ROBOTIC EXPLORATION AND NAVIGATION ON HIERARCHICAL SPATIAL MAPS

by

Kadir Türksoy

B.S., Electrical & Electronics Engineering, 2016

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical & Electronics Engineering

Boğaziçi University

2019

ACKNOWLEDGEMENTS

When I applied for a summer term research program in the Intelligent System Laboratory five years ago, I could have never imagined that this place would become an indispensable part of my life. I have learned many things which changed the way I look and see the world. ISL provided a significant amount of experience to develop a much more clear picture about myself. In that sense, I would like to thank my thesis supervisor, Prof. H. Işıl Bozma, to give me that opportunity at the beginning and also thank for her advisory support during my thesis studies.

I would like to thank also Prof. Yağmur Denizhan and Dr. Ömür Arslan for participating my thesis committee and their constructive comments and suggestion to my thesis. It was a great pleasure to work with such valuable people who have numerous contributions to the research in their field.

There is a long list of people who I have worked with in ISL. I would like to specially thank starting from the old members Dr. Haluk Bayram, Dr. Hakan Karaoğuz, Mahmut Demir, Esen Yel, Berkan Höke, to the current ones, Serhat Işcan, Kadir Cumali, Meriç Durukan, Kemal Bektaş, Berkay Gümüş and Gizem Özdil for their contribution and support during my studies.

Finally, I would like to thank all of my friends, but in particular Serkan Buğur, Samet Keskin and Mert Tiftikçi for having their time and patiently revising all the thesis material. I also especially thank Esranur İşcan for the immediate support in every time I was frustrated. Similarly, I would like to thank my family for their patience and support during the thesis process.

This work has been supported in part by TUBITAK EEAEAG-115E380.

ABSTRACT

ROBOTIC EXPLORATION AND NAVIGATION ON HIERARCHICAL SPATIAL MAPS

This thesis is concerned with topological map building and navigation using these maps. Topological maps are compact spatial representations in which i) the continuous world is modeled as a discrete set of places and their spatial relations; and ii) what is meant by places ranges from particular locations to large spatial regions. This thesis proposes a series of approaches that altogether enable a mobile robot to evolve its knowledge of topological maps and use them in navigating among places. First, navigating reliably between two locations is considered. A reactive strategy that allows the robot to arrive at the goal location with the desired heading is proposed. Next, exploration of unknown environments is addressed and an approach in which the robot determines the nodes of the map through discovering canonical appearances is presented. As such, the robot can build a topological map of its surroundings autonomously. Following, the integration of this knowledge with the existing knowledge in the robot's map memory is considered. As knowledge accumulates, state-of-the-art approaches face real-time applicability issues. This is addressed by proposing an approach based on hierarchical topological maps. The proposed approach assumes that the robot is capable of topological spatial cognition as presented previously. As such, the hierarchical map is constructed so that the level preceding the terminal nodes corresponds to places - where the concept of a *place*, is defined as a collection of appearances or locations sharing common perceptual signatures or physical boundaries. Finally, navigation using the hierarchical map is considered and a novel algorithm that enables the robot to generate a plan of the route between any given two nodes in the hierarchical map is developed. The proposed approaches are implemented on a mobile robot and extensive experimental results are reported.

ÖZET

SIRADÜZENSEL HARİTALARDA ROBOTİK KEŞİF VE HAREKET

Bu tezde topolojik harita oluşturma ve oluşturulan haritalar üzerinde hareket incelenmektedir. Uzamsal ilişkileri verimli biçimde göstermeyi amaçlayan bu haritalarda; i) sürekli olan fiziksel dünya ayrık bir yer kümesi ve yerler arasındaki uzamsal ilişkiler şeklinde modellenir; ii) yerlerin tanımı belli başlı konumlardan büyük ölçekli uzamsal bölgelere kadar değişkenlik gösterebilir. Bu tezde gezgin bir robotun içinde bulunduğu topolojik harita bilgisini geliştirmesi ve bu bilgiyi kullanarak ortamdaki hareketi sağlaması ile ilgili çeşitli yaklaşımlar önerilmektedir. Çalışmada öncelikle iki nokta arasında güvenilir hareket etme problemine odaklanılmıştır. Robotun verilen hedef konum ve doğrultuya ulaşabilmesini sağlayan tepkin bir yaklaşım önerilmiştir. Sonrasında bilinmeyen ortamların keşfine değinilmiş ve ortamdaki özgün görünüşlerin tespiti ile gerçekleştirilen bir haritalama yöntemi önerilmiştir. Bir sonraki adımda ise oluşturulan yeni haritalar robotun daha önceden sahip olduğu bilgi ile tümleştirilmektedir. İşlenen bilgi miktarı biriktikçe güncel yöntemlerin gerçek zamanlı olarak uygulanabilmesi zorlaşmaktadır. Bu durum sıradüzensel topolojik haritaların kullanılması ile iyiliştirilebilmektedir. Önerilen yaklaşımda gezgin robotun daha önceden sunulmuş olan topolojik uzamsal hafıza oluşturabilme yeteneğine sahip olduğu varsayılır. Öyle ki, sıradüzensel haritalar oluşturulurken en alt seviyedeki düğümler topolojik uzamsal hafıza modelinden gelen yerlere karşılık gelir. Burada *yer*, ortak algısal simgeler ya da fiziksel sınırlara sahip görünüş veya konumlar kümesi olarak tanımlanır. Son bölümde ise sıradüzensel haritalar kullanarak hareketin sağlanmasına odaklanılmış ve robotun harita üzerinde verilen iki düğüm arasındaki rotayı planlamasını sağlayan bir algoritma geliştirilmiştir. Önerilen yaklaşımlar gezgin bir robot üzerinde pek çok sayıda deney yapılarak gerçeklenmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$a(c; x_k)$	Accessibility of position c with respect to base point x_k
A	Set of connected component
$\mathcal{A}_k = \{A_{k1}, \ldots, A_{kM}\}$	Set of connected component
$b \in \mathcal{B}$	Point in bubble surface
$\mathcal{B} = R^2 \times S^1 \times \mathcal{F}$	Bubble space
$c \in R^2$	Robot's position
$D(x, \mu)$	Function of robot dynamic
$e_{1,2}$	Unit vectors
E	Set of edges
f	Viewing direction
f_1	Pan angle
f_2	Tilt angle
F	State transition matrix
$\mathcal{F} \subset S^2$	Space of viewing directions
g	Goal position
G_k	Topological map
$h(c, c_g, \alpha_g)$	Heuristic function for A*
H	Hierarchical graph
$k \in \mathcal{K}$	Index of the last node added to the graph
K	Kalman Gain
L	Level
N_I	Descriptor dimension
N_r	Cardinality of $\mathcal{N}$
$\mathcal{N}$	Set of nodes
M	Extend of simulation
o	Obstacle
$\mathcal{O}$	Set of obstacles
$\mathcal{O}_v$	Set of visible obstacles

$\mathcal{O}_r$	Set of recent obstacles
$\mathbb{O}_k$	Open viewing direction
P	Path
$\mathbf{P}$	Set of paths
$\mathcal{P}$	Place
$\mathbb{P}$	Set of Places
r	Number of levels in the hierarchical graph
Q^o	Odometry covariance matrix
Q^l	Laser covariance matrix
$q_i(b,t)$	Sensor reading
t	Time
t^+	Time just after t
s	Node transition function
S_k	Obstructed regions that become visible
$x \in \mathcal{X}_k$	Base point
$\mathcal{X}_k$	Set of base points
$\mathcal{U}_k$	Unexplored viewing direction
$u : A_k \to S^1$	Function of component directions
$v(c; x_k)$	Visibility of position c with respect to base point x_k
$\mathcal{V}_k$	Union of obstructed regions that become visible
w_k	Waypoint
$\mathcal{X}_k$	set of nodes
$\alpha \in S^1$	Robot's heading
α_ν	Bearing of gradient $-\nabla_c \varphi_{w,\theta,\mathcal{O}}(x)$
$\beta_{\mathcal{O}}(x)$	Obstacles' function
$\gamma_{w,\theta} : \mathcal{X} \to \mathbb{R}$	Goal function
δ_p	Safety margin
δ_t	Time step
Δ_t	Time difference
$\epsilon_-, +\epsilon_+ \geq 0$	Boundary parameters

ζ	Partitioning function
$\eta_1 \in \mathbb{R}$	Goal location bias coefficient
$\eta_2 \in \mathbb{R}$	Goal heading bias coefficient
θ	Direction
ι	Edge category function
$\kappa : \mathcal{B} \to \{0, 1\}$	The function of potential exploration directions
λ	Label function
μ	Input
$\nu(A_{ki})$	Pan extension
$\xi(A)$	Utility of a component
$\rho_i \in \mathbb{R}^{\geq 0}$	Radius
ϱ	Edge cost
ς	Level criteria
Σ	Covariance prediction matrix
τ_α	Goal heading threshold
τ_b	Goal bias for A* heuristic function
τ_c	Goal position threshold
τ_d	Discontinuity threshold
τ_I	Similarity threshold
τ_{max}	Maximum depth range which can be observed by the robot
τ_l	Allowable minimum extent
τ_p	Overlap threshold
τ_s	Step size
τ_e	Matching threshold
τ_u	Allowable maximum extent
υ	Linear velocity
Υ	Intersection of to-be-reached and explored locations
$\varphi_{w,\theta,\mathcal{O}} : \mathcal{X} \to R^{\geq 0}$	Potential function
$\psi_k(b_m)$	Extend of a viewing direction

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
AHC	Agglomerative Hierarchical Clustering
APF	Artificial Potential Field
COLD	COsy (Cognitive Systems for Cognitive Assistants) Localization Database
CPU	Central Processing Unit
DFC	Discrete Fourier Coefficient
DWA	Dynamic Windowing Approach
EKF	Extended Kalman Filter
ICP	Iterative Closest Point
ISL	Intelligent System Laboratory
IMU	Inertial Measurement Unit
PMICP	Point Matcher Iterative Closest Point
RA	Reactive Approach
RHPS	Recursive Hierarchical Path Search
ROS	Robot Operating System
SLAM	Simultaneous Localizaiton and Mapping
SLINK	An optimally efficient algorithm for the single-link cluster method
TSC	Topological Spatial Cognition

1. INTRODUCTION

This thesis is concerned with topological mapping and navigation using these maps. With topological mapping, the continuous world is represented compactly by a discrete set of places and their spatial relations. Interestingly, what is meant by a place varies from a particular location to a spatial area to a possibly very large region. Thus, the robot is able to have efficient spatial characterization. If mobile robots are ever to become autonomous, it is crucial for them to be capable of building these maps on their own and using them to navigate as required. Both have proven to be complex tasks as they require the robot to be capable of i) navigating reliably between two points, ii) exploring unknown environments, iii) self-localizing using these maps iii) organizing the acquired knowledge in an efficient manner and iv) using the stored map knowledge in deciding how to navigate between learned places. This thesis proposes a series of approaches that altogether enable a mobile robot to evolve its knowledge of topological maps and use them in navigating among places.

1.1. General Approach

Consider a mobile robot. Assume that the robot is endowed with a camera and laser sensor. Each incoming appearance (image) is encoded internally using bubble space representation [1]. We also assume that the robot's spatial reasoning is based on the topological spatial cognition (TSC) model [2]. The TSC model consists of two parts: spatial memory and processing modules. In the long-term spatial memory, knowledge of places and their spatial relations are retained separately in place and map memories respectively. Both are based on the concept of a *place* that refers to a specific spatial unit or area that encapsulates observations within one's sensory horizon - similar to human's concept [3] or a *region* [4]. In order to decrease the dependency on odometric data, each place is defined by the collection of appearances sharing common perceptual signatures. The processing part enables each robot to detect places, recognize them or learn them as necessary in a completely unsupervised, incremental and organized manner. Throughout this processing, it continually builds up and uses its long-term

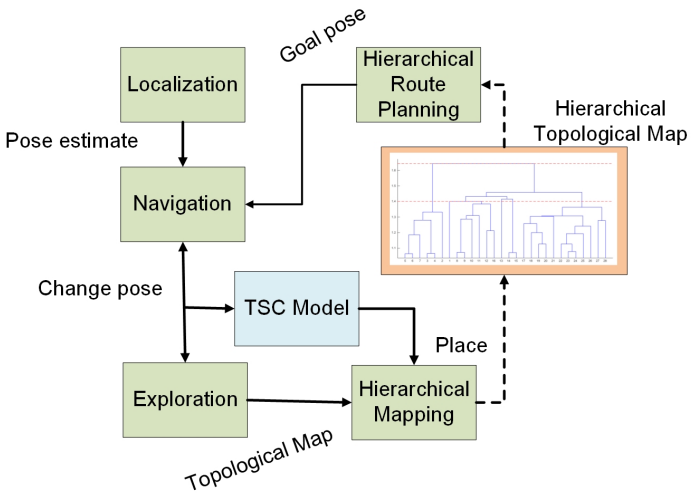


Figure 1.1. General approach.

spatial memory simultaneously. Topological map building and navigation problem is addressed in five stages as shown in Figure 1.1:

- First, navigating to a point in unknown environments is considered. The goal is to reach the given goal location with the desired heading. A reactive navigation method is proposed.
- Next, exploration of unknown environments is addressed. The goal is to have the robot build the topological map of its surroundings. A novel approach in which the robot determines the nodes of the map through discovering canonical appearances is proposed.
- Third, navigating along topological map is considered. This requires localization with respect to the nodes of the map. An approach in which provides both position and heading estimations is developed.
- Following, the robot needs to have an accumulated knowledge base of topological maps that it can refer to and update throughout its operation. One major issue is to handle the scalability problem that the robot would face when operating in real world. A hierarchical mapping strategy is proposed in order to address this issue.
- Finally, route planning using the knowledge in the hierarchical maps is studied. A hierarchical route planning method is proposed.

1.1.1. Navigation

The necessary waypoints are determined by continually *imagining* the furthest, yet visible location towards its goal node from its current location through simulating its motion in the background. These controllers are parametrized with not only the waypoint location, but also consider its orientation as well. The controllers track both the current obstacles and the ones which the robot lost visibility recently. The advantage of this approach is that it enables the robot to navigate with sparse topological maps and move through its nodes with headings as necessary.

1.1.2. Exploration

In the proposed approach, the exploration directions are chosen based on the openness and discontinuities on the data from a laser range-finder sensor. There are various criteria such as distance to previously visited locations, visibility, visual similarity in order to determine the most representative appearances in a place. As such, the collected canonical views represent the *place* in a compact and comprehensive manner.

1.1.3. Localization

The position estimation part is based on a previous work given in [5] with a critical improvement namely, local interpolation in the neighborhood of closest node. It is especially important in the case of visual similarity between a number of physically distant nodes in the map. The heading estimation is another contribution to [5]. It uses the closest node in the map and utilizes the phase difference between the DFC of the sensory information obtained at closest node and current location. As a second source of localization the laser sensor readings. Initialized by the camera based first-stage estimation, the reference and current readings are matched by using an ICP variant similar to [6]. Finally these two sensor based estimations together with odometry and IMU data are fused via an Extended Kalman Filter (EKF) [7] and a high-frequency drift-free location estimation is obtained.

1.1.4. Hierarchical Mapping

The proposed approach builds a hierarchical map in which i) the first level of hierarchy is constructed based on the TSC model; and ii) the ground level encodes the topological maps with the nodes of each map linked to the respective place node in the first level; and iii) the remaining levels of the hierarchy are determined using the SLINK algorithm [8] based on spatial and connectivity relations between nodes as well as a level criteria. The resulting algorithm is incremental and thus is able to achieve real-time performance in large scale environments.

1.1.5. Navigation Using Hierarchical Maps

The hierarchical maps are used to in route planning performance on graphs with extensive number of nodes and edges. The planning starts from the top level and refined recursively though the lowest level via some special map representation similar to the ones given in [9]. A significant performance improvement in processing time with respect to single plain graph search is shown by numerous experiments using a real-world dataset given in [10].

1.2. Contributions

The contributions of this thesis can be summarized as follows:

- A reactive navigation approach which considers not only the target position, but also target heading as well. The new method also utilize a short term memory in order to handle sensor limitation in complex environment, even for distant targets.
- Local interpolation based pose and heading location estimation on topological maps. In addition to the approach given in [11], here a method in order to estimate the robot heading is proposed. Moreover, the localization performance further improve by filtering misleading similarities stem from perceptual aliasing. Finally, the camera based estimated is fused with faster estimations coming from

different sensor modalities such as lidar, wheel odometry and IMU via an extended kalman filter (EKF) [7].

- A novel and efficient exploration approach based on the place awereness of the robot. It enables a robot be able to construct a more compact and comprehensive map of a specific place with respect to the work given in [12].
- A multi-level hierarchical mapping scheme with bases the place information at the first hierarchy level and some practical methods to find the shortest paths on this structure in an efficient manner.

1.3. Outline

The organization of the thesis is as the following: In Chapter 2 the navigation approach is detailed. The exploration approach is presented in Chapter 3. In the next chapter a method for estimation of both position and heading the robots is explained. The construction and use of multi-level hierarchical structures are presented in Chapter 5. The thesis concludes with a brief summary of results and possible future directions.

2. NAVIGATION

This chapter studies the problem of reliable navigation to a given goal location in an unknown environment. Consider a differential drive robot with radius ρ. Suppose that is located at $c \in \mathbb{R}^2$ with heading $\alpha \in S^1$. Define $x = \begin{bmatrix} c^T & \alpha \end{bmatrix}^T$ as being its state and note that $x \in \mathcal{X} \subseteq \mathbb{R}^2 \times S^1$. Its dynamics are defined by:

$$\dot{x} = D(x, \mu) \text{ with } c(0) = c_0,\ x(0) = \begin{Bmatrix} 0 & 0 & 0 \end{Bmatrix} \tag{2.1}$$

where $\mu = \begin{bmatrix} \upsilon & \omega \end{bmatrix}^T$ is the velocity control input consisting of linear velocity $\upsilon \in \mathbb{R}$ and angular velocity $\omega \in \mathbb{R}$. The robot is endowed with a laser sensor so that at each location and heading, it is able to sense the surrounding obstacles $q_x(f, t)$ where $q_x : \mathcal{F} \times \mathbb{R} \to \mathbb{R}^+$ and Here, $\mathcal{F} \subseteq S^2$ denotes all possible sensing pan and tilt directions and $f = \begin{bmatrix} f_1 & f_2 \end{bmatrix}^T \in \mathcal{F}$ denotes the pan and tilt directions. Note that if the laser is one-dimensional, then f_2 is constant. The navigation problem is formulated as follows: The robot needs to move to g without any collisions along the way and arrive at this location with the heading α_g.

The organization of the chapter is as follows. Related literature is discussed in Section 2.1. Reactive controllers are constructed as explained in Section 2.2. Extensive experimental results with a mobile robot demonstrate that the robot is able to navigate without any a priori generation of waypoints or in arbitrary cluttered settings without any assumptions regarding the geometry of obstacles as discussed in Section 2.5.

2.1. Related Literature

There has been extensive work done in navigation. The proposed methods can be grouped as search-based methods or reactive approaches. The distance of the goal location is an important consideration. As this distance increases, more advanced local navigation approaches need to be used as it requires the generation of extra waypoints

between the nodes and then following them.

Search-based methods formulate the problem as an optimization problem in which the environment is modeled as a grid. One of the most popular approaches is the dynamic window averaging method [13]. The DWA method has been extensively used after being available in ROS [14]. It is generally used together with a sampling based global planner such as A* or RRT* [15]. While these algorithms tend to have optimality guarantees, their performance is affected by grid resolution. Unfortunately, as grid resolution increases, the computational complexity increases as well. Reactive approaches are based on formulating closed-loop feedback policies [16–18]. They are affected by grid resolution. On the other hand, optimality may not be attained or simplistic assumptions are made regarding the obstacles in the environment.

Most methods focus only on reaching the target location without considering its heading. A simple solution is to rotate the robot around itself at the target location. However, this may not be possible due to several factors such as limited sensor field or motion limitations. The target orientation is also studied in [19], but it is not clear how it can be utilized in arbitrary environments since an obstacle-free environment is assumed.

2.2. Reactive Controllers

Navigation is done by using an approach that is based on the sequential composition of reactive controllers. The idea of sequential composition has been previously proposed. The proposed method differs in how the reactive controllers are constructed. In particular, there is no assumption about the geometry of the environment. The flow of processing is based on a loop of three stages:

- Determining the goal waypoint
- Tracking obstacles
- Computation of linear and angular speeds

Integral to all is the construction of a function that encodes the goal location as well as the obstacles. The robot uses the gradient of this function to determine how it moves.

The controllers are constructed considering the gradient vector fields of analytic functions $\varphi_{w,\theta,\mathcal{O}} : \mathcal{X} \to R^{\geq 0}$. They are defined as follows:

$$\varphi_{w,\theta,\mathcal{O}}(x) = \sigma_d \circ \sigma \circ \hat{\varphi}_{w,\theta,\mathcal{O}}(x) \tag{2.2}$$

The function $\sigma : [0, \infty) \to [0, 1)$ is defined as $\sigma(x) = \frac{x}{1+x}$ and $\sigma_d : [0, 1) \to [0, 1)$ is defined as $\sigma_d(x) = x^{1/k}$.

The functions $\hat{\varphi}_{w,\theta,\mathcal{O}}$ are parametrized by i) waypoints $w \in \mathbb{R}^2$ and associated headings θ_k that need to be reached; and ii) the so-far encountered obstacles $\mathcal{O}$. Their constructions are defined by the ratio of two terms similar to [20]:

$$\hat{\varphi}_{w,\theta,\mathcal{O}}(x) = \frac{\gamma_{w,\theta}(x)}{\beta_{\mathcal{O}}(x)} \tag{2.3}$$

Here, $k \in \mathbb{Z}^+$ is the relative weighting parameter of the numerator term with respect to the denominator term. The numerator term $\gamma_{w,\theta} : \mathcal{X} \to \mathbb{R}$ encodes the distance to the waypoint w as well as deviation from the goal heading θ:

$$\gamma_{w,\theta}(x) = (c - w)^T (c - w) \left(\eta_1 + \eta_2 \left(atan2 \left(e_2^T (w - c), e_1^T (w - c)\right) - \theta\right)\right) \tag{2.4}$$

The parameters $\eta_1, \eta_2 \in \mathbb{R}$ weigh the distances to goal position and heading. Here, both are taken to be equal to 1. As the waypoint w and associated heading θ change as the robot is navigating, so does $\gamma_{w,\theta}$.

The denominator term $\beta_{\mathcal{O}}(x)$ encodes the distance to all the so-far detected obstacles.

$$\beta_{\mathcal{O}}(x) = \prod_{o \in \mathcal{O}} \left(|c - o|^2 - (\rho + \rho_o)^2\right) \tag{2.5}$$

where ρ_o is the radius of obstacle o. Both the waypoints and obstacles are updated as the robot navigates.

