TOPOLOGICAL NAVIGATION ALGORITHM DESIGN AND ANALYSIS USING SPHERICAL IMAGES

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

BY

YASİN ŞAHİN

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

AUGUST 2022

Approval of the thesis:

TOPOLOGICAL NAVIGATION ALGORITHM DESIGN AND ANALYSIS USING SPHERICAL IMAGES

submitted by **YASİN 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 Kalıpçılar _____
Director, Graduate School of **Natural and Applied Sciences**

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

Assoc. Prof. Dr. Ahmet Buğra Koku _____
Supervisor, **Mechanical Engineering, METU**

Examining Committee Members:

Assoc. Prof. Dr. Ali Emre Turgut
Mechanical Engineering Department, METU _____

Assoc. Prof. Dr. Ahmet Buğra Koku
Mechanical Engineering Department, METU _____

Assoc. Prof. Dr. Erol Şahin
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Gökhan Bayar
Mechanical Engineering Department, Bulent Ecevit University _____

Assist. Prof. Dr. Hakan Çalışkan
Mechanical Engineering Department, METU _____

Date: 23.08.2022

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, Last Name: YASİN ŞAHİN

Signature :

ABSTRACT

TOPOLOGICAL NAVIGATION ALGORITHM DESIGN AND ANALYSIS USING SPHERICAL IMAGES

Şahin, Yasin
Master of Science, Mechanical Engineering
Supervisor: Assoc. Prof. Dr. Ahmet Buğra Koku

August 2022, 169 pages

A topological navigation algorithm that has the capability of mapping and localization based on visual contents is proposed. Keypoint detection and feature matching are conducted on spherical images to extract significant features among sequential frames. Robot movement direction is estimated based on historical angle differences of significant features to reach the final destination. The navigation process is supported with visual egocentric localization to gain simultaneous localization and mapping competence. The algorithm is tested in different scenarios in which accurate results are obtained in terms of autonomous navigation objectives. Development opportunities are examined to criticize possible future steps.

Keywords: Topological Navigation, Qualitative Navigation, Spherical Image, Equirectangular Image, Keypoint Detection, Feature Matching, Landmark Detection

ÖZ

KÜRESEL GÖRÜNTÜLER KULLANILARAK TOPOLOJİK SEYRÜSEFER ÇÖZÜM YOLU TASARIMI VE ANALİZİ

Şahin, Yasin
Yüksek Lisans, Makina Mühendisliği
Tez Danışmanı: Doç. Dr. Ahmet Buğra Koku

Ağustos 2022, 169 sayfa

Görsel içeriklere dayalı olarak haritalama ve lokalizasyon yapabilen bir topolojik seyrüsefer çözüm yolu önerilmiştir. Sıralı görüntüler arasında önemli öznitelikleri çıkarmak için küresel görüntüler üzerinde ana nokta tespiti ve öznitelik eşleştirmesi yapılmıştır. Robot hareket yönü, nihai hedefe ulaşmak için önemli özniteliklerin tarihsel açı farkına dayalı olarak kestirilmiştir. Seyrüsefer süreci, görsel benmerkezci lokalizasyon ile desteklenerek eş zamanlı lokalizasyon ve haritalama yetkinliği kazandırılmıştır. Çözüm yolu, otonom seyrüsefer hedefleri açısından başarılı sonuçların elde edildiği farklı mekan senaryolarında test edilmiştir. Gelecekteki olası adımları tanımlamak için geliştirme fırsatları incelenmiştir.

Anahtar Kelimeler: Topolojik Seyrüsefer, Nitel Seyrüsefer, Küresel Görüntü, Eşdikdörtgensel Görüntü, Ana Nokta Eşleştirme, Öznitelik Eşleştirme

To Eda, my precious partner in crime

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to my supervisor, Assoc. Prof. Dr. Ahmet Buğra Koku for his continuous support, and constructive feedback whenever I need it. I was able to change my point of view to see the big picture with his patient guidance and deep insight.

I would like to thank my friends and professors for helping me to broaden my perspective during my studies.

My thanks also goes to my lovely parents Yaşar and Firdevs who were always supporting me throughout my life, my big brother Yakup, who was consistently advising me to push my limits, and my sister-in-law Oya, who was mentioning about the positive effects. Finally, I would like to express my deepest thanks to my wife Eda for making my life joyful, always believing in me, and motivating me to see the light at the end of the tunnel. I could not be finalizing this thesis without her encouragement and special smile.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	Two dimensional
AHAE	Absolute Heading Angle Error
AUC	Area Under Curve
CPU	Central Processing Unit
DCPE	Distance Covered Percent Error
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
IGS	International GNSS Service
IMU	Inertial Measurement Unit
mAP	Mean Average Precision
RANSAC	Random Sample Consensus
SLAM	Simultaneous Localization and Mapping
STD	Standard Deviation
vSLAM	Vision-based Simultaneous Localization and Mapping

CHAPTER 1

INTRODUCTION

This chapter describes the overall extent of the thesis. In other words, the problem is defined explicitly to explain possible improvement suggestions. It can be classified as a topological navigation problem in its simplest form. Moreover, spherical cameras are used as the only sensor, therefore, a new aspect is introduced to combine with the navigation. In the final form, the primary purpose is to develop an algorithm that is eligible to achieve the navigation process with the help of spherical camera output.

1.1 Introduction

Mobile robots' operational environments are separated as indoor and outdoor. They achieve their objectives in three main parts namely navigation, mapping, and localization independent of the environment they are working in. The first part, navigation, is stated as the science of finding a way from one place to another in [69]. In literature, the most popularly applied type of navigation is metric navigation which relies on accurate metric mapping of free space. They represent geometrical relationships among objects with the help of location representations such as cartesian environment representation. After the start of public usage of the International GNSS Service (IGS) [2], metric map construction became much simpler and easier. However, GNSS needs a direct line of sight to the satellites [52]. Therefore, it is not useful in indoor or narrow outdoor spaces. In addition, metric maps have two major shortcomings: they do not scale well with the amount of experience, and actuation noise on real robots makes it challenging to build consistent presentations [6]. In other

words, they do not take advantage of previous experience and are highly dependent on precise and noiseless location readings. To deal with noise and position errors, grid-based metric mapping is introduced and applied in [19, 20, 49]. However, it increased the complexity of map construction, hence, is increasing the computation time significantly. On the contrary, it is revealed by physiological studies that spatial memory, where free space object locations and their relative qualitative information are recorded without having any metric knowledge, is the primary navigation source of animals [23, 12]. This approach also shows how humans behave to navigate themselves [24, 74, 71]. Imagine that you are at a new university campus entrance, and your task is to find the gymnasium. Once you reach the final destination, assume that your new goal is going backward to the entrance. What would you do? The first but not realistic option would be the metric maps where you need to know the exact steps that you took while coming from the entrance to the gymnasium. The second option would be following the significant surrounding places, which you recognized before reaching the gymnasium, in reverse order. The latter one is called topological mapping. In topological mapping, significant surrounding places are defined as natural landmarks that construct nodes of the map and are connected to each other with crossable paths. At the end of the process, a map that has the form of a graph will be created. Graph-like structure makes topological mapping more compact and less computationally expensive than metric maps. There are different approaches stated for topological mapping in [35, 36, 55] to even lower computational time and increase efficient map building. Moreover, topological maps are robust against position errors and can recover easily from drift and slippage [66]. Since the position is not crucial in topological map building, recognizing a node would be satisfactory to recover from an uncertain condition and to locate the robot on the map. However, uncertain conditions in the navigation process do not just occur due to sensor-related issues. The environment, in which the robot operates the navigation process, may be affected by dynamic changes such as illumination and moving objects. It is even possible to face an unmapped, i.e. unknown, environment so that the robot should also be able to gain the skill of navigation in those situations. The first solution to the problem with the Simultaneous mapping and navigation (SLAM) approach is explored in [37]. With the application of SLAM in an off-road robot race called DARPA Grand challenge [4], it started gaining widespread attention due to its significant performance. Ac-

cording to [15], the "Holy Grail" of the autonomous vehicle research community is declared as SLAM. It is defined as the capability of an autonomous vehicle to place itself in an unknown environment, to map the unknown environment with respect to relative free space information, and achieve a synchronous navigation process with the unknown environment map. In the early stages of SLAM, various distance measuring sensors such as laser scanners, sonars, or GPS are widely used. However, due to challenges stated for metric mapping earlier in this section and in [66], the popularity of cameras at the implementation of SLAM gained a lot of attention. Recently, applications in which a camera is used as the primary and the only sensor have been published extensively. This is due to the public availability of wide field-of-view and high-resolution cameras, and their increased effect on mapping, and therefore, SLAM algorithm performance. In the case of having a camera as the only sensor in a SLAM application, the case is stated as a new branch of SLAM and is called vision-based SLAM or in short vSLAM. Throughout the years, as camera technology developed, different vSLAM technologies were proposed in the literature to utilize necessary environmental details. Although the most commonly used camera type is monocular cameras, spherical cameras are still gaining demand because of their 360° view of the environment. Since they provide a generous field of view, feature finding and tracking become easier due to their long stay in the camera's line of sight.

Vision-based navigation algorithm development is the primary objective of the thesis. It has been proposed that information taken from spherical images is sufficient to describe the environment with only topological relationships. Therefore, equirectangular images obtained with the help of a spherical camera are the only input of the algorithm to obtain mandatory data. To conclude, the developed algorithm is classified as vision-based in terms of sensor input, topological in terms of mapping and navigation, and natural in terms of environment interaction. Hence, there are three categories this thesis includes at the same time. It is a study of vision-based topological natural navigation.

1.2 Motivation and Problem Statement

Visual odometry which estimates the ego-motion of the agent is one of the major topics of vision-based SLAM applications. It is divided into two parts namely direct and indirect. In the indirect method, image features are extracted from the image and they are tracked as applied in [53, 26, 70]. On the contrary, in the direct method, instead of extracting features from the image, optical flow on the image is traced explicitly as in [21]. Global map optimization is another major topic of vision-based SLAM applications. It is mainly based on map construction of the environment and suppression of errors, which occur during camera movement, to maintain consistency of the entire map. One technique extensively used in global map optimization is loop closing which tries to match the currently taken image with one of the map images. Detection of a loop indicates revisiting of a node in the map, and it allows to diminish camera movement errors. However, since most of the visual-based SLAM algorithms use monocular cameras as the only sensor [50, 11, 43], it provides a small field of view compared with spherical cameras. Therefore, they lack capturing features outside the field of view of the camera. Uncertainties, which occur due to environment and sensor effects, are compensated with the help of additional cameras installed to increase the field of view or round off perfect panorama as stated in [78]. Addition to this, cameras are fused with various distance measuring sensors such as IMU [39], GPS [60] or lidar [32]. Even if the fusion of distance measurement would help the navigation process, it affects the complexity, and the performance of the algorithm when there is no accurate location estimation. With the increased popularity and availability of spherical cameras, features close to the robot would be able to detect with a 360° field of view. With spherical images taken by spherical cameras, we suggest that

- a robust localization and loop detection can be made,
- a 360° visual odometry can be achieved,
- the need for the additional sensor can be beaten,
- a graph-like topological map can be constructed.

Although the algorithm lack metric information, the drawback will be overcome with

the rich visual content of a spherical camera. Moreover, building and maintaining the topological map and egocentric location estimation according to robot movements becomes simpler as a result of the wide field of view of a spherical camera. Besides, using a single sensor will be boosting the algorithm in terms of computational efficiency.

