

DEVELOPMENT OF AN INDOOR REACTIVE NAVIGATION METHOD
USING SPHERICAL IMAGES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YİĞİT ŞAHİN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

DECEMBER 2021

Approval of the thesis:

**DEVELOPMENT OF AN INDOOR REACTIVE NAVIGATION METHOD
USING SPHERICAL IMAGES**

submitted by **YİĞİT ŞAHİN** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil KALIPÇILAR
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. M. A. Sahir Arıkan
Head of Department, **Mechanical Engineering**

Assoc. Prof. Dr. A. Buğra KOKU
Supervisor, **Mechanical Engineering, METU**

Examining Committee Members:

Assoc. Prof. Dr. Mehmet Bülent ÖZER
Mechanical Engineering, METU

Assoc. Prof. Dr. A. Buğra KOKU
Mechanical Engineering, METU

Assist. Prof. Dr. Ali Emre TURGUT
Mechanical Engineering, METU

Prof. Dr. Osman PARLAKTUNA
Electrical and Electronics Eng., Eskişehir Osmangazi University

Assist. Prof. Dr. Kutluk Bilge ARIKAN
Mechanical Engineering, TED University

Date:



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Yiğit ŞAHİN

Signature :

ABSTRACT

DEVELOPMENT OF AN INDOOR REACTIVE NAVIGATION METHOD USING SPHERICAL IMAGES

ŞAHİN, Yiğit

M.S., Department of Mechanical Engineering

Supervisor: Assoc. Prof. Dr. A. Buğra KOKU

December 2021, 86 pages

An algorithm for visual topological localization and navigation is proposed. Landmark detection and tracking is performed in successive frames to determine angular changes between repetitive and salient features. This angular state is used to generate heading vector which will lead the robot to the target scene. This reactive navigation behavior is aided with localization and path planning algorithms to complete the SLAM capability. Our algorithm is tested in 2 different natural indoor environments. Results are promising in the sense that the accuracy is enough for autonomous navigation purposes. Room to improve topics are discussed.

Keywords: topological navigation, qualitative navigation, spherical image, landmark detection, equirectangular scene matching

ÖZ

KÜRESEL GÖRÜNTÜLER KULLANARAK İÇ MEKAN REAKTİF SEYRÜSEFER YÖNTEMİNİN GELİŞTİRİLMESİ

ŞAHİN, Yiğit

Yüksek Lisans, Makina Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. A. Buğra KOKU

Aralık 2021 , 86 sayfa

Görsel topolojik lokalizasyon ve navigasyon için bir algoritma önerilmiştir. Tekrarlayan ve göze çarpan özellikler arasındaki açısal değişiklikleri belirlemek için ardışık görüntülerde yer işareti tespiti ve takibi gerçekleştirilmiştir. Bu açısal fark bilgisi, robotu hedef konuma götürecek yön vektörü oluşturmak için kullanılmıştır. Tepkisel seyrüsefer(navigasyon) davranışı, SLAM kabiliyetini tamamlamak için konum belirleme ve yol planlama algoritmaları ile desteklenmiştir. Algoritmamız 2 farklı doğal kapalı ortamda test edilmiştir. Sonuçlar, doğruluğun otonom seyrüsefer görevleri için yeterli olması anlamında umut vericidir. Son olarak geliştirmeye açık konular aktarılmıştır.

Anahtar Kelimeler: topolojik navigasyon, nitel navigasyon, küresel görüntüler, eşdörtgensel sahne eşleştirme



To my family

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my supervisor Assoc. Prof. Dr. A. Buğra Koku of Mechanical Engineering at Middle East Technical University. Thanks to him I was able to delve deep into computer vision and robotics studies which I enjoyed a lot. In addition to his positive and reasoning attitude, he helped me whenever i asked for it even though he was occupied by his own work.

Lastly, I must express my very profound gratitude to my parents and to my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xv
CHAPTERS	
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Motivation and Problem Statement	4
1.3 Proposed Methods and Models	5
1.4 Contributions and Novelties	6
1.5 The Outline of the Thesis	6
2 METHODOLOGY	7
2.1 Literature Survey	7
2.2 Problem Formulation	9
2.2.1 Reactive Navigation	10

2.2.1.1	Scene Matching	11
2.2.1.2	Heading Generation	14
2.2.2	Map Operations	16
2.2.2.1	Mapping & Exploration	16
2.2.2.2	Localization & Path Planning	17
3	IMPLEMENTATION	19
3.1	Mapping & Exploration	19
3.2	Localization & Path Planning	25
3.3	Reactive Navigation	29
4	EXPERIMENTS & RESULTS	31
4.1	Hardware Configuration	31
4.2	Mapping & Exploration	33
4.3	Navigation	35
4.4	Results	50
5	CONCLUSIONS & FUTURE WORK	59
5.1	Conclusions	59
5.2	Future Work	62
	REFERENCES	65
	APPENDICES	
	Appendix A. Custom Code	71
	Appendix B. Example Use of the Library	85

LIST OF TABLES

TABLES

Table 3.1	Node Addition Strategy	24
Table 4.1	Summary of the Navigation Experiments	48
Table 4.2	Summary of the Image Processing	49
Table 4.3	Average Image Processing Results of Environments	49

LIST OF FIGURES

FIGURES

Figure 2.1	Rectification at 0°(top), 90°(right), 180°(bottom), 270°(left) Pan Angles	12
Figure 2.2	Scene Rectification at 45° Tilt Angle with 90° FOV	13
Figure 2.3	Keypoint Thresholds = 0.005 (a), 0.1 (b), 0.5 (c)	14
Figure 2.4	Keypoint Projection and Unit Vectors	15
Figure 2.5	Example Connectivity Between Scenes	17
Figure 3.1	Flowchart of Mapping Process	20
Figure 3.2	Calculation of Exploration Direction	21
Figure 3.3	Obstacle Avoidance	22
Figure 3.4	Recall	23
Figure 3.5	Full Connectivity of the Living Room	24
Figure 3.6	Full Connectivity of the Laboratory	25
Figure 3.7	Panoramic camera used in capturing images and axes used	26
Figure 3.8	Equirectangular Panorama	26
Figure 3.9	Rectification Results of Figure 3.8	27
Figure 3.10	Navigation Flowchart	29

Figure 4.1	Tilt Angle with Height Difference	32
Figure 4.2	Living Room	33
Figure 4.3	Laboratory	33
Figure 4.4	Room Connectivity	35
Figure 4.5	Laboratory Connectivity	35
Figure 4.6	Initial (a) and Target (b) Scenes of the First Scenario	36
Figure 4.7	Scene Matching - 1	38
Figure 4.8	Scene Matching - 2	39
Figure 4.9	Scene Matching - 3	40
Figure 4.10	Scene Matching - 4	42
Figure 4.11	A path that connects two closest nodes	43
Figure 4.12	Representation of the Local Sights	44
Figure 4.13	Overlapped Local Sight and the Heading Vector.	44
Figure 4.14	Scenario 2	45
Figure 4.15	Scenario 3	45
Figure 4.16	Scenario 4	46
Figure 4.17	Laboratory Scenarios	47
Figure 4.18	Trapping During Random Wanderer Exploration	51
Figure 4.19	Indirect Map Connection	52
Figure 4.20	False Positive Matches	53
Figure 4.21	Final Scene Pair - 1	54
Figure 4.22	Final Scene Pair - 2	54

Figure 4.23	Final Scene Pair - 3	55
Figure 4.24	Final Scene Pair - 4	56
Figure 4.25	Final Scene Pair - 5	56
Figure 4.26	Final Scene Pair - 6	57
Figure 4.27	Final Scene Pair - 7	58
Figure 5.1	Node Cluttering	60
Figure 5.2	Obscured Line of Sight	61
Figure 5.3	Indirect Map Connection	62

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
API	Application Programming Interface
FOV	Field of View
GPS	Global Positioning System
ROI	Region Of Interest
SLAM	Simultaneous Localization and Mapping
OSC	Open Spherical Camera



CHAPTER 1

INTRODUCTION

This chapter is intended to outline the general scope of the thesis. Before going deeper into the details, it is essential to clarify what has been seen as a problem as well as what has been taken into consideration in solving it. Concisely, the problem that we are going to address falls within the SLAM territory. In its most compact form, it can be described as a qualitative navigation problem. Besides, the method we intend to apply to the problem is based on a spherical camera whose image outputs challenge existing image processing techniques. Therefore, we would like to develop a capable algorithm that can function in indoor spaces by processing equirectangular images.

1.1 Introduction

Depending on the operational environment mobile robotics can be split into 2 categories which are indoor and outdoor. While the latter enables external measurements like GPS to be used for localization, mapping, and navigation purposes, this is not valid for the former[1]. Although external modifications of the environment that are in the form of fiducial markers or artificial landmarks can help the agent in algorithmic decisions in both indoors and outdoors, such changes might not be feasible to apply in general. Besides, this also restricts the ability to operate in an unknown environment. Capability to execute tasks within foreign spaces is an important skill for autonomous agents. For the first time, Durrant-Whyte and Leonard had showed that a solution for this problem exist in their study [2]. In the 2000s, SLAM has attracted worldwide attention thanks to the STANLEY and JUNIOR cars. STANLEY came in first place at DARPA Grand Challenge in 2005 [3] meanwhile JUNIOR got second place at

DARPA Urban Challenge which is the third competition of DARPA Grand Challenge in 2007 [4]. Both cars were equipped with SLAM-systems by Sebastian Thrun. Since then, lots of studies have been published on the topic. Meanwhile, the idea of utilizing existing features as references is coined a term *marker-less SLAM* or *natural navigation*. It can be described as a navigation scheme that relies on the properties that are inherent to the environment rather than the aforementioned modifications as references [5]. During the early stages of SLAM, a variety of sensors such as sonars, laser range finders, GPS, cameras or multitudes of these have been used. The sensor suite of an application is determined by the approach that is to be implemented. For example, while an application can use laser range finders for mapping purposes, it can use GPS or camera for navigation purposes. Evidently, the sensor complexity of such applications is increased. However, recently a lot of work has been concentrated on applications where a camera is used as the only sensor. This is because low-cost consumer-grade cameras are now capable of capturing high-resolution and wide FOV images. Since the only input to the system is in the form of a visual, the vision-only approach is called *visual SLAM*, *vision-based SLAM*, or *vSLAM*. Furthermore, the complexity of the sensor configuration has been reduced drastically when compared to the approaches where a combination of sensors are used, thanks to the advances in computer vision and camera technologies [6]. On the other hand, because the fact that the tasks such as localization, mapping, path planning, etc. require the interpretation of incoming data in different forms and viewpoints, this simple camera configuration will require more complex algorithms.

During a SLAM task, one of the essential requirements that the agent has to fulfill is the fact that it needs to construct a representation of its surrounding in the form of a map to use later. Depending on the mission, task, or sensor configuration, this representation can take one of 3 forms: quantitative, qualitative, and hybrid representation [7]. Quantitative representation relies on expressing spatial information in the form of precise coordinate measurements. Provided that the required sensor suite is available, they are easy to construct. As the complexity and the size of the map increase, it becomes heavier computationally. However, for the tasks which require meticulous motor skills and precision such as surgical robots, pick & place robots, etc. accuracy cannot be compromised. Therefore, the computational overhead is inevitable nevertheless. On the other hand, qualitative representation aims to model the

neighborhood in the form of nodes and edges that connect the former which results in a graph representation or to express the spatial relations utilizing landmarks [8]. They are more lightweight than their quantitative counterparts and less demanding on the resources. In return, they contain less detail usually. Although containing less detail, qualitative representation is more intuitional and more suitable for behavior-based approaches[9]. For instance, the way other species that are capable of navigation and humans handle the representation is analogical to the qualitative approach. In another example, it had been stated in [10] and [11] that the angular information is preferred over the metric information in learning and localization processes of places by humans. For instance, while people are exchanging information for navigation purposes, they describe the directions concerning the distinct landmarks along the route. Lastly, the hybrid representation as the name suggests tries to take advantage of the best of the two worlds. That is, local metric maps are treated as nodes and linked together when they have a direct path in between [12]. Regarding when to choose which, the deciding factor could be derived from the focal point of the task at hand. Unlike the precision tasks which were mentioned previously, assignments that focus on locomotion can compromise the accuracy to a certain extend. For such circumstances, any amount of accuracy that will allow the agent to clear obstacles and avoid collision is enough. In particular to the thesis, the development of a reactive navigation algorithm that is based on spherical images is of main concentration. Besides, equirectangular images are the only data source to extract the information. We suggest that even if our setup lacks metric information, what the spherical camera provides us is enough to represent the environment with qualitative relationships. To sum up, it is possible to classify SLAM problem in different viewpoints. In terms of map representation, it can be divided as quantitative, qualitative, or hybrid. In terms of sensory input, it can be divided as vision-based, sonar-based, lidar-based, etc. In terms of interaction with the environment, it can be artificial or natural. Lastly, it can be classified as reactive or deliberate in terms of reasoning. The former is analogous to the reflexive behavior. In other words, agent's actions are the results of its instantaneous perception rather than its internal state which is the case for the latter. Deliberate approaches are more analogous to the systematic thinking. They tend to predict the results of agent's actions and conduct a search within a space of pre-defined behaviors. The thesis can constitute an example for 4 categories simultaneously. Namely, it is a

study of qualitative vision-based natural reactive navigation.

1.2 Motivation and Problem Statement

Vision-based SLAM is composed of 2 main components. The first is visual odometry. It is the process of interpreting the environment through features, landmarks, etc. to gather the necessary information such as camera pose, position, orientation. The second is the global map operations [6] which cover the tasks like localization and path planning. Depending on how this interpretation is implemented, visual odometry can be either feature-based or direct. Whereas, feature-based methods process images to extract local features and use them in-between frames, direct methods process the image as a whole through global features. These can be exemplified by color histograms, principal components, etc [7]. So far, regardless of the approach, the inferences that are to be made were constrained by the narrow FOV angle of the conventional perspective cameras. To compensate for the algorithmic ambiguities or uncertainties that arose as a consequence, additional cameras are placed to increase the FOV (stereovision) or even to complete the entire panorama [13], [14], [15]. Likewise, a laser range finder is used to support the visual measurements with distance information [16], [17], [18]. However, the introduction of spherical cameras has enabled the largest FOV so far. Through panoramic images, we suggest that

- the additional sensor requirement can be overcome.
- the reactive navigation can be accomplished.
- a topological map can be generated.

Having a single sensor helps keep the hardware set as simple as possible. Meanwhile, reactive navigation is more intuitional than deliberate navigation. On the other hand, the reactive behavior is constrained by momentary perception as stated before. To address this restriction for larger-scale tasks that extend beyond reactive behavior, an ability to devise a plan is crucial. This is the main reasoning behind the third item above. Additionally, since the reactive behavior does not maintain an internal state, anything that can cause a change in momentary perception such as moving keypoints

directly affects the outcome. This constraint is left for future work. Within the thesis, it is assumed that the keypoints are stationary and the only source of motion is the camera itself.

1.3 Proposed Methods and Models

We partition the development of the algorithm into mapping & exploration, localization & path planning and the reactive navigation. For the mapping & exploration phase, topological representation is considered to be the most suitable. The first reason is being the fact that as long as the navigation task is considered without the additional capabilities such as obstacle avoidance, metric precision is not an obligation. In nature, few to none of the animals, to our knowledge, relies on metric information for navigation purposes. Likewise, humans do not have a natural way of measuring distance. Nonetheless, neither of the examples that are given are incapable of navigation. Furthermore, topological representation is also beneficial in terms of resources [8]. When the map is updated with new nodes which are in the form of equirectangular images, keypoint descriptors of the node image are saved as well for further matching operations. Here the key assumption in updating the map is the fact that each node must not be farther from each other than the reactive navigation can cover.

Reactive navigation is a process of traversing in-between two locations without the help of localization and use of any information other than current sensory information and target descriptors. This method relies on the fact that relative angular separation between keypoint pairs can be tracked and that these angular differences between successive frames can be used to generate the heading which expresses the change that has been observed. At a given moment, if the agent detects numerous keypoints that are common to the given target, it is possible to compute a series of movements that terminates at the destination with a significant amount of topological accuracy. If a reactive navigation scheme is not applicable then localization and path planning sequences are initiated successively to decide the topological state of the agent within the environment and generate a series of intermediate destinations which are called "*via points*" that are available through the reactive navigation.

1.4 Contributions and Novelties

Our contributions are as follows:

- A simple yet versatile indoor visual SLAM algorithm.
- Novel method for visual odometry [19].
- Equirectangular images are used in indoor navigation problem contrary to omnidirectional (catadioptric) images.

1.5 The Outline of the Thesis

The rest of the thesis is organized as follows: Literature survey, problem formulation, and algorithmic details are covered in Chapter 2. Implementation details of the algorithm are given in Chapter 3. Chapter 4 delves into the experiments and the results. Lastly, Chapter 5 is the final chapter where the conclusions are drawn and room for improvements for the proposed algorithm is presented to the reader.

To sum up, a brief classification of the SLAM framework is introduced hoping that it boosts the interest and familiarity of the reader to the topic. Different viewpoints on the topic are put forward to better understand the extent of the subject. The points that have room for improvement addressed together with a proper solution approach. Lastly, the chapter is concluded by revealing the contribution and outlining the subject so that the reader becomes accustomed to the structure of the thesis.

CHAPTER 2

METHODOLOGY

Before moving on to the details, it is essential to introduce the terminology and their context to make the problem easier to follow. The first one is the *reactive behavior*. This means the ability to behave or act just by taking the momentary information that is provided by the spherical camera into consideration without the need for any memory or past information contrary to the deliberate behavior. Since the action that will be executed is planned according to the momentary measurements, it is limited to a sensory range. This definition brings us to the second concept which is the term *local*. Within the context of the thesis, *local* is used to refer to the extent of the sensory range at a given moment. Unfortunately, most of the time, an agent is required to operate in a larger environment that is far beyond its sensory range. This necessitates the functionality of the reactive behavior outside the local region. This is managed by an algorithm that can partition the task into smaller chunks which can be completed with a chain of reactive actions with the help of a topological map and path planning. We can now move forward.

2.1 Literature Survey

Spherical cameras, fisheye lenses, and such wide FOV devices draw a significant amount of attention due to their high capability-to-cost ratio in the last decade. However, previously developed image processing techniques to extract features and descriptors, match images, etc. are not sufficient in dealing with images captured by such devices because of the technology and algorithms that they employ. Specifically, in the equirectangular projection, the image undergoes a distortion mostly in

the radial direction. A well-known example can be given as the fact in the world map with equirectangular projection Greenland is seen as bigger than Australia. The reason is that Australia is located closer to the equator than Greenland. Hence Greenland undergoes a heavier distortion. To begin with, [20] compares the state-of-the-art methods such as SIFT, SURF, and ORB in terms of performance. What they conclude is the fact that there is a trade-off between speed and accuracy. On top of SIFT which is the most famous feature detector for the time being, numerous studies tried to focus on the different shortening of the algorithm. ASIFT is one of them which was proposed by Fu et al. [21]. On top of SIFT itself, they proposed to simulate two remaining parameters of affine transform which were camera axis orientations that are latitude and longitude. With the addition of these 2 parameters, the proposed algorithm was fully affine invariant. However, it performed slower due to increased complexity and poorer under different lighting conditions. To deal with the issue [22] altered the ASIFT algorithm by means of reducing computation time and increased robustness and accuracy. Another solution is proposed by [23]. They base their solution on the fact that even if the spherical images are distorted during projection, the equatorial line in the resultant equirectangular panorama has the least distortion. They tried to detect the features within the masked regions which were the least distorted areas. J.Cruz-Mota et al. [24] aim to solve the same problem by computing the scale-space in the spherical domain as opposed to what Lowe [25] proposed. G. Tong et al.[26] thought of using distortion information to define a curve which is the result of a projection of a line in space. However, it is seen that this method tends to perform poorer than even conventional methods in the case of small distortion because they focused heavily on large distortion. Meanwhile, sRD-SIFT developed by M.Lourenco et al.[27] also intends to solve the same problem by using adaptive convolution that they have derived to compensate for the radial distortion but it increases the computation time a lot. The solution that they come up with to prevent this is a filter approximation which enables a high detection rate and low computation load. Most recently, Sarlin et al. [28] trained a neural network that excels on feature matching. Although it can work in combination with a variety of keypoint detectors, it is shown that it synergizes best with the keypoint detector trained by Detone et al. in [29]. Besides image processing, another building block of any visual navigation algorithm is the way they use to represent spatial information. Kuipers and Byun [30]

had proposed the pioneering work in the topological representation. Unlike similar studies, they derive the representation by recognizing and exploiting the qualitative properties rather than extracting the topological relations after the refinement of geometrical precise measurements. Meng and Kak used a neural network to extract features and matching purposes meanwhile opting for topological representation as a map in NEURO-NAV system [31]. As an extension to NEURO-NAV, FUZZY-NAV [32] is studied by Pan. In the work, fuzzy logic is incorporated in a high-level rule-based controller to control the navigation behavior. Another example of vision-based topological navigation is investigated by Hashima in [33] where correlations have been used to track landmarks across images that are captured along with the robot movement. While numerous solutions are proposed in the field of image processing in terms of feature extraction and descriptor matching in equirectangular images and qualitative navigation, combining these in topological SLAM applications with visual servoing is not studied as rigorously as metric navigation. One of the few examples belongs to Maohai et al. [5]. In their work, while they use an omnidirectional camera to map the environment which is to be used afterward for localization and path planning purposes they employ a path tracking algorithm. Due to the fact that the ASIFT is used as the main feature detector, to compensate for the computational load, GPU is utilized. A similar study is conducted by Goedeme et al. [34]. In their work, instead of using one of the existing feature detectors they propose a new method which they called *fast baseline matching* that they claim to be fast enough to be used on the go. However, cameras that are used in both of the studies are single-lens omnidirectional cameras. These use fisheye lenses to manipulate the angle of the shot to fit the larger radius of the scene. When used alone, they are not able to capture an entire 360 degrees sphere. To our knowledge, there are not any other studies that utilize dual fisheye configuration. In other words, 360 degrees sphere images are not widely used in vSLAM applications and are yet to be studied.

