in the previous chapters. Further details are included on how the various aspects of the program can be efficiently modified to analyze different behaviors with minimal programming changes. Additional examples are provided on how APEXMAP can generalize to other existing approaches, and increase their modularity to handle future development.

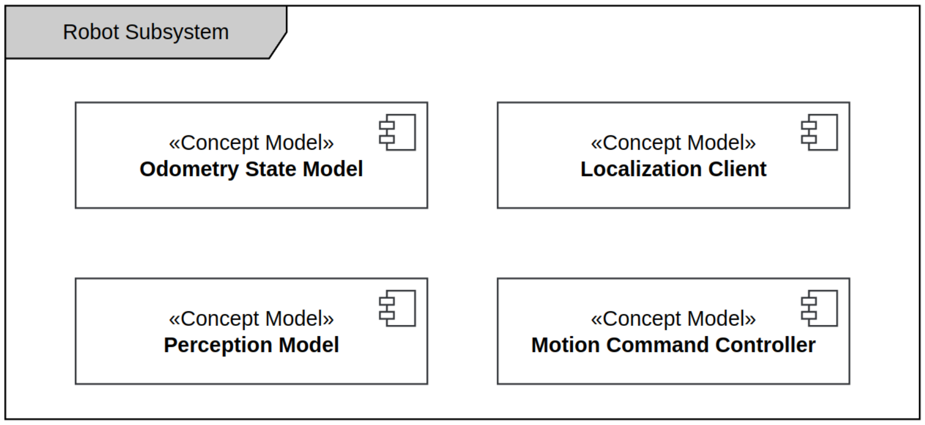

Robot Subsystem
«Concept Model» Odometry State Model
«Concept Model» Localization Client
«Concept Model» Perception Model
«Concept Model» Motion Command Controller

Figure 9.8: Robot subsystem component diagram.

9.7.1 Robot Subsystem

The *robot subsystem* provides the fundamental concept models related to the sensing, control, and other aspect of the robot, depicted in Figure 9.8. Its primary components handle tasks related to representation of the odometry state and perception models, and provides components for sending motion commands using the *Motion Command Controller*, and for receiving localization state information by the *Localization Client* component.

Localization Client makes use of the Publish-Subscribe protocols to provide a standard communication topic for the localization state, which can be subscribed by any other system component. This approach decouples the details of how this state is acquired from the the rest of the system, providing very high modularity and reusability. The particular approach can be completely hidden from the external components, which depend only on the output communication topic, independent from how it is computed.

For example, localization could be computed and published by a separate ROS node, in which *Localization Client* can implement a ROS subscriber to receive the data and republish it to the system. Alternatively, an existing localization procedure could be encapsulated and run asynchronously by the *Async Procedure* class, where the rest of the system requires only that the output results are published on the corresponding topic provided by *Localization Client*.

9.7.2 Mapping Subsystem

The components of the mapping subsystem is shown in Figure 9.7.2. This subsystem is centered around the concept model component *Spatial Map*, which provides the abstract interface of the map, and its implementation is realized using the OctoMap library. The subtask of building the map is divided into several components, which provide modularity between different data type representations of the map model, sensor data, and how the sensor data is integrated.

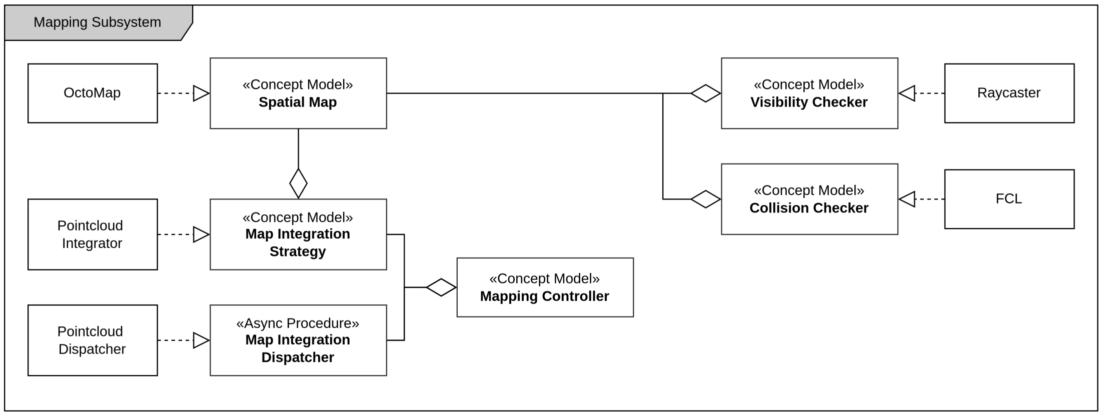

Mapping Subsystem
OctoMap
«Concept Model» Spatial Map
«Concept Model» Visibility Checker
Raycaster
«Concept Model» Collision Checker
FCL
Pointcloud Integrator
«Concept Model» Map Integration Strategy
«Concept Model» Mapping Controller
Pointcloud Dispatcher
«Async Procedure» Map Integration Dispatcher

Figure 9.9: Mapping subsystem component diagram.

The *Map Integration Dispatcher* component is responsible for receiving the sensor inputs, decoupling their data types and the mechanisms behind their communication and reception. In our implementation, sensor inputs are received in the form of a pointcloud, and the *Pointcloud Dispatcher* class is predefined component provided by APEXMAP to handle these.

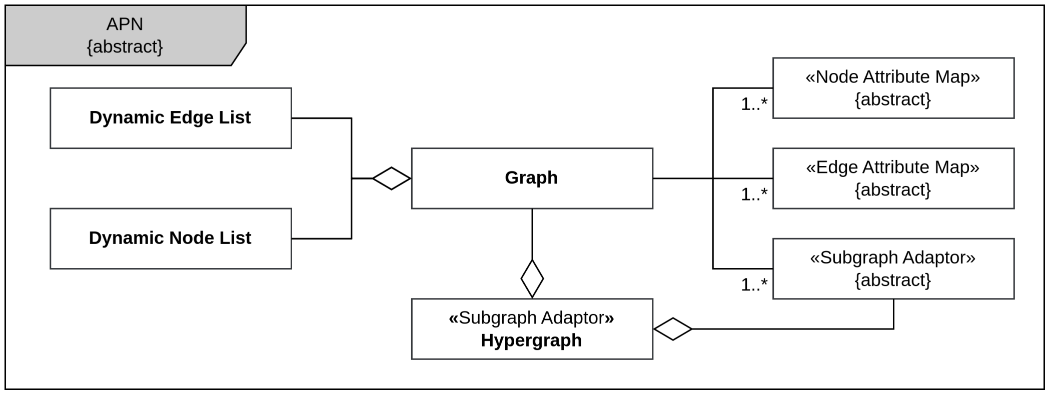
Upon receiving the sensor inputs, the dispatcher sends them to the *Map Integration Strategy* component, which separately handles how they are used to update the map. Similarly, APEXMAP provides a pre-implemented specialization of this component, *Pointcloud Integrator*, to perform the necessary operations. *Map Integration Dispatcher* is modeled as the *Async Procedure* concept, which allows it to continuously run concurrent to the rest of the system. This component is responsible for executing the integration strategy upon receiving each sensor scan. Finally, *Mapping Controller* serves as an encapsulation of the other two components, simplifying their setup and operations.