1.3 Proposed Methods and Models

As the first step, the algorithm is divided into three main parts; localization and path planning, mapping and exploration, and topological navigation. Topological mapping, which is applied for the mapping and exploration process, is inspired by its existence in nature. Further, it reduced the need for additional sensors and boosted computational efficiency. As stated in [23, 12], metric information is not necessary when animals navigate themselves concerning free space locations and their qualitative relative relations. Similar behavior also holds for humans [74]. Since metric information is not required in nature, distance measuring sensors are not used in the algorithm development process. Therefore, needed sensors are reduced significantly, and only a spherical camera is used. Moreover, the developed algorithm is less computationally demanding compared to metric maps due to its flexibility in accurate robot location estimation [66]. In the localization part, indirect visual odometry based on extracted equirectangular image features is used to take advantage of the 360° field of view of a spherical camera. During the mapping and exploration phase, equirectangular images captured by the camera are defined as nodes of the map. The keypoint properties, such as keypoint pixel coordinates and descriptors, along with relative locations of the nodes concerning surrounding nodes are saved for each node. When an image is taken, its keypoint properties are compared with each map node to find out the most similar map node. If the current image features do not match with any nodes of the map, the current image is stated as a new node. After each comparison of the current image features with a map node, a connection, i.e. an edge, is created if the number of matched keypoint pairs is above a certain threshold.

In the navigation process, the robot is located with the help of the relative location of the current image with map nodes. Also, it is assumed that keypoint pairs of an

image can be re-generated along with the robot movement. Therefore, keypoint pairs and their angular differences with the robot movement can be followed to locate the robot among map nodes and to calculate the heading vector that allows specifying robot movement direction. Since relative locations and connections of map nodes are saved in the mapping and exploration phase, the robot path is generated according to the current state of the robot on the topological map. If the robot starts matching keypoint pairs with the target node, the robot movements are ended when the predefined matching limit is achieved.

1.4 Contribution and Novelties

Our contributions are as follows:

- An equirectangular image based topological mapping and navigation application,
- A robust and effortless visual SLAM algorithm,
- A novel visual odometry application.

1.5 The Outline of the Thesis

The rest of the thesis is organized as follows: Chapter 2 is dedicated to literature survey and problem formulation. Chapter 3 is focused on selection among algorithms which are mentioned in Chapter 2. Chapter 4 is committed to the implementation of selected algorithms. The simulation environment created for hands-on applications is expressed in Chapter 5. Results of conducted experiments and discussions of the results are published in Chapter 6. Ultimately, in the final chapter, Chapter 7, the conclusion is shaped and development opportunities are revealed to the reader for the proposed algorithm.
To conclude, SLAM framework topics are shortly explained to gain the attention of the reader regarding the general principles. Topics are investigated in further detail with respect to various branches to express the core idea behind each branch. In

addition, development opportunities are undertaken with a possible solution. At last, contributions are specified and the outline of the thesis is described to increase the awareness of the reader of the structure of the thesis.

CHAPTER 2

METHODOLOGY

Explaining the background and the necessary wording of the problem is a must before going into detail. In the implementation, data supplied by a spherical camera at a given instant is used to navigate the robot instead of using any memory-related information that depends on previous states. This type of approach helps the robot to gain reactive behavior. Since only a spherical camera is used as the measurement unit, there will be no metric information during any implementation. Therefore, both mapping and navigation processes acquire topological titles. Moreover, using sensor input from a given instant limits movement decisions with sensor range, hence, the term local is used to specify the instant sensor data implementation. This term is mostly used when two camera frames are compared with each other. To compensate for the coverage of environments that are beyond camera range and to navigate a robot in larger environments, from particular to general approach is applied, that is, environments are divided into smaller regions with the help of topological maps, in which sensor range limit is satisfied and reactive behavior is safely applied. The next sections can be followed easier if mentioned concepts are kept in mind.

2.1 Literature Survey

Vision-based navigation takes attention due to its simplicity and capability of extracting a high amount of data from the current scene. The first central principle of vision-based navigation is image processing. In the common approach, features are directly extracted from the current scene image and they are matched concerning

their descriptors at traditional methods applied for processing the image. The early start of this kind of approach is mentioned as Harris corner detector [29] in which only the feature extraction part is covered. Lowe gained popularity with SIFT [41] and introduced a scale-invariant algorithm that includes both extractor and descriptor parts. Different methods such as SURF [1], ORB [58], BRISK [38], and BEBLID [64] have been published to provide different approaches to the problem. With the advancement of deep learning methodologies in the last decade, instead of only extracting hand-crafted local features as mentioned earlier, features started to be created with end-to-end processes. In one of the first approaches called LIFT [75], features extraction and description are handled with the help of a convolutional neural network trained with small image patches. Balntas et al. published TFeat [72] in which shallow convolutional neural networks are used and the features extracted after the convolutional layers of the CNN are directly adopted to descriptor generation without training a specialized distance layer as applied in [61]. Tian et al. proposed a different approach called L2Net [67] and took the whole batch into account while creating descriptors with the help of a loss function. This approach gained a lot of attention, therefore, improved with additional methodologies as in HardNet [48], where a new loss function is adopted to L2Net architecture. With the adaptation of a graph matching algorithm developed based on second order similarity, a new methodology SOSNet [68] aimed to improve patch description by improving robustness due to shape distortion. Another approach, MagicPoint [14], takes the complete image as input and creates well-distributed and learning-based features extracted from the image. SuperPoint [13], a robust training approach is built upon MagicPoint and proposed to solve multi-view geometry problems by using complete images. However, local feature detector and descriptor algorithms are developed for stereo cameras in the primary scope. They may not be directly applicable to spherical images. The main reason is the distortion occurring when the spherical grid is unfolded to represent the image in planar view. Different methodologies such as learning spherical convolutional networks [62] which include trainable kernels to transform planar CNN to an equirectangular projection of a spherical image started to be published to eliminate the issue. Instead of converting planar CNN to spherical, Cohen et al. produced spherical CNNs [8], a straightforward solution to contribute to the necessary rotational invariance for convolutions on the sphere. Eder et al. [16, 17] introduced

the usage of icosahedrons to diminish the distortion problem to increase robustness in CNNs. Furthermore, they also mentioned a new spherical image representation called tangent images [18], where distortion is decreased with the help of icosahedrons. Zhao et al. also implemented icosahedrons in their work to create a feature extractor and descriptor SPHORB [77] to obtain robust binary features from spherical images. Guan et al. [27] tuned another descriptor algorithm BRISK into spherical images by adding a geodesic grid and triangular meshing methodology. Taira et al. [65] generated an n-rectified equirectangular image from the input equirectangular image and generated rectified image distortions are eliminated based on covered angular regions. SURF keypoints are generated and image patches are created around those points before mapping onto the tangent plane in [7]. Micusik et al. [46] divided a spherical image into several perspective images and then applied different planar detectors and descriptors. Hadj-Abdelkader et al. [28] introduced a spherical Harris corner detector to increase robustness on the feature detection part. The SIFT algorithm is also adapted to the spherical domain in [9] with the help of spherical harmonics. For the matching part, most public approaches depend on using the nearest neighbor search to find the closest matching descriptors between two images [5, 3]. To geometrically verify matches and to eliminate outliers, algorithms like random sample consensus (RANSAC) [22, 54], direct linear transform (DLT) [30] are widely used. Sarlin et al. come up with a joint and novel solution for both matching and outlier elimination problems with the help of graph neural networks in SuperGlue [59]. Since these matching methodologies still lack separated detector and descriptor algorithms, NCNet [57] is proposed to enable detector-free and end-to-end design. It is further boosted to be more efficient on sparse convolutions with Sparse NCNet [56]. Parallel to the design, possible accuracy increment is investigated with DualRC-Net [40]. LoFTR [63] offered another point of view to descriptor-free matching by using Transformers [73].

The second central principle of vision-based navigation is spatial relationship handling. Chaplot et al. [6] proposed a supervised neural network-based algorithm that maps semantic features as nodes and geometrically verifies their connections among each other. Accordingly, Meng et al. introduced NEURO-NAV system [44], where a neural network is used to extract features, perform matching, and decide on topological representation as a map. Neural network activations are handled by a rule-based

controller which receives commands from a human supervisor. By improving the rule-based controller to a fuzzy logic controller, Kak et al. published FUZZY-NAV [51]. Hu et al. mentioned the memory problem with the increased environment exploration duration in [33], and they proposed a self-organizing neural network-based structure to eliminate memory inefficiency for additive topological mapping and navigation. Hashima [31] leveraged from correlations between landmarks recorded in addition to robot movement. Zhao in [42] uses convolutional neural networks to find the most similar frames from the topological map and introduces a sharpness measure to eliminate blurry frames without processing.

Although studies that are mentioned cover topological mapping, navigation, and process of spherical images with distinct representations such as equirectangular and tangent images in terms of feature extraction and matching, a study of combined topological mapping and navigation with spherical images is not worked extensively. One study is published by Goedeme et al. [25], in which a topological map is constructed from images taken within equal intervals. They also introduced a new matching methodology called fast wide baseline matcher to perform faster matching. However, their setup consists of a single-lens omnidirectional camera, therefore, a fisheye lens is necessary to surround 360°. In [76], Huang et al. investigated hand-crafted local feature detector performance with spherical images for vision-based SLAM application. Even though their approach shows a consistent framework for vision-based SLAM, it can be investigated further for future detection and investigation. Hence, vision-based 360° SLAM operations are not studied excessively with spherical cameras, and there is still room for additional studies.

2.2 Problem Formulation

The approach that we are designing is built upon two major peripherals, namely topological navigation and topological map operations. Topological navigation is based on tracking an estimated heading vector which is calculated by comparing local features of two camera frames, and topological map operations are dedicated to creating a topological map in the mapping stage, locating the robot in the constructed topological map, and defining a path for executing tracking process and finalizing the

navigation after reaching the marked destination.

2.2.1 Topological Navigation

Navigation processes differ from each other in the purpose of applications. In applications such as robot-assisted surgeries, industrial automation, autonomous driving, and aerial robotics, where precision is seen as the primary concern, a map is constructed based on metric measurements, and the navigation process is implemented mainly with the help of metric sensors. However, psychological studies [23, 74] demonstrate that both humans and animals use their spatial memory, in other words, surrounding salient objects and their relative locations, as the essential navigation source regardless of any metric knowledge. In addition to spatial relationships, our approach is built upon reactive behavior. In detail, although a topological map is created during the mapping stage to improve the localization phase, reactive behavior does not leverage any global map operation. In reactive behavior, direction estimation is limited with the simultaneous sensor measurements, therefore, robot movement decisions are made in accordance with the current sensor status. To be able to succeed in this task, the most important finding is catching as similar local features as possible between two successive frames. This procedure let us expose intuitive relations such as reminiscence and perceptual similarity between scenes. With the help of rich local features extracted from a spherical camera at a specific instant, we expect to solve dependencies of reactive behavior which are detection of common features between two successive frames, and heading estimation leading through the target scene.

2.2.1.1 Local Feature Matching

In our applications, a feature can be defined as a sample that carries data about the specific content of a camera frame. In other words, a feature shows if a dedicated region of the frame has the desired attribute such as the event of highlighting edges in the current frame. Moreover, we refer to local feature matching as the matching process of local features extracted from two different camera frames. The first task of achieving local feature matching is detection. Harris corner detector [29] and Differ-

ence of Gaussian (DoG) [41] are two of the first methods that have been developed to cover the feature detector part. After features are detected in the camera frame, the next task is creating a vector for each detected feature to provide a common ground for the comparison of features with each other. To facilitate the common ground creation process algorithms called descriptors such as SIFT [41], one of the pioneers of descriptor algorithms, started to be developed. In time, to cover the lacking part of existing algorithms and to provide different approaches to the description problem, new algorithms like ORB [58] and BEBLID [64] are declared. As the popularity of deep learning has increased among researchers in the last decade, novel feature descriptor algorithms such as TFeat [72], HardNet [48], SOSNet [68] and SuperPoint [13] have been published to develop end-to-end processes. The final task of local feature matching is the matching of descriptors. The most popular approach to perform descriptor matching is Nearest Neighbor matching which couples descriptors by calculating the L2 distance among each matching candidate. Calculated distances are sorted out, and the candidates with the minimum distance are matched with each other. Recently, a deep learning algorithm SuperGlue [59] is developed with the help of graph neural networks to be able to construct matching by processing descriptors as a whole. Since heading estimation between the current camera frame and the target camera frame is dependent on local feature matching due to reactive behavior, robustness and accuracy of local feature matching are fundamental to completing the topological navigation process.