2.2 Problem Formulation

In terms of algorithmic components, our method is based on 2 fundamental elements which are the reactive navigation and the map operation modules. The former refers

to the ability by following the heading vector generated by comparing two scenes. Whereas the latter refers to the ability to locate itself within the database which has been generated apriori during a mapping phase and planning a path that terminates at the desired target location.

2.2.1 Reactive Navigation

Navigation is defined as an act moving from one place to another in a controlled manner in [35]. This implies that anything that moves purposefully has some sort of navigation functionality. Depending on the requirements of the task at hand, specifications of the navigation can vary. In the case of industrial pick & place robots, surgical assistance agents, missiles, a focus is concentrated on precision. On the other hand, in nature or daily life, neither animals nor humans do not depend on metric information most of the time. Even then, they are able to navigate within the neighborhood to a certain extent. To understand how it is done, a reactive navigation scheme has to be clarified.

The idea of reactive navigation relies on the fact that it is not necessarily needed to have precise information on the whereabouts of the target location with respect to a reference frame in metric sense, to be able to navigate. None of the 5 fundamental senses provides us with precise information. Rather, they allow us to form intuitive relations like resemblance, reminiscence, perception between scenes, and elements forming them. As long as one or more of these relations can be formed between the current state and destination, a navigation action can be realized with remarkable topological accuracy. Often times this navigation act is in the form of reacting to these relations. This is where the name comes from. Expectedly, what we look for during the reactive navigation phase are similar features that can be tracked over successive frames to navigate in between. The process of the reactive navigation depends on the success of 2 different components. First, initial and terminal scenes have to undergo a matching process that will detect common keypoints. Second and lastly, these keypoints have to be processed to generate a heading that leads to the terminal scene.

2.2.1.1 Scene Matching

Scene matching can be divided into 2 subsequent tasks first of which is the feature detection. A *feature* refers to a small image patch that contains a *keypoint* and a *descriptor*. Evidently, feature detection is defined as the process of keypoint detection and descriptor generation. *Keypoints* are characteristic points in an image that are searched for with different methods. Harris corner detector [36] and laplacian of gaussian blob detector [37] are 2 major examples among others. Finally, *descriptors* are the expressions, mostly in the form of a vector, that enable operations like comparison between features. The last step of scene matching is the descriptor matching. Brute force matcher which is one of the famous matchers finds correspondent descriptors by computing L2-distance in-between for each of the candidate matches. The candidate with the minimum distance to the query descriptor is matched with it.

Scene matching is essential in determining if the reactive navigation is available between current and desired locations. The reason for such significance is the fact that the heading vector which will direct the robot towards the target will be computed according to the matching result. Since we are working with a dual fisheye configuration, the images we get from the camera are in the form of equirectangular panorama and suffer from distortion as a result of planar projection. This affects the saliency and distinctiveness of the keypoints. To prevent such performance degradation we suggest converting these equirectangular images to perspective images such that the entire panorama can be seen as a collection of perspective images with minimal distortion. Rectification process which has been obtained at 45° of tilt angle and 90° FOV is illustrated in Figure 2.1.

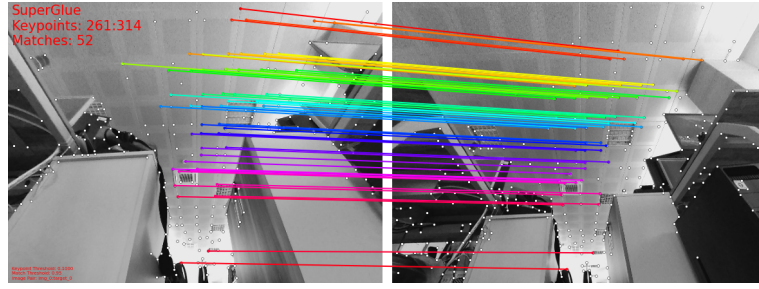
Rectification of images is followed by the scene matching process. For this task we employed the SuperGlue network since it shows superior performance in viewpoint changes when compared to the other matchers [28]. In addition, it synergizes best with SuperPoint feature detector [29] although it is compatible with other detectors like SIFT. Due to the structure of the indoor environments clutter decreases from the floor towards the ceiling. This implies that the distribution of the features is not homogeneous. The ceiling and floor can differ in terms of texture, geometry, symmetry, etc. This may obscure the detection of keypoints hence reduce the robustness of the



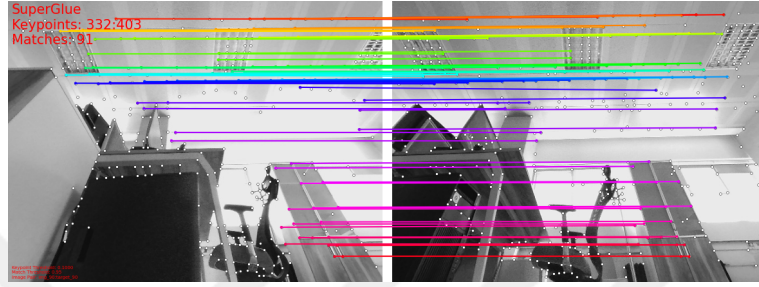
Figure 2.1: Rectification at 0° (top), 90° (right), 180° (bottom), 270° (left) Pan Angles

algorithm. Focusing solely on the ceiling or floor can be another factor that reduces the robustness as well because of the nonhomogeneous distribution of features. This is why we propose to render the images not at 0° tilt angle but at 45° tilt angle. By doing this, we aim to capture as much features as possible. Matching results of rendered images with the explained approach are illustrated in Figure 2.2.

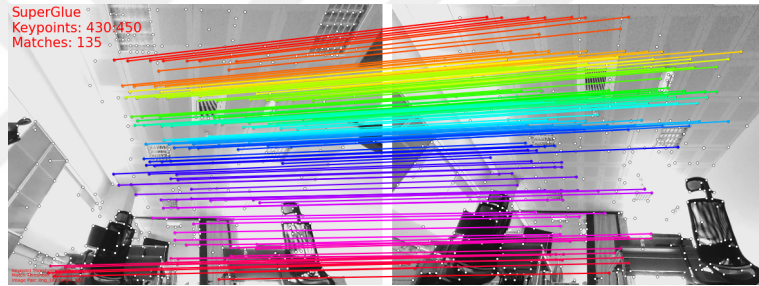
Confidence scores are used for algorithmic decisions throughout the thesis. Every detected keypoint is assigned a confidence score which is a measure of its saliency. This is also used as a pass/fail test for candidate keypoints. This means if a detected candidate has lower confidence than the threshold value, then it is rejected. Therefore, setting the threshold strongly depends on the environment. Figure 2.3 shows how increasing keypoint detection threshold affects the number of detected keypoints. As the first trials showed that the default parameters of the SuperGlue network give promising results, they are left as they are. Out of these parameters, keypoint detection threshold is left as 0.05. Although optimization and adaptation of design parameters play a crucial role in the success of our algorithm, it is out of the scope of the study and is left for future work. Similar to the keypoint detection, every matching keypoint pairs are also assigned a matching score which shows the confidence of the matching operation.



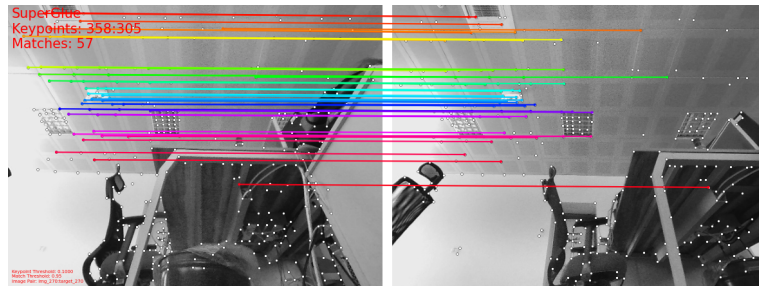
(a) Rectification at 0° Pan Angle



(b) Rectification at 90° Pan Angle

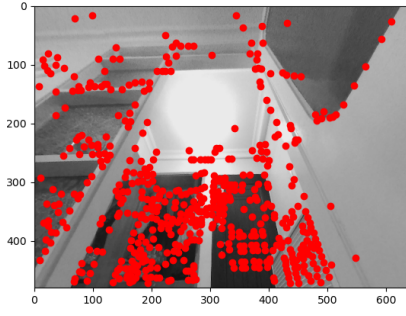


(c) Rectification at 180° Pan Angle

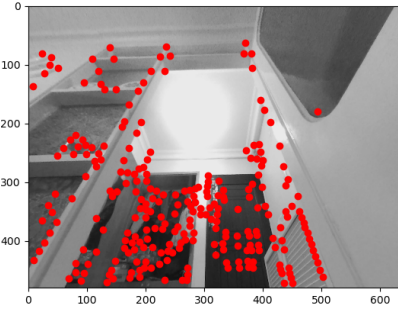


(d) Rectification at 270° Pan Angle

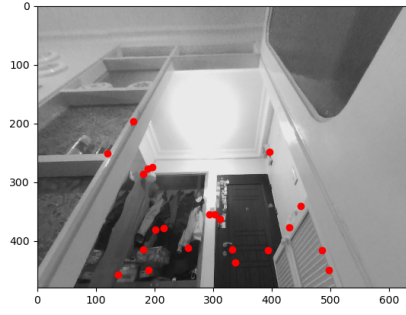
Figure 2.2: Scene Rectification at 45° Tilt Angle with 90° FOV



(a) Number of Detected Keypoints = 521



(b) Number of Detected Keypoints = 256



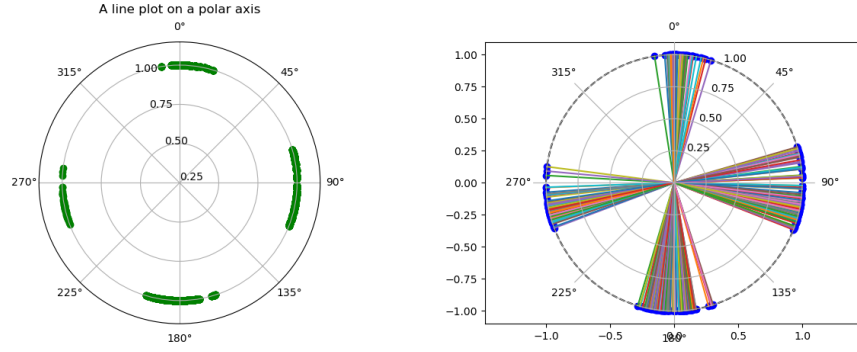
(c) Number of Detected Keypoints = 22

Figure 2.3: Keypoint Thresholds = 0.005 (a), 0.1 (b), 0.5 (c)

2.2.1.2 Heading Generation

Once the algorithm confirms the reactive navigation availability, it takes an action by generating a heading vector. This is done in 3 steps. First, all of the keypoints are projected onto a unit circle as in figure 2.4a. Second, each keypoint is assigned a unit vector as shown in figure 2.4b which will be the input to the vector analysis which is the final step that generates the heading.

Projected keypoints are paired up so that the angle between them can be computed through the drawn unit vectors. In the equations below, $\underline{u}^{c_i}$ and $\underline{u}^{c_j}$ are the first and second unit vectors of a selected pair in the current scene (i.e. superscript c) respectively. Likewise, $\underline{u}^{t_i}$ and $\underline{u}^{t_j}$ are the first and second vectors of the same pair which are tracked in the second scene (i.e. superscript t). The vector algebra which is employed



(a) Keypoint Projections on a Unit Circle

(b) Keypoints and Unit Vectors

Figure 2.4: Keypoint Projection and Unit Vectors

for the process is explained in [19] and given in equations 2.1 to 2.5 as follows:

$$\begin{aligned} \mathbf{d}^{c_{ij}} &= \mathbf{u}^{c_i} \cdot \mathbf{u}^{c_j} & \mathbf{C}_{ij} &= \mathbf{u}^{c_i} \times \mathbf{u}^{c_j} \\ \mathbf{d}^{t_{ij}} &= \mathbf{u}^{t_i} \cdot \mathbf{u}^{t_j} & \mathbf{T}_{ij} &= \mathbf{u}^{t_i} \times \mathbf{u}^{t_j} \end{aligned} \quad \text{where } i \neq j \quad (2.1)$$

$$\mathbf{A}_{ij} = \text{sgn}(\mathbf{d}^{c_{ij}} - \mathbf{d}^{t_{ij}}) \quad (2.2)$$

$$\mathbf{B}_{ij} = [\text{sgn}(\mathbf{C}_{ij} \cdot \mathbf{T}_{ij}) + 1]/2 \quad (2.3)$$

$$\mathbf{u}_{ij} = (1 + \mathbf{B}_{ij}(\mathbf{A}_{ij} - 1)) / (\mathbf{u}^{c_i} + \mathbf{u}^{c_j} / \|\mathbf{u}^{c_i} + \mathbf{u}^{c_j}\|) \quad (2.4)$$

$$\mathbf{h} = \sum \mathbf{u}_{ij} \quad (2.5)$$

Where $\mathbf{A}_{ij}$ indicates that the angle between any keypoint pair is increasing or not between current and target scenes. $\mathbf{B}_{ij}$ represents the order of the pair. Finally $\mathbf{u}_{ij}$ is the bisector along which the motion is headed of the pair. The advantage of using bisectors is that it gives enough information concerning whether it is needed to increase

the angle between the given keypoint pair or not so that the current angular state coincides with the target state. Additionally, implementation ease is another factor that encouraged the specified use.

2.2.2 Map Operations

Map operations refer to the set of activities that enable the robot to operate outside its sensing range and extend the reactive navigation beyond the local region. This will be detailed under 2 main topics as follows: Mapping & Exploration, Localization & Path Planning.

2.2.2.1 Mapping & Exploration

The mapping process plays a crucial role in dividing the navigation problem which extends to a point beyond the local sensing range. Since the reactive navigation is constrained by the sensory range, actions that are outside of this range will be planned with the help of the generated map. During the mapping phase, the agent adds nodes to the map depending on the similarity of the captured scenes. A new node is added to the map when the similarity between the newly captured image and the previous one or any of the other nodes drops below a certain threshold. Such threshold ensures that the connected 2 nodes in the graph have a certain amount of similarity such that the reactive navigation can be facilitated. Each node does not have to have just a single connection to other nodes. As long as the similarity threshold is satisfied, a node can be connected to many more with the corresponding edges. At the end of the mapping phase, we have a raw topological map of the environment whose nodes can be updated with the changing environmental conditions. Moreover, the selection of topological representation rather than metric one makes it possible to accommodate such a dynamic structured map. Figure 2.5 shows how a generated map looks like when the nodes are represented with their associated equirectangular scenes.



Figure 2.5: Example Connectivity Between Scenes

2.2.2.2 Localization & Path Planning

Most of the time a navigation problem reaches beyond the local region irrespective of the sensor suite. This also implies that if the agent is required to accomplish such a task then there must be a feasible solution. Based on this, we suggest that it is possible to convert the global task to a set of local missions and the chain of successive reactive navigation applications can function properly. The aforementioned conversion takes place with the help of the generated map and implicitly requires that each local mission can be accomplished with reactive navigation. First, the closest nodes to the current and target scenes are located. This is the localization process. Then it is followed by path planning which is a graph search problem whose main function is to find a path in between the found nodes. After the path is generated, nodes of the path are named as "*via points*". *Via points* act as an intermediate destination for the reactive navigation module.

Closing on the chapter, literature had been revisited to enhance the familiarity of the reader about the general state of the mobile robotics and image processing fields. It has been followed by the definition of the problem and how we tried to formulate and

express it for the sake of clarity. The next chapter will address the problem from the viewpoint of coding and implementation.



CHAPTER 3

IMPLEMENTATION

The way the algorithm is implemented is essential in understanding what it does and how it does. In addition, to increase the clarity as much as possible and make it more understandable for the reader, explaining the implementation details and algorithmic decisions as extensively as possible is of utmost importance. This chapter serves this purpose. Particular details on the topic will be handled in algorithmic order. In summary, implementation of the algorithm has been completed in 3 steps. These are *Mapping & Exploration*, *Localization & Path Planning* and *Reactive Navigation*. Representation of the environment and the discovery strategy is expressed in the first section. Map operations that are based on the generated map such as localization and path planning will be the second section as the name suggests. Lastly, implementation details of reactive navigation scheme will conclude the chapter.

3.1 Mapping & Exploration

Similar to almost every other SLAM application, this one starts with a kidnapped robot as well. When the agent wakes up in an unknown environment, the mapping phase is the first algorithmic component that runs. The significance of the mapping algorithm stems from the fact that the agent learns the environment by this method and all of the future operations and tasks will be relying on the resulting map.

When the mapping is started, an image is captured at the initial position and added directly to the map as a new node without any further assessment. Aside from the initial node, all other node additions will have to satisfy a certain similarity condition. Namely, each candidate scene must be distinctive enough to qualify for a new node

but at the same time it should be similar enough to facilitate the reactive navigation. These two opposing conditions that are symbolized with S_{th} and S_{RN} respectively, implicitly specify an interval which is shown in equation 3.1 meanwhile the similarity between a candidate node (CN) and a closest map node is symbolized as η_{CN}^{CMN} . Flowchart of the exploration algorithm is presented in Figure 3.1 as well.

$$S_{RN} \leq \eta_{CN}^{CMN} \leq S_{th} \quad (3.1)$$

where

$$\begin{aligned} \eta_{CN}^{CMN} &= \text{Similarity response of candidate node to closest map node} \\ S_{RN} &= 1.25(\text{Reactive Navigation Threshold}) \\ S_{th} &=: 1.65(\text{Similarity Threshold}) \end{aligned}$$

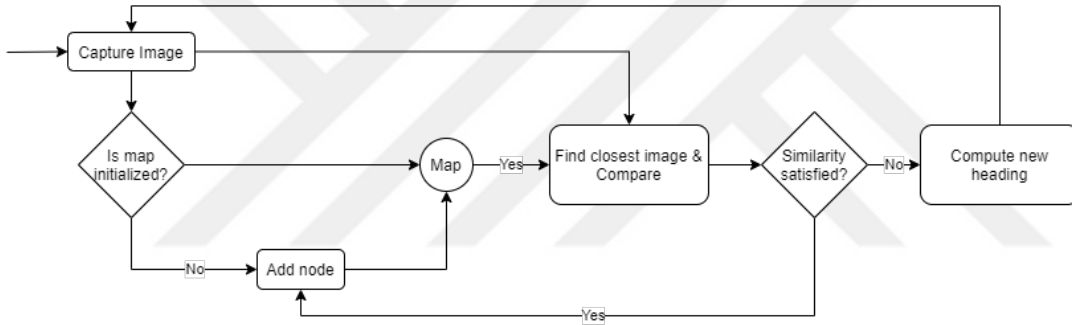


Figure 3.1: Flowchart of Mapping Process

S_{th} will impact the total number of nodes in the map. As the number decreases, the distance between the nodes increases which reduces the resemblance. In this case the reactive navigation can fail due to the lack of common features in-between successive scenes. On the other hand, as the number of nodes increases, the size of the map increases as well as the amount of resources needed. In our experiments $S_{th} = 1.65$ and $S_{RN} = 1.25$ has been observed to work reasonably well in the home environment. As the threshold value becomes greater similarity between nodes increases. Although reactive navigation benefits from this in terms of accuracy, unnecessary nodes are added to the map that requires more resources to manage. Conversely, if the similarity threshold is decreased the number of nodes in the map is decreased in return. This has the opposite impact on the map. That is, reactive navigation starts to suffer from

not being able to connect two map nodes, despite the map becoming lighter resource-wise.

