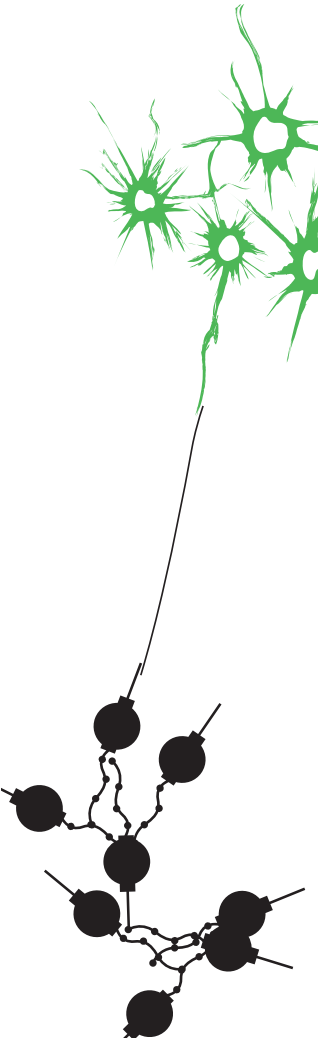


RAM ROBOTICS AND MECHATRONICS

A SPATIAL COGNITIVE EXPLORATION ALGORITHM FOR AUTONOMOUS MAPPING OF UNKNOWN INDOOR ENVIRONMENTS

A. (Atul) Hari

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
dr. B. Sirmaçek
N. Botteghi, MSc
dr. ir. A.Q.L. Keemink

February, 2020

006RaM2020
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

Autonomous exploration has gained remarkable attention among robotics researchers because of its application in automating various tasks such as search and rescue, mapping of underground tunnels, and space exploration.
Similarly, exploration can be used to automate the process of indoor mapping, which includes three-dimensional reconstruction of the houses and virtual reality imaging of each room. However, traditional exploration algorithms based on 2D LIDAR do not explicitly determine locations within an environment that is suitable for scanning the surrounding.
Inspired by humans, A novel algorithm capable of modeling spatial relation of points in a two-dimensional plane to its boundaries is developed in this research. This design models the boundaries by a polygon map approximated from the two-dimensional point cloud created by a laser range scanner. Then it relates the polygon to the points inside it, to develop a two-dimensional function representing the visibility of the environment. To develop this algorithm, a Gaussian process model is actively trained with the spatial relation between the polygon map and the points within. The trained model is then used to detect the points to visit while performing mapping. To develop an exploration framework around point detection, the research proposes two approaches. The first hypothesis, a novel exploration strategy by remembering the past visited regions, and the second adapts a traditional approach of exploration by locating the unknown regions. Further, for optimizing the sequence of visiting these points, a traveling salesman framework, is used along with an ant colony optimization algorithm.
On verification, it was seen that the algorithm was capable of predicting the points to visit in an unknown indoor environment with an average prediction error of 1m, using both the proposed methods. Further, on comparing the algorithm with traditional approaches, an equivalent performance was observed by the proposed exploration framework that maintains a memory. However, both the algorithms were capable of exploring 90% of the area on an average in the five test environment.

Contents

List of acronyms

LIDAR	Light detection and ranging
VR	Virtual reality
SLAM	Simultaneous localization and mapping
NBS	Next best scan
OG	Occupancy grid
GP	Gaussian processes
RRT	Rapidly exploring random trees
EKF	Extended Kalman filter
AR	Augmented reality
BO	Bayesian optimization
ICP	Iterative closest point
SCEAM	Spatial cognitive exploration and mapping
iSCEAM	Informative spatial cognitive exploration and mapping
VM	Visibility mesh
IM	Informative mesh
FOV	Field of view
RBF	Radial basis function
WFD	Wavefront frontier detection
FFD	Fast frontier detection
TSP	Travelling salesman problem
ACO	Ant colony optimization
GPU	Graphics processing unit
ROS	Robotic operating system

1 Introduction

1.1 Context and problem description

Maps have been one of the greatest human inventions for centuries. Earlier, people explored the world and represented their understanding in the form of maps, which allowed people to plan and navigate to unknown places. Nowadays, there are maps developed by Google that even allows the user to experience a 360-degree 3D view of places on a cellphone. Google maps are manually built by driving a mapping vehicle equipped with cameras and LIDAR (Light detection and ranging), and soon self-driving cars will be used to update the map regularly.
Similarly, to navigate in an indoor environment having access to digitized maps is beneficial. Current indoor mapping methods can successfully generate precise 3D maps, which are suitable for indoor navigation.
From the last couple of years, the real-estate industry has been reconstructing the 3D models of houses and providing users with virtual reality (VR) tours of the properties listed. In application, these maps are constructed manually by an operator using user-mounted or user-driven devices equipped with cameras and (or) LIDAR.
While mapping an unknown environment, the operator determines the points at which the scanning should be performed and plans a sequence to visit them. This process by which the operator determines the scanning-points and plans the path is called *exploration*. An exploration process is completed when the operator visits all the points and scans the entire house. However, this process is tiresome for an operator, especially with an increase in the size of the houses. Moreover, when there is no complete visualization available during mapping, the operators are more prone to miss certain regions to be mapped. Hence, this research aims to develop an exploration algorithm by which the process of indoor mapping can be done autonomously without a human operator. Furthermore, it is believed that automating the process of mapping could be convenient and scalable with the increasing demand for digitized maps.
The algorithm will be developed using measurements from a 360-degree 2D LIDAR made by a mobile robot. Further, the mapping devices can be mounted on the robot to construct high-definition maps. However, this work covers the development of the algorithm to generate mapping decisions and not map construction and accuracy.

To perform autonomous indoor exploration, first, the robot has to determine its location and create a map[†] of the environment using its onboard 360-degree 2D LIDAR. This process is called simultaneous localization and mapping (SLAM) (Thrun et al.,2005, Mahrami et al.,2013, Cadena et al.,2016). Further, the viewpoints[‡] at which the scanning has to be performed should be detected, and an optimal sequence needs to be planned to visit the points.
However, the following are the challenges to detect the points and to find the sequence.

- *Comprehending geometric structure*: Given an area visible to the sensor, locating the visually centered region, which contains the candidate viewpoints.
- *Identification of viewpoints in an environment that is unknown*: The robot has to detect viewpoints and make decisions based on partial maps generated by the SLAM algorithm.
- *Selecting the sequence of exploration action*: Given a set of viewpoints, it is challenging to determine a sequence by which they should be visited to generate a globally optimal trajectory that reduces travel distance during mapping an unknown environment.

[†]This is a 2D map used by the algorithm internally for planning and is not related to 3D maps discussed earlier.
[‡]Viewpoints are indoor locations that provide an excellent 360-degree view of the surroundings

However, several existing algorithms can be used to explore indoor environments autonomously.

1.2 Related work

In this section, previous research aiming to achieve indoor exploration using different map representations is described.

Autonomous exploration problems have been studied for more than 20 years from now. Yamauchi (1997) introduced a method that could explore a generic 2D environment. In his algorithm, the *frontiers* are considered as *the separation between the known and unknown regions*. The exploration was achieved by sequentially navigating to the nearest frontiers. Holz et al. (2010) used a similar technique, but repeatedly observed the frontiers during exploration, to update the decision when the area near the frontier is completely mapped.

Further, the concepts of frontier based exploration were adopted for autonomous 3D structure reconstruction by Prieto et al. (2017), Kurazume et al. (2017), and Meng et al. (2017). Prieto et al. (2017), developed the next Best Scan (NBS) algorithm for automatic 3D scanning of furnished buildings by exploring each room separately and then combining the 3D models. However, the frontier-based algorithms do not account for the quality of data acquired, and hence the scanning performed can be close to obstacles. To detect the regions with higher visibility Iocchi and Pellegrini (2009) and Prieto et al. (2017) used stereo vision to identify the structural elements like doors, walls, roof, stairs, etc. These semantic structural elements were used to plan the scanning order. Most of the researches use both 2D and 3D range finders to explore a building and perform scanning. However, the selection of sensors is based on the specific purpose of each research. Along with 3D LIDAR, Borrmann et al. (2014) used additional temperature sensors to develop a thermal model of indoor environments and perform exploration to inspect temperature distribution systems. Considering indoor mapping, scans must be taken from points with higher visibility of the surrounding. Kim et al. (2018) developed an algorithm to scan indoor sites by identifying points on the planned trajectory that maximize the visibility of the surroundings. Even though Kim's approach succeeded in performing scans at locations with higher visibility, they used 2D, 3D LIDAR, and camera sensors.

A different approach to formulating the exploration problem was using an *information-theoretic* framework. The concept of *entropy*, first coined by Shannon (1948) is used to quantify the information available on the map. The uncertainty of the map signifies the unavailability of information. The robot performs exploration by visiting the uncertain regions of the map using 2D LIDAR, in an information-theoretic exploration model pioneered by Bourgault et al. (2002). Further, Stachniss et al. (2005) used Rao-Blackwellized particle filter (Murphy, 2000) to increase the accuracy of mapping. Charrow et al. (2015) developed 3D reconstruction of long corridors by finding trajectories that maximize information-theoretic objective. Exploration algorithms considering 2D measurements mostly solve the problem considering an occupancy grid (OG) map (Elfes, 1989, Thrun et al.[p. 281-308], 2005), which is a discrete map representation made of independent grid cells. The OG map provide occupancy information of individual cells and do not contain any collective structural information about the building.

A continuous occupancy map representation, which, unlike the OG map, spatially relates neighboring points to a continuous surface occupancy function. A predominant method used for continuous map representation is Gaussian processes (GP). Yang et al., (2013) used rapidly exploring random tree (RRT) to sample paths for continuous GP map. Jadidi et al., (2014), Jadidi et al., (2016) constructed a GP map by performing regression on occupancy probabilities obtained from sparse sensor measurement. The informative areas of GP map were used for greedy exploration that maximizes information gained. The research done on indoor exploration for mapping along with the approach, sensors and environmental representations utilized is visualized in Figure 1.1.

Researcher	Year	Approach						Sensors				Environment representation					Purpose
		Frontier based	Information theortic	RRT	3D voxel	3D Volumetric	Maximizing visibility	2D LIDAR	3D LIDAR	Camera	Others	OG map	Point cloud 3D	GP map	Polygon map	Semantic map	
Yamauchi	1998																Generic 2D exploration
Bourgault	2002																Acurate 2D mapping exploration
Stachniss	2005																Accurate 2D mapping exploration
Iocchi	2009																Generalization of visually realistic 3D maps
Holz	2010																Room-wise indoor exploration
Yang	2013																UAV exploration of indoor cluttered env
Borrmann	2014										Thermal sensors						Thermal model for indoor inspection
Charrow	2015																3D maping for construction, mining and search and rescue operations
Carrillo	2016																Generic 2D exploration
Jadidi	2014, 2016																Generic indoor exploration and mapping
Prieto	2017																Scanning of high oclussion indoor env
Meng	2017																UAV exploration for indoor 3D reconstruction (office corridor env)
Kurazume	2017																Multi-robot exploration for large scale 3D reconstrution
Kim	2018						Along the trajectory										Fast and optimized 3D reconstruction for construction industry (costruction management)
Selin	2019																UAV search and rescue in large indoor environment
Niroui	2019																Search and rescue exploration in cluttered indoor environment
This research	2019						Entire floor surface										Scanning indoor environment for 360-degree VR imaging and 3D reconstruction
This research	2019						Entire floor surface										Scanning indoor environment for 360-degree VR imaging and 3D reconstruction

LEGEND

Real-estate industry	Construction industry (cluttered env)	UAV based exploration	General 2D exploration	Indoor thermal inspection	Multi-robot exploration

Figure 1.1: Related research on autonomous indoor exploration.

The traditional 2D exploration approaches do not define behavior that explicitly identifies the location to be scanned during mapping. State of the art algorithms for autonomous indoor scanning uses 3D point cloud matching and semantic structural element modeling along with 2D measurement to detect points at which scanning has to be performed, which makes it expensive. The concept of maximizing visibility used by Kim et al. (2018) is suitable for selecting scan points effectively. However, he found visible regions only along the planned trajectories. Hence, this research will focus on developing an exploration algorithm that models the visibility of the entire explorable regions by using only a 2D LIDAR.

1.3 Research questions

Considering the problem description and various related work, the research can be formulated into three questions:

1. How to develop an algorithm to detect viewpoint within the rooms using geometric relationships, considering a complete map of the house?
2. Given a partial map from the SLAM algorithm, how accurately can viewpoint be predicted during the exploration?
3. How does optimizing the route between the viewpoints affect exploration compared to heuristics used by baseline methods?

1.4 Assumptions

In an approach to solve the problem, five assumptions were made regarding the environment and the robot's abilities.

1. *Zero elevation*: The exploration is assumed to be performed on a plane without inclination. Hence any inclination above the ground is considered as obstacles.
2. *Sensors*: The robot is supposed to be equipped with a 360-degree, 2D Light detection and ranging (LIDAR) sensor along with dead-reckoning.
3. *Static obstacles*: The environment is simplified to a case with objects that remain stationary throughout the exploration.
4. *Concave polygon map*: The house like environments considered as the use-case for the research is assumed to have a floor plan such that its inner perimeter forms a closed concave polygon.
5. *Discrete goals to navigate*: Discrete goals are selected and the shortest path is used to navigate between the selected locations. The action execution package in Robotic Operating System (ROS)* called `move_base`** is used to provide control input.

1.5 Approach

An exploration algorithm suitable to develop a digitized map autonomously should necessarily detect viewpoints in every room of a house. Then a sequence to visit them should be planned to explore further using the local observations (map of the region that was explored).
In this thesis, to detect the viewpoints, the 2D LIDAR measurements are used to approximate a concave polygon boundary model, which will be used to develop a novel surface model that defines the visibility of the surrounding at every point in the environment. To develop this

*ROS https://www.ros.org/.
**`move_base` http://wiki.ros.org/move_base.

model, the structural arrangement of the environment should be comprehended. An operator performing the mapping, on a high level, uses his spatial cognition to identify the viewpoints. Spatial cognition (Colby,2009) is concerned with acquiring knowledge about the spatial arrangement of objects in the surrounding. Inspired by this ability, a model will be developed to detect points that maximize the visibility of the surrounding, considering the polygon map of the building. This model is obtained by performing non-linear regression using Gaussian processes.
Further, to get the sequence by which the points has to be visited, two exploration frameworks will be constructed. The first framework will develop a novel approach of exploration, which only maximizes visibility, and the other will additionally adopt traditional concepts from information-based exploration. The decisions made on the sequence of visiting the viewpoints will aim to optimize complete travel during mapping by formulating a traveling-salesman problem (TSP) framework. Figure 1.2 shows a flowchart describing the approach that will be developed. After developing the algorithm, the viewpoint detection will be studied on a full

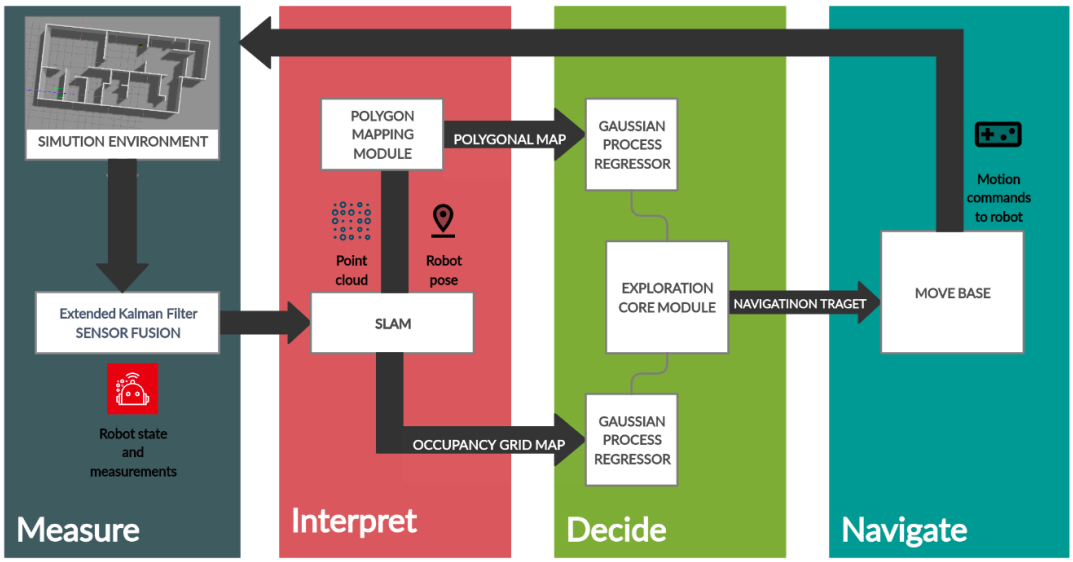


Figure 1.2: Flowchart describing the algorithm framework

map of the house, and it will be extended to a case while only the local (partial) map is available. The exploration performed by the algorithm will be compared to the traditional algorithms to evaluate the accomplishment.

1.6 Report organization

The rest of the chapters in the thesis are organized as follows. The background information on the concepts of SLAM, with detailing on different map representations and concepts of exploration, are introduced in Chapter 2.
The analysis and design of the proposed algorithm are organized in Chapter 3, which includes the design of a Gaussian process model to detect viewpoints and design of the exploration framework. Further, the planning module to determine the sequence of visiting the viewpoints is also designed in the chapter.
The experiments designed for validating the designed algorithm is described in Chapter 4. Further, various parameters involved in the development of the proposed algorithm are also described.
After conducting the experiments, the results obtained are shown in Chapter 5.
Finally, the conclusions drawn from the research and ideas for possible future directions are presented in Chapter 6.

2 Background

Developing a system that can autonomously navigate in an unknown environment requires ability to localize itself and construct a map of the surrounding. In this chapter, an overview of how the robot perceives the environment and estimates its position relative to its surroundings is provided. Further, discussion on discrete and continuous map representations are used to explain the classical exploration approaches.

2.1 State estimation

Before performing exploration, the robot should acquire information about its state. The state of a robot describes its motion in time using a set of quantities such as position, orientation, and velocity. In general, the robot uses sensors to perceive the environment and estimates its state from the noisy measurements. However, due to the noise, the states are expressed in terms of its probability called belief. The estimation of the robot's belief can be obtained using a *Bayes filter* algorithm. The belief *bel*, is the probability about the current state x_t, of the robot given the current sensor measurements z_t. Further, using the prior belief x_{t-1}, the sensor measurement z_t and the control action u_t the Bayes filter updates the belief of the current state recursively as given in Equation 2.1.

$$bel(x_t) = \eta P(z_t|x_t) \int P(x_t|u_t - 1) bel(x_{t-1}) dx_{t-1} \tag{2.1}$$

An important requirement for such an estimation is a *motion model* and a *measurement model*. The motion model defines the motion control actions performed by the robot. These actions are used as input to the Markovian model to predict the current states given the prior belief and measurement likelihood. Now considering the exploration problem description in Section 1.1, the robot needs to identify its location in the environment to plan the control input. Hence, the robot uses motion made to reach the current position and the measurements made from the current location to estimate its current position.

2.1.1 Motion model

The motion model uses the translation and angular velocity by which the robot should move and maps them into the velocity of each wheel using the robot's design parameters like the wheel dimension and wheel separation (Ben-Ari and Mondada, 2018). Considering the differential drive robot used in this research, the kinematic model of the robot's motion can be used to relate the robot's translation (v) and rotational (ω) velocity to individual wheel velocities v_r and v_l for right and left wheels respectively using Equation 2.2.

$$\begin{aligned} v_r &= \frac{2v + \omega L}{2R} \\ v_l &= \frac{2v - \omega L}{2R} \end{aligned} \tag{2.2}$$

Where R is the wheel radius, and L is the wheelbase. The velocity of individual wheels allows computing the position of the robot from the kinematic model. However, due to noise, the position computed using the motion model needs to be corrected using the measurements to estimate the position of the robot better.

2.1.2 Measurement model

To obtain the measurement, the robot considers its on-board sensors. The wheel odometry data of the robot is acquired using a dead reckoning model. The dead reckoning model incrementally evaluates the measurement and is used as the primary global measurement source.

However, to perform better state estimation, a range sensor has to be used, which in this research is a LIDAR (light detection and ranging). A LIDAR sensor transmits an array of laser beams and measures the echoed rays. Even though the LIDAR sensor provides sufficiently accurate measurement, it is difficult to develop an accurate model because of the uncertainties in the internal physical system. This restricts modeling of the sensor as a deterministic system defined by $z_t = f(x_t)$. To take the uncertainty into account, a probabilistic model is considered by applying conditional probability. This defines measurement by $p(z_t|x_t)$. The measurement probability is the product of the likelihood of each beam, given in Equation 2.3.

$$p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t,) \tag{2.3}$$

Where, $p(z_t|x_t, m)$ is the conditional probability of each z_t given x_t and *map* m, where map is considered as an instance of the environmental model. To obtain the probability of measurement, various models can be constructed by evaluating the log-likelihood.

2.1.3 Environmental model

In the measurement model, the probability is evaluated on the current robot state x_t and the *map* m, where a map is an instance of the environmental representation. A map is generated using the measurement data obtained from the sensors and contains the information about the occupancy of observable subspace. The robot has to build a map of the surrounding and localize itself using the same measurements.

2.2 Simultaneous localization and mapping

SLAM could be framed as a scenario in which the robot is uncertain about both its position and the environment. The task of the robot is to simultaneously create a map representation of the environment using its measurement model and determine its states based on the observations. This can be stated as a joint-estimation problem. The Simultaneous localization and mapping (SLAM) problem can be visualized from the Figure 2.1. The figure represents the robot state x at various instances of its motion denoted by k and various observations z, of the environment. This demonstration is a rather simple case of the SLAM representation that considers landmarks m as map instances. The drift observed in the estimate of the robot states is due to the accumulated measurement error called dead-reckoning.

2.2.1 SLAM formulation

From the SLAM problem observed previously, a mathematical model can be obtained. The model has the control input u_t and measurement z_t as the initial input and an estimation of the environmental map m and robot position x_t needs to be made. The problem can now be represented as in Equation 2.4:

$$p(x_{0:T}, m|z_{1:T}, u_{1:T}) \tag{2.4}$$

Where the equation is represented in discrete-time instances ranging from 0 to T. This is a general model to estimate the map and pose of the robot over the entire trajectory, generally called *full SLAM*. However, a more interesting expression would be to consider only the current estimate of the robot and map. This is known as *online SLAM* and can be formulated, as shown in Equation 2.5.

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{2.5}$$

where, the Equation 2.6 can be obtained by eliminating past states from Equation 2.4 by integration.

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int_{x_0} ... \int_{x_{t-1}} p(x_{0:t}, m|z_{1:t}, u_{1:t}) dx_{t-1} ... dx_0 \tag{2.6}$$

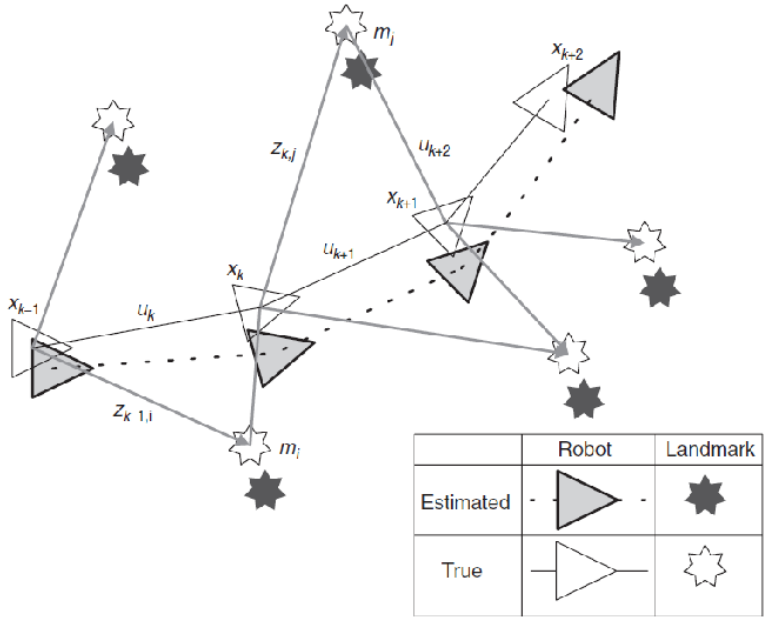


Figure 2.1: Demonstration of the SLAM problem (Khairuddin et al., 2015).

The past states are eliminated to make the solution computationally tractable. There are several techniques to solve these equations, depending on the applications. In work by Cadena et al. (2016) various SLAM approaches and some open problems are discussed.

2.3 Mapping

The exploration problems consider that prior knowledge about the map representation is not available. To obtain knowledge about the map, the robot has to gather information from the surrounding. However, it is not possible to predetermine the complete sequence of control inputs to navigate. Therefore the inputs required for navigation is decided based on the observations made by the robot.
The measurement data acquired by the robot could be used to create an environmental model. This process of developing such a model from the available sensor data is called *mapping*, and the resultant representation is called a *map*. The map is an occupancy likelihood representation of the environment and can be a discrete or continuous model.

2.3.1 Discrete map

A discrete map representation is not defined at all points in the exploration space. However, they represent distinct occupancy information in observed regions of the environment. Two common approaches to model using discrete maps are using landmarks and an occupancy grid map or OG map (Elfes, 1989).

landmark-based map

A landmark-based map in generally are used by *extended Kalman filter(EKF)-based SLAM* (Dissanayake et al., 2001). The landmark-based points identify distinct features from the environment. The robot uses the location of features to estimate its state. A continuous correlation between the landmarks and the pose can lead to a complete estimation of the trajectory. However, there will still be noise which could be filtered using popular methods like EKF. Figure 2.2, visualizes an implementation of landmark-based method by Newman et al. (2002). He conducted an experiment called *explore and return*. Where the robot was teleoperated to explore

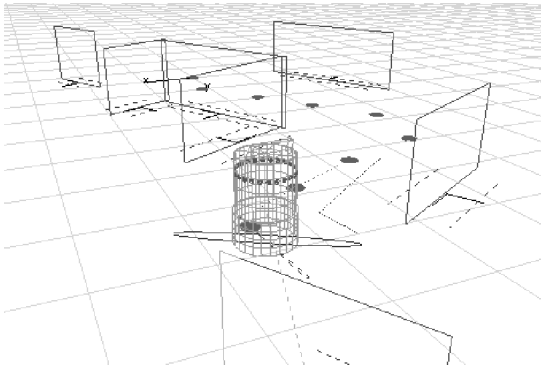


Figure 2.2: A feature-based SLAM, using concurrent mapping and localization algorithm by Newman et al. (2002). Grey dots: saved position, obstacle planes: observed landmark.

and save the landmark as feature planes and then return to the home autonomously assisted by the landmarks.