The primary topological navigation sensor we use, a spherical camera, represents an image in a spherical grid. Therefore, transforming a spherical grid into a planar grid in order to represent spherical images as equirectangular images result in distortion as shown in Figure 2.1. In our implementations, we first try to address how spherical distortion behaves through the whole image frame, and then we construct a distortion analysis to find out the best possible combination of spherical image representation, descriptor, and matcher.

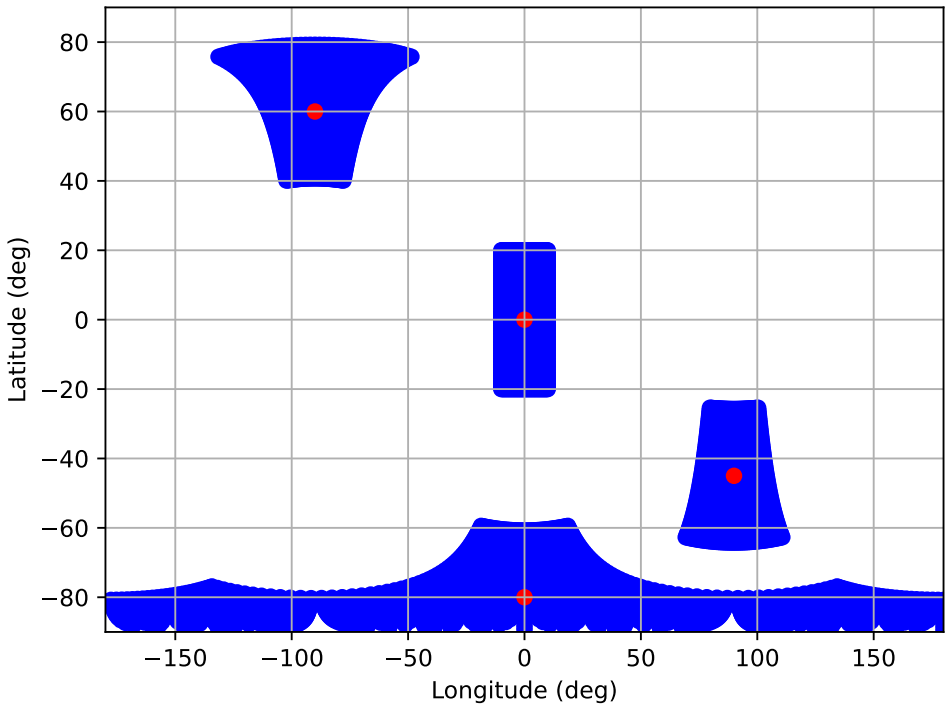
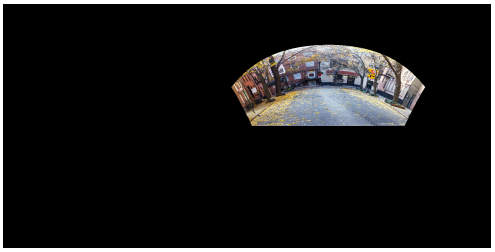


Figure 2.1: Spherical Distortion Change with Latitude & Longitude

2.2.1.2 Spherical Distortion Behavior

One of the most common approaches while projecting spherical images into planar images is equirectangular representation. However, due to changing metric distance between longitudes, equirectangular projection results in spherical distortion which is a function of pixel latitude. Since the image grid is affected by distortion, it directly influences detector and descriptor performance, especially descriptors based on convolutional neural networks as mentioned in [18], at local feature matching. In order to see how spherical distortion affects local feature performance, an image is placed at different center latitudes and center longitudes as in Figure 2.2. In order to investigate local feature matching performance, matching is performed between the image that is placed at different center latitudes and longitudes, and the image placed at the center of the spherical grid. It can be seen from Figure 2.3 that local feature matching is not affected by pixel longitudes. Therefore, we will be keeping the center longitude as zero at further distortion analyses.

(a) Center Latitude:$-60°$, Longitude:$30°$



(b) Center Latitude:$60°$, Longitude:$30°$

(c) Center Latitude:$0°$, Longitude:$30°$

(d) Center Latitude:$0°$, Longitude:$-30°$

Figure 2.2: Images Placed to Different Center Latitude & Longitude

In addition to equirectangular representation, different viewpoints have started to be developed to diminish the distortion. Eder et al. [18] proposed a methodology called tangent images to mitigate spherical distortion regardless of spherical representation by creating tangent planes. In their approach, they try to discover the optimal planar grid projection from a spherical grid by dividing the spherical grid into icosahedrons. The proposed tangent images approach is implemented directly to our use case to see how it affects local feature matching performance. Moreover, we expect to use equirectangular images with dimensions of 1024x2048, therefore, our hyperparameters base level and subdivision level are selected as 9 and 1, respectively as suggested. In the end, 80 tangent images with resolutions of 256x256 are generated from an equirectangular image, feature detection and description processes are applied for each tangent image, and results are merged to represent the whole equirectangular image. As a final task, descriptor matching is performed to merge descriptors that represent the equirectangular image as a whole. Entire process is also explained in Figure 2.4.

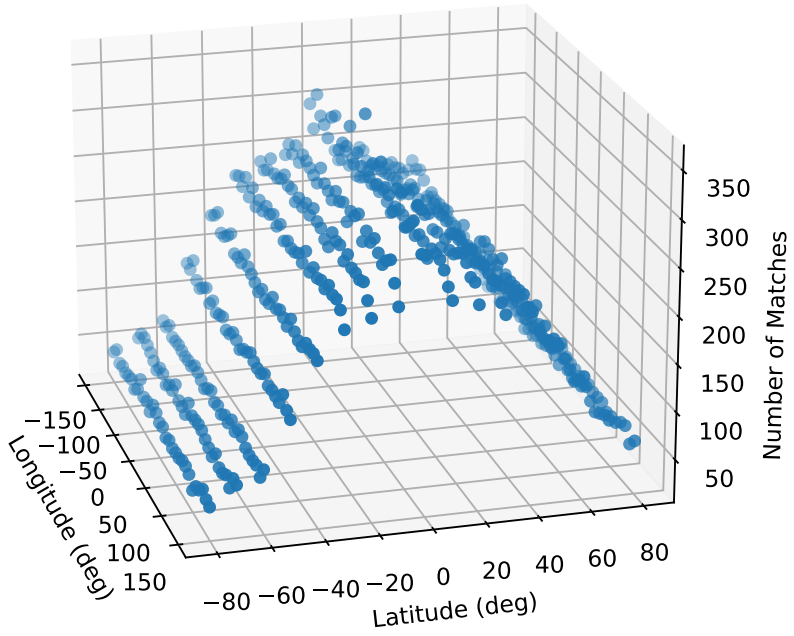


Figure 2.3: Local Feature Matching Performance Change with Latitude & Longitude

2.2.1.3 Heading Estimation

When the most similar camera frame to the current camera frame is selected from the map with the help of local feature matching, the next step is performing heading vector estimation that could route the robot to navigate from the current frame to the target frame. To decide on the best one, we tried out four different methodologies. It is recommended for the reader to keep in mind that axes and angles are implemented by creating an egosphere around the robot in all implementations as in Figure 2.5.

Our first heading estimation method is adopted from [34]. In the first one, a unit vector is created to represent each keypoint by the help of longitudes. Then, different keypoints are paired with each other to enable vector analysis which helps computation of angular difference in between paired keypoints. In all implementations, we represented current frame with superscript c, and target frame with superscript t. Therefore, unit vectors for each keypoint pair in the current frame, which calculated in equation (2.1), represented with $\underline{u}^{c_i}$ and $\underline{u}^{c_j}$. Same procedure is applied also for

(a) Input Spherical Image

(b) Generated 80 Tangent Images

(c) Output Spherical Image

Figure 2.4: Tangent Image Generation & Local Feature Matching Process

the target camera frame unit vectors $\underline{u}^{t_i}$ and $\underline{u}^{t_j}$ in equation (2.2). After unit vectors are calculated, vector operations performed to show whether the angle between each keypoint pair increased (A_{ij}), and if keypoint order is changed (B_{ij}) from current frame to target frame. As the final step, $\underline{u}_{ij}$, bisector vector of each keypoint pair, is calculated in equation (2.6) and summed up to estimate the heading vector $\underline{h}$ in equation (2.7). First method is also visually explained in Figure 2.6 to be able to observe in term of the robot egosphere.

$$\begin{aligned} \underline{u}^{c_i} &= cos\big(\mu^{c_i}\big)\underline{i} + sin\big(\mu^{c_i}\big)\underline{j} \\ \underline{u}^{c_j} &= cos\big(\mu^{c_j}\big)\underline{i} + sin\big(\mu^{c_j}\big)\underline{j} \end{aligned} \tag{2.1}$$

$$\begin{aligned} \underline{u}^{t_i} &= cos\big(\mu^{t_i}\big)\underline{i} + sin\big(\mu^{t_i}\big)\underline{j} \\ \underline{u}^{t_j} &= cos\big(\mu^{t_j}\big)\underline{i} + sin\big(\mu^{t_j}\big)\underline{j} \end{aligned} \tag{2.2}$$

$$\begin{aligned} d^{c_{ij}} &= \underline{u}^{c_i} \cdot \underline{u}^{c_j} \qquad \underline{C}_{ij} = \underline{u}^{c_i} \times \underline{u}^{c_j} \\ d^{t_{ij}} &= \underline{u}^{t_i} \cdot \underline{u}^{t_j} \qquad \underline{T}_{ij} = \underline{u}^{t_i} \times \underline{u}^{t_j} \end{aligned} \quad \text{where } i \neq j \tag{2.3}$$

$$A_{ij} = sgn\big(d^{c_{ij}} - d^{t_{ij}}\big) \tag{2.4}$$

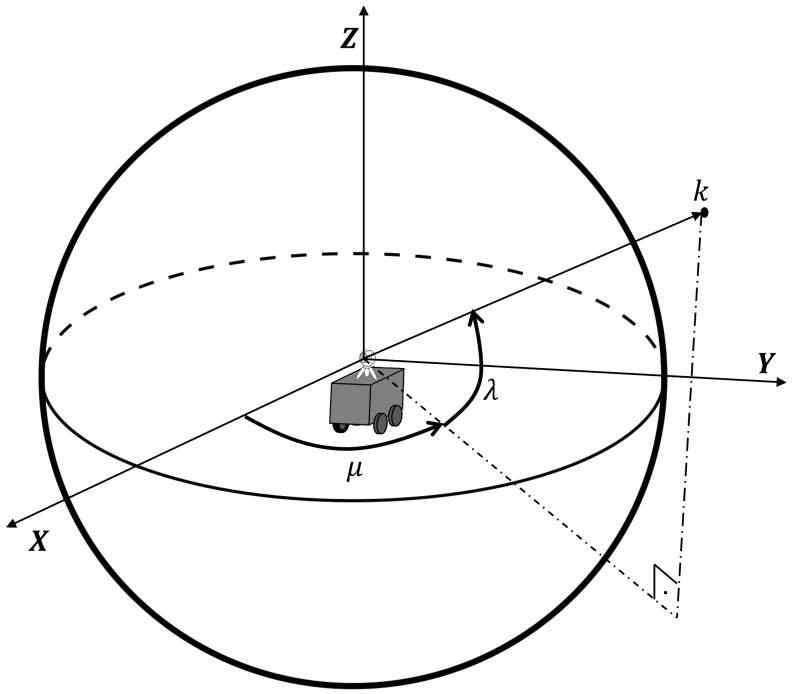


Figure 2.5: Axes and Angles Around the Robot Egosphere

$$B_{ij} = \frac{sgn\big(\underline{C}_{ij} \cdot \underline{T}_{ij}\big) + 1}{2} \tag{2.5}$$

$$\underline{u}_{ij} = \big(1 + B_{ij}\big(A_{ij} - 1\big)\big)\frac{\underline{u}^{c_i} + \underline{u}^{c_j}}{||\underline{u}^{c_i} + \underline{u}^{c_j}||} \tag{2.6}$$

$$\underline{h} = \sum \underline{u}_{ij} \tag{2.7}$$

In the second methodology, we seek improvement opportunities for the first one. Since all keypoints have the same priority on the heading calculation at the first method, it would face a heading bias in case of unevenly distributed matched keypoints around the true heading. In other words, if matched keypoints are localized in an area, method-1 tends to focus on that area, hence, deviating from the true heading. Therefore, we reinforced the heading calculation of method-1 with a gain which is a function of the change of longitude difference of keypoint pair between frames. Firstly, longitude difference of keypoint pair for the current frame $\Delta\mu^{c_{ij}}$ and the target frame $\Delta\mu^{t_{ij}}$ are calculated separately as in (2.8). Also, the process of calculating