We let random heading vectors to guide the agent to discover until the map expands enough such that it has enough nodes to interpret so that the next mapping strategy can be applied. The strategy is the guidance of exploration towards undiscovered areas. This is a 2 step process. First, individual heading vectors are computed from the agent's location to the 2 map nodes with the highest similarity. Finally, these individual heading vectors are summed and reversed to guide the robot in opposite direction. Summing over 2 heading vectors requires at least 3 previously added nodes. To clarify, the map is deemed to be expanded enough when the number of nodes reaches 3. This process is visualized in Figure 3.2.

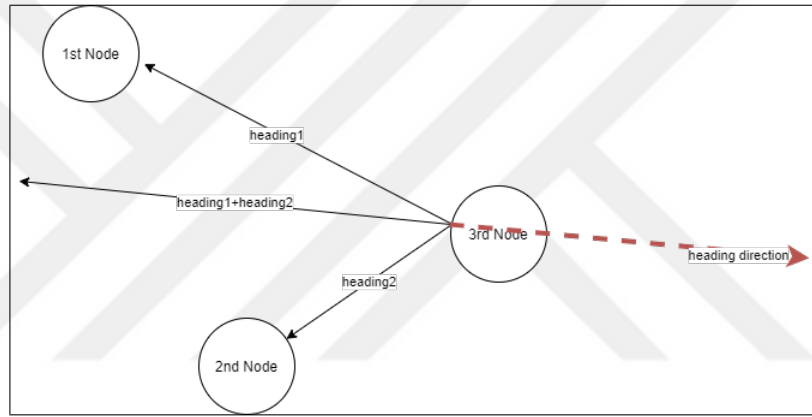


Figure 3.2: Calculation of Exploration Direction

During the experiments, it has been observed that this approach tends to fail when the agent cannot compute similarity scores for multiple nodes. This is likely to occur when the agent gets too close to the boundaries or in tunnel-like spaces where the field of view is highly obscured. To circumvent this tendency, a concept of *admissible heading* is introduced. Since we know in which direction we have headed to get to the current location, a reasonable action to take would be moving further to keep exploring. That is why *admissible heading* is defined as the subset of possible heading vectors which will lead the agent towards unexplored regions. In our case, the admissible heading set would be the hemisphere in front of us. Hence, when the above exploration scheme fails to compute a heading vector, a new heading is generated by picking a random vector from the admissible heading set. The exploration

stage relies heavily on the heuristics that are used to determine how the process will advance. Since the map representation is not unique for a given environment, using a different heuristics will likely to result in different connectivity graphs.

Another implementation detail regarding the mapping phase is the obstacle avoidance behavior. Since this issue is not addressed within the scope of the thesis it is simply implemented by an external obstacle signal given by the operator. Within the flow of the algorithm, a user input is requested upon computation of the heading to check if the action is taken or not. If the agent comes close to an obstacle as a result, an operator input is provided so that the agent changes its course. Under such circumstances, to make sure that the agent clears the obstacle the new heading vector is generated in the reverse direction. However, this reversal is not desired as it is since it directs the agents to an already mapped area. As a workaround, *exploration noise*, $\delta \in [-15, 15]^\circ$ is added to this reversed heading vector so that the resulting vector does not take the agent back to where it came from. Visualization of obstacle avoidance behavior is given in Figure 3.2.

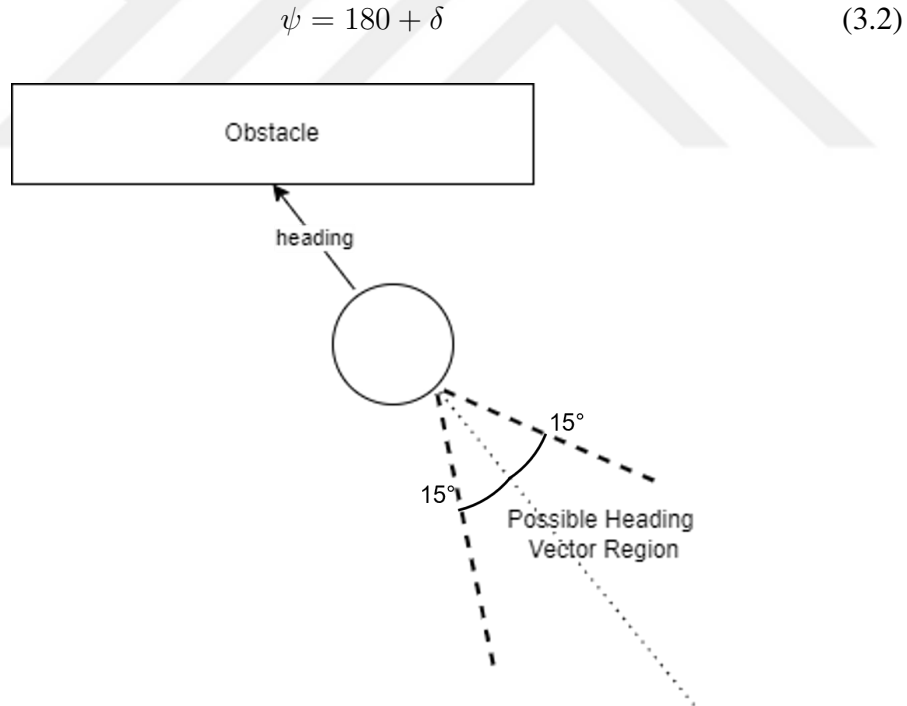


Figure 3.3: Obstacle Avoidance

Final feature concerning the mapping is the ability to recall. This refers to the ability to detect previously visited locations. These can be in the form of reminiscent places

that the exploration algorithm thinks that it has been there before or loops that are inherent features of indoors that enable people to walk around and that connect different places. In our algorithm recall has been achieved by combining 2 different attributes. First of which is the closest node detection. That is, since the agent does not follow a straight path majority of the time, the closest node to the candidate location is not the latest node in the map necessarily. Depending on the geometry of the setting it can be any of the nodes in the map. That is why the primary action to take is to find the closest node. Second and last action has been explained as the similarity inequality in equation 3.1 which prevents the unnecessary addition of nodes. If a candidate responds higher than S_{th} to the closest node it is interpreted as a recall. A simple recall action is illustrated in Figure 3.4 .

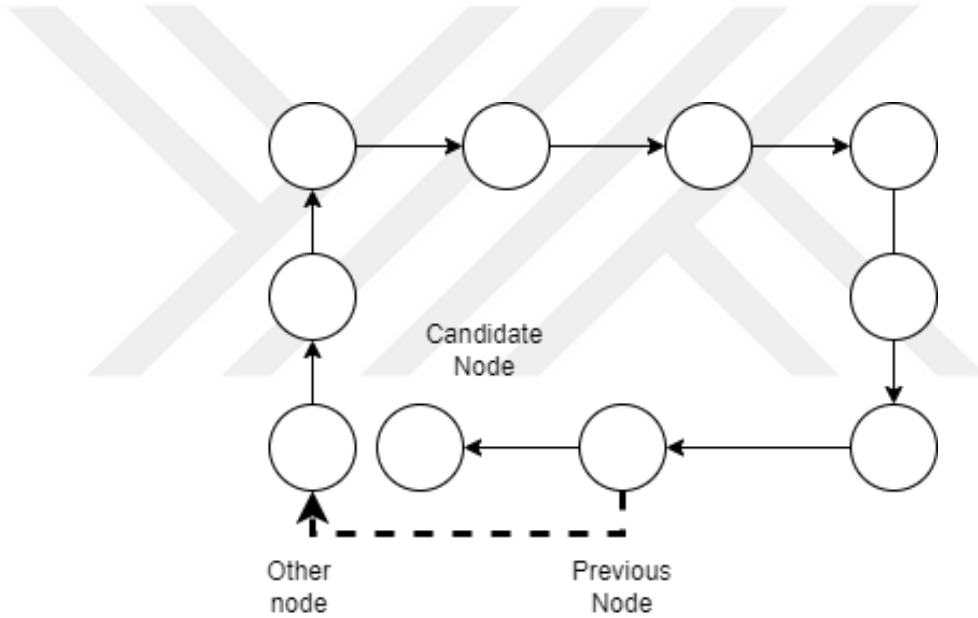


Figure 3.4: Recall

If the response of the *candidate node* (CN) to the *other node* (ON), $\eta_{CN}^{ON} \geq S_{th}$ and to *previous node* (PN), $\eta_{CN}^{PN} < S_{th}$ then *candidate node* is not added to the map and the correct connection would be the one that is shown in Figure 3.4 with dashed arrow. Node addition and recall detection decisions are made according to the similarity scores between the candidate and the nodes of the map that is generated by then. Tabulated decision-making process is shown in Table 3.1.

		$\eta_{\text{candidate node}}^{\text{previous node}}$		
		$1.25 \leq$	$1.25 < \eta \leq 1.65$	$1.65 <$
$\eta_{\text{candidate node}}^{\text{other node}}$	$1.25 \leq$	Back Trace	New Node	Keep Going
	$1.25 < \eta \leq 1.65$	Back Trace	New Node	New Node
	$1.65 <$	Back Trace	New Node Link	New Node Merge

Table 3.1: Node Addition Strategy

Back Trace in Table 3.1 means the act of moving back to the previous node. *New Node* refers to the addition of a new node to the map as the name suggests. *Link* means checking *previous node* and the *other node* for a possible link if they do not have an established connection yet. Lastly *merge* indicates that the current state is the same as *previous node* and *other node* topologically hence they must be checked for merge. This table is revisited in a loop until the all nodes of the map is tested for η_{CN}^{ON} . Introduction of such a decision table allows us to find inner connectivity within the map. The node addition strategy in Table 3.1 is applied in 2 natural indoor environments which are a living room and a laboratory and the connectivity graphs are generated as shown in Figures 3.5 and 3.6 respectively. The figures are not representing the related environment geometrically because of the topology and visualization library used in coding.

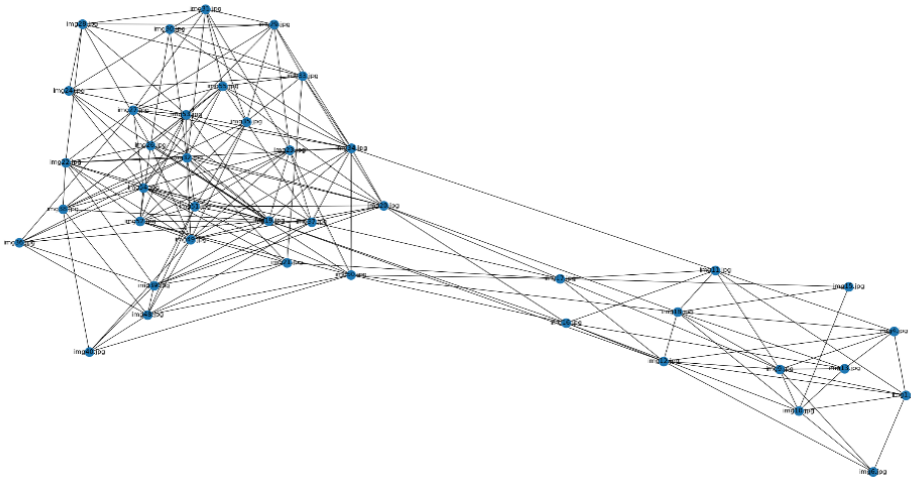


Figure 3.5: Full Connectivity of the Living Room

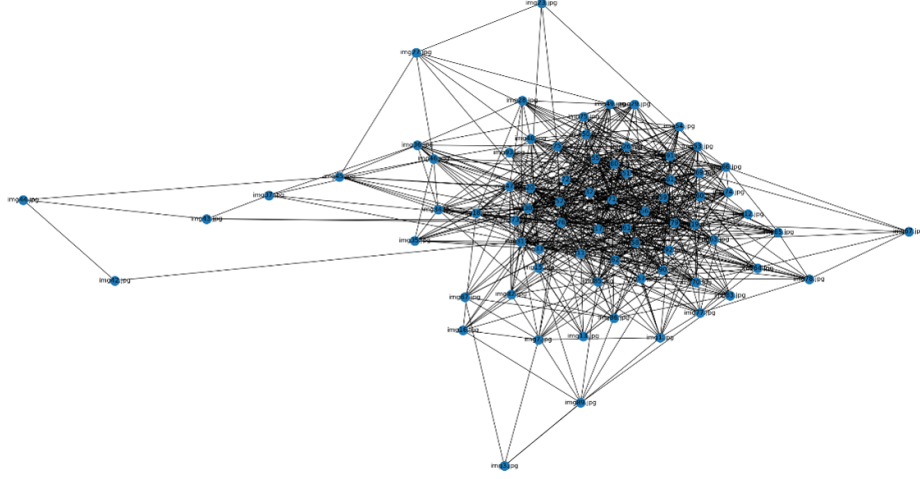


Figure 3.6: Full Connectivity of the Laboratory

Once the mapping is completed, our map consists of 2 major components. One is the *connectivity graph* which is stored as python dictionary whose keys are the nodes of the map. Furthermore, each key has a list which contains its connected nodes. The another is the *node list*. Each element of *node list* is a compact information package which holds the image processing results of associated scene.

3.2 Localization & Path Planning

We define the localization as the ability to locate itself relative to a reference of some sort by using information contained in the local region. In our case, this instantaneous information is in the form of equirectangular images. Since our sensor suite contains only a spherical camera, the map that has been generated also has equirectangular images as its nodes. Thanks to the simplicity of the sensor setup, localization becomes a process of finding the most similar image on the map. This is a 3 step process in our algorithm. To begin with, an image is rectified with respect to the axis system shown in Figure 3.7.

The rectification took place according to the predefined camera parameters which are *pan* and *tilt* angles. *Pan* refers to the rotation of the camera about Z-axis (i.e vertical axis) and is set as 0, 90, 180, 270 degrees and the *tilt* refers to the rotation about



Figure 3.7: Panoramic camera used in capturing images and axes used

Y-axis (i.e horizontal axis) and is set at 45 degrees. Rectification process has been completed with a 90° FOV and is illustrated in the Figures 3.8 and 3.9.



Figure 3.8: Equirectangular Panorama

By the way, it should be noted that these rotations are not physical rotations. That is, the camera has not been rotated around the axes itself but images have been. This is one of the strengths of spherical imagery. Since we have the entire spherical field of view captured, such rotations can be applied artificially without resorting to physical action. Such selection of pan angles helps in the process of recovering peripheral information, meanwhile more reasoning can be done on the selection of the tilt angle. Indoor spaces are enclosed by the the walls, the floor, and the ceiling. This influences the distribution of features within. For example, looking directly at the ceiling will not provide additional information since it is one of the regions that lacks texture.

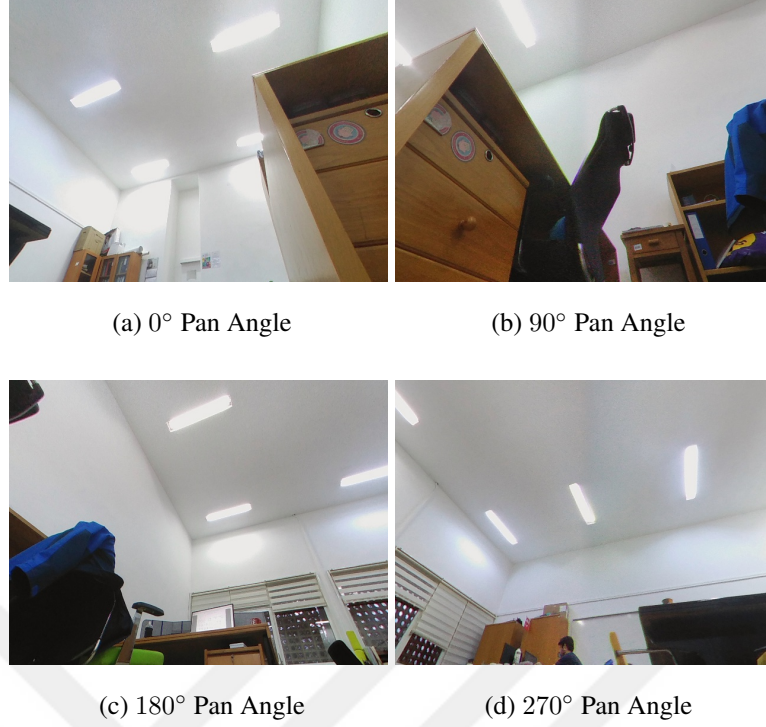


Figure 3.9: Rectification Results of Figure 3.8

Hence, it is likely to remain similar in successive frames. Similarly, looking directly at the floor is not beneficial as well. Because of the relative motion of the robot, the floor will experience rapid changes which will make tracking harder. Therefore, tilt angle has been determined as 45° to keep the focus in-between. After the image rectification, scene matching is performed with the SuperGlue network[28] on the perspective images that have been obtained in the first step. In their study, Sarlin et al. showed that the SuperGlue outperforms state-of-the-art feature detectors like SIFT, SURF, and matchers like FLANN. Unfortunately, due to the rectification process, 2 scenes that are to be compared have a total number of 8 perspective images for a given pan angle set. This leads us to the third and last step which is the similarity matrix computation. Due to camera rotation, perspective images that correspond to the same pan angle might not match. Such cases necessitate insensitivity to rotation. Rotation invariance is achieved by brute force matching. That is, each rectified image of the candidate location is compared against the 4 rectifications of the map node. As a result, similarity matrix $S_{m \times m}$ where m is the number of pan angles is obtained. Each element s_{ij} of S is the similarity test result of the comparison between i_{th} &

j_{th} rectifications of the candidate scene and the map node respectively. To clarify, each comparison gives us numerous matching keypoints. Confidence score of each matching keypoint is symbolized with ν . The average of the confidence scores that are computed by the SuperGlue after matching are used to represent the overall similarity between images and formulated in equation 3.3.

$$s_{ij} = \left(\sum_{k=1}^n \nu_k \right) / n \quad (3.3)$$

where

n : Number of matching keypoints

ν : Confidence score of each matching keypoint

Meanwhile, path planning is defined as the process of connecting three or more points with the help of the database map. That is why path planning in this study is a search process that aims to connect initial and target points. One thing to note is the fact that if the initial and target points can be traversed with the reactive navigation (i.e. $S_{RN} \leq \eta_{IP}^{TP}$), path planning is redundant. Although this path is topologically the shortest one (i.e. least number of nodes), it does not necessarily have to be the shortest metrically. If the reactive navigation cannot be used, then the localization module locates the closest map nodes to the given points and transfers these to path planning afterward. Eventually, 2 points are connected through the generated map. Each element of this path is called a *via point* which acts as an intermediate destination for the reactive navigation module. In addition to planning the initial path, localization and path planning module is also responsible for tracking and updating the route during the operation. When the reactive navigation brings the agent to the vicinity of a via point it may or may not see its next destination depending on the geometry and topology of the environment. That is why satisfying a threshold is not enough for path planning to update the route. As a secondary condition, the next via point must be visible in the sense that the agent should be able to compute a minimum similarity value of 1.25 to the next via point to be able to navigate there. Once these 2 conditions are satisfied, path planning accepts the newly arrived point as the via point and updates the path by removing reached via point. Navigation process is illustrated in Figure 3.10.