The resulting controllers μ corresponding to the the linear and angular velocity inputs are as follows:

$$\mu = \begin{bmatrix} v_{max} \min(1, |c-g|) \cos(\alpha_\nu - \alpha) \\ \omega_{max} \sin(\alpha_\nu - \alpha) \end{bmatrix} \tag{2.6}$$

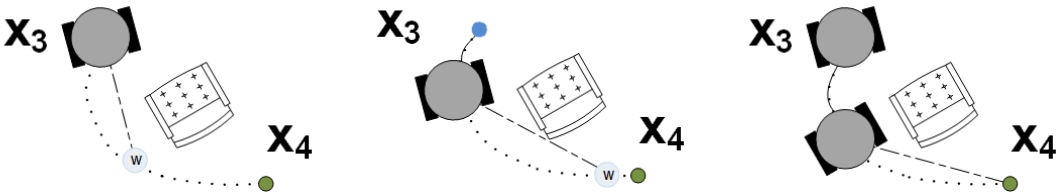
Here, v_{max} ve ω_{max} are linear and angular speed limits. The linear speed of the robot is also scaled according to the distance to the target location. In the second term, the variable α_ν denotes the bearing of gradient $-\nabla_c \varphi_{w,\theta,\mathcal{O}}(x)$. As the magnitude of difference $\alpha_\nu - \alpha$ becomes smaller, the robot will have only linear velocity. Alternatively, as it reaches 90°, there will be only rotational motion. As the the waypoints and the obstacles are updated, the resulting controllers are changed accordingly.

Navigation continues until goal location g is reached - namely:

- $|c-g| < \tau_c$ and
- $|\alpha - \alpha_g| < \tau_\alpha$

The parameters τ_p and τ_α denote proximity thresholds for position and heading respectively.

We do not have at present the desired analytical proof for the correctness of this hybrid approach. However, the prior examples of correct algorithms along with the success of our experimental evaluation provide a very strong motivation for generalizing the previous theoretical results to the present setting.



(a) First waypoint (b) Second waypoint (c) Last waypoint is the goal node.

Figure 2.1. Determining waypoints. The robot continually imagines its path and updates its waypoints as it is navigating from node (blue) to next (green).

2.3. Updating Waypoints

The robot continually checks the waypoint w and associated heading as it is navigating and updates it as necessary until it reaches its goal. The update of the waypoint is done as follows: First, it checks for visibility and accessibility of the goal g. If it is found to be both visible and accessible, then the waypoint is set as $w = g$. Otherwise, it *imagines* where it would move given its knowledge of the current goal g and obstacles $\mathcal{O}$. This is achieved in simulation with the reactive controller, but this time the gradient $-\nabla_c \varphi_{g,\alpha_g,\mathcal{O}}(x)$ is used in constructing the linear and angular speed inputs.

The way point location w is set to be the first location that is not visible or accessible in the simulated movement while the waypoint heading θ is defined by the heading that is obtained at this way point. The visibility $v(p; x)$ of a location $p \in \mathbb{R}^2$ with bearing f^p is determined by simply comparing the respective laser reading $q_x(f^p)$ to the distance $|c - p|$:

$$v(p; x) = \begin{cases} 1 & \text{if } q_x(f^p) > \|p - c\| \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

As an example, consider Figure 2.3. Here, the point p is not visible while the point p' is visible from robot's location c.

The accessibility of this point is determined through checking whether the separation between the vector $p - c$ and the obstacle would allow the robot to go through or not. This is checked by determining if there exists an obstacle $o \in \mathcal{O}$ with relative bearing f_o such that

$$a(p; x) = \begin{cases} 1 & \text{if } |q_x(f_o)\sin(\delta f_1')| \geq \rho + \delta_p \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

where $\delta f_1'$ is the relative bearing of $p - c$ with respect to the obstacle o and δ_ρ is the safety threshold. In Figure 2.3, point p' is visible, but not accessible while the point p'' is both visible and accessible.

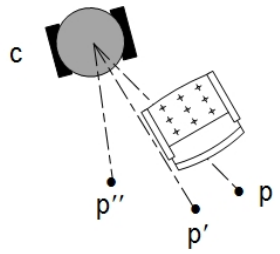


Figure 2.2. Visibility and accessibility with respect to location c: Here, location p is not visible, p' is visible, but not accessible and p'' is visible and accessible.

Thus, each waypoint maintains visibility with the previous way point. In this aspect, the definition of waypoints is conceptually similar to the *projected goal* [21]. As such, it continually switches its controller to the next depending on the change in the way point. This is repeated until the way point at which g_{i+1} becomes visible. A sample case is shown in Figure 2.1. The robot needs to navigate from to the goal g_4. Initially, it determines the waypoint as shown in Figure 2.1(a). As it is navigating towards to this waypoint, the waypoint changes as shown in Figure 2.1(b). It then starts navigating towards this waypoint. Finally, at some location, the waypoint is updated to be the goal location.

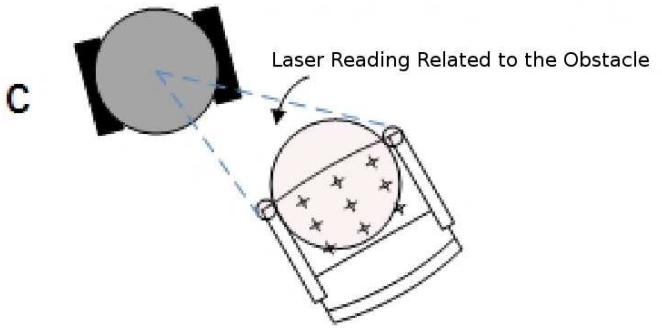


Figure 2.3. Representation of an obstacle as detected from location c by the covering 3 disks.

2.4. Obstacle Representation Using Covering Disks

The robot keeps track of obstacles $\mathcal{O}$ it has encountered as it is navigating. Obstacle knowledge is represented by a set of disks covering laser data. It includes the currently visible obstacles $\mathcal{O}_v$ as well as those recently encountered $\mathcal{O}_r$ even if they are no longer visible:

$$\mathcal{O} = \mathcal{O}_v \cup \mathcal{O}_r \tag{2.9}$$

The obstacles are represented by a set of disks that cover the depth data. Visible obstacles are determined after segmenting the depth data q_x using a connected components algorithm and obtaining a set of segments $\mathcal{A}(x,t) = \{A_m(x,t) \mid m = 1,\ldots,N_A\}$:

$$A_m(x,t) = \left\{ f_i \mid \left| \frac{q_x(f_i,t) - q_x(f_{i+1},t)}{q_x(f_i,t) + q_x(f_{i+1},t)} \right| \leq \tau_d, i = 1,\ldots,N_m \right\} \tag{2.10}$$

where N_m refers to pan extent. The parameter τ_a is determined empirically as it changes depending laser sensor and the environment. This is followed by the approximation of each segment $A_m(x,t)$ with a set of covering disks with maximal radius parameter ρ_{max} [22, 23]. Each disk is associated with a center o, a radius $\rho(o) \leq \rho_{max}$ and a timestamp $t(o)$. The number disks depends on the length $l(A_m(x,t))$ of each segment defined as:

$$l(A_m(x,t)) = \sum_{l=1}^{m-1} |q_x(f_{l+1},t)v(f_{l+1}) - q_x(f_l,t)v(f_l)| \tag{2.11}$$

where $v(f)$ is the unit vector in the direction f. If the length is less than $2\rho_{max}$, then the result is a single disk. Otherwise, there are $n_m(x,t)+2$ covering disks with radius $\rho_m = l(A_m(x,t))/\rho$ where

$$n_m(x,t) = \frac{l(A_m(x,t))}{\rho_{max}} - 1$$

The remaining two disks have smaller radius

$$\rho = l(A_m(x,t)) - \rho_{max}\left(\frac{l(A_m(x,t))}{\rho_{max}} - 1\right) \tag{2.12}$$

All the covering disks are then added to the set $\mathcal{O}_v$ of currently visible obstacles. An example of this process is as shown in Figure 2.3. The parameter $\rho_{max} = \rho$ is set so that narrow passages are naturally blocked.

In order to recall previously seen obstacles, the robot tracks recently observed obstacles $\mathcal{O}_r$. For this it considers all obstacles with timestamps $t' \in [t - K\delta t, t]$ and checks whether they continue to be both visible and accessible at time t. The parameter $M \in \mathbf{N}$ specifies the extent of obstacle recall. As obstacle representation is robocentric, obstacles are transformed to the robot's current frame based on relative odometry of its current state $x(t)$ with respect to $x(t - k\delta t)$. Consider such an obstacle o with timestamp $t - k\delta t$ with $k = 1, \ldots, M$. Let its transformed position be denoted by $\hat{o}$ at time t. Then, the set $\mathcal{O}_r$ is defined by

$$\mathcal{O}_v = \{o \in \mathcal{O} \mid t(o) < t, v_x(\hat{o}) = 1, a_x(\hat{o}) = 1\} \tag{2.13}$$

The set $\mathcal{O}_r$ is updated with the update period δt.

2.5. Experimental Results

We have conducted an extensive set of experiments with a TurtleBot Kobuki robot in order to evaluate the applicability of the proposed approach. The robot has

Table 2.1. Navigation in Place 1: Comparative results for 3 different scenarios using either a dense or sparse goal sequence.

		Time (s)		Normalized Path Length (%)		Arrival Heading Error (rad)	
		Proposed	A*+DWA	Proposed	A*+DWA	Proposed	A*+DWA
Dense	1	19.87	32.68	1.22	1.11	0.02	0.81
	2	29.80	35.27	1.10	0.95	0.38	1.06
	3	18.58	39.50	1.38	1.00	0.56	0.82
Average		22.75	35.82	1.23	1.02	0.32	0.90
Sparse	1	20.16	39.41	1.15	1.29	0.57	0.60
	2	30.30	47.26	1.07	0.96	0.33	0.99
	3	43.42	44.93	1.30	1.09	0.02	0.84
Average		31.29	43.87	1.17	1.11	0.31	0.81

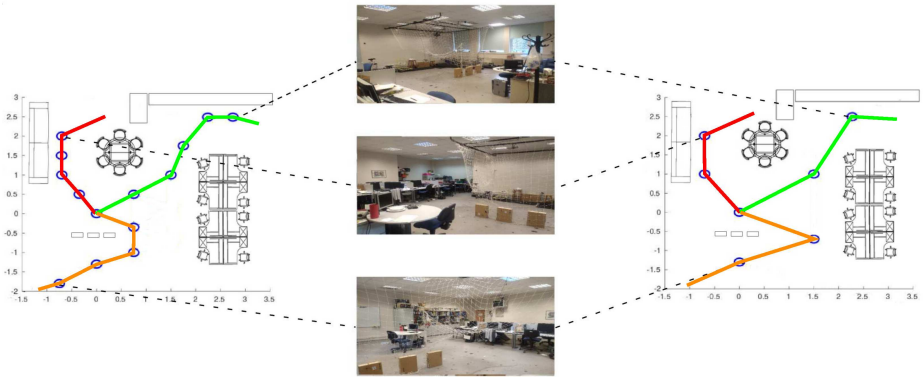


Figure 2.4. Navigation in Place 1: Left: Three scenarios (red, green, orange) with dense sequences. Right: Three scenarios (red, green, orange) with sparse sequences. Sample appearances from sample goal locations are also shown.

radius $\rho = 0.2$ m and has maximum linear and angular speeds of 0.2 m/s and $\frac{\pi}{3}$ rad/s respectively. It is endowed with two Hokuyo URG-04LX model laser scanners so that there is 360° depth sensing. The robot's processor is an Intel Core i5-4210Y model processor.

The experiments are repeated in two different places Place 1 (ISL) and Place 2 (EE Student Lounge). In each place, six different scenarios are considered. In each scenario, the robot is given a sequence of goal locations and headings which it needs to move to. The scenarios differ in the average distances between consecutive goals,

Table 2.2. Methods and used parameters for navigation.

	v_{max}	ω_{max}	path_bias	goal_bias	τ_c	τ_α	vx_samples	vth_samples	sim_time	k_{max}	δ_ρ	δt	K	τ_{wp}	η_1	η_2	τ_d
Proposed	0.2	1.0	-	-	0.2	0.2	-	-	3.0	50	0.2	0.03	300	0.5	1	1	0.5
A*+DWA	0.2	1.0	72.0	24.0	0.2	0.2	10	20	3.0	-	-	-	-	0.5	-	-	-

the number of goal locations in this sequence and the obstacles encountered along the way. Performance is evaluated considering three measures: elapsed time, normalized path length (the resulting path length divided by the shortest path to the set of goal locations) and goal heading error (absolute difference between the goal heading and the heading in which robot arrives the goal location).

The parameter values used in these experiments are given in Table 2.2. For comparison purposes, we consider DWA method combined with A* method. As ROS implementation of A* does not consider heading restrictions, a modified version of A* is used as explained in Appendix A. Furthermore, the linear velocity input in the DWA method is limited to positive values in order to avoid oscillations. The parameters values used in the experiments are given in Table 2.2.

The first set of experiments are performed in a $5 \times 5\,m^2$ sized place. In the first three scenarios, dense sequences are considered. Here, average distance between consecutive goals is around 0.5 m. The first scenario is shown by the red path in Figure 2.4. It consists of 6 goal locations with a corresponding heading. The robot encounters a simple obstacle (red circle along the way as seen in Figure 2.6(a). The resulting paths are also seen in these figures. The second scenario as shown by the green path in Figure 2.4 consists 7 goal locations. In this case, the robot encounters two simple obstacles as seen in Figure 2.6(b). The resulting navigation behaviors are also shown in these figures. The third scenario is shown by the orange path in Figure 2.4. The given sequence consists of 6 goal locations. In this case, the robot encouters a V-shaped obstacle. As this is not a convex shape, there is a possibility of robot getting trapped. In all, the robot is observed to reach the target location without colliding any obstacles. However, especially in the first scenario, the A*+DWA method is observed

Table 2.3. Navigation in Place 2: Comparative results for 3 different scenarios using either a dense or sparse goal sequence.

		Time (s)		Normalized Path Length (%)		Arrival Heading Error (rad)	
		Proposed	A*+DWA	Proposed	A*+DWA	Proposed	A*+DWA
Dense	1	41.43	44.93	1.42	1.00	0.08	0.51
	2	73.44	43.94	1.33	1.03	0.23	0.69
	3	53.02	45.68	1.68	1.18	0.88	0.91
Average		55.96	44.85	1.47	1.07	0.40	0.70
Sparse	1	44.52	55.61	1.43	1.00	0.24	0.62
	2	45.18	46.59	1.66	1.28	0.93	0.45
	3	50.30	72.65	1.67	1.16	0.16	0.71
Average		46.66	58.28	1.58	1.14	0.44	0.59

to oscillate frequently in regions close to the obstacles. The numerical results can be seen from Table 2.1. It is observed that with both of the methods, the robot aims for the target heading. With the proposed method, the goal headings are attained simultaneously with the goal location. With the A*+DWA method, the robot tends to reach the target location with a small goal heading error which it then corrects by rotating. It is observed that achieving the goal heading comes with an increased total path length.

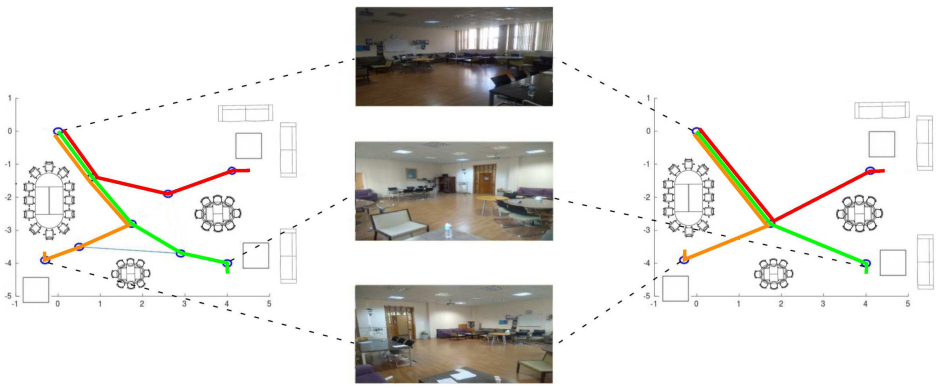
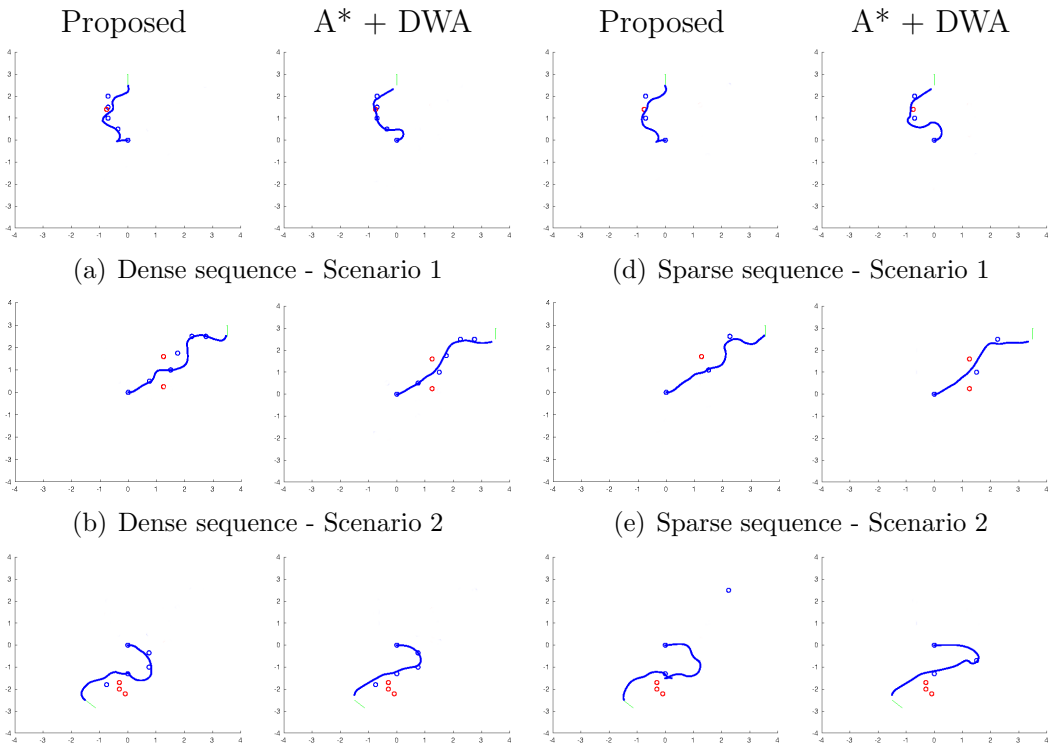


Figure 2.5. Navigation in Place 2: Left: Three scenarios (red, green, orange) with dense sequences. Right: Three scenarios (red, green, orange) with sparse sequences. Sample appearances from sample goal locations are also shown.

In Place 2, the experiments are repeated in a similar manner. The first scenario is

shown by the red path in Figure 2.5. It consists of 4 goal locations with a corresponding heading. The robot encounters a simple obstacle (red circle along the way as seen in Figure 2.7(a). The resulting paths are also seen in these figures. The second scenario as shown by the green path in Figure 2.5 consists 5 goal locations. In this case, the robot encounters two simple obstacles as seen in Figure 2.7(b). The resulting navigation behaviors are also shown in these figures. The third scenario is shown by the orange path in Figure 2.5. The given sequence consists of 5 goal locations. In this case, the robot encouters a V-shaped obstacle. As this is not a convex shape, there is a possibility of robot getting trapped. In all, the robot is observed to reach the target location without colliding any obstacles.

A close examination indicates that the robot navigates along the sequence of goal poses in a manner similar to the previous set of experiments. Again, when it uses a dense map, the robot reaches the target location without colliding any of obstacles in the environment. On the other hand, especially for the second and third scenarios, with the A*+DWA approach, the robot's trajectory tends to have oscillations in regions close to the obstacles. The proposed approach is also observed to have some difficulty in the second and third scenario. However, this behavior can be attributed to the partially observable nature of the environment so that whenever the robot obtains some new information from the environment, it can change its path if it finds a better route.

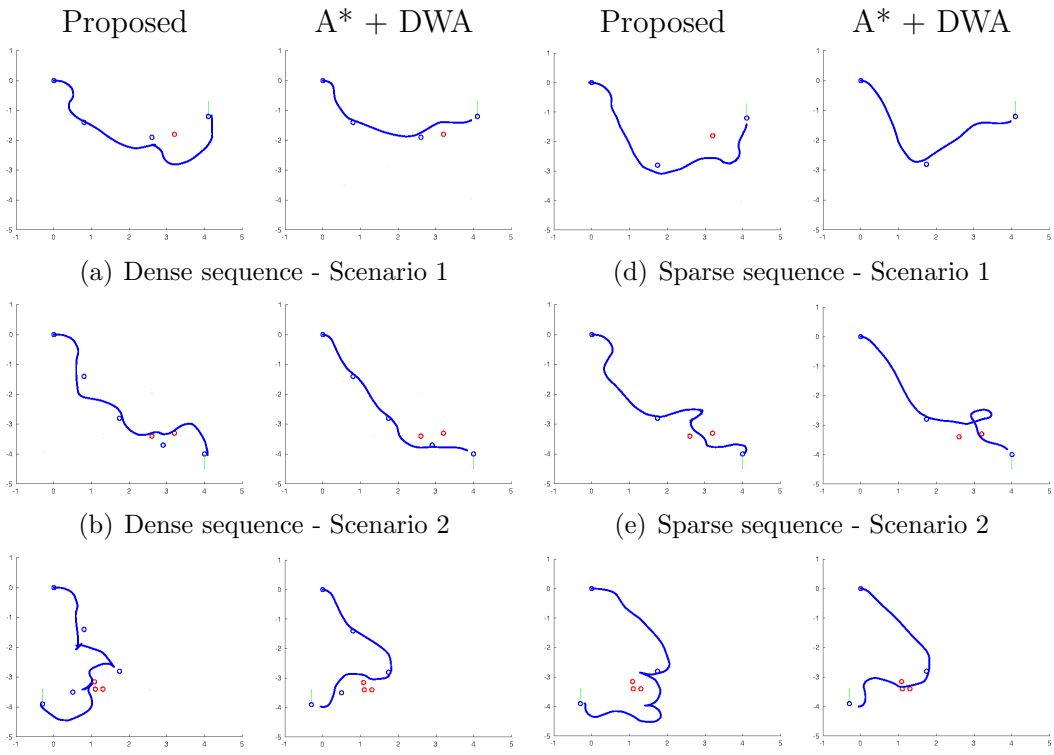


(a) Dense sequence - Scenario 1 (d) Sparse sequence - Scenario 1

(b) Dense sequence - Scenario 2 (e) Sparse sequence - Scenario 2

(c) Dense sequence - Scenario 3 (f) Sparse sequence - Scenario 3

Figure 2.6. Resulting paths in Place 1: Initially, the robot is at (0,0) with heading 0 radian. The resulting path (in blue), a set of the goal points (blue circles), new obstacles (red circles), goal location and heading (green arrow) are as seen.



(a) Dense sequence - Scenario 1 (d) Sparse sequence - Scenario 1

(b) Dense sequence - Scenario 2 (e) Sparse sequence - Scenario 2

(c) Dense sequence - Scenario 3 (f) Sparse sequence - Scenario 3

Figure 2.7. Resulting paths in Place 2: Initially, the robot is at (0,0) with heading 0 radian. The resulting path (in blue), a set of goal locations (blue circles), new obstacles (red circles), goal location and heading (green arrow) are as seen.

3. EXPLORATION

In this chapter, the exploration of an unknown place is explained. By a *place*, we refer to a specific spatial unit or area that encapsulates observations within one's sensory horizon - similar to human's concept [3] or a *region* [4].

The outline of the chapter is as follows: First, related literature is discussed. Next, an overview of the proposed approach is presented in Section 3.3. Tracking obstructed views is explained in Section 3.4. Local decision-making is presented in Section 3.5. This is followed by global decision-making in Section 3.6. Experimental results from outdoor settings demonstrate that the robot is able discover the canonical appearances efficiently - including a comparative study with a previous method in Section 3.7. The paper concludes with a brief summary and future directions.