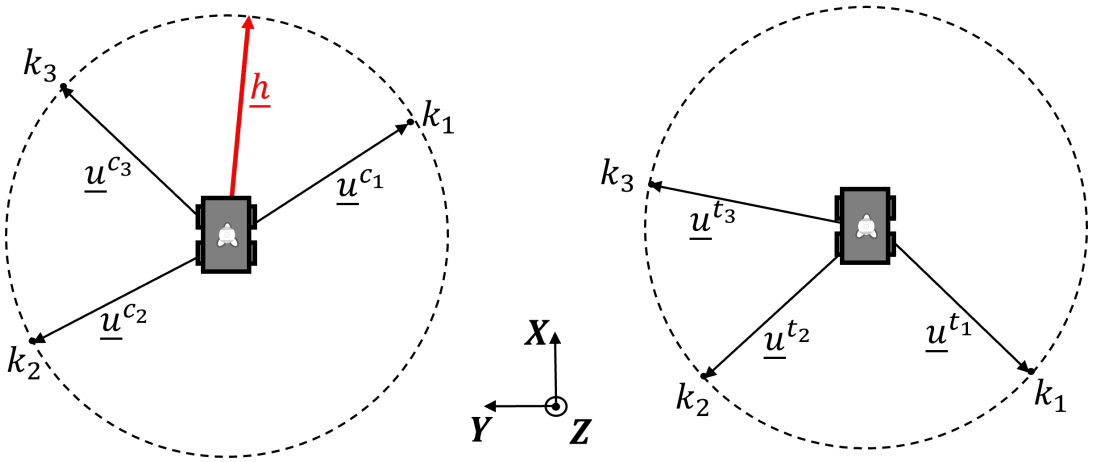


Figure 2.6: Heading Calculation Method-1

the longitude difference of keypoint pairs is visualized in Figure 2.7, in which the left image represents the longitude of each keypoint, the right image shows longitude differences for the keypoint pair for each frame. Then, the change of longitude difference of keypoint pair between frames is calculated in (2.9). As the last step, a function of the $\Delta\mu_{ij}$ is multiplied with the bisector unit vector $\underline{u}_{ij}$ to assign priorities to bisector unit vectors with respect to the change of longitude difference of each keypoint pair between frames.

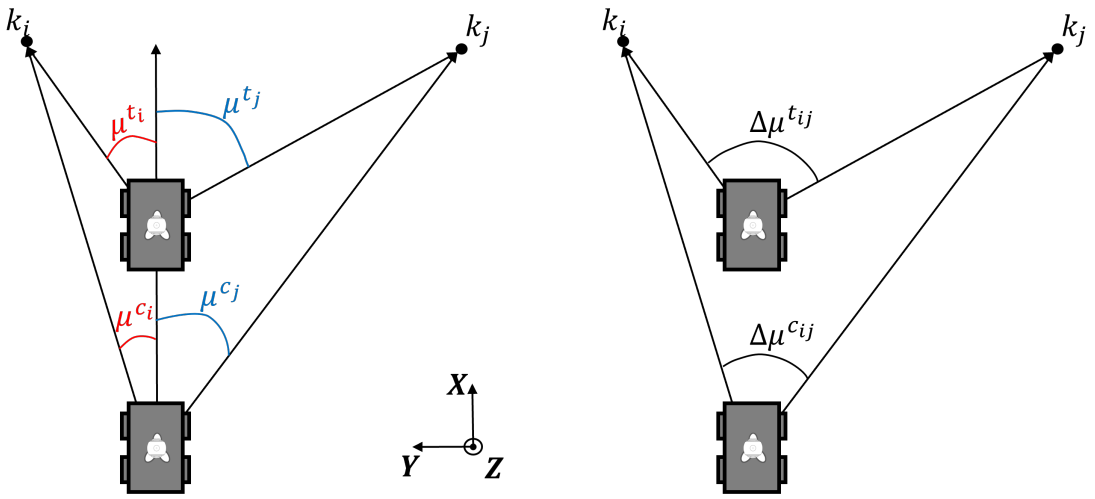


Figure 2.7: Longitude Difference of Keypoint Pairs

$$\begin{aligned}\Delta\mu^{c_{ij}} &= \left|\mu^{c_i} - \mu^{c_j}\right| \\ \Delta\mu^{t_{ij}} &= \left|\mu^{t_i} - \mu^{t_j}\right|\end{aligned} \tag{2.8}$$

$$\Delta\mu_{ij} = \Delta\mu^{t_{ij}} - \Delta\mu^{c_{ij}} \tag{2.9}$$

$$\underline{h} = \sum f\big(\Delta\mu_{ij}\big)\underline{u}_{ij} \tag{2.10}$$

In the third methodology, latitudes are taken into account on decision process of heading calculation. It is assumed that if the latitude of the selected keypoint is increasing in magnitude from current frame to target frame, then the selected keypoints is in front of the robot for both frames as visualized in Figure 2.8. Otherwise, the selected keypoint is classified as staying at behind of the robot for both frames. One of the main purposes of taking latitudes into account is eliminating keypoint pairing procedure introduced in the previous methods. Firstly, unit vectors are created to represent longitude of the keypoint in the current frame $\underline{u}^{c_i}$, latitude of the keypoint in the current frame $\underline{v}^{c_i}$, latitude of the keypoint in the target frame $\underline{v}^{t_i}$, and x-axis $\underline{v}^{x}$ to use on the decision making process in (2.11). Then, latitude unit vectors for both frames $\underline{v}^{c_i}$, $\underline{v}^{t_i}$ are paired with the x-axis unit vector $\underline{v}^{x}$ to represent both relative magnitude of latitudes between frames with A_i in (2.13) and ordering of pairs with respect to x-axis with B_i in (2.14). After calculating, A_i and B_i are used to define the direction of the robot for a specific keypoint whether it is towards or away from the current frame longitude unit vector $\underline{u}^{c_i}$. As the last step, longitude unit vectors with assigned directions $\underline{u_i}$ are summed up to estimate heading in (2.16). Whole process for this methodology is summarized in Figure 2.9.

$$\begin{aligned}\underline{u}^{c_i} &= cos\big(\mu^{c_i}\big)\underline{i} + sin\big(\mu^{c_i}\big)\underline{j} \\ \underline{v}^{c_i} &= cos\big(\lambda^{c_i}\big)\underline{i} + sin\big(\lambda^{c_i}\big)\underline{j} \\ \underline{v}^{t_i} &= cos\big(\lambda^{t_i}\big)\underline{i} + sin\big(\lambda^{t_i}\big)\underline{j} \\ \underline{v}^{x} &= 1\,\underline{i} + 0\,\underline{j}\end{aligned} \tag{2.11}$$

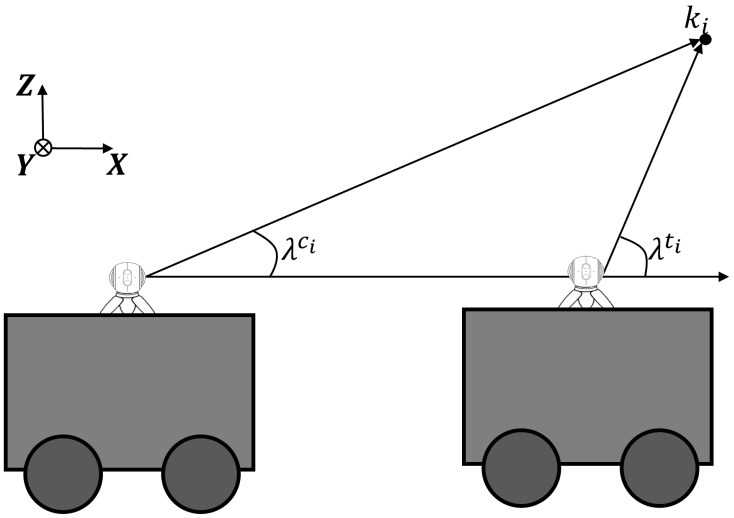


Figure 2.8: Change of Latitude Between Frames

$$\begin{aligned} d^{c_i} &= \underline{v}^{c_i} \cdot \underline{v}^{x} \qquad \underline{C_i} = \underline{v}^{c_i} \times \underline{v}^{x} \\ d^{t_i} &= \underline{v}^{t_i} \cdot \underline{v}^{x} \qquad \underline{T_i} = \underline{v}^{t_i} \times \underline{v}^{x} \end{aligned} \tag{2.12}$$

$$A_i = sgn\big(d^{c_i} - d^{t_i}\big) \tag{2.13}$$

$$B_i = \frac{sgn\big(\underline{C_i} \cdot \underline{T_i}\big) + 1}{2} \tag{2.14}$$

$$\underline{u_i} = \big(1 + B_i\big(A_i - 1\big)\big)\underline{u}^{c_i} \tag{2.15}$$

$$\underline{h} = \sum \underline{u_i} \tag{2.16}$$

In the fourth methodology, enhancements are undertaken to boost the effectiveness of the third methodology. As the main assumption, the selected keypoints are in front of the robot if the latitude of the selected keypoint is increasing in magnitude from the current frame to the target frame, may not hold for opposite cases. In more detail, in the case of keypoints that are crossed while moving from the current frame to the target frame, latitude difference in magnitude may still be negative as in Figure 3.15. Since each keypoint have the same significance in heading calculation, crossed keypoints directly affect heading estimation performance. Therefore, latitude difference

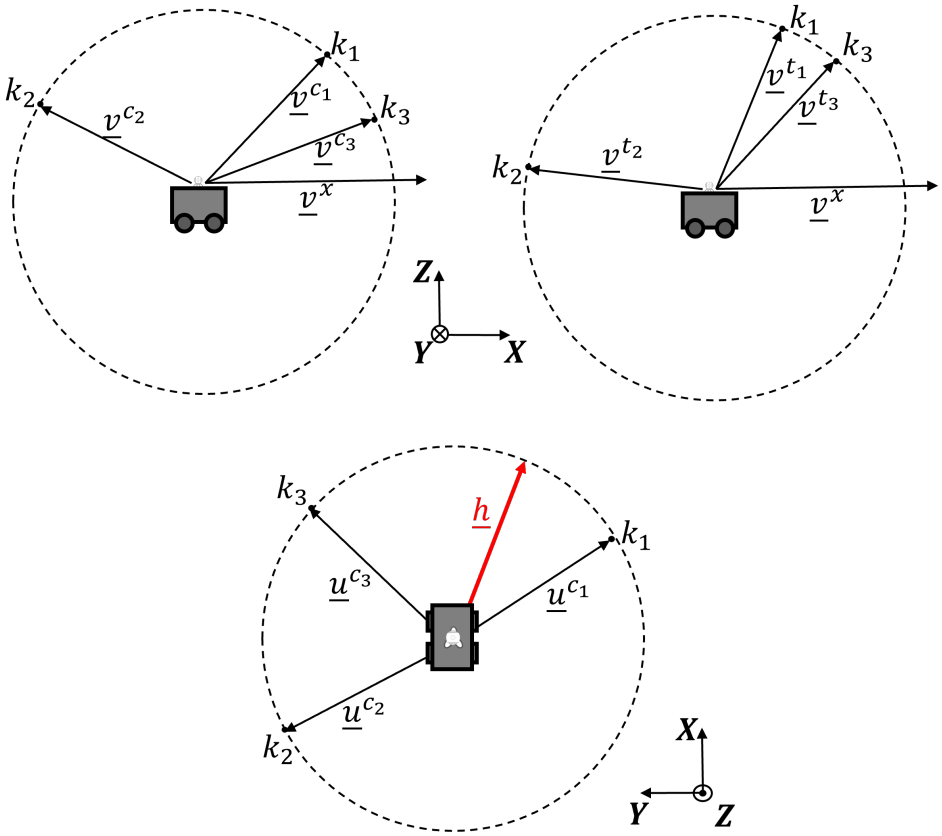


Figure 2.9: Heading Calculation Method-3

in magnitude $\Delta\lambda_i$ is calculated by subtracting absolute latitude of the current frame λ^{c_i} from absolute latitude of the target frame λ^{t_i} in equation (2.17). Then, a function of $\Delta\lambda_i$ is calculated and multiplied with the latitude unit vector $\underline{u_i}$ to be able to define ranking among keypoints in heading calculation as specified in equation (2.18).

$$\Delta\lambda_i = |\lambda^{t_i}| - |\lambda^{c_i}| \tag{2.17}$$

$$\underline{h} = \sum f(\Delta\lambda_i)\underline{u_i} \tag{2.18}$$

2.2.2 Topological Map Operations

Topological map operations can be defined as processes that are beyond current sensor measurements. Therefore, they allow the robot to be located on the overall topological map. The main implementation steps of topological map operations contain two major parts, namely mapping and exploration, and localization and path planning. While mapping and exploration is dedicated to adding new places to the topological map, localization and path planning is devoted to location and route estimation of the robot inside the previously constructed topological map to reach the target camera frame. These concepts will be investigated separately in detail in the following two sections.