Occupancy grid map

The occupancy grid map, initially introduced by Elfes (1989) is a standard grid, which represents the occupancy of a location in the environment. The occupancy of the environment is generally identified as *free*, *occupied* or *unknown* and are indicated using a *cell*, where cell is an unit grid. Considering a range measurement sensor (LIDAR), the free cells are identified as those which allow the laser to pass through. An occupied cell is identified by the cell, which undergoes multiple hits from the laser. The occupied cells represent the obstacles or hindrances in the environment. The unknown cells are unobserved or unexplored; hence, no conclusion about the occupancy can be made with the available observation. A visualization of an occupancy grid map is visualized in Figure 2.4.

Considering the map m, which is a collection of cells identified by its location coordinates.

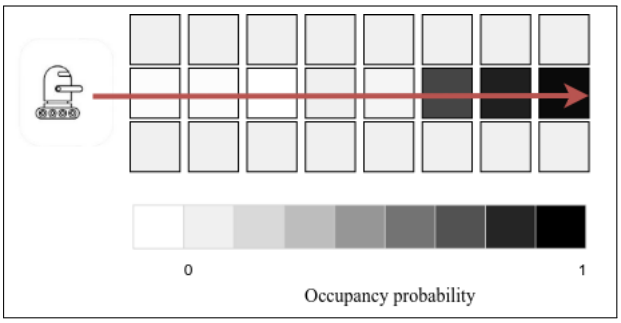


Figure 2.3: Robot's perception represented on a 2D occupancy grid. The red arrow indicates the laser beam.

Every cell in the map is identified as 1, for a *hit* of the laser and 0, for *no occlusion*. Based on the measurement the cells are classified as occupied, unknown and free. considering the probabilities, the probability of free cell will always be less than one and hence the total probability of the cell converges to zero. To avoid this, the log probabilities are considered. Although the cell values are obtained, the evaluation is not completely certain. Hence, all the measurements made by a particular cell are summed to obtain the confidence of occupancy. This generates a probabilistic model of the map shown in Figure 2.3.

Finally, the expected log-likelihood is maximized to obtain a map estimate. The work done by Thrun (2003) demonstrates the occupancy grid map formulation using a forward model. The classical inverse model-based method (Elfes, 1989) considers the occupancy of each grid inde-

pendently; hence, there can be conflict when measurements of the same cell are made from different locations. However, the forward model (Thrun, 2003) populates the log-probabilities to obtain cell values, which can result in consistent occupancy values. The OG map discussed in the coming chapters assumes a forward model. More detailed understanding can be obtained from the works by Elfes (1989) and Thrun (2003).

2.3.2 Continuous map

Continuous maps, as the name suggests, are defined for all points in the observable space. This also means that the map is not a representation of independent random variables, but are correlated to each other. The continuous occupancy probability distribution provided by these maps is more suitable for continuous control decisions. Two most popularly used continuous map representations are:

Gaussian process maps

The Gaussian process (GP) maps (Yang et al., 2013), (Jadidi et al., 2014), (Jadidi et al., 2016) in general are trained based on occupancy probabilities to generate a multi-variant Gaussian distribution. These distributions hugely benefit from the fact that the GP relates various cells in the observable space, used for training. This relation inherently gathers structural data about the occupancy. Figure 2.4 shows a GP representation of an environment. The GP maps can

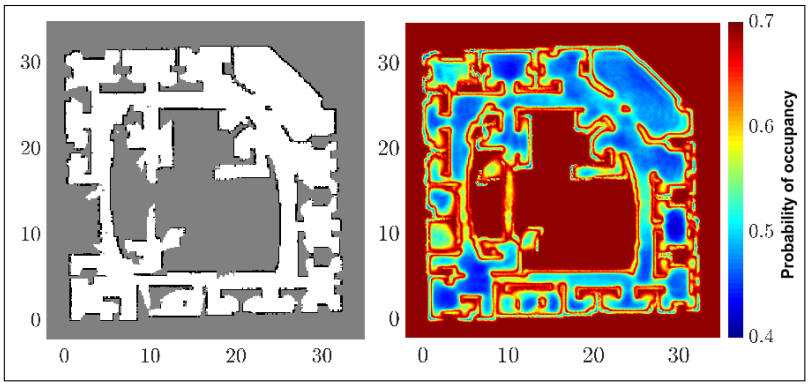


Figure 2.4: Left: Occupancy grid map representation using the Intel dataset. Right: Gaussian process map using the occupancy grid map as demonstrated by Jadidi et al. (2016)

account for sparse measurements or information since they approximate a continuous distribution over the available sparse data.

Hilbert maps

Hilbert map is a continuous map representation similar to GP maps, demonstrated by Ramos and Ott (2016), Francis et al. (2018). Given the occupancy at a point obtained using the laser measurement, a set of samples can be drawn from the observable region. Further, a conditional model $p(z|x,w)$ depending on a vector w can be learned using logistic regression classifier (LR) (Brzezinski and Knafl, 1999). The parameter w, is obtained by performing stochastic gradient descent (Kiefer and Wolfowitz, 1952). So the occupancy is visualized as projections of a linear discriminative model on a reproducing kernel Hilbert spaces (RKHS) (Shin and Lee, 2016). Similar to GP maps the Hilbert map also captures spatial correlations and show robustness towards outliers. A visualization of a Hilbert map is shown in Figure 2.5.

2.4 Exploration

In this section, the discussions made previously are used to define the exploration problem. The robot estimates its pose and builds the map using the SLAM algorithm discussed in Sec-

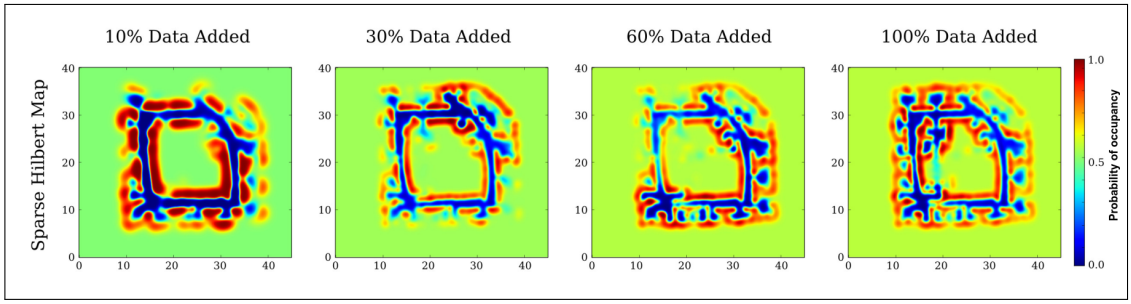


Figure 2.5: A sparse Hilbert map using the Intel dataset. The sharpness of the map increases with an increase in the number of samples as shown by Ramos and Ott (2016)

tion 2.2. Further, different types of map constructed from the robot's LIDAR data is discussed in Section 2.3. During exploration, the robot's observable region is restricted. The measurements obtained from the sensors along with observable environment representation is used to decide the control inputs. Making the robot autonomously navigate.
The classic approaches for exploration in a generic 2D environment are frontier based and information-theoretic and serve as the foundation for more complex algorithms. The exploration problem is hard to make optimal navigation decisions. The classic approaches make decisions based on simple heuristics that tend to maximize information about the robot's environment. Hence the robot decides to navigate to the uncertain regions of the environment.

2.4.1 Frontier based approach

Yamauchi (1997) ideated that:*"To gain the newest information about the world, move to the boundary between open space and uncharted territory."* According to Yamauchi, *frontiers* are these boundaries. Considering an occupancy grid framework, these frontiers are represented as the boundary between the free and unknown areas, as shown in Figure 2.6.

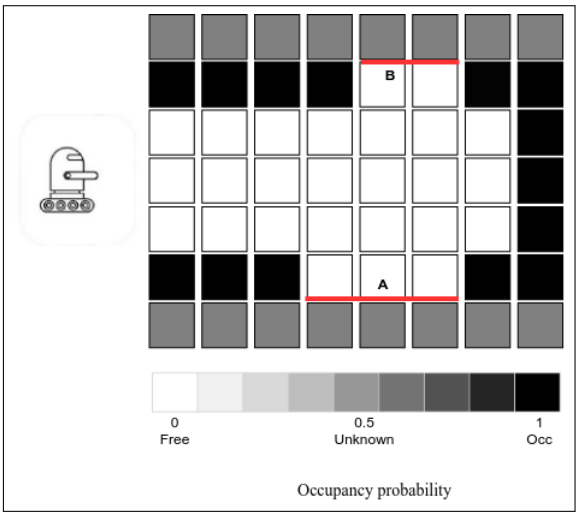


Figure 2.6: An occupancy grid map with free cells (white), occupied (black), unknown (grey), and frontier (red).

Nearest frontier based exploration

The nearest frontier exploration proposed by Yamauchi identifies the frontiers and uses a distance-based heuristic to select the action to execute. Considering the illustration shown in Figure 2.6, suppose the frontier A is the nearest, then the robot sends a goal to either the

mid-point or the nearest cell of the frontier. This process is iterated until no more frontiers are available.

2.4.2 Information theoretic approach

According to Shannon (1948), *entropy* is defined as a measure of the average information of a cell, knowing its probability. A Bernoulli random number represents the occupancy probability of each grid cell. Now considering the map, highly informative regions are the space near the frontiers because of the uncertain occupancy probability at that point.
To explore the environment, the robot should maximize the information about its surroundings. The information gain in general can be calculated based on Shannon entropy given by Equation 2.7 or 2.8.

$$H_p(\mathrm{x}) = -\int p(\mathrm{x}) log\, p(\mathrm{x}) d\mathrm{x} \tag{2.7}$$

$$H_p(\mathrm{x}) = -\sum_{\mathrm{x}} p(\mathrm{x}) log\, p(\mathrm{x}) \tag{2.8}$$

The Equation 2.7 calculates entropy which represents uncertainty of the map. Considering a robot with pose x, the map (grid cell) m, z as the measurements and control input u, the SLAM posterior can be represented as in Equation 2.9,

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1,t}).p(m|x_{1:t}, z_{1:t}, t_{1:t}) \tag{2.9}$$

For readability considering $(x_{1:t}|z_{1:t}, u_{1,t})$ as x and $(m|x_{1:t}, z_{1:t}, t_{1:t})$ as m. The entropy of $p(x, m)$ can be written as in Equation 2.10:

$$H(p(x, m)) = H(p(x)) + \int_x p(x) H(p(m|x)).dx \tag{2.10}$$

The *information greedy* exploration approach selects frontiers, that if visited has the highest gain in information($I(\hat{z}, a_t)$). The information gain can be computed by measuring the difference between the entropy of the current state of the robot and the estimated entropy following an action a_t and obtaining the measurement $z_{t+\hat{1}:T}$, given by Equation 2.11.

$$I(\hat{z}, a_t) = H(p(m, x)) - H(p(m, x, \hat{x})) \tag{2.11}$$

The implementation of above-mentioned exploration approaches will be further evaluated.

2.5 Summarizing the concepts

In this chapter, the concept of SLAM and various map representation obtained using them are discussed. Further, the concept of exploration is introduced, and popular approaches are discussed.
Considering exploration in general, the robot has uncertainty about its position as well as a map. The robot makes a representation of the environment as a continuous or discrete function capturing its occupancy state. These representations are used to identify highly uncertain regions, i.e., regions with high entropy. The robot then executes an exploitative action towards the uncertain region that maximizes the knowledge about its environment. In the approach used by this research, a discrete occupancy representation, as well as a polygonal approximation of the visible region, are used to design the exploration algorithm.

3 Analysis and design of the proposed algorithm

The mapping of an unknown indoor environment is attempted to be automated using the algorithm proposed in this chapter. In the real-estate based applications, a visual walk through of the property is provided to a distant customer, using different environmental representations. These representations can be as simple as a 2D floorplan or detailed as 3D reconstruction or virtual-reality (VR) imaging. High-definition maps have also shown to be beneficial for augmented reality (AR) navigation and semantic representation. The appropriate location (viewpoints) for scanning is identified by the operator who walks around the house. Therefore, the detection of viewpoints is integrated with an exploration problem, in the process of mapping a house.
Autonomously detecting points suitable for scanning is not straight forward, and this becomes even more challenging in complex environments with higher occlusion. Previously, Prieto et al. (2017), Iocchi and Pellegrini (2009) have used methods like semantic segmentation to identify structural elements (SE) of a building like doors, windows, etc. However, such an approach requires multiple sensors and hence adds to the overall expense. An approach that can be used in 2D is *maximizing visibility* as used by Kim et al. (2018). However, Kim first planned a path to the detected frontiers and then evaluated every point in the trajectory to find points with high visibility. However, this method locates scan points on a planned trajectory only.
Therefore, in this research, a novel method that models the visibility at every point in the environment is designed. The complete algorithm for autonomously scanning the environment can be developed as three modules. The first for detecting the viewpoints (Section 3.1), the second module determines the regions that have to be explored (Section 3.2) and the third module makes a decision on which region has to be explored first (Section 3.3). The block diagram of the system illustrating the different modules is shown in Figure 3.1.

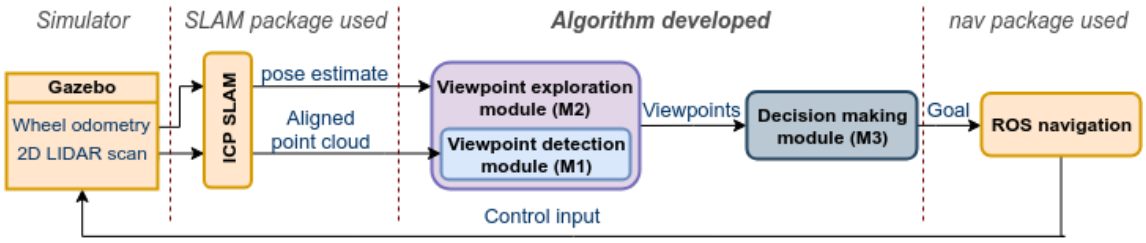


Figure 3.1: System block diagram.

The input and output data for each module is arranged along with the algorithm references in Table 3.1. Further, the design of each of the modules will be discussed in detail in the remaining parts of the chapter.

3.1 Detecting viewpoints

Viewpoints are regions with higher visibility of the surroundings and are suitable to perform scanning. However, to detect those points in 2D, a representation of the environment is generally modelled, which specifies the occupancy state based on the measurements made by the range-sensors. The occupancy grid (OG) map discussed in Section 2.3 is a grid-based map representation where each grid is an independent or unrelated cell. However, to achieve the primary intention of the research, it is essential to understand the spatial relation between free and occupied regions.
The *first research question* intends *"to locate the viewpoint within the rooms using geometric relationships, provided a complete house map"*. Hence for the detection of viewpoints, a new

Input data	Module	Output data	Reference	Algorithm
Wheel odometry, 2D LIDAR scan	**ICP SLAM**	Aligned 2D point cloud, OG map, Pose estimate	MRPT toolbox (Icp, 2013) Appendix C	(Segal et al., 2009)
Aligned 2D Point cloud	**Viewpoint detection module (M1)**	Viewpoints	Section 3.1	Algorithm 1,2,6
Aligned 2D Point cloud	Scan to polygon converter	Polygon map	Section: 3.1.1, Appendix B	Algorithm 6
Polygon map, Training samples	+ Visibility function + Construct GP + Perform regression	Visibility mesh (VM)	Section 3.1.2	
Visibility mesh	Detect viewpoints	Viewpoints	Section 3.1.2	Algorithm 2
Polygon map	**Viewpoint exploration module (M2)**	Viewpoints	Section 3.2	Algorithm 3, 4
Aligned point cloud viewpoints, pose estimate	Hypothesis 1: SCAEM algorithm **+ Viewpoint detection module** + Remove visited viewpoints	Viewpoints Unvisited viewpoints	Section 3.2.1	Algorithm 3
OG map VM,IM Combined mesh Viewpoints, pose estimate	Hypothesis 2: iSCEAM algorithm + Informative GP + Combine VM and IM + Detect viewpoints + Remove visited viewpoints	informative mesh (IM) Combined mesh Viewpoints Unvisited viewpoints	Section 3.2.1	Algorithm 4
Viewpoints	**Decision making module (M3)**	Navigation goal	Section 3.3	Algorithm 5
	+ Ant colony optimization		Section 3.3.1	Gupta (2019)
Navigation goal	**ROS navigation**	Velocity commands	Section 3.3.2	
	+ Move base			ROS `movebase`*

Table 3.1: Algorithm design overview

environmental representation is designed by developing a polygonal map and then for every point inside the polygon, the visibility is modelled geometrically using Gaussian process regression.

3.1.1 2D scan to polygon map

The 2D LIDAR sensor generates range measurements to the obstacle to obtain a 2D scan. The endpoints of each laser beam can be connected to form a closed concave polygon. Further, the polygons obtained for each measurement are combined to construct a polygon map. However, each scan of the 2D LIDAR used contains 720 points equally spread out in 360-degree, and hence it is not feasible to combine all the scans into one single map. Therefore, the scans should be first filtered and down-sampled such that the noisy scans points are removed, and structural features are intact.

The measurement Z made by the 2D LIDAR is a vector representing the distance of the obstacles from the sensor mounted on the robot. The range $\{z_1, z_2 ... z_n\} \in \mathbb{R}^n$ and orientation $\{\theta_1, \theta_2 ... \theta_n\} \in \mathbb{R}^{3n}$ of n laser beams is represented in the sensor frame, and the robot's position $(r_x, r_y, r_z) \in \mathbb{R}^{3n}$ is in the global frame M. Transformation, ${}^{B}_{S}T$ from sensor frame S to robot-based frame B, is known. The robot-base frame is defined at the midpoint of the line joining the wheels. The sensor frame is fixed at a point that emits the beams. The odometry measurement of the robot gives the transformation ${}^{M}_{B}T$ between the robot-base frame and the global frame. The n scan points can be expressed in the sensor frame S as a point cloud P given by the Equation 3.1,

$$
\begin{aligned}
{}^{S}P = \{p_1, p_2 ... p_n\} \in \mathbb{R}^{2n}, \quad p = (p_x, p_y)^T \\
\text{where, } p_{x_i} = z_i cos\theta_i, \;\; p_{y_i} = z_i sin\theta_i, \;\; i \in (0, n)
\end{aligned}
\tag{3.1}
$$

For a point cloud P_t obtained at time step t, a line connecting all these points p will result in a closed polygon $\bar{P}$. Since the polygon mapping algorithm is an additional contribution to the research the complete design of the algorithm including the point cloud filtering, down-sampling and method used for updating new measurements to the map will be described in Appendix B.

Representing the wall boundaries of a house in the form of a polygon makes it easier to evaluate the relationship between the polygon and the points enclosed by it using geometric techniques. The developed polygon map is a tool that will be used to comprehend spatial relations to detect the viewpoints.

3.1.2 Modelling visibility using Gaussian process regression

The points inside the polygon map carries the information regarding the spatial arrangement of structural elements of the building. Which is represented in terms of the surrounding visibility at the point. Hence, to extract this information, it is essential to formulate a mathematical model that can learn the relationship between the points contained by the polygon and its boundary.
To model the relationships, the first module of the proposed algorithm proceeds by designing a visibility function that models the spatial depth property, which is part of the *human cognitive system*. The minimum distance to an object from the point it is viewed describes the depth property. In a 2D space, the distance will be calculated along a plane parallel to the ground.
The depth property should be defined at every point inside the polygon for the detection of viewpoints in a given environment. However, it is not feasible to calculate the depth at every point. A better approach can be to observe how the property changes along the plane by understanding the relationship between the contained points. However, the depth varies non-linearly, considering the complexity of house-like environments. Hence, a non-linear regression is performed to learn the relation between the points. Therefore, a non-parametric Bayesian approach of regression called *Gaussian process regression* is used to model the visibility at every point in the environment.
Gaussian processes (GPs) are popularly considered as a powerful tool in machine learning (Rasmussen, 2004). GPs can be used to predict the value of a function given some prior training. Hence, it can capture the spatial data to predict a function by performing *regression*. Besides regression, GPs are used for clustering (Kim and Lee, 2007) and classification (Kapoor et al., 2010) tasks. While predicting a function from data, there are infinitely many possibilities. However, using a Gaussian process, this infinite number of solutions are probabilistically weighted to converge into a single prediction based on the training, using Bayes rule. Hence the visibility function is used to train a Gaussian process to predict the depth property at any point in the given environment.
Further, the design of the visibility function, training, and regression of GP are discussed in this section to develop the viewpoint detection module.
Note: It is important to note that the geometric center might be the same as the viewpoint considering a convex polygon, but for a concave polygon like in the map of a house, the geometric center can even be outside the polygon. Therefore, identifying the viewpoints is challenging using general geometric methods.

Design of the visibility function

The visibility function computes the surrounding visibility from a point inside the polygon. Formulating the function requires an introduction to two terms, the field of view (FOV), ϕ, and depth δ. The FOV at a point is the angular range of the sensor. In this case, it is a 360-degree LIDAR and is constant for every point. The depth $\delta \in \mathbb{R}$ is the minimum 2D LIDAR range measurement obtained at that point. A generalization of the function can be represented as the smallest circle with center $x_c \in \mathbb{R}^2$ contained by the polygon. This generalization describes depth as a property associated with the location x_c and hence captures the spatial element of the algorithm. The Figure 3.2 provides a visualization of generalized FOV and depth at a point x_c. In Figure 3.2, it can be observed that circles with the center near the middle region of the 'L' shape are bigger than the circles with the center closer to the walls and corners. Therefore, it is clear that there exists a continuous correlation between the points inside the polygon map

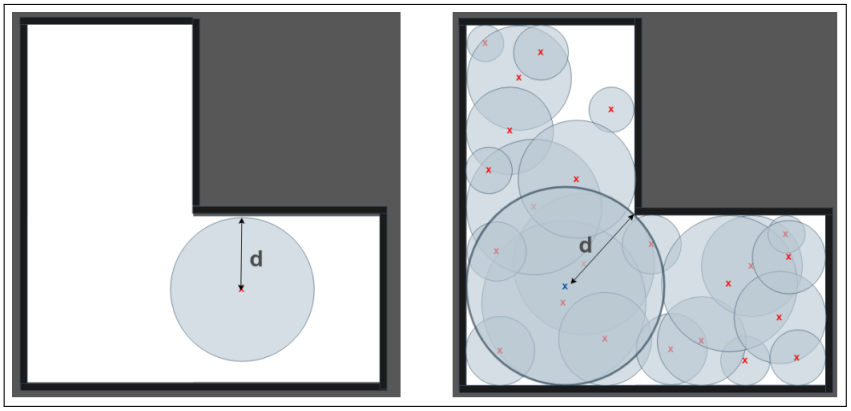


Figure 3.2: Left: an 'L' shaped environment and an arbitrary circle with depth as the radius. Right: 25 random points inside the 'L' shaped region, the biggest circle marked in a thicker circumference.

when evaluated on the depth property.
Considering the set of points p making the polygon map $\bar{P}$, and an arbitrary point x_c contained by the polygon. The depth δ at point x_c can be computed using Equation 3.2.

$$\delta = min\{ \parallel p - x_c \parallel \} : \forall\ p \in \bar{P} \tag{3.2}$$

On a practical note, point p is identified using the nearest neighbor search. A k-d tree-based search algorithm (Maneewongvatana and Mount (1999)) that has only O($logn$) computation, is used to find the nearest point of x_c in $\bar{P}$. Hence, the visibility function computes the depth as a Euclidean distance between the point sampled within the polygon and the nearest neighbor on the polygon, as shown in Algorithm 1.

Algorithm 1: Visibility

This algorithm calculates visibility at a given point inside the polygon map

Input: X_c: $\{x_c^1, x_c^2 .. x_c^m\}$; `// Training sample points inside the polygon`
$\bar{P}$: $\{p_1, p_2 .. p_{n_p}\}$; `// Polygon map (see Appendix B Algorithm 6)`
Output: Δ: $\{\delta_1, \delta_2 .. \delta_m\}$; `// Depth at the training points`
for $i \leftarrow 1$ ***to*** m **do**
  $p_*^{(i)}$ = nearestNeighbour($\bar{P}, x_c^{(i)}$) ; `// kd-tree search algorithm (see Maneewongvatana and Mount (1999))`
  $\delta^{(i)}$ = distanceEuclidean($p_*^{(i)}, x_c^{(i)}$) ; `// Distance between training point and nearest point on polygon`
end

Design of Gaussian process model

The Gaussian process model is trained with the visibility computed at random locations inside the polygon. The trained GP can be used to predict the visibility at any point in the environment, by learning a mapping from input space $\mathscr{X}$: $\mathbb{R}^2$ of points inside the polygon to an output space $\mathscr{Y}$: $\mathbb{R}$ of real-valued depth.
A GP is a distribution over functions. Hence, for any subset of functions $f \in \mathscr{F}$ such that $\mathscr{F}$ is a set of functions mapping from $\mathscr{X} \rightarrow \mathscr{Y}$, then there exists a multivariate Gaussian distribution (see Appendix A.1) over $\mathscr{F}$. Therefore GP is a distribution with mean (m) and covariance (k) as functions of $x_c \in \mathscr{X}$. Hence for an infinite number of points in $\mathscr{X}$ contained by the polygon map, there exists a finite set of training sample points $U : \{u_1, u_2 \ldots u_m\} \in \mathscr{X}$. The associated

finite set of random variables $f(u_1), f(u_2) \cdots f(u_m)$ have a distribution given by Equations 3.3 and 3.4.

$$\begin{bmatrix} f(u_1) \\ \vdots \\ f(u_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(u_1) \\ \vdots \\ m(u_m) \end{bmatrix}, \begin{bmatrix} k(u_1,u_1) & \cdots & k(u_1,u_m) \\ \vdots & \ddots & \vdots \\ k(u_m,u_1) & \cdots & k(u_m,u_m) \end{bmatrix} \right) \tag{3.3}$$

i.e,

$$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot)) \tag{3.4}$$

Where $m(u)$ is the *mean function*, which in this case is the visibility function that evaluates the visibility at u. And $k(u,u')$ is the *covariance function* for any $u, u' \in U$. A covariance function or kernels are positive semi-definite matrix used to define the correlation between the different samples.
Figure 3.3 shows m random training samples and their evaluated depth in the 'L' shaped environment.

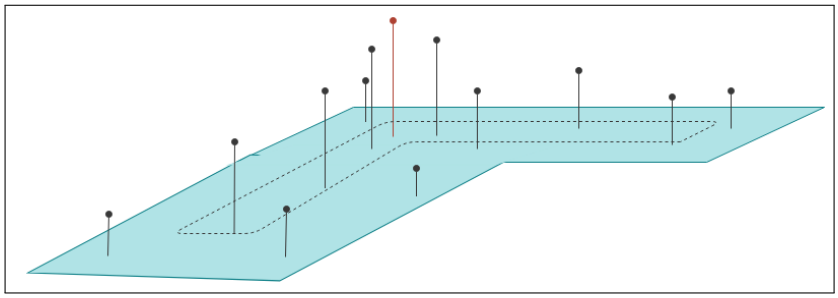


Figure 3.3: Random training samples inside the polygon and their depth evaluation.
Note: The figure intends to provide visualization of the concept and hence the scale is not maintained.