3.1. Related Literature

Extensive work has been done in mapping. Firstly, the map building can be done with a metric approach [24]. Topological maps are preferred for their superior efficiency in the representation of the environment with respect to the metric ones. Topological maps consist of canonical appearances which aims at encoding as sufficient and necessary information about a specific place. Appearances are chosen as the main mapping element as they are free from some adverse effects such as drift as in the case of odometry based mapping.

There are two main issues about the collection of canonical appearances: the exploration direction and frequency of which the appearances are taken. In most applications, the exploration trajectory is chosen by the human operator [2, 25–27]. An alternative is autonomously selected directions such as random walk [28], greedy mapping [29–31], frontier-based approaches [32–37] or topological approaches [12].

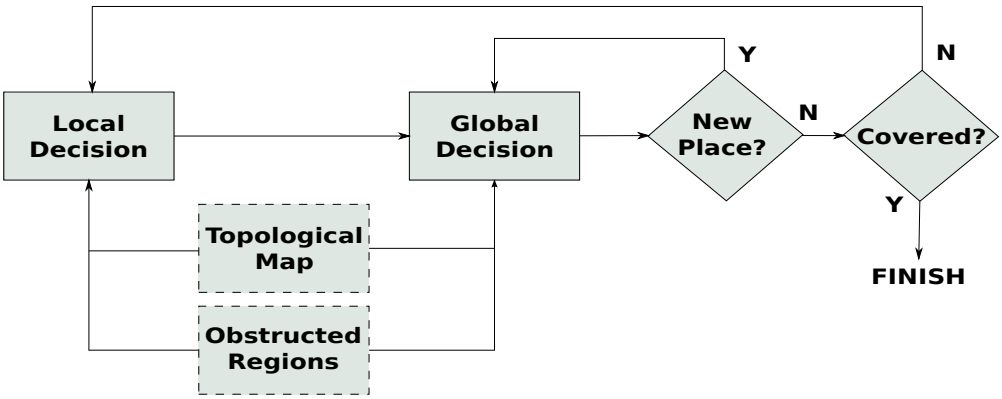


Figure 3.1. Exploration: General Approach.

In all these methods, the appearances are not collected efficiently such as predetermined constant step-size. This can be solved via post-processing as in [38, 39]. The node density obviously affects the knowledge available for localization. When the spatial density of the nodes is very dense, the quality of localization becomes higher [40–44], but it also increases required computational resources due to cost of matching processes. Therefore, it is more desirable to have a sufficiently sparse map representation from a computation oriented point of view.

3.2. Exploration: General Approach

In the proposed approach, the exploration is performed by autonomously choosing vantage points of the current place represented by a topological map. The strategy consists of concurrent local and global decision making as in work [12]. Differing from that the proposed approach also considers the compactness and coverage of the resulting representation. Rather than post-processing of densely sampled information, the appearances are chosen simultaneously using a number of criteria-namely: visibility, blockedness, exploredness and appearance similarity. The process continues until the current place is completely covered. In that way, the robot also avoid from unnecessarily traveling in a place which is already covered.

The possible exploration directions are determined at the *local decision-making*, stage. In the *global decision-making*, stage, the robot determines whether the movement should continue in this direction, the robot enters a new place and the current

appearance should be added to the topological map as a new node. This procedure is in way similar to [45], but uses only the so far constructed topological map G_k in its reasoning. This cycle is repeated until the current place is completely covered which is determined by existence of obstructed regions. The overall flow and reasoning is visualized in 3.1.

3.3. Vantage Points & Exploration

Canonical appearances are collected from vantage points which are connected to each other and have significant contribution to the coverage of the place. The vantage points and their relative odometry define a topometric map $G^0(k) = (\mathcal{X}^k(0), E^k(0), \iota, \varrho, \mathcal{C})$ of the place where $k \in \mathcal{K}$ indicates the index of the last node added to the graph. Each node $x \in \mathcal{X}^0(k) = \{x_0, \ldots, x_k\}$ corresponds to one different vantage point $x = \begin{bmatrix} c^T & \alpha \end{bmatrix}^T$ where $c \in R^2$ is the planar position and $\alpha \in S^1$ is the heading. The map ι^0 specifies the type (internal or transition) of each edge. The map ϱ assigns a cost to each edge as defined by the set $\mathcal{C}$ of intervals as explained in Chapter 5. We assume that while global odometric data is not available, the robot has access to local (relative) odometric data within each place. As such, the coordinates of the base point x_k in a place is known only relative to the first base point in the respective place. Thus, the cost between two nodes is computed based on the relative distance between them.

Vantage points are determined by autonomous exploration. In each newly detected place, the exploration strategy consists of a continuous cycle of local and global decision-making - similar to previous work [12]. Potential movement directions are determined during local decision-making while the robot actually moves along one of these directions during global decision-making. Differing from previous work, the robot aims for a compact, yet comprehensive set of canonical appearances. This requires the robot to be selective in choosing the respective vantage points while ensuring that the place is completely covered. Both are achieved through incorporating continuous checking for place coverage and place change into overall reasoning as seen in the Figure 3.1. As such, the mission is terminated if and only if there is no obstructed region left and the robot back-traces in case of it detects transition to a new place. Let us consider a

sample scenario as shown in Figure 5.2. The robot start exploring the unknown environment. It continues with the exploration considering visibility. If it detects transition to a new place, it stops and returns to the previous place as shown in Figure 3.2(b). The exploration ends when the place is fully covered as seen in Figure 3.2(c).

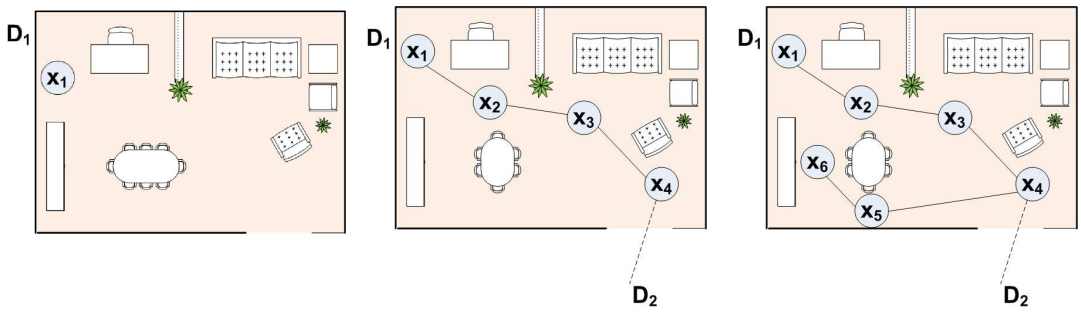
At each vantage point x_k, the robot looks around and collects appearance data including three-dimensional laser data. Each viewing direction $f \in \mathcal{F}$ is defined as $f = [f_1\, f_2]^T$ where f_1 is the pan angle, f_2 is the tilt angle and $\mathcal{F} \subset S^2$ is the space of viewing directions. The appearance data is then internally represented by a set of bubble surfaces. Bubble surfaces encode different visual features in bubble space in a manner that is implicitly dependent on robot pose while preserving the local S^2- geometry of the sensory data [1]. The bubble space $\mathcal{B} = R^2 \times S^1 \times \mathcal{F}$ is an abstract representation of the robot's base along with its viewing directions. Each point $b \in \mathcal{B}$ is defined as $b = [x, f]^T$. The bubble surface is a robot-centric hypothetical spherical surface $B_i(x_k, t))$ defined as:

$$B(x_k, t) = \left\{ \begin{bmatrix} f \\ \rho_i(b,t) \end{bmatrix} \mid \forall f \in \mathcal{F} \text{ and } b = \begin{bmatrix} x_k \\ f \end{bmatrix} \right\} \tag{3.1}$$

Each bubble surface is initialized to be a S^2 sphere with radius $\rho_0 \in \mathbb{R}^{\geq 0}$ – namely $\rho_i(b, 0) = \rho_0$. The map $\rho_i : \mathcal{B} \times \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ is a Riemannian metric encoding the response to the i^{th} sensory feature. As the robot looks around, for each viewing direction $f \in \mathcal{F}$, an observation $q_i(b, t)$ is made, each bubble surface is deformed at the corresponding bubble point b as $\rho_i(b, t^+) = \rho_i(b, t^-) + q_i(b, t)$ where the superscript t^+ denotes time just after t. For example, the bubble surface $B_l(x_k, t)$ encodes the depth readings as obtained from the two-dimensional laser sensor.

3.4. Unexplored Viewing Directions

In each place, the robot keeps track of the unexplored viewing directions $\mathcal{U}_k$ in its memory. Viewing behind them is integral to ensure place coverage. At each canonical base point x_k, it evolves the set $\mathcal{U}_k$ iteratively as follows: $\mathcal{U}_k = \left(\mathcal{U}_{k-1}^k \cup \mathbb{O}_k \cup \mathbb{O}_k^\infty\right) -$



(a) (b) (c)

Figure 3.2. The robot discovers the canonical appearances in a place. (a) Exploration starts from the current location; (b) Detection of a new place and turning back as the place is not fully covered; (c) The current place is now completely covered.

$\bigcup_{i=1}^{k} \mathcal{V}_i^k$. Upon entering a place, $\mathcal{U}_0 = \emptyset$. Unexplored viewing directions from the previous vantage point x_{k-1} are transformed to the current vantage point as $\mathcal{U}_{k-1}^k$ based on the relative odometric data.

The set $\mathbb{O}_k$ encodes the obstructed viewing directions as observed from the current base point x_k. These directions are computed from the ground slant rectified bubble surface $B_l(x_k)$ as explained in Appendix C. Ground slant rectification of the raw laser reading is required - since otherwise being on a slanted ground may deceive the robot in regards to obstacles and open passages [46]. Obstructed viewing directions are determined considering viewing directions with depth discontinuities on the bubble surface $B_l(x_k)$ along $f_2 = 0$. Thus, the set $\mathbb{O}_k$ is defined as:

$$\mathbb{O}_k = \left\{ b_{km} \mid \rho(b_{km}^-) \neq \rho(b_{km}^+),\ b_{km} = \begin{bmatrix} x_k \\ f_m \end{bmatrix} \right\} \tag{3.2}$$

Here, for a given bubble space point b_{km}, the variables b_{km}^- and b_{km}^+ are defined as bubble space points with slightly perturbed pan angles as:

$$b_{km}^- = \begin{bmatrix} x_k \\ f_{m1} - \delta f_1 \\ f_{m2} \end{bmatrix} \text{ and } b_{km}^+ = \begin{bmatrix} x_k \\ f_{m1} + \delta f_1 \\ f_{m2} \end{bmatrix} \tag{3.3}$$

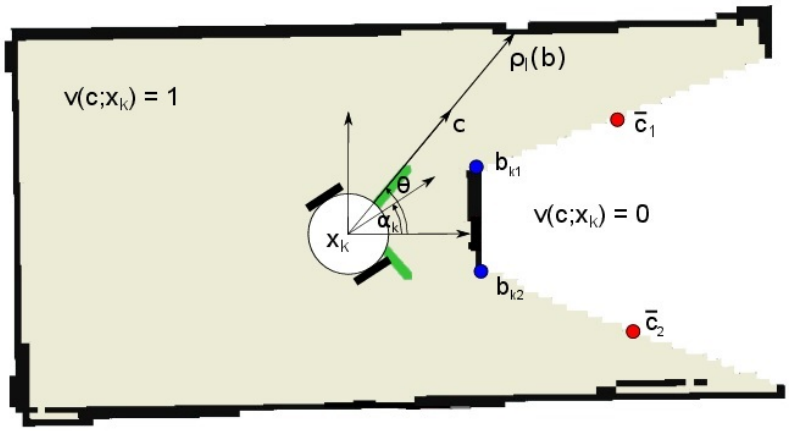


Figure 3.3. Obstructed viewing directions and visibility. Laser bubble surface (black), obstructed viewing directions (blue dots), obstructed centers (red dots), candidate movement directions (green lines) are shown.

The perturbation amount $\delta f_1 \in S^1$ is small and is defined to be positive in the counterclockwise direction. Note that adjacent bubble space points b_{km} and $b_{k(m+1)}$ will correspond to the two sides of an obstructed viewing direction depending on the relative proximity with respect to their neighborhood. Finally, this set is added only if the robot is in the place being explored.

The set $\mathbb{O}_k^\infty$ encodes the open viewing directions. They are determined by the bubble space points having with maximum depth readings - considering the sensor's technical specifications. Each connected set that is wider than τ_d is divided into smaller segments. Each segment is represented by the start and end bubble space point. Assuming that there are altogether N_o such segments, the set $\mathbb{O}_k^\infty$ is defined as:

$$\mathbb{O}_k^\infty = \left\{ b_{kl}, b_{k(l+1)} \mid \rho(b_{kl}) = \rho(b_{k(l+1)}) = \rho_0 + \tau_{max} \text{ and } \left| f_{l1} - f_{(l+1)1} \right| \leq \tau_d \right\} \qquad (3.4)$$

Here, the parameter τ_{max} is maximum depth range which can be observed by the robot.

The robot also keeps track of all obstructed views that it later views. The corresponding areas are considered to be explored. Let $\mathcal{V}_k \subset \mathbb{O}_k$ be the corresponding set.

The robot evolves it also iteratively:

$$\mathcal{V}_k = \mathcal{V}_{k-1}^k \cup S_k \text{ where } S_k = \{b_{km} \in \mathbb{O}_k \mid v(\bar{c}_{km}; x_k) = 1\} \tag{3.5}$$

with initialization $\mathcal{V}_0 = \emptyset$. The set $S_k \subset \mathbb{O}_k$ contains the obstructed views that can now be viewed from base point x_k. The robot constructs this set by considering the visibility of each obstructed viewing direction.

Visibility is determined by checking whether the center of each obstructed viewing direction is now or not from the current base point x_k. The center of each viewing direction is defined as:

$$\bar{c}_{km} = c_k + \begin{bmatrix} \frac{\rho(b_{km}^+)+\rho(b_{km}^-)}{2} \begin{bmatrix} \cos(\alpha_k + f_{m1}) \\ \sin(\alpha_k + f_{m1}) \end{bmatrix} \\ 0 \end{bmatrix} \tag{3.6}$$

The visibility function $v : R^2 \times \mathcal{X} \to \{0, 1\}$ is defined by comparing the proximity of a location c from a base point x_k with the respective laser reading:

$$v(c; x_k) = \begin{cases} 1 & \rho_l(b) > |c - c_k| \text{ s.t. } b = \begin{bmatrix} x_k \\ \theta - \alpha_k \\ 0 \end{bmatrix} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Here, θ is the direction from c_k to c. A location c is deemed to be visible from base point x_k if the depth reading along the pan viewing direction $\theta - \alpha_k$ is greater than the distance of location c. For example, in Figure 3.3, the robot would determine all locations in the shaded region to be visible.

3.5. Local Decisions

The goal of local decision-making is to determine the potential movement directions. This is achieved by finding viewing directions that enable the robot to see more as determined by their associated utility values. Such viewing directions are then taken as potential movement directions. As such, the robot adds them as edges to the current node $x_k \in \mathcal{X}_k$. Each added edge is initially connected to only one node. The robot finds the movement directions through two stages. First, candidate movement directions are determined. Next, the utility of these directions are computed.

3.5.1. Movement Directions

The robot determines the candidate movement directions by considering the unexplored viewing directions $\mathcal{U}_k$. Recall that these correspond to either obstructed or open viewing directions. In the former, the robot has to explore behind them while in the latter, the robot has to move towards them. Thus, for each $b_{km} \in \mathcal{U}_k$ where $\rho(b_{km}^-) \neq \rho(b_{km}^+)$, all viewing directions $f_1 \in [f_{m1} + \epsilon_-, f_{m1} + \epsilon_+]$ when $\rho(b_{km}^-) < \rho(b_{km}^+)$ and $f_1 \in [f_{m1} - \epsilon_+, f_{m1} - \epsilon_-]$ when $\rho(b_{km}^-) > \rho(b_{km}^+)$ are candidate movement directions. The boundary parameters $\epsilon_-, +\epsilon_+ \geq 0$ ensure that the robot will not collide with an obstruction. Note that $\epsilon_+ \leq \frac{\pi}{2}$, otherwise the robot tends to move away from the obstructed viewing direction.

All candidate directions are then checked whether they lead to previously explored areas. Exploredness is decided based on two considerations. First, all viewing directions with depth $\rho_l(b)$ closer than a threshold distance $\rho_0 + \tau_{max}$ are considered to be explored. Second, the robot calculates a series of locations that would be reached if it moved by multiples of a pre-determined step size $\tau_s > 0$ in the candidate direction and then computes the overlap $\Upsilon(b)$ of to-be-reached locations with the previously explored locations $\mathcal{X}_{k-1}$: $\Upsilon(b) = \frac{1}{n} \min_{x' \in \mathcal{X}_{k-1}} \left| c_k + n\tau_s \begin{bmatrix} \cos f_1 \\ \sin f_1 \end{bmatrix} - c' \right|$ where $x' = \begin{bmatrix} c' \\ \alpha' \end{bmatrix}$ and $n \in \{1, 2, \ldots, \lfloor \frac{\tau_{max}}{\tau_s} \rfloor\}$. The robot decides on this without actually

moving. The parameter τ_s is set a priori and depends on the sensor, robot speed and environment details. The scaling factor $1/n$ ensures that the check area gets larger as the to-be-reached location gets more distant. Thus, moving towards previously visited areas is avoided more strictly. The overlap is checked against a threshold τ_p as to determine any movement direction that leads the robot to the vicinity of a previously visited location.

The resulting knowledge is then represented by a binarized bubble surface $B'(x_k)$.

$$B'(x_k) = \left\{ \begin{bmatrix} f \\ \kappa(b) \end{bmatrix} \mid \forall f \in \mathcal{F} \text{ s.t. } b = \begin{bmatrix} x_k \\ f \end{bmatrix} \right\} \tag{3.8}$$

The function $\kappa : \mathcal{B} \rightarrow \{0, 1\}$ sets all viewing directions towards obstructed or open viewing directions to 1 - as long as they have not been so-far explored: It is required to $\tau_p < \tau_s$ since otherwise $\kappa(b) = 0$ which implies no feasible direction. Choosing smaller values for either results in increased computation and smaller $\tau_s - \tau_p$ difference results in a denser map.

Next, the number of potential movement directions is reduced to a much smaller set by performing connected component analysis on the binarized bubble surface $B'(x)$ - while checking whether the selected directions allow robot's passage. Let $\mathcal{A}_k = \{A_{k1}, \ldots, A_{kM}\}$ be the set of resulting connected components. Each connected component $A_{ki} \subset \mathcal{F}$ corresponds to one potential search direction $u(A_{ki})$ where the map $u : A_k \rightarrow S^1$ is defined as $u(A_{ki}) = \frac{1}{|A_{ki}|} \sum_{b \in A_{ki}} f_1$. When the horizontal extent of a component A_{ki} is too narrow, the search direction needs to be ignored as it won't allow passage. Otherwise, it is too wide so that the robot will not be able to cover the corresponding territory sufficiently. This is avoided by dividing the cluster into sub-clusters with narrower extents. Both are determined by computing the pan extension $\nu(A_{ki})$ of each component along the horizon as $\nu(A_{ki}) = |\{b \in A_{ki} \mid f_2 = 0\}|$ and comparing it with two thresholds τ_l and τ_u where τ_l is the allowable minimum extent and τ_u is the allowable maximum extent size.

3.5.2. Utility Computation

The second step is to compute the utility $\xi(A)$ of each potential direction $A \in \mathcal{A}_k$. The utility value determines the relative priority of each movement direction. The robot adds an unexplored edge e_{k_i} for each potential direction and labels it with the movement direction $u(A)$ and its utility value $\xi(A)$.

The function $\xi : \mathcal{A}_k \rightarrow \mathbb{R}$ encodes expected coverage. Expected coverage is based on robot estimating the accessibility of the unexplored regions and how much it expects to see of the unexplored regions $\mathcal{U}_k$ - if it moves by multiples of a minimum step size τ_s in the direction $u(A)$.

$$\xi(A) = \sum_{b_m \in S(A,x)} \psi_k(b_m) \tag{3.9}$$

where

$$S(A,x) = \left\{ b_m \mid v(\bar{c}, x') = 1 \text{ s.t. } x' = x + \begin{bmatrix} \tau_s \cos(\alpha_k + u(A)) \\ \tau_s \sin(\alpha_k + u(A)) \\ u(A) \end{bmatrix} \right\} \tag{3.10}$$

Here, we should note that x' is a hypothetical base point and the bubble surface $B(x', t)$ is constructed by geometrically transforming $B(x_k, t)$ by an amount of $\delta x = \begin{bmatrix} c\tau_s \cos(\alpha_k + u(A)) & \tau_s \sin(\alpha_k + u(A)) & u(A) \end{bmatrix}^T$. This transformation is not exact and done in an optimistic manner, such that, all depth readings of the points in the transformed surface $B(x', t)$ are initialized with τ_{max}. Then, the surface is deformed by utilizing the knowledge coming from each bubble point $b \in B(x_k, t)$. The final $\rho_l(b)$ value is the minimum one in case of multiple correspondences.

The accessibility of an unexplored viewing direction $b_m \in \mathcal{U}_k$ is computed based on its extent $\psi_k(b_m)$ - after ensuring that it allows the robot to pass through. Any discontinuity with an extent that is greater than robot's physical extent is a potential passage to possibly unexplored regions in the place. This is checked by comparing

the extent with the robot's width as approximated by $2\rho_r$. Thus the extent $\psi_k(b_m)$ is defined as:

$$\psi_k(b_m) = \begin{cases} |\rho_l(b_m^+) - \rho_l(b_m^-)| & \text{if } |\rho_l(b_m^+) - \rho_l(b_m^-)| > 2\rho_r \\ |\rho_l(b_m) - \rho_l(b_{(m+1)}| & \text{if } \rho(b_m) = \tau_{max} \\ 0 & \text{o.w.} \end{cases} \tag{3.11}$$

3.6. Global Decisions - Moving in a Place

In the global decision-making mode, the robot decides where to move from its current node and starts doing so. In any point if $\mathcal{U}_k = \emptyset$, which is called as full coverage, the exploration mission is terminated. If there are no unexplored movement directions, it goes back to to previous base point x_{k-1}. Otherwise, it chooses the direction $u(A^*)$ having maximal utility among the unexplored search directions that is associated with its current node: $A^* \in argmax_{A \in \mathcal{A}_k} \xi(A)$ It starts to move in the direction $u(A^*)$ and continues to do so until it reaches a new vantage point. It employs a variety of criteria for deciding whether its current base point $x(t)$ is a vantage point or not. The first is visual connectivity - namely if visual connectivity is lost. This occurs if $v(c(t); x_k) = 0$. The second is blockedness. As such, further movement along $u(A^*)$ is not possible. The third criteria is whether there is any expected coverage from $x(t)$ or not - namely if $\xi(u(A^*)) = 0$. In all the three cases, the next vantage point is set as $x_{k+1} = x(t)$. The addition of a new canonical appearance is finalized by two additional checks. The first involves checking whether the associated appearance is sufficiently distinct from any of the previously determined canonical appearances through comparing their pairwise distances against a minimum similarity threshold τ_I. The second checks whether $x(t)$ is indeed in the same place. This can be done automatically achieved through place detection [2]. If the robot determines the detection of another place, it simply returns back to x_k. Otherwise, if both checks are passed, a new canonical appearance is added. As such, vantage points failing to meet the two checks may not be associated with a canonical appearance. The robot then cycles back to the local decision-making mode.