2.2.2.1 Mapping and Exploration

In the mapping and exploration phase, the robot recognizes and appends new places, i.e. nodes, to the topological map concerning their similarity with already existing nodes on the map. In more detail, a new node is inserted into the topological map if the comparison of the current node with existing map nodes satisfies the defined similarity thresholds. Moreover, a connection, i.e. an edge, is added between the current node and existing map nodes if the comparison is higher than the lowest similarity threshold. This kind of approach enables the robot to explore new nodes while avoiding duplication of the same node. Therefore, we ensure that two distinctive camera frames with limited similarity are compared to establish robust topological navigation. When the mapping and exploration phase ends, a topological map that is essential in the topological navigation process will be ready to adopt. In addition, an example of a topological map, whose nodes symbolize equirectangular camera frames, can be seen in Figure 2.10.

2.2.2.2 Localization and Path Planning

We refer to localization as the location estimation of the robot with respect to the closest nodes in the constructed map during the topological navigation process. Localization is needed when a target frame is defined for the robot and a global assignment

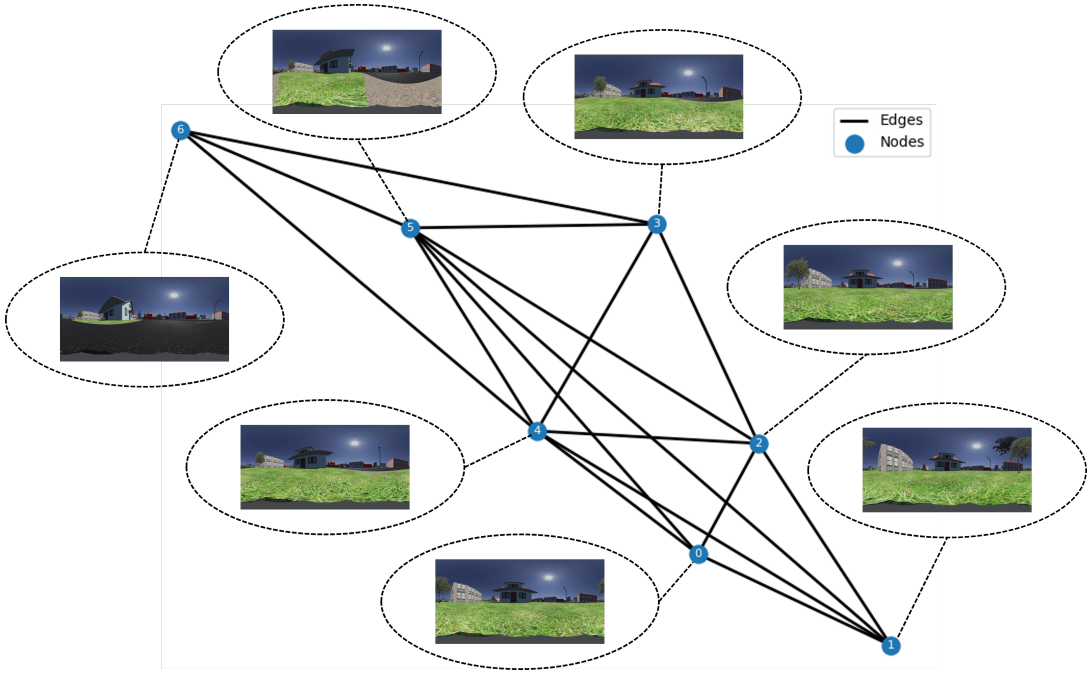


Figure 2.10: Sample Topological Map

wants to be computed. Since our topological navigation is based on reactive behavior which is limited by the sensor range, operations beyond the sensor range are mapped with global assignments. Therefore, we expect to complete defined global assignments by dividing them into smaller local tasks which then can be established with reactive behavior. Firstly, localization is performed to localize both the current frame and the target frame. After that, a path is defined by going through the map to find the nodes between the localized current and target frame to complete path planning. Then, nodes that are included by the planned path are defined as via points whose primary function is being intermediate checkpoints for topological navigation as in [78].

In summary in this chapter, image processing, and mobile robotics literature is expanded in detail to reinforce the awareness of the reader to ease following. We continued explaining the problem formulation and implementation steps of the solution to be more familiar with the terminology.

In the next chapter, the mentioned candidate methodologies for both Local Feature Matching and Heading Estimation will be evaluated to select methods that will be used in the implementation.

CHAPTER 3

METHOD COMPARISON

Selection of the correct algorithm is the central part of any development process. As explained in the Chapter 2, there are plenty of proposed algorithms both in Local Feature Matching and Heading Estimation sides. Therefore, to ease the method selection process with related comparisons and confirm that the applied method to the use case is the most convenient one among candidates, we conducted individual analyses by assigning ground-truth metrics. From the image processing point of view, 9 different local feature matching algorithms are proposed to implement them with the spherical images. Since the most popular spherical image representation method equirectangular representation results in spherical distortion due to the spherical grid as explained in Section 2.2.1.2, a candidate representation method tangent images is also applied to each local feature matching algorithm to compare the difference. Moreover, sample random scenes are created by generating Monte Carlo simulations to see the behavior of heading estimation methods that are introduced in Section 2.2.1.3. As the final argument, the chapter will be closed by specifying the local feature matching algorithm and the heading estimation method that will be used in the implementations.

3.1 Local Feature Matching Algorithm Comparison

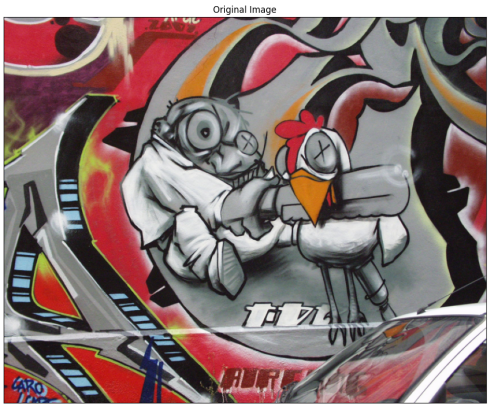
Various approaches published to solve the local feature matching problem. However, the literature lacks a comparison of state-of-the-art algorithms to find out the most compatible methodology that works in harmony with equirectangular images. Current comparison studies are not able cover due to spherical distortion shown in

Figure 2.1. Therefore, before diving into implementation details and case studies, we constructed an analysis to decide on the most suitable local feature matching algorithm combination according to our preferences. In the analysis, we investigated 9 different local feature matching algorithms by placing images with a 90° field of view to different center latitudes as in Figure 3.4. After that, local feature matching is performed between the image that is placed at the center of the spherical grid as in Figure 3.3 and each image that is placed at different center latitudes as in Figure 3.4, Figure 3.6, Figure 3.8 and Figure 3.10. In order to examine algorithm efficiency concerning the distinctive point of views such as motion blur Figure 3.1d, image rotation Figure 3.1b, scaling difference Figure 3.1c, and image representation, related parameters are changed accordingly. Metrics used to measure effectiveness of algorithms in the distortion analysis are inspired from [47] and stated in equations (3.1) and (3.2). Moreover, since images are fed into the local feature matching algorithm as a whole, keypoints extracted from black regions or at the border of an image are discarded and not taken into account in the performance calculation.

$$recall = \frac{\#\, correct\, matches}{\#\, correspondences} \tag{3.1}$$

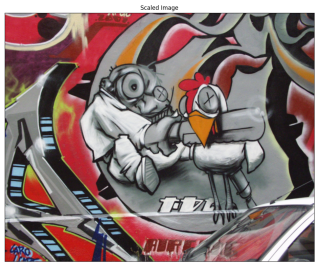
$$precision = \frac{\#\, correct\, matches}{\#\, correct\, matches + \#\, false\, matches} \tag{3.2}$$

In the analysis, we followed the methodology to generate the overall score mean Average Precision (mAP) mentioned in [13]. At this point, we would like to note that, Nearest Neighbor is used as a matcher for all algorithms except SuperPoint [13]. Since it is stated in SuperGlue implementation [59] that SuperPoint is more efficient with SuperGlue, SuperGlue is assigned as the matcher of SuperPoint. Firstly, we selected different matching thresholds, which will be applied in each local feature matching process, to perform matching. In addition, in each performed matching, the extracted keypoints are rectified with respect to the image center latitude to get rid of the spherical distortion. A geometrical matched keypoint verification method RANSAC [22] is executed to classify matched keypoints as correct or false matches, and correspondences in the equation (3.1) are assigned as the number of keypoints extracted from an image. Then, precision-recall curve is constructed with metrics

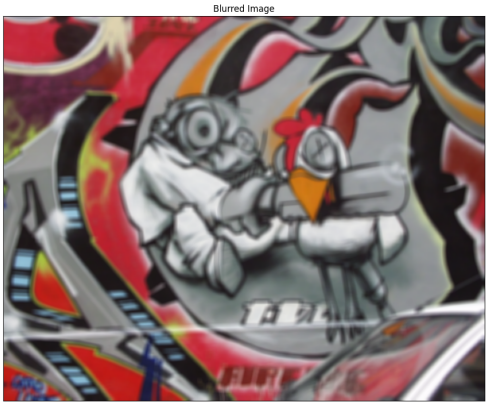


(a) Original Image

(b) Rotated Image



(c) Scaled Down Image



(d) Blurred Image

Figure 3.1: Sample Investigation Points for Local Feature Matching Performance

specified in equations (3.1) and (3.2) for each image pair as in Figure 3.11 to compute Area Under Curve (AUC). To make AUC computation easier, the curve is assumed as a step so that each recall data point is connected with a constant horizontal line. Before constructing the curve, both recall and precision values are sorted out concerning increasing order of recall since it is assigned as the x-axis. Then, AUC is computed to see descriptor performance. After the AUC computation for each curve, the AUC is combined with respect to different parameters and averaged to calculate mAP as can be seen from figures Figure 3.2, Figure 3.5, Figure 3.7 and Figure 3.9. It is detected that dividing equirectangular images into tangent images to apply the keypoint description process eliminates distortion, which increases proportional with latitude, as seen in Figure 3.2. Although equirectangular representation results in a drastic mAP drop when latitude is separated from the zero-latitude zone, SuperPoint still holds its efficiency. Parallel results were detected from scaling difference