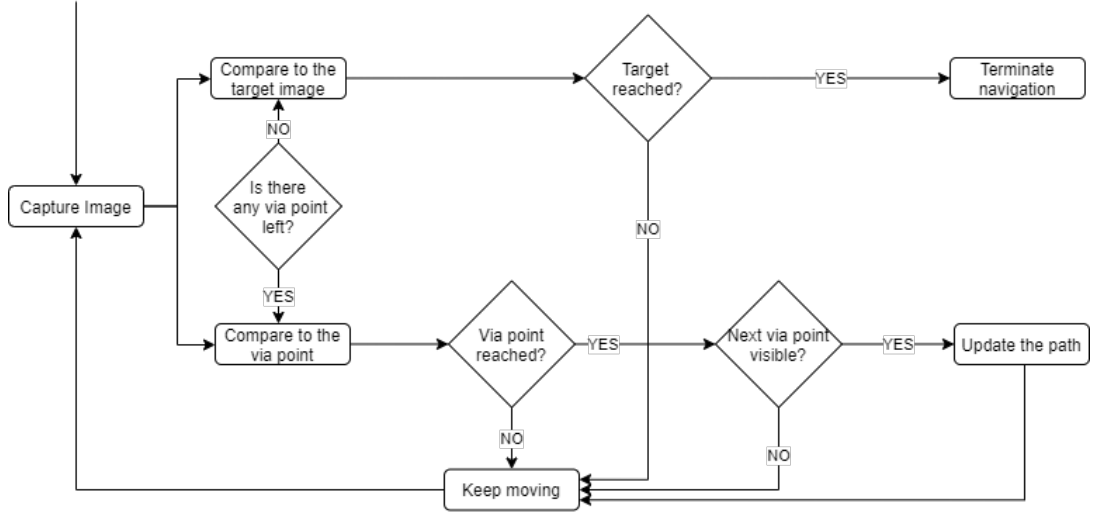


Figure 3.10: Navigation Flowchart

3.3 Reactive Navigation

This is the main algorithmic component that is responsible for locomotion. Reactive navigation relies heavily on the momentary perception of the environment to take an action. The implementation takes place in 3 steps. First of all, 2 scenes are compared and common features are extracted. This is followed by locating these features on the spherical domain by back projection. This is because all of the keypoints are expressed in their local image coordinate system after each rectification. These local coordinate systems are related to the source equirectangular panorama through pan and tilt angles. Since we know at which orientation our camera is and the remaining parameters such as FOV and output image size, we can determine the exact ROI on the sphere that is used to generate corresponding perspective image. Spherical coordinates of the ROI is used to express all of the keypoints uniquely. After the back projection, all of the matching features are expressed in terms of their spherical coordinates μ , λ which refer to the longitude, and the latitude respectively. For the last step of reactive navigation, matching features are projected onto the unit circle by using their longitude values. This allows our algorithm to compute a unit vector that points to each of the features. Unit vectors that belong to the selected feature pair (i.e. u^{c_i} and u^{c_j}) are then used to determine the angular separation in between as stated in equations 2.1 to 2.5. To remind the reader, the vector algebra involved in the process

is repeated below.

$$\begin{aligned}\vec{u}^{c_i} &= \cos(\mu^{c_i}) \vec{i} + \sin(\mu^{c_i}) \vec{j} \\ \vec{u}^{c_j} &= \cos(\mu^{c_j}) \vec{i} + \sin(\mu^{c_j}) \vec{j}\end{aligned}\tag{3.4}$$

$$\begin{aligned}\vec{u}^{t_i} &= \cos(\mu^{t_i}) \vec{i} + \sin(\mu^{t_i}) \vec{j} \\ \vec{u}^{t_j} &= \cos(\mu^{t_j}) \vec{i} + \sin(\mu^{t_j}) \vec{j}\end{aligned}\tag{3.5}$$

$$\begin{aligned}\mathbf{d}^{c_{ij}} &= \underline{\mathbf{u}}^{c_i} \cdot \underline{\mathbf{u}}^{c_j} & \underline{\mathbf{C}}_{ij} &= \underline{\mathbf{u}}^{c_i} \times \underline{\mathbf{u}}^{c_j} \\ \mathbf{d}^{t_{ij}} &= \underline{\mathbf{u}}^{t_i} \cdot \underline{\mathbf{u}}^{t_j} & \underline{\mathbf{T}}_{ij} &= \underline{\mathbf{u}}^{t_i} \times \underline{\mathbf{u}}^{t_j}\end{aligned}\quad \text{where } i \neq j\tag{3.6}$$

$$\mathbf{A}_{ij} = \text{sgn}(\mathbf{d}^{c_{ij}} - \mathbf{d}^{t_{ij}})\tag{3.7}$$

$$\mathbf{B}_{ij} = [\text{sgn}(\underline{\mathbf{C}}_{ij} \cdot \underline{\mathbf{T}}_{ij}) + 1]/2\tag{3.8}$$

$$\underline{\mathbf{u}}_{ij} = (1 + \mathbf{B}_{ij}(\mathbf{A}_{ij} - 1))/(\underline{\mathbf{u}}^{c_i} + \underline{\mathbf{u}}^{c_j}/\|\underline{\mathbf{u}}^{c_i} + \underline{\mathbf{u}}^{c_j}\|)\tag{3.9}$$

$$\underline{\mathbf{h}} = \sum \underline{\mathbf{u}}_{ij}\tag{3.10}$$

With the reactive navigation is elaborated on, the chapter can now be concluded. Shortly, implementation has taken place in 3 different steps which are *Mapping & Exploration*, *Localization & Path Planning* and *Reactive Navigation*. Although the subject of the chapter has been taken into consideration in 3 parts, in the final code they work simultaneously. This is due to the fact that, all of the tasks are intertwined in an incremental navigation problem. Evidently, this is not specific to our application but to all SLAM examples. Next chapter delves into the experiments and results.

CHAPTER 4

EXPERIMENTS & RESULTS

To examine the performance of the reactive navigation algorithm, 2 distinct indoor environments are selected. The first one is a living room and the second is a laboratory. The main reason behind the selection of these 2 places is their different characteristics like clutter, texture, structure, etc. that may have a strong influence on the captured images which can cause image processing module to produce inconsistent results. Nonetheless, these are not the only differences. Illumination is another element that can become impactful on the images. Subjecting the reactive navigation algorithm to a different set of environmental attributes helps us assess the performance more thoroughly. Distinct test environments are also beneficial to test our choice of design parameters such as S_{th} and S_{RN} .

It should be reminded that to be able to investigate the performance as objective as possible, S_{th} and S_{RN} are kept unchanged in-between the experiments and tweaking of them is left for future work. Giving this brief introduction is enough to advance and elaborate on the experiments.

4.1 Hardware Configuration

Experiments have been performed with LG Cam360 spherical camera that is shown in Figure 3.7 and a laptop with an i5-4210H 2.90 GHz CPU and a NVIDIA GTX 950M GPU. In addition, LG Cam360 is compatible with the Google's Open Spherical Camera API [38] and 30cm tall with the mounting equipment. Since the computed heading vectors were to be actuated by hand, the size of 30cm height was considered to be reasonable for portability. Moreover, the height has been taken into account dur-

ing the determination of the tilt angle. To be more precise, as the height increases, the tilt angle has to decrease to capture the same ROI as shown in Figure 4.1. Whereas, an agent with the red FOV and the height of H_2 needs to have the tilt angle of λ_2 to capture the black ROI, an agent with the orange FOV and the height of H_1 needs the tilt angle of λ_1 . According to the camera axes that are given at the top left in the below figure, a counterclockwise rotation in the Y-axis, according to the right-hand rule, corresponds to a positive angle of rotation. This imposes the condition in equation 4.1.

$$\lambda_1 \leq \lambda_2 \quad (4.1)$$

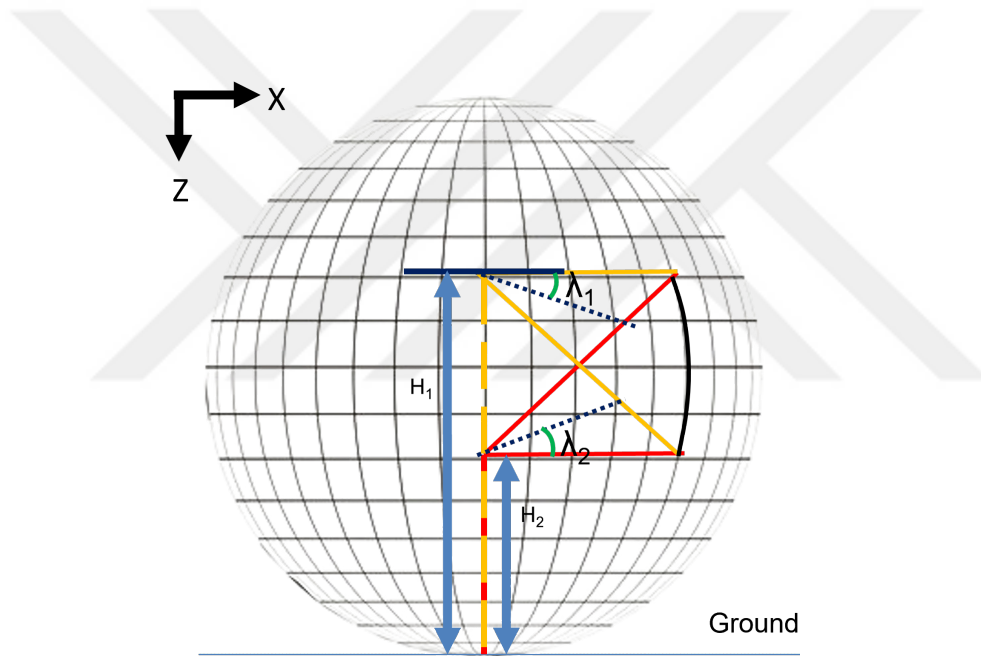


Figure 4.1: Tilt Angle with Height Difference

To prepare the hardware for the test, the laptop is connected to the LG Cam360 over the camera's WiFi network. A custom function named *capture_images* has been written to capture images wirelessly via the laptop by utilizing OSC API. The python library which contains *capture_images* and all of the other functions written during the study is given in Appendix 5.2. The implemented code had been run on the laptop and the generated heading vectors have been tracked in steps of 50cm with the help of a compass app on a smartphone.

4.2 Mapping & Exploration

To begin with, the agent is placed at a random point within the setup and the map generation is initiated. The living room and the laboratory layouts that will be explored and the initial locations of the agent are shown in Figures 4.2 and 4.3.

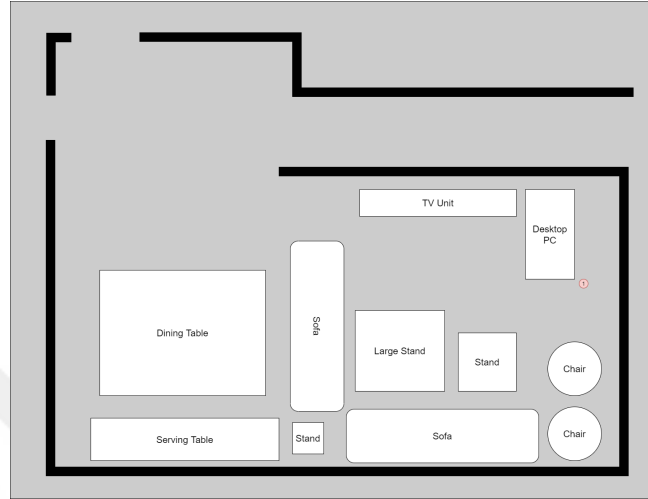


Figure 4.2: Living Room

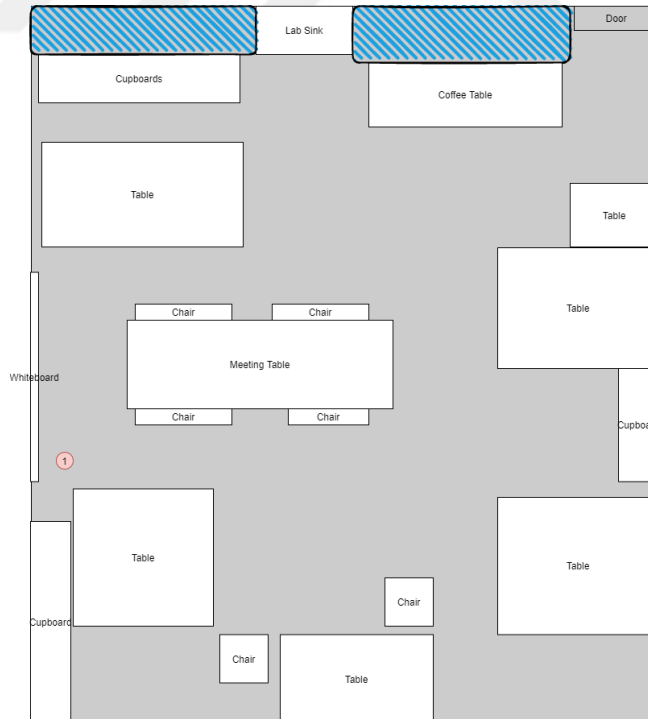


Figure 4.3: Laboratory

We have approached the exploration problem with a random wanderer viewpoint. In addition, as a strategy, we tried to guide the mapping towards the areas that were not visited yet. This is achieved by directing the agent away from the previously visited areas by computing the average heading vector of the 2 closest nodes and reversing it. If we have the heading vectors $\vec{h}_1$ and $\vec{h}_2$ which point towards the first and the second closest nodes respectively and denote the exploration vector as $\vec{h}_{exp}$ then the vector $\vec{h}_{exp}$ can be computed as shown in equation 4.2.

$$\vec{h}_{exp} = -\frac{(\vec{h}_1 + \vec{h}_2)}{2} \quad (4.2)$$

It should also be noted that the qualitative representation of the setting is not unique and highly dependent on the exploration strategy, initial location and the heuristics involved in the mapping phase. That is why different approaches on the subject likely to result in different representations.

69 images were captured during the exploration phase of the living room. 21 of them have satisfied the similarity condition in equation 4.3 and been identified as distinct nodes to build the map. The resulting connectivity graph is presented in Figure 4.4 together with the set of initial and target points for the reactive navigation experimentation. Shaded areas represent the regions that are not explorable by the camera due to physical restrictions although the illustrative material may seem off the scale. Likewise, 94 images were captured to complete the laboratory exploration. 29 of these panoramic images have satisfied the similarity condition in equation 4.3 and have been selected as nodes to represent the explored area. The exploration has resulted in a connectivity graph which is shown in Figure 4.5.

$$S_{RN} \leq \eta_{CN}^{CMN} \leq S_{th} \quad (4.3)$$

where

η_{CN}^{CMN} = Similarity response of candidate node to closest map node

S_{RN} = 1.25(Reactive Navigation Threshold)

S_{th} =: 1.65(Similarity Threshold)

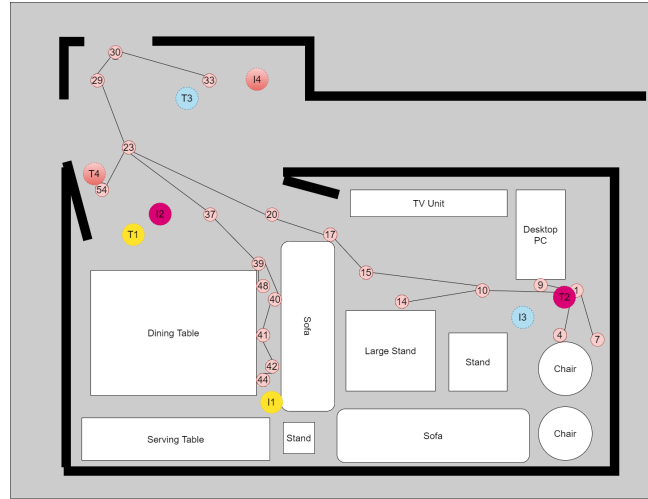


Figure 4.4: Room Connectivity

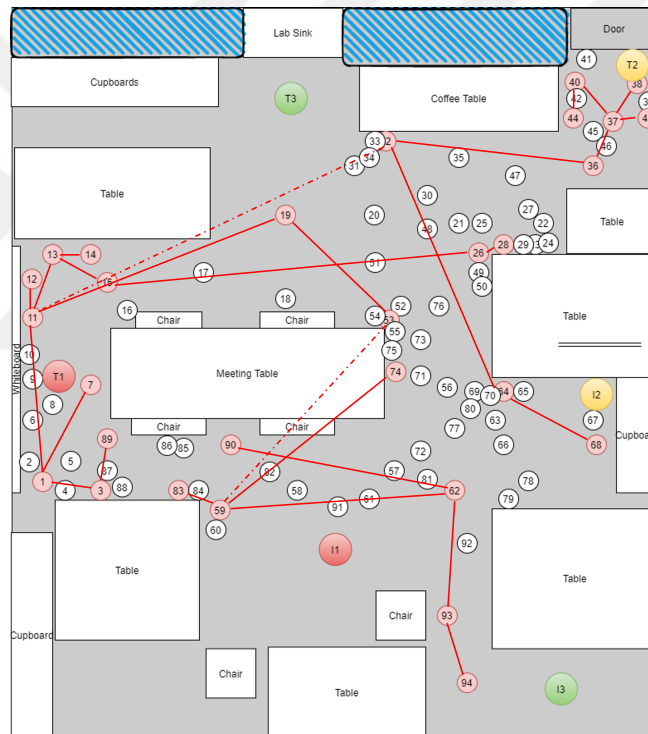


Figure 4.5: Laboratory Connectivity

4.3 Navigation

To investigate the reactive navigation performance thoroughly, 4 and 3 pairs of arbitrary initial and target points are selected in the living room and the laboratory respectively and shown in Figures 4.4 and 4.5) with matching colors.

Experimental procedure is summarized as follows.

- First image is captured at the initial point.
- First image and the target image is compared to validate if the images are similar enough to activate reactive navigation.
- If the reactive navigation is applicable to the images, it is initiated. If not, the closest nodes to the initial and target points are located by utilizing the generated connectivity graph.
- Located nodes are used to generate a path that connects the initial and the target points and transferred to the reactive navigation module as a list of *via points*.
- The reactive navigation module uses via points as intermediate destinations which means that as the reactive navigation manages to reach the via point, it updates its destination with the next via point until the target point is reached.

Obtaining via points is crucial since it serves as an intermediate process in transforming a global navigation problem into a chain of local reactive navigation objectives.

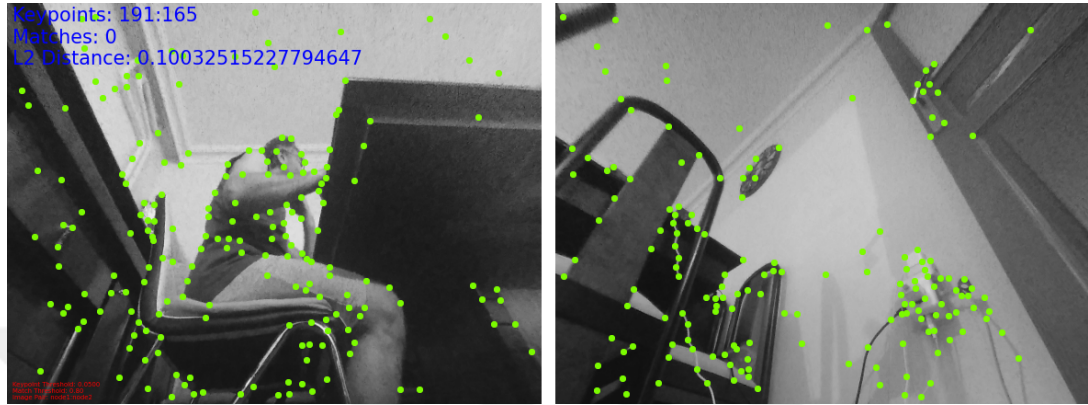
Throughout the experiments, the reactive navigation is considered to be successful if it can bring the agent towards the proximity of its target such that the two scenes respond higher than the similarity threshold, S_{th} , in the scene matching.

The reactive navigation scheme is tested in 4 different scenarios in the living room. To exemplify the navigation process steps, the first scenario is selected. To begin with, the initial scene has been captured and shown in Figure 4.6a.

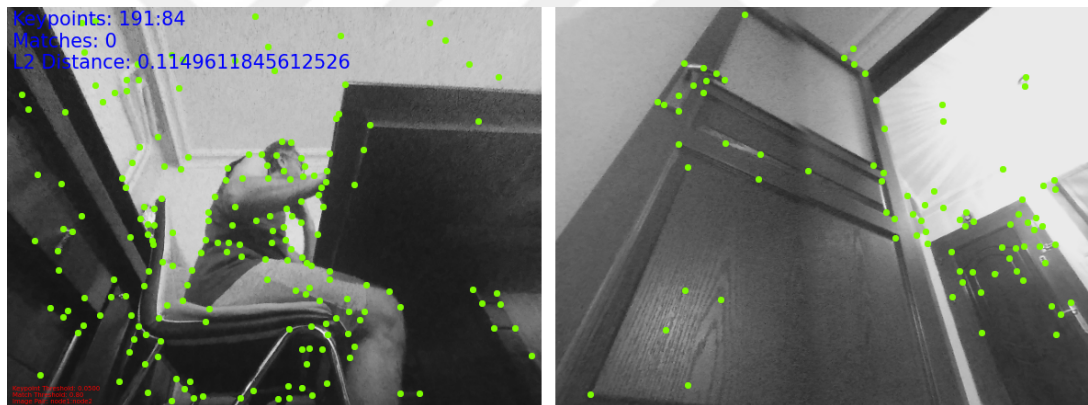


Figure 4.6: Initial (a) and Target (b) Scenes of the First Scenario

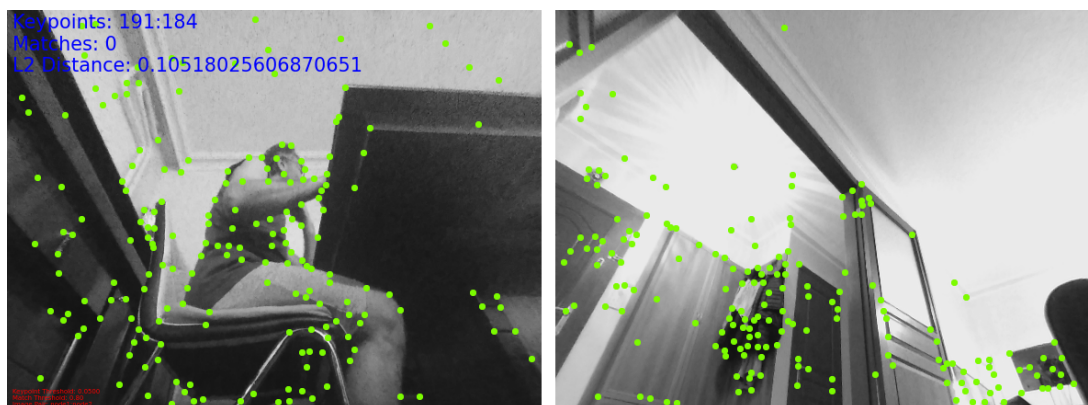
Next, the initial scene and the target scene in Figure 4.6 are compared to check for the reactive navigation availability. The comparison is illustrated through Figures 4.7 and 4.10.