Two additional concepts are also used, consisting of the *Visibility Checker* and *Collision Checker* components indicated in Figure 9.7.2. These provide generalized abstraction layers for the common operations involved for visibility checking and collision checking, allowing the concrete methods to perform them can be hidden from the rest of the system. In our implementation, visibility checking uses a custom class *Raycaster* (included in APEXMAP), and the FCL library is used for collision checking.

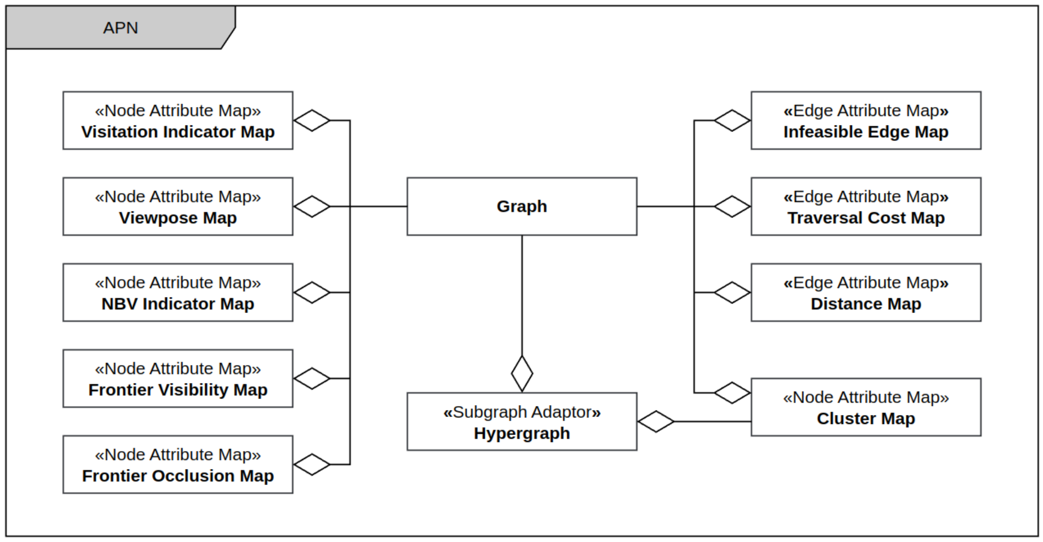
9.7.3 APN Subsystem

The structural diagram of the APN subsystem is depicted in Figure 9.10, with its abstraction model shown in (a) and its implementation in (b). The abstract representation of the APN is defined by the *Graph* and *Hypergraph* interfaces, which are represented using generic data types independent from any other part of the system. Instead, the concrete data types needed for the graph are reconfigurable through associative data structures, primarily consisting of *Node Attribute Map* and

Edge Attribute Map. The APN can be reconfigured to have any number of different attribute types associated with the graph structure.



(a)



(b)

Figure 9.10: APN subsystem component diagram. The abstract model is shown in (a), and its realization is shown in (b).

An additional interface type, *Subgraph Adaptor*, is also provided which allows the basis graph structure to be efficiently represented as subgraph by conditional inclusion/exclusion of its underlying nodes and edges, without needing copy their memory structure or any of their data associations. This structure is used as the base structure for efficiently constructing the *Hypergraph* model, which is derived from the graph model, and can be similarly reconfigured to associate its abstract compositional elements to other concrete data types.

Given the abstract component model, the primary elements of our implementation are depicted in Figure 9.10b. This diagram shows the use of the associative data structures to define the various data concepts defined by the theoretical models, such as viewposes, frontier visibility states, and edge costs.

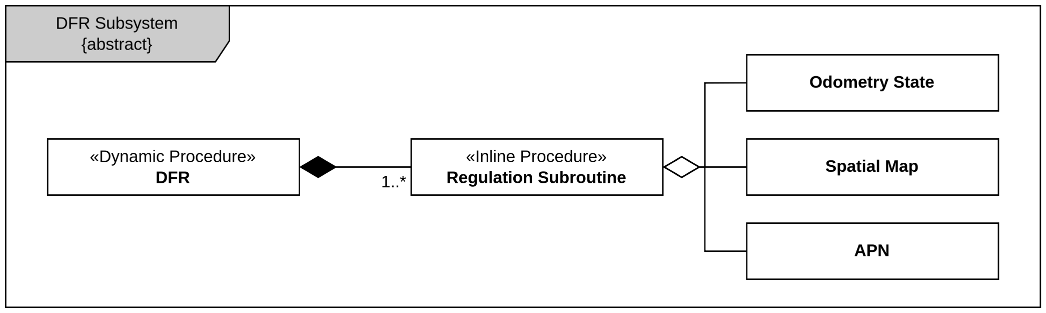
9.7.4 DFR Subsystem

The overview of the DFR subsystem is similarly depicted in Figure 9.11, with the abstract model shown in 9.11a. A standardized component type, *Regulation Procedure*, is defined to represent an individual update process to be applied to the APN, and encapsulates the relevant data needed for the update. These can be dynamically defined and added to the component type *DFR*, which then executes them as a single linear process.