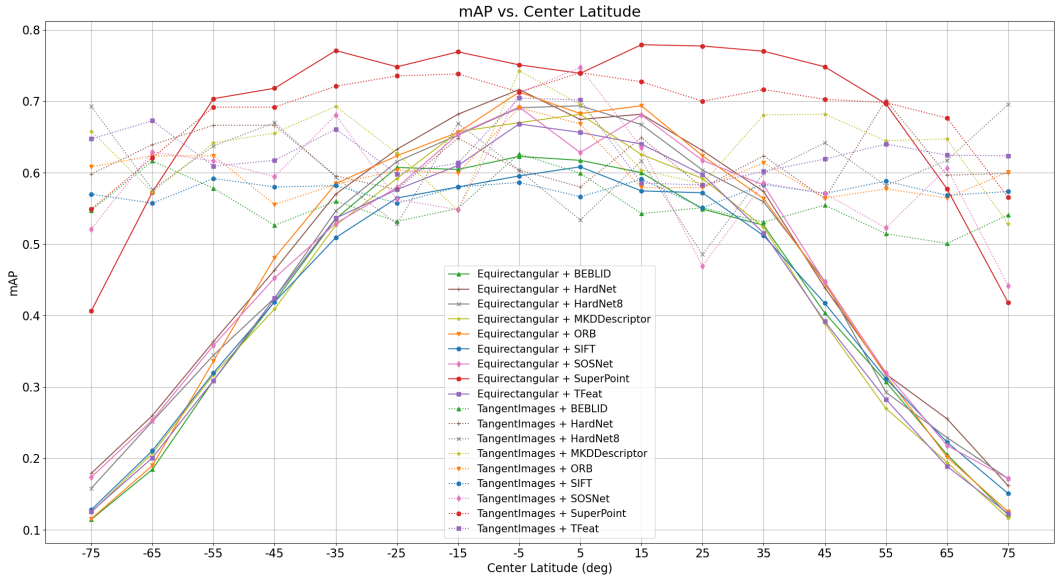
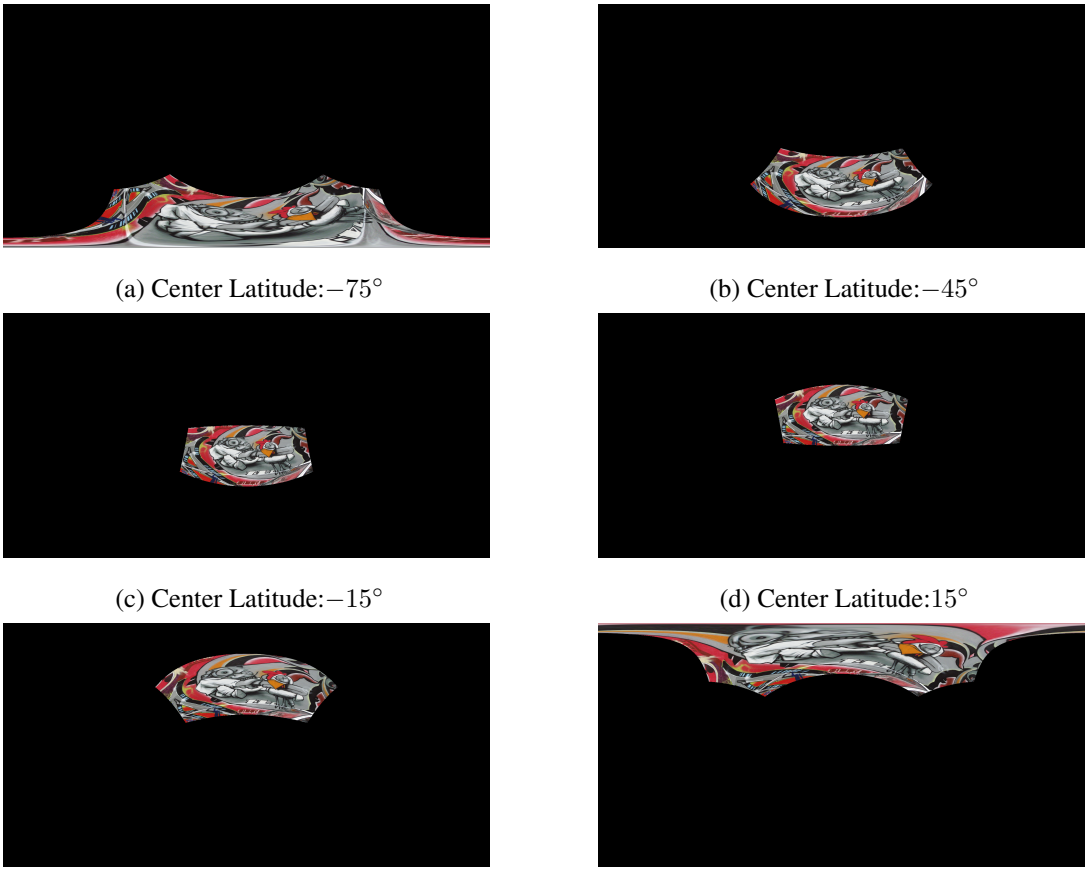


Figure 3.2: mAP Change with Latitude

Figure 3.3: Original Image Placed to Center of the Spherical Grid, 0° Latitude

Figure 3.9, image rotation Figure 3.7 and Gaussian blur Figure 3.5 in which equirectangular representation except with SuperPoint is not as effective as tangent image representation. Therefore, at this point, we started looking at the computation time differences for each local feature matching algorithm and the representation combination to add up a different perspective. Analyses are generated on a computer equipped with Quadro RTX 4000 GPU and Intel Xeon Gold 5218 2.30 GHz CPU, and computation times are calculated by taking the average of specified local feature matching method and image representation combination. All methods except SIFT, ORB, and BEBLID are run on GPU due to their speed increment. As can be seen from Table 3.1 that tangent images representation drastically drops the computational effectiveness of local feature matching algorithms. Since tangent images divide an equirectangular image into 80 small tangent images to get rid of the spherical distortion behavior as shown in Figure 2.4, performing local feature matching to all of these small pieces increases computation time reasonably. Therefore, in the implementation part of the



(a) Center Latitude:$-75°$ (b) Center Latitude:$-45°$

(c) Center Latitude:$-15°$ (d) Center Latitude:$15°$

(e) Center Latitude:$45°$ (f) Center Latitude:$75°$

Figure 3.4: Original Image Placed to Different Center Latitudes

thesis, using SuperPoint and SuperGlue with the equirectangular representation will be the best possible combination.

3.2 Heading Estimation Algorithm Comparison

Heading estimation is another important algorithm that will strengthen the backbone of topological navigation. Since the primary sensor that is used in the topological navigation process is a spherical camera, understanding the heading estimation behavior along with the extracted keypoints from spherical images would be essential to build up deeper insight into the overall performance. Therefore, we generated different Monte Carlo Simulation analyses for two main purposes. As the common point, in each scene, the robot's current and target locations and randomly generated keypoint locations are specified. In any run, a specified number of keypoint

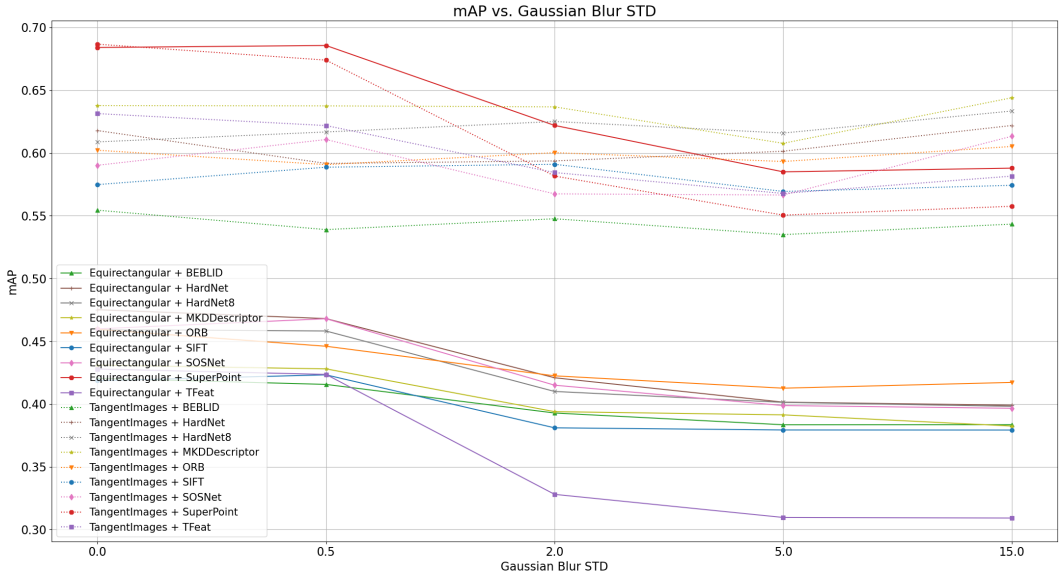
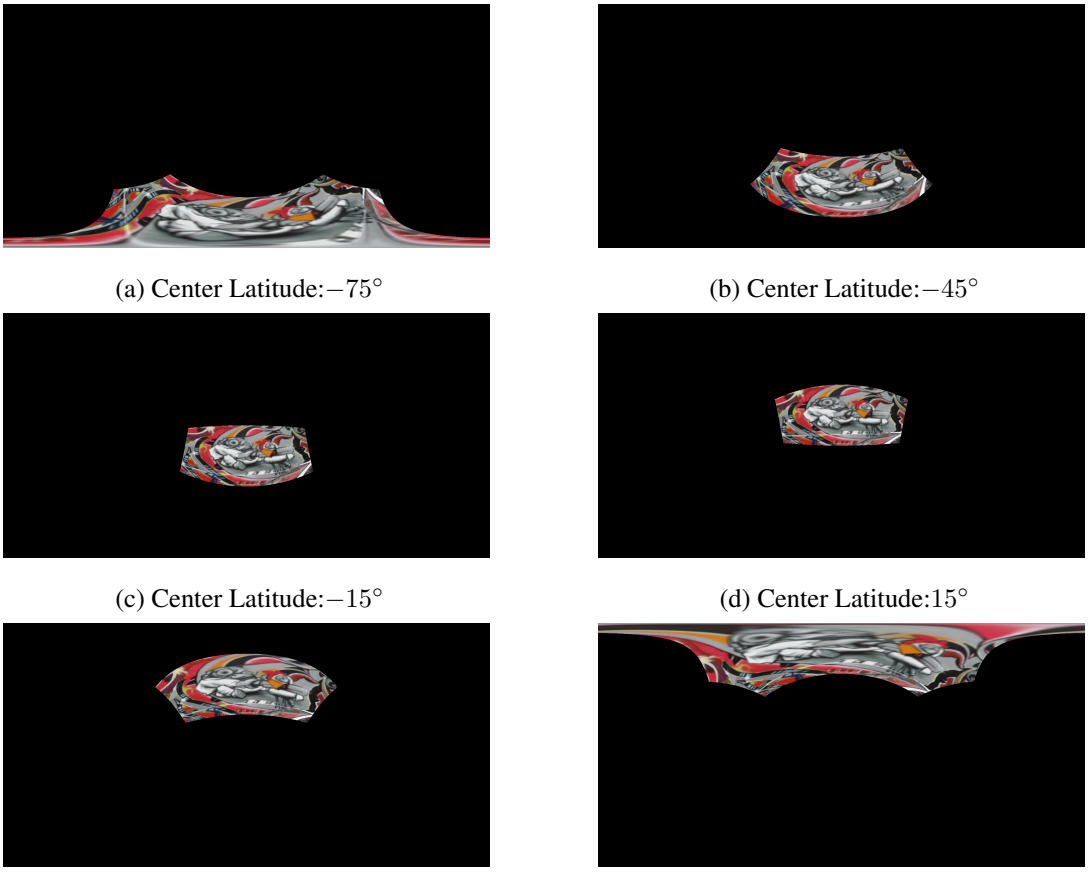


Figure 3.5: mAP Change with Gaussian Blur

is selected randomly to simulate matched keypoints between the robot's current and target locations. Then, the run is performed and the results are compared according to distinct purposes. In the first analysis, robot locations are kept constant, and heading estimation is compared with the ground truth heading to investigate heading error change among methods and with respect to the changed number of matched keypoints. In the second analysis, an iterative process is performed to move the robot from its current location to the target location to compare path tracking performances of methods. While investigating the results, the distance covered by the robot with the lead of the specified heading estimation method is compared with the minimum distance between robot locations, which is simply a direct line. Moreover, to gather and review results in terms of a metric, we have defined two distinct error metrics Absolute Heading Angle Error (AHAE) and Distance Covered Percent Error (DCPE) which will be explained in more detail in the upcoming sections. At the end of the analyses, we intend to select both the most appropriate heading estimation method for our use case and the optimum number of matched keypoints after which performance increment is limited.