Covariance function (Kernels)

To capture the relation between points inside the polygon there are several kernels available. However, only a *radial basis function kernel (RBF)* (Rasmussen and Williams, 2006) and *Matern52* (Stein, 2015) kernel will be evaluated in this research. An RBF kernel given by Equation 3.5 is a popularly used kernel that correlates nearby points more compared to the farther.

$$k_{RBF}(u,u') = exp\left(-\frac{\| u-u'^2 ||}{2l^2}\right) = \begin{cases} 0, & \text{if } ||u-u'|| \to \infty \\ 1, & \text{if } u = u' \end{cases} \quad \forall u, u' \in U \tag{3.5}$$

Where the length scale l defines how far apart, the points should be correlated. Moreover, the functions drawn from a GP that uses an RBF kernel will be smooth because the RBF is infinitely differentiable.
The Matern kernel was used by (Jadidi et al., 2014, 2016) to construct continuous GP maps, due to its capability to model structural correlation of complex environments. Compared to an RBF kernel, the Matern covariance function contains a term ν, such that the covariance function is $[\nu - 1]$ times differentiable. Hence the Matern covariance function can correlate more complex distribution by controlling the degree of smoothness with finite differentiability. A Matern52 kernel with $\nu = 5/2$ is two times differentiable and is given by the Equation 3.6.

$$k_{Matern\_\nu=5/2}(u,u') = \left(1 + \frac{\sqrt{5}|u-u'|}{l} + \frac{5|u-u'|^2}{3l^2}\right) exp\left(-\frac{\sqrt{5}|u-u'|}{l}\right) \ \forall u, u' \in U \tag{3.6}$$

Further, both the kernels will be tested in Section 5.1 to identify its suitability for this application. The work by Rasmussen and Williams (2006) is a good reference for further details into kernels.

Regression model

Considering a set of samples $S_{train} = \{(u^{(i)}, y^{(i)})\}_{i=1}^{m}$ from an unknown distribution, it is possible to consider a GP regression model given by Equation 3.7,

$$y^{(i)} = f(u^{(i)}),\ i = 1 \dots m \tag{3.7}$$

A prior distribution over functions $f(\cdot)$ can be assumed to be a GP defined in Equation 3.3. The prior distribution can be used to evaluate the posterior for a set of test samples $S_{test} = \{(u_*^{(i)}, y_*^{(i)})\}_{i=1}^{m_*}$ from same unknown distribution by applying Bayes rule, assuming that S_{train} and S_{test} are mutually independent. However, for a GP, the posterior can be conveniently evaluated by performing a conditioning operation (see Appendix A).
The function $f(\cdot)$ drawn from a multivariate Gaussian process prior, will belong to a joint distribution which holds for both the training and testing samples, given by the Equation 3.8

$$\begin{bmatrix} \vec{f} \\ \vec{f}_* \end{bmatrix} | U, U_* \sim \mathcal{N}\left(\begin{bmatrix} M(U) \\ M(U_*) \end{bmatrix}, \begin{bmatrix} K(U,U) & K(U,U_*) \\ K(U_*,U) & K(U_*,U_*) \end{bmatrix} \right), \tag{3.8}$$

where,

$$\begin{aligned} \vec{f} = [f(u^{(1)}), f(u^{(2)}) \dots f(u^{(m)})]^T \in \mathbb{R}^m, \quad & \vec{f}_* = [f(u_*^{(1)}), f(u_*^{(2)}) \dots f(u_*^{(m_*)})]^T \in \mathbb{R}^{m_*}, \\ M(U) = [m(u^{(1)}), m(u^{(2)}) \dots m(u^{(m)})]^T \in \mathbb{R}^m, \quad & M(U_*) = [m(u_*^{(1)}), m(u_*^{(2)}) \dots m(u_*^{(m_*)})]^T \in \mathbb{R}^{m_*}, \\ K(U,U)_{ij} = k(u^{(i)}, u^{(j)}) \in \mathbb{R}^{m \times m}, \quad & K(U,U_*)_{ij} = k(u^{(i)}, u_*^{(j)}) \in \mathbb{R}^{m \times m_*}, \\ K(U_*,U)_{ij} = k(u_*^{(i)}, u^{(j)}) \in \mathbb{R}^{m_* \times m}, \quad & K(U_*,U_*)_{ij} = k(u_*^{(i)}, u_*^{(j)}) \in \mathbb{R}^{m_* \times m_*} \end{aligned}$$

The conditioning operation on GP is shown in Equation 3.9,

$$\vec{y}_* | \vec{y}, U, U_* \sim \mathcal{N}(\vec{\mu}_*, \vec{\Sigma}_*) \tag{3.9}$$

where, the posterior distribution with mean $\vec{\mu}_*$ and variance $\vec{\Sigma}_*$ is the predicted model.
Therefore, this results in a continuous model such that at any point, the predicted depth $\tilde{\Delta}$=$\{\tilde{\delta}_1, \tilde{\delta}_2 \dots \tilde{\delta}_{m_*}\}$ is the mean $\vec{\mu}_*$, and the confidence of the prediction is given by variance $\vec{\Sigma}_*$. This can be calculated using Equations 3.10 and 3.11.

$$\tilde{\Delta} = \vec{\mu}_* = K(U_*,U)K(U,U)^{-1}\vec{y} \tag{3.10}$$

$$\vec{\Sigma}_* = K(U_*,U_*) - K(U_*,U)K(U,U)^{-1}K(U,U_*) \tag{3.11}$$

In a GP, a careful arrangement of samples can ensure low prediction uncertainty. Arranging the test samples U_* in a $m_* \times m_* \in \mathbb{R}^{2m_* \times 2m_*}$ grid ensures a uniform spread of samples. This grid generated is termed as *visibility mesh (VM)*. Hence every point in the mesh has a mean and covariance obtained using Equation 3.9.
The mesh has its x-axis, and the y-axis limits identical to that of the polygon map, hence, a higher number of test samples increases the resolution of the GP model. Therefore, in practice, the number of training samples used to model the GP is sufficiently less compared to the number of test samples, i.e., $m << m_*$, for efficient computation load.

The scale of the predicted depth $\tilde{\delta}$ depends on the size of the environment or, more precisely, the size of the free space. Hence, to further operate upon the mesh, the mean value of each point in the mesh is normalized. The normalized depth $\tilde{\delta}_{normalized}$ remains between range [0,1] and is calculated using Equation 3.12.

$$\tilde{\delta}_{normalized}^{(i,j)} = \frac{\tilde{\delta}^{(i,j)} - \tilde{\Delta}_{min}}{\tilde{\Delta}_{max} - \tilde{\Delta}_{min}},\ i,j \in 1,2 \dots m_* \tag{3.12}$$

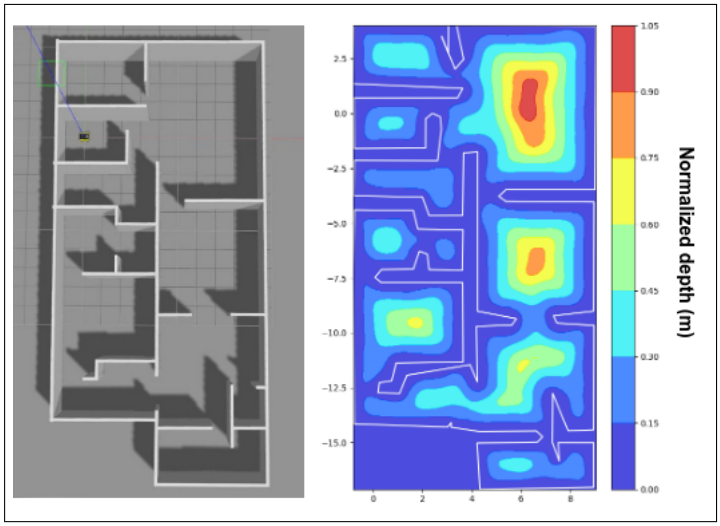


Figure 3.4: GP posterior (visibility mesh) trained with 700 samples, using a Matern52 kernel on a 100 × 100 mesh of test samples for a house model in Gazebo simulator. White outline is the polygon map.

The steps followed by the proposed method to correlate the points inside the polygon with its boundaries results in a posterior prediction of visibility, which is visualized using a contour plot in the Figure 3.4.

From Figure 3.4, it can be observed that the multivariate distribution has high means towards the central region of the rooms. However, the effect of inaccuracies in the polygon map is also visible on the GP model.

It is important to note that the GP model is dependent on the number of samples, the kernel function used, and the resolution of the mesh. Hence the effect of each of the parameters will be evaluated in Section 5.1, intending to obtain suitable value of parameters with least resource utilization.

Detecting viewpoints

So far, in the algorithms, a visibility function is used to train a GP model. Further, a mesh of predicted depth (visibility) had been obtained by performing GP regression on the trained model. From the GP posterior, it is possible to identify subregions that have high visibility. However, to detect the specific viewpoint, the local maximums, or the peaks of the predicted function, f_* is detected.

The peak is detected using image-processing toolkit called scikit-image (van der Walt et al. (2014)) by defining a minimum threshold and a minimum peak to peak distance. Where the minimum threshold defines the minimum visibility and the peak to peak distance signifies the minimum distance between two viewpoints. The viewpoints V: $\{v^{(1)}, v^{(2)} \dots v^{(r)}\} \in \mathbb{R}^{2r}$ are detected using Algorithm 2. The output after detecting the peaks is shown in Figure 3.5.

The figure shows that the robot successfully detects the viewpoint inside the rooms; however, the robot fails to detect four viewpoints. Hence more experiments are done in Section 5.1 to determine the suitable parameters.

First research question aimed at developing an algorithm to locate viewpoints within a room, given the point cloud of the environment. The viewpoints detected in the given point cloud is used as ground truth for evaluating the prediction made using partial maps during exploration. The algorithm contains different parameters that can drastically affect the GP model, like the kernel and its length scale, the number of training, and testing samples. Therefore in Section 4.1, a set of experiments will be designed that can be used to evaluate how successfully the algorithm detects viewpoints for different combinations of parameters.

Algorithm 2: detectViewpoints

This algorithm detects the viewpoints

Input: U: $\{u^{(1)}, u^{(2)} \dots u^{(m)}\}$; // Random training samples

U_*: $\{u_*^{(1)}, u_*^{(2)} \dots u_*^{(m_*)}\}$; // Testing sample mesh

$\bar{P}$: $\{p^{(1)}, p^{(2)} \dots p^{(n_p)}\}$; // Polygon map (see Appendix B Algorithm 6)

Output: V: $\{v^{(1)}, v^{(2)} \dots v^{(r)}\}$; // Detected viewpoints

ϵ_i: Min peak intensity,

ϵ_δ: Min peak to peak distance

Δ = visibilty$(U, \bar{P})$; // See Algorithm 1

GP_{model} = modelGP$(\Delta, k_{matern52})$; // See Equation 3.4

$visibility\_mesh$ = grid(m_*, m_*);

for $i, j \leftarrow 1$ ***to*** m_* **do**

  $\mu_*^{(i,j)}$ = predictGP$(GP_{model}, visibility_mesh_{i,j})$; // See Equation 3.10

end

$\tilde{\Delta}_{normalized}$ = normalize$(\mu_*^{(i,j)}$; // See Equation 3.12

$\tilde{\Delta}_{mesh}$ = reshape$(\tilde{\Delta}_{normalized}, m_*, m_*)$; // 1D depth vector to 2D grid

V = detectPeak$(\tilde{\Delta}_{mesh}, \epsilon_i, \epsilon_\delta)$; // Scikit-image:van der Walt et al. (2014)

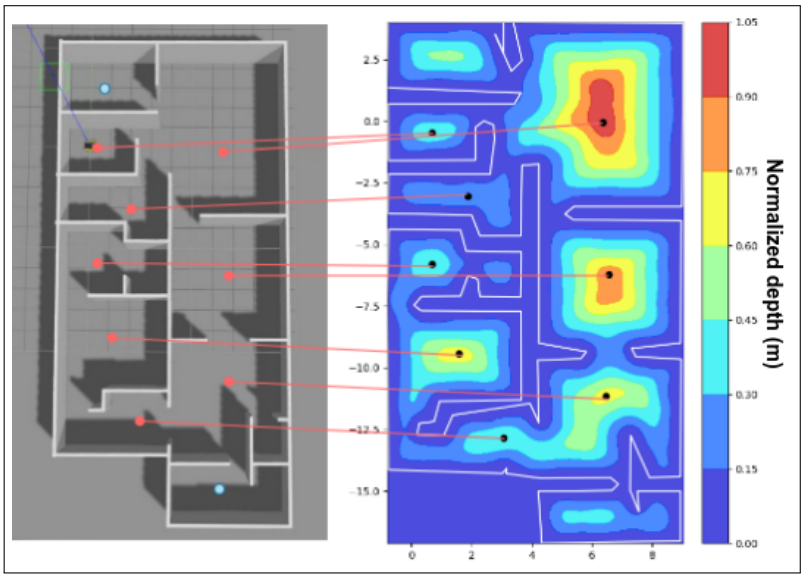


Figure 3.5: Black dots: viewpoints detected by the proposed algorithm. White outline: the polygon map. Red dots: corresponding location in the simulation environment. Blue dots: viewpoints that are not detected.

Until now, the algorithm has been designed considering the availability of a complete point cloud. However, in an unknown environment, the robot has to explore and built the map based on the measurements obtained. Consequently, the prediction of viewpoints is made based on this partial observation.
Therefore, to allow the robot to predict the viewpoints simultaneously, and detect the travel direction, an viewpoint exploration module is designed.

3.2 Detecting explorable viewpoints

In the exploration problem, a complete map is not available, besides a map is partially built during each step of exploration. Solving the exploration problem requires the robot to estimate its position, construct a local map, and navigate to all the viewpoints in the given environment.

However, the robot has to perform this action using the measurements from the 2D LIDAR and wheel encoders. Therefore, to estimate the position and construct the map during navigation, a SLAM algorithm is required.
The robot's odometry measurement is a global source for a position estimate. However, in reality, this model can drift over time due to mechanical design errors and slippage. Hence, a SLAM algorithm finds a transformation that corrects this drift using the LIDAR measurements. The transformation is obtained by aligning the point clouds measured at each time step with the previous. An approach called iterative closest point (ICP) (Segal et al., 2009) algorithm is commonly used to obtain transformations, which can be used to correct the robot's pose estimate made by odometry. Further, the polygon mapping module uses the aligned point cloud to update the polygon for each new measurement made.
The ICP is an iterative algorithm used to find the transformation between scans. The ICP executes in 3 steps (see Appendix B Table B.1), association, transformation, and error evaluation. The algorithm repeats the three steps until the error between the two scans is minimum.
The output of an ICP process is a matched scan, which can be used further to build a map representation. The ICP SLAM algorithm created by the mobile robot programming toolkit (MRPT) (Icp, 2013) is applied, taking into account the implementation aspect. The detailed formulation of the SLAM algorithm is given in Appendix C. The MRPT's ICP package is a suitable design selection due to its low computational requirement. Therefore, the probabilistic occupancy grid map and point cloud representations generated as outputs are used by the designed algorithm. The next step after obtaining the robot's pose estimate and the map is to explore, by detecting regions in the environment that leads to unvisited viewpoints.

3.2.1 Design of exploration framework

At the first step of exploration, the robot has only a partial map of the unknown environment to detect the viewpoints. The exploration module is developed to recognize regions that can lead to more information about the environment and detect new viewpoints.
The second research question investigates "*how accurately the viewpoints can be predicted during the exploration, given a partial map from the SLAM algorithm.*" In the proposed algorithm, the GP predicts the visibility at every point within the polygon map to identify the viewpoints. However, it is crucial to note that the visibility model is independent of the map uncertainties since the polygonal map does not model uncertainty.
The author believes that the exploration problem can be modeled using two hypotheses:

1. *It is possible to determine explorable regions by remembering the areas explored in the past.*

2. *It is possible to decide on explorable regions by identifying regions that can provide more information about the surroundings.*

Hypothesis 1 is a novel approach proposed in this research, and the second hypothesis additionally adapts the traditional information-theoretic concepts to perform exploration.

Hypothesis 1: Remembering the explored

Hypothesis 1 (3.2.1) aims at extending the viewpoint identification algorithm directly to the exploration framework by remembering the past states of the robot, hence resulting in an affinity towards unknown regions.
During the mapping process, the viewpoints should be detected in the regions that have not been visited already. Hence, it is essential to distinguish unvisited viewpoints from the visited in a visibility mesh. A possible method is to augment the robot's state vector x_* to obtain the travel trajectory. Therefore, the trajectory node is a register that remembers all the visited viewpoints, and the robot's past pose. Further, the viewpoints in the unvisited regions of the map

are prioritized. In the proposed approach, this is achieved by removing viewpoints detected within a defined neighborhood of equidistant samples from the robot's trajectory. Further, the detected viewpoints in explorable regions are passed to the decision-making module (see Section 3.3) to plan the travel between them. The proposed approach is developed to obtain a *spatial cognitive exploration and mapping **(SCEAM)*** algorithm (see Algorithm 3).
In the algorithm, a search radius $search_{max}$ is defined around every trajectory node sample

Algorithm 3: SCEAM Algorithm

Input: V: $\{v^{(1)}, v^{(2)} \dots v^{(r)}\}$; // Detected viewpoints
x_*: $\{s_*^{(1)}, s_*^{(2)} \dots\}$; // State vector (trajectory nodes)
Output: V_e: $\{v_e^{(1)}, v_e^{(2)} \dots v_e^{(r_e)}\}$; // Unvisited viewpoints
// Initialize
τ, k, λ; // Trajectory sampling rate, time step, Decay rate
$search_{min}, search_{max}$; // Min search distance, max search distance
$step \leftarrow 0$
$X_e \leftarrow 0$
while $k > 0$ **do**
  if $k - step > \tau$ **then**
    X_s.append($x_*[step]$); // Sampling trajectory nodes
  for $i \leftarrow 1$ ***to*** r **do**
    $x_{nearest}^{(i)} \leftarrow$nearestNeighbor($X_s, v^{(i)}$)// Nearest trajectory node to a viewpoint (see Maneewongvatana and Mount (1999))
    if *distanceEuclidean*($x_{nearest}^{(i)}, v^{i}$) $< search_{max}$ **then**
      V_e.append($v^{(i)}$)// Append viewpoints if no trajectory node in $search_{max}$
  if $V_e \leftarrow 0$ **then**
    if $search_{max} < search_{min}$ **then**
      print Exploration completed; // Exploration completed if no viewpoints left in min radius search
      break;
    else
      $search_{max} = search_{max} - \lambda.search_{max}$; // Decay $search_{max}$ in steps till $search_{min}$ if no viewpoint found
  else
    $search_{max} = search_{max}$; // Restore value of $search_{max}$ if viewpoints are found
  $step = step + k$; // Increment the steps
$k = k + 1$

(Green markers in Figure 3.6) to check if the viewpoints are in the traversed region. However, to prevent the robot from getting stuck in a local exploration, the search radius is decayed till a minimum search radius of $search_{min}$ with a decay rate of λ. When no viewpoints are found even with a minimum search radius, the exploration is assumed to be completed. An example exploration sequence achieved using the SCEAM algorithm is shown in Figure 3.6.

Hypothesis 2: Detecting the unexplored

Hypothesis 2 (3.2.1) aims at extending the viewpoint identification algorithm by considering the environmental uncertainty.

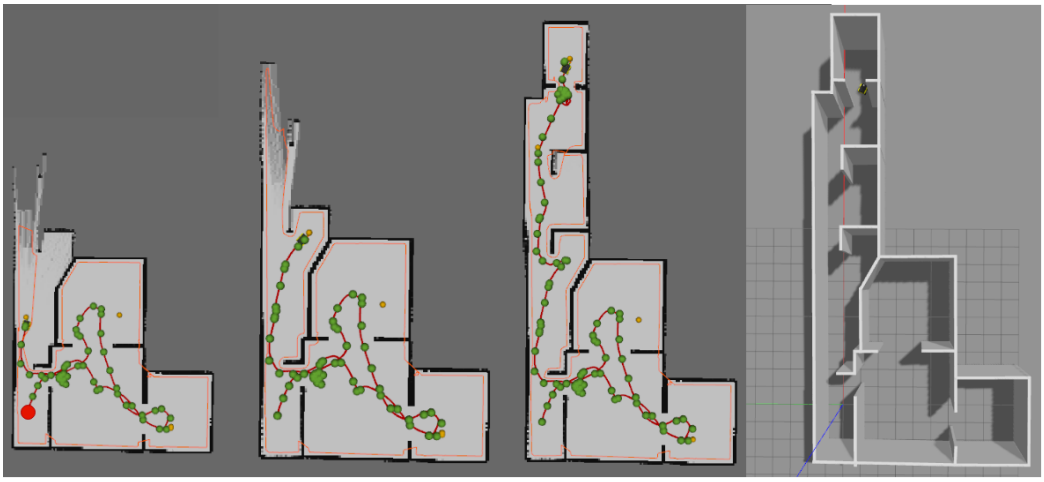


Figure 3.6: Exploration sequence using SCEAM algorithm. Green node are samples from the trajectory, and red marker is the robot's starting position.

Section 2.3 introduces various map representations, and each of them quantifies the occupancy probability of different spaces in the environment. It is visible from such representations that the robot is certain about regions that are entirely observable and uncertain about which are not. Therefore robot has low uncertainty about the regions already visited and high uncertainty about regions that need exploration. Hence occupancy probability can be defined as the property that models the robot's uncertainty about its surroundings. Further, the second algorithm uses map uncertainty to identify viewpoints near uncertain regions. Classical approaches mentioned in Section 2.4 primarily use frontier detection or information-theoretic approaches to detect uncertain regions. Traditional methods like wavefront frontier detection, fast frontier detection, and information-based detection used to detect uncertain regions are mentioned in Appendix D.
From the literature, Bai et al.(2016), in his work on exploration using Bayesian optimization (BO), generated a map representation used for BO based exploration. He used the occupancy probability of the grids to develop a continuous occupancy map using GP. The posterior GP map defines the occupancy probability of every point, and can be adapted to the previously developed visibility GP to perform exploration.

Design of informative GP model
The frontier based approaches detect edges between observed and unobserved regions. Even though such approaches provide a distinct detection of an exploration target, there are no measures of visibility at the selected goal. Moreover, a line segment representing the explorable region fails to capture planar properties, as discussed previously.
Considering the information-theoretic map representation used by Bai et al.(2016), it is possible to model a continuous representation that captures map uncertainty. Such a representation also allows easy adaptation to the proposed exploration framework.
Adapting the concept used by Bai et al., the proposed algorithm develops an informative mesh with a procedure similar to Section 3.1.2. The informative Gaussian process model is developed by sampling cells from the occupancy grid map $\mathcal{M}$, and using the occupancy probability $p(c)$ as the mean of the GP for each cell c. Therefore, the GP model trained using m cell samples is given by Equations 3.13 and 3.14.

$$\begin{bmatrix} f(c_1) \\ \vdots \\ f(c_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} p(c_1) \\ \vdots \\ p(c_m) \end{bmatrix}, \begin{bmatrix} k(c_1,c_1) & \cdots & k(c_1,c_m) \\ \vdots & \ddots & \vdots \\ k(c_m,c_1) & \cdots & k(c_m,c_m) \end{bmatrix} \right), \{c_1, c_2 \ldots c_m\} \in \mathcal{M} \tag{3.13}$$

i.e,

$$f(\cdot) \sim \mathcal{GP}(p(\cdot), k(\cdot,\cdot)) \tag{3.14}$$

However, the remaining part of the algorithm follows the same steps to develop a continuous representation by predicting the occupancy probability at any queried point.
The testing samples are in the form of a mesh called *informative mesh (IM)* that spreads between the x and y-axis bounds of the polygon map. Figure 3.7 illustrates the informative layer formed during an exploration sequence. While sampling the cells for training, cells with occupancy probability in the range (0.3, 0.5) are selected to ensure that the cells are informative and are computationally feasible. The test samples used to generate the informative mesh follows the same structure as that in Section 3.1.2. However, it is essential to note that both the Gaussian processes models are independent. The proposed algorithm uses the generated informative layer to extract the explorable regions.

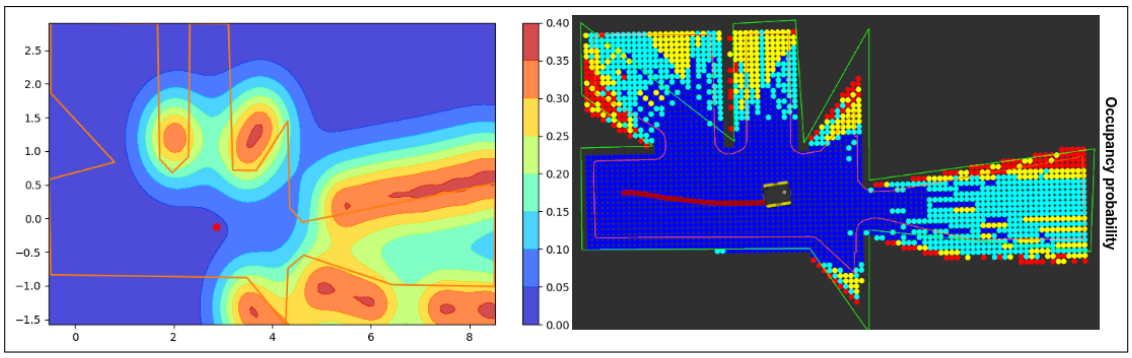


Figure 3.7: Right: probabilistic grid map (RGB). Orange line: polygon map, and green: polygon of current laser scan. Left: GP posterior (informative mesh) trained with 80% of the population (All colored grids on the right) for detailed visualization of occupancy.