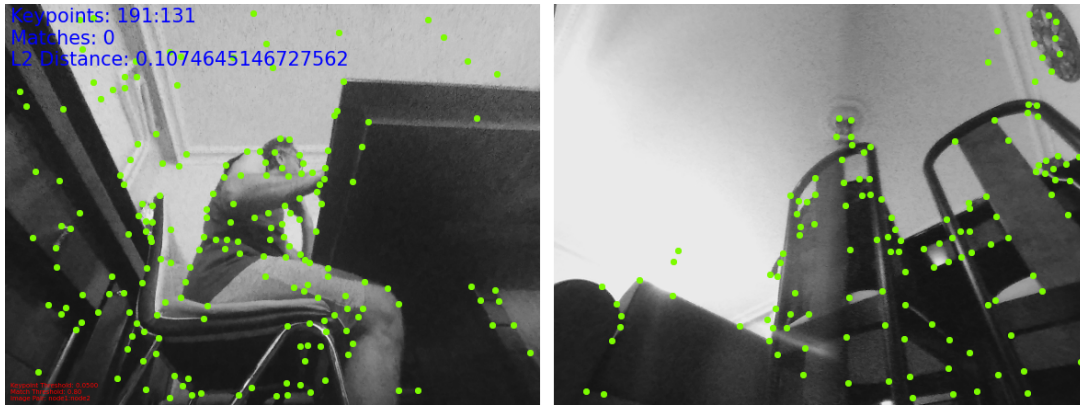
(a) Pan Angles of Rectification: 0° - 0°



(b) Pan Angles of Rectification: 0° - 90°

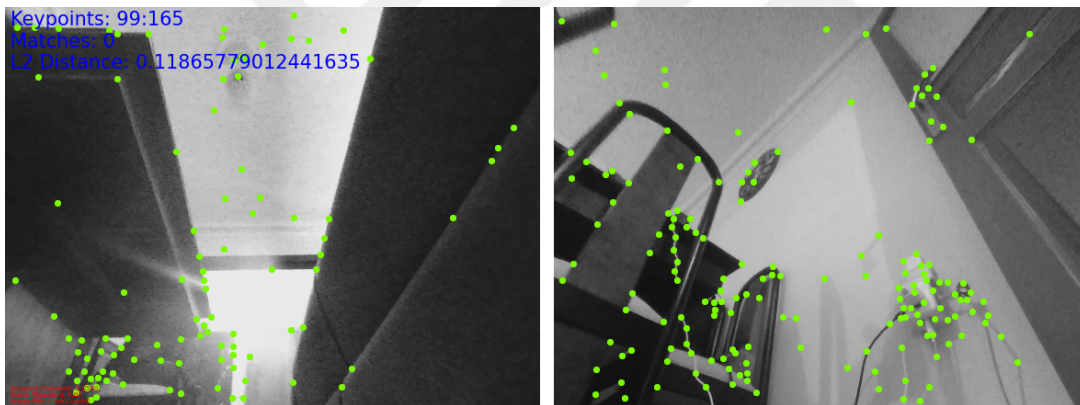


(c) Pan Angles of Rectification: 0° - 180°

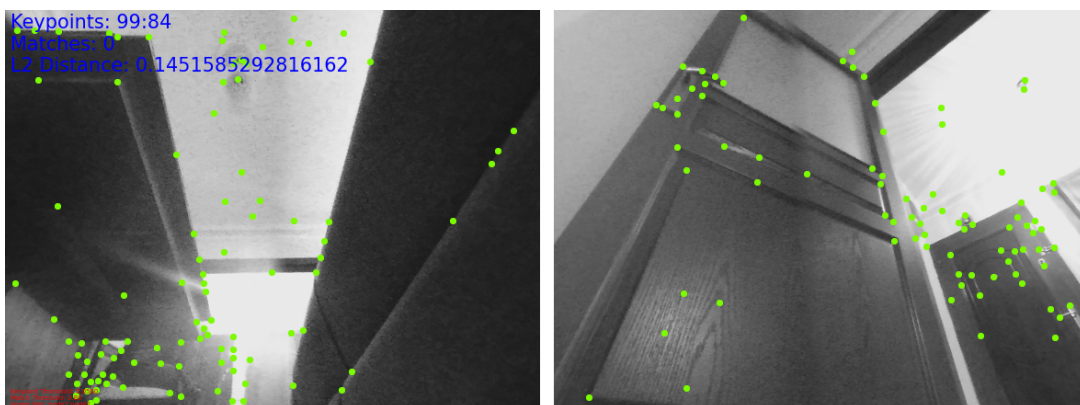


(d) Pan Angles of Rectification: 0° - 270°

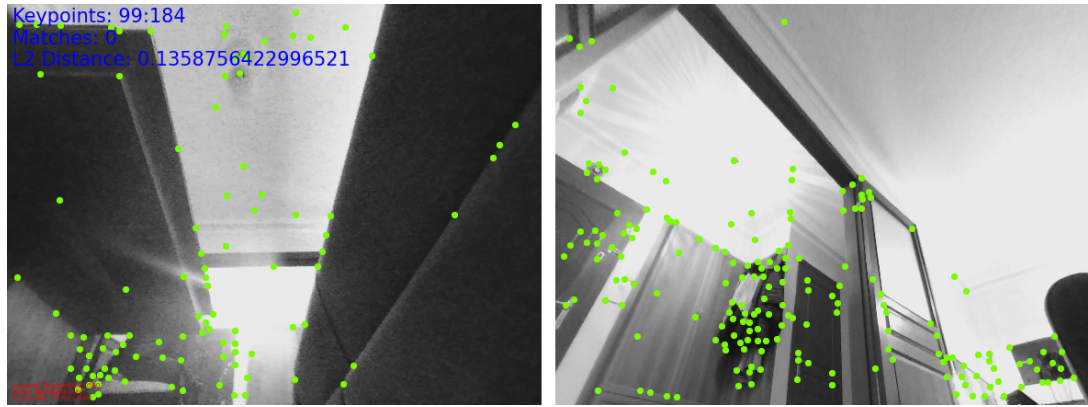
Figure 4.7: Scene Matching - 1



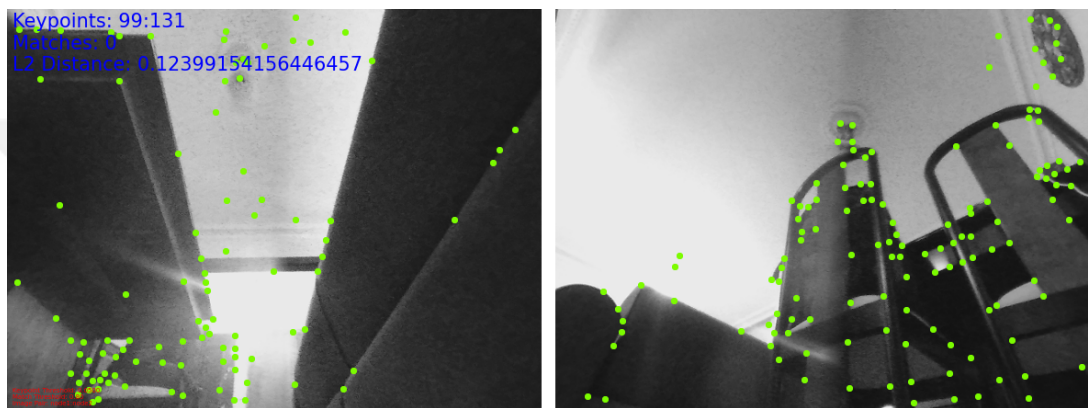
(a) Pan Angles of Rectification: 90° - 0°



(b) Pan Angles of Rectification: 90° - 90°

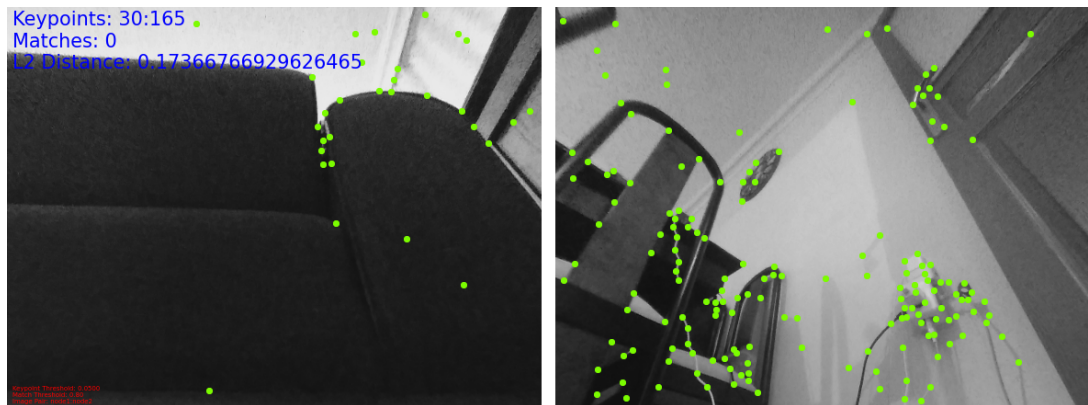


(c) Pan Angles of Rectification: 90° - 180°

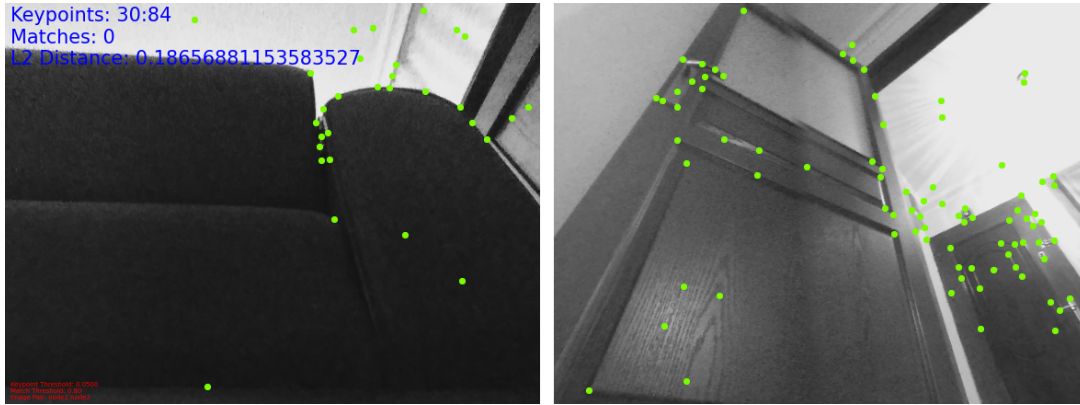


(d) Pan Angles of Rectification: 90° - 270°

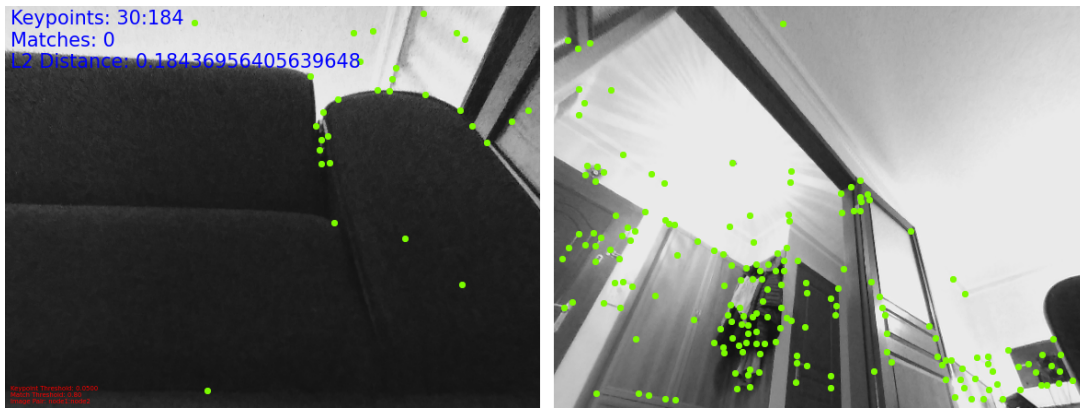
Figure 4.8: Scene Matching - 2



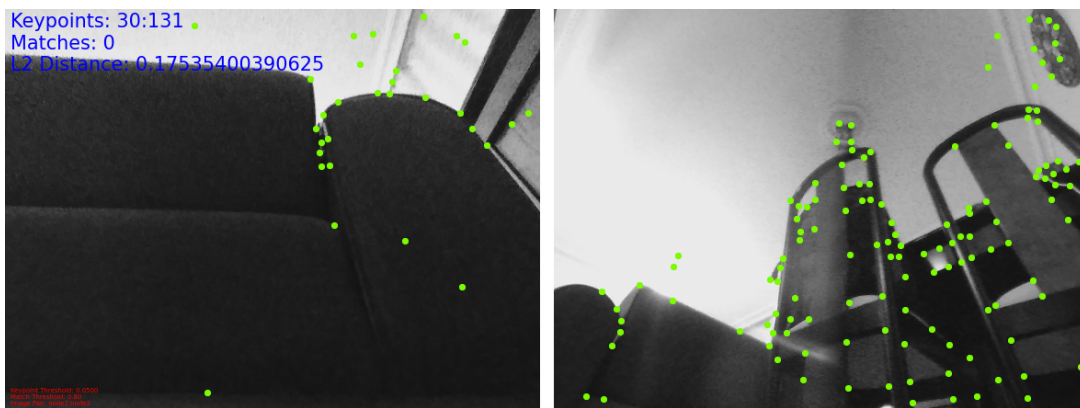
(a) Pan Angles of Rectification: 180° - 0°



(b) Pan Angles of Rectification: 180° - -90°

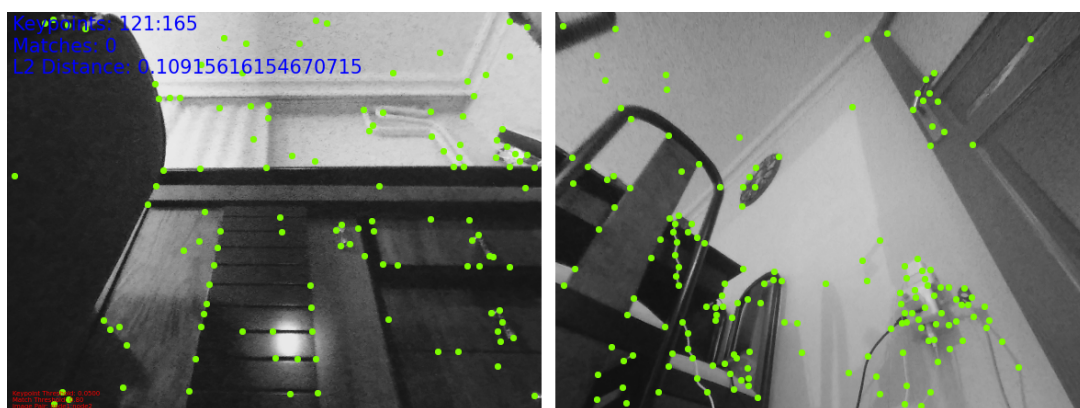


(c) Pan Angles of Rectification: 180° - -180°

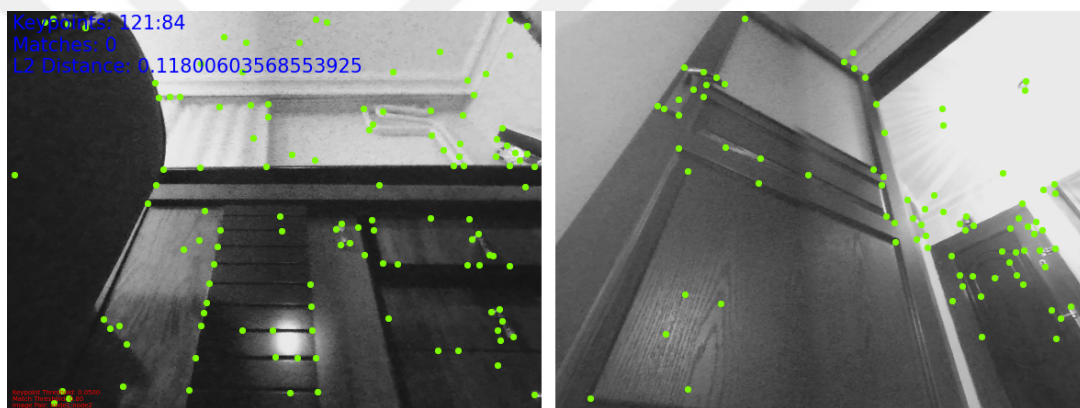


(d) Pan Angles of Rectification: 180° - -270°

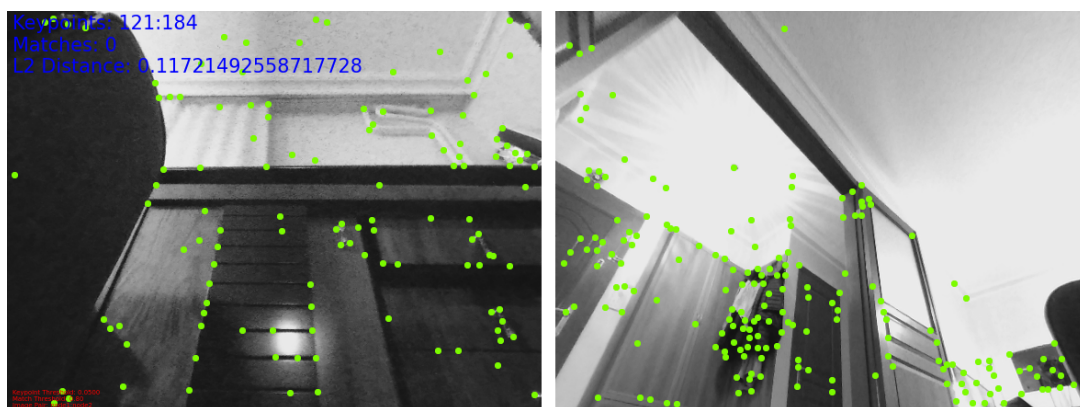
Figure 4.9: Scene Matching - 3



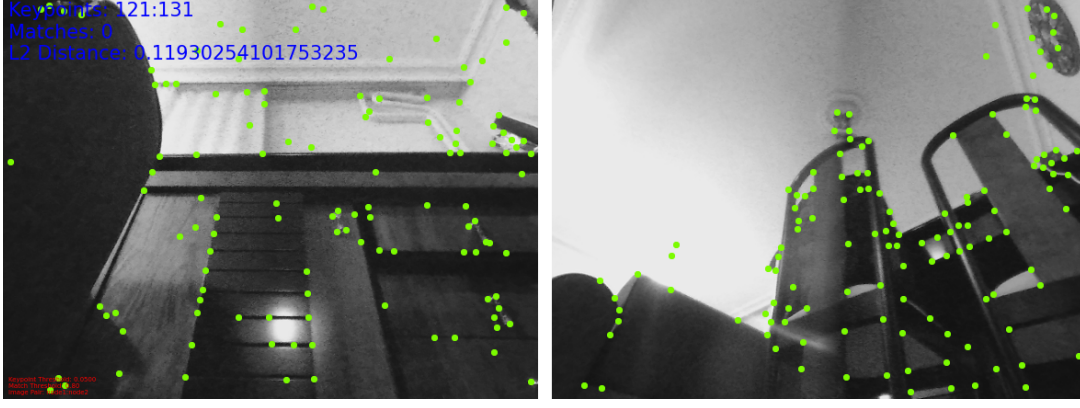
(a) Pan Angles of Rectification: 270° - 0°



(b) Pan Angles of Rectification: 270° - 90°



(c) Pan Angles of Rectification: 270° - 180°



(d) Pan Angles of Rectification: 270°-270°

Figure 4.10: Scene Matching - 4

The image processing module could not locate any matches during the comparison between the initial and the target images. This has resulted in the following similarity matrix, η_{IS}^{TS} where IS & TS refer to the initial scene and the target scene respectively.

$$\eta_{IS}^{TS} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \|\eta_{IS}^{TS}\| = 0$$

Each element of the η_{IS}^{TS} corresponds to the similarity scores of each comparison in Figures from 4.7 to 4.10. To express the similarity of the entire scene with a single number, the norm of the η_{IS}^{TS} matrix is used. Therefore, the similarity score for this comparison is 0. This means that the reactive navigation is not applicable to the initial and target scene pair. The next step is the generation of via points with the localization & path planning module. As the first step, the closest nodes to the initial and the target scenes are located within the connectivity graph with the *find_nearest_node* function that is given in the Appendix 5.2. The function has returned the images *img42* and the *img37* as the most similar scenes to the initial and the target images respectively. Similarly, example use cases of the custom functions in Appendix 5.2 are given in Appendix 5.2. Afterwards, a path that connects *img42* and *img37* is searched over the connectivity graph. The path that is found is shown in the Figure 4.11.