(a) Center Latitude:$-75°$ (b) Center Latitude:$-45°$

(c) Center Latitude:$-15°$ (d) Center Latitude:$15°$

(e) Center Latitude:$45°$ (f) Center Latitude:$75°$

Figure 3.6: Blurred Image Placed to Different Center Latitudes

3.2.1 Heading Estimation Performance with Angle Comparison

While local feature matching affects navigation performance, it also influences computational efficiency significantly due to keypoint pairing mentioned in Section 2.2.1.3. To better illustrate the change of array length with the keypoint pairing for the different number of matched keypoints, Figure 3.12 is plotted. In the figure, it can be easily detected that array length due to keypoint pairing increases quadratically with the incremented number of selected keypoints. Therefore, the best solution would be to generate an as low amount of keypoints as possible while maintaining the navigation performance. To find out the optimum number of keypoints that need to be found in frames, Monte Carlo simulation is performed by setting up sample scenes that include various keypoints with random current and target frame locations as in Figure 3.13. In detail, we uniformly generated 1000 keypoints to a sample scene as labeled with green scatter in Figure 3.13. In the scene, while the target frame is placed

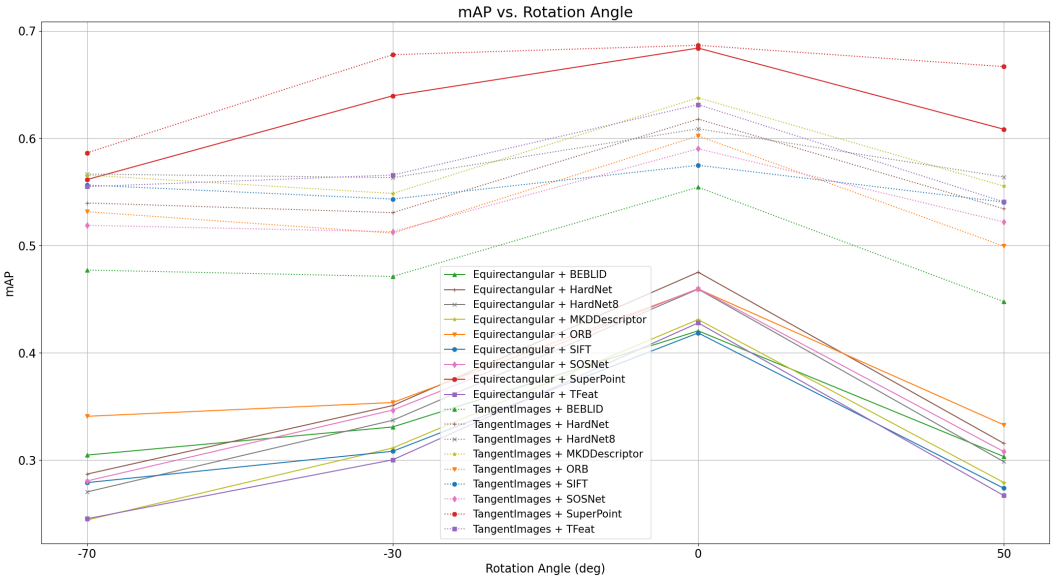
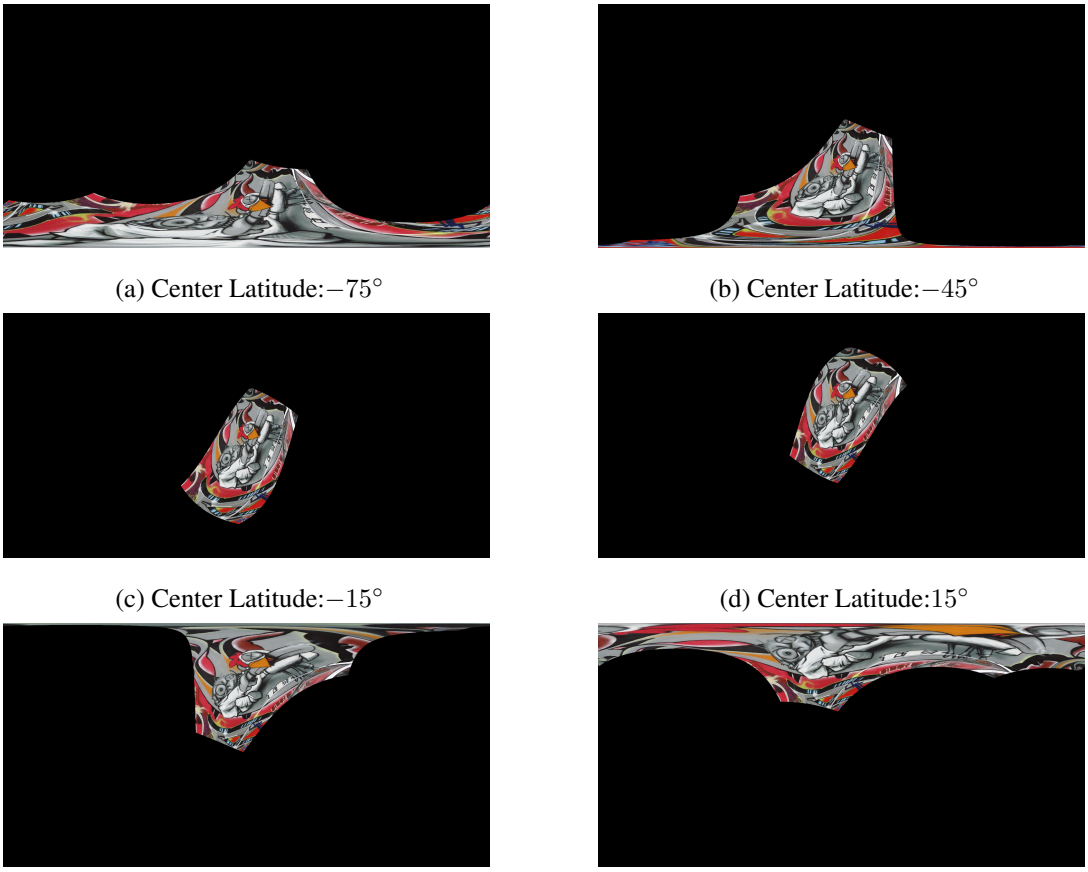


Figure 3.7: mAP Change with Image Rotation

in a specific location to study different keypoint distributions around the target frame, current frame and keypoint locations are randomly specified to make performance investigation independent of those parameters. Then, random keypoints are selected to simulate matched keypoints between frames, and the number of matched keypoints is kept constant for 2000 runs. After each 2000 run, the number of matched keypoints is increased to iterate through the whole batch. When the last number of matched keypoints is investigated, a new scene is generated and the process is repeated with 100 scenes. For each run, the Absolute Heading Angle Error (AHAE) is calculated and symbolized with $\Delta\phi$ as expressed in the equation (3.3). In the error calculation process, true heading vector $\underline{h_t}$ is subtracted from estimated heading vector $\underline{h}$ that is output of each method to calculate error heading vector $\Delta\underline{h}$, and the Absolute Heading Angle Error (AHAE) is computed by taking absolute arc-tangent of the $\Delta\underline{h}$ components. At the end of each scene, the standard deviation and mean of gathered AHAE are calculated for a distinct number of matched keypoints. In the figures, box plots include the mean and standard deviation of AHAE for every scene with respect to the number of matched keypoints. In addition to the 4 heading estimation methodologies mentioned earlier, we created two additional combined methods method-5 and method-6 to investigate the combination of method-1 with method-4, and method-2 with method-4 respectively. Moreover, assigned keypoint priorities with an explicit function at method-2 and method-4 in equations (2.10) and (2.18) are still open to



(a) Center Latitude:$-75°$ (b) Center Latitude:$-45°$

(c) Center Latitude:$-15°$ (d) Center Latitude:$15°$

(e) Center Latitude:$45°$ (f) Center Latitude:$75°$

Figure 3.8: Rotated Image Placed to Different Center Latitudes

improvements. We have only taken the exponential function into account to define the significance of each angle difference to see its effect on the heading calculation performance. Therefore, while function of $\Delta\mu_{ij}$ calculated for method-2 in (2.10) is replaced with (3.4), function of $\Delta\lambda_i$ calculated for method-4 in (2.18) is changed as (3.5). During experiments, it has been realized that the change of longitude difference of keypoint pair $\Delta\mu_{ij}$ increases while bisector heading of paired keypoints approaches to the true heading between frames as shown in Figure 3.14. Therefore, heading vector priority is assigned to highlight keypoints that have a higher change of longitude difference of keypoint pair with the help of exponential function. In addition, the main motivation of using exponential function for the function of $\Delta\lambda_i$ in (2.18) is sketched in Figure 3.15. In the case of having a keypoint in the middle of two frames, the keypoint could be closer to the current frame, i.e. current frame could have higher latitude compared to the target frame. Since latitude decrease in magnitude is stated as moving away from the longitude unit vector of specified key-

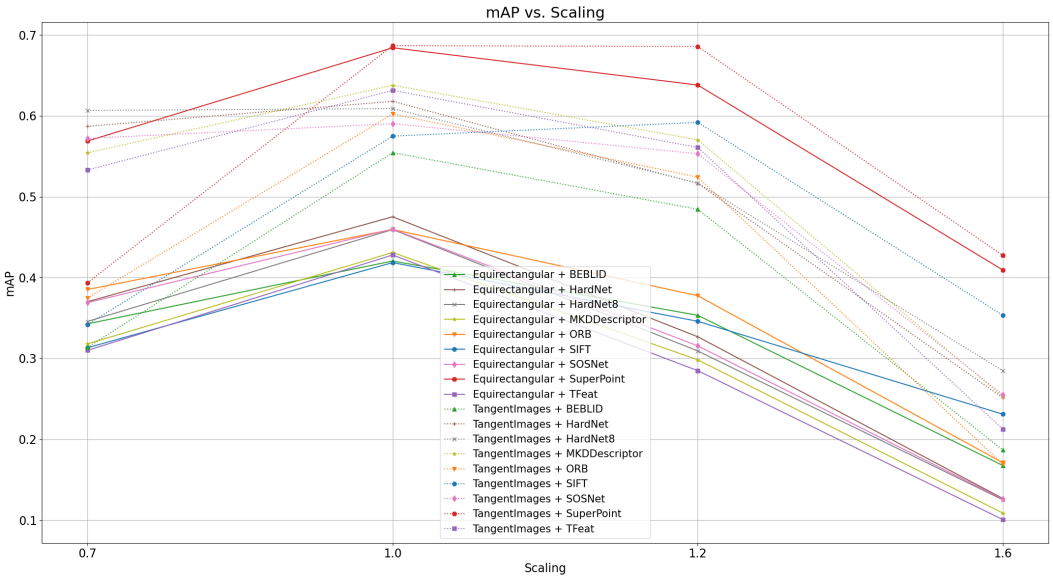


Figure 3.9: mAP Change with Image Scaling

point, the occurrence of a situation in Figure 3.15 would mislead the robot. Therefore, we aim to diminish this problem by increasing the priority of longitude unit vectors with respect to latitude increase in magnitude between frames. Further development opportunities for those functions are left as future work of this thesis.