Integrating visibility mesh with the informative mesh
The informative mesh highlights regions that have high uncertainty. Moreover, following such regions, it should be possible to discover unknown regions of the map. However, to detect viewpoints, visibility mesh is combined with the informative mesh.
Hence the *informative-SCEAM* algorithm (**iSCEAM**) is developed, which is designed to find the local maximum in VM that is in the vicinity of high uncertainty regions in the IM. The addition of IM to VM amplifies the region in VM, which is in the high uncertainty regions of the map. A thresholding operation is performed to round-off means below a threshold ζ to zero, for better detection of the local maximum. Hence the peaks of the combined mesh are selected as viewpoints.
However, the number of informative cells degrades as the exploration progresses. The robot should be prevented from getting stuck in the local exploration. Hence, the viewpoints in the neighborhood of peaks in the combined mesh are selected as viewpoints for exploration. The proposed iSCEAM algorithm adapting the traditional information-theoretic approach is described in Algorithm 4. Figure 3.8 demonstrates the design of the combined mesh.
The *research question two* intends to investigate the prediction accuracy to judge how well the algorithm can work with partial measurement. The prediction of viewpoints during exploration is evaluated by comparing each decision to the prediction given a complete map (used as a ground truth model) developed in Section 3.1. The experiments set up for the verification will be discussed in Section 4.2.
Based on the hypothesis, both SCEAM and iSCEAM algorithms produce a different list of viewpoints. However, to determine which viewpoint has to be visited first, the last part of the algorithm is designed to make decisions by generating a sequence of the navigation goals from the

Algorithm 4: iSCEAM Algorithm

Input: V_{meash}; // Visibility mesh: Algorithm 2
I_{mesh}; // Informative mesh: informative GP posterior
Output: V_e: $\{v_e^{(1)}, v_e^{(2)} \dots v_e^{(r_e)}\}$; // Unvisited viewpoints
ζ, k; // Mean threshold, time step
while $k > 0$ **do**
  $mesh_{normalized}$= normalize($V_{mesh} + I_{mesh}$); // Combined mesh (Eq 3.12)
  $mesh_{thresh}$ = thresholding($mesh_{normalized}, \zeta$)// Round values below ζ to 0
  // Detect peak: scikit-image (van der Walt et al., 2014)
  V_δ=detectPeak($V_{mesh}, \epsilon_i, \epsilon_\delta$)
  V_I=detectPeak($I_{mesh}, \epsilon_i, \epsilon_I$)
  V_C=detectPeak($mesh_{normalized}, \epsilon_i, \epsilon_C$)
  if $V_C \leftarrow 0$ **then**
    for $i \leftarrow 1$ ***to*** r **do**
      if $distanceEuclidean(V_c, v_\delta^{(i)}) < search_{min}$ **then**
        V_e.append($v_\delta^{(i)}$); // Select viewpoints near to peaks in combined mesh when no unvisited combined peaks
  else
    $V_e \leftarrow V_c$; // Select peaks of combined mesh as viewpoints
  if $V_e \leftarrow 0$ **then**
    if $search_{max} < search_{min}$ **then**
      print Exploration completed; // Exploration completed if no viewpoints left in max radius search
      break;
    else
      $search_{min} = search_{min} + \lambda.search_{min}$; // Extend $search_{min}$ in steps till $search_{max}$ if no viewpoint found
  else
    $search_{min} = search_{min}$; // Restore value of $search_{min}$
  $k = k + 1$; // Update the step

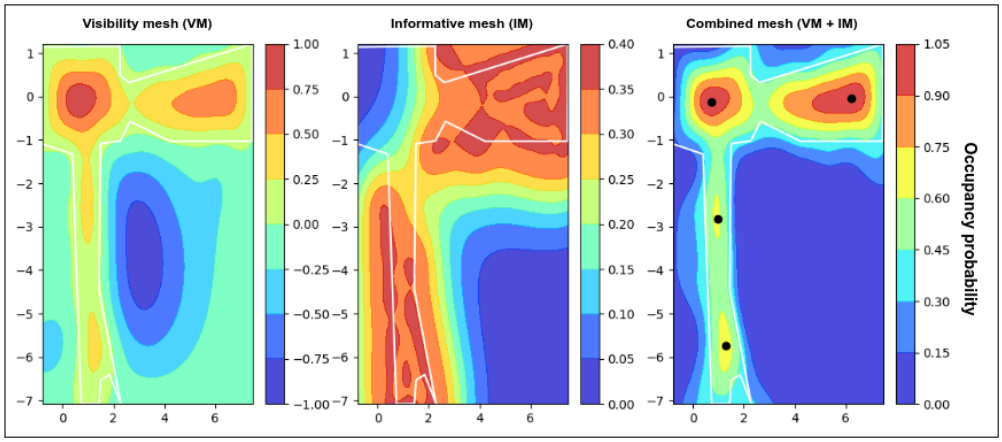


Figure 3.8: Left: visibility mesh. Middle: informative mesh. Right: combined mesh after thresholding and normalization. Black dots: detected viewpoints. White: polygon map.

suggested viewpoints. Note that the decision-making module is independent of the algorithm that generates the list of viewpoints.

3.3 Decision-making

Decision-making in the exploration problem deals with the heuristics or cost function used to prioritize one action over the other. In Section 1.2, various approaches used in literature to make exploration decisions are discussed. These approaches are generally either continuous or discrete. In literature, authors Jadidi et al. (2014, 2016) and Francis et al. (2018) implemented continuous motion controls. These types of motion planning generally use information gradient to generate motion controls. However, such approaches lack a planning layer because of continuous decisions. The second and more commonly used method involves discrete actions done by Yamauchi (1997), Stachniss et al. (2005) and Holz et al. (2010). These approaches identify locations that are explorable and then select one location to execute an exploration action.
Most works in the literature use cost-functions to determine the cost of executing each action, and few others also consider the profit. The goal is selected by either minimizing the cost or maximizing the profit. The cost evaluated in general is the distance the robot has to travel to reach the desired location. The nearest frontier approach by Yamauchi and Holz et al. explores by traveling to the frontier, which minimizes the distance-cost. The profit is the information gained by the robot on visiting the location. Methods in the literature either approximate information gained by calculating the entropy of uncertain cells on the frontiers, or simulate the observations on traveling to the frontier to compute the gain as done by Stachniss et al.. The profit is maximized by traveling to the frontier with the highest information gain irrespective of the distance. Some approaches combine both cost and profit for making these decisions. These methods are compared with the proposed algorithms in Section 5.3.
Even though there are several techniques for decision-making that reduce the cost of individual decisions, only a few works focus on optimizing the cost over a sequence of decisions (see Zhu et al., (2018)). In this work, the sequence of goals is optimized to minimize the total path cost, rather than minimizing the cost of a single step. Therefore in this research, "*How optimizing the route between the viewpoints affects exploration compared to heuristics used by baseline methods*" is investigated. Formulating the decision-making problem requires discussion on the requirements and challenges in the context of this research.

Requirements

- All the viewpoints selected should be visited. Hence, all points have the same preference.
- The sequence of decisions needs optimization, intending to reduce the total travel cost during exploration.

Challenges

- All the points between which the cost has to be optimized are not visible at once. Hence it is hard to find the shortest route for the entire exploration.
- The decisions taken cannot be reversed when a better decision is available later.

3.3.1 Proposed heuristics

All explorable candidate viewpoints $V_e \in \mathbb{R}^{2r_e}$ have an equal probability of being selected as a goal $\bar{V}_{goal} \in \mathbb{R}^2$, and there is no contribution by the profit term since all the viewpoints have to be visited. Hence the only cost incurred is of traversing between the goals.
The following statement describes the approach used to optimize the trajectory:

If visiting all the available goals lead to complete exploration, then it is possible to determine a sub-optimal sequence that minimizes the cost of the entire exploration.
However, this is not the case in reality. Nevertheless, it is possible to approach the problem with such a heuristic. During exploration, the robot determines such a sequence and then starts to visit the first. However, in most of the cases, the robot ends up discovering more goals that were not initially observable in a partial SLAM map available during exploration. The next iteration is performed, believing that visiting the new set of goals will lead to complete exploration. The robot repeats the strategy until the exploration is complete. However, optimizing each step does not lead to a globally optimal result. Besides, it is itself the definition of a greedy optimization. However, considering the stochasticity of the environment, the proposed approach considers beyond minimizing the cost of a single action.

Traveling salesman problem

Finding the sub-optimal route between the available targets can be formulated as a traveling salesman problem (Htun and Thi, 2018). A traveling salesman problem (TSP) is an NP-hard (Woeginger, 2001) problem used to determine the shortest route between the candidates. The problem initially ideated by Menger et al., (1998) was an inspiration for the various derivations of TSP. Even though there several methods that solve the TSP problem, it is hard to converge to an optimal solution for a large number of nodes. However, for this thesis, the number of targets available at any instance is small and hence can provide near-optimal results. Due to the small number of targets, the thesis implements a combinatorial-optimization algorithm called ant colony optimization (ACO) (Gupta (2019), Khatri, (2014), Htun and Thi, (2018)) to find the shortest path between the viewpoints available at any instance.
ACO mimics the behavior of ants to develop a probabilistic technique that finds the shortest path through graphs. Ants, during its quest for food, leaves behind a chemical called *pheromone*, which can be sensed by other ants. The pheromone has a property by which it evaporates in time. Hence the path taken by the ant, which could fetch food and reach back in minimum time, has the highest pheromone deposits. The ants sensing the pheromone, guide themselves to the path with the highest concentration and hence tends to the shortest path.
The ACO algorithm generates a sequence of viewpoints that are evaluated using the Euclidean distance between them, graphically. Using Euclidean distance has an evident shortcoming because the actual trajectory length between the viewpoints might be different from the Euclidean. However, the graph-based Euclidean distance calculation is used in this work for simplicity. After executing the first goal in the sequence, the algorithm repeats the search for a new sequence based on the new observations. The procedure repeats until the end of the last sequence to complete the exploration, as shown in Algorithm 5.

Algorithm 5: makeDecision

Input: V_e: $\{v_e^{(1)}, v_e^{(2)} \dots v_e^{(r_e)}\}$; `// Explorable viewpoints, given by Algorithms 3 and 4`
Output: $\tilde{v}_{goal}$; `// Target viewpoint to visit`
`// Initialize the parameters`
$size_{colony}, iter, k$; `// Colony size, maximum iteration, time step`
while $k > 0$ **do**
  $\tilde{V}_{seq} = \text{ACO}(V_e, size_{colony}, iter)$; `// Route optimization (Gupta, 2019)`
  $\tilde{v}_{goal} = \tilde{V}_{seq}[1]$; `// Select first viewpoint in the sequence as goal`
$k = k + 1$; `// Update the time step`

In the proposed approach, optimization is done to minimize the total route cost between all the available viewpoints compared to the baseline methods that minimize the cost of only a

single action. The *third research question* intends to compare the performance of the proposed algorithm with the baseline methods. The comparison includes methods that minimize the cost and maximize the profit for better evaluation. The exploration can be compared by quantitatively analyzing the area explored, the time taken for exploration, and the length of the trajectory. Further, qualitative analysis, including the path repetition and the quality of the goals generated, can help in making conclusive remarks on the decisions made. Section 4.3 will describe a structured procedure to perform the comparison study.

3.3.2 Navigation

The decisions made by the decision-making module are in the form of x and y coordinates defining a point inside the polygon map. Navigating the robot to the selected goal coordinates involves two steps,

- Planning a path from the robot's current position to the goal
- Sending velocity commands to the controller to ensure the path is followed

A global planner uses Dijkstra's (Javaid, (2013)) algorithm to achieve the former, and a local planner archives the later. The global planner performs path planning using a global navigation cost. The local planner considers the robot's dimension to avoid collisions with the obstacles and uses the motion model to determine the control inputs to the actuators. The motion planning is done by a local map representation that defines the navigational cost. The navigation cost maps is implemented using the `costmap`[†] ROS package. Further, the local planner also performs recovery procedures like rotation and oscillation when the robot's global planner fails to produce a safe trajectory. Both global planner and local planner is part of in-built navigation toolbox of ROS called `move_base` [‡], which directly generates velocity commands for navigating to the desired goal.

[†] `costmap`:http://wiki.ros.org/costmap_2d
[‡] `move_base`: http://wiki.ros.org/move_base

4 Experimental design

The algorithm designed in the previous chapter consists of modules that need to be tested and verified. The tests are performed in a simulated environment, and the experiments involved are designed in this chapter. The chapter begins with a short description of the software setup. In later sections of the chapter is the design of three experiments that address the research questions referred to in Section 1.3. The parameters used for the development of each module are described along with the design. The supplementary research goals, simulation setup, and verification approach is also defined for each experiment.
The development uses the ROS (Robotic Operating System) framework and script the algorithms in Python for easy integration with the simulation and visualization modules. Moreover, Python is selected for scripting because of the vast library support. The software setup along with the robot and sensor description is summarized in Table 4.1.

Platform	**Software stack**	**Robot** Simulation model	**Sensors** Simulation model	**Simulation & Visualization**
+ Intel core 17-7700HQ processor + 2.8 GHz base processing frequency + 8GB Nvidia GTX 1050 GPU + 12 GB RAM + Ubuntu 16.04 LTS OS	+Framework: ROS [*] + Language: Python 2.7[‡] + Libraries: *Gaussian processes* + `gpflow`0.5[¶], + `Tensorflow` 1.12.0[‖] *Polygon mapping* + `Shapely` 1.6.4[**]	Jackal [§] + Developed by Clearpath robotics + ROS APIs and drivers available + Max speed 2.0 m/s + 4 Wheel Differential drive	*RP-LIDAR* [†] + Developed by Slamtec + 2D, 360-degree FOV + Range [0.15, 18] m + Angular resolution 0.9^o + 10Hz scan rate + 720 samples/scan + 600 rpm motor *Wheel encoders* + Quadrature encoders + 78000 pulses/m	+ Gazebo 7.0.0[††] Simulator + Rviz visualizer

Table 4.1: Development and test setup

The verification of the developed algorithm is done by setting up experiments, as described in the following sections. An overview of the experiments designed and the modules tested are described in Table 4.2.

Exp	**Test module**	**Aim**	**Measured quantity**
1	Viewpoint detection (M1)	Study the effect of GP parameters on viewpoint prediction	Viewpoint prediction score (complete map)
2	Explore viewpoints (M2)	Prediction of viewpoint during exploration	Viewpoint prediction error (partial map)
3	Complete algorithm (M1, M2, M3)	Comparing exploration with the baseline approaches	Area explored, exploration time, path length, goal quality, path repetition

Table 4.2: Overview of experimental design

[*]ROS: https://www.ros.org/
[‡]Python2.7: https://www.python.org/
[§]Jackal: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/
[†]RP LIDAR: https://www.slamtec.com/en/Lidar/A2
[¶]gpflow: https://pypi.org/project/gpflow/
[‖]TensorFlow: https://www.tensorflow.org/
[**]Shapely: https://pypi.org/project/Shapely/
[††]Gazebo: http://gazebosim.org/

4.1 Experimental 1: Detection of viewpoints

The viewpoints are local maximum detected from the GP posterior mesh representing the visibility. However, to approximate the underlying visibility function effectively, the GP parameter has to be carefully studied. The algorithm setup for detecting viewpoints has a number of parameters in different functional modules. The important parameters are shortly described in Table 4.3.

Function	Parameters	Libraries	Description
Polygon mapper	Simplification factor	simplification 0.4.2[‡‡]	+ A simplification parameter used by the Ramer-Douglas-Peucker algorithm for filtering and down-sampling the point cloud to a polygon.
	Polygon buffer	Shapely 1.6.4 **	+ Length in meters the polygon's boundary should be compressed to avoid erosion.
Visibility GP	Num. training samples	gpflow 0.5[¶]	+ The number of random location samples within the polygon, used for training the GP.
	Kernel		+ The covariance function used to define the correlation between different training samples.
	Num. testing samples		+ The number of (testing) samples in the meshgrid used for regression.
	Length scale		+ Spatial distance between the training samples that the selected kernel should relate.
Peak detection	Min intensity of peaks	scikit-image[§§]	+ Intensity above which a point can be considered as a peak.
	Min peak to peak distance		+ The minimum distance between points to be considered as distinct peaks.

Table 4.3: Parameters used in viewpoint detection module

4.1.1 Aim

In this experiment, the first research question is addressed by achieving the following research goals.

- Study the effect of the parameters on the Gaussian process model.
- Quantify the observations to evaluate the detection capability.
- Infer an adequate combination of these parameters for further use.

4.1.2 Experiment description

An intuitive understanding of the influence of various parameters of the GP model can help capture the spatial features and to locate viewpoints within the rooms. The kernel function correlates every training sample provided to obtain the GP posterior. Therefore, a higher number of training samples indicate a better understanding of spatial relations. However, a higher number of samples also increases the computation requirement. An estimation of the minimal number of training samples that can be correlated to make a sound prediction is essential. However, a low number of samples are more complicated to correlate; hence the complexity a kernel can handle also makes an influence. While defining the kernel, it is also essential to define the minimum distance between the points that can be correlated. Hence a lower number of samples would need a suitable length scale to ensure the smoothness of the kernel function. The modeled information is extracted using the mesh grid by predicting the mean and variance using Gaussian process regression. The number of testing samples defines the resolution of this grid. Identifying the local maximum in the grid, which serves as the detected viewpoint, implements a peak detection which defines the neighborhood of a local maximum. Ideally, a fine resolution increases the number of testing samples and should improve the results of the peak detection.

[‡‡] simplification: https://pypi.org/project/simplification/
[§§] scikits-image: https://scikit-image.org/docs/0.7.0/api/skimage.feature.peak.html

Simulation setup

The designed viewpoint detection algorithm requires a complete polygon map (obtained from scanned point cloud) of the environment as input. Therefore, for simulation, a Dutch house floor plan that contains rooms of different sizes was built into a 3D Gazebo model, as shown in Figure 4.1. The rooms of different sizes in the environment allows evaluating the adaptability of the algorithm towards structural variations. A 2D point cloud was recorded manually for the complete house shown in Figure 4.1, and the polygon map of which was used by the algorithm to locate the viewpoints.

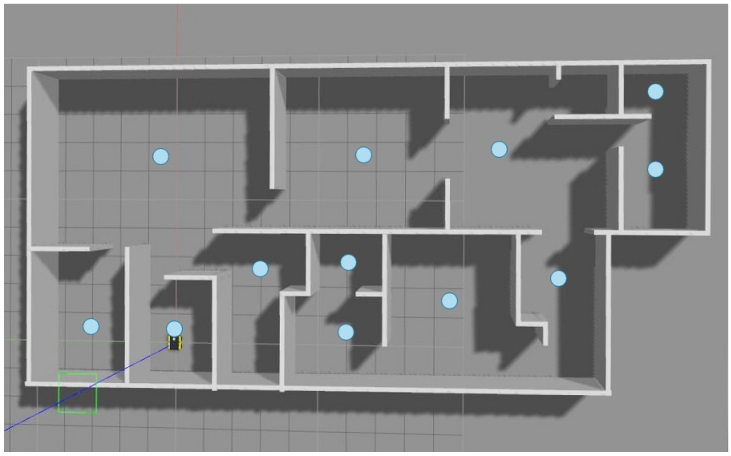


Figure 4.1: Gazebo model of Dutch house with rooms of varying sizes, indicating viewpoints located manually in blue dots.

Verification setup

The gazebo model in Figure 4.1 was manually analyzed to locate r viewpoints within the rooms. In total, 12 viewpoints are selected. The experiment is repeated for a different combination of parameters, as shown in Table 4.4. Each correct prediction of the viewpoints is evaluated as true positive (TP) and every incorrect prediction as False positive (FP). The viewpoints that the algorithm failed to predict does not contribute to the score. Hence the ratio of the difference between the correct and incorrect predictions, and the total viewpoints, evaluates the total prediction score. The prediction score is given by Equation 4.1.

$$Prediction\ score = \frac{TP - FP}{n} \tag{4.1}$$

Each prediction is manually evaluated qualitatively by human verification. The results obtained after conduction the experiment are shown in Section 5.1.

Table 4.4: Experimental parameters and their ranges

Parameter	**Experimental range**
Num training samples	50-1000
Kernel	RBF, Matern52
Length scale	1-3
Num testing samples (mesh)	100x100, 500x500

4.2 Experiment 2: Detection of explorable viewpoints

According to the previous experiment designed, the viewpoint detection algorithm assumes a known environment with map available in the form of aligned point cloud. However, in reality,

there is no point cloud beforehand. Hence the robot has only partial visibility of viewpoints. The designed algorithm enables the robot to detect regions where it has to travel to locate more viewpoints in a partial map. Further, to detect regions to explore, two approaches are designed, each of which hypothesis the problem differently. The important parameters used in the SCEAM and iSCEAM algorithms are described in Table 4.5.

Function	Parameters	Description
Navigation history register	Trajectory sampling rate	+ The minimum distance travelled by the robot before which a sample is recorded from the robot's estimated pose.
	Max search distance	+ The maximum search distance around the path taken by the robot, inside which the goals will be eliminated.
	Decay rate	+ The factor by which the search distance is decayed when no goals are found, to ensure that small regions are not missed.
	Min search distance	+ The minimum search distance until which the decay can continue. A minimum range of zero detects goals visited points.
Informative GP module	Sampling threshold	+ The minimum threshold of occupancy probability sampled for training informative GP. A value closer to 0.5 ensures that it captures a highly informative cell.

Table 4.5: Parameters used in viewpoint exploration module

4.2.1 Aim

In the second experiment, the performance of the proposed viewpoint detection algorithms during exploration is evaluated by addressing the following research goals.

- Quantitatively examines the performance of SCEAM algorithm.
- Quantitatively examines the performance of iSCEAM algorithm.

4.2.2 Experiment description

Exploration is used to visit all unknown viewpoints present in a given unknown environment in the least possible time. However, detecting accurate viewpoints is tricky, considering the occluded vision of the robot. Therefore this experiment is designed to evaluate the accuracy by which the algorithm can detect viewpoints with a partial map. The prediction error of each viewpoint is recorded in comparison to the viewpoints detected with a complete map in Experiment 4.1.

Simulation setup

The SCEAM algorithm requires the point cloud measurements to make the polygon map, and the robot's pose estimate to sample the trajectory followed. The iSCEAM algorithm requires the polygon map and also an occupancy grid map provided by the SLAM algorithm to sample the cell occupancy. At the beginning of the exploration, the robot is placed at an arbitrary position inside the house in a Gazebo simulated environment shown in Figure 4.1. The robot is made to explore the environment autonomously to test the proposed algorithms. Every goal point the robot selects during exploration is recorded for calculating the error in prediction.

Verification setup

The viewpoints recorded during exploration is compared with the viewpoints detected using a complete map (ground truth). The error between each selected goal and the ground truth viewpoint is used to evaluate the accuracy of the predictions made during the exploration, given by the Equation 4.2.

$$prediciton\ error = \hat{v}_i - v_i,\ i \in [1, r] \tag{4.2}$$

Where $\hat{v}$ is the predicted viewpoints and v is the ground truth of the r viewpoints.
Experiment 1[4.1] is performed to identify the right combination of the parameters that can detect all viewpoints. Hence, the results of Experiment 1[4.1] is used as ground truth for the second experiment. The ground truth viewpoints resembles Figure 4.1.
The results obtained during the exploration of both the proposed algorithms are shown in Section 5.2.

4.3 Experiment 3: Decision making and exploration

In the final step of the algorithm, a viewpoint is selected to navigate from a list of detected viewpoints. This selection is made in a TSP setting using an ant colony optimization algorithm (ACO) that optimizes the sequence of visiting the viewpoints using a Euclidean distance heuristic. The ACO algorithm depends on the parameters described in Table 4.6.

Function	Parameters	Reference	Description
Decision making module	Colony size	aco_tsp[¶¶]	+ Each colony denote a unit of ants, and the size represents the combinations executed to optimize the trajectory. Larger colony size tends to optimality faster but at the cost of computation.
	Maximum iteration		+ The maximum number of iterations available for optimization. A higher number of iterations has a greater convergence towards the global optimum.

Table 4.6: Parameters used in decision making module

4.3.1 Aim

The previous two experiments focus on observing the accuracy of viewpoint detection in a known and, then, in an unknown environment, respectively. In this section, an experiment is designed to evaluate the performance of the exploration compared to popular methods in the literature to accomplish the following research goals.

- Observe the performance of the proposed method of route optimization compared to the nearest frontier method.
- Investigate the performance of the algorithm in comparison with an information-greedy heuristic.
- Analyse the changes in comparison with the combination of distance and information-based methods.
- Evaluate the decision and trajectory of each method qualitatively.

4.3.2 Experiment description

From the literature, *the nearest frontier approach,* selects the mid-point of the frontier segment that has the least distance cost. The proposed method also uses distance cost heuristic. However, it evaluates the distance between all the available viewpoints to select the first viewpoint of the sequence that minimizes the distance of the complete tour.

An *information-greedy approach* selects the frontier with maximum uncertainty or entropy. On a practical note, this is the widest frontier segment.
Further, combining both the heuristics by weighting distance and information gain by a parameter α, three *hybrid exploration algorithm* are also defined for comparison. The cost function C used is shown in Equation 4.3, which combines both the distance-cost D and the Information gain I_g.

$$C = D - \alpha I_g \tag{4.3}$$

[¶¶] aco_tcp: https://github.com/rochakgupta/aco-tsp

A frontier with the least cost is selected as the exploration goal. This Experiment proceeds by selecting these methods as a baseline for comparing the performance of the proposed method.

Simulation setup

The exploration by the proposed algorithm is evaluated by performing autonomous exploration in complex simulated environments modeled in the Gazebo simulator and will be compared with the baseline algorithms.

Baseline setup: The baseline exploration algorithms are adapted from the frontier detection based exploration package for ROS developed by Hörner (2016) called `explore_lite`. Further, by improvising the cost function used in the package, a baseline of five methods was created, as described in Table 4.7.

Methods	**Experiment code**
Nearest frontier	D
Information greedy	I
Nearest frontier and information greedy, $\alpha = 0.25$	D25I
Nearest frontier and information greedy, $\alpha = 0.50$	D50I
Nearest frontier and information greedy, $\alpha = 0.75$	D75I

Table 4.7: Baseline methods for comparison

Environment setup: The environments used for testing are Dutch houses models with just the walls. The environments selected has different floor area and different configurations. The varying complexity allows observing diverse exploration behavior. The floorplans obtained from funda.nl* were modeled using Gazebo's building editor, as in Figure 4.2. In this experiment, exploration by the baseline methods and the two proposed methods are simulated on all the five house models.