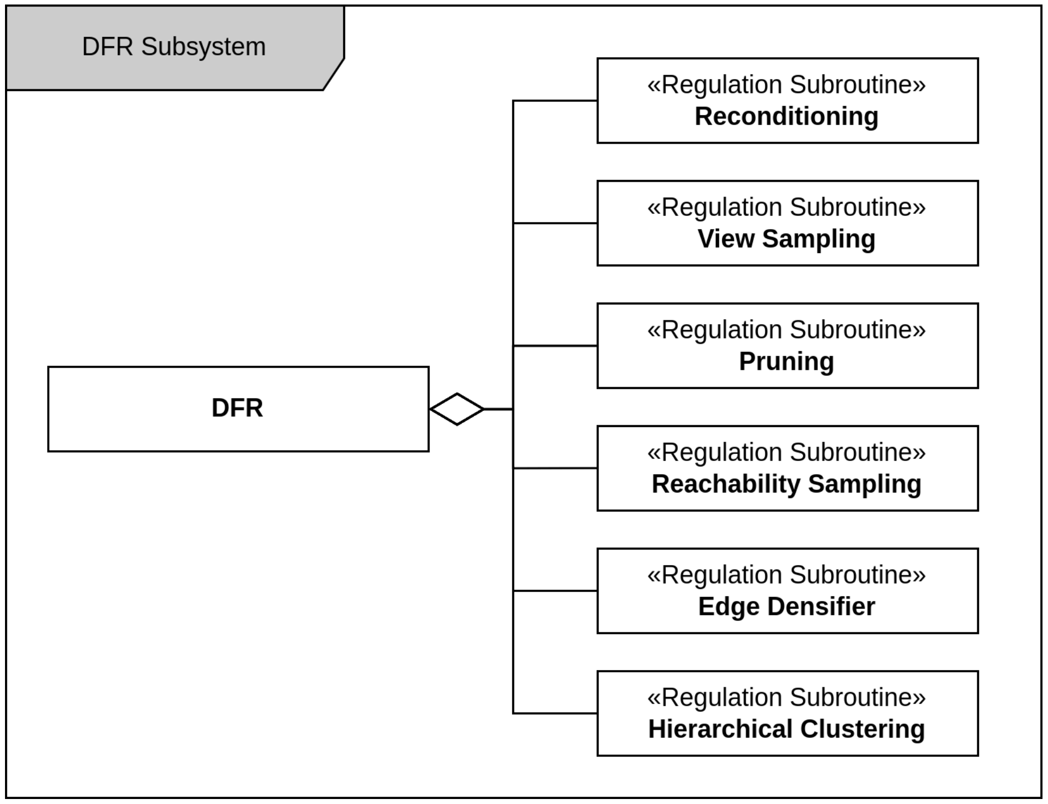
The specific realization of the abstract DFR subsystem is illustrated for our implementation in Figure 9.11b, where each subprocess corresponds to those outlined in Chapter 6. This approach offers high modularity in multiple regards. Each process represents a task modeled by the generic form of the *Procedure* interface, which hides the details over the tasks purpose or how it is achieved. Given an application that has defined a particular set of tasks, their implementation can be dynamically and transparently redefined. Furthermore, additional tasks can be defined independently, then simply added to the DFR procedure sequence to execute their behavior.

9.7.5 Planning Subsystem

The structural diagram of the planning subsystem layer is summarized in Figure 9.12. This layer provides further abstraction and decoupling between the maintenance of the APN, and how it is applied for planning purposes. The class *APN Planner* is modeled using the *APN Planner* component type, allowing the behavioral semantics



(a)



(b)

Figure 9.11: DFR subsystem component diagram. The abstract model is shown in (a), and its realization is shown in (b).

of the planner to have a generic form, hiding its approach details which can be dynamically redefined. This component contains a reference to the *APN* model, and the openGA library [125] was used to support the base implementation structure of the evolutionary planner described in Chapter 7.

An additional class, *Mission Controller*, serves as the high level executive controller

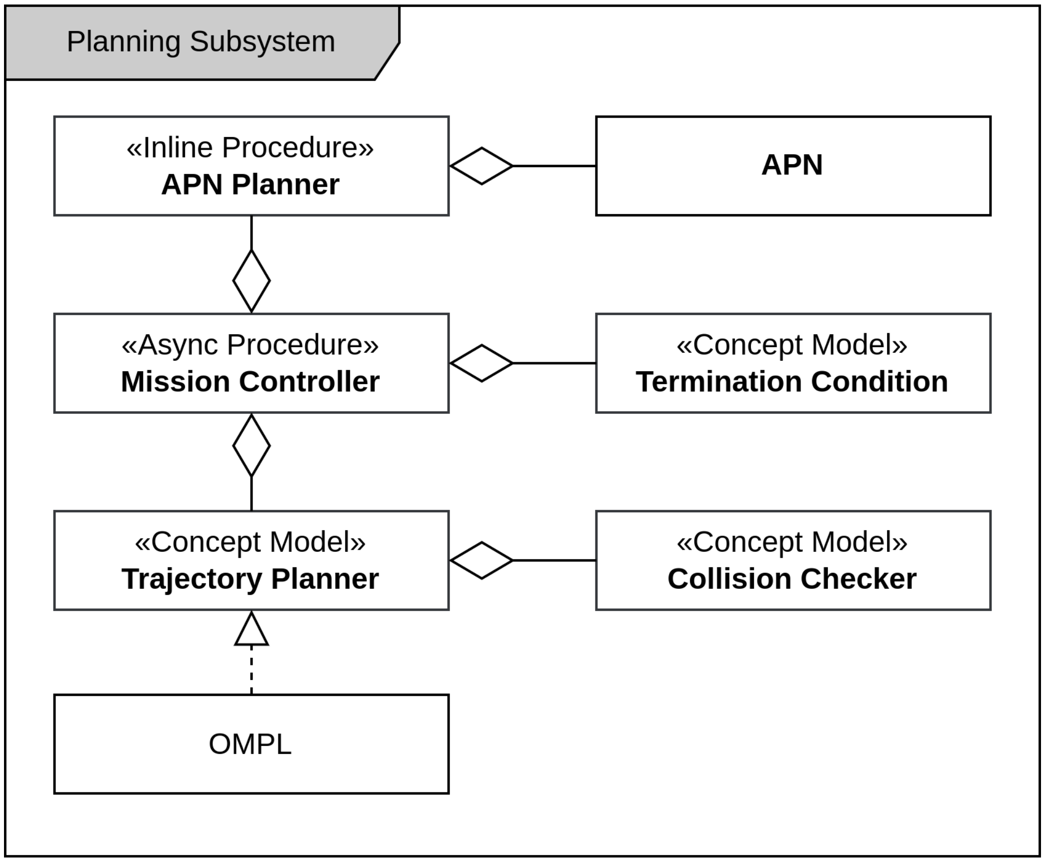


Figure 9.12: Planning subsystem component diagram.

over the runtime operation of the system. It receives the planning output of the *APN Planner*, and is responsible for the motion planning tasks needed to achieve the goal using the *Trajectory Planner* model. *Trajectory Planner* can be reconfigured to used different motion planning algorithms, where OMPL was selected for use in our implementation. The executive controller operates continuously as a concurrent process to mapping and other subsystems, using *Termination Condition* to specify the desired termination conditions for mission completion.