3.7. Experimental Results

In this section, we report results using the outdoors Canadian Planetary data set [47]. A robot with $\rho_r = 0.5m$ is to discover the canonical views in an unknown 7200 m^2 place. Its laser has maximum range $\tau_{max} = 30$ meters. As spatial sampling is not uniform or dense, spatial interpolation is applied in order to increase spatial density of the data. During the ground slant rectification, only regions higher than 0.3 m in both lateral directions and with slopes greater than 45° are considered as obstructions. As such, the robot is assumed to climb over obstructions lower than 0.3 m or having a slope less then 45°. The parameters are set as: $\tau_d = 45°$ m, $\tau_s = 2$ m, $\tau_p = 1.9$ m and $\epsilon_- = 0$, $\epsilon_+ = \pi/2$. By having close τ_s and τ_p, a smaller number of vantage points is expected. Movement directions on the binarized bubble surface are determined considering $\tau_l = 5°$ and $\tau_u = 45°$. Appearances are represented in bubble space using $N_I = 100$-dimensional bubble descriptors with 10 harmonics [1]. The appearance similarity parameter is set as $\tau_I = 1e12$, but due to the discrete nature of the data, appearance similarity turns out to be low.

Table 3.1. Path statistics of previous approach.

Starting Point		# Canonical	Path	Covered area	Covered area
$c_1(0)$	$c_2(0)$	Appearances	Length (m)	with overlap (%)	without overlap (%)
-17.4	10.5	19	720	313.21	95.17
-19.2	-70.4	19	720	333.60	95.87
13.4	-93.4	19	720	310.27	94.86
-38.4	11.0	17	640	287.63	94.50
Average		18.5	700	311.18	95.10

Comparative performance metrics for varied starting points are given in Table 3.1 and Table 3.2. For comparison, we consider previous work [12] in which the same data set was used. Performance is evaluated based on the number of canonical appearances, path length, total area covered with and without overlap. It is observed that the path length is 624 meters on average which is 76 meters (12%) less than that of the reference

Table 3.2. Path statistics of proposed approach.

Starting Point		# Canonical	Path	Covered area	Covered area
$c_1(0)$	$c_2(0)$	Appearances	Length (m)	with overlap (%)	without overlap (%)
-17.4	10.5	17	656	274.64	96.92
-19.2	-70.4	15	536	261.69	93.19
13.4	-93.4	16	644	275.88	94.56
-38.4	11.0	18	660	274.26	93.57
Average		16.5	624	271.62	94.55

approach. This is partly due to the fact with the proposed approach, exploration is finished in case of sufficient coverage. The number of canonical appearances is reduced by 2 on average. Area coverage with overlap is calculated as the arithmetic sum of disks (each corresponding to the respective field of view) obtained from the laser bubble surface. Here, instead of using a single tilt angle, the slant rectified disks are used as described in Appendix. As such, inconsistencies due to readings in slanted regions do not affect the results. As a lower overlap indicates compactness, it is required to be as close to 100% as possible. In the area coverage without overlap, the intersecting regions are removed. It is observed that while in both cases, about 95% of the total area is covered, there is 18% improvement in the average overlap with the proposed approach.

Sample results for one starting point are given in Figure 3.4. It is observed that the robot is able to determine exploration directions around the obstructed regions for scenarios similar to ones shown in Figure 3.3. This is in contrast to previous approach that is observed to have difficulties in finding new exploration directions. The overlaps of the appearances are shown in Figure 3.4 where lighter color indicates higher overlap. The resulting topological maps are better in mapping the environment. This is attributed to the visibility criteria as it enables the robot to adapt the node (appearance) density as needed. For example, nodes 2 and 3 in the topological map of the proposed approach are relatively closer to each other - possibly due to the small

hill between them. The higher density of vantage points and thus appearances in that region will provide more detailed information to the robot as it tries to navigate or localize itself around that region. In parallel, with open environments, the number of canonical appearances is expected to decrease accordingly.

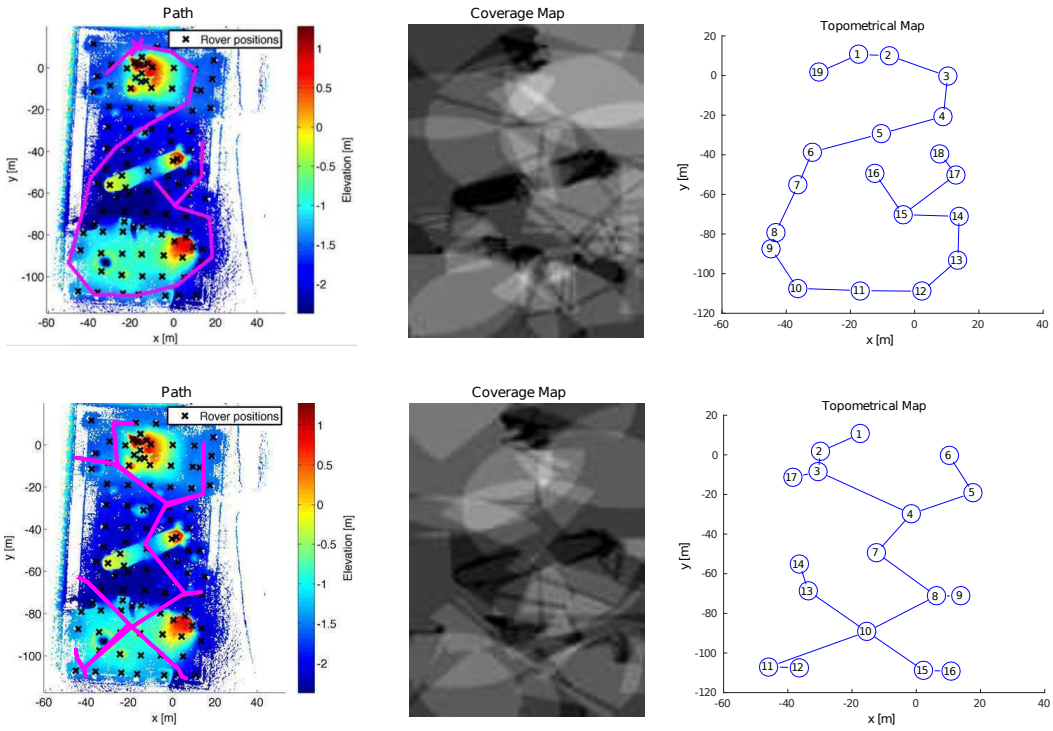


Figure 3.4. Total path lengths for previous (Top) and proposed approach (bottom) are 720 and 656 meters respectively. The covered regions where lighter color indicates increased overlap (middle) the resulting topological maps (right) are shown.

4. LOCALIZATION

The constructed maps can be used as a guide for a robot to localize itself. In this chapter, localization using topological maps is considered - since The organization of this chapter is as follows: Related literature is summarized in Section 4.1. Appearance-based localization is explained in Section [5]. Next, laser based pose refinement method is formulated in Section 4.3. In Section 4.4, the fusion of all estimations with odometry and IMU sensors is explained. The chapter concludes with experimental results.

4.1. Related Literature

In case of metric maps, where very dense information about the environment is available, probabilistic [24] or Iterative Closest Point (ICP) based [6] approaches can be used in order to estimate the position and heading of the robot. In topological mapping feature matching between the current and learned information can be used for localization [5, 48].

Hierarchical structures may further improve localization process via a kind of place recognition stage which starts from the top level of hierarchy and goes to the bottom until reaches a sufficiently detailed estimation [49] [50] [2] [51]. Hierarchical structures can be obtained via various graph clustering approaches [52]. When the cluster size is not explicitly given, fast agglomerative methods such as [53] can be used. This approach also allows the map structure evolves iteratively and thus it is very convenient for dynamically changing large-scale environments. On the other hand, using this type of clustering in real environment is not straight-forward and requires some additional processing to apply realistic graph structures.

After localizing itself on a conveniently constructed structure, the robot can use its pose information to navigate to a target location given in the map. Hierarchical structures are also useful for the path planning process [9] [54]. Instead of working with very high number of nodes on topological maps, hierarchical approaches systematically

reduces the search space and improves both time and space complexity of shortest-path calculation [55]. In this study we propose an iterative approach for hierarchical map building and an efficient recursive path planning method which enables a mobile robot navigates through resulting large-scale maps in real time.

4.2. Camera Sensor Based Estimation

4.2.1. Position Estimation

This part is mostly based on the method given in [5]. There are two major differences. The first is when there is some perceptual similarity between two distant nodes in the same place, the interpolation might lead erroneous results due to averaging effect. To overcome this issue a local interpolation approach is proposed. The second one is heading estimation which is based on Discrete Fourier Coefficients (DFC) of the image data taken at this specific point.

The location estimation is done by interpolating around the node with one of closest invariant distance to the current location. The candidate reference nodes are determined by normalized distance to the closest one. Let $x_{pk} = \begin{bmatrix} c_{pk}^T & \alpha_{pk} \end{bmatrix}^T, k = 1, \ldots, N$ be a base point in the place p having N nodes, given that $c_{pk} \in R^2$ is the location and $\alpha_{pk} \in S^1$ is the heading of the robot at base point x_{pk}. Also consider each base point is encoded with an descriptor Ix_{pk}. Finally, if $I(x(t))$ is the descriptor constructed at the time instant t, we first find the most similar node $k^* = min_k |I(x_{pk}) - I(x(t))|$. Then we obtain the set of candidates as the following:

$$\Omega_1 = \left\{ \forall i \text{ s.t. } \frac{|I(x_{pk^*}) - I(x(t))|}{|I(x_{pk^*}) + I(x(t))|} < \tau_I \right\} \tag{4.1}$$

Then the most consistent interpolation will be used in order to estimate the current location. First we determine interpolation radius:

$$\tau_d = \max_i \left(\min_{j \neq i} |c_i - c_j| \right) \tag{4.2}$$

For each candidate, nodes in radius τ_d are selected:

$$\Omega_2^{(i\in\Omega_1)} = \left\{ \forall j \text{ s.t. } |c_i - c_j| < \tau_d \right\} \tag{4.3}$$

An estimation is performed for each local interpolation:

$$\hat{c}^{(i)} = \frac{\sum_{j\in\Omega_2^{(i)}} \omega_j c_j}{\sum_{j\in\Omega_2^{(i)}} \omega_j} \tag{4.4}$$

The estimation with minimum cost is selected:

$$i^* = \min_{i\in\Omega_1} \sum_{j\in\Omega_2^{(i)}} \omega_j (c_j - \hat{c}^{(i)})^T (c_j - \hat{c}^{(i)}) \tag{4.5}$$

$$\hat{c}^* = \frac{\sum_{j\in\Omega_2^{(i^*)}} \omega_j c_j}{\sum_{j\in\Omega_2^{(i^*)}} \omega_j} \tag{4.6}$$

The pseudo-code of camera based pose estimation is given in three parts as Figures F.1-F.3.

4.2.2. Heading Estimation

In the heading estimation part the Discrete Fourier Coefficients (DFC) of the current image and closest node are calculated as the following [1]:

$$\begin{bmatrix} a(m,n) \\ b(m,n) \\ c(m,n) \\ d(m,n) \end{bmatrix} = \sum_{i=0}^{360} \sum_{j=-90}^{90} \begin{bmatrix} \rho(i,j)\cos(\frac{mi\pi}{180})\cos(\frac{2nj\pi}{180}) \\ \rho(i,j)\sin(\frac{mi\pi}{180})\cos(\frac{2nj\pi}{180}) \\ \rho(i,j)\cos(\frac{mi\pi}{180})\sin(\frac{2nj\pi}{180}) \\ \rho(i,j)\sin(\frac{mi\pi}{180})\sin(\frac{2nj\pi}{180}) \end{bmatrix} \tag{4.7}$$

Assume there is a $\delta\alpha$ heading difference from the reference point. Then the new coefficients will become:

$$\begin{bmatrix} a'(m,n) \\ b'(m,n) \\ c'(m,n) \\ d'(m,n) \end{bmatrix} = \sum_{i=0}^{360} \sum_{j=-90}^{90} \begin{bmatrix} \rho(i,j)\cos(m(\frac{i\pi}{180}+\delta\alpha))\cos(\frac{2nj\pi}{180}) \\ \rho(i,j)\sin(m(\frac{i\pi}{180}+\delta\alpha))\cos(\frac{2nj\pi}{180}) \\ \rho(i,j)\cos(m(\frac{i\pi}{180}+\delta\alpha))\sin(\frac{2nj\pi}{180}) \\ \rho(i,j)\sin(m(\frac{i\pi}{180}+\delta\alpha))\sin(\frac{2nj\pi}{180}) \end{bmatrix} \tag{4.8}$$

using trigonometric expansions we have the following:

$$\begin{bmatrix} a'(m,n) \\ b'(m,n) \\ c'(m,n) \\ d'(m,n) \end{bmatrix} = \begin{bmatrix} \cos(m\delta\alpha)a(m,n) - \sin(m\delta\alpha)b(m,n) \\ \sin(m\delta\alpha)a(m,n) + \cos(m\delta\alpha)b(m,n) \\ \cos(m\delta\alpha)c(m,n) - \sin(m\delta\alpha)d(m,n) \\ \sin(m\delta\alpha)c(m,n) + \cos(m\delta\alpha)d(m,n) \end{bmatrix} \tag{4.9}$$

We can solve for $\delta\alpha$ from this equation:

$$\sin(m\delta\alpha) = \frac{a(m,n)b'(m,n) - a'(m,n)b(m,n)}{a(m,n)^2 + b(m,n)^2} \tag{4.10}$$

$$\cos(m\delta\alpha) = \frac{a(m,n)a'(m,n) + b(m,n)b'(m,n)}{a(m,n)^2 + b(m,n)^2} \tag{4.11}$$

$$\delta\alpha = \frac{atan2(\sin(m\delta\alpha), \cos(m\delta\alpha))}{m} \tag{4.12}$$

Averaging for multiple harmonics (m,n) one can obtain a more precise heading estimation. The pseudo-code for camera based heading estimation is given in Figure F.4.

4.3. Laser Sensor Based Estimation

Another sensory source for localization is laser readings. Let $\hat{x} = \begin{bmatrix} \hat{c}^T & \hat{\alpha} \end{bmatrix}^T$ denotes the corrected base point, $o_j(x(t)) \in R^2$ is the j^{th} obstacle constructed as in

Section 2.4 with $\mathcal{O}(x(t))$ is the set of all $o_j(x(t))$. Here, we propose a method in which the robot uses the obstacles as approximated by the disks and computes $\hat{x}$ via a pair of coupled optimization problems. The problems are solved in an iterative manner. The iterations stop when a certain convergence criteria is satisfied.

The heading correction is based on comparing the current obstacles $\mathcal{O}(x(t))$ with those associated with a reference $\mathcal{O}(x_{pk^*})$. The estimated position $\hat{o}_j(x_{pk^*})$ of $o_j(x(t))$ with respect to the base point x_{pk^*} is defined by

$$\hat{o}_j(x_{pk^*}) = R_{\hat{\alpha(t)}-\alpha_{pk^*}} o_j(x(t)) + c(t) - c_{pk^*} \tag{4.13}$$

The corrected heading $\hat{\alpha}$ is determined via maximizing the number of matching disks. The associated optimization problem is defined as:

$$\hat{\alpha} \in \underset{\theta\in(-\pi,\pi]}{\arg\max} |\{o_j \in \mathcal{O}(x(t)) \mid \exists o_i \in \mathcal{O}(x_{pk^*}) \text{ s.t. } \|R_\theta \hat{o}_j(x_{pk^*}) - o_i(x_{pk^*})\| \leq \tau_e \varepsilon\}| \tag{4.14}$$

Here, $\tau_e \in \mathbb{R}^+$ is a matching threshold. Its value is set depending on the uncertainly in the sensory input. In case of multiple maxima the mean value of the widest plateau or the value closest to current heading is chosen with a priority in the given order.

The search can be done in various more efficient ways, but, not to loose generality of the approach, here we follow a brute force search in the whole angle interval using a step size $\Delta\alpha = 0.03$ rad which is a sufficiently feasible value both in terms of precision and computational time for our experimental setup. The matching part is given in Figure F.6 and the pseudo-code related to heading estimation is detailed in Figure F.7.

The location correction is done considering the disks that prevail from pk^* to t.

Let $\mathcal{O}^s(x(t)$ denote this set as:

$$\mathcal{O}^s(x(t)) = \{o_j \in \mathcal{O}(x(t)) \mid \exists o_i \in \mathcal{O}(x_{pk^*}) \text{ s.t. } \|\hat{o}_j(x_{pk^*}) - o_i\| \leq \tau_e\} \tag{4.15}$$

and we can also define the transformed form of this set as:

$$\hat{\mathcal{O}}^s(x_{pk^*}) = \{\hat{o}_j(x_{pk^*}) \mid o_j(x(t)) \in \mathcal{O}^s(x(t))\} \tag{4.16}$$

and $\pi : \hat{\mathcal{O}}^s(x_{pk^*}) \rightarrow \mathcal{O}(x_{pk^*})$ denotes the map that designates the matching disks which can be formally define as:

$$\pi(o_j) = \underset{o_i(x_{pk^*})}{\arg\min} \|o_j - o_i(x_{pk^*})\| \tag{4.17}$$

Note that the respective obstacles can be viewed as being stationary obstacles. The pseudo-code related to determination of static obstacles is given in Figure F.5.

The optimization problem is defined as:

$$\delta c^* = \underset{\delta c}{\arg\min} \sum_{o_j \in \hat{\mathcal{O}}^s(x_{pk^*})} \|o_j + \delta c - \pi(o_j)\|^2 \tag{4.18}$$

where $\hat{c} = c + \delta c^*$. The solution is solved by:

$$D_{\delta c} \sum_{o_j \in \hat{\mathcal{O}}^s(x_{pk^*})} \|o_j + \delta c - \pi(o_j)\|^2 = 0 \tag{4.19}$$

where $D_{\delta c}$ is the gradient operator with respect to δc. Thus:

$$\delta c = \sum_{o_j \in \hat{\mathcal{O}}^s(x_{pk^*})} (\pi(o_j) - o_j) \tag{4.20}$$

The coupled optimization problems are solved iteratively until δc is less than a convergence threshold of τ_c or a maximum iteration of N_{max}. After the last iteration the

final estimation is given as:

$$\hat{x} = \begin{bmatrix} c_0 + \delta c^* \\ \hat{\alpha} \end{bmatrix} \tag{4.21}$$

The wrapper for laser based pose correction is given in Figure F.8. Here, some type of optimization such as ordered sets are used. A further improvement might be the usage of k-tree data structures in order to find closest obstacles in a much efficient way.

4.4. Sensor Fusion

Sensor fusion is simple combines the odometry for location and IMU information for heading with previously estimated $\hat{c}$ and $\hat{\alpha}$. The basic steps for discrete version of extended kalman filtering is as the following.First we update x_k^- using odometry and IMU reading:

$$x_k^- = x_{k-1} + \begin{bmatrix} \upsilon_{k-1}\cos(\alpha_{k-1})\Delta t \\ \upsilon_{k-1}\sin(\alpha_{k-1})\Delta t \\ \omega_{k-1}\Delta t \end{bmatrix} \tag{4.22}$$

Here, we need Q_{k-1}^o the covariance estimate of odometry and IMU sensors and its value determined experimentally. State transition matrix F_k is also needed it can be calculated by taken the partial derivatives of state transition function $D(x_k, u_k)$ with respect to the state variables. In our case the state variables are simply the components of the recent base point x_k. Our transition function can be approximated from Equation 4.22:

$$F_k = \frac{\partial D}{\partial x} = \begin{bmatrix} \frac{\partial D_1}{\partial c_1} & \frac{\partial D_1}{\partial c_2} & \frac{\partial D_1}{\partial \alpha} \\ \frac{\partial D_2}{\partial c_1} & \frac{\partial D_2}{\partial c_2} & \frac{\partial D_2}{\partial \alpha} \\ \frac{\partial D_3}{\partial c_1} & \frac{\partial D_3}{\partial c_2} & \frac{\partial D_3}{\partial \alpha} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\upsilon\sin(\alpha_{k-1})\Delta t & \upsilon\cos(\alpha_{k-1})\Delta t & 1 \end{bmatrix} \tag{4.23}$$

Then we update predicted covariance estimate and calculate the kalman gain by using covariance matrices:

$$\Sigma_k^- = F_k \Sigma_{k-1} F_k^T + Q_{k-1}^o \tag{4.24}$$

$$K_k = (\Sigma_k^- + Q_k^l)(\Sigma_k^- + Q_k^l)^{-1} \tag{4.25}$$

And finally we calculate the fused estimation by using Kalman gain:

$$\tilde{x_k} = x_k^- + K_k(\hat{x_k} - \tilde{x_k^-}) \tag{4.26}$$

The final step is estimation covariance for the next iteration:

$$\Sigma_k = (I - K)\Sigma_k^- \tag{4.27}$$

4.5. Experimental Results

The experiments are conducted by using Kobuki Turtlebot base, V360 omni directional camera and two URG-04LX laser scanners, in the following order: First, the performance of camera based localization is tested. Then, the laser matching approach is validated using real world data and compared with an open-source library given in [56]. Finally the sensor fusion approach is tested.

4.5.1. Camera Based Estimation

The dataset is constructed via images taken with 0.2 m spatial density. The tests are performed images taken close, but different locations. For the heading estimation, results for harmonics m=3 and n=2,..,10 are combined. The detailed results can be seen from Figure 4.1.