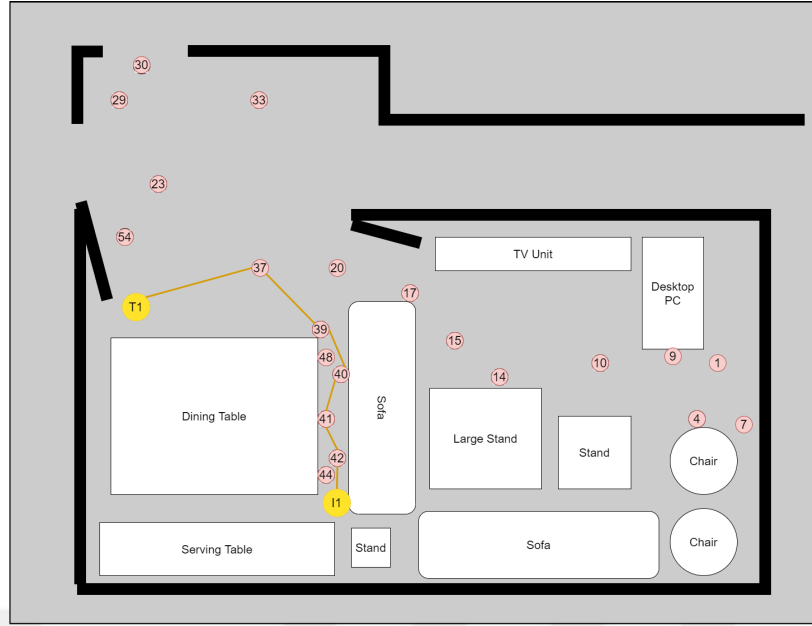


Figure 4.11: A path that connects two closest nodes

Additionally, the list of via points is obtained as:

$$via\ points = [42, 41, 40, 39, 37, T1]$$

as can be seen in 4.11. Following the transfer of the via points to the reactive navigation module, the reactive navigation is initiated between the initial scene and the first via point. As the beginning step of the reactive navigation, the scene matching is performed in the same way as in Figures 4.7 to 4.10 and the matching keypoints are extracted. Then, by using *get_keypoint_latlon* function in Appendix 5.2, the matching keypoints are projected back onto the sphere and their spherical coordinates are obtained. Importance of the back projection stems from the fact that it functions as a unifying step which means that instead of dealing with 4 different rectified images of the scene, we can express the entire scene in the spherical domain as a collection of uniquely addressed keypoints thanks to the spherical coordinates. Out of the spherical coordinates, the longitude is used to compute the local sights of the initial scene and the first via image by the rectangular decomposition. Lastly, the heading vector is generated by processing the keypoint pairs in-between the scenes. These keypoint pairs are formed on the local sight of the initial scene. Since the same keypoints are also available in the first via image thanks to the matching, the same pairs can be formed in the via image as well. The reactive navigation derives the heading vector

from the change of the angle of these pairs in-between the scenes. Figure 4.12 shows the local sights of the initial scene and the first via image together with a couple of keypoint pair examples in matching colors.

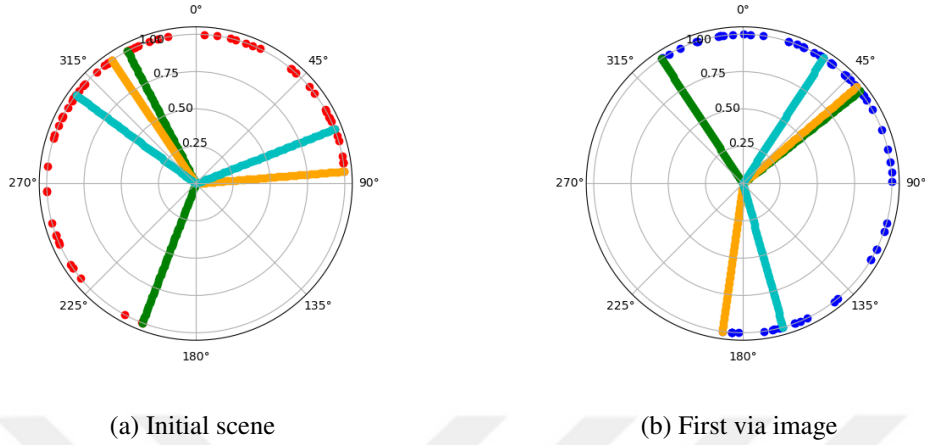


Figure 4.12: Representation of the Local Sights

Processing the keypoint pairs has generated the heading vector which is presented in the Figure 4.13.

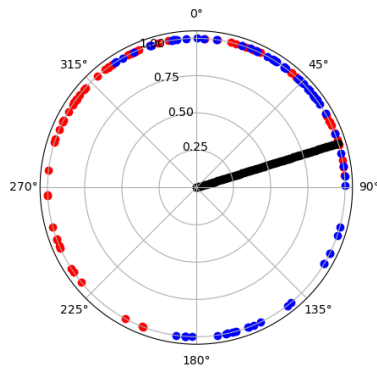


Figure 4.13: Overlapped Local Sight and the Heading Vector.

Generating the heading vector is the last step for the reactive navigation. Once a step is taken in the direction of the heading, the navigation process is repeated until the last via point is reached.

Meanwhile, the navigation process has been visualized through the first reactive navigation attempt of the first scenario in the living room, paths that have been generated

Figure 4.14: Scenario 2

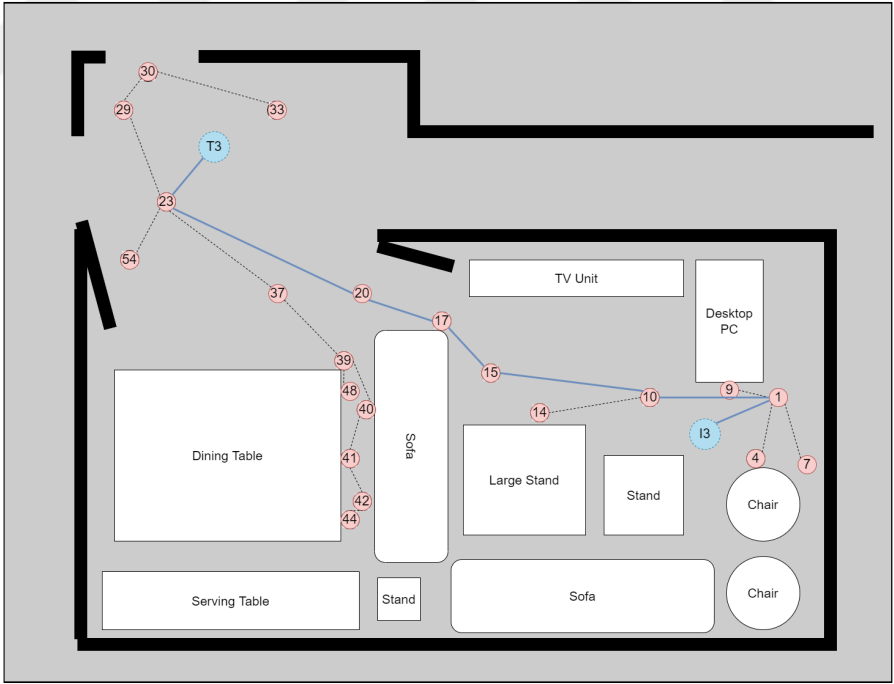


Figure 4.15: Scenario 3

45

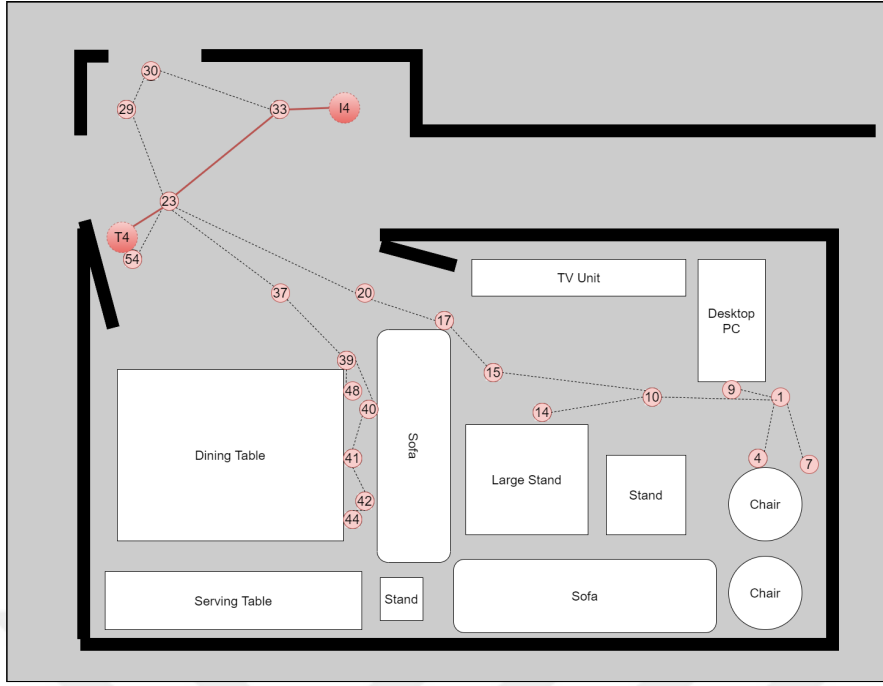


Figure 4.16: Scenario 4

following:

via points of Scenario 2 = [37, 23, 20, 17, 15, 10, 1, T2]

via points of Scenario 3 = [1, 10, 15, 17, 20, 23, 29, 30, 33, T3]

via points of Scenario 4 = [33, 23, T4]

Besides the living room, the reactive navigation algorithm has been tested in 3 different laboratory scenarios. Besides, testing the reactive navigation algorithm in another environment allows for the assessment of its response to different external characteristics like geometry, lighting, occlusion, etc. In order to examine the influence of the aforementioned characteristics as objectively as possible, the design parameters (i.e. similarity threshold, S_{th} and the reactive navigation threshold, S_{RN}) have been kept as they were in the living room during the entire laboratory experimentation. All of the scenarios, as in the living room study, had been initiated with their respective initial scenes that are shown in Figure 4.17 and none of them could engage the reactive navigation directly. In other words, all of the scenarios required a path planning. A search within the connectivity graph resulted in the paths that are shown in the Figure 4.17 with the colors red, orange and green successively.



Figure 4.17: Laboratory Scenarios

Via points of each scenario are also detected as the following:

via points of Scenario 1 = [59, 53, 19, 11, 1, 7, T1]

via points of Scenario 2 = [68, 64, 32, 36, 37, T2]

via points of Scenario 3 = [94, 93, 62, 59, 53, 19, T3]

Table 4.1 presents the summary of the navigation scenarios. In the table, *LR* and *L* refer to the living room and laboratory respectively. *# of Images Captured* column is the total number of images captured during the scenario including the initial capture. Additionally, *Steps Taken for* refers to how many steps the agent have taken to get to

the specified via point after the previous one. For example, for the scenario 3 of the living room (i.e. LR 3), it took 4 steps for the agent to arrive at the 4th via point from the 3rd one. Also,

$$\# \text{ of Images Captured} = \left(\sum_{j=1}^m \text{Steps Taken for } j_{th} \text{ via point} \right) + 1 \quad (4.4)$$

where m is the # of via points and 1 is the addition of the initial image capture.

Test Area	# of Via Points (VP)	# of Images Captured	Steps Taken for									
			1st VP	2nd VP	3rd VP	4th VP	5th VP	6th VP	7th VP	8th VP	9th VP	10th VP
LR 1	6	13	1	2	4	2	2	1	-	-	-	-
LR 2	8	21	1	2	1	1	4	9	1	1	-	-
LR 3	10	33	2	4	4	4	4	1	9	1	2	1
LR 4	3	4	1	1	1	-	-	-	-	-	-	-
L 1	7	15	2	5	1	3	1	1	1	-	-	-
L 2	6	18	1	1	5	1	8	1	-	-	-	-
L 3	7	16	1	1	2	2	8	1	0	-	-	-

Table 4.1: Summary of the Navigation Experiments

At a first glance, 6th via point of LR 2, 7th via point of LR 3 and 5th via points of both L 2 and L 3 scenarios stand out. The number of steps that have been taken to arrive at the specified via points are substantially higher than the rest of the table. Taking a closer look at the recorded data reveals that the agent has exhibit a motion around the specified via point which causes it to experience rapid changes in the local sight. To clarify, an imaginary example would be a state of distorted line of sight with the via point. When an agent approaches to the via point, due to the sudden change of the line of sight, the similarity is decreased. This can be caused by the restricted line of sight to the via point which makes it visible only in a certain FOV. As the similarity decreases, the reactive navigation computes a heading vector to compensate for the missing similarity. However, the translation due to the computed heading vector disturbs the line of sight more contrary to the expectation. Hence, the reactive navigation algorithm exhibits such oscillatory behavior.

Besides the navigation summary, Table 4.2 presents the total and the average image processing results in terms of features and matches. N_f and N_m refer to the total number of detected features and matches respectively for the sum of the *# of Via Points* and the *# of Images Captured* in the Table 4.1. Likewise, $\overline{N_f}$ and $\overline{N_m}$ are obtained by averaging the N_f and N_m over the same sum.

	N_f	N_m	$\overline{N_f}$	$\overline{N_m}$	$\overline{N_m}/\overline{N_f}(\%)$
LR 1	9504	786	500.21	41.37	8.27
LR 2	14719	1134	507.55	39.10	7.70
LR 3	25348	1659	589.49	38.58	6.54
LR 4	3684	320	526.29	45.71	8.69
L 1	12047	665	547.59	30.23	5.52
L 2	12415	567	517.29	23.63	4.57
L 3	13817	660	600.74	28.70	4.78

Table 4.2: Summary of the Image Processing

In its more compact form, the Table 4.2 can be expressed by averaging the data over test spaces (i.e. the living room and the laboratory). This generates the Table 4.3 and helps in understanding the influences of the environmental characteristics like texture, symmetry, etc. more clear.

Test Area	N_f	N_m	$\overline{N_f}$	$\overline{N_m}$	$\overline{N_m}/\overline{N_f}(\%)$
$\overline{LR}$	13314	975	531	41	8
$\overline{L}$	12760	631	555	28	5

Table 4.3: Average Image Processing Results of Environments

As opposed to the living room, the average matching performance is %31.7 (i.e. 41/28) lower in the laboratory whereas the average keypoint detection is %4.1 (i.e. 555/531) higher as presented in Table 4.3. Once paid closer attention, it is seen that the rectification parameters cause the algorithm to focus on the texture-less and highly symmetrical regions. This can be overcome by adjusting the rectification parameters such that the resulting perspective image captures more of the distinctive areas. Another feasible solution would be the relaxation of detection and matching thresholds

provided that the false positive matches are kept under supervision. Because the false positives can alter the local sight and hence mislead the heading vector.

In different point of view, Table 4.3 shows that the reactive navigation algorithm has been able to generate a heading vector by using a fewer number of features in the laboratory. However, concluding that the number of matches that are involved in the heading generation process does not matter would be misleading. In fact, it contributes directly to the robustness of the method since under different circumstances like illumination, clutter, etc. may the same matches not be detected consistently. Therefore, having more features than enough to track is desirable.

Thanks to the different characteristics of the selected spaces, it can be concluded that the tuning of the design parameters depending on the environmental conditions is one of the aspects that needs further consideration in the future. In other words, keeping the design parameters same meanwhile changing the experimental conditions creates an off-design condition for the algorithm and this can cause the algorithm to fail if not addressed properly. Regardless, for all of the scenarios, the reactive navigation algorithm managed to compute the heading vector although it is tested in its off-design condition in some of them. The fact that the reactive navigation succeeded in finding true positive matches that are enough to terminate the navigation task successfully in the off-design scenario is an indication of robustness whose limit needs to be discovered.

4.4 Results

The presented navigation scheme has been experimented within the living room and the laboratory. The fact that these two environments have different characteristics in terms of clutter, texture, etc. enabled us to observe how the algorithm responds to these attributes and to draw conclusions.

Both experiments are started with a mapping phase. The exploration algorithm is called following the initial arbitrary placement of the camera. The random wanderer approach is used for the exploration task. Unfortunately, we had cases where this approach had failed to complete the mapping and stuck. Trapping has been observed when the agent gets to the vicinity of a boundary where the majority of the FOV is

obscured. Additionally, in our case, the boundary that the agent gets too close lacks features. The blockage of the FOV and the lack of features causes that particular scene (i.e. 4.18b) to pretend as a new node and to satisfy the similarity inequality although it does not in reality. The algorithm responds to the situation by connecting it to the Scene 24 which is the newest node in the map (i.e. 4.18a) by that moment, since the node addition condition is satisfied. As a result, a loop emerges in which the agent gets stuck. That is, due to the falsely added node, exploration strategy tries to drift apart from Scene 24 and Scene 25. Moreover, the obstacle avoidance signal tries to keep the agent away from the boundaries. Inevitably, two strategies start to act against each other hence trapping the robot. Figure 4.18 illustrates the setting where the trapping has been observed.

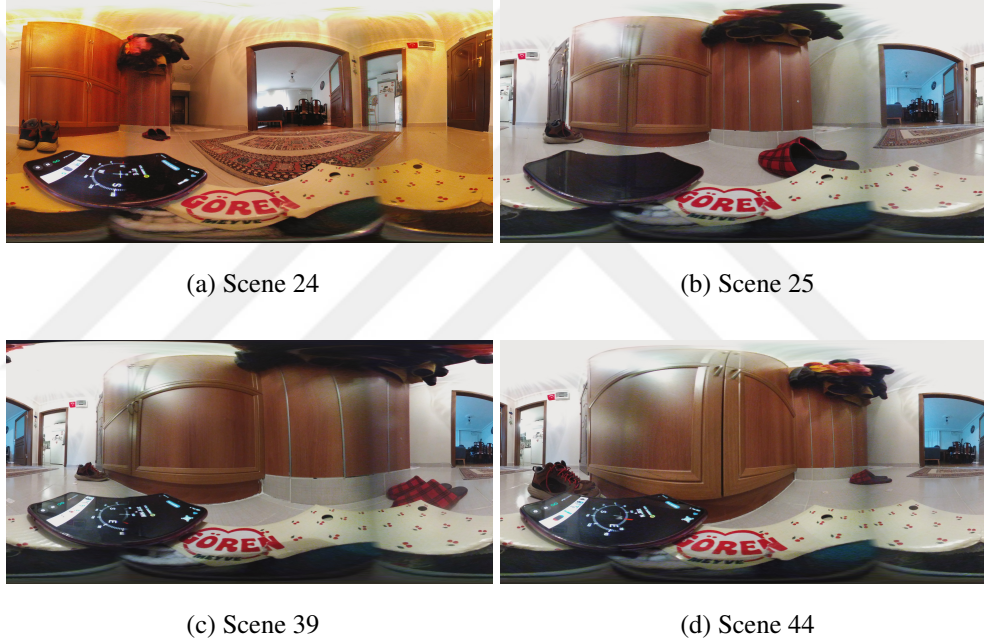


Figure 4.18: Trapping During Random Wanderer Exploration

Therefore, mapping of the living room took a couple of trials to complete. Likewise, although we have observed the trapping behavior during the laboratory mapping, it was not as severe and we managed to complete the mapping. During the mapping phase, it was noticed that the algorithm was inclined to detect newer nodes and update the map near the boundaries due to the rapid change in similarity score as a result of the obscured FOV.

Besides, in the laboratory experiment, the distance between the nodes has been observed to be larger than that of the living room. The fact that the laboratory has larger open spaces enables an unobscured FOV that allows a certain feature to stay in the sight longer and therefore to be tracked for an extended period of time. In return, it takes longer distance to decrease the similarity score enough to add a new node. Being able to detect, track and match features within unobscured FOV allows the algorithm to form connections within the map, even if the connected nodes are not within the direct line of sight of each other. Nodes 11 & 32 and nodes 53 & 59 in the laboratory map shown in Figure 4.5 are the examples of such connections. Scenes that are captured at the former node pair are shown in Figure 4.19.



(a) Scene 11



(b) Scene 32

Figure 4.19: Indirect Map Connection

After the completion of the mapping phase, reactive navigation experiments were conducted. Since the algorithmic parameters (i.e. detection and matching thresholds) and design parameters (S_{th} and S_{RN}) are determined in the living room area, a devia-

tion of performance has been observed in the laboratory in terms of decreased number of average matches together with a increased number of false positive feature matches whose example is given in Figure 4.20.