9.7.6 Application Design and Execution

At the uppermost level of control flow of the application, the component-based object system enables the application to be completely represented by a single homogeneous collection of objects. This is depicted by Figure 9.13, which illustrates only the high-level subsystems containing the objects for conciseness.

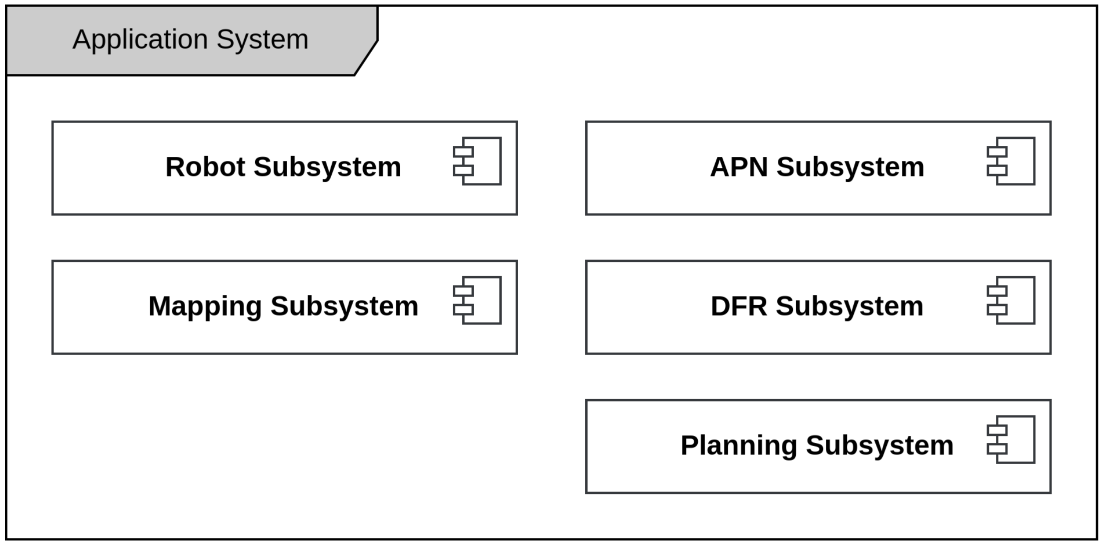


Figure 9.13: Application system diagram.

Each object independently defines its own configuration and setup requirements which are hidden by their abstraction layers, such that implementation of the run-time application can be greatly simplified. The *Component Resource Server* is used to allocate an object instance for each subsystem component. Then, the runtime execution control flow is depicted in Figure 9.14, which demonstrates the key stages of execution. After all objects have been allocated, each stage of operation is called from the main program using the *Component Resource Server*, which corresponds to the entry points of init(), setup(), start(), and stop() in the diagram.

From the diagram of Figure 9.14, it can be seen that none of the implementation-specific details of our approach are exposed. This is indicative of the ability for the control flow structure to take a standardized structural form that is application independent. In this way, once the necessary component classes have been defined, any application can be quickly built by simply specifying the desired components it needs, where the rest of their setup, configuration, and operation procedures can be largely handled in an automated fashion. The modularity of the components then helps to maximize the reuse of existing components for different purposes, and the pre-implemented components natively provided by APEXMAP minimize the amount of end-user work related to common subproblems. These aspects help to greatly reduce the amount of time and effort needed from the end-user by reducing much of the tedious boilerplate code needed throughout development, contributing to the underlying goals of rapid development.

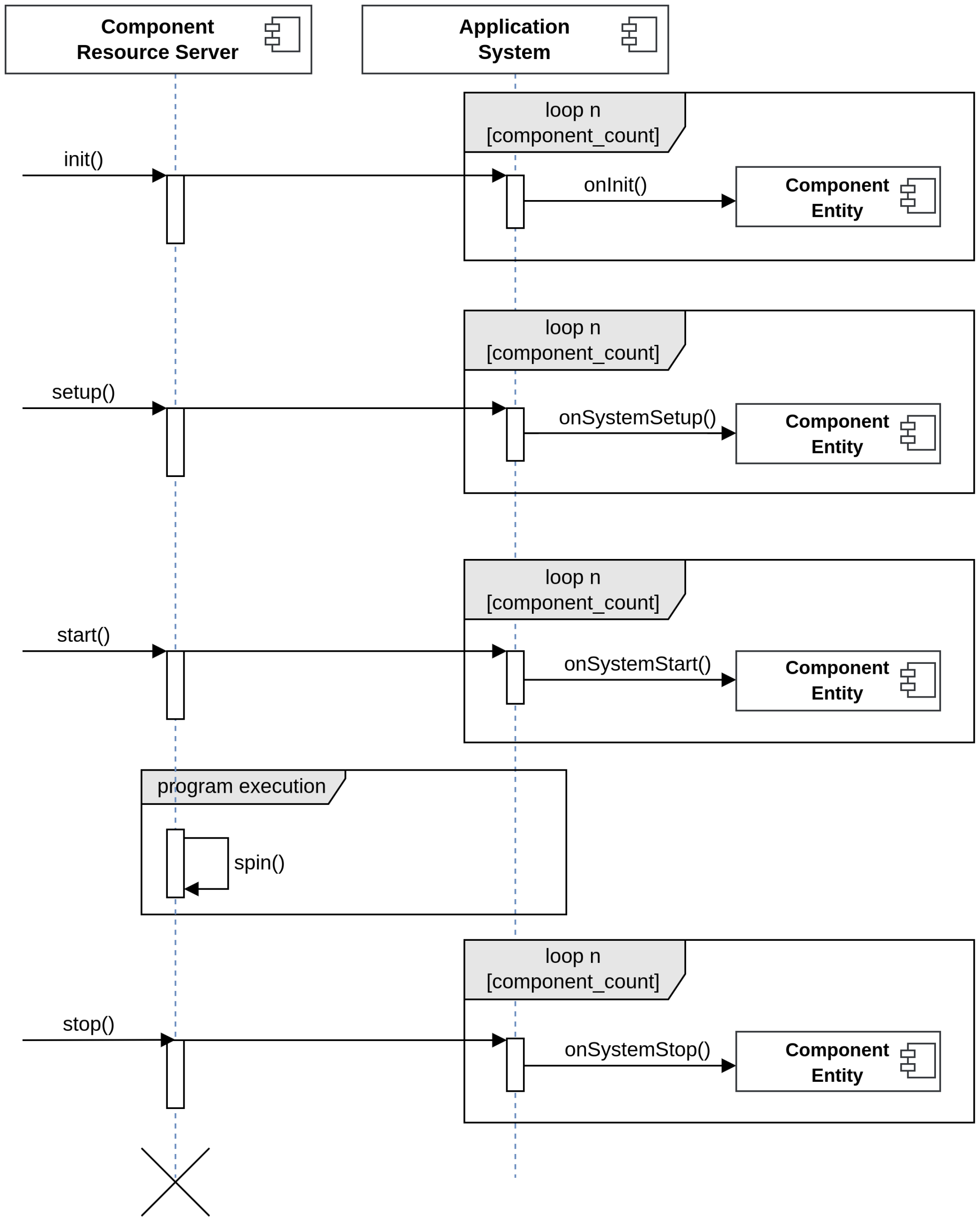


Figure 9.14: Application control flow diagram.