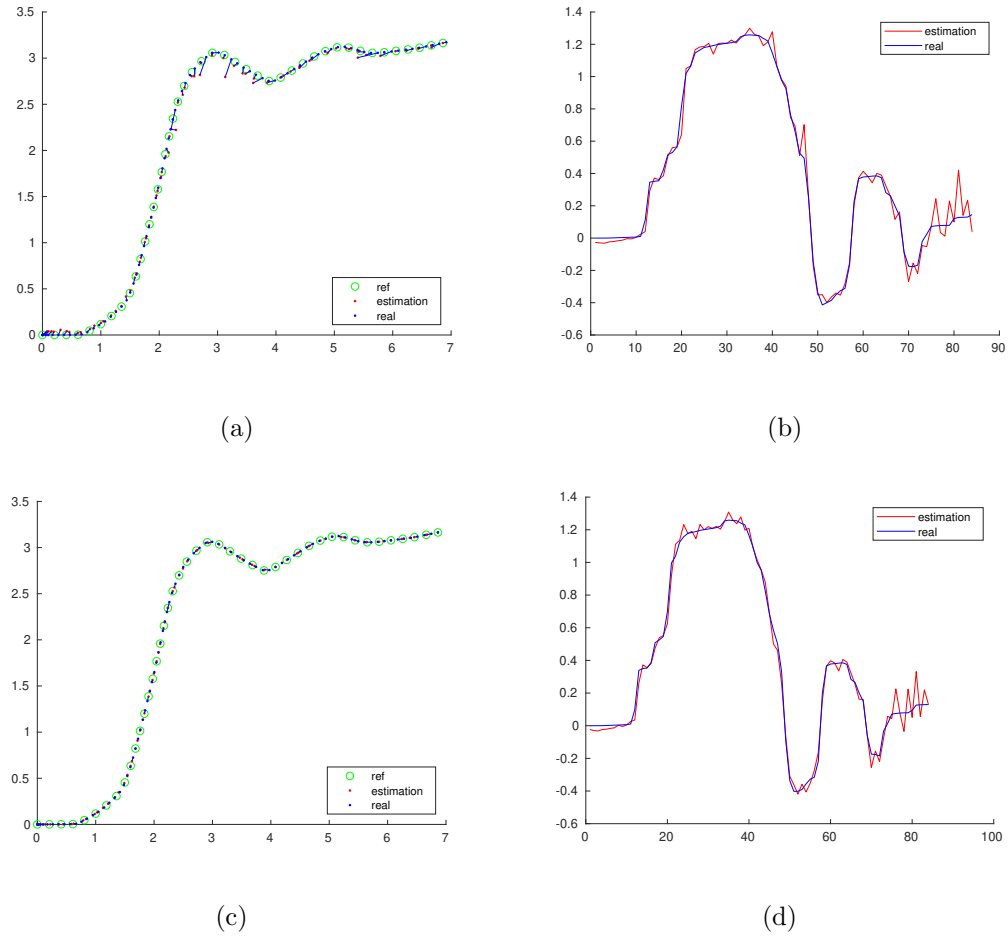
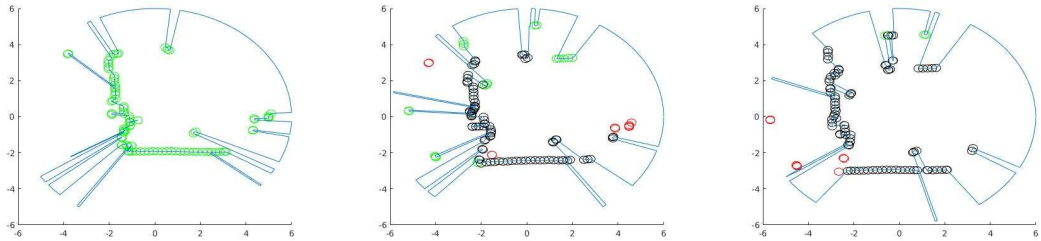
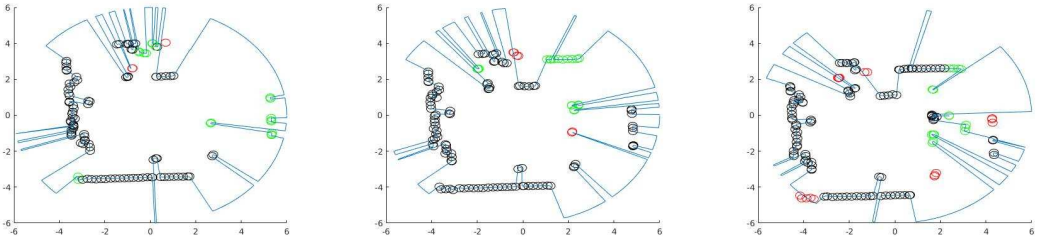


Figure 4.1. Visual data based estimation results: (a) location estimation for complete interpolation; (b) heading estimation for complete interpolation; (c) location estimation for local interpolation; (d) heading estimation for local interpolation.

The average position estimation error for complete interpolation is 0.0854 m, whereas its decreased to 0.0342 m for the local. This significant decrease is due to removal of averaging effect can be observed from second sharp turn of the robot. Heading error on the other hand is not affected in this extend. However, we should note that when the spatial density of sampling decreased, the heading estimation performance will also be lower as the mismatches with the closest reference will become farther away.



(a) $c(t_1) = [0.0, 0.0]^T$ (b) $c(t_2) = [0.5, 0.5]^T$ (c) $c(t_3) = [1.0, 1.0]^T$



(d) $c(t_4) = [1.5, 1.5]^T$ (e) $c(t_5) = [2.0, 2.0]^T$ (f) $c(t_6) = [2.5, 2.5]^T$

Figure 4.2. The evolution of obstacles as approximated by a set of covering disks from six consecutive locations. Stationary disks (black) prevail while the rest are either appearing newly (green) or disappearing (red).

4.5.2. Laser Based Estimation

First, we consider the approximation of obstacles based on the laser data from consecutive six base points along a path of 3.54m are shown in Figure 4.2. The related obstacle knowledge is given in Table 4.1. Here, at the first base point, the robot approximates the obstacles with 93 disks shown in green. In the next base point $[0.5, 0.5]^T$, most disks remain as observed. However, the robot also determines some new disks as seen in the top left corner of Figure 4.2. In the next two base points, 8 disks have disappeared while there are 4 new disks. This is because the visible part and thus the overall shape of the related obstacle is changed. On the other hand, along this path, more than 83% of the disks turn out to be stationary between consecutive frames.

Next, we test the base point correction performance - considering an unstructured dynamic environment. The parameters are set as $\tau_e = 0.8$, $\Delta\alpha = 0.03$, $N_{max} = 50$ and $\tau_c = 0.01$. The τ_e parameter can be increased if the uncertainty of the environment

Table 4.1. Obstacle knowledge from 6 consecutive locations.

Location i Number	Number of Obstacles	Stationary Disks	Appearing Disks	Disappearing Disks
1	28	-	93	-
2	29	75	20	10
3	24	81	4	8
4	29	88	20	4
5	33	98	19	6
6	33	94	17	16

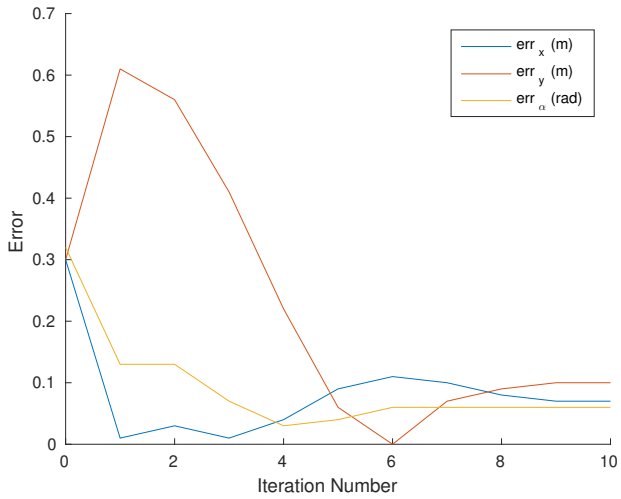
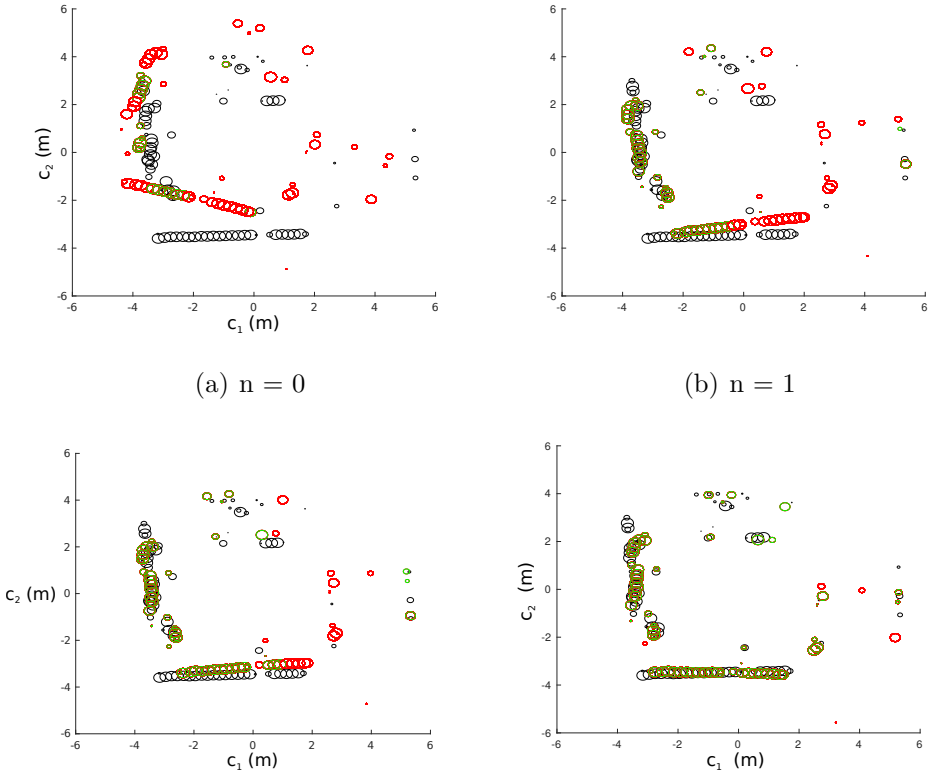


Figure 4.3. The convergence of base point correction algorithm. The actual location $c = [1, -1]$. The initial estimation is $\hat{c}(0) = [0.70, -0.70]$ and $\hat{\alpha}(0) = -0.3$ while the corrected estimation is $\hat{c} = [0.93, 0.93]$ and $\hat{\alpha} = -0.04$.

is harsher and a lower $\Delta\alpha$ value results in a more precise heading estimation with the cost of proportionally increasing computational time. Finally, decreasing N_{max} and/or increasing τ_c parameters will cause the iterations stop earlier. For example, with a 0.3 m error in the c_1 and c_2 coordinates and 0.3 rad error in the heading direction α, localization error is reduced around 19 cm down to 11 cm and heading error is reduced down to 0.05 rad in less than 120 ms of CPU time. The resulting correction is shown in Figure 4.4.



(a) n = 0 (b) n = 1

(c) n = 3 (d) n = 10

Figure 4.4. Comparison of obstacles before and after base point correction is applied at $c = [1, -1]$. Black denotes reference obstacles, green is for static and red is for missing obstacles in the current obstacle map.

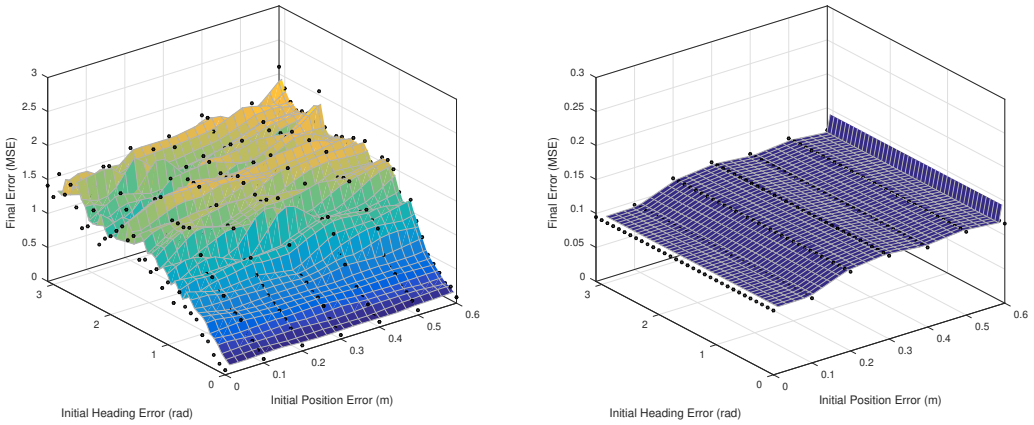
The average time for each iteration is around 15 ms and entire process typically terminates in less than 300 ms. The bottle-neck in the calculation process is the optimal heading search which consists of a simple brute force search. Using a better search policy and/or appropriate data structures for matching, these values might be improved a lot. However, considering 10 Hz scanning rate of laser range finder and need for multiple readings the unoptimized version is still suitable practical purposes.

Finally, we should also note that there are various much more efficient and precise Iterative Closest Point (ICP) based pose estimation algorithms two of which are compared in [56]. Here, we propose a method integrated with our obstacle approximation approach and it can deal with a significant amount of uncertainty in sensory input. A comparative study is also included as shown in Figure 4.5. For the comparison purposes, the open source point matcher library [56] is used with default parameters and

point to point error metric. The experiments are repeated in 4 different points with 1 meter seperation from the reference point in both coordinates and various initial pertubration rates in an unstructured environment. As the pmicp method frequently failed for initial heading error rates higher than 0.3 rad and our method has diffuculty to converge wor position errors near the static obstacle $\tau_e = 0.8$ m we calculated the average for 0 to 0.6 meter and 0 to 0.3 radians only. As a result our method outperformed the pmicp by 0.23 vs 0.12 in final mean square error (MSE) and 0.116 s vs 0.997 s in average CPU time. The latter improvement is relatively obviously as we decrease the number of matching items whereas the former is more subtle and possibly due to an implicit filtering operation during the obstacle estimation process. Also note that our method was not affected from the initial heading error as a result of global brute-force approach.

Table 4.2. Base point correction algorithm for $c = [1, -1]^T$.

Iteration Number	x	y	θ	Static Obstacles
0	0.70	-0.70	-0.30	19
1	1.01	-0.41	0.15	41
2	1.03	-0.46	0.15	46
3	0.99	-0.61	0.09	48
4	0.96	-0.80	0.05	67
5	0.91	-0.96	-0.02	70
6	0.89	-1.02	-0.04	71
7	0.90	-1.09	-0.04	72
8	0.92	-1.11	-0.04	73
9	0.93	-1.12	-0.04	71
10	0.93	0.93	-0.04	71



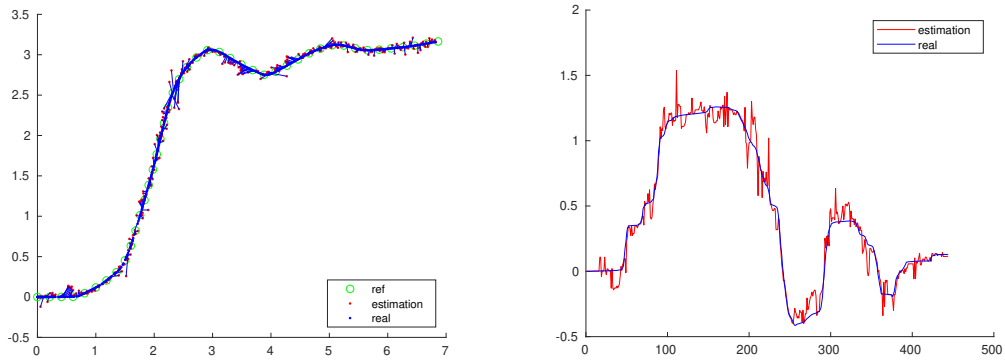
(a) pmicp (b) our method

Figure 4.5. Comparison of two pose correction methods. Average final error and CPU time for initial error rates of 0-0.6 m and 0-0.3 rad are (a) 0.23 and 0.997; (b) 0.11 and 0.116 respectively. Note also that the method in (b) converges with the same final error rate mostly irrespective of initial heading error.

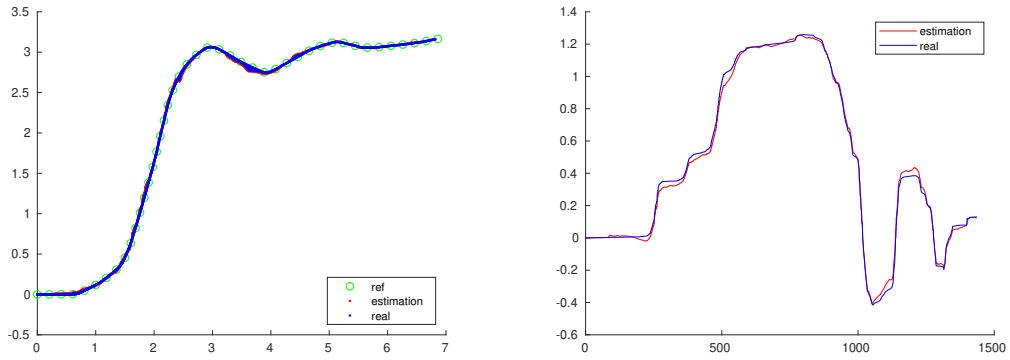
4.5.3. Fused Estimation

In this part, the same experimental setup with Section 4.5.1 is used. The covariance matrix is used as $Q^o = 0.1I_{3x3}$, which is the approximate standard deviation value for combined odometry and IMU sensors. The raw laser scan estimation values are compared with the fused one. The detailed results can be seen from Figure 4.6.

The average position estimation error for raw laser estimation is 0.0811 m, whereas its decreased to 0.0272 m for the local. A similar improvement is also observed in the heading estimation which decreased to 0.0184 from 0.0526 radians. More importantly, the output is smoothed significantly after the filtering process.



(a) (b)



(c) (d)

Figure 4.6. Sensor fusion results: (a) Location estimation for laser scan; (b) Heading estimation for laser scan; (c) Location estimation after sensor fusion; (d) Heading estimation after sensor fusion

5. HIERARCHICAL TOPOLOGICAL MAPS

This chapter considers hierarchical topological maps - namely their building, evolution and usage.

The organization of this chapter is as the following: Related literature is summarized in Section 5.1. The construction of hierarchical maps are explained in Section 5.2. This is based on the SLINK hierarchy as explained in Section 5.4. The evolution of the hierarchical map is formulated in Section 5.5. The proposed route search is presented in Section 5.6. Navigation using the resulting route is done as explained in Section 5.7. Experimental results obtained from a large scale environment is presented in Section 5.8.

5.1. Related Literature

In topological mapping, scalability is also a crucial issue. While it is much better compared to metric methods, nevertheless, as the number of known environments increase, so does the accumulated knowledge and the associated topological maps. This comes with a computational cost which limited processor units on the robots could not handle in real-time. Thus, a major consideration in building these maps is that they should enable efficient route planning.

Various approaches have been proposed in order to solve this scalability issue. For example [57, 58] proposes a hybrid method which generates compact SLAM based maps in the lowest level and a topological map which determines neighborhood relation between the maps below. Another study given in [59] introduces a multi-level hierarchical structure which is created in a supervised manner, but it restricted with four levels. For autonomous operation, the structure of the the hierarchical map H needs to be constructed automatically. This is a problem of graph partitioning. Thus, it is important to have a *good partion*. Typically, the goodness of a partition depends on:

- Similarity among the nodes in a cluster
- Number of nodes grouped in a cluster
- Cost balance between internal and external edges of clusters

The parameters of the partitioning affect:

- They should encode real or commonsense knowledge
- Range of possible partitions that can be obtained
- Redundancy of clusters

In order to obtain these kind of structures more efficiently the graph clustering methods given in [52] might be utilized. However, for the tasks such as exploration the cluster size is not given explicitly in the beginning. A generic multi-level hierarchy is given [9], but the reported complexity values for map construction are not feasible for real-time applicability.

The efficiency of using the resulting hierarchical map has also been considered. One approach is to use interval based edge costs instead of single value as in most of the graph [9]. In that way multiple edges can be merged at an upper level accordingly. Another point again in [9] is at an upper level the traversing through nodes might also have some interval based cost. Considering all these issues, some significant modifications to the standard path planning are needed in order to use hierarchical structures in path planning. Here, we propose methods for both map construction and path planning utilizing these maps.

5.2. Hierarchical Map

The hierarchical topological map H is defined by a set L of levels as follows:

- The set of levels $L = \{0, , \ldots, L^r\}$. The value $l = 0, \ldots, r$ is referred to as the depth of the map H. The level L^0 is referred as the ground level and the level L^r is the top level.

- Each level L is associated with a height $h(L) \geq 0$.

Each level L_i of the hierarchy H is denoted by the set $L^i = (G^i, s^i)$. Here, $G^i = (\mathcal{N}^i, E^i, \iota, \varrho_e^i, \varrho_n^i, \mathcal{C}^i)$ denotes the graph associated with level L_i as follows:

- $\mathcal{N}^i$ refers to the set of nodes at level L^i. Individual nodes are defined by $n_j \in \mathcal{N}^i$.
- E^i refers to the set of edges at level L^i. Individual edges are defined by $e(n_i, n_j)$ - namely an edge from node n_i to node n_j.
- The function $\iota : E^i \to \{0, 1\}$ maps the edge to their types.
- $\mathcal{C}^i$ - The set of interval costs. Here, each interval is a partially ordered set.
- $\varrho_e^i : E^i \to \mathcal{C}^i$ is the edge cost function - namely $\varrho_e^i(e(n_i, n_j)) \in \mathcal{C}^i$.
- $\varrho_n^i : \mathcal{N}^i \to \mathcal{C}^i$ is the node internal cost function - namely $\varrho_n^i(n_i) \in \mathcal{C}^i$. This is determined using shortest path algorithm applied to all pairs of transition nodes, determining the least cost paths and then setting up the interval based on the least and highest resulting costs.

The remaining elements of L^i are as follows:

- The function $s^i : \mathcal{N}^{i-1} \to \mathcal{N}^i$, $i < r$ is a node partition function that maps subsets of nodes of level L^{i-1} to nodes $\mathcal{N}^i$ of level L^i. Note that it is not necessarily one-to-one or onto. The node s_j^i is referred to as the supernode of node $n_j \in \mathcal{N}^{i-1}$. The refining functions $(s^i)^{-1} : \mathcal{N}^i \to 2^{\mathcal{N}^{i-1}}$ are inverse functions defined for $0 < i \leq r$:

$$(s^i)^{-1}(n_h) = \left\{ n_j \in \mathcal{N}^{i-1} \mid s^i(n_j) = n_h \right\} \tag{5.1}$$

Note that the set $S^i(n_h) = (s^i)^{-1}(n_h)$ corresponds to a cell on the partition of $\mathcal{N}^{i-1}$. All nodes $n_j \in S^i(n_h)$ are referred to as subnodes of node n_h. This set along with all the associated edges (namely $\forall e(n_j, n_{j'})$ where $n_j \in S^i(n_h)$ or $n_{j'} \in S^i(n_h)$) defines a subgraph of super node n_h. The nodes of this graph are grouped into two:

(i) Transition (Border) nodes: A node $n_j \in S^i(n_h)$ is a transition node iff $\exists \iota(e(n_j, n_d)) =$transition such that $n_d \notin S^i(n_h)$.

(ii) Internal nodes: A node $n_j \in S^i(n_h)$ iff $\nexists\, \iota(e(n_j, n_d)) =$ transition such that $n_d \notin S^i(n_h)$.

Note that the function s^i enables the abstraction of level L^i. The composition of h abstraction functions is defined as follows:

$$s^{i,h}(n_j) = s^{i+h} \circ s^{i+h-1} \circ \ldots \circ s^{i+1} \circ s^i(n_j) \tag{5.2}$$

It corresponds to a chain of abstraction to level L^{i+h} starting at level L^i.

A path $P^i(n_s, n_g)$ specified at level L_i is given by sequence of nodes

$$P^i(n_s, n_g) = \{e(n_0, n_1), \ldots, e(n_{l-2}, n_{l-1})\}$$

where $n_0 = n_s$, $n_{l-1} = n_g$ and $e(n_s, n_{s+1}) \in E^i$. The set of paths from n_s to n_g are denoted by $\mathbf{P}^i(n_s, n_g)$. Two paths $P(n_0, n_j)$ and $P(n'_o, n'_j)$ can be concatenated iff $n_j = n'_j$. The resulting path

$$P(n_0, n'_j) = P(n_0, n_j) + P(n'_o, n'_j) \tag{5.3}$$

5.3. Relating Topological Maps

The ground (terminal) level contains the topological maps associated with the places $\mathbb{P}_n$. Let $G^0(\mathcal{P}) = (\mathcal{N}^0(\mathcal{P}), E^0(\mathcal{P}))$ be the topological map associated with place $\mathcal{P} \in \mathbb{P}_n$. This topological map is obtained either through teleoperation as presented in [2] or through exploration as presented in Chapter 3. The nodes $\mathcal{N}^0(\mathcal{P})$ correspond to selected locations as defined by the respective appearances while the edges $E^0(\mathcal{P})$ represents adjacent nodes. Each time, a new place $\mathcal{P}$ is added to the first level, the respective topological map $G^0(\mathcal{P})$ is added to the ground level. Furthermore, the robot determines whether it is possible to navigate from this place $\mathcal{P}$ to any other place $\mathcal{P}' \in \mathcal{P}_n$ in its vicinity and if so, this is represented by adding a transition edge

between the respective two nodes of the respective topological maps.

5.4. SLINK Hierarchy

The SLINK hierarchy T_n is then used to evolve the hierarchical map H. As such,

- The levels of the hierarchical map are obtained.
- Internal cost of upper level nodes are calculated.
- Transitions among places at different levels of the hierarchy are defined.