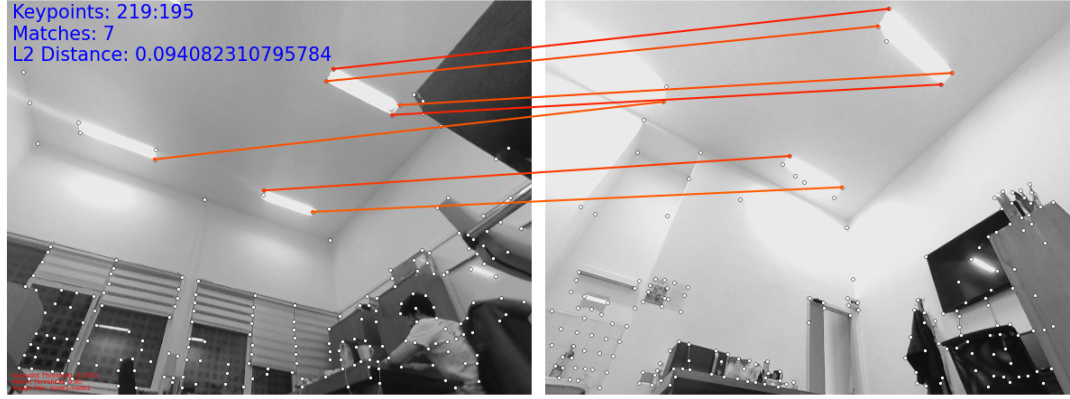


Figure 4.20: False Positive Matches

In overall, all of the reactive navigation scenarios are completed successfully. Final images that terminate the navigation task upon capture are given through Figures 4.21 to 4.27 together with the provided target scenes.

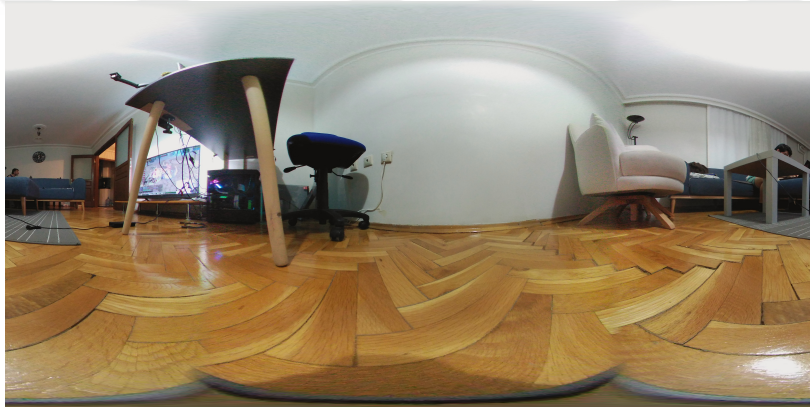


(a) Target Scene



(b) Last Capture

Figure 4.21: Final Scene Pair - 1



(a) Target Scene



(b) Last Capture

Figure 4.22: Final Scene Pair - 2

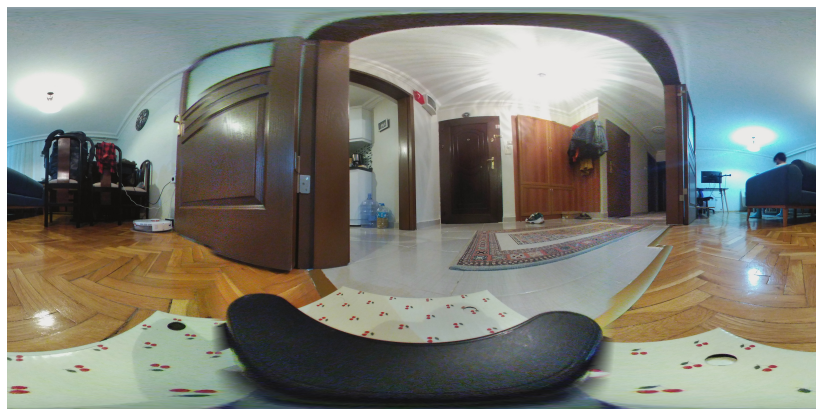


(a) Target Scene



(b) Last Capture

Figure 4.23: Final Scene Pair - 3



(a) Target Scene



(b) Last Capture

Figure 4.24: Final Scene Pair - 4

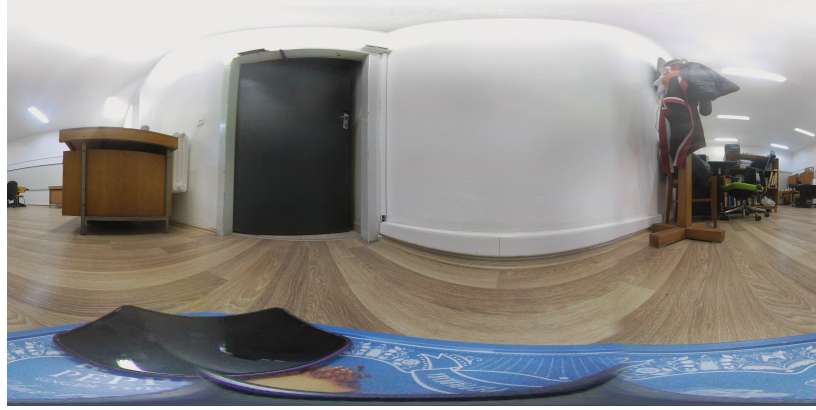


(a) Target Scene



(b) Last Capture

Figure 4.25: Final Scene Pair - 5



(a) Target Scene



(b) Last Capture

Figure 4.26: Final Scene Pair - 6



(a) Target Scene



(b) Last Capture

Figure 4.27: Final Scene Pair - 7

On the whole, experiments were conducted in 2 places that differ in illumination, clutter, texture, etc. To be able to observe how the reactive navigation algorithm responds to changing environmental conditions, design parameters had been kept unchanged throughout the experiments. In return, an unexpected results such as false positive and indirect matches have been encountered. Further analysis reveals that the false-positive matches seem to stem from the fact that the SuperGlue network could not manage to discriminate the descriptors of those features. On the contrary, indirect matches are true positive matches. Nevertheless, being able to connect scenes through the intersection of the line of sight without considering the direct visibility has been observed to cause misconnections within the map. In spite of the findings, all of the experiments were completed successfully but the overall conclusion and rooms for improvement will be given in the next chapter.

CHAPTER 5

CONCLUSIONS & FUTURE WORK

5.1 Conclusions

Technological advances, especially in the sensor technologies, enhances the novel approaches in the field of visual SLAM. For instance, algorithms like OpenVSLAM [39], MonoSLAM [40], ORB-SLAM[41] and ORB-SLAM2[42] are considered to be the state-of-the-art examples of vision-based approaches in the SLAM field. The goal of the work that is presented in the thesis is to compose an algorithm that utilizes a spherical vision to navigate reactively within the environment with the help of a topological map. Concisely, a reactive navigation algorithm that uses panoramic images for topological navigation is devised. The algorithm is reactive in the sense that the information it uses is restricted to the local region. Moreover, it does not keep neither the history of actions nor an internal state as in the deliberate navigation approach. The direction to go is computed by observing relative angular state changes between frames. Use of panoramic images is more advantageous than perspective images during the observation, since they enable tracking of a certain feature for a longer period of time thanks to their wide FOV. To extend the capability of reactive navigation beyond the local sight, the reactive navigation algorithm is supported with an exploration module with random wanderer approach. The aim is to map the environment so that the agent, through a path planning, can partition the assignment into smaller segments which can be covered with the reactive navigation.

During the mapping phase, we tried to guide the agent towards unexplored regions by computing the heading vector that points away from the two closest nodes. As a

drawback, we have observed a behavior similar to repulsive potential fields in particular situations. That is, when the computed heading vector leads the camera towards the boundaries, it tends to stuck. Because after turning back from the boundary, the closest nodes that are used to compute the heading are still the same. An example of such behavior is shown in Figure 5.1. As a protective measure, an iteration limit has been introduced to the algorithm such that if a new node could not be added to the map within the iteration limit, the exploration is disabled temporarily and the agent is sent back to the newest node in the map. Then, the exploration is resumed.

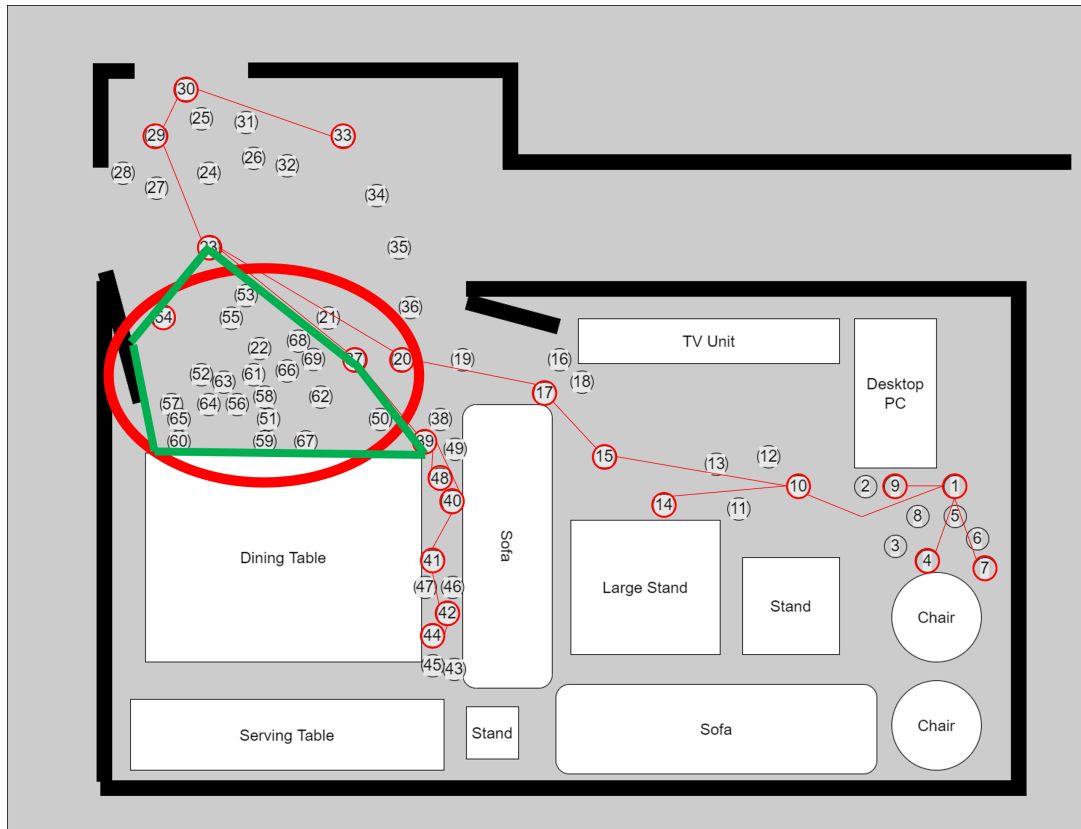


Figure 5.1: Node Cluttering

The reactive navigation scheme is tested in 7 different scenarios within 2 distinct settings. These are shown in Figures from 4.11 to 4.16 and 4.17. When the initial and target scene pair are rejected in an attempt of the reactive navigation, a path is searched in the map between the nodes that are closest to each of the scenes. The nodes of this path are called *via points*. These are the nodes that divide the global navigation problem into a chain of local reactive navigation objectives. Nevertheless, when the camera moves into the vicinity of a via point it is not always possible to

compute the heading vector to the next via point. This inconvenience arise when the next via point is occluded due to the geometry, clutter, etc. or is visible in a particular orientation. For example, a narrow passages that behave like tunnels limit visibility to the linear line of sight). Hence, once the adjacency of a via point is reached, the visibility of the next via point is checked. The visibility check is done by comparing the similarity between scenes to the reactive navigation threshold, S_{RN} . This is exemplified in Figure 5.2. When an image is captured at a blue dot, it is accepted as node 15 however, it cannot navigate to node 17. Therefore, a further attempt is made to increase visibility.

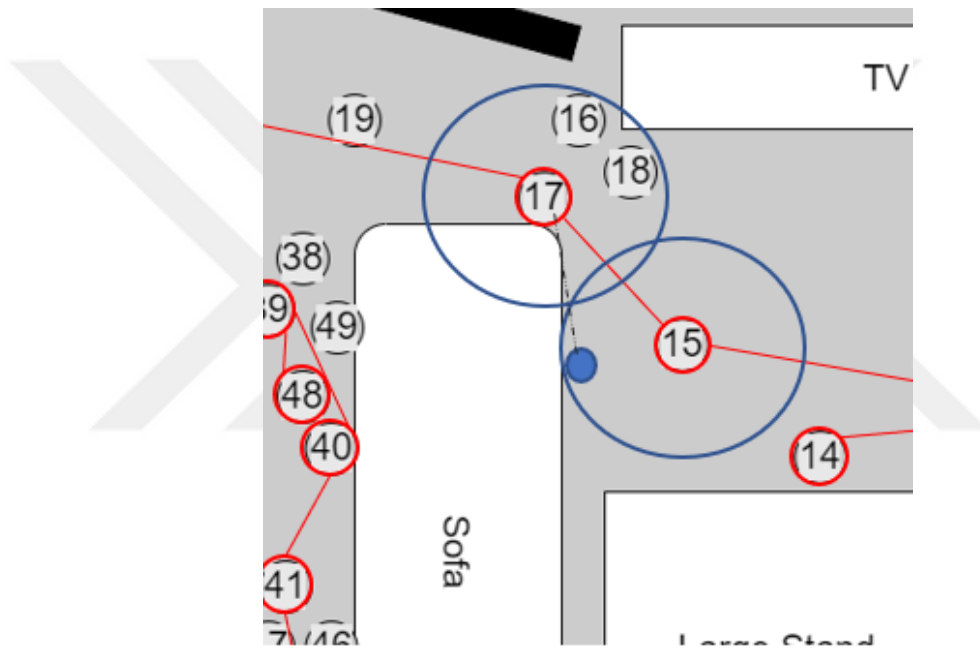


Figure 5.2: Obscured Line of Sight

Presented work in the thesis is built upon the images captured by the spherical camera. Contrary to perspective imagery devices, what spherical cameras bring to the table is the spherical vision which captures the entire line of sight without any blind spot. Thanks to it, the feature can be kept within the sight as long as possible throughout the navigation task. Although this may sound opportunistic at a first glance, it comes with a drawback. Picturing the entire line of sight causes different images to share a patch of the scene at the intersection of their line of sight. This is not a problem when these images are visible to each other as well. However, if they are not visible to



Figure 5.3: Indirect Map Connection

each other, linking them as a result of matching the features that are detected within the patch constitutes a false connection within the map. This flaw has been managed to be detected during analysis conducted after the mapping phase of the laboratory and is visualized in Figure 5.3. Therefore, it is concluded that further reasoning is required regarding the evaluation of the matching features and contribution of them to the map connectivity.

With the current state of the implemented reactive navigation algorithm, mapping & exploration module benefits from the open space the most. However, as the clutter is increased, node cluttering problem may arise. Contrarily, large open spaces such as empty rooms degrades the image processing performance ought to the geometry and the lack of texture. Hence, the functionality of the navigation module depends on the design parameters, S_{th} and S_{RN} . Although the adjustment of them is left for the future work, the values that are set within the thesis give results that are accurate enough for the navigation task. To conclude, as long as the requirements of the mapping & exploration module and the navigation module are not contradicting, the reactive navigation algorithm in its current state is expected to function reasonably.

5.2 Future Work

The reactive navigation scheme that is experimented upon in the thesis is aimed to serve as a framework or backbone for further studies. Therefore, it is aimed to lay out the proof of concept regarding the topic. That is why optimization of the algorithmic parameters have not been studied in detail. This is one of the improvements that could

be worked upon. In addition, on the navigation side, the exploration methodology can be improved in terms of coverage. A method that focuses more on the coverage can help reduce the amount of time that is required for mapping since it will lead to more homogeneous exploration and can overcome the main drawback of the present implementation which is the trapping tendency when adjacent to boundaries. Another aspect of the navigation scheme that can be improved is the adaptive or automatic detection of the similarity threshold, S_{th} and the reactive navigation threshold, S_{RN} . This can alleviate the performance deviation and enable the algorithm to function better in different environments. On the image processing side, tweaking of detection and matching thresholds can ameliorate the scene matching hence, the robustness to different environmental characteristics. As addressed in the Section 5.1, the way the true positive matches are used in node addition and map generation is another issue that should be improved.



REFERENCES

- [1] G. Dissanayake, S. Huang, Z. Wang, and R. Ranasinghe, “A review of recent developments in simultaneous localization and mapping,” in *2011 6th International Conference on Industrial and Information Systems*, IEEE, Aug. 2011.
- [2] J. J. Leonard and H. F. Durrant-Whyte, “Simultaneous map building and localization for an autonomous mobile robot.,” in *IROS*, vol. 3, pp. 1442–1447, 1991.
- [3] “Darpa grand challenge.” <https://cs.stanford.edu/group/roadrunner//old/announcements.html>. Accessed: 2021-12-24.
- [4] “Darpa urban challenge.” , <https://www.darpa.mil/about-us/timeline/darpa-urban-challenge>. Accessed: 2021-12-24.
- [5] L. Maohai, W. Han, S. Lining, and C. Zesu, “Robust omnidirectional mobile robot topological navigation system using omnidirectional vision,” *Engineering Applications of Artificial Intelligence*, vol. 26, pp. 1942–1952, 9 2013.
- [6] T. Taketomi, H. Uchiyama, and S. Ikeda, “Visual SLAM algorithms: a survey from 2010 to 2016,” *IPSI Transactions on Computer Vision and Applications*, vol. 9, June 2017.
- [7] E. Garcia-Fidalgo and A. Ortiz, “Vision-based topological mapping and localization methods: A survey,” *Robotics and Autonomous Systems*, vol. 64, pp. 1–20, Feb. 2015.
- [8] N. Winters, J. Gaspar, G. Lacey, and J. Santos-Victor, “Omni-directional vision for robot navigation,” in *Proceedings - IEEE Workshop on Omnidirectional Vision, OMNIVIS 2000*, pp. 21–28, 2000.
- [9] M. Skubic, S. Blisard, A. Carle, and P. Matsakis, “Hand-drawn maps for robot navigation,” in *AAAI Spring Symposium, Sketch Understanding Session*, p. 23, 2002.

- [10] K. Kawamura, R. A. Peters, D. M. Wilkes, A. B. Koku, and A. Sekman, "Toward perception-based navigation using egosphere," in *Mobile Robots XVI*, vol. 4573, pp. 137–147, International Society for Optics and Photonics, 2002.
- [11] S. E. Healy, *Spatial representation in animals*. Oxford University Press, 1998.
- [12] N. Tomatis, I. Nourbakhsh, and R. Siegwart, "Hybrid simultaneous localization and map building: a natural integration of topological and metric," *Robotics and Autonomous systems*, vol. 44, no. 1, pp. 3–14, 2003.
- [13] D. Schleicher, L. M. Bergasa, M. Ocaña, R. Barea, and M. E. López, "Real-time hierarchical outdoor slam based on stereovision and gps fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 440–452, 2009.
- [14] P. Elinas, R. Sim, and J. J. Little, "/spl sigma/slam: Stereo vision slam using the rao-blackwellised particle filter and a novel mixture proposal distribution," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1564–1570, IEEE, 2006.
- [15] D. Murray and J. J. Little, "Using real-time stereo vision for mobile robot navigation," *autonomous robots*, vol. 8, no. 2, pp. 161–171, 2000.
- [16] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278, IEEE, 2016.
- [17] R. Li, J. Liu, L. Zhang, and Y. Hang, "Lidar/mems imu integrated navigation (slam) method for a small uav in indoor environments," in *2014 DGON inertial sensors and systems (ISS)*, pp. 1–15, IEEE, 2014.
- [18] X. Chen, A. Milioto, E. Palazzolo, P. Giguere, J. Behley, and C. Stachniss, "Suma++: Efficient lidar-based semantic slam," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4530–4537, IEEE, 2019.
- [19] D. M. Wilkes, K. Kawamura, a. B. Koku, R. A. Peters li, a. Sekmen, D. M. Wilkes, R. A. Peters, and a. Sekmen, "Toward egocentric navigation," *International Journal of Robotics and Automation*, vol. 17, no. 4, p. 135–145, 2002.