CHAPTER 10: CONCLUSIONS

This dissertation has presented the Active Perception Network (APN), serving as a topological roadmap of the dynamically changing exploration state space, the differential regulation (DFR) update procedure that incrementally adapts the APN to the changing environment knowledge, and an exploration planner APN-P, which leverages the APN to find non-myopic exploration sequences through the APN.

The performance results demonstrated the efficiency of DFR in performing each cyclic update and its scalability with increasing map sizes. In comparison to several state-of-the-art approaches, the non-myopic planning approach of APN-P consistently achieved improved performance in terms of total exploration time and coverage completeness. The ability to generalize was also demonstrated over a variety of different environments, both indoor and outdoor, with only minor parameter adjustments between them.

The APEXMAP open-source software framework was developed to address the software engineering challenges inherent to online exploration, many of which generalize to the problem domain of active perception as a whole. The large-scale and complex software development tasks creates an entry-barrier to researchers and typically results in strong coupling with problem formulation and approach-specific requirements, limiting its applicability. APEXMAP helps to reduce the research entry-barrier and increase software reusability over a broad range of problems and approaches, which can accelerate ongoing increase research progress and innovations in this domain.

10.1 Contributions

This dissertation addressed the problem of online exploration in unknown environments, which is highly relevant to many applications. Limitations of existing research is largely driven by the high problem complexity and scalability. The majority of works focus myopic planning that plan each incremental action using greedy strategies which can be incomplete, and are incapable of providing performance guarantees for the long-term global objectives. We address these limitations and the performance quality, complexity, and scalability challenges by the following contributions:

- A novel dynamic multi-layer topological graph designated as the *Active Perception Network* (APN). The APN serves as a global hierarchical roadmap over the spatial map that accumulates the incrementally computed knowledge of the exploration state space. The APN provides a model of the information space can be efficiently search and queried for planning tasks, and can be dynamically updated with high efficiency.

- A dynamic update procedure referred to as *Differential Regulation* (DFR) to incrementally build and refine the APN as environment knowledge is increased. DFR leverages the incremental and local nature of map changes minimize the complexity of updating the APN as its size and the map scale increase. This achieves increased efficiency and scalability by ensuring the size of the processed data remains locally bounded, relying less on simplifications of the update procedures themselves.

- A non-myopic planning approach denoted as APN Planner (APN-P) that demonstrates how the APN can be leveraged to adaptively compute and refine a globally informed exploration sequence.

- A detailed performance analysis and comparison to existing approaches among the state-of-the-art.

- The APEXMAP software framework was present which plays a vital practical role in realizing the methodology and its performance efficiency. It provides the implementation details related to the implementation of the APN, DFR, and planning methodologies, which are built from a highly modular and efficient set of underlying software components that allow the framework to effectively generalize to a wide array of different planning methodologies or to solve for different problems in active perception. APEXMAP is made open-source [126] for the benefit of the research community and to promote accelerated innovations through collaborative research.

10.2 Future Work

A variety further research directions have been identified related to addressing remaining challenges in the problem domain, and potential insights that could be valuable from further analysis of our methodologies. These were deemed beyond the scope of this dissertation, but will be briefly discussed in this section.

Further system analysis could provide a more thorough understanding of how each aspect of the approach contributes to the the overall performance. This may also help identify any existing functional limitations or weaknesses and how they might be addressed. Future studies could include parametric analysis, evaluation of stability and robustness under varying conditions, and ablation studies. A variety of parameters are used in the approach which are beneficial in controlling its behaviors, but further analysis is needed to better understand their interactions, sensitivities, and how they should be tuned to optimize the system performance under different conditions.

Several involved subproblems are able to be approached by families of related algorithms, where the specific choice of algorithm could have unknown effects on the overall operation. Evaluating the overall system behaviors using varying algorithms for its subproblems would be beneficial. For example, different clustering methods for hierarchical decomposition could be worth investigating. The current clustering algorithm does not constrain the number of clusters formed, which can be advantageous by allowing them to better conform to the underlying data, rather than imposing artificial restrictions. However, extremal cases can occur where either a single cluster is formed that contains all nodes, or each node is assigned its own cluster. Both are degenerate cases which reduce to a single TSP instance over the complete node set. While in practice this was not typically observed, it suggests that some optimal balance may exist between these two extrema.

Several strategies to further improve performance have been conceptualized. There are potential opportunities to further optimize performance by increasing the amount of computational reuse between planning iterations. This can involve increasing the amount of cached data artifacts from sub-processes that can be efficiently reapplied to reduce redundant computations. This involves a tradeoff where memory is increased to replace the repetitive computations with a lookup operation of their previous results. It would be insightful to study how to ideally balance such tradeoffs.

There is significant potential in the generalizability of the conceptual approach and its modular software implementation to handle further problem variations. This could be leverage to to consider reformulated problem objectives for different active perception problems, such as free space exploration rather than surface coverage, or search and recognition of object targets. Additional reformulations could include operation using mobile ground robots for 2D exploration tasks.

The evolutionary optimization strategy and its software implementation (using openGA) can be readily adapted to handle multiobjective optimization problems. This could allow for consideration of uncertainty as a joint optimization problem to the exploration objective. The current high computational efficiency provides excellent latitude to handle the increased problem complexity. This could make the approach more robust for practical use in GPS-denied environments.

Hardware experiments will be useful to validate the practical feasibility on a physical hardware system, and to evaluate the realistic performance in non-ideal conditions with noise and uncertainty. This is a necessary next step to help advance the methodology for practical applications.

REFERENCES

[1] Anqi Xu, Chatavut Viriyasuthee, and Ioannis Rekleitis. Optimal complete terrain coverage using an unmanned aerial vehicle. In *2011 IEEE International conference on robotics and automation*, pages 2513–2519. IEEE, 2011.