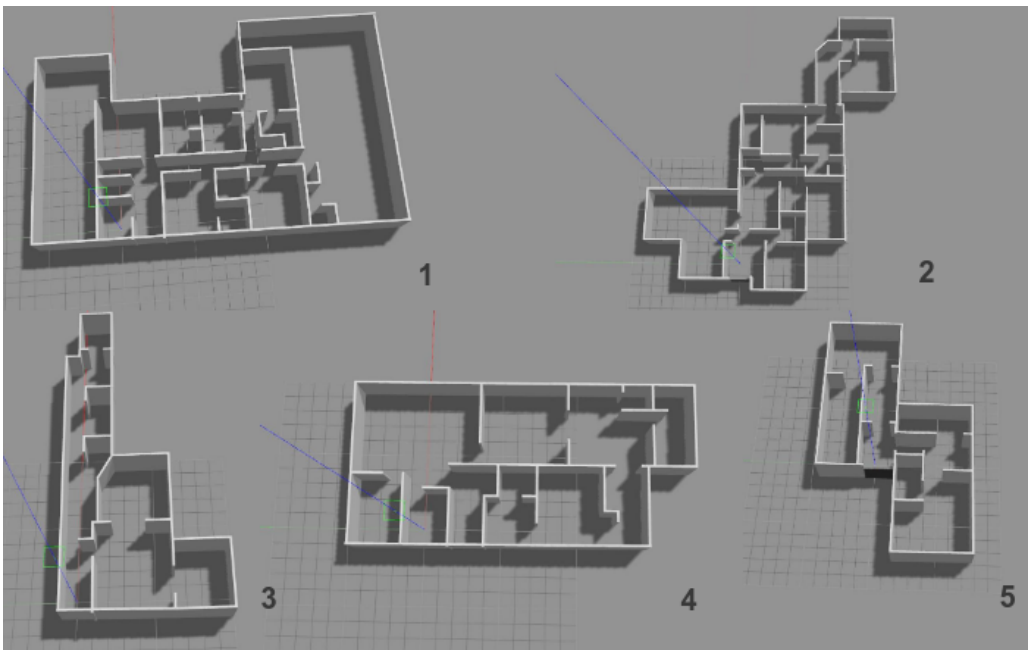


Figure 4.2: Gazebo simulation environments of Dutch houses built from the floor plans.

*funda.nl: https://www.funda.nl/.

Verification setup

The performance of the proposed algorithm is verified by performing both quantitative and qualitative analyses in comparison with the baseline.

Quantitative analysis: This analysis is designed to measure the performance of each method by computing and comparing the following attributes.

- *Path length*: Distance traveled by the robot.
- *Percentage of area explored*: The number of free cells is counted by iterating over the entire cells in a map. The product of the number of free cells, with its resolution, evaluates the total area explored. The percentage is calculated using the actual floor plan area.
- *Exploration time*: The total exploration time is the duration between the beginning of the exploration to the time when it ends the exploration. The exploration terminates for the following conditions:
  - No more goals were available.
  - All the available goals were attempted for 60 seconds each and then canceled.

Qualitative analysis: Investigating the quality of the goals and repetition of the trajectory will be done for qualitative analysis.

- *Quality of goal selection*: The goals selected are analyzed in terms of safety and coverage.
- *Path repetition*: The sequence of goal selection is observed to analyze the repetition of the trajectory followed by the robot during exploration.

The baseline exploration is repeated ten times in all the five environments for consistency. The mean value of the percentage area explored, path length, and exploration time will be used for comparison. The results of the experiments conducted are presented in Section 5.3. The results obtained after running each experiment are illustrated, and the observations are discussed in Chapter 5.

5 Results and discussion

In this chapter, the results obtained by investigating the research questions are discussed. The effects of various GP model parameters on the detection of viewpoints, given a complete map of the environment, are studied in the first section. In the second section, the GP parameters are selected based on the study performed and are adapted to predict the viewpoint in a partially visible map setting. Further, the predictions made by both SCEAM and iSCEAM algorithms are analyzed. Finally, the proposed exploration algorithms are compared to baseline exploration heuristics to evaluate the exploration while using a global optimization-based planning.

5.1 Experiment 1: Results

The algorithm designed in Section 3.1 to detect viewpoints, given a complete map of the environment, is assessed with the first experiment. The effect of the GP model parameters on viewpoint prediction is studied through this experiment and the results are shown according to Table 5.1. The inferred relationship among the parameters is then used to determine the combination of parameters for the successful detection of all viewpoints. The simulation and verification procedure for this experiment is discussed previously in Section 4.1.

Parameters				**Results**
Training samples	**Length scale**	**Kernel**	**Testing Samples**	
Random samples inside the polygon 50, 100, 250, 500, 750, 1000	1,2,3	RBF	100 x100	Figure 5.1
			500 x 500	Figure 5.2
		Matern52	100 x 100	Figure 5.3
			500 x 500	Figure 5.4

Table 5.1: GP parameters and results

The prediction score calculated using Equation 5.1 is used to analyse the effect of parameters quantitatively.

$$\text{prediction score} = \frac{\text{num of correct predictions} - \text{num of incorrect predictions}}{\text{total number of rooms}} \tag{5.1}$$

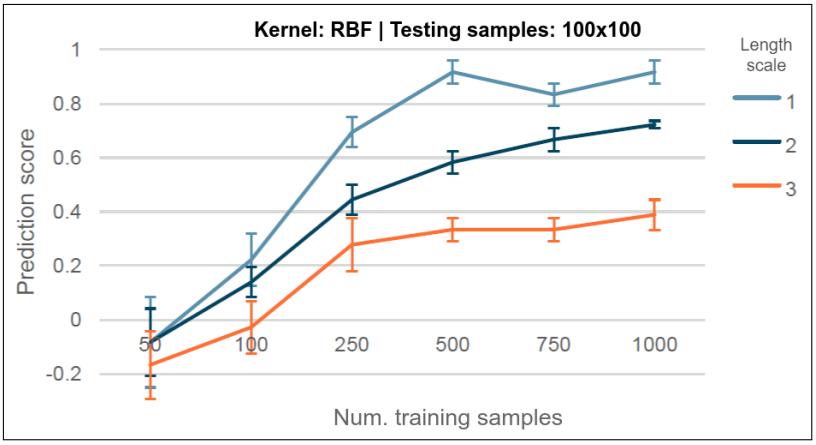


Figure 5.1: Viewpoint prediction score using RBF kernel for 100x100 testing sample mesh.

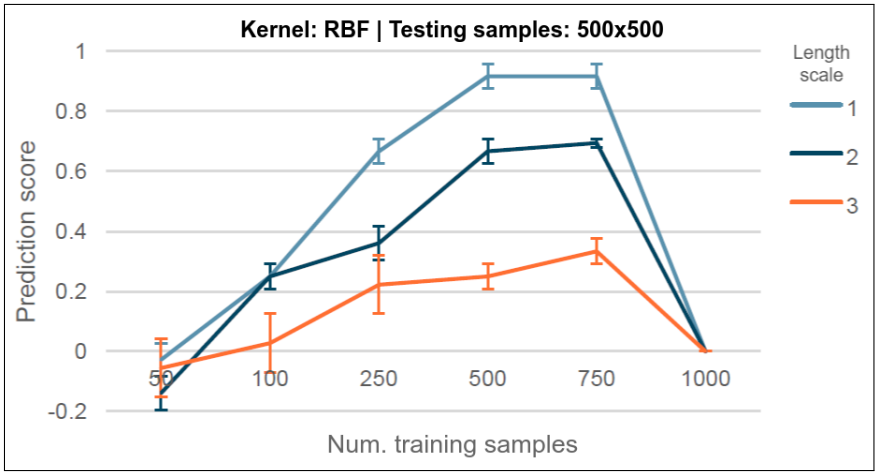


Figure 5.2: Viewpoint prediction score using RBF kernel for 500x500 testing sample mesh.

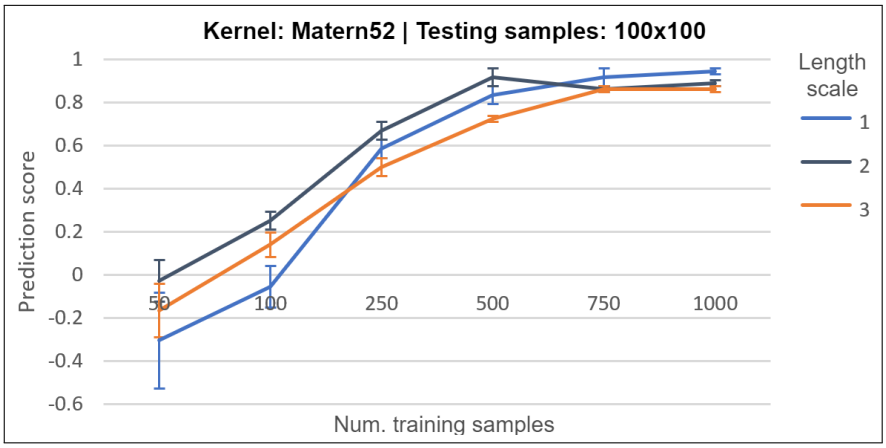


Figure 5.3: Viewpoint prediction score using Matern52 kernel on 100x100 testing samples.

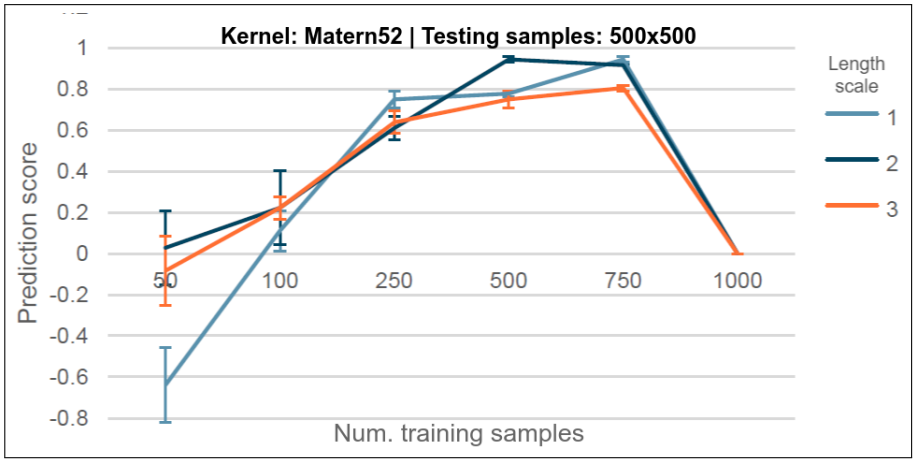


Figure 5.4: Viewpoint prediction score using Matern52 kernel on 500x500 testing samples.

The examined range of parameters resulted in detecting all the viewpoints in the given environment on repeated trials, and the results obtained are shown in Figure 5.5.

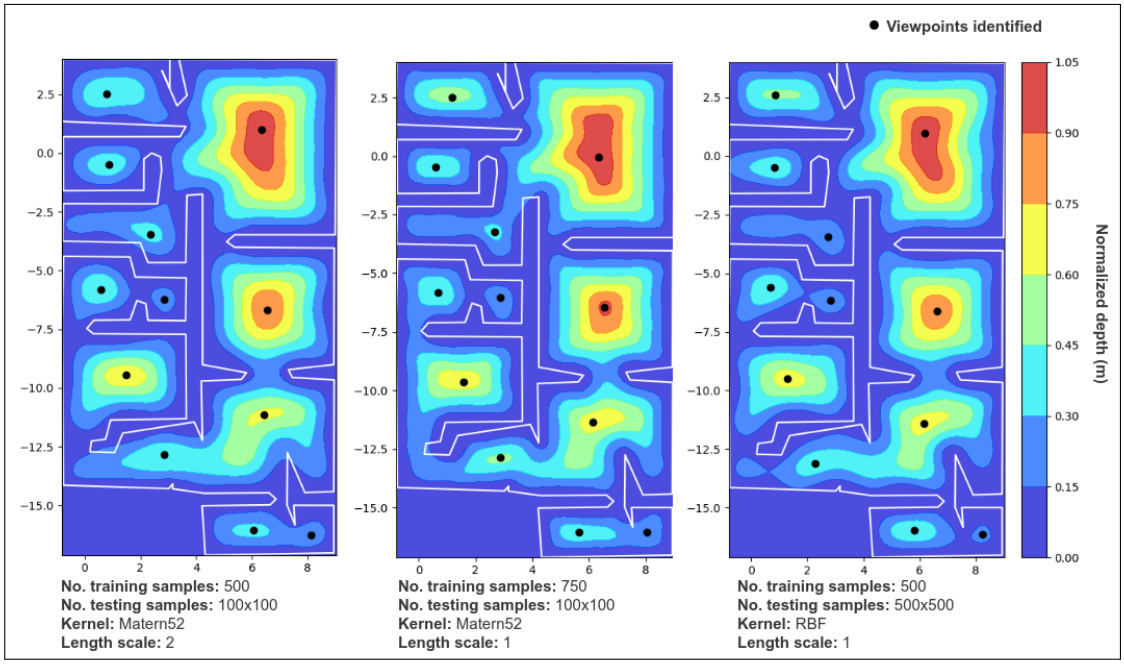


Figure 5.5: GP posterior predicting viewpoints in all the twelve rooms of the given environment.

5.1.1 Experiment 1: Discussion

In this section, the results shown in Figure 5.1, 5.2, 5.3 and 5.4 are discussed. Further, an intuitive discussion of the parameters and their selection is presented.

Effect of GP parameters on prediction score

- The prediction score, in general, has an increasing trend with the number of training samples: For a higher number of training samples, there is more information available for the GP to predict the function, hence the predictions are more accurate and consistent as shown by the less variance and vice versa. The high variance (inconsistency) caused by random sampling, is more visible in the lower number of training samples.

- The prediction score varies with the length scale: The RBF kernel is a distance-based kernel that correlates the training samples radially. The RBF fails to correlate complex spatial data from the map as the length scale varies, because an RBF covariance function is infinitely differentiable and hence very smooth. For a longer length scale, there will be more variations in data, which is hard for a smooth function to capture.
  In the Matern family of kernels, there exists a parameter ν, such that the kernel is $\nu - 1$ times differentiable. When $\nu \to \infty$, the Matern kernel converges to an RBF kernel. Therefore parameter ν can control the smoothness of the function for capturing complex spatial data. Hence, the prediction score has lower variation with length scale for a Matern52 kernel with the value of ν as $5/2$.

- The prediction score drops to zero for 1000 training samples for a mesh of 500x500 testing samples: This is because of the high computational requirement that causes a processing error on the testing work station. The process termination is included as a zero prediction score to eliminate that range from the evaluation.

- The prediction score is below zero for a fewer number of training samples: This is because, the model is not trained properly when fewer samples are used, and hence makes incorrect predictions than correct ones. Therefore it leads to a negative success ratio.

To obtain a more intuitive understanding of the effect of length scale on RBF and Matern52 kernels a different range of length scale is used and the supportive discussions are provided in Appendix E.

Selection of GP parameters

In this experiment, it is showed that the designed algorithms are capable of detecting viewpoints inside a room. However, the parameters used for the GP model affect the detection of viewpoints.

- Based on the observation, training samples of size 250-500 is an adequate range. Hence the visibility GP will use 300 training samples, considering adequate resource allocation during the exploration setup.
  Moreover, The random sampling does not generate samples in every room, and hence those regions are unmodeled or untrained. Besides the successful detection of viewpoints, complete elimination of randomness is not possible even with an increasing number of samples. However, a better sampling procedure should result in more consistent results.

- A Mater52 kernel with a length scale two and 100x100 testing sample mesh will be used. This choice is evident from the robust performance of the Matern52 kernel with structural changes of the environment using a 100x100 testing sample, as shown in Figure 5.3. However, the length scale should be comparable to the spread of a given number of samples even for successful detection by the Matern52 kernel.

- For testing samples, the choice of 100x100 or 500x500 grid changes the resolution of the prediction made. The minimum search length between the two local maximum should be selected according to the resolution. Hence for maintaining the same physical search distance, a 100x100 grid used a peak to peak distance of 10 and 500x500 used 50.

In this experiment, the viewpoints are predicted in a known map. However, in an unknown environment, the robot has to explore to build the map. Hence, based on the intuitions obtained, the viewpoint detection is investigated in an exploration framework to answer research question 2 (1.3) in Experiment 2.

5.2 Experiment 2: Results

In this experiment, the robot is made to explore the environment used in Experiment 1, by using SCEAM and iSCEAM algorithms on a partial map given by the SLAM algorithm. The experiment designed in Section 4.2 uses the viewpoints detected in Experiment 1 as ground truth.
The informative GP used by iSCEAM algorithm is trained with cells having a probability between 0.3 and 0.5, rather than random sampling. Since the data is sparse, a Matern52 kernel with length scale three is used. The number of testing samples remains 100x100.
Figure E.3 and E.4 (Appendix E) shows the sequence of goal selection during the exploration using SCEAM and iSCEAM algorithms respectively.

Results of viewpoint detection during exploration

The viewpoints predicted during the exploration using SCEAM and iSCEAM algorithms, along with the ground-truth viewpoints, is shown in Figure 5.6.

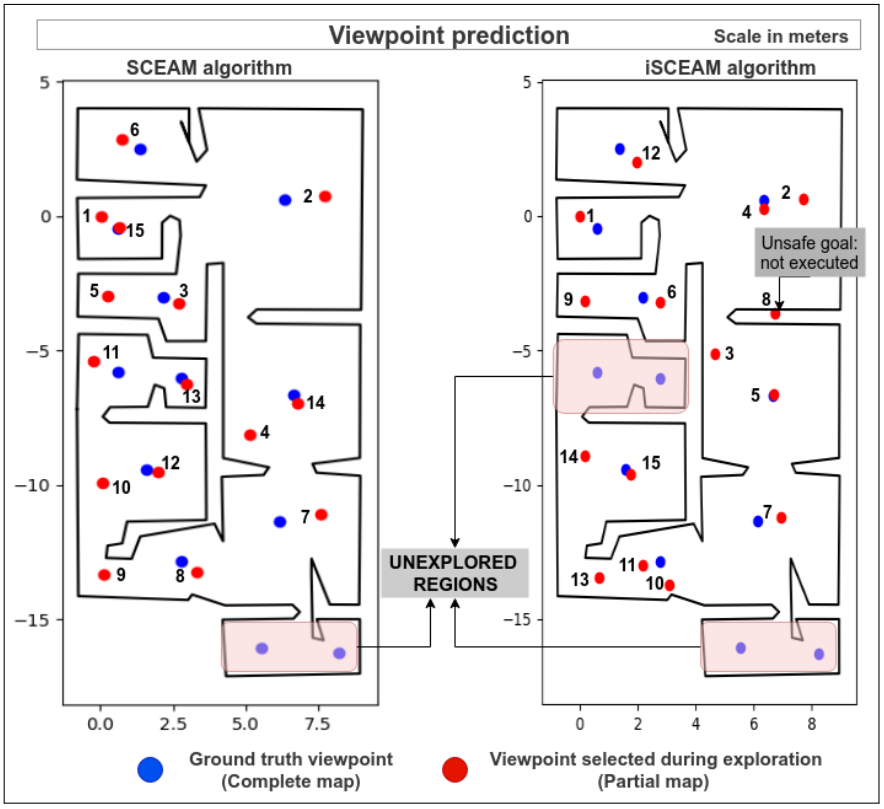


Figure 5.6: Viewpoints predicted during exploration using partial map from SLAM algorithm.

Prediction error results of SCEAM and iSCEAM algorithms

The prediction error is calculated as the Euclidean distance between the ground truth viewpoint and predicted viewpoint. The error for each goal selected by the SCEAM and iSCEAM algorithms are shown in Figure 5.7.

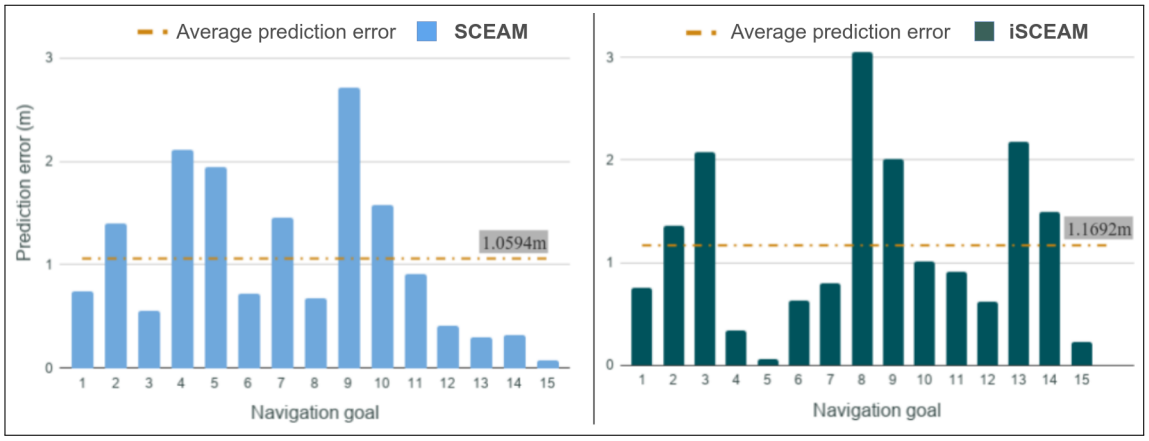


Figure 5.7: Prediction error of each navigation goal selected, as shown in Figure 5.6

5.2.1 Experiment 2: Discussion

In this section, the results of viewpoint prediction by SCEAM and iSCEAM algorithms in a partial map setting, shown in Figure 5.6 and the prediction error, shown in Figure 5.7 are discussed.

Viewpoint detection using SCEAM algorithm

- The algorithm can detect 10 viewpoints out of 12, with the partial map obtained from the SLAM algorithm. All the viewpoints in the explored region were detected with an average error of 1.059 m.
- Multiple viewpoints are detected in some rooms because the explored regions are rechecked when the algorithm fails to find new viewpoints. The rechecking allows improving predictions previously made with a partial map.
- Predicted goals 4,9 and 10 as shown in Figure 5.6 (left) is sufficiently away from the actual viewpoints. Interestingly these points lead to regions that have a wider area and were not visible from the previous goal. Hence, the off-target predictions made are like supporting goals that prevent the robot from getting stuck in a local exploration.

Viewpoint detection using iSCEAM algorithm

- The iSCEAM algorithm detected 8 viewpoints out of 12, given a partial map of the environment. All viewpoints in the explored regions were detected with an average prediction error of 1.069 m.
- The robot revisits some rooms multiple times even though the algorithm was designed to visit viewpoints at uncertain locations because the algorithm searches for regions with higher visibility when it fails to detect new viewpoints in unexplored regions.
- Goals 3,10,13 and 14 are the off-target prediction that leads to more information. The robot first visits viewpoints in uncertain regions and then corrects the prediction based on the new observations.
- In iSCEAM algorithm, the correction step reduces the map uncertainty of a region around the robot because it makes a repeated observation of the same region. Hence, the number of informative training samples is reduced, and the GP fails to model the explorable areas correctly.

5.3 Experiment 3: Results

This experiment focuses on analyzing the exploration based on the sequence by which these viewpoints are visited, hence to answer the third research question (1.3). The experiment designed in Section 4.3 is extended to explore five environments shown in Figure 4.2, to evaluate the robustness and scalability of the algorithm towards structural changes. To compare the performance of the proposed algorithms, classical approaches shown in Table 4.7 are used as the baseline. Further, the results are both quantitatively and qualitatively discussed.

5.3.1 Quantitative experimental results

For quantitative evaluation of exploration, the time taken, the total distance traveled, and the percentage of total area explored are the metrics on which the comparisons are made.
The results obtained are shown in Figures 5.8, 5.9 and 5.10. Further, the average percentage of area explored in all the environments is shown in Table 5.2.

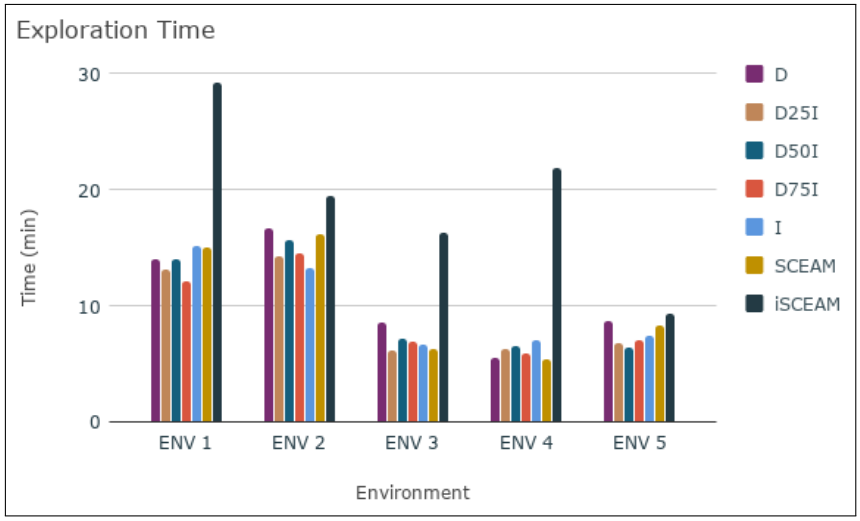


Figure 5.8: Exploration time taken for different algorithms in all the five environments.

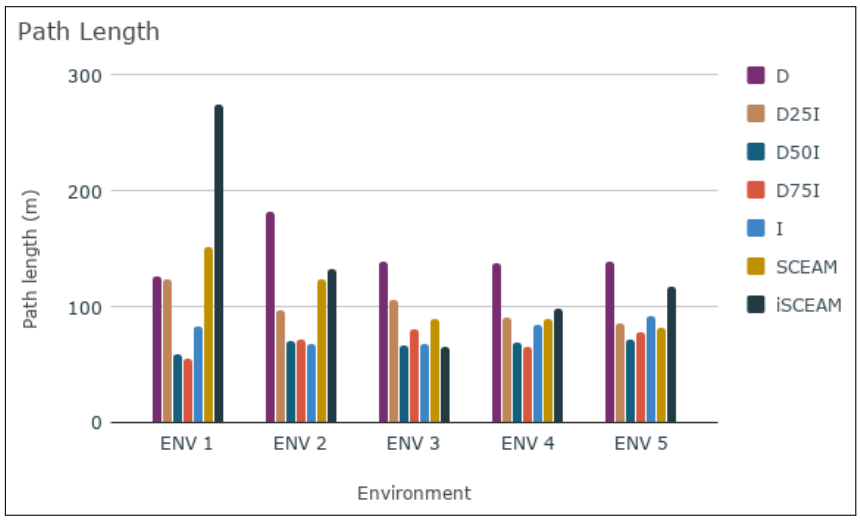


Figure 5.9: Total distance travelled during exploration in each of the five environments.