The structure of the hierarchy starting with the first level is obtained using the SLINK method [53]. This is an agglomerative clustering method. We prefer to use this method since it is also incremental. The similarity measure is based on geometrical proximity and connectivity. As such, partitioning of all the nodes is based on minimum distances. That makes sense as the resulting hierarchy encodes spatial knowledge.

The hierarchy T_n has a tree structure that is defined by a nested sequence of partitions of $\mathbb{P}_n$ based on their physical proximities. Thus, each node N in the tree represents a particular cluster of places $\mathbb{P}(N) \subseteq \mathbb{P}_n$. This is encoded by a partitioning function $\zeta : R^{\geq 0} \rightarrow E(\mathbb{P}_n)$ where $E(\mathbb{P}_n)$ denotes the set of equivalence relations on $\mathbb{P}_n$. The function ζ satisfies the following:

$$
\begin{aligned}
&0 \leq h \leq h' \rightarrow \zeta(h) \subseteq \zeta(h') \\
&\zeta(h + \delta h) = \zeta(h)\ \forall \delta h \approx 0 \\
&\exists h' \text{ s.t. } \mathbb{P}_n \in \zeta(h')
\end{aligned}
$$

The first constraint states that as h gets progressively larger, so do the clusters in size. Furthermore, we need to consider a finite number of h values - as specified by the second constraint. Finally, the last constraint ensures that the top level corresponds to a single cluster $\mathbb{P}(N) = \mathbb{P}_n$.

Given a particular $h \in R^{\geq 0}$, consider all subtrees whose edges link pairs of places from the $\mathbb{P}(N)$ and $\mathbb{P}(N')$ having a distance of at most h as measured by a distance function $d : \mathbb{P}_n \times \mathbb{P}_n \rightarrow R^{\geq 0}$. Then $\zeta(h)$ is the equivalence relation corresponding to the partition of $\mathbb{P}_n$ defined by the connected components of this graph.

Given two places $\mathcal{P}, \mathcal{P}' \in \mathbb{P}_n$, let $G^0(\mathcal{P})$ and $G^0(\mathcal{P}')$ be the respective topological maps associated with these places. Then, the distance between $d(\mathcal{P}, \mathcal{P}')$ is defined only if there is at least one transition edge between the nodes of their respective topological maps - namely:

$$d(\mathcal{P}, \mathcal{P}') = \begin{cases} |\mathcal{P} - \mathcal{P}'| & \text{if } \exists N \in \mathcal{N}^0(\mathcal{P}), N' \in \mathcal{N}^0(\mathcal{P}') \text{ s.t. } \iota(e(N, N')) = \text{transition} \\ \infty & \text{otherwise} \end{cases} \tag{5.4}$$

5.5. Evolving the Hierarchical Map

The hierarchical map H evolves as the robot navigates to different places and constructs their topological maps. In the first level of the hierarchy, the nodes correspond to distinct places $\mathbb{P}_n$ that have been learned so far. Here, each *place* is defined by the *collection of appearances or locations sharing common perceptual signatures or physical boundaries* [2]. These are obtained using the TSC model [2]. For completeness, a brief summary of the TSC model is presented in Appendix D. Places are added incrementally to the first level of the hierarchical map H using SLINK algorithm [53]. Suppose a new place $\mathcal{P}$ is added. Thus the set of places $\mathbb{P}_n$ now becomes $\mathbb{P}_{n+1}$. Let T_{n+1} be the corresponding SLINK hierarchy.

Each time the SLINK hierarchy is updated, the corresponding levels of the hierarchical map are determined in a bottom-up manner. Recall that the height h values can be ordered from $0, \ldots, h(L^r)$. The first level corresponds to the partition $\zeta(0)$. As h increases, a new hierarchy level L^i is assigned if there is significant reduction in the cardinality of the corresponding partition. This is computed using a function $\varsigma : Z \rightarrow R$ as follows: A level L^i is at the height $h \in [h(L^{i-1}), h']$ in the SLINK hierarchy at which

$\varsigma(|L^{i-1}|) \geq |\zeta(h)|$.

The algorithm used for evolving the hierarchical map is given in Figure F.9. Here, we initialize the lowest level plain graph consisting of places and apply SLINK clustering algorithm for each hierarchical level as described in Figure F.9. The addition of a new level L^{r+1} can easily be determined by considering the list $0, \ldots, h(L^r)$ and exploiting the binary nature of SLINK hierarchy. In other words, the cardinality of the partition $\zeta(h)$ increases or decreased by one when considering adjacent heights.

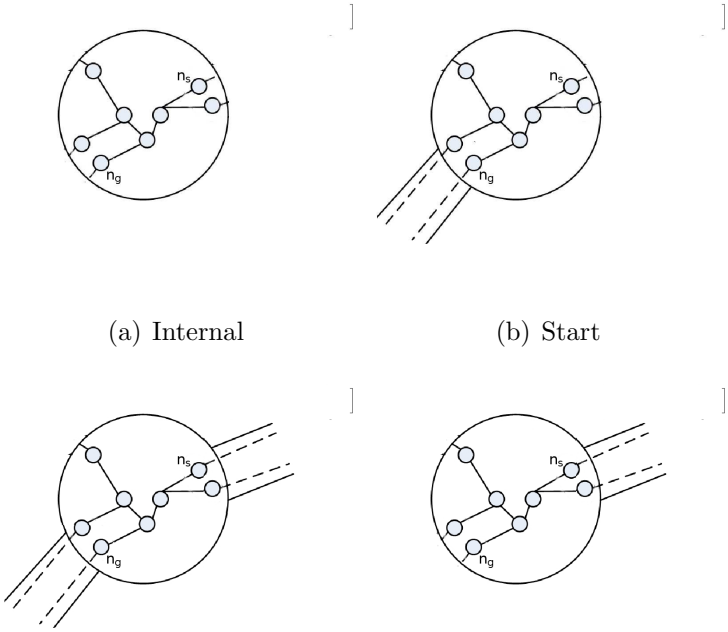
Next, for each level L^i, $i = 2, \ldots, r$, the associated nodes are determined based on the partitions $\zeta(h(L^i))$ in the previous SLINK hierarchy T_n. This is also done for the updated hierarchy T_{n+1}. The resulting nodes are compared and those that are different are determined based on whether their children nodes have changed or not. The changed nodes are then updated considering the new hierarchy T_{n+1}. First, their internal costs are re-computed. Costs are calculated by using shortest and loop-free longest paths between children of related nodes in the lower level. Next, edges are updated. The update can result in both the type or the cost of an edge to change. The details of the algorithm are given in Figure F.10.

5.6. Hierarchical Path Search

The goal is to use the hierarchy to reduce the computational cost of the search between two given nodes while being as close to optimality as possible.

There are 4 types of backtracking in refining as seen in Figure 5.1:

- Backtracing between the internal nodes (Figure 5.1(a)).
- Backtracing between a known starting node to the closest node which has a transition to target node at the upper level (Figure 5.1(b)).
- Backtracing between an arbitrary starting node which has a transition from the previous upper level node to the closest node which has a transition to target node at the upper level (Figure 5.1(c)).



(a) Internal (b) Start

(c) Transition (d) Goal

Figure 5.1. Backtracking at a node may occur through 4 mechanisms.

- Backtracing between an arbitrary starting node which has a transition from the previous uppler level node to the known goal node (Figure 5.1(d)).

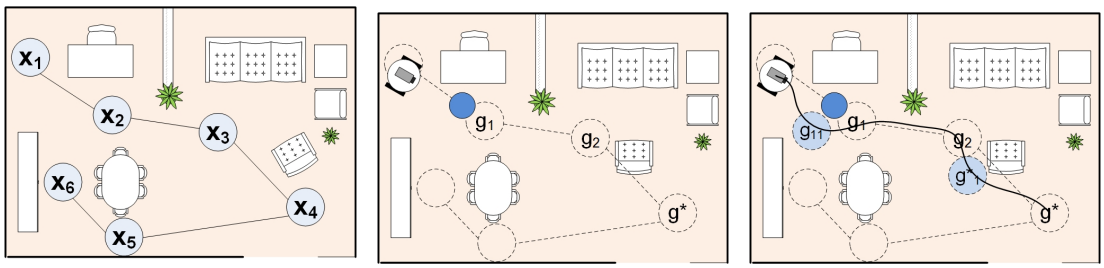
The algorithm is Recursive Hierarchical Path Search (RHPS) first considers L^i - the highest hierarchical level for the search and finds a path at L^i. Then it starts to refine each node in the resulting path until all nodes are extracted. Then the search recursively repeated for the L^{i-1}.

The level L^i can be found by finding the common ancestor node (a supernode) of n_s and n_g. This corresponds to level L^i. The pseudo code for RHPS is given in Figure F.14 with an interval cost based shortest path implementation given in Figure F.13. In this optimistic approach, only the minimum cost is considered during the calculations which may yield to some loss of optimality in some certain cases. A more clever implementation might be done by considering both minimum and maximum values together.

The search starts by refining the first common ancestor node at L^{h_c} via applying Figure F.13 between the ancestors of n_s and n_g at the level L^{h_c-1}. This first refinement does not require any transitions as start and goal nodes have the same parent. The next refinements, on the other hand, are not necessarily to be between same parents. For the lower levels, the first node refinement, as the starting node is known, search for the closest path to an arbitrary node which leads a route to the next parent calculated in the upper level refinement. The consecutive node refinement starts from the transition node found in the previous step and search for the closest node for the next parent known from the path found for the upper level. The level refinement ends with a search from an arbitrary transition node, from the previous parent, to the known goal not at this level.

5.7. Navigation With a Topological Map

The robot has a previously learned topological map $G = (\mathcal{N}, E)$ where $\mathcal{N}$ denotes the set of nodes and E denotes the set of edges. Let the cardinality of $\mathcal{N}$ be N_r. Each node is associated with depth data. The edges are labeled with relative geometry of the respective node pairs.



(a) (b) (c)

Figure 5.2. Navigation using a topological map of a 5m × 5 m room . (a) The topological map of the room; (b) The robot follows a sequence of nodes to the goal g^*; (c) The robot determines the waypoints by imagining the furthest visible location.

The robot should be capable of navigating to any goal location g^* within the coverage of the topological map and arrive there with the goal heading $\alpha^* \in S^1$ without any collisions along the way. The general algorithm is as follows: First, the robot determines the pair of nodes of the topological map that are closest to its initial and goal locations respectively. Following, it then determines a topological map path consisting of a sequence of nodes that leads to the goal position g^* using graph search algorithm such as Dijkstra.

Let this path be denoted by $P = \{g_i \in R^2\}_{i=1}^{N}$. Note that in order for the motion to be graceful, the path dictates that the robot arrives at each node g_i with a goal heading as well [19]:

$$\alpha_i = atan2((g_{i+1} - g_i) \cdot e_2, (g_{i+1} - g_i) \cdot e_1),\ i < N \tag{5.5}$$

where e_i denote the unit vectors in the local coordinate system. It then navigates from node g_{i-1} to the next g_i one-by-one until the goal location g^* is reached.

5.8. Experimental Results

The experiments are performed by using images and laser scan COLD Freiburg Indoor Dataset [60]. The plain graph have all the data points as the nodes and assign an edge between two nodes up 0.2 m as shown in Figure 5.3. The plain graph has 1459

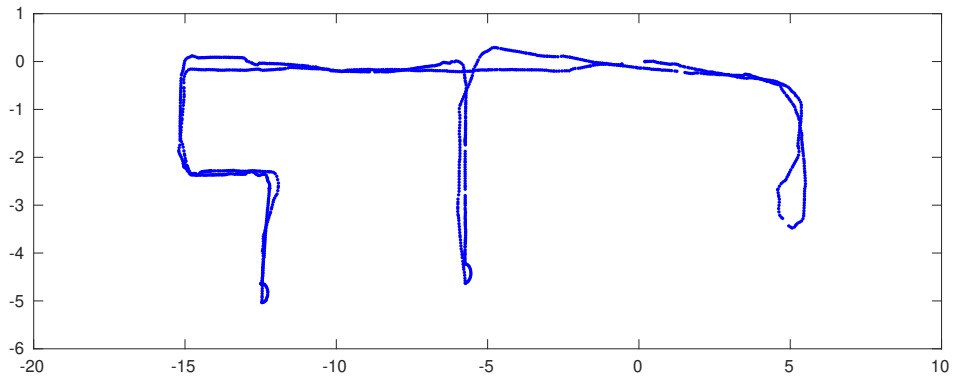


Figure 5.3. The plain graph structure with 1459 nodes and 21328 edges obtained by 0.2m maximum edge distance.

nodes and 21328 edges. The places are assigned each node manually considering the place detection results in [2]. Three different criteria ς_i are used for determining a new level:

(i) $\varsigma_1(i) = \lfloor |L^i|/5 \rfloor$
(ii) $\varsigma_2(i) = \lfloor |L^i|/3 \rfloor$
(iii) $\varsigma_3(i) = \sqrt{|L^i|}$

The results are evaluated in terms of processing time of 100 random initial-destination node pairs and the results are compared with the required computation time for plain graph search. On average 70% performance increase is observed. The plain graph search takes 0.16 seconds, whereas all of the other methods complete the search in 0.05 seconds. The reason of similarity between different hierarchy criteria might be the high ratio of the number of nodes plain graph level to the place level. The detailed results can be seen from Figure E.1-E.4

6. CONCLUSION

Chapter 2 presented a practical sensor-based reactive navigation method. Our method can be employed even with sparse topological maps and enables the robot to reach the nodes of the topological map with the desired heading. In this method, the control inputs are defined by the sequential composition of reactive controllers as parametrized by waypoints and the associated headings as well as obstacles that have been so far observed. Differing from related work, the robot computes the waypoints and the associated headings as it is navigating through continually imagining its motion and there are no assumptions regarding the geometry of environmental clutter. Our experimental results conducted in two different places under three scenarios varying in the number of newly appearing obstacles demonstrate that the robot is able to navigate successfully among the nodes of a given topological map regardless of its density and without any collisions along the way. Our ongoing work is focused on using this method for large-scale navigation.

Chapter 3 is on the discovery of canonical appearances in an unknown place. These appearances are integral to appearance-based place learning. In the proposed approach, the robot determines the vantage points for these appearances completely on its own through deciding how to explore the place. Differing from previous work, it does not stop until the place is fully covered - considering all accessible areas. A series of experiments using an outdoor benchmark data set including a comparative study demonstrate that resulting canonical appearances are both compact and comprehensive. Our ongoing work is focused on using the resulting knowledge in navigation and mapping.

In the Chapter 4 local interpolation based location estimation and DFC based heading methods are introduced. The local interpolation is shown to solve the averaging problem occurs in complete interpolation. Also an sensor fusion based on EKF provided some fast and smooth estimation results. As a future work a more clever way to detect and overcome sensory outliers can be considered.

Finally, in the Chapter 5 a hierarchical mapping approach is introduced. From the path search results it is shown that how these kind of structures can be useful for the scalability problem. An extension to the current study might be a method which adaptively choses new level assignment criteria.

REFERENCES

1. Erkent, O. and H. I. Bozma, "Bubble Space and Place Representation in Topological Maps", *The International Journal of Robotics Research*, Vol. 32, No. 6, pp. 671 – 688, 2013.

2. Karaoguz, H. and H. I. Bozma, "An Integrated Model of Autonomous Topological Spatial Cognition,", *Autonomous Robots*, Vol. 40, pp. 1379–1402, 2016.

3. Miller, S., *Space and Sense*, Psychology Press, 2008.

4. Kuipers, B., "The Spatial Semantic Hierarchy", *Artificial Intelligence*, Vol. 119, No. 1-2, pp. 191 – 233, 2000.

5. Yel, E. and H. I. Bozma, "Verifying the Recognized Place Through Localization", *Workshop on Introspective Methods for Reliable Autonomy, IEEE/RSJ Internation Conference on Intelligent Robots and Systems*, 2017.

6. Pomerleau, F., F. Colas and R. Siegwart, "A Review of Point Cloud Registration Algorithms for Mobile Robotics", *Foundations and Trends in Robotics*, Vol. 4, pp. 1–104, May 2015.

7. Kiriy, E. and M. Buehler, *Three-State Extended Kalman Filter for Mobile Robot Localization*, Tech. rep., 2002.

8. Sibson, R., "SLINK: an optimally efficient algorithm for the single-link cluster method", *The Computer Journal*, Vol. 16, No. 1, pp. 30–34, 1973.

9. Fernández-Madrigal, J.-A. and J. González-Jiménez, *Multi-hierarchical representation of large-scale space. Applications to mobile robots*, January 2001.

10. Pronobis, A., O. M. Mozos, B. Caputo and P. Jensfelt, "Multi-modal Semantic

Place Classification", *The International Journal of Robotics Research*, Vol. 29, No. 2-3, pp. 298–320, 2009.

11. Yel, E., *Appereance Based Self Localization and Navigation Using Place Memory*, M.S. Thesis, Bogazici University, 2014.

12. Akdeniz, B. C. and H. I. Bozma, "Exploration and topological map building in unknown environments", *International Conference on Robotics and Automation*, pp. 1079–1084, May 2015.

13. Fox, D., W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance", *IEEE Robotics Automation Magazine*, Vol. 4, No. 1, pp. 23–33, 1997.

14. Marder-Eppstein, E., "DWA ROS Wiki page", `http://wiki.ros.org/dwa_local_planner`, accessed in June 2018.

15. Karaman, S. and E. Frazzoli, "Sampling-based algorithms for optimal motion planning", *The International Journal of Robotics Research*, Vol. 30, No. 7, pp. 846–894, 2011.

16. Conner, D. C., H. Choset and A. A. Rizzi, "Integrating planning and control for single-bodied wheeled mobile robots", *Autonomous Robots*, Vol. 30, No. 3, pp. 243–264, April 2011.

17. Vasilopoulos, V. and D. E. Koditschek, "Reactive Navigation in Partially Known Non-Convex Environments?", *International Workshop on the Algorithmic Foundations of Robotics*, pp. 4445–4450, 2018.

18. Arslan, O. and D. E. Koditschek, "Sensor-based reactive navigation in unknown convex sphere worlds", *The International Journal of Robotics Research*, Vol. 38, No. 2-3, pp. 196–223, 2019.

19. Park, J. J. and B. Kuipers, "A smooth control law for graceful motion of differential

wheeled mobile robots in 2D environment", *2011 IEEE International Conference on Robotics and Automation*, pp. 4896–4902, May 2011.

20. Rimon, E. and D. E. Koditschek, "Exact robot navigation using artificial potential functions", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5, pp. 501–518, Oct 1992.

21. Arslan, O. and D. E. Koditschek, "Exact robot navigation using power diagrams", *IEEE International Conference on Robotics and Automation*, pp. 1–8, 2016.

22. Tron, E., "A Note on Rectangle Covering with Congruent Disks", , 2014.

23. Carmi, P., M. J. Katz and N. Lev-Tov, *Covering Points by Unit Disks of Fixed Location*, pp. 644–655, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

24. Durrant-Whyte, H. and T. Bailey, "Simultaneous localization and mapping: part I", *Robotics Automation Magazine*, Vol. 13, No. 2, pp. 99–110, June 2006.

25. Remolina, E. and B. Kuipers, "Towards a general theory of topological maps", *Artificial Intelligence*, Vol. 152, pp. 47–104, 2004.

26. Tapus, A. and R. Siegwart, "Incremental robot mapping with fingerprints of places", *Int. Conf. Intell. Rob. Sys.*, pp. 2429–2434, 2005.

27. Cummins, M. and P. Newman, "Appearance-only SLAM at large scale with FAB-MAP 2.0", *The Int. J. Rob. Res.*, Vol. 30, No. 9, pp. 1100–1123, 2011.

28. Thrun, S., *The Handbook of Brain Theory and Neural Networks*.

29. Deng, X., T. Kamade and C. Papadimitriou, "How to learn in an unknown environment", *32nd Symp. Foundations Comput. Sci*, pp. 298–303, 1991.

30. Moorehead, S., R. Simmons and W. Whittaker, "Autonomous exploration using multiple sources of information", *IEEE Int. Conf. Robot. Autom*, pp. 3098–3103,

2001.

31. Koenig, S. and C. Tovey, "Improved analysis of greedy mapping", *IEEE/RSJ Int. Conf. on IROS*, pp. 3251–3257, Las Vegas, 2003.

32. Yamauchi, B., "A frontier-based approach for autonomous exploration", *International Symposium on Intelligence in Robotics and Automation*, pp. 146–151, 1997.

33. Oriolo, G., G. Ulivi and M. Vendittelli, "Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments", *Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 28, No. 3, pp. 316–333, 1998.

34. Gonzalez-Banos, H. H. and J.-C. Latombe, "Navigation strategies for exploring indoor environments", *The International Journal of Robotics Research*, Vol. 21, No. 10, pp. 829–848, 2002.

35. Bourgoult, F., A. A. Makarenko, S. B. Williams, B. Grocholsky and F. Durrant-Whyte, "Information based adaptive robotic exploration", *International Conference on Intelligent Robots and Systems*, pp. 540–545, 2002.

36. Stachniss, C., G. Grisetti and W. Burgard, "Information gain-based exploration using Rao-Blackwellized particle filters", *Robotics: Science and Systems*, pp. 65–72, 2005.

37. Shade, R. and P. Newman, "Choosing where to go: Complete 3D exploration with stereo", *International Conference on Robotics and Automation*, pp. 2806–2811, 2011.

38. Paul, R., D. Feldman, D. Rus and P. Newman, "Visual precis generation using coresets", *International Conference on Robotics and Automation*, pp. 1304–1311, 2014.

39. Murphy, L. and G. Sibley, "Incremental Unsupervised Topological Place Discov-

ery,", *International Conference on Robotics and Automation*, pp. 1312 – 1318, June 2014.

40. Matsumoto, Y., M. Inaba and H. Inoue, "Visual navigation using view-sequenced route representation", *International Conference on Robotics and Automation*, Vol. 1, pp. 83–88 vol.1, 1996.

41. Fraundorfer, F., C. Engels and D. Nister, "Topological mapping, localization and navigation using image collections", *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3872–3877, Oct 2007.

42. Booij, O., B. Terwijn, Z. Zivkovic and B. Krose, "Navigation using an appearance based topological map", *International Conference on Robotics and Automation*, 2007.

43. Bonin-Font, F., A. Burguera, A. Ortiz and G. Oliver, "Combining obstacle avoidance with robocentric localization in a reactive visual navigation task", *International Conference on Industrial Technology*, pp. 19–24, 2012.

44. "Humanoid navigation using a visual memory with obstacle avoidance", *Robotics and Autonomous Systems*, Vol. 109, pp. 109 – 124, 2018.

45. Kuipers, B. J. and Y. T. Byun., "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations", *Journal of Robotics and Autonomous Systems*, Vol. 8, pp. 47–63, 1991.

46. Akdeniz, B. C. and H. I. Bozma, "Local terrain mapping via 3D laser based bubble surfaces", *European Conference on Mobile Robots*, pp. 44–49, 2013.

47. Tong, C. H., D. Gingras, K. Larose, T. D. Barfoot and Érick Dupuis, "The Canadian planetary emulation terrain 3D mapping dataset", *The International Journal of Robotics Research*, Vol. 32, No. 4, pp. 389–395, 2013.

48. Font-Llagunes, J. and J. Batlle, "Localization of a Mobile Robot With Omnidirectional Wheels Using Angular Kalman Filtering and Triangulation", Vol. 2006, 2006.