[2] Marina Torres, David A Pelta, José L Verdegay, and Juan C Torres. Coverage path planning with unmanned aerial vehicles for 3d terrain reconstruction. *Expert Systems with Applications*, 55:441–451, 2016.

[3] Juan Irving Vásquez and L Enrique Sucar. Next-best-view planning for 3d object reconstruction under positioning error. In *Mexican International Conference on Artificial Intelligence*, pages 429–442. Springer, 2011.

[4] J Irving Vasquez-Gomez, L Enrique Sucar, Rafael Murrieta-Cid, and Efrain Lopez-Damian. Volumetric next-best-view planning for 3d object reconstruction with positioning error. *International Journal of Advanced Robotic Systems*, 11(10):159, 2014.

[5] Qing Li, Da-Chuan Li, Qin-fan Wu, Liang-wen Tang, Yan Huo, Yi-xuan Zhang, and Nong Cheng. Autonomous navigation and environment modeling for mavs in 3-d enclosed industrial environments. *Computers in Industry*, 64(9):1161–1177, 2013.

[6] Youngjib Ham, Kevin K Han, Jacob J Lin, and Mani Golparvar-Fard. Visual monitoring of civil infrastructure systems via camera-equipped unmanned aerial vehicles (uavs): a review of related works. *Visualization in Engineering*, 4(1):1, 2016.

[7] Zehui Meng, Hailong Qin, Ziyue Chen, Xudong Chen, Hao Sun, Feng Lin, and Marcelo H Ang Jr. A two-stage optimized next-view planning framework for

3-d unknown environment exploration, and structural reconstruction. *IEEE Robotics and Automation Letters*, 2(3):1680–1687, 2017.

[8] Alberto Quattrini Li. Exploration and mapping with groups of robots: Recent trends. *Current Robotics Reports*, 1:227–237, 2020.

[9] Sebastian B Thrun and Knut Möller. Active exploration in dynamic environments. *Advances in neural information processing systems*, 4, 1991.

[10] Hans Jacob S Feder, John J Leonard, and Christopher M Smith. Adaptive mobile robot navigation and mapping. *The International Journal of Robotics Research*, 18(7):650–668, 1999.

[11] Frederic Bourgault, Alexei A Makarenko, Stefan B Williams, Ben Grocholsky, and Hugh F Durrant-Whyte. Information based adaptive robotic exploration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 540–545. IEEE, 2002.

[12] Iker Lluvia, Elena Lazkano, and Ander Ansuategi. Active mapping and robot exploration: A survey. *Sensors*, 21(7):2445, 2021.

[13] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, and Maxim Batalin. Efficient planning of informative paths for multiple robots. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2204–2211, 2007.

[14] Jonathan Binney, Andreas Krause, and Gaurav S Sukhatme. Informative path planning for an autonomous underwater vehicle. In *2010 IEEE International Conference on Robotics and Automation*, pages 4791–4796. IEEE, 2010.

[15] Cyrill Stachniss and Wolfram Burgard. Mapping and exploration with mobile robots using coverage maps. In *Proceedings 2003 IEEE/RSJ International Con-*

ference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), volume 1, pages 467–472. IEEE, 2003.

[16] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[17] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[18] Di Deng, Runlin Duan, Jiahong Liu, Kuangjie Sheng, and Kenji Shimada. Robotic exploration of unknown 2d environment using a frontier-based automatic-differentiable information gain measure. In *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1497–1503. IEEE, 2020.

[19] Ana Batinovic, Tamara Petrovic, Antun Ivanovic, Frano Petric, and Stjepan Bogdan. A multi-resolution frontier-based planner for autonomous 3d exploration. *IEEE Robotics and Automation Letters*, 6(3):4528–4535, 2021.

[20] Hanlin Niu, Yu Lu, Al Savvaris, and Antonios Tsourdos. An energy-efficient path planning algorithm for unmanned surface vehicles. *Ocean Engineering*, 161:308–321, 2018.

[21] Chunhui Zhou, Shangding Gu, Yuanqiao Wen, Zhe Du, Changshi Xiao, Liang Huang, and Man Zhu. The review unmanned surface vehicle path planning: Based on multi-modality constraint. *Ocean Engineering*, 200:107043, 2020.

[22] Brendan Englot and Franz Hover. Sampling-based coverage path planning for inspection of complex structures. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 29–37, 2012.

[23] Hailong Qin, Zehui Meng, Wei Meng, Xudong Chen, Hao Sun, Feng Lin, and Marcelo H Ang. Autonomous exploration and mapping system using heterogeneous uavs and ugvs in gps-denied environments. *IEEE Transactions on Vehicular Technology*, 68(2):1339–1350, 2019.

[24] Y. Lu, Z. Xue, G-S. Xia, and L. Zhang. A survey on vision-based uav navigation. *Geo-spatial information science*, 21(1):21–32, 2018.

[25] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[26] Francesco Amigoni, Wonpil Yu, Torsten Andre, Dirk Holz, Martin Magnusson, Matteo Matteucci, Hyungpil Moon, Masashi Yokotsuka, Geoffrey Biggs, and Raj Madhavan. A standard for map data representation: Ieee 1873-2015 facilitates interoperability between robots. *IEEE Robotics & Automation Magazine*, 25(1):65–76, 2018.

[27] Aolei Yang, Yu Luo, Ling Chen, and Yulin Xu. Survey of 3d map in slam: localization and navigation. In *Advanced Computational Methods in Life System Modeling and Simulation*, pages 410–420. Springer, 2017.

[28] XZ Zhang, Ahmad B Rad, Yiu-Kwong Wong, George Quan Huang, Ying-Leung Ip, and Kai Ming Chow. A comparative study of three mapping methodologies. *Journal of Intelligent and Robotic Systems*, 49(4):385–395, 2007.

[29] A. Hornung. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(5):189–206, 2013.

[30] Matt Whalley, Marc Takahashi, P Tsenkov, G Schulein, and C Goerzen. Field-testing of a helicopter uav obstacle field navigation and landing system. In *65th Annual Forum of the American Helicopter Society, Grapevine, TX*, 2009.

[31] Peter Tsenkov, Jason Howlett, Matthew Whalley, Greg Schulein, Marc Takahashi, Matthew Rhinehart, and Bernard Mettler. A system for 3d autonomous rotorcraft navigation in urban environments. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7412, 2008.

[32] Sheraz Khan, Athanasios Dometios, Chris Verginis, Costas Tzafestas, Dirk Wollherr, and Martin Buss. Rmap: a rectangular cuboid approximation framework for 3d environment mapping. *Autonomous Robots*, 37:261–277, 2014.