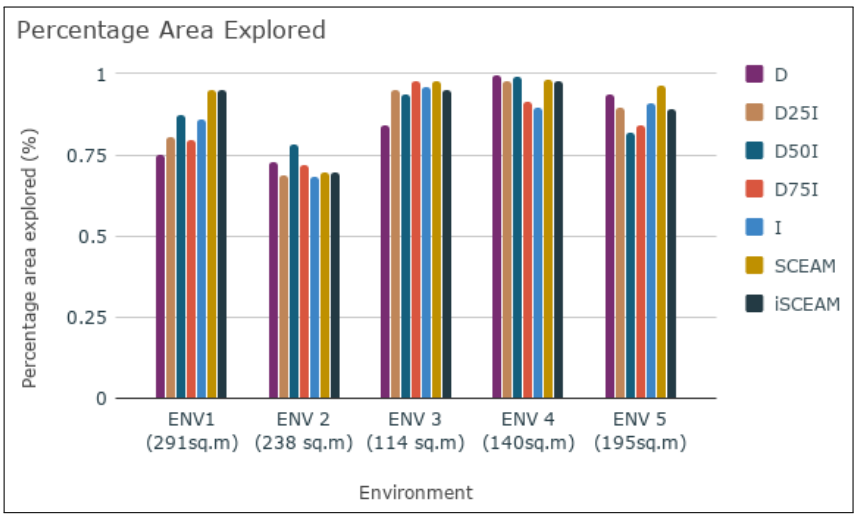


Figure 5.10: The percentage of area explored by the algorithms in each of the five environments.

Method	Average percentage area explore(%)
D	85.0
D25I	86.2
D50I	87.9
D75I	84.85
I	86.08
SCEAM	91.3
iSCEAM	89.24

Table 5.2: The average percentage area explored in all the five test environments.

5.3.2 Quantitative discussion

In this section, the results shown in Figures 5.8, 5.9 and 5.10, and the percentage area explored shown in Table 5.2 are discussed using the following comparisons are made with the baseline.

General comparison

- The exploration results show that the environment 1 and 2 required comparatively higher time to explore and also required a higher path length; this is clear from the area to be explored.
- The second environment has the least area explored, higher exploration time, and lower path length. This signifies that most of the algorithms failed to deduce explorable regions in the map, including the proposed algorithm.

Comparison of SCEAM algorithm

- The exploration time for the proposed SCEAM algorithm is comparable to the other baseline approaches. Even though the time consumed is low in some environment, it does not show consistent performance. However, the exploration time is sufficiently low compared to iSCEAM.
- The path length of the SCEAM algorithm is high compared to the other methods and is mostly lower to the iSCEAM algorithm. However, this is due to the extra length the propose methods have to travel to reach the viewpoint, considering which the SCEAM algorithm still has a comparable trajectory length.
- On comparing the area explored, a remarkable consistency is maintained throughout all the given environment, except the second environment. The SCEAM algorithm on average explores 91.34% of the area and is the highest on the list.

Comparison of iSCEAM algorithm

- The exploration time for iSCEAM algorithm is high compared to all the other methods in the list. This behavior is primarily due to two reasons.

  In environment 1, the exploration time and path length, both have high values; this signifies the repetition of the path. As discussed previously, the iSCEAM algorithm shuffles between the known and unknown regions to reduce viewpoint prediction error. Hence increases the path length.

  In the other environments, the path length is low but still has high exploration time. This is because the exploration time includes both times taken to make a decision, and that to execute an action. Hence, the increase in time is due to the delay in making the decision. This algorithm uses two GP to make decisions compared to one in SCEAM, which

on the workstation requires more resources to process. The computation and memory usage of the algorithms are shown in Appendix E.

- Even though the time consumption and path length are more, the iSCEAM maintains the consistency in the area explored similar to the SCEAM. The iSCEAM algorithm has the second-highest area explored with an average of 89.24%.

5.3.3 Qualitative experimental results

To demonstrate the results of qualitative investigation, the exploration in Environment 1 with total area of 291sq.m is selected. The exploration trajectory of an Information greedy approach, which has area explored near to the proposed method, is used to make a comparison. The environment selected is entirely different from the one in Experiment 2. Hence this allows observing the exploration based on path repetition and quality of goal selection of both the proposed algorithms. The trajectory followed by information greedy, SCEAM and iSCEAM algorithms used for illustrating the qualitative discussion is shown in Figure 5.11.
Further, to observe the trajectory executed in other four environments, the exploration paths of proposed algorithms is illustrated along with one performing baseline algorithm in Figure E.5 (Appendix E).

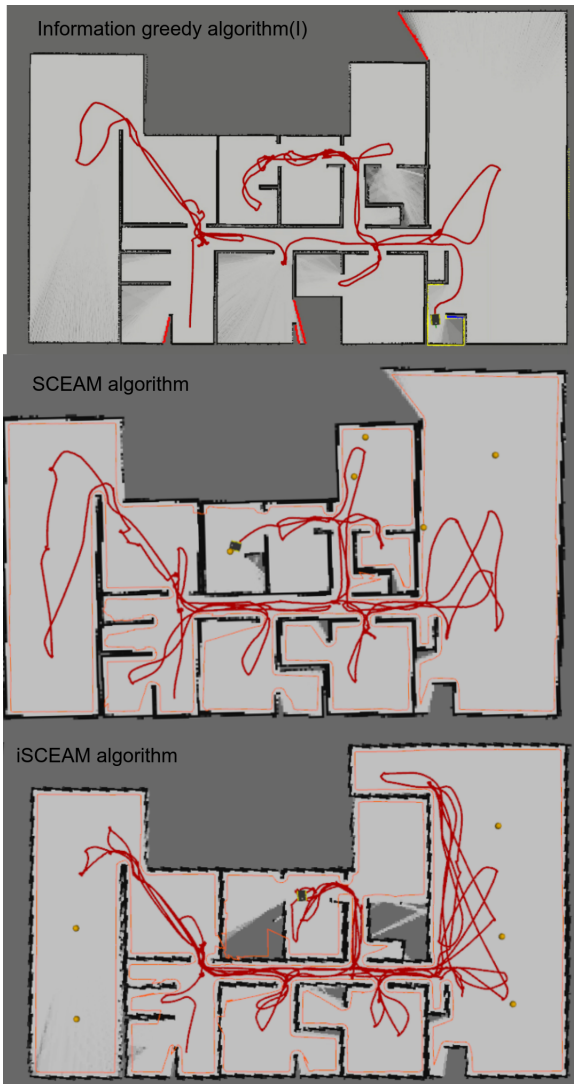


Figure 5.11: Exploration trajectory of the robot in Environment 1.

5.3.4 Qualitative discussion

In this section, the explorations showed in Figure 5.11 is discussed qualitatively.

Evaluation of path repetition

The path repetition of both the proposed algorithms are higher compared to the baseline. Moreover, the iSCEAM algorithm has even higher path repetition. This behavior is because the algorithm tries to improve the predictions made when it gains more knowledge about the environment and hence re-plans a new tour between all available viewpoints.

Evaluation of goal selection quality

The traditional exploration algorithms only aim to explore the whole region. Hence it is not necessary to enter every room to obtain an occupancy representation from the SLAM algorithm. However, The proposed algorithm aims to visit all the rooms to capture the 360-degree view inside the room for a mapping application. The trajectory of the SCEAM algorithm shows that most of the rooms were visited during the exploration, and the viewpoints selected seems to provide a sufficient view of the surrounding. Similarly, iSCEAM also visits the rooms more than the baseline but still fails to detect some of the viewpoints.

Effect of optimization on viewpoint selection

The proposed methods are compared against the baseline exploration methods to investigate the effect of optimizing the travel between the viewpoints.
The TSP framework plans the optimal path to visit all the explorable viewpoints in every iteration and executes the first viewpoint in the optimized sequence. However, because of the cyclic planning, the first goal in the planned sequence can also be a long path. Hence the global path optimization strategy can negatively affect the exploration, causing a considerable increase in path length. This effect is noticeable in iSCEAM algorithm because of the more planning iteration required to find viewpoints using sparse map uncertainty data.
Furthermore, it is not entirely fair to make a verdict on the performance based on comparing the path length and exploration time since the robot has to travel more to visit the viewpoints. However, a comparable time and path length of the SCEAM algorithms are notable.
Exploring the complete house without missing out on any rooms is the goal of the algorithm. The consistency in the area explored by both algorithms on all the environment supports its expected behavior.
Moreover, in all the environments, both the proposed algorithms can locate viewpoints within the explored regions. While the exploration algorithms in the literature do not visit all the rooms. Hence considering the application of mapping, the proposed algorithm is functional. However, the algorithm is in the initial versions and needs more attention to eliminate behavior that causes path repetitions. Further, the sampling technique used to train the GP for detecting the viewpoints should be improved to a more even distribution of samples.

6 Conclusion and Future work

6.1 Conclusion

In this thesis, the viewpoints to perform scanning were detected using a 2D LIDAR sensor to automate the process of mapping. Further, a sequence to visit the viewpoints were determined. This research addressed the following questions.

> "How to develop an algorithm to detect viewpoint within the rooms using geometric relationships, provided a complete house map?"

In this research, a novel Gaussian processes based algorithm is developed to transform the spatial information to a surface function that can evaluate the visibility at every point in the environment to detect viewpoints. Further, it has been investigated that a consistent and robust detection of viewpoints required GP model with a Matern52 kernel, trained with 250-500 samples and tested with a 100x100 mesh. The results confirm that the GP is a powerful tool to correlate spatial information while using a 2D LIDAR and is capable of detecting viewpoint in all the 12 rooms of varying sizes in the test environment.

The following research question studies the effect of viewpoint detection during exploration.

> "Given a partial map from the SLAM algorithm, how accurately can viewpoints be predicted during the exploration?"

To evaluate the prediction of viewpoints with a partial map, the viewpoint detection was adapted to develop two exploration algorithms called SCEAM and iSCEAM. The former explores by actively learning spatial relations, and the latter uses map uncertainty also. The results indicate that the GP model trained with the identified parameters succeeded in detecting viewpoints with an approximate error of 1m using only the partial map. The SCEAM algorithm locates 10 out of 12 viewpoints, while sparseness in map uncertainty data used to train iSCEAM algorithm results in detection of only 8 viewpoints. Further, substituting random sampling with a more efficient technique is believed to improve GP training and increase prediction accuracy.

The following research question compares the exploration of the proposed algorithm with the baselines.

> "How does optimizing the route between the viewpoints affect exploration compared to heuristics used by baseline methods?"

To find the sequence by which the viewpoints should be visited, a TSP based formulation had been used to optimize the travel distance for visiting all the viewpoints in each exploration cycle. A comparison of the proposed algorithms with baseline methods showed that SCEAM algorithm performs equivalent to others considering the path length and exploration time. Nevertheless, iSCEAM takes a longer time and travels more during the exploration because the optimization affects the algorithm negatively. Results of the area explored showed that both SCEAM and iSCEAM algorithms explore 90.34% and 89.2% respectively on an average in all the environments and are the highest compared to the baseline.
The results of exploration demonstrate that the baseline methods explore the environment with shorter and simple paths. However, these methods do not visit the viewpoints inside every room. On the other hand, the proposed algorithms have longer paths and are repeated, but still, they succeed in visiting viewpoints of all the visible rooms. However, since the SCEAM algorithm has comparable path length and time of exploration with the baseline, it is the best

among the choices for exploration in the mapping application.
This research shows that learning spatial relations is a convenient approach to locate viewpoints by using only the 2D LIDAR. Moreover, the proposed SCEAM algorithm can be used to automate mapping based on the results obtained. However, the algorithm explores only 90% of the area, and hence more developments are required to improve the exploration.

6.2 Future work

The presented research can be extended in the following directions:

Probabilistic sampling

In the proposed algorithm, the GP model is trained by randomly sampling points inside the polygon map. However, when the explored area becomes larger, the density of samples reduces and hence can fail to model certain regions. An improvement over the current method can be to learn the explored area so that more samples can be drawn near the unexplored region. The trajectory samples could be used to identify the explored regions. Further, using a probabilistic sampling technique like the Monte-Carlo Markov chain (MCMC) can be used to predict the posterior from the information about the explored region.

Multi-polygon map

The current implementation of the polygon map only considers the exterior boundary assuming a concave polygon environment. However, the objects in the interior polygon will be eliminated by this assumption. Hence, the polygon mapper should be improved to model interior boundaries as well. This could be done by extending the single polygon map to a multi-polygon model, which includes both interior and exterior boundaries.

Motion planning

In the current version, discrete points are selected as viewpoints, and the navigation stack provided by ROS is used for motion planning. However, the Gaussian process model provides a distribution of depth property, which models the safe traversable regions in the map. Due to the continuity of GP models, it is possible to use the predicted values as a cost to plan paths explicitly to the viewpoints. Further, the values could be used as rewards in a reinforcement learning framework to develop continuous motion planing models.

Exploration in a 3D environment

The research shows that modeling spatial relations can also provide a safe traversable region. However, the same concept can be extended to volumetric models. The 3D point cloud can be used to capture more detailed spatial representation. Using semantic maps along with the spatial model will enable extending the algorithm to be used in more cluttered and occluded environments. Moreover, identifying safe traversable volumes will be highly beneficial for unmanned ariel vehicles (UAVs) operating in indoor environments.

A Fundamentals of Gaussian processes

Gaussian processes (GPs) are popularly considered as a powerful tool in machine learning Rasmussen (2004). GP can be used to predict the value of a function given some prior knowledge. This provides an ability to capture the data to fit into a function, commonly known as *regression.*

A.1 Multivariate Gaussian distributions

In a multivariate Gaussian distribution X, each of the random variable belongs to a normal distribution and has a *joint distribution* which is a Gaussian. The Multivariate Gaussian distribution is defined using a $n \times 1$ vector of mean μ and $n \times n$ covariance matrix Σ. Which can be mathematically represented as shown in Equation A.1.

$$X \sim \mathcal{N}(\mu, \Sigma) \tag{A.1}$$

Where, diagonal elements of Σ represents variance of all points and the off-diagonal elements

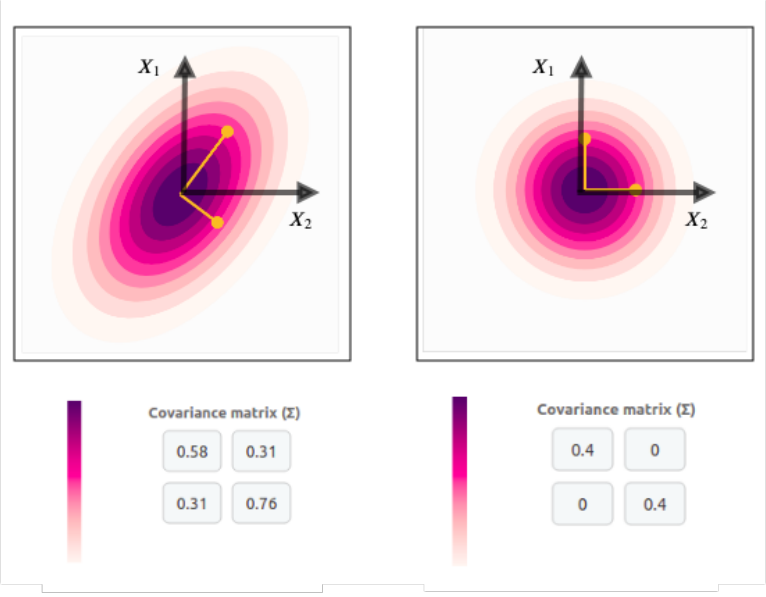


Figure A.1: A joint distribution of X_1 and X_2, created using the visualization tool provided by Görtler et al. (2019). Left: X_1 and X_2 are correlated (). Right: X_1 and X_2 are non-correlated.

describes the correlation between each other random variable in X. The covariance matrix Σ visualises the shape of the distribution. Considering any two points from the input vector X, the covariance can be represented as $\mathbb{E}(X_i, X_j)$ where $\mathbb{E}$ denotes the expectation, see A.2.

$$\Sigma = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)^T] \tag{A.2}$$

In the Equation, $(X_i - \mu_i)(X_j - \mu_j)^T$ is basically the *dot product*, and dot products are used to measure similarity. Considering the joint probability of a 2D example, given by $X = [X_1\ X_2]^T$, the covariance matrix can be visualized as shown in Figure A.1

After representing the data as a Gaussian distribution, to extract information, some operations have to be performed.

A.1.1 Marginalization

Marginalization is an operation performed on Gaussian distribution to separately extract information provided by a single stochastic source of a joint distribution. The peculiar property

of this operation is that it always yields a Gaussian, which is part of the joint distribution. Considering a joint probability distribution of variables X_1 and X_2, as denoted in Equation A.3:

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) = \mathcal{N}(\begin{bmatrix} \mu_{X_1} \\ \mu_{X_2} \end{bmatrix}, \begin{bmatrix} \Sigma_{X_1X_2} & \Sigma_{X_1X_2} \\ \Sigma_{X_2X_1} & \Sigma_{X_2X_2} \end{bmatrix}) \tag{A.3}$$

Given the joint distribution, the marginalized X_1 and X_2 can be formulated as A.6 and A.7

$$X \sim \mathcal{N}(\mu_{X_1}, \Sigma_{X_1X_1}) \tag{A.4}$$

$$Y \sim \mathcal{N}(\mu_{X_2}, \Sigma_{X_2X_2}) \tag{A.5}$$

This corresponds that the marginalized variable only has dependency on its own μ a Σ. The Figure A.2 (left) shows marginalization of X_2.

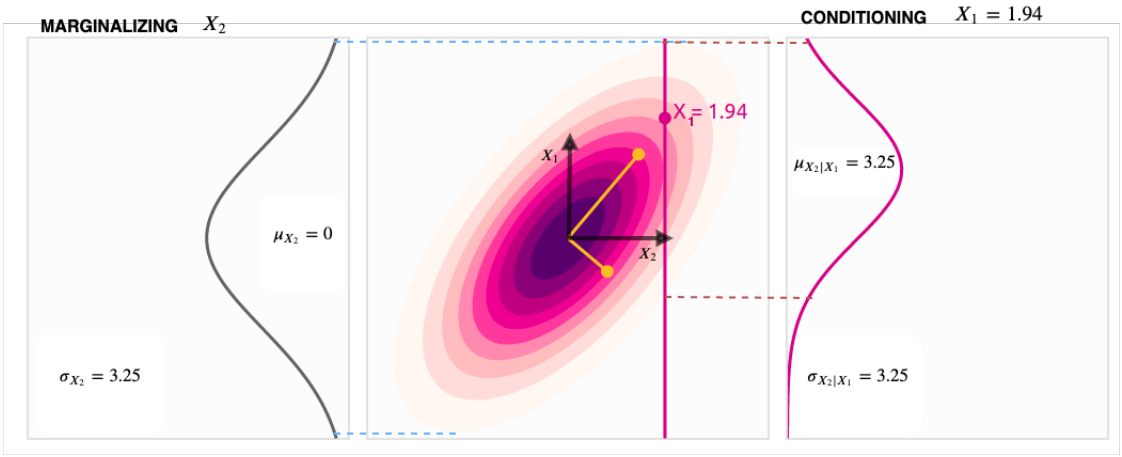


Figure A.2: A joint distribution of X_1 and X_2, created using the visualization tool provided by Görtler et al. (2019). Left: marginalization of X_2 Right: conditioning X_1 at $X_1 = 1.94$.

A.1.2 Conditioning

Conditioning is an operation performed on Gaussian distribution to obtain the probability value of a variable given the other. The result of a conditioning operation is also a Gaussian. Conditioning can be visualized as slicing the multivariate distribution through the condition. The projection obtained while slicing is the conditioned Gaussian. Conditioning of X_1 given X_2 and the vice versa are in Equation:

$$X_1|X_2 \sim \mathcal{N}(\mu_{X_1} + \Sigma_{X_1X_2}\Sigma^{-1}{}_{X_2X_2}(X_2 - \mu_{X_2}), \Sigma_{X_1X_1} - \Sigma_{X_1X_2}\Sigma^{-1}{}_{X_2X_2}\Sigma_{X_2X_1}) \tag{A.6}$$

$$X_2|X_1 \sim \mathcal{N}(\mu_{X_2} + \Sigma_{X_2X_1}\Sigma^{-1}{}_{X_1X_1}(X_1 - \mu_{X_2}), \Sigma_{X_2X_2} - \Sigma_{X_2X_1}\Sigma^{-1}{}_{X_1X_1}\Sigma_{X_1X_2}) \tag{A.7}$$

In the Equation A.6 and A.7, the conditioning variable is required to compute the conditioned μ. However, the computation of the covariance matrix Σ is independent of the variables directly. This dependency is clearly visible from the Figure A.2, where the covariance remains the same and mean shifts.

A.2 Gaussian processes

Considering a multivariate distribution, of a vector with two variables as shown in Equation A.3, the Equation can be simplified by taking vector X with mean vector as μ and covariance matrix as Σ. To obtain a representation like in Equation A.1.
Considering a univariate Gaussian distribution, any sample in the distribution can be represented as the sum of their mean and standard deviation, where standard deviation is the square root of variance. Now, to obtain such a representation in the case of a multivariate distribution, the Equation A.1 can be represented as in Equation A.8.

$$X \sim \mu + L\mathcal{N}(0, I) \tag{A.8}$$

In the Equation A.8, L is the root of the covariance matrix, which is then multiplied to a Gaussian with zero mean and covariance as the identity matrix. This is a case of representing several linear equations in the form of vectors and matrices. The elegance of this simplification lies in obtaining the root of the covariance matrix Σ, represented as L, where L is denoted as the *Cholesky*Dereniowski and Kubale (2003) of the covariance matrix Σ.
From *Cholesky decomposition*, it can be stated that, for any $n \times n$ symmetric positive definite matrix A, there exists a lower triangular matrix L such that L times its transpose gives the matrix itself. As shown in Equation A.9.

$$A = LL^T \;\; i.e., \;\; \Sigma = LL^T \tag{A.9}$$

The discussion above demonstrates a representation of any sample in a multivariate distribution. However, something more interesting is to obtain a model of a function $f(X)$ for any given X in the distribution. Then, the multivariate distribution over the functions can be represented by the Equation A.10.

$$F \sim \mathcal{N}(\mu, \mathrm{K}) \tag{A.10}$$

Where, in the Equation A.10, F is the vector of functions, μ is the mean of the distribution of functions and K is the covariance matrix. Given a function and the sample, the trick to create the joint model to find the relation between each term in the function vector. Generally, the covariance matrix that measures the similarity between the terms can be calculated using the covariance function called a *kernel*. The idea behind using such functions is from the logic that the value of the function of nearby samples from the distribution will be similar, and that of far away will be less correlated. There are several kernels (covariance functions) that can be used to approximate the covariance matrix based on the complexity of the function to be modeled.

B Design of polygon mapping algorithm

This section gives step-wise description of the polygon mapping algorithm. This algorithm connects the scanned points into a polygon map. When the robot starts to move, the measurements obtained at new instances will be stitched to the old polygon by performing a *union* operation. Even though this appears to be straight forward, a problem with this approach is that the LIDAR measurements called scans obtained at each instance are not entirely aligned. This misalignment is due to the error in localization due to drift in odometry and can result in the map as shown in Figure B.1.

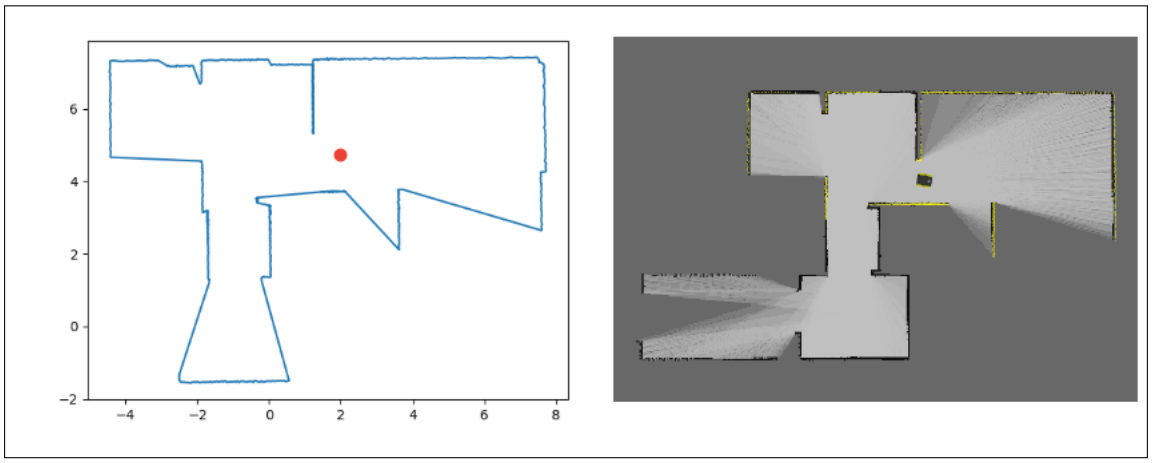


Figure B.1: Polygon map generated without using a SLAM correction (left). Occupancy grid map representation of the same environment built by the robot using a SLAM algorithm (right).

Hence to correct this issue, simultaneous localization and mapping (SLAM) algorithm are essential to correct the robot's pose relative to the measurements obtained. The SLAM discussed in Section 2.2 focuses on providing a general idea of estimating the robot's pose by matching the landmarks observed. Hence, for each new measurement, the point cloud should be aligned with the previous point cloud, and the pose of the robot should be corrected. An approach called iterative closest point (ICP) (Segal et al., 2009) algorithm is commonly used to obtain transformations between new LIDAR scan and the previous. The steps in ICP is given in the next section.

Steps in ICP algorithm

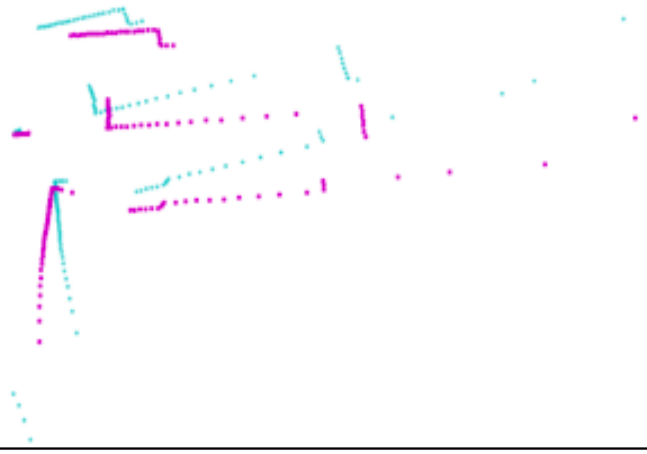
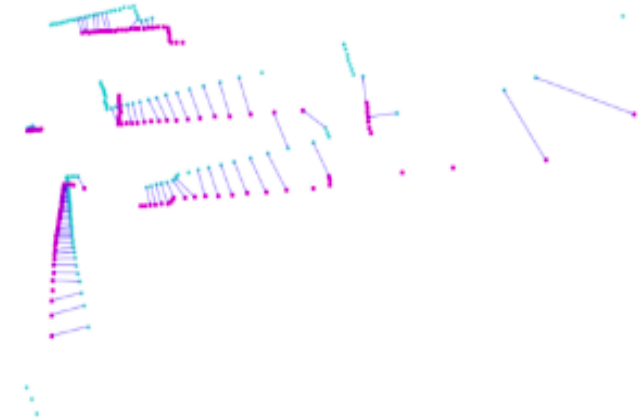
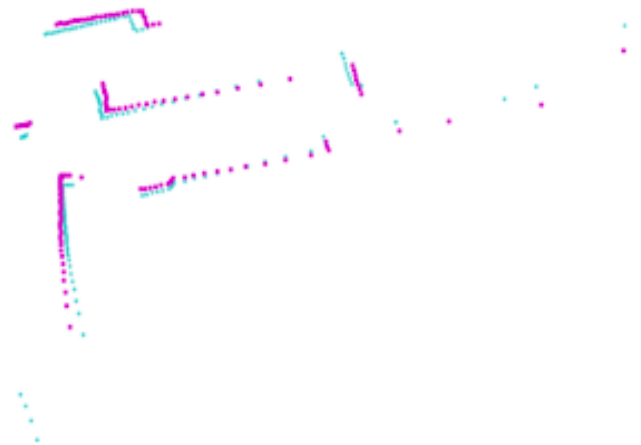
Association	
	*Association* is the process of identifying points in the new scans that correspond to points in the previous scan and are usually achieved using a nearest neighbor search. Hence this operation pairs up the points from both the scans. The cyan color represents the new scan and the purple the previous.
Transformation	
	*Transformation* is the process of identifying the transformation between the two scans so that the mean square error between the pairs is minimum. Moreover, the transformation that minimizes the error corresponds to the correction factor of the robot's pose, and hence, the scans can be nearly aligned. The line connecting the associated points of two scans visualizes the transformation needed.
Error evaluation	
	*Error evaluation* is the process of comparing the two nearly aligned scans to find out the error between them. This error is minimized below a particular threshold value by iteratively repeating the steps. Hence, the accuracy of the prediction is improved. The purple and cyan scans during the iteration visualizes the correction.