49. Stimec, A., M. Jogan and A. Leonardis, "Unsupervised Learning of a Hierarchy of Topological Maps Using Omnidirectional Images", *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 22, pp. 639–665, 01 2008.

50. Lim, J., J.-M. Frahm and M. Pollefeys, "Online environment mapping using metric-topological maps", *The International Journal of Robotics Research*, Vol. 31, No. 12, pp. 1394–1408, 2012.

51. Garcia-Fidalgo, E. and A. Ortiz, "Hierarchical Place Recognition for Topological Mapping", *Transactions on Robotics*, Vol. 33, No. 5, pp. 1061–1074, 2017.

52. "Graph clustering", *Computer Science Review*, Vol. 1, No. 1, pp. 27 – 64, 2007.

53. Sibson, R., "SLINK: An optimally efficient algorithm for the single-link cluster method", *The Computer Journal*, Vol. 16, No. 1, pp. 30–34, 1973.

54. Holzer, M., F. Schulz and D. Wagner, "Engineering Multilevel Overlay Graphs for Shortest-path Queries", *Journal of Experimental Algorithmics*, Vol. 13.

55. Madkour, A., W. G. Aref, F. U. Rehman, M. A. Rahman and S. Basalamah, "A Survey of Shortest-Path Algorithms", *arXiv preprint arXiv:1705.02044*, 2017.

56. Pomerleau, F., F. Colas, R. Siegwart and S. Magnenat, "Comparing ICP Variants on Real-World Data Sets", *Autonomous Robots*, Vol. 34, No. 3, pp. 133–148, 2013.

57. Blöchliger, F., M. Fehr, M. Dymczyk, T. Schneider and R. Siegwart, "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps", *arXiv preprint arXiv:1709.05533*, 2017.

58. Ravankar, A. A., A. Ravankar, T. Emaru and Y. Kobayashi, "A hybrid topological

mapping and navigation method for large area robot mapping", *Annual Conference of the Society of Instrument and Control Engineers of Japan*, pp. 1104–1107, 2017.

59. Zhang, Q., X. Wu, B. Liu, A. H. Adiwahono, T. A. Dung and T. W. Chang, "A hierarchical topological planner for multi-storey building navigation", *International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 731–736, 2015.

60. Pronobis, A. and B. Caputo, "COLD: COsy Localization Database", *Int. J. Robot. Res.*, Vol. 28, No. 5, pp. 588–594, 2009.

61. Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, Vol. 1, No. 1, pp. 269–271, 1959.

62. Hart, P. E., N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *Transactions on Systems Science and Cybernetics*, Vol. SSC-4(2), pp. 100–107, 1968.

63. Dechter, R. and J. Pearl, "Generalized Best-first Search Strategies and the Optimality of A*", *Journal of the ACM*, Vol. 32, No. 3, pp. 505–536, 1985.

64. Hoppe, H., T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, "Surface Reconstruction from Unorganized Points", *Annual Conference on Computer Graphics and Interactive Techniques*.

65. Erkent, O., H. Karaoguz and H. I. Bozma, "Hierarchically self-organizing visual place memory", *Advanced Robotics*, Vol. 31, No. 16, pp. 865–879, 2017.

APPENDIX A: TARGET ORIENTATION BASED A*

It is quite popular to choose grid based mapping representation for local navigation purposes. Ones the map constructed in this form each grid becomes a node connected with its specified neighbors with some edge cost depending on the distance between each grid. Then, the local navigation problem can be solved by some well-known shortest path algorithms such as [61]. It is actually a special case of A* algorithm [62] with heuristic function $h(c) = 0$. The heuristic functions are used to add some bias toward goal node and as long as they do not overestimate the real cost to the goal node, in other words it is admissible [63]. If we assume a uniform physical environment which does not have any preferable area for the movement of a mobile robot, then simply the euclidean distance is an admissible heuristic and converges to the optimal path much faster than standard Dijkstra's algorithm.

Heuristic functions can also be used for the purpose adding some goal orientation bias to the path with the APF modification done in Equation 2.4. The formulation will be as the following:

$$h(c, c_g, \alpha_g) = |c_g - c| + \frac{\tau_b}{|c_g - c|} \left(1 - \begin{bmatrix} \cos(\alpha_g) & \sin(\alpha_g) \end{bmatrix} \begin{bmatrix} \cos(\angle(c_g - c)) \\ \sin(\angle(c_g - c)) \end{bmatrix} \right) \tag{A.1}$$

Here, c_g and α_g are target position and heading respectively. First additive term is simply the euclidean heuristic. The second term adds an extra cost for the positions c whose unit vector to c_g is a smaller dot product with the unit vector defining α_g. This bias is controlled by the parameter τ_b and the bias increases as the position c becomes closer to the target c_g.

APPENDIX B: BUBBLE SPACE

Bubble space representation is briefly summarized for completeness. For details, interested readers are kindly referred to [1]. The bubble space $\mathcal{B} = \mathcal{X} \times \mathcal{F}$ is an abstract representation of the robot's base along with its viewing directions with $\mathcal{X} \subseteq \mathbb{R}^2$ and $\mathcal{F} \subseteq SO(2)$. Each point $b \in \mathcal{B}$ is defined as $b = [x\, f]^T$ where $x \in \mathcal{X}$ and $f \in \mathcal{F}$. In bubble space, for each feature i in $\mathcal{V}$, the robot is visualized to be surrounded by an hypothetical spherical surface $B_i(x_k, t)$) - referred to as bubble surface:

$$B_i(x_k, t) = \left\{ \begin{bmatrix} f \\ \rho_i(b, t) \end{bmatrix} \mid \forall f \in \mathcal{F} \text{ and } b = [x_k\, f]^T \right\} \tag{B.1}$$

where $\rho_i : \mathcal{B} \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ is a Riemannian metric encoding the response to the v_i^{th} sensory feature. Each bubble surface is initialized to be a S^2 sphere with radius $\rho_0 \in \mathbb{R}^{\geq 0}$ – namely $\rho_i(b, 0) = \rho_0$. As the robot looks around, for each viewing direction $f \in \mathcal{F}$, an observation $q_i(b, t)$ is made, each bubble surface is deformed at the corresponding bubble point b as $\rho_i(b, t^+) = q_i(b, t)$ where the superscript t^+ denotes time just after t.

As the robot's sensor moves through a sequence of N_s viewing directions, it makes a sequence of sensory observations. Correspondingly, each bubble surface $B_i(x_k, t)$ is deformed exactly at the corresponding N_s points. The interested reader is referred to [1] for details. As such, each node is associated with a set of bubble surfaces.

APPENDIX C: GROUND SLANT RECTIFICATION

Consider the raw laser data as obtained from the two-dimensional laser sensor. As explained, being on a slanted ground may deceive the robot in regards to obstacles and open passages. For example, in case when the robot is at the base of a sloped terrain, it will detect an obstacle whereas this is not the case at all. The raw laser data is processed in order to remove such data and rectify the slanted ground effects. A well-known technique which can be used to solve this problem is given in [64].

First of all the robot is assumed to know the inclination of its current position. For each laser reading point, a normal direction is calculated through fitting a plane using the readings from n_f viewing directions that are at least τ_e (e.g 0.3 m) distant from the reading point. This τ_e condition ensures that the estimated surface is less affected by noise. Whenever the angle between the estimated normal and the gravity direction is larger than 45°, the corresponding point needs to be considered an obstacle and is thus added to the set $\mathcal{Z}$. In the next step, the bubble surface is constructed by projecting the obstacles $p \in \mathcal{Z}$ back to the subset of bubble space corresponding to x_k and $f_2 = 0$. If $p = \begin{bmatrix} \Delta_x & \Delta_y & \Delta_z \end{bmatrix}^T$, then the resulting bubble surface is:

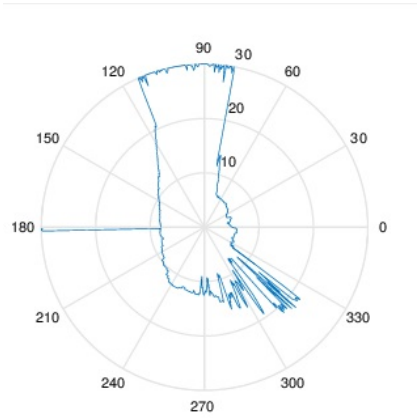
$$B_l(x_k) = \left\{ \begin{bmatrix} f_1 \\ 0 \\ \min\limits_{b' \in B^o} \rho_l(b') \cos(f_2) \end{bmatrix} \mid b \in \mathcal{B}^o, b = [x_k \, f]^T \right\} \tag{C.1}$$

where

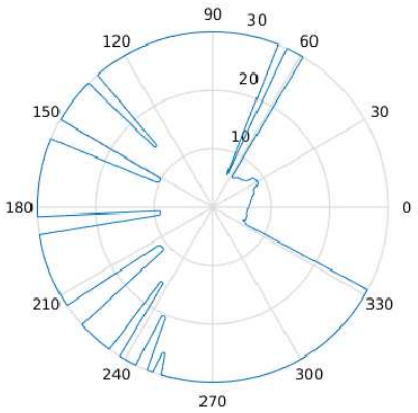
$$\mathcal{B}^o = \left\{ \begin{bmatrix} x_k \\ f'(p) \end{bmatrix} \mid p \in \mathcal{Z} \right\} \text{ and } f'(p) = \begin{bmatrix} \tan^{-1}(\frac{\Delta_y}{\Delta_x}) \\ \cos^{-1}\left(\frac{\sqrt{\Delta_x^2 + \Delta_y^2}}{\Delta_z}\right) \end{bmatrix}^T \tag{C.2}$$

The cases of single tilt projection ($f_h = -5°$) and ground slant rectified laser are given in Fig. C.1(b) and Fig. C.1(c) respectively. In the latter the spuriously obstructed regions due to slope effect are cleared in addition to using the full range of 30 meters

(a) Raw laser data $B_l^r(x_k)$.



(b) Single tilt ($f_h = -5°$) projection



(c) Ground slant rectified projection $B_l(x_k)$

Figure C.1. Laser bubble surface.

effectively.

APPENDIX D: TSC MODEL

The TSC model enables a robot to build its knowledge of places and refer to this knowledge in a completely autonomous manner. The model works in conjunction with a place memory that is powered by the operations on the consecutive appearances. No prior information of environment or localization are required. There are two integral parts: spatial memory and processing modules. In this section, we briefly summarize the relevant aspects. Due to space restriction, interested readers are kindly referred to [2] for details.

Place memory enables the robot to store and retrieve the knowledge of learned places as indexed by the set $\mathcal{P}$ [65]. It is organized hierarchically as defined by a nested sequence of partitions of the set $\mathcal{P}$. The partition at the top level is the whole set $\mathcal{P}$. All inner nodes correspond to particular subclusters while each of terminal nodes corresponds to a distinct place $p \in \mathcal{P}$. Such an organization allows the robot to associate with its learned place knowledge efficiently. Furthermore, the memory is ensured of having both storage and construction efficiency as well as being order-invariant. Its structure evolves in an unsupervised and incremental manner and is viewed as encoding the semantic hierarchy among different places. In particular, places belonging to each cluster can be viewed as sharing certain common attributes. As there are no externally provided labels expressed in natural language, human users can make such associations after analyzing the memory contents.

There are three relevant capabilities: place detection, recognition, and learning. These get activated when necessary as the robot navigates through a sequence of locations $c_k \in R^2$ and headings $\alpha_k \in S^1$ $k \in \mathcal{K}$ where $\mathcal{K}$ is the index set. The appearance from each location c_k is then internally encoded by a d-dimensional descriptor $I(c_k) \in R^d$. The goal of place detection is to partition the index set $\mathcal{K}$ so that appearances belonging to each distinct place are grouped together as $\mathcal{D} \subset \mathcal{K}$. The partitioning process enables the robot to know where a place starts and ends. It is achieved by the iterative clustering of the index set $\mathcal{K}$ - considering the informative-

ness, coherency, and plenitude of the associated visual descriptors that are obtained from the incoming sequence of appearances. As such, each detected place is associated with a set of appearances that describe the place. The plenitude check ensures that the number of locations is large enough as to enable the robot to have a sufficient knowledge of the place.

Whenever a place $\mathcal{D}$ is detected, the robot then attempts to recognize it as one of the learned places $p \in \mathcal{P}$ via relating the collected appearances to its place memory - as detailed in [65]. This is based on associating the current appearances (if possible) with those retained in the place memory through traversing down the memory hierarchy. The traversal proceeds downwards level-by-level and decisions are progressively combined to hierarchically refine the final decision. Consider a place memory with N_l levels. At each level l, $l = 1, \ldots, N_l$, a decision regarding one attribute is made by choosing a particular node $N \in S(N^{l-1})$ among children $S(N^{l-1})$ nodes of node N^{l-1} that has been reached at level $l-1$. The decision-making is based of finding the node N with the minimum cost function $g_N(D)$ while ensuring that the minimum cost is below a recognition cost threshold τ_r.

$$N \in \arg \min_{N' \in S(N^{l-1})} g_{N'}(D) \tag{D.1}$$

subject to

$$g_N(D) \leq \tau_r \tag{D.2}$$

This process is repeated either until the condition of Eq. D.2 is not satisfied or terminal nodes (final level) is reached. In the former case, no decision is made while in the latter, the place is recognized to be the place associated with the terminal node. Due to space restrictions, the interested reader is referred to [65] for details. The recognition threshold τ_r is a designating factor in the tradeoff between precision and recall. As its value is decreased, while precision increases, recall decreases. In case of recognition, the robot simply updates its memory via incorporating the new knowledge appropriately

and goes back to the beginning where it waits for new sensory input.

In case of no recognition, it invokes place learning in order to add the detected place D into its place memory. Place learning enables the robot to add the new knowledge to its place memory. Whenever a place is learned, the cardinality of the set $\mathcal{P}$ increased by one. This is achieved using the hierarchical single link clustering method SLINK [65]. The clustering is incremental with both storage and construction efficiency as well as the resulting hierarchy being order-invariant. Thus, it enables the robot to evolve its place memory over time as it detects places, but cannot recognize them. The clustering is done in the appearance space so that each place p is learned based on appearances from a set of $M_p = |D|$ locations associated with the corresponding detected place D. The number M_p of learned locations (while being greater than the plenitude threshold) actually depends on how much the robot moves in this place during place detection.

APPENDIX E: Hierarchical Mapping Results

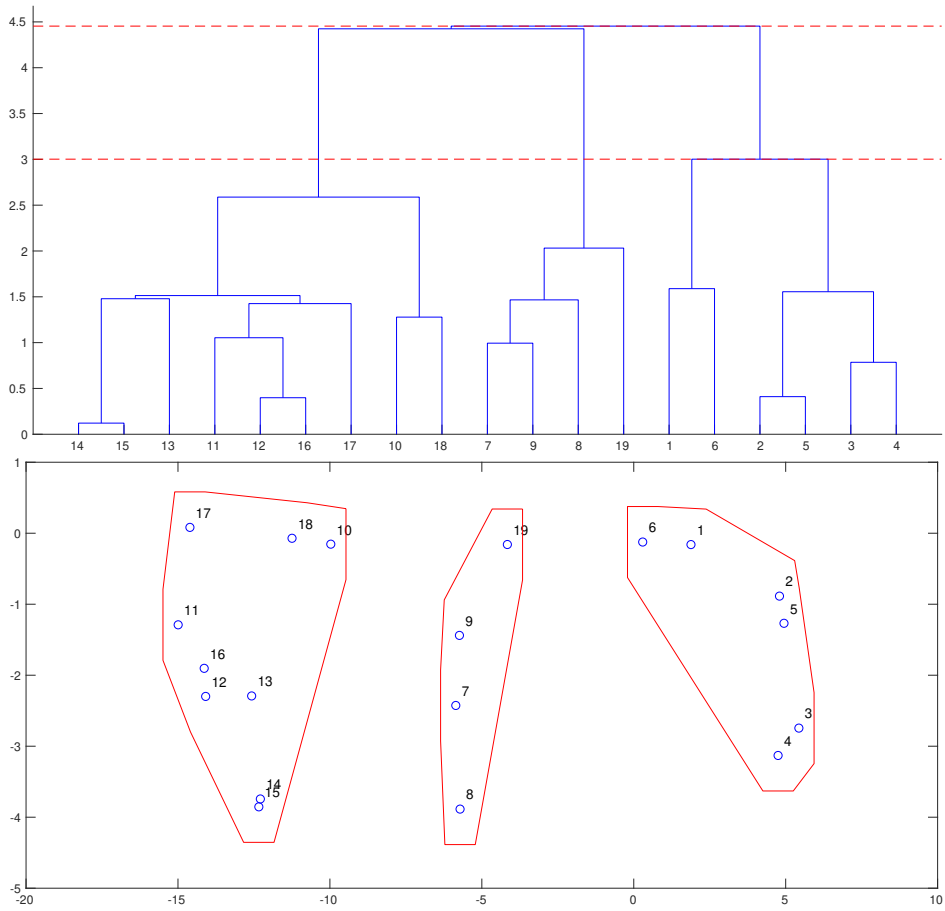


Figure E.1. Hierarchical structure for $\varsigma(|L^i|) = \lfloor |L^i|/5 \rfloor$: Dendrogram (top), the first level(bottom).

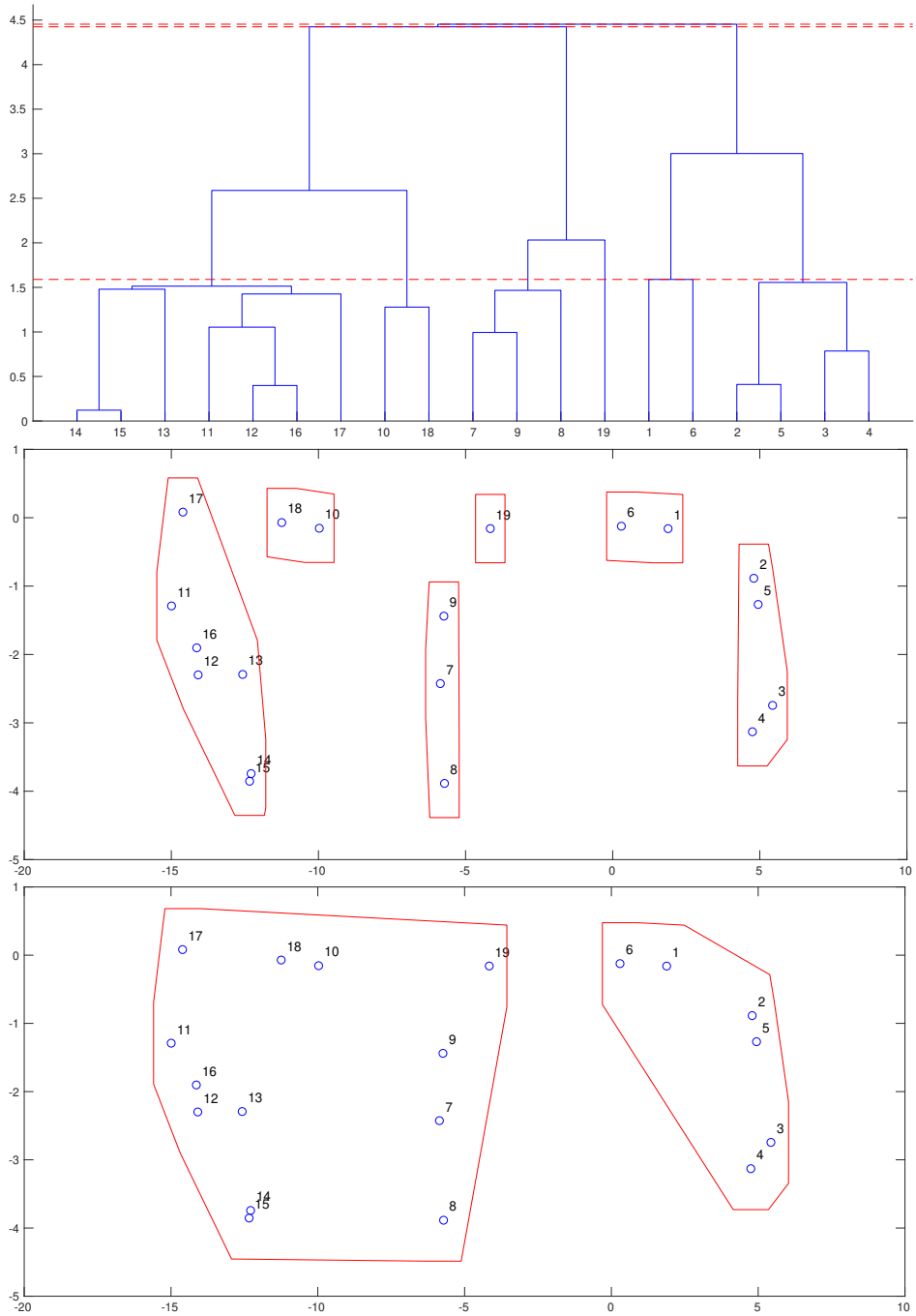


Figure E.2. Hierarchical structure for $\varsigma(|L^i|) = \lfloor |L^i|/3 \rfloor$: Dendrogram (top), the first level(middle), the second level(bottom).

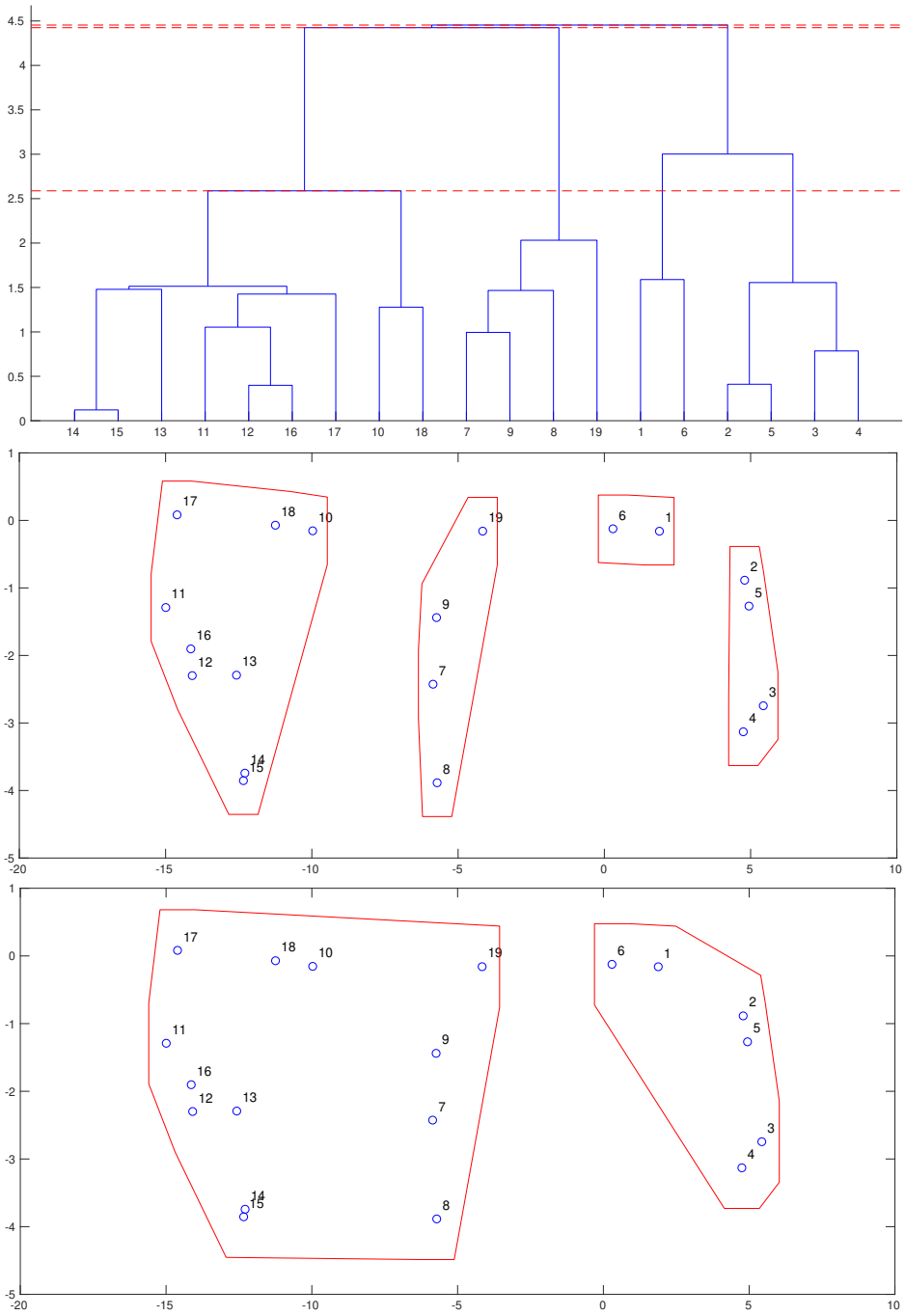


Figure E.3. Hierarchical structure for $\varsigma(|L^i|) = \sqrt{|L^i|}$: Dendrogram (top), the first level(middle), the second level(bottom).