[33] Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous visual mapping and exploration with a micro aerial vehicle. *Journal of Field Robotics*, 31(4):654–675, 2014.

[34] Cheng Zhu, Rong Ding, Mengxiang Lin, and Yuanyuan Wu. A 3d frontier-based exploration tool for mavs. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 348–352. IEEE, 2015.

[35] Peter Beno, Vladimír Pavelka, Frantiek Duchon, and Martin Dekan. Using octree maps and rgbd cameras to perform mapping and a* navigation. In *Intelligent Networking and Collaborative Systems (INCoS), 2016 International Conference on*, pages 66–72. IEEE, 2016.

[36] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[37] Margarida Faria, Ivan Maza, and Antidio Viguria. Applying frontier cells based exploration and lazy theta* path planning over single grid-based world representation for autonomous inspection of large 3d structures with an uas. *Journal of Intelligent & Robotic Systems*, pages 1–21, 2018.

[38] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2161–2168. Ieee, 2006.

[39] Dagmar Lang and Dietrich Paulus. Semantic maps for robotics. In *Proc. Workshop Workshop AI Robot.(ICRA)*, pages 14–18, 2014.

[40] Xiaoning Han, Shuailong Li, Xiaohui Wang, and Weijia Zhou. Semantic mapping for mobile robots in indoor scenes: a survey. *Information*, 12(2):92, 2021.

[41] Sebastian Thrun. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping*, pages 13–41. Springer, 2007.

[42] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[43] Sherif AS Mohamed, Mohammad-Hashem Haghbayan, Tomi Westerlund, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7:97466–97486, 2019.

[44] Josep Aulinas, Yvan R Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *CCIA*, 184(1):363–371, 2008.

[45] Tang Swee Ho, Yeong Che Fai, and Eileen Su Lee Ming. Simultaneous localization and mapping survey based on filtering techniques. In *2015 10th Asian Control Conference (ASCC)*, pages 1–6. IEEE, 2015.

[46] Vandad Imani, Keijo Haataja, and Pekka Toivanen. Three main paradigms of simultaneous localization and mapping (slam) problem. In *Tenth International Conference on Machine Vision (ICMV 2017)*, volume 10696, page 106961P. International Society for Optics and Photonics, 2018.

[47] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[48] Georges Younes, Daniel Asmar, and Elie Shammas. A survey on non-filter-based monocular visual slam systems. *arXiv preprint arXiv:1607.00470*, 413:414, 2016.

[49] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment - a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[50] Brian Williams, Mark Cummins, José Neira, Paul Newman, Ian Reid, and Juan Tardós. A comparison of loop closing techniques in monocular slam. *Robotics and Autonomous Systems*, 57(12):1188–1197, 2009.

[51] Ayoung Kim and Ryan M Eustice. Active visual slam for robotic area coverage: Theory and experiment. *The International Journal of Robotics Research*, 34(4-5):457–475, 2015.

[52] David G Vutetakis and Jing Xiao. An autonomous loop-closure approach for simultaneous exploration and coverage of unknown infrastructure using mavs. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2988–2994. IEEE, 2019.

[53] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997.

[54] Brian Yamauchi, Alan Schultz, and William Adams. Mobile robot exploration and map-building with continuous localization. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3715–3720. IEEE, 1998.

[55] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Robotics: Science and Systems*, volume 2, pages 65–72, 2005.

[56] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3-4):181–198, 2003.

[57] Paul Newman, Michael Bosse, and John Leonard. Autonomous feature-based exploration. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 1234–1240. IEEE, 2003.

[58] Markus Achtelik, Abraham Bachrach, Ruijie He, Samuel Prentice, and Nicholas Roy. Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments. In *First Symposium on Indoor Flight*. Citeseer, 2009.

[59] Chintasid Pravitra, Girish Chowdhary, and Eric Johnson. A compact exploration strategy for indoor flight vehicles. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 3572–3577. IEEE, 2011.

[60] Luigi Freda and Giuseppe Oriolo. Frontier-based probabilistic strategies for sensor-based exploration. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3881–3887. IEEE, 2005.

[61] L. Heng, A. Gotovos, A. Krause, and M. Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. *ICRA*, 3(2):3–5, 2015.

[62] Shaojie Shen, Nathan Michael, and Vijay Kumar. Stochastic differential equation-based exploration algorithm for autonomous indoor 3d exploration with a micro-aerial vehicle. *The International Journal of Robotics Research*, 31(12):1431–1444, 2012.

[63] Cl Connolly. The determination of next best views. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 432–435. IEEE, 1985.

[64] William R Scott, Gerhard Roth, and Jean-François Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Computing Surveys (CSUR)*, 35(1):64–96, 2003.

[65] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Robotics: Science and Systems*, volume 2, pages 65–72, 2005.

[66] Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1071–1078. IEEE, 2015.

[67] Evan Kaufman, Taeyoung Lee, and Zhuming Ai. Autonomous exploration by expected information gain from probabilistic occupancy grid mapping. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 246–251. IEEE, 2016.

[68] Soohwan Song and Sungho Jo. Surface-based exploration for autonomous 3d modeling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[69] Yuki Okada and Jun Miura. Exploration and observation planning for 3d indoor mapping. In *2015 IEEE/SICE International Symposium on System Integration (SII)*, pages 599–604. IEEE, 2015.

[70] Tung Dang, Frank Mascarich, Shehryar Khattak, Christos Papachristos, and Kostas Alexis. Graph-based path planning for autonomous robotic exploration in subterranean environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3105–3112. IEEE, 2019.

[71] Yves Kompis, Luca Bartolomei, Ruben Mascaro, Lucas Teixeira, and Margarita Chli. Informed sampling exploration path planner for 3d reconstruction of large scenes. *IEEE Robotics and Automation Letters*, 6(4):7893–7900, 2021.

[72] Mitch Bryson and Salah Sukkarieh. An information-theoretic approach to autonomous navigation and guidance of an uninhabited aerial vehicle in unknown environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3770–3775. IEEE, 2005.

[73] Luke Yoder and Sebastian Scherer. Autonomous exploration for infrastructure modeling with a micro aerial vehicle. In *Field and service robotics*, pages 427–440. Springer, 2016.

[74] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon "next-best-view" planner for 3d exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468, 2016.

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

[76] Christos Papachristos, Shehryar Khattak, and Kostas Alexis. Uncertainty-aware receding horizon exploration and mapping using aerial robots. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 4568–4575. IEEE, 2017.