Table B.1: Three steps in an iterative closest point algorithm. The visualization is adapted from Kramer,(2019).

B.1 Point cloud filtering and down-sampling

The LIDAR scan measurements have sensor noise, which produces an error in their range measurements. The 2D point cloud containing the noise should to be filtered for obtaining the required polygon map. Even though there are several filter techniques in 3D, such techniques do not work effectively for 2D. Moreover, maintaining the geometrical structure is also a challenge. Hence to tackle this problem, rather than filtering, a *curve simplification* approach is more suitable. Therefore a general curve simplification algorithm called *Douglas–Peucker algorithm* (Visvalingam and Whyatt, 1990) is used to filter the point cloud by simplifying the connection between them.
This algorithm uses a simplification factor γ to determine the smoothness of the curve. Which executes piecewise evaluation that divides the entire set of points, top-down iteratively. Further, the points less than the simplification threshold in each segment are eliminated. Hence each division is a simplified segment, and a lower simplification factor increases the smoothness. However, the simplification operation should also ensure the reduction of the number of points in the cloud. The cloud has a considerable number of points, which increases the computational requirement and creates memory issues. Hence the curve simplification also should down-sample the cloud by increasing the simplification factor. Hence a trade-off between the number of points and the simplification quality is necessary. After several trials, a value of 0.04 was obtained, to work well in down-sampling and still succeeds in simplifying the curve without losing much of the geometric information.
Given a set of boundary points that are obtained after filtering the point cloud, a polygon can

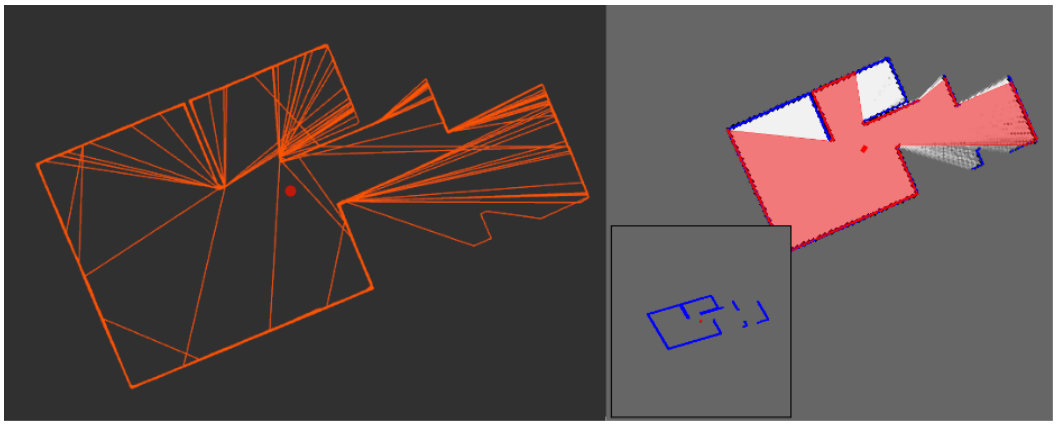


Figure B.2: Left: polygon map formed using disordered point cloud registration. Right: occupancy grid map of the same environment, red region indicating the robot's current scan.

be constructed by connecting them. However, a valuable property that needs to be satisfied is the order of the points, which maintains the polygon's structure. The point cloud gets updated every time a new measurement is available. This also means that two nearby *points can be registered at different instances of time* and need not follow the order. A polygon constructed by connecting the points is shown in Figure B.2.
To address the problem faced in Figure B.2, the polygon generated at every instance is stitched to the polygon obtained at the next measurement by performing a *union* operation. This continuous stitching ensures that the lines passing through the interior of the polygon are eliminated.
Further, since the scans have noisy range measurements, there is still a high chance for *erosion* of some areas of the map, where erosion indicates the failure to construct a particular part of the map due to the noise in the point cloud. This effect is visible when the robot scans both sides of the walls. To prevent this from happening, a *negative buffer* β is added to the polygon map. The buffer is a small region around the boundary of the polygon, with a width equal

to the buffer value. A negative buffer causes the region to be inside the polygon and hence compresses the actual boundary by the buffer value. Even though the buffer parameter cannot eliminate the effect of erosion, it successfully reduces its occurrence. The polygon map constructed using the proposed method, and the effect of the buffer parameter is shown in Figure B.3. The polygon mapping algorithm is showed in Algorithm 6. The proposed polygon mapping algorithm aims at creating a continuous relation between the discrete set of scan points. This implementation is imagined as correlating the occupied cells of an OG map.

Algorithm 6: polygonMapping

Input: P: Point cloud ; `// Aligned point cloud from ICP SLAM`
Output: $\bar{P}$: Polygon map
$k \leftarrow 0$: time step;
$\beta \leftarrow -0.25$: Buffer value ;
$\gamma \leftarrow 0.04$: Simplification factor ; `// Simplify polygon edges less than` γ
$P_{init} \leftarrow P$
$S_{init} \leftarrow$ LineSimplification(P_{init}, γ); `// Filter, down-sampling (Visvalingam and Whyatt, 1990)`
$S_{init\_buffer} \leftarrow$ applyBuffer(S_{init}, β); `// Pad the polygon map with` β
while $k > 0$ **do**
  $S_k \leftarrow$ LineSimplification(P, γ)
  $S_{k\_buffer} \leftarrow$ applyBuffer(S_k, β)
  $\bar{P} \leftarrow S_{k\_buffer} \cup S_{init\_buffer}$
  $S_{init\_buffer} \leftarrow$ LineSimplification$(\bar{P}, \gamma)$;
  $k = k + 1$;
end

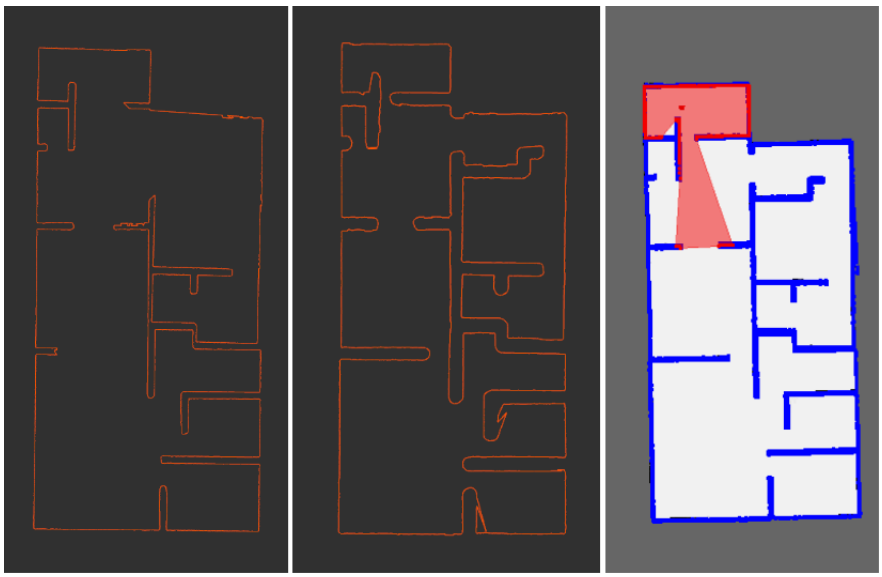


Figure B.3: Polygon map using the proposed algorithm and the effect of buffer parameter. (a) polygonal map without a buffer parameter. (b) polygonal map with a buffer value β = 0.25, which is less prone to erosion. (c) occupancy grid map, red: robot's current scan.

Representing the wall boundaries in the form of a polygon makes it easier to evaluate the relationship between the polygon and the points enclosed by it using geometric techniques. The developed polygon map is a tool that will be used to comprehend spatial relations to detect the viewpoints.

C Iterative closest point SLAM formulation

The ICP SLAM algorithm starts by finding the transformation between two consecutive point cloud measurements. For two consecutive point clouds $P = \{p_1, p_2, ..p_n\} \in \mathbb{R}^{2n}$ and $Q = \{q_1, q_2, ..q_n\} \in \mathbb{R}^{2n}$, the aim is to find a translation t_e and rotation R_e, such that the sum of the squared error $E(R_e, t_e)$ between the point cloud is minimized.
The translation t_e is found using the difference between the center of mass of each point cloud, given by points p_{com} and q_{com}, which is calculated as in Equation C.1.

$$p_{com} = \frac{1}{n}\sum_{i=1}^{n} p_i, \;\; q_{com} = \frac{1}{n}\sum_{i=1}^{n} q_i$$
$$t_e = p_{com} - q_{com} \tag{C.1}$$

To obtain the rotation matrix, consider a matrix A such that the decomposition of A can be written as in Equation C.2.

$$A = (p_i - p_{com})(q_i - q_{com})^T = UWV^T \tag{C.2}$$

Given the decomposition, the rotation matrix R_e can be found using singular value decomposition (SVD), given by Equation C.3.

$$R_e = UV^T \tag{C.3}$$

Once both R_e and t_e are obtained, the homogeneous transformation $\tilde{T}$ between point clouds P and Q can be found by running optimization to minimize the squared error as shown in Equations C.4 and C.5.

$$E(e_e, t_e) = \frac{1}{n}\sum_{i=1}^{n} ||p_i - R_e q_i - t_e||^2 \tag{C.4}$$

$$\tilde{T} = \underset{R_e, t_e}{\operatorname{argmin}}\{E(R_e, t_e)\} \in SE(3) \tag{C.5}$$

The obtained transformation can align the two point clouds to form a merged point cloud $P = \{p_1, p_2 ... p_m\}$ of m points. Similarly, the pose of the robot also should be corrected based on the measured point clouds.
Consider that, point cloud P is measured at a sensor pose $s_i \in \mathbb{R}^6$ and Q at $s_j \in \mathbb{R}^6$, represented in global frame. Then a graph SLAM can be formulated with each sensor pose representing a node on the graph and the edges represents the motion constraint. The SLAM algorithm starts by predicting the transformation between the sensor's pose, and then the measurements are used to update the prediction made. The predicted transformation represented by the twist $T_{ij} \in \mathbb{R}^6$ is the expected relative transformation between two point cloud measurements as given in Equation C.5. Twist allows minimal representation suitable for optimization compared to homogeneous transformation. Therefore, a motion composition operator $\oplus$ is used to concatenate two sensor poses using the transformation as given in Equation C.6.

$$s_j = s_i \oplus T_{ij} \tag{C.6}$$

The observation is given by the odometry data $o_{ij} \in \mathbb{R}^6$ which is the relative motion between the two sensor pose. To define the optimization setting an error function e_{ij}(x) between the predicted and measured transformation is defined, where x = $\{s_1, s_2, ..s_k\} \in \mathbb{R}^{6k}$, is the sensor state vector. The error can be computed using the motion composition operator $\ominus$ as shown in Equation C.7.

$$e_{ij} = o_{ij} \ominus T_{ij} \tag{C.7}$$

The error is distributed normally with covariance Σ to account for the observation noise.

$$e_{ij} \sim \mathcal{N}(0, \Sigma_{ij}) \tag{C.8}$$

The aim is to obtain the state vector x_*, such that the observation error is minimized. However, to perform minimization, a better option is to compute the negative log-likelihood L and minimize its summation over all the motion steps taken by the robot, as shown in Equation C.9.

$$L_{ij}(\mathrm{x}) = -\log p(e_{ij}) \quad \in \mathbb{R} \tag{C.9}$$

For all the sensor poses in the state vector x, the pose that minimizes $L_{ij}(\mathrm{x})$ is given in Equation C.10 and C.11.

$$\mathrm{x}_* = \underset{x}{\operatorname{argmin}} \sum_{ij} L_{ij}(\mathrm{x}) \tag{C.10}$$

$$\mathrm{x}_* = \underset{x}{\operatorname{argmin}} \sum_{ij} e_{ij}(\mathrm{x})^T \Sigma_{ij}^{-1} e_{ij}(\mathrm{x}) \tag{C.11}$$

The minimization can be solved by non-linear optimization techniques like Gauss-Newton or Levenberg-Marquardt to estimate the state vector $\mathrm{x}_* \in \mathbb{R}^{6k}$, which gives the robot's position at any instance. Moreover, the measurements are also concatenated to update the polygon map at every step.

D Methods to detect explorable regions

D.1 Wavefront frontier detection

A wavefront frontier detection (WFD) algorithm developed by Keidar and Kaminka (2014), implements two levels of breadth-first search (BFS*)algorithm. The BFS is a graph-search algorithm that is used to find the shortest paths. The search proceeds with the starting node and evaluates all accessible node from it, the search continuous by repeating this until the evaluation of all the available nodes. Keider and Kaminka considered the grids in an occupancy grid map as the nodes. He evaluated all the unoccupied cells in the observed subspace until he finds a frontier cell. A frontier cell is distinguished by the neighbors, which would be classified as free on one side and unknown on the other. After detecting the frontier point, the algorithm executes another BFS that progresses along the frontiers to identify the entire frontier segment. These segments will lead to more exploration.

D.2 Fast frontier detection

Fast frontier detection (FFD) is another algorithm for detecting exploration doorways developed by Keidar and KaminkaKeidar and Kaminka (2014). The FFD algorithm is not implemented on the map representations, besides it uses the laser scans measurements. The laser scans are used to build a contour by implementing a line drawing algorithm known as Bresenham'swik (2019). Bresenham's algorithm is used to approximate a line segment through a given set of points. These selected contours are stored and updated to accomplish continuous frontier detection.

D.3 Information based detection

In an occupancy grid composition, the cells are valued based on their probability of occupancy. These values update continuously on obtaining new measurements. Moreover, in a LIDAR, the beams will be denser in regions close to the sensor and sparse towards the maximum measurable range. Hence the laser beams that do not hit any obstacle will have an uncertain estimate in the region near the maximum range. Such beams usually cannot make a conclusive decision on their occupancy. Hence a probability of occupancy is considered, such that the explorable region will have an occupancy probability of around 0.5.

*BFS: https://en.wikipedia.org/wiki/Breadth-first_search

E Additional simulation and results

E.1 Understanding the effect of length scale

E.1.1 RBF kernel for different length scale

The example considered to understand the effect of the length scale is shown in Figure E.1.

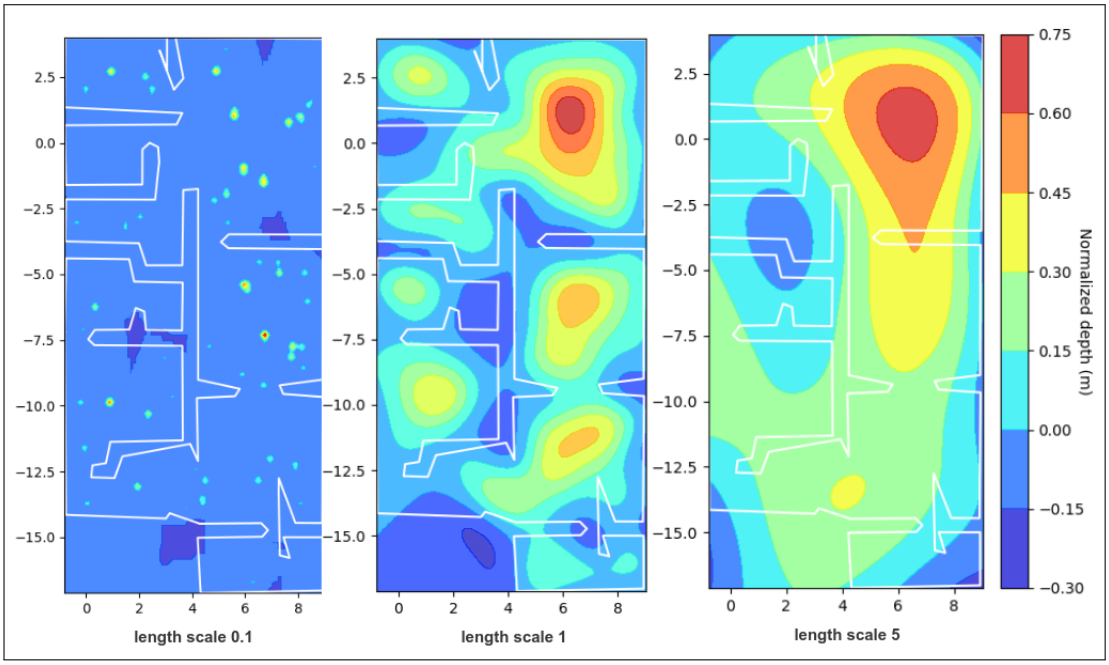


Figure E.1: GP posterior showing the effect of length scales 0.1, 1, and 5 on an RBF kernel for 100 training samples.

A short length scale of 0.1, a length scale of 1, and a long length scale of 5 for 100 training samples are used in the example. When the length scale is 0.1, very close by samples are correlated. However, there are almost no nearby samples when 100 training samples are used. Hence, the GP has peaked at only distinct points where it has been sampled. For a length scale 1, it can be seen that the change in the correlation of samples across the walls is sufficiently captured. However, for a longer length scale of 5, even the samples from two different rooms with similar mean are related. The spacious rooms have more samples with high mean and less number of samples with low mean. Therefore a smooth function like RBF correlates the dominant mean within the range of the length scale, which fails to model the walls. However, it is the vice-versa for the region with smaller rooms, hence the central region of the rooms with higher mean are not modeled.

E.1.2 Matern52 kernel for different length scales

Figure E.2 illustrates an example used to understand the effect of different length scale for the same number of training samples on a Matern52 kernel.

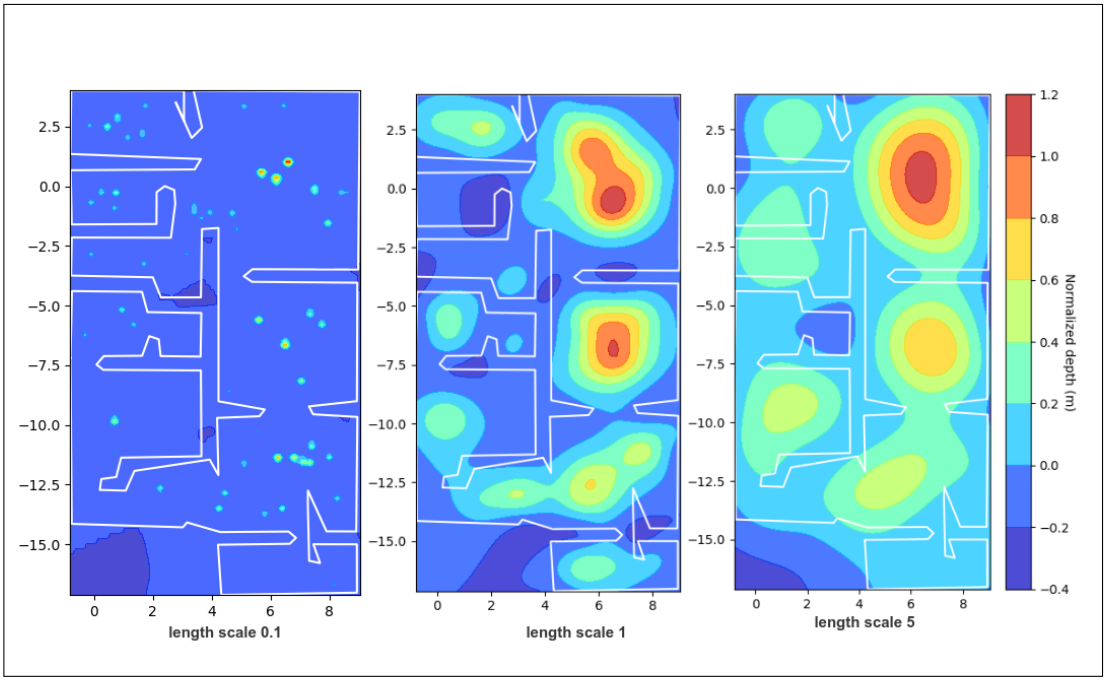


Figure E.2: GP posterior showing the effect of length scales 0.1, 1, and 5 on Matern52 kernel for 100 training samples.

The figure shows that for 100 samples, the length scale of 0.1 and 1 performs similar to RBF. However, for a length scale 5, the reduced smoothness of Matern52 kernel is beneficial to capture a more significant number of different samples within a given length scale. This supports the deduction that the Matern52 kernel can model varying spatial data.

E.2 Simulation of exploration using SCEAM and iSCEAM algorithms

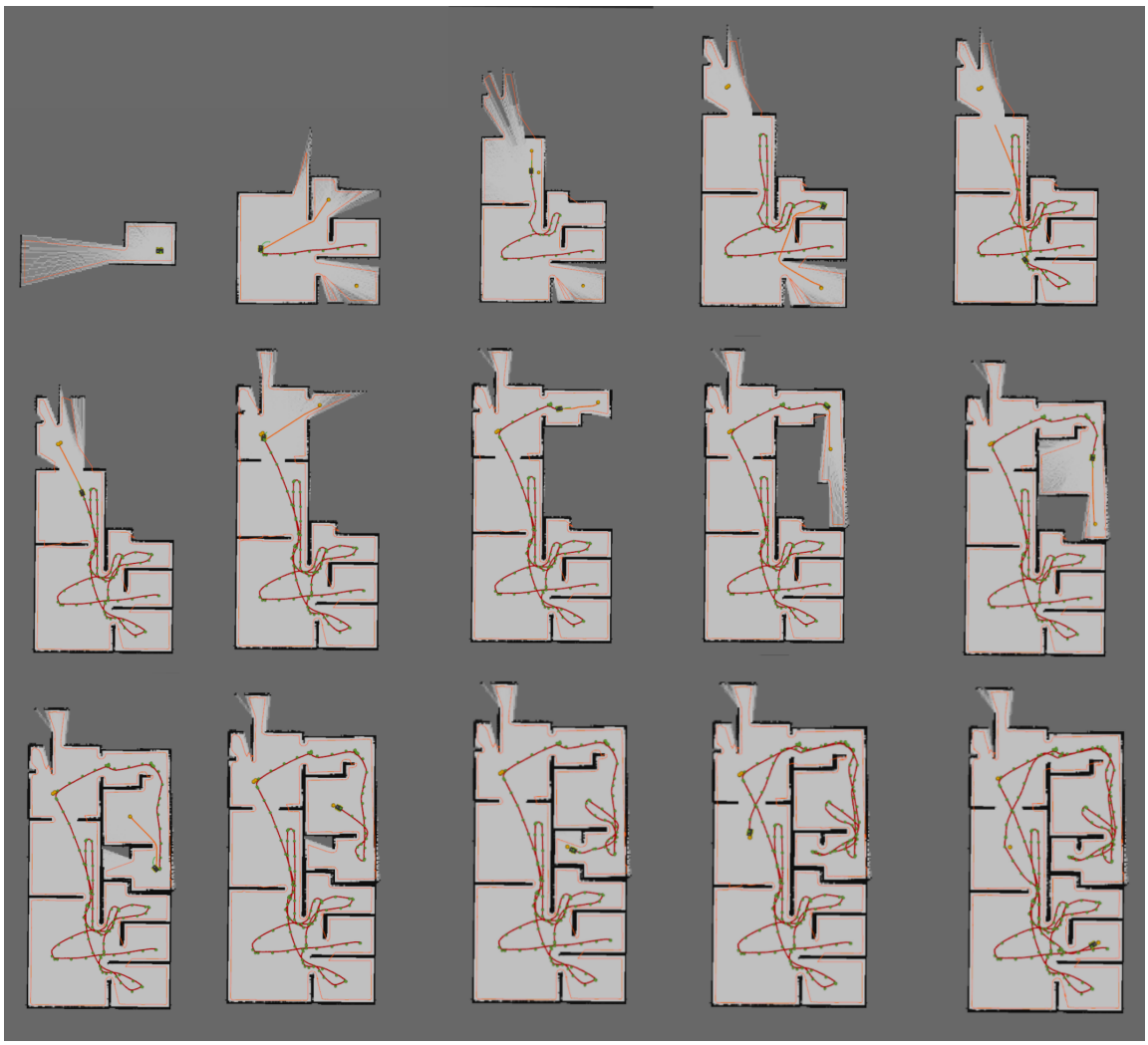


Figure E.3: Exploration using SCEAM algorithm. Red line: robot trajectory, green markers: saved trajectory nodes used for remembering the visited region, orange line: path planned, yellow: goal selected.