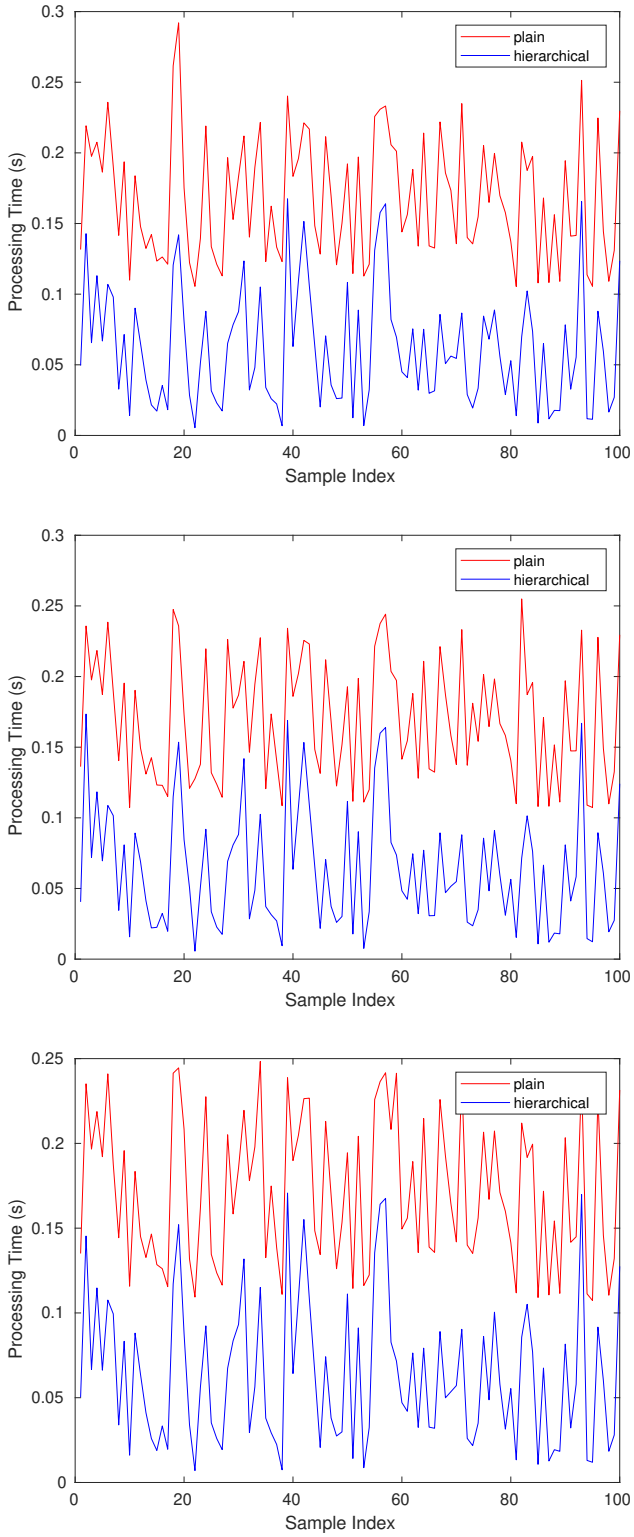


Figure E.4. Path search performance comparison with respect to the plain graph search: $\varsigma(|L^i|) = \lfloor |L^i|/5 \rfloor$ (left), $\varsigma(|L^i|) = \lfloor |L^i|/5 \rfloor$ (middle), $\varsigma(|L^i|) = \sqrt{|L^i|}$ (right).

APPENDIX F: ALGORITHMS

procedure ESTIMATEPOSITION($I_c, I_p[n_{samples}, loc[n_{samples}]$)
    $dist\_mat = inf$
    for $i = 1 : n_{samples}$ **do**
        for $j = i + 1 : n_{samples}$ **do**
            $dist\_mat[i][j] = |I_p[i] - I_p[j]|$
            $dist\_mat[j][i] = |I_p[i] - I_p[j]|$
        end for
    end for
    $\tau_d = min(dist\_mat) + \tau_{dm}$ ▷ add some margin for nonzero sets
    $min\_dist = |I_c - I_p[1]|$
    $min\_idx = 1$
    for $i = 2 : n_{samples}$ **do**
        if $|I_c - I_p[i]| < min\_dist$ **then**
            $min\_dist = |I_c - I_p[i]|$
            $min\_idx = i$
        end if
    end for
    $\Omega_1 = \emptyset$
    for $i = 1 : n_{samples}$ **do**
        if $\frac{|I_c - I_p[i]| - min\_dist}{|I_c - I_p[i]| + min\_dist} < \tau_I$ **then**
            $\Omega_1.push\_back(i)$
        end if
    end for

Figure F.1. Estimate Position Algorithm - Part 1.

```
24:  Ω_2 = ∅
25:  for i = 1 : Ω_1.size() do
26:      tmp = ∅
27:      for j = 1 : n_samples do
28:          if dist_mat[Ω_1[i]][j] < τ_d then
29:              tmp.push_back(j)
30:          end if
31:      end for
32:      Ω_2.push_back(tmp)
33:  end for
34:  ĉ = ∅
35:  w_list = ∅
36:  for i = 1 : Ω_1.size() do                    ▷ interpolate for each candidate
37:      weight = ∅
38:      count = ∅
39:      tmp_ĉ = {0, 0}
40:      weight_sum = 0
41:      τ_dl = 0                                 ▷ calculate local τ_d
42:      min_dist_l = |I_p[Ω_2[i][1]] − I_p[Ω_2[i][2]]|
43:      for j = 1 : Ω_2[i].size() do
44:          for k = j + 1 : Ω_2[i].size() do
45:              if |I_p[Ω_2[i][j]] − I_p[Ω_2[i][k]]| < min_dist_l then
46:                  min_dist_l = |I_p[Ω_2[i][j]] − I_p[Ω_2[i][k]]|
47:              end if
48:          end for
49:      end for
50:      τ_dl += τ_dm
```

Figure F.2. Estimate Position Algorithm - Part 2.

```
51:         for j = 1 : Ω2[i].size() do
52:             tmp_count = 1
53:             for k = j + 1 : Ω2[i].size() do
54:                 if |Ip[Ω2[i][j]] − Ip[Ω2[i][k]]|< τdl then
55:                     tmp_count + +
56:                 end if
57:             end for
58:             count.push_back(tmp_count)
59:         end for
60:         weight_sum = 0
61:         for j = 1 : Ω2[i].size() do                          ▷ calculate weights
62:             tmp_ĉ+ = e^{−|Ic−Ip[Ω2[i][j]]|^2}/count[j] ∗ loc[Ω2[i][j]]
63:             weights.push_back(e^{−|Ic−Ip[Ω2[i][j]]|^2}/count[j])
64:             weight_sum+ = e^{−|Ic−Ip[Ω2[i][j]]|^2}/count[j]
65:         end for
66:         tmp_ĉ/= weight_sum; ĉ.push_back(weight_sum); w_list.push_back(weight)
67:     end for
68:     min_cost = inf; result = 0, 0
69:     for i = 1 : Ω1.size() do                  ▷ choose the best cost candidate
70:         tmp_cost = 0
71:         for j = 1 : Ω2[i].size() do
72:             tmp_cost+ = w_list[i][j]|ĉ[i] − loc[Ω2[i][j]]|^2
73:         end for
74:         if tmp_cost < min_cost then
75:             min_cost = tmp_cost;result = ĉ[i]
76:         end if
77:     end for
        return result
78: end procedure
```

Figure F.3. Estimate Position Algorithm - Part 3.

```
1: procedure ESTIMATEHEADING(DFC_c, DFC_p, h_max)
2:     result = 0; coeff_count = 0; sin_sum = 0; cos_sum = 0
3:     for i = 1 : h_max do
4:         for j = 1 : h_max do
5:             num_s = DFC_c.a[i][j] * DFC_p.b[i][j] − DFC_c.b[i][j] * DFC_p.b[i][j]
6:             num_c = DFC_c.a[i][j] * DFC_p.a[i][j] + DFC_c.b[i][j] * DFC_p.b[i][j]
7:             denum = DFC_c.a[i][j] * DFC_p.a[i][j] + DFC_c.b[i][j] * DFC_p.b[i][j]
8:             sin_k = num_s/denum;cos_k = num_c/denum
9:             da = atan2(sin_k, cos_k)/i
10:            if !isnan(da) then
11:                result = result + da
12:                sin_sum+ = sin(da);cos_sum+ = cos(da)
13:                coeff_count + +
14:            end if
15:            num_s = DFC_c.c[i][j] * DFC_p.d[i][j] − DFC_c.d[i][j] * DFC_p.c[i][j]
16:            num_c = DFC_c.c[i][j] * DFC_p.c[i][j] + DFC_c.d[i][j] * DFC_p.d[i][j]
17:            denum = DFC_c.c[i][j] * DFC_c.c[i][j] + DFC_c.d[i][j] * DFC_c.d[i][j]
18:            sin_k = num_s/denum;cos_k = num_c/denum
19:            da = atan2(sin_k, cos_k)/i
20:            if !isnan(da) then
21:                result = result + da
22:                sin_sum+ = sin(da);cos_sum+ = cos(da)
23:                coeff_count + +
24:            end if
25:        end for
26:    end for
27:    result = atan2(sin_sum/coeff_count, cos_sum/coeff_count)
       return result
28: end procedure
```

Figure F.4. Estimate Heading Algorithm.

```
1: procedure FIND STATIC OBSTACLES(O^c, O^r, τ_s)
2:     for i = 1 : O^r.size do
3:         for j = 1 : O^c.size do
4:             if dist(O^r[i], O^c[j]) < τ_s then
5:                 O^r_s.push_back(O^r[i])
6:                 O^c_s.push_back(O^c[j])
7:                 break
8:             end if
9:         end for
10:    end for
11:    return O^c_s, O^r_s
12: end procedure
```

Figure F.5. Find Static Obstacles Algorithm.

```
1: procedure MATCH STATIC OBSTACLES(O^c_s, O^r_s)
2:     for i = 1 : O^c_s.size do
3:         d_min ← infinity
4:         m_i ← 0
5:         for j = 1 : O^r_s.size do
6:             d ← dist(O^c[i]), O^r[j]
7:             if d < d_min then
8:                 d_min ← d
9:                 m_i ← j
10:            end if
11:        end for
12:        M.push_back(m_i)
13:    end for
14:    return M
15: end procedure
```

Figure F.6. Match Static Obstacles Algorithm.

```
1: procedure CORRECT HEADING(O^c, O^r, τ_s, Δθ)
2:     n_max ← 0
3:     θ* ← 0
4:     count ← −1
5:     for i = 0 : Δθ : 2π do
6:         Õ^c ← R_i O^r
7:         O^c_s, O^r_s =FIND STATIC OBSTACLES(Õ^c, O^r, τ_s)
8:         M =MATCH STATIC OBSTACLES(O^c_s, O^r_s)
9:         if M.size == n_max and count > −1 then
10:            count ← count + 1
11:            θ* = i + count Δθ/2
12:        else if M.size > n_max then
13:            n_max = M.size
14:            θ* = i
15:            count ← 0
16:        else
17:            count ← −1
18:        end if
19:    end for
20:    return θ*
21: end procedure
```

Figure F.7. Heading Correction Algorithm.

```
1: procedure CORRECT POSE(δc_0, O^c, O^r, τ_s, Δθ, n_max, τ_c)
2:     δc* ← δc_0
3:     for i = 1 : O^c.size do
4:         O^c[i].x ← O^c[i].x + δc_0.x
5:         O^c[i].y ← O^c[i].y + δc_0.y
6:     end for
7:     n ← 0
8:     δc ← infinity
9:     while n < n_max and δc > τ_c do
10:        δc ← 0
11:        θ = CORRECT HEADING(O^c, O^r, τ_s, Δθ)
12:        O^c ← R_θ O^c
13:        O^c_s, O^r_s =FIND STATIC OBSTACLES(O^c, O^r, τ_s)
14:        M =MATCH STATIC OBSTACLES(O^c_s, O^r_s)
15:        for i = 1 : O^c_s.size do
16:            δc.x ← δc.x + O^r_s[M[i]].x − O^c_s[i].x
17:            δc.y ← δc.y + O^r_s[M[i]].y − O^c_s[i].y
18:        end for
19:        δc.x ← δc.x/O^c_s.size
20:        δc.y ← δc.y/O^c_s.size
21:        for i = 1 : O^c.size do
22:            O^c[i].x ← O^c[i].x + δc.x
23:            O^c[i].y ← O^c[i].y + δc.y
24:        end for
25:        δc* ← R_θ* δc* + δc
26:        θ* ← θ* + θ
27:        n ← n + 1
28:    end while
29:    return δc*
30: end procedure
```

Figure F.8. Pose Correction Algorithm.

```
1: procedure UPDATETREE(tree, n_new)
2:     [n*, h] = findClosestNeighbour(tree_old, n_new)
3:     tree.add(n*, n_new, h)
4: end procedure
```

Figure F.9. Tree Update Algorithm.

```
1: procedure DETERMINELIST(list, tree, ς)
2:     k = #nodes, r = 1
3:     for i = 1 : k − 1 do
4:         if (k − i + 1) ≥ ς(|list(r)|) then
5:             h(list(r)) = h(tree(i))
6:         else
7:             r = r + 1
8:             if  then|list|< r
9:                 list.add(k − i + 1, h(tree(i)))
10:             end if
11:         end if
12:     end for
13: end procedure
```

Figure F.10. Determine Level List Algorithm.

```
1: procedure COMPARENODES(m, list_prev, list_new)
2:     ch_prev = assignChildren(list_prev)
3:     ch_new = assignChildren(list_new)
4:     for ∀n^i_{prev,new} do
5:         n^{i+1}_{(prev,new)idx} = [min(ch^i_{(prev,new)idx}), max(ch^i_{(prev,new)idx})]
6:         sl_{prev,new}(n^{i+1}_{(prev,new)idx}).add(n^i_{prev,new})
7:     end for
8:     [sl'_prev, sl'_new] = findUnmatchedSets(sl_prev, sl_new)
9:     [{n^i_idx}, {n^{i+1}_{(prev)idx}}, {n^{i+1}_{(new)idx}}] = compare(sl'_prev, sl'_new)
10: end procedure
```

Figure F.11. Compare Nodes Algorithm.

```
1: procedure compareLists(sl_p, sl_n)
2:     for ∀n_i ∈ sl_n do
3:         p_ip = parent(sl_p, n_i)
4:         p_in = parent(sl_n, n_n)
5:         for ∀n_j ∈ neighbour(n_i, sl_n) do
6:             if p_in = parent(sl_n, n_j) then
7:                 p_in.edge_list.add_edge(n_i, n_j)
8:             else
9:                 if p_ip = parent(sl_p, n_j) then
10:                    p_in.edge_list.delete(n_i, n_j)
11:                end if
12:                n_i.add_trans(n_j, parent(sl_n, n_j))
13:                n_j.add_trans(n_i, p_in)
14:            end if
15:        end for
16:    end for
17: end procedure
```

Figure F.12. Compare Lists Algorithm.

```
1: procedure SHORTESTPATH(n_s, n_g)
2:     for ∀n_i ≠ n_s do
3:         t_cost(n_i) = [inf, inf]
4:         finished(n_i) = false
5:     end for
6:     t_cost(n_s) = [0, 0]
7:     n_proc = n_s
8:     while n_proc ≠ n_g do
9:         for ∀n_i ∈ neighbour(n_proc) do
10:            if t_cost(n_i, 1) > t_cost(n_proc, 1) + internalCost(n_proc, 1) then
11:                t_cost(n_i) = t_cost(n_proc) + internalCost(n_proc)
12:            end if
13:        end for
14:        finished(n_proc) = true
15:        n_proc = argmin_{n∈{n|finished(n)=false}} t_cost(n, 1)
16:    end while
17:    path = {n_g}
18:    while path.end() ≠ n_s do
19:        path.add(argmin_{n∈neigbour(path.end())} t_cost(n, 1))
20:    end while
21:    path.reverse() return path
22: end procedure
```

Figure F.13. Interval Cost Shortest Path Algorithm.

1: Initialize n_s, n_g
2: $k, l = 0$
3: **while** $s^{i,k}(n_s) \neq s^{j,l}(n_g)$ **do** ▷ find the first common ancestor
4: **if** $h(s^{i,k}(n_s)) \leq h(s^{j,l}(n_h))$ **then**
5: $k = k + 1$
6: **end if**
7: **if** $h(s^{i,k}(n_s)) \geq h(s^{j,l}(n_h))$ **then**
8: $l = l + 1$
9: **end if**
10: **end while**
11: $h_c = i + k = j + l$ ▷ the first common ancestor
12: $P^{h_c}(n_s, n_g) = \{s_n^{i,k}(n_s)\}$
13: $h_c = h_c - 1$
14: **for** $m = 1 : min(k, l)$ **do** ▷ repeat for each level
15: $n'_s = n_s$ ▷ initial node is known
16: $n'_g = P^{l_c+1}(n_s, n_g)(2)$
17: $v_s = [s_n^{i,k-1}(n'_s), s_n^{i,k}(n'_s)]$ ▷ both place and node idx passes to the algorithm
18: $v_g = [s_n^{j,l-1}(n'_g), s_n^{j,l}(n'_g)]$ ▷ as the latter is not available at the lower level
19: $P^{l_c}(n_s, n_g) = shortestPath(v_s, v_g, G^{l_c}(n'_s))$
20: **for** $p = 2 : |P^{h_c+1}(n_s, n_g)|-1$ **do** ▷ intermediate nodes
21: $n'_s = P^{l_c+1}(n_s, n_g)(p)$
22: $n'_g = P^{l_c+1}(n_s, n_g)(p + 1)$
23: $v_s = [s_n^{i,k-1}(n'_s), s_n^{i,k}(n'_s)]$
24: $v_g = [s_n^{j,l-1}(n'_g), s_n^{j,l}(n'_g)]$
25: $P^{l_c}(n_s, n_g) = P^{l_c}(n_s, n_g) + shortestPath(v_s, v_g, G^{l_c}(n'_s))$
26: **end for**
27: $n'_s = P^{l_c+1}(n_s, n_g)(p)$ ▷ final node is known
28: $n'_g = n_g$
29: $v_s = [s_n^{i,k-1}(n'_s), s_n^{i,k}(n'_s)]$
30: $v_g = [s_n^{j,l-1}(n'_g), s_n^{j,l}(n'_g)]$
31: $P^{l_c}(n_s, n_g) = P^{l_c}(n_s, n_g) + shortestPath(v_s, v_g, G^{l_c}(n'_s))$
32: $h_c = h_c - 1$ ▷ switch to the lower level
33: $k = k - 1$
34: $l = l - 1$
35: **end for**

Figure F.14. RHPS Algorithm.

APPENDIX G: USER'S GUIDE

The Turtlebot Koboki base is used during the experiments as shown in Figure G.1. The sensors are mounted to the robot via 3D printed parts.

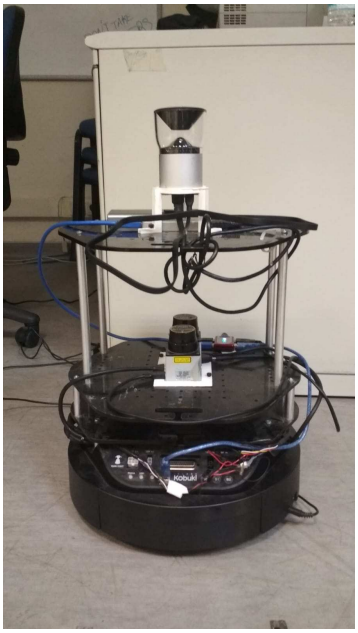


Figure G.1. Turtlebot Kobuki.

G.1. Hardware

The detailed specifications of Kobuki base is given in Table G.1 . The robot uses two external sensors: V360 omni-directional camera and two Hokuyo URG-04LX scanning range finders. The main processing unit is a Lenovo Yoga 11S Ultrabook with Intel Core i3 3229Y processor and 2 GB of RAM. The camera sensor has its own battery and connected to the processing unit via an external HDMI-USB converter which limits the maximum resolution of images to 142 x 640 pixels. The energy of LIDAR sensors is supplied by 5V external power output of Kobuki base. Both the sensors and the base are connected to the processing unit via USB interface.

Table G.1. Robot base specifications.

Specification	Value	Unit
Maximum translational velocity	0.7	m/s
Maximum rotational velocity	180	deg/s
Payload	4	kg
Threshold Climbing	12	mm
Rug Climbing	12	mm
Battery Voltage	14.8	V
Battery Capacity	2200	mAh
Expected Operating Time	3	hour
Expected Charging Time	1.5	hour
Sensor Data Rate	50	Hz

G.2. Software

G.2.1. Installed Software

The following software is recommended in order to operate the system:

- Ubuntu 14.04 LTS
- ROS Indigo
- Qt 4.8
- OpenCV 3.4.0

G.2.2. Running the Robot

The robot, processing unit and the camera sensor should be charged and LIDAR should be connected to 5V external power output of the Kobuki base. After making sure that all USB connections are plugged-in the processing unit, use the following commands to launch the robot:

- Open a new terminal and run the command:

  roscore

- Each a new terminal or tab run the command:

  roslaunch kobuki_node minimal.launch

  roslaunch hokuyo_2.launch

  rosrun laser_merger_isl laser_merger_isl_node

  rosrun camera2ros camera2ros_node

- In order to run the navigation package, open a new terminal or tab and use the following command:

  rosrun apes_base apes_base_node

- The target command can be given via:

  rostopic pub /apes_robot/baseTargets apes_base/baseMsg

  or by using 2D Nav Goal button after launching the RViz:

  rosrun rviz rviz

- In order to run the localization module use following the commands each in a new terminal or tab:

  rosrun localization_isl bubble_localization_node

  rosrun localization_isl scan_localization_node

  rosrun localization_isl localization_wrapper_node

- In order to run the TSC model use the following commands each in a new terminal or tab:

  rosrun create_bdst_isl create_bdst_isl_node

  roslaunch place_detection_isl test.launch

  rostopic pub /placeDetectionISL/nodecontrol std_msgs/Int16 "data: 1"

- In order to run the hierarchical mapping use the following commands each in a new terminal or tab:

  rosrun generate_hmap_isl generate_hmap_node