- [20] E. Karami, S. Prasad, M. Shehata, and S. M. Karami Ebrahim, Prasad Siva, “Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison of Distorted Images,” vol. 2017, pp. 1–18, 2017.
- [21] H. Fu, D. Xie, R. Zhong, Y. Wu, and Q. Wu, “An improved ASIFT algorithm for indoor panorama image matching,” *Ninth International Conference on Digital Image Processing (ICDIP 2017)*, vol. 10420, p. 104201C, 7 2017.
- [22] H. Fu, D. Xie, R. Zhong, Y. Wu, and Q. Wu, “An improved ASIFT algorithm for indoor panorama image matching,” in *Ninth International Conference on Digital Image Processing (ICDIP 2017)*, vol. 10420, p. 104201C, SPIE, 7 2017.
- [23] H. Taira, Y. Inoue, A. Torii, and M. Okutomi, “Robust Feature Matching for Distorted Projection by Spherical Cameras,” *IPSJ Transactions on Computer Vision and Applications*, vol. 7, no. 0, pp. 84–88, 2015.
- [24] J. Cruz-Mota, I. Bogdanova, B. Paquier, M. Bierlaire, and J. P. Thiran, “Scale invariant feature transform on the sphere: Theory and applications,” *International Journal of Computer Vision*, vol. 98, pp. 217–241, 6 2012.
- [25] D. G. Lowe, “Distinctive Image Features from Scale Invariant Keypoints,” *International Journal of Computer Vision*, pp. 1–28, 2004.
- [26] G. Tong, X. Chen, and N. Ye, “A spherical model based keypoint descriptor and matching algorithm for omnidirectional images,” *Advances in Mechanical Engineering*, vol. 2014, 2014.
- [27] M. Lourenco, J. P. Barreto, and F. Vasconcelos, “SRD-SIFT: Keypoint detection and matching in images with radial distortion,” *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 752–760, 2012.
- [28] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, “SuperGlue: Learning Feature Matching with Graph Neural Networks,” 11 2019.
- [29] D. Detone, T. Malisiewicz, and A. Rabinovich, “SuperPoint: Self-Supervised Interest Point Detection and Description,” *Cvpr2018*, vol. 2018-June, pp. 337–349, 2018.

- [30] B. Kuipers and Y.-T. Byun, “A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations,” *Robotics and autonomous systems*, vol. 8, no. 1-2, pp. 47–63, 1991.
- [31] M. Meng and A. C. Kak, “Neuro-nav: a neural network based architecture for vision-guided mobile robot navigation using non-metrical models of the environment,” in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 750–757, IEEE, 1993.
- [32] J. Pan, D. Pack, A. Kosaka, and A. Kak, “Fuzzy-nav: A vision-based robot navigation architecture using fuzzy inference for uncertainty-reasoning,” in *Procs. of the World Congress on Neural Networks*, pp. 602–607, 1995.
- [33] M. Hashima, F. Hasegawa, S. Kanda, T. Maruyama, and T. Uchiyama, “Localization and obstacle detection for robots for carrying food trays,” in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS’97*, vol. 1, pp. 345–351, IEEE, 1997.
- [34] T. Goedemé, M. Nuttin, T. Tuytelaars, and L. Van Gool, “Omnidirectional vision based topological navigation,” *International Journal of Computer Vision*, vol. 74, no. 3, pp. 219–236, 2007.
- [35] “Navigation.” =<https://dictionary.cambridge.org/dictionary/english/navigation>. Accessed: 2021-12-20.
- [36] C. Harris, M. Stephens, *et al.*, “A combined corner and edge detector,” in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988.
- [37] T. Lindeberg, “Scale-space theory: A basic tool for analyzing structures at different scales,” *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994.
- [38] “Open spherical camera api.” , <https://developers.google.com/streetview/open-spherical-camera>, Sep 1997. Accessed: 2021-12-22.
- [39] S. Sumikura, M. Shibuya, and K. Sakurada, “OpenVSLAM,” in *Proceedings of the 27th ACM International Conference on Multimedia*, ACM, Oct. 2019.

- [40] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “MonoSLAM: Real-time single camera SLAM,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.
- [41] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “Orb-slam: A versatile and accurate monocular slam system,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.
- [42] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [43] “Topological navigation.” , http://users.metu.edu.tr/kbugra/research/topological_navigation/, Dec 2021. Accessed: 2021-12-27.



Appendix A. Custom Code

"""

Created on Mon Jun 14 20:54:48 2021

All of the custom code written during the thesis is uploaded to [43].

@author: ygtshn

"""

```
1  def initialize_matching(config_in):
2      """
3      Initialize SuperGlue for scene matching.
4
5      SuperGlue requires weights, number of sinkhorn iterations
6      and matching threshold to initialize.
7
8      Superpoint feature detector requires nonmaximum suppression radius, keypoint
9      threshold and maximum number of keypoints for inialization.
10
11     Parameters
12     -----
13     config_in : list
14         Parameter set that is required by the SuperGlue and SuperPoint.
15
16     Returns
17     -----
18     torch.nn.Module
19         Scene matching front-end.
20     """
21  def initialize_keypoint_extractor(config_in):
22      """
23      Initialize Superpoint feature detector.
24
25      Superpoint feature detector requires nonmaximum suppression radius, keypoint
26      threshold and maximum number of keypoints for inialization.
```

```

27
28     Parameters
29     -----
30     config_in : list
31         Parameter set that is required by the SuperPoint network.
32
33
34
35     Returns
36     -----
37     torch.nn.Module
38         Keypoint extractor front-end.
39     """
40 def get_keypoint_latlon(latlon_pix,matching_results):
41     """
42     Extract spherical coordinates of keypoints.
43
44     Pixel locations of matching keypoints in the perspective image can be
45     projected back on the a sphere. This projection allows for extraction of
46     spherical coordinates of matching keypoints.
47
48     Parameters
49     -----
50     latlon_pix : numpy.ndarray
51         Coordinates of the pixels of the perspective image on the source
52         panorama.
53     matching_results : numpy.ndarray
54         Locations of the matching keypoints on the perspective image.
55
56     Returns
57     -----
58     numpy.ndarray
59         Spherical coordinates of the matching keypoints.
60     """
61 def get_keypoint_unitvector(keypoints):
62     """
63     Generate unit vectors of keypoints.
64
65     Rectangular decomposition of the latitude of the keypoint on a unit circle
66     after discarding latitude results in its unit vector.

```

```

67
68     Parameters
69     -----
70     keypoints : numpy.ndarray
71         Keypoint array that comprises latitude and longitude.
72
73
74
75     Returns
76     -----
77     numpy.ndarray
78         Unit vectors that are drawn to each of the keypoints.
79     """
80 def get_unit_heading(u_ci,u_cj,u_ti,u_tj,headingThreshold):
81     """
82     Compute unit heading vector.
83
84     Angular difference of each keypoint pair between first and second scene is
85     used to compute unit heading vector.
86
87     Parameters
88     -----
89     u_ci : numpy.ndarray
90         Unit vector of first element of keypoint pair in the first image.
91     u_cj : numpy.ndarray
92         Unit vector of second element of keypoint pair in the first image.
93     u_ti : numpy.ndarray
94         Unit vector of first element of keypoint pair in the second image.
95     u_tj : numpy.ndarray
96         Unit vector of second element of keypoint pair in the second image.
97     headingThreshold : numpy.ndarray
98         Threshold value that the angular difference of the selected keypoint
99         pair must be larger than so that the pair is accepted in heading
100         computation.
101
102     Returns
103     -----
104     numpy.ndarray
105         Unit heading vector that is computed for input keypoint pair.
106     """

```

```

107 def get_equ2pers(img_path,theta):
108     """
109     Render 2D image for given camera parameters.
110
111     Image rectification that is applied on source equirectangular panorama with
112     a given camera properties.
113
114
115     Parameters
116     -----
117     img_path : str
118         Path of the source panoramic image.
119     theta : float
120         Pan angle that the camera should be rendered at.
121
122     Returns
123     -----
124     img_pers : numpy.ndarray
125         Resulting perspective image.
126     latlon_pix : numpy.ndarray
127         Source addresses of the pixels of perspective image on the source image.
128     img_lon : numpy.ndarray
129         Longitude values of pixels of perspective image.
130     img_lat : float
131         Latitude values of pixels of perspective image.
132     """
133 def perform_reactive_navigation(node1,node2,headingThreshold,plot_flag=False):
134     """
135     Actuate reactive navigation.
136
137     Reactive navigation requires 2 nodes that are going to be
138     traversed in between.
139
140     Parameters
141     -----
142     node1 : str
143         Relative path of the first node.
144     node2 : str
145         Relative path of the second node.
146     headingThreshold : numpy.ndarray

```



```

147         Threshold value that the angular difference of the selected keypoint
148         pair must be larger than so that the pair is accepted in heading
149         computation.
150     plot_flag : (bool, optional)
151         Flag to enable or disable plotting action for the matching process.
152         Defaults to False.
153
154
155     Returns
156     -----
157     heading1 : numpy.ndarray
158         Resulting heading vector.
159     mconf_sum : numpy.ndarray
160         Sum of the average confidence scores after each matching process.
161     xNorm : numpy.ndarray
162         Sum of the normalized x-components after rectangular decomposition.
163     match_summary : dict{'img1':int,'img2','num_match': float}
164         Matching process summary.
165     """
166     def plot_keypoint_polar_projection(kpts_latlon, renk, newflag=False, radius=1.0):
167         """
168         Plot keypoints in polar projection.
169
170         Parameters
171         -----
172         kpts_latlon : numpy.ndarray
173             Unit vectors that are drawn to the keypoints.
174         renk : str
175             Color specification for the plot.
176         newflag : (bool, optional)
177             Flag to plot in a new figure. Defaults to False.
178         radius : (float, optional)
179             Radius of the resulting polar plot. Defaults to 1.0.
180         """
181     def plot_keypoint_cartesian_projection(kpts_units, newflag=False,
182                                         plot_heading=False, col='b'):
183         """
184         Plot keypoints in polar projection.
185
186         Parameters

```

```

187 -----
188 kpts_units : numpy.ndarray
189     Unit vectors that are drawn to the keypoints.
190 newflag : (bool, optional)
191     Flag to plot in a new figure. Defaults to False.
192 plot_heading : (bool, optional)
193     Flag for plotting unit vectors together with their locations.
194     Defaults to False.
195 col : (str, optional)
196     Color specification for the plot. Defaults to 'b'.
197 """
198 def match_images(img_input, img_target, resize_dim, matching,
199                 kpts_dict={},
200                 plot_flag = False,
201                 fname = 'imgSave.png'):
202     """
203     Conduct matching operation
204
205     Matching process takes place by resizing images to SuperGlue
206     default size then are transferred to the GPU.
207
208     Parameters
209     -----
210     img_input : str
211         Source path of the first perspective image.
212     img_target : str
213         Source path of the second perspective image.
214     resize_dim : list
215         Output size to which are the images resized.
216     matching : torch.nn.Module
217         Image matching front-end.
218     kpts_dict : (dict{float:numpy.ndarray}, optional)
219         Processed image dictionary. Defaults to {}. If not default, the
220         function looks for keypoints and descriptors in this parameter.
221     plot_flag : (bool, optional)
222         Visualization flag for the matching process. Defaults to False.
223     fname : (str, optional)
224         Filename to save visualization results. Defaults to 'imgSave.png'.
225
226     Returns

```

```

227     -----
228     mkpts0 : numpy.ndarray
229         Locations of the matching keypoints on the first image.
230     mkpts1 : numpy.nd.array
231         Locations of the matching keypoints on the second image.
232     mconf : numpy.ndarray
233         Confidence scores of matching keypoints.
234     matches : numpy.ndarray
235         Matching correspondences between 2 images.
236     desc0 : numpy.ndarray
237         Descriptors of the first image.
238     desc1 : numpy.ndarray
239         Descriptors of the second image.
240     """
241     def get_heading_new(keypoint0_unitvectors, keypoint1_unitvectors, headingThreshold) :
242         """
243         Compute heading vector to track.
244
245         Reactive navigation requires a heading vector to follow
246         so that it can alter its angular state of the landmarks.
247
248         Parameters
249         -----
250         keypoint0_unitvectors : numpy.ndarray
251             Unit vector array of matching keypoints in the first scene.
252         keypoint1_unitvectors : numpy.ndarray
253             Unit vector array of matching keypoints in the second scene.
254         headingThreshold : numpy.ndarray
255             Threshold value that the angular difference of the selected keypoint
256             pair must be larger than so that the pair is accepted in heading
257             computation.
258
259         Returns
260         -----
261         heading : float
262             Resulting heading vector.
263         heading_xNorm : numpy.ndarray
264             Sum of the x-component of the unit heading vectors.
265         heading_yNorm : numpy.ndarray
266             Sum of the y-component of the unit heading vectors.

```

```

267
268     """
269
270 def extract_keypoints(img_input,resize_dim):
271     """
272     Extract features of the input image.
273
274     Calling feature extractor front-end to obtain
275     keypoint and generate descriptors.
276
277     Parameters
278     -----
279     img_input : str
280         Path to the input image.
281     resize_dim : list
282         Image dimensions that are used for resizing.
283
284     Returns
285     -----
286     kpts0 : numpy.ndarray
287         Extracted features.
288     desc0 : numpy.ndarray
289         Associated descriptors.
290     scores0 : numpy.float
291         Keypoint scores.
292     """
293 def find_nearest_node(map,node):
294     """
295     Locate the nearest map node to the queried node.
296
297     Conducting a search over the entire map to find
298     the node with the highest similarity.
299
300     Parameters
301     -----
302     map : ygt_Graph.Graph
303         Topological map of the environment that is generated during mapping
304         phase.
305     node : ygt_Graph.Node
306         Query node of which the closest node is to be searched for.

```

```

307
308     Returns
309     -----
310     closest_id : int
311         Node id of the closest node.
312     second_closest : int
313         Node id of the second closest node.
314     similarity : float
315         Similarity score of the closest node.
316     similarity_previous : float
317         Similarity score of the second closest node.
318     score_list : list
319         List of similarity values that is computed during the search.
320     """
321     def extract_ORB_features_and_match(node1,node2,matching=True,
322                                       preloaded_images=False,
323                                       save_flag=False):
324     """
325     Extract ORB features and match using FLANN.
326
327     Image processing is done through ORB features to
328     accelerate the map operations.
329
330     Parameters
331     -----
332     node1 : dict{int:numpy.ndarray}
333         Image dictionary that belongs to the first image.
334     node2 : dict{int:numpy.ndarray}
335         Image dictionary that belongs to the second image.
336     matching : (bool, optional)
337         Flag for the matching operation. If set to False the function just
338         extracts the ORB features and descriptors of the "node1".
339         Defaults to True.
340     preloaded_images : (bool, optional)
341         Flag for readily loaded images. If set to True the function looks for
342         the keypoints and descriptors in the "node1" & "node2".
343         Defaults to False.
344     save_flag : (bool, optional)
345         Flag for saving "cv2.keypoint" objects. If set to True keypoints are
346         processed since "cv2.keypoint" instances cannot be pickled.

```

```

347         Defaults to False.
348
349     Returns
350     -----
351     dict{str:numpy.ndarray}
352         Dictionary of keypoints, descriptors. Matching results are included if
353         "matching" is True.
354     """
355
356
357 def rotate_equirect_image(img, x=0, y=0, z=0):
358     """
359     Rotate input image to apply artificial rotations.
360
361     Equirectangular panoramic images allow the user to apply
362     artificial rotations and rendering on the scene because they have
363     spherical field of view.
364
365     Parameters
366     -----
367     img : numpy.ndarray
368         Equirectangular panoramic source image.
369     x : (int, optional)
370         Rotation amount about x-axis. Defaults to 0.
371     y : (int, optional)
372         Rotation amount about y-axis. Defaults to 0.
373     z : (int, optional)
374         Rotation amount about z-axis. Defaults to 0.
375     """
376
377 def recover_CV2_Keypoints(pickled_keypoints):
378     """
379     Recreate "cv2.keypoint" keypoints.
380
381     To be able to use openCV API with the saved image dictionaries
382     keypoints must be an instance of cv2.keypoint class.
383
384     Parameters
385     -----
386     pickled_keypoints : list
387         List of tuples. Each tuple contains cv2.keypoint attributes.

```

```

387
388     Returns
389     -----
390     list
391         List of keypoints that are in the cv2.keypoint format.
392     """
393
394
395 def process_image(img_name, save_flag) :
396     """
397     Generate dictionary for source image.
398
399     Processing an image generates a dictionary in which
400     its perspective images for the pan angles, spherical
401     coordinates of the detected keypoints, local keypoints
402     addresses and their descriptors are stored.
403
404     Parameters
405     -----
406     img_name : str
407         Path to the source image.
408     save_flag : bool
409         Saving flag to process cv2.keypoint class since they cannot be pickled.
410
411     Returns
412     -----
413     thetabase : dict{int:numpy.ndarray}
414         Dictionary of the processed image.
415     lat : numpy.ndarray
416         Latitudes of the pixels of corresponding perspective image.
417     lon : numpy.ndarray
418         Longitudes of the pixels of corresponding perspective image.
419     """
420 def compute_similarity(node1, node2) :
421     """
422     Compute similarity between images.
423
424     Similarity score between images is computed in the form of a
425     similarity matrix whose each element corresponds to individual
426     similarity scores of each matching attempt.

```

```

427
428     Parameters
429     -----
430     node1 : ygt_Graph.Node
431         First image in the form of a Node object. Contains processed image
432         properties.
433     node2 : ygt_Graph.Node
434         Second image in the form of a Node object. Contains processed image
435         properties.
436
437     Returns
438     -----
439     float
440         Overall matching score between images.
441     """
442 def capture_image(img_name):
443     """
444     Capture an image over the wifi.
445
446     Parameters:
447     -----
448     img_name : str
449         Image name that will be assigned to the capture image.
450     """
451 class Graph(object):
452     """
453     Graph data structure, undirected by default.
454
455     ...
456
457     Attributes
458     -----
459     _graph : defaultdict
460         Connectivity information of the Graph.
461     _directed : (bool, optional)
462         Direction flag for bi or unidirectional Graph
463         Defaults to False.
464     nodes : list
465         Node list of the Graph.
466

```



```

467     Methods
468     -----
469     add_connections(connections):
470         Adds connections in the "connection" to the Graph.
471     add_node(node):
472         Adds node to the Graph without forming a connection.
473     add(node1, node2):
474         Adds connection between node1 and node2.
475     remove(node):
476         Removes "node" and all its references.
477     is_connected(node1, node2):
478         Checks the "node1" and "node2" for connectivity.
479     find_path(node1, node2, path=[]):
480         Searches for a path between "node1" and "node2".
481
482     """
483     class Node(object):
484         """
485         Node data structure.
486
487         ...
488
489     Attributes
490     -----
491     descriptor : dict
492         Dictionary of processed image that becomes the Node.
493     occurence_table : list
494         List of occurence information.
495     name : str
496         Name of the Node.
497
498     Methods
499     -----
500     add_occurence(node):
501         Updates occurence information of the Node when visible from "node".
502
503     """

```



Appendix B. Example Use of the Library

```
1  import ys_ThesisLib as ys
2  from ygt_Graph import Node
3  import os
4  # SUPERGLUE & SUPERPOINT INITIALIZATION PARAMETERS
5  KEYPOINT_THRESHOLD_ARR = np.arange(0.05,0.4,0.905)
6  MATCHING_THRESHOLD_ARR = np.arange(0.8,1,0.905)
7
8  headingThreshold_arr = np.arange(0,1)
9  for matching_thresh in MATCHING_THRESHOLD_ARR:
10     for keypoint_thresh in KEYPOINT_THRESHOLD_ARR:
11         for headingThreshold in headingThreshold_arr:
12             NMS_RADIUS = 4
13             MAX_KEYPOINTS = 1024
14             SINKHORN_ITERATIONS = 20
15             config = [NMS_RADIUS, keypoint_thresh, MAX_KEYPOINTS,
16                     SINKHORN_ITERATIONS, matching_thresh]
17
18     # Assume that the Topological Map "TopMap" is given and the laptop is
19     # connected to the camera
20     ys.capture_image('example_capture.jpg') # 'example_capture.jpg' is the
21                                             # raw equirectangular panorama.
22     ys.rotate_equirect_image('example_capture.jpg',x=30,y=30,z=45)
23     example_img_dict = ys.process_image(os.getcwd()+r'\example_capture.jpg',
24                                       save_flag = True)
25     example_node = Node('example_capture.jpg',example_img_dict,[])
26     control_node = TopMap.nodes[0] # control_node is the Node type object
27     similarity = compute_similarity(example_node,control_node)
28     closest_node, second_closest, similarity, similarity_previous, score_list=
29                                     ys.find_nearest_node(TopMap,example_node)
30     heading1,xNorm,yNorm,match_summary = ys.perform_reactive_navigation(
31                                     'example_capture.jpg',control_node.name,
32                                     0,False)
33     img_perspective,img_latlon_pix,_,_ = get_equ2pers('example_capture.jpg',90)
34     dummy_matcher = ys.initialize_matching(config)
```

```

35 mkpts0,mkpts1,mconf,matches,desc0,desc1=ys.match_images(
36         img_perspective,control_node.descriptor[0]['img'],
37         [640,480],dummy_matcher,{},plot_flag = False,'imgSave.png')
38
39 img_keypoint_latlon = ys.get_keypoint_latlon(img_latlon_pix,mkpts0)
40 img_unit_keypoints = ys.get_keypoint_unitvector(img_keypoint_latlon)
41 heading1,xNorm,yNorm = get_heading_new(
42         img_unit_keypoints,img_unit_keypoints,headingThreshold)

```