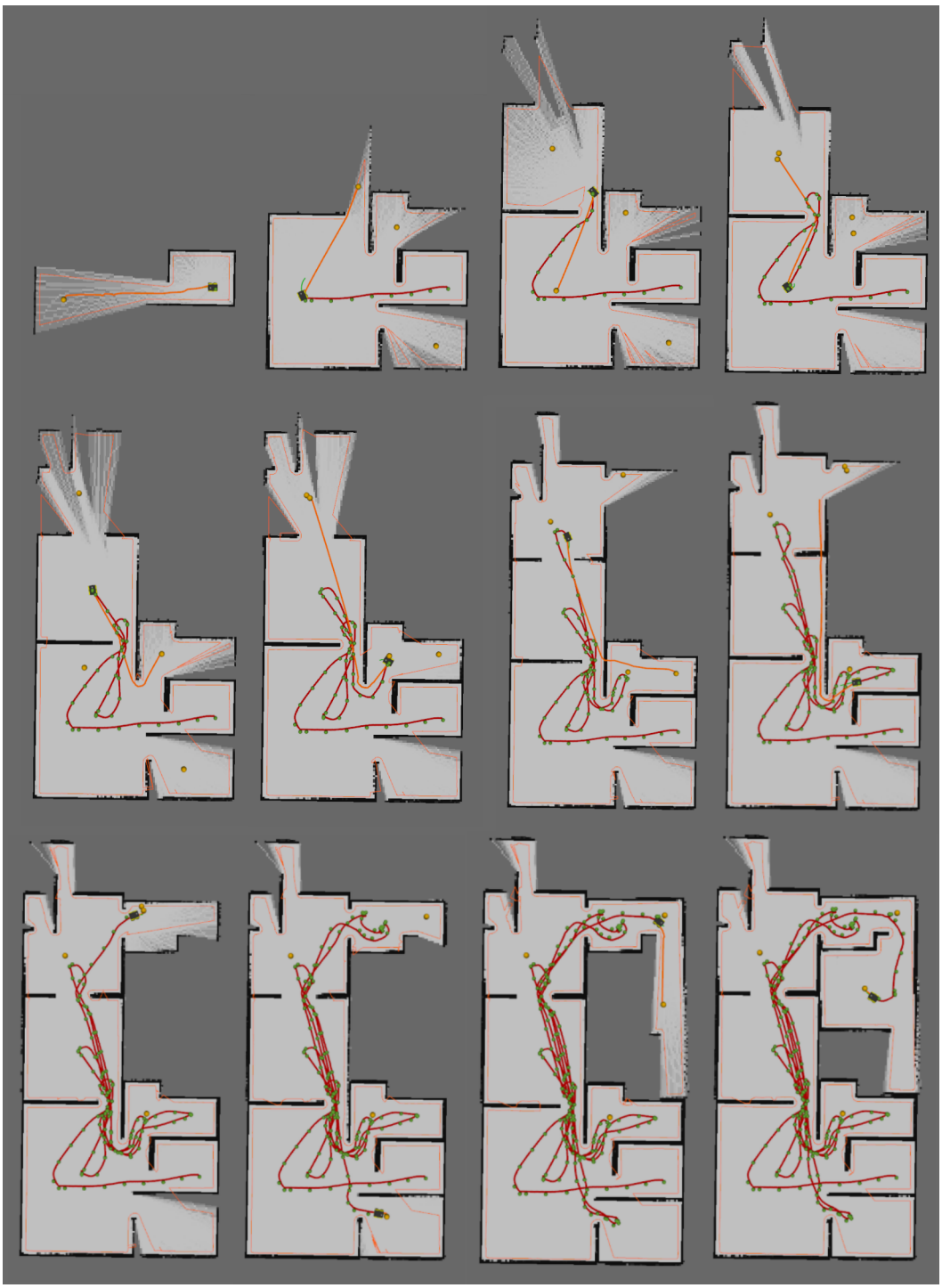


Figure E.4: Exploration using iSCEAM algorithm. Red line: robot trajectory, green markers: saved trajectory nodes used for remembering the visited region, orange line: path planned, yellow: goal selected.

E.3 Simulation of exploration in different environments

The SCEAM and iSCEAM algorithms are simulated to explore in five environments. Figure E.5 shows the exploration trajectory of the developed algorithms along with the baseline algorithm that demonstrated best performance in each of the environments.

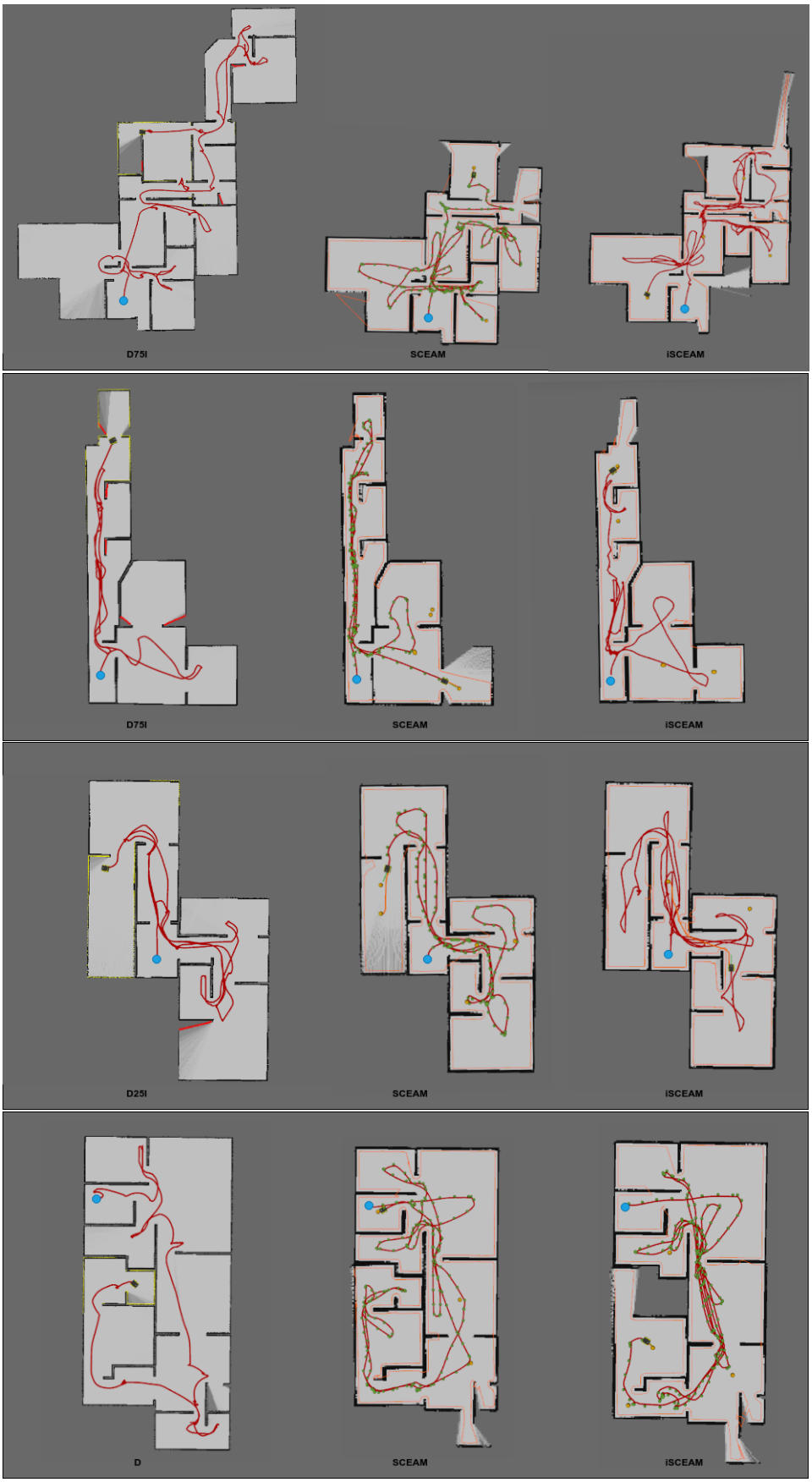


Figure E.5: Comparison of exploration trajectory (starting: blue dot) with the baseline in Environments 2,3,4 and 5.

E.4 Resource utilization

In this section, the CPU and memory usage of the developed algorithms are illustrated. The resource utilization of the SCEAM and iSCEAM algorithms are shown in Figure E.6 and E.7.

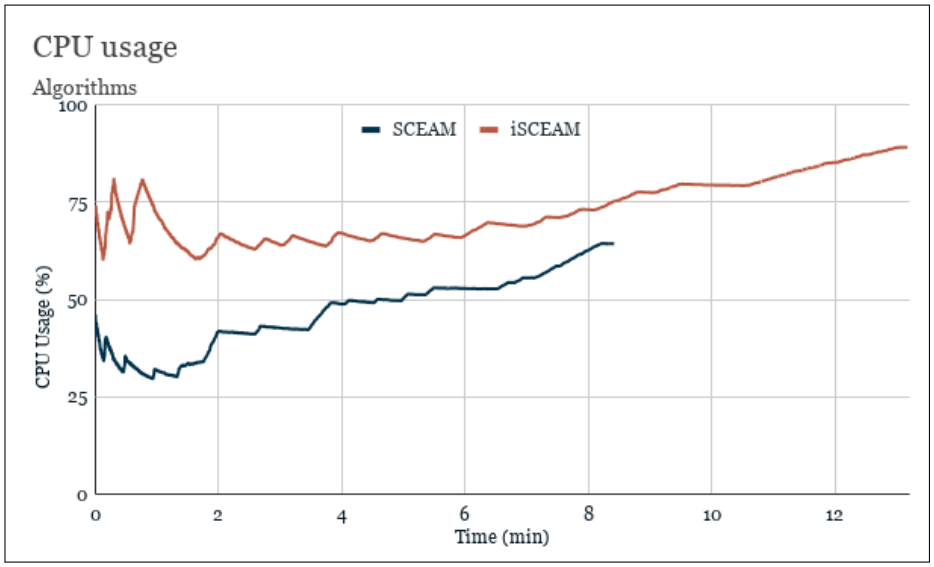


Figure E.6: CPU usage of the developed algorithms during exploration.

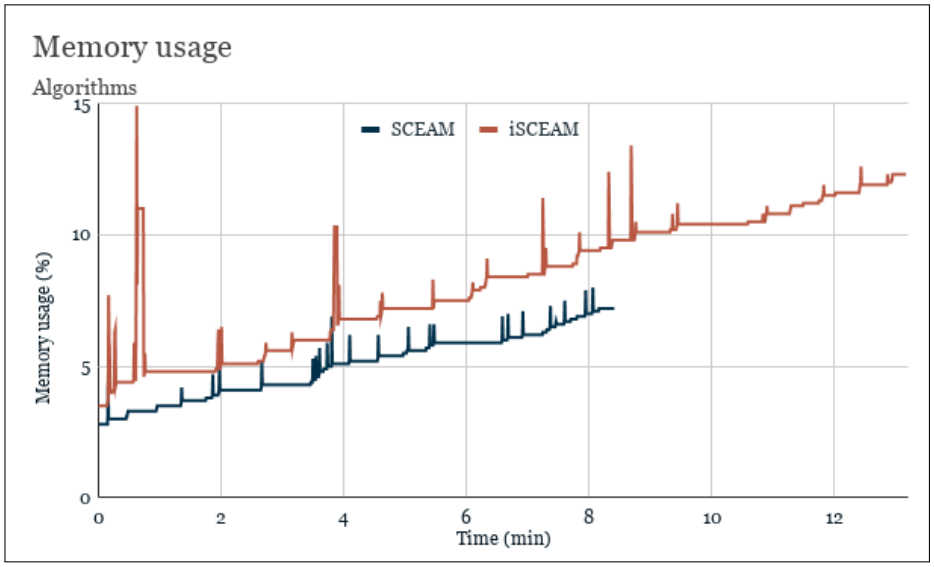


Figure E.7: Memory usage of the developed algorithms during exploration.

The additional modules that enables exploration includes the polygon mapping module and the trajectory node, which utilise resources as shown in Figure E.8 and E.9.

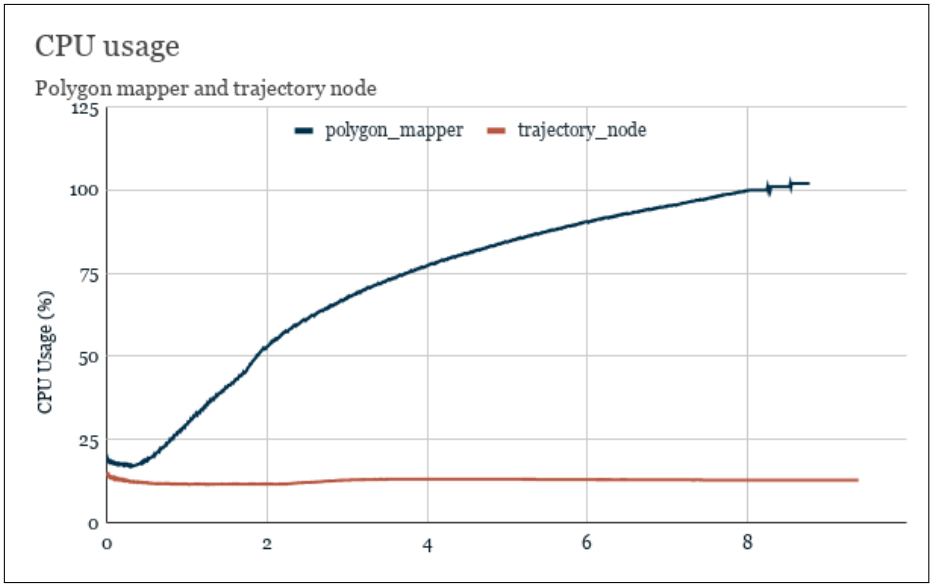


Figure E.8: CPU usage of trajectory node and polygon mapping module.

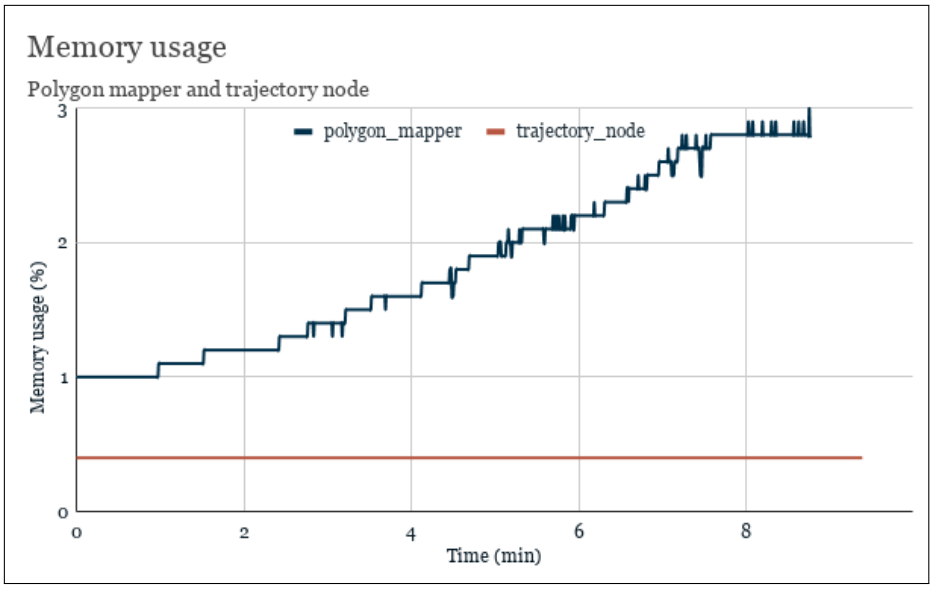


Figure E.9: Memory usage of trajectory node and polygon mapping module.

Bibliography

(2019), Bresenham's line algorithm.
https://en.wikipedia.org/wiki/Bresenham's_line_algorithm

Bai, S., J. Wang, F. Chen and B. Englot (2016), Information-theoretic exploration with Bayesian optimization, *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1816–1822.

Ben-Ari, M. and F. Mondada (2018), *Robotic Motion and Odometry*, Springer International Publishing, Cham, pp. 63–93, ISBN 978-3-319-62533-1, doi:10.1007/978-3-319-62533-1_5.
https://doi.org/10.1007/978-3-319-62533-1_5

Borrmann, D., A. Nüchter, M. Đakulović, I. Maurović, I. Petrović, D. Osmanković and J. Velagić (2014), A mobile robot based system for fully automated thermal 3D mapping, **vol. 28**, no.4, pp. 425 – 440, ISSN 1474-0346, doi:https://doi.org/10.1016/j.aei.2014.06.002.
http://www.sciencedirect.com/science/article/pii/S1474034614000408

Bourgault, F., A. A. Makarenko, S. B. Williams, B. Grocholsky and H. F. Durrant-Whyte (2002), Information based adaptive robotic exploration, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pp. 540–545 vol.1, doi:10.1109/IRDS.2002.1041446.

Brzezinski, J. R. and G. J. Knafl (1999), Logistic regression modeling for context-based classification, in *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*, pp. 755–759, doi:10.1109/DEXA.1999.795279.

Cadena, C., L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid and J. J. Leonard (2016), Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age, **vol. 32**, no.6, pp. 1309–1332, ISSN 1552-3098, doi:10.1109/TRO.2016.2624754.

Cadena, C., L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid and J. J. Leonard (2016), Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age, *CoRR*, **vol. abs/1606.05830**.
http://arxiv.org/abs/1606.05830

Charrow, B., G. Kahn, S. Patil, S. Liu, K. Y. Goldberg, P. Abbeel, N. Michael and V. Kumar (2015), Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping, in *Robotics: Science and Systems*.

Colby, C. (2009), Spatial Cognition, in *Encyclopedia of Neuroscience*, Ed. L. R. Squire, Academic Press, Oxford, pp. 165 – 171, ISBN 978-0-08-045046-9, doi:https://doi.org/10.1016/B978-008045046-9.01120-7.
http://www.sciencedirect.com/science/article/pii/B9780080450469011207

Dereniowski, D. and M. Kubale (2003), Cholesky Factorization of Matrices in Parallel and Ranking of Graphs, pp. 985–992, doi:10.1007/978-3-540-24669-5_127.

Dissanayake, G., P. Newman, S. Clark, H. F. Durrant-Whyte and M. Csorba (2001), A solution to the simultaneous localization and map building (SLAM) problem, *IEEE Trans. Robotics and Automation*, **vol. 17**, pp. 229–241.

Elfes, A. (1989), Using occupancy grids for mobile robot perception and navigation, **vol. 22**, no.6, pp. 46–57, ISSN 0018-9162, doi:10.1109/2.30720.

Elfes, A. (1989), Using Occupancy Grids for Mobile Robot Perception and Navigation, *Computer*, **vol. 22**, pp. 46 – 57, doi:10.1109/2.30720.

Francis, G., L. Ott and F. Ramos (2018), Functional Path Optimisation for Exploration in Continuous Occupancy Maps, *CoRR*, **vol. abs/1805.01079**.
http://arxiv.org/abs/1805.01079

Gupta, R. (2019), rochakgupta/aco-tsp.
https://github.com/rochakgupta/aco-tsp

Görtler, J., R. Kehlbeck and O. Deussen (2019), A Visual Exploration of Gaussian Processes, *Distill*, doi:10.23915/distill.00017,
https://distill.pub/2019/visual-exploration-gaussian-processes.

Holz, D., N. Basilico, F. Amigoni and S. Behnke (2010), Evaluating the Efficiency of Frontier-based Exploration Strategies, pp. 1 – 8.

Htun and Thi (2018), A Survey Review on Solving Algorithms for Travelling Salesman Problem (TSP), *International Journal of Scientific and Research Publications (IJSRP)*, **vol. 8**, doi:10.29322/IJSRP.8.12.2018.p8481.

Hörner, J. (2016), Map-merging for multi-robot system.
https://is.cuni.cz/webapps/zzp/detail/174125/

Icp, M. (2013), icp-slam.
https://www.mrpt.org/list-of-mrpt-apps/application-icp-slam/

Iocchi, L. and S. Pellegrini (2009), Building 3D maps with semantic elements integrating 2D laser, stereo vision and IMU on a mobile robot.

Jadidi, M. G., J. V. Miró and G. Dissanayake (2016), Gaussian processes autonomous mapping and exploration for range-sensing mobile robots, *Autonomous Robots*, **vol. 42**, pp. 273–290.

Jadidi, M. G., J. V. Miró, R. Valencia and J. Andrade-Cetto (2014), Exploration on continuous Gaussian process frontier maps, in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6077–6082, ISSN 1050-4729, doi:10.1109/ICRA.2014.6907754.

Javaid, A. (2013), Understanding Dijkstra Algorithm, *SSRN Electronic Journal*, doi:10.2139/ssrn.2340905.

Kapoor, A., K. Grauman, R. Urtasun and T. Darrell (2010), Gaussian Processes for Object Categorization, **vol. 88**, no.2, pp. 169–188, doi:10.1007/s11263-009-0268-3.
https://doi.org/10.1007/s11263-009-0268-3

Keidar, M. and G. A. Kaminka (2014), Efficient frontier detection for robot exploration, **vol. 33**, no.2, pp. 215–236, doi:10.1177/0278364913494911.
https://doi.org/10.1177/0278364913494911

Khairuddin, A. R., M. S. Talib and H. Haron (2015), Review on simultaneous localization and mapping (SLAM), *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pp. 85–90.

Khatri, K. (2014), A Survey Paper On Solving TSP using Ant Colony Optimization on GPU.

Kiefer, J. and J. Wolfowitz (1952), Stochastic Estimation of the Maximum of a Regression Function, **vol. 23**, no.3, pp. 462–466, doi:10.1214/aoms/1177729392.
https://doi.org/10.1214/aoms/1177729392

Kim, H.-C. and J. Lee (2007), Clustering Based on Gaussian Processes, **vol. 19**, no.11, pp. 3088–3107, doi:10.1162/neco.2007.19.11.3088.
https://doi.org/10.1162/neco.2007.19.11.3088

Kim, P., J. Chen and Y. Cho (2018), SLAM-driven robotic mapping and registration of 3D point clouds, *Automation in Construction*, **vol. 89**, pp. 38–48, doi:10.1016/j.autcon.2018.01.009.

Kramer, A. (2019), colin Archives.
http://andrewjkramer.net/tag/colin/

Kurazume, R., S. Oshima, S. Nagakura, Y. Jeong and Y. Iwashita (2017), Automatic large-scale three dimensional modeling using cooperative multiple robots, *Computer Vision and Image Understanding*, **vol. 157**, pp. 25 – 42, ISSN 1077-3142, doi:https://doi.org/10.1016/j.cviu.2016.05.008, large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans.
http://www.sciencedirect.com/science/article/pii/S1077314216300558

Mahrami, M., M. N. Islam and R. Karimi (2013), Simultaneous Localization and Mapping: Issues and Approaches, *International Journal of Computer Science and Telecommunications*, **vol. 4**, p. 1.

Maneewongvatana, S. and D. M. Mount (1999), Analysis of approximate nearest neighbor searching with clustered point sets, *CoRR*, **vol. cs.CG/9901013**.
http://arxiv.org/abs/cs.CG/9901013

Meng, Z., H. Qin, Z. Chen, X. Chen, H. Sun, F. Lin and M. H. A. Jr (2017), A Two-Stage Optimized Next-View Planning Framework for 3-D Unknown Environment Exploration, and Structural Reconstruction, **vol. 2**, no.3, pp. 1680–1687, ISSN 2377-3774, doi:10.1109/LRA.2017.2655144.

Menger, K., E. Dierker, K. Sigmund, J. Dawson, R. Engelking and W. Hildenbrand (1998), *Ergebnisse Eines Mathematischen Kolloquiums*, Springer, ISBN 9783211831045.
https://books.google.nl/books?id=V9uew6_F9VQC

Murphy, K. P. (2000), Bayesian Map Learning in Dynamic Environments, in *Advances in Neural Information Processing Systems 12*, Eds. S. A. Solla, T. K. Leen and K. Müller, MIT Press, pp. 1015–1021.
http://papers.nips.cc/paper/1716-bayesian-map-learning-in-dynamic-environments.pdf

Newman, P., J. J. Leonard, J. D. Tardós and J. Neira (2002), Explore and return: experimental validation of real-time concurrent mapping and localization, *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, **vol. 2**, pp. 1802–1809 vol.2.

Prieto, S., B. Quintana, A. Adán and A. Vázquez (2017), As-is building-structure reconstruction from a probabilistic next best scan approach, *Robotics and Autonomous Systems*, **vol. 94**, pp. 186 – 207, ISSN 0921-8890, doi:https://doi.org/10.1016/j.robot.2017.04.016.
http://www.sciencedirect.com/science/article/pii/S092188901530227X

Ramos, F. T. and L. Ott (2016), Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent, *I. J. Robotics Res.*, **vol. 35**, pp. 1717–1730.

Rasmussen, C. and C. Williams (2006), *Gaussian Processes for Machine Learning*, Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, USA.

Rasmussen, C. E. (2004), *Gaussian Processes in Machine Learning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 63–71, ISBN 978-3-540-28650-9, doi:10.1007/978-3-540-28650-9_4.
https://doi.org/10.1007/978-3-540-28650-9_4

Segal, A., D. Hähnel and S. Thrun (2009), Generalized-ICP, doi:10.15607/RSS.2009.V.021.

Shannon, C. E. (1948), A Mathematical Theory of Communication, **vol. 27**, no.3, pp. 379–423, doi:10.1002/j.1538-7305.1948.tb01338.x.
https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x

Shin, H. and S. Lee (2016), An RKHS Approach to Robust Functional Linear Regression, *Statistica Sinica*, **vol. 26**, doi:10.5705/ss.202014.0063.

Stachniss, C., G. Grisetti and W. Burgard (2005), Information Gain-based Exploration Using Rao-Blackwellized Particle Filters, in *Robotics: Science and Systems*.

Stein, M. (2015), *Interpolation of Spatial Data: Some Theory For Kriging*, doi:10.1007/978-1-4612-1494-6.

Thrun, S. (2003), Learning Occupancy Grid Maps with Forward Sensor Models, *Autonomous Robots*, **vol. 15**, pp. 111–127.

Thrun, S., W. Burgard, D. Fox and R. Arkin (2005), *Probabilistic Robotics*, Intelligent Robotics and Autonomous Agents series, MIT Press, ISBN 9780262201629. https://books.google.nl/books?id=k_yOQgAACAAJ

Visvalingam, M. and J. D. Whyatt (1990), The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization, *Comput. Graph. Forum*, **vol. 9**, pp. 213–228.

van der Walt, S., J. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. Warner, N. Yager, E. Gouillart, T. Yu and t. contributors (2014), scikit-image: Image processing in Python, *PeerJ*, **vol. 2**, doi:10.7717/peerj.453.

Woeginger, G. (2001), Exact Algorithms for NP-Hard Problems: A Survey, pp. 185–208, doi:10.1007/3-540-36478-1_17.

Yamauchi, B. (1997), A frontier-based approach for autonomous exploration, in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151, doi:10.1109/CIRA.1997.613851.

Yang, K., S. K. Gan and S. Sukkarieh (2013), A Gaussian process-based RRT planner for the exploration of an unknown and cluttered environment with a UAV, **vol. 27**, no.6, pp. 431–443, doi:10.1080/01691864.2013.756386. https://doi.org/10.1080/01691864.2013.756386

Zhu, D., T. Li, D. Ho, C. Wang and M. Q. . Meng (2018), Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7548–7555, ISSN 2577-087X, doi:10.1109/ICRA.2018.8463213.