[77] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon" next-best-view" planner for 3d exploration. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1462–1468. IEEE, 2016.

[78] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, 42:291–306, 2018.

[79] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. Efficient autonomous exploration planning of large-scale 3-d environments. *IEEE Robotics and Automation Letters*, 4(2):1699–1706, 2019.

[80] Anna Dai, Sotiris Papatheodorou, Nils Funk, Dimos Tzoumanikas, and Stefan Leutenegger. Fast frontier-based information-driven autonomous exploration with an mav. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 9570–9576. IEEE, 2020.

[81] Titus Cieslewski, Elia Kaufmann, and Davide Scaramuzza. Rapid exploration with multi-rotors: A frontier selection method for high speed flight. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2135–2142. IEEE, 2017.

[82] Christian Witting, Marius Fehr, Rik Bähnemann, Helen Oleynikova, and Roland Siegwart. History-aware autonomous exploration in confined environments using mavs. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[83] Zhefan Xu, Di Deng, and Kenji Shimada. Autonomous uav exploration of dynamic environments via incremental sampling and probabilistic roadmap. *IEEE Robotics and Automation Letters*, 6(2):2729–2736, 2021.

[84] David Silver, Dave Ferguson, Aaron Morris, and Scott Thayer. Topological exploration of subterranean environments. *Journal of Field Robotics*, 23(6-7):395–415, 2006.

[85] Fan Yang, Dung-Han Lee, John Keller, and Sebastian Scherer. Graph-based topological exploration planning in large-scale 3d environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12730–12736. IEEE, 2021.

[86] Leonardo Fermin-Leon, José Neira, and José A Castellanos. Incremental contour-based topological segmentation for robot exploration. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561. IEEE, 2017.

[87] Emanuele Palazzolo and Cyrill Stachniss. Effective exploration for mavs based on the expected information gain. *Drones*, 2(1):9, 2018.

[88] Zehui Meng, Hao Sun, Hailong Qin, Ziyue Chen, Cihang Zhou, and Marcelo H Ang. Intelligent robotic system for autonomous exploration and active slam in unknown environments. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 651–656. IEEE, 2017.

[89] Soohwan Song, Daekyum Kim, and Sungho Jo. Online coverage and inspection planning for 3D modeling. *Autonomous Robots*, 44(8):1431–1450, aug 2020.

[90] Andreas Bircher, Mina Kamel, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. *Autonomous Robots*, 40(6):1059–1078, 2016.

[91] Zhexiong Shang, Justin Bradley, and Zhigang Shen. A co-optimal coverage path

planning method for aerial scanning of complex structures. *Expert Systems with Applications*, 158:113535, 2020.

[92] Kai Olav Ellefsen, Herman Augusto Lepikson, and Jan C Albiez. Multiobjective coverage path planning: Enabling automated inspection of complex, real-world structures. *Applied Soft Computing*, 61:264–282, 2017.

[93] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373. IEEE, 2017.

[94] David Vutetakis and Jing Xiao. Active perception network for non-myopic online exploration and visual surface coverage. *arXiv preprint arXiv:2309.11695*, 2023.

[95] Martin Ester, Hans-Peter Kriegel, J. Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *kdd*, 96(34):226–231, 1996.

[96] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[97] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[98] Padmavathi Kora and Priyanka Yadlapalli. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162(10), 2017.

[99] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[100] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[101] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors-a modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[102] David Kortenkamp, Reid Simmons, and Davide Brugali. Robotic systems architectures and programming. *Springer handbook of robotics*, pages 283–306, 2016.

[103] John Harwell and Maria Gini. Sierra: A modular framework for accelerating research and improving reproducibility. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9111–9117. IEEE, 2023.

[104] Odd Erik Gundersen, Saeid Shamsaliei, and Richard Juul Isdahl. Do machine learning platforms provide out-of-the-box reproducibility? *Future Generation Computer Systems*, 126:34–47, 2022.

[105] Alejandro Bellogín and Alan Said. Improving accountability in recommender systems research through reproducibility. *User Modeling and User-Adapted Interaction*, 31:941–977, 2021.

[106] Girish Suryanarayana, Ganesh Samarthyam, and Tushar Sharma. *Refactoring for software design smells: managing technical debt*. Morgan Kaufmann, 2014.

[107] Robert C Martin. Design principles and design patterns. *Object Mentor*, 1(34):597, 2000.

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

[109] The mobile robot programming toolkit. `https://docs.mrpt.org/reference/latest/`.

[110] William D Smart. Writing code in the field: Implications for robot software development. *Software Engineering for Experimental Robotics*, pages 93–105, 2007.

[111] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[112] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.

[113] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[114] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[115] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam. *arXiv preprint arXiv:2007.11898*, 2020.

[116] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9, 2014.

[117] Lukas Schmid, Michael Pantic, Raghav Khanna, Lionel Ott, Roland Siegwart, and Juan Nieto. An efficient sampling-based method for online informative path planning in unknown environments. *IEEE Robotics and Automation Letters*, 5(2):1500–1507, 2020.

[118] Lukas Schmid, Victor Reijgwart, Lionel Ott, Juan Nieto, Roland Siegwart, and Cesar Cadena. A unified approach for autonomous volumetric exploration of large scale environments under severe odometry drift. *IEEE Robotics and Automation Letters*, 6(3):4504–4511, 2021.

[119] Tung Dang, Marco Tranzatto, Shehryar Khattak, Frank Mascarich, Kostas Alexis, and Marco Hutter. Graph-based subterranean exploration path planning using aerial and legged robots. *Journal of Field Robotics*, 37(8):1363–1388, 2020.

[120] Mihir Kulkarni, Mihir Dharmadhikari, Marco Tranzatto, Samuel Zimmermann, Victor Reijgwart, Paolo De Petris, Huan Nguyen, Nikhil Khedekar, Christos Papachristos, Lionel Ott, et al. Autonomous teamed exploration of subterranean environments using legged and aerial robots. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3306–3313. IEEE, 2022.

[121] Boyu Zhou, Yichen Zhang, Xinyi Chen, and Shaojie Shen. Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters*, 6(2):779–786, 2021.

[122] Corrado Böhm and Giuseppe Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5):366–371, 1966.

[123] Qt. `https://www.qt.io`, 2023.

[124] GTK. `https://www.gtk.org`, 2023.

[125] Arash Mohammadi, Houshyar Asadi, Shady Mohamed, Kyle Nelson, and Saeid Nahavandi. Openga, a c++ genetic algorithm library. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2051–2056. IEEE, 2017.

[126] David Vutetakis. `https://github.com/dvutetakis`, 2023.