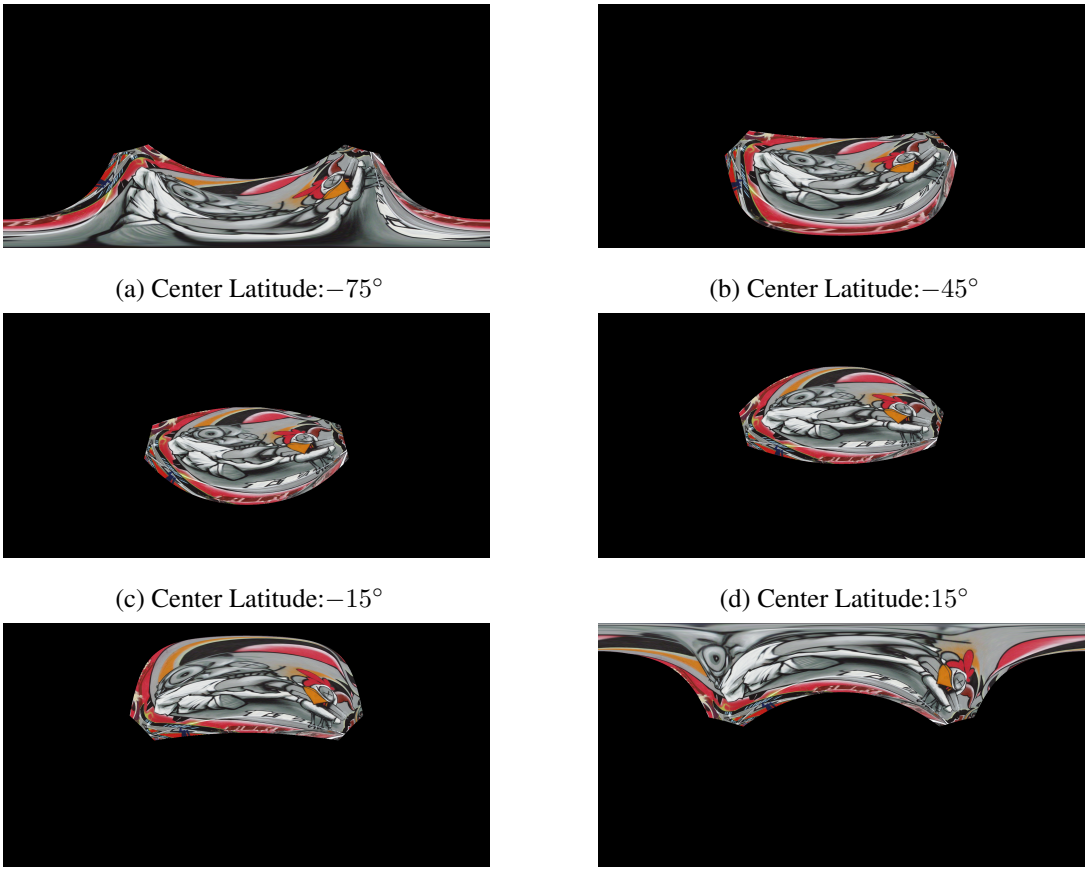
$$\begin{aligned} \Delta \underline{h} &= \underline{h} - \underline{h_t} \\ \Delta \phi &= |\arctan(\Delta \underline{h}_y, \Delta \underline{h}_x)| \end{aligned} \tag{3.3}$$

$$f\big(\Delta \mu_{ij}\big) = exp\big(\Delta \mu_{ij}\big) \tag{3.4}$$

$$f\big(\Delta \lambda_{i}\big) = exp\big(\Delta \lambda_{i}\big) \tag{3.5}$$

3.2.1.1 Target Frame, Located In the Center of the Scene

In this section of the analysis, the target frame is located in the center of the scene in order to expand not only keypoint distribution around the frame but also heading between robots to the whole domain. While figures Figure 3.17, Figure 3.18, Figure 3.19, Figure 3.20, Figure 3.21 and Figure 3.22 are showing heading error standard deviations and means for each method, Figure 3.23, and Figure 3.24 dedicated



(a) Center Latitude:$-75°$ (b) Center Latitude:$-45°$

(c) Center Latitude:$-15°$ (d) Center Latitude:$15°$

(e) Center Latitude:$45°$ (f) Center Latitude:$75°$

Figure 3.10: Scaled Up Image Placed to Different Center Latitudes

to comparison of methods among each other. In box plots for each method, as the first step, AHAE standard deviations and means are calculated for each scene. Then, they are gathered together with respect to the number of matched keypoints to see their statistical results. As a specific example, in the top figure of Figure 3.17, firstly, the standard deviation of AHAE is calculated for each scene. Secondly, standard deviations for all scenes are merged with respect to the number of matched keypoints. Thirdly, the mean of standard deviations is calculated to draw a dashed blue line in the graph. Note that, orange horizontal lines inside boxes show the median of standard deviation distribution, boxes show interquartile range where 50% standard deviations are located, lines starting from the border of boxes called whiskers to include left 25% of the distribution in both ranges, and circles outside whiskers are stated as outliers. Also, details of box plot parts can also be followed with the help of Figure 3.16. When methods are compared with each other, means or standard deviations calculated from heading errors are used. In detail, in Figure 3.23, calculated standard deviations for

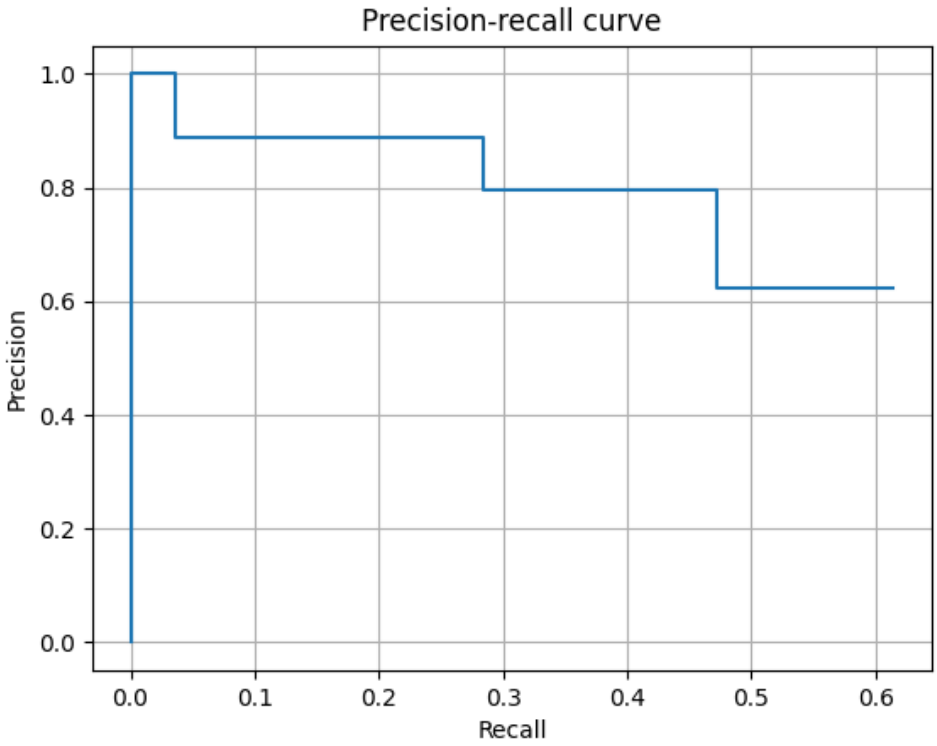


Figure 3.11: Sample Precision-recall curve

each scene are clustered with respect to the number of matched keypoints, and their mean and standard deviation are taken to result in one value that represents the overall behavior in the specified number of matched keypoint. At this point, it is safe to say that the dashed blue line showing the mean of standard deviations in the top figure of Figure 3.17 is the same line labeled with method-1 in the top figure of Figure 3.23.

It can be seen from Figure 3.23 and Figure 3.24 that methods having no priority on keypoints, method-1, method-3, and method-5, tend to have better performance in terms of the standard deviations for all scenes. This pattern is also visible in box plots Figure 3.17, Figure 3.19 and Figure 3.21 so that method-1, method-3, and method-5 have narrower box widths, i.e. interquartile ranges, than method-2, method-4, and method-6 in Figure 3.18, Figure 3.20 and Figure 3.22. Therefore, heading estimation varies slightly less when keypoint priority is not assigned. However, it does not mean that heading estimation performance is better on those methods. As it is noticeable in Figure 3.23 and Figure 3.24 that method-3 has significantly higher AHAE mean

Table 3.1: Computation Time Comparison of Local Feature Matching Algorithms

Algorithm	**Representation**	**Computation time (s)**
SuperPoint	Tangent images	1.12
TFeat	Tangent images	5.04
HardNet	Tangent images	5.09
SOSNet	Tangent images	5.08
HardNet8	Tangent images	5.10
MKDDescriptor	Tangent images	5.11
SIFT	Tangent images	0.97
ORB	Tangent images	0.85
BEBLID	Tangent images	0.77
SuperPoint	Equirectangular	0.16
TFeat	Equirectangular	0.34
HardNet	Equirectangular	0.34
SOSNet	Equirectangular	0.36
HardNet8	Equirectangular	0.36
MKDDescriptor	Equirectangular	0.34
SIFT	Equirectangular	0.11
ORB	Equirectangular	0.06
BEBLID	Equirectangular	0.04

than other methods. Therefore, when keypoints are evenly distributed around the target frame, methods except method-3 have similar performance with each other, and assigning priority for keypoints with respect to the elevation angle difference increases heading estimation performance.

3.2.1.2 Target Frame, Located In the North of the Scene

In this section of the analysis, the target frame is located in the north of the screen in order to limit both keypoint distribution around the frame in [-180°, 0°] and heading between robots in [0°, 180°].

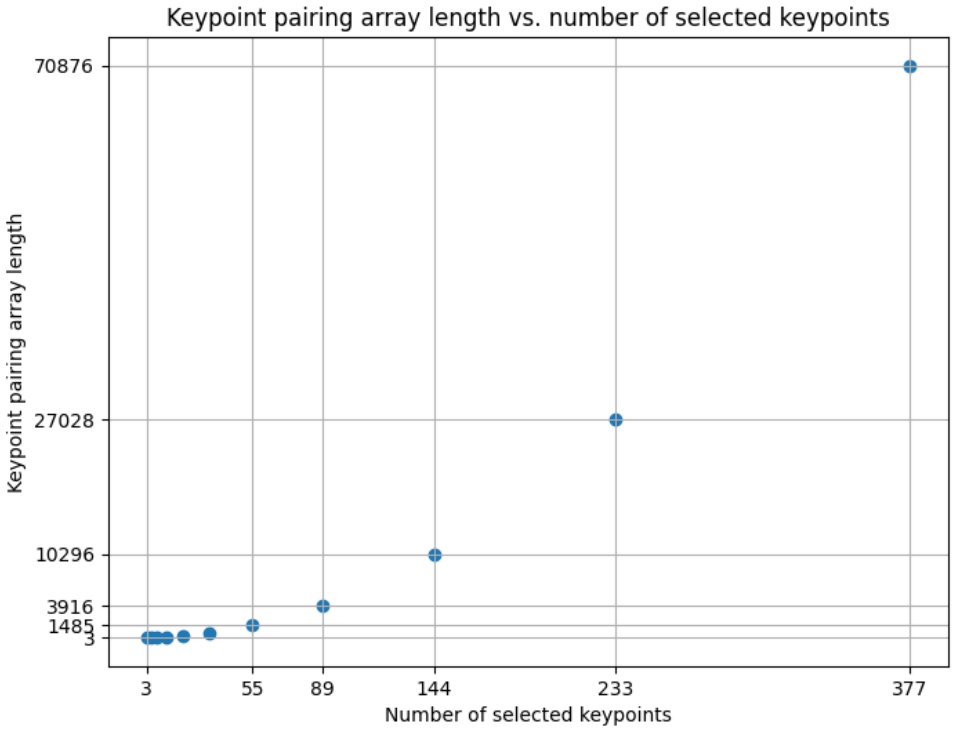


Figure 3.12: Change of Array Length With Number of Selected Keypoints

It can be seen from Figure 3.31 that method-4, method-5, and method-6 have better performance on the standard deviation of AHAE standard deviations for all scenes at lower number of matched keypoints which is an indication of less abnormality in terms of AHAE. Furthermore, when keypoint distribution focuses on an interval and keypoints are not evenly distributed around the true heading, methods that have no priority on keypoints suffer from heading bias as realized from the mean of AHAE means in Figure 3.32. In more detail, in the case of having matched keypoints that are localized in an area, method-1, method-3 and method-5 tend to focus on that area, hence, deviate from true heading as seen with wide boxes and whiskers in bottom figures of Figure 3.25, Figure 3.27 and Figure 3.29. Therefore, it is directing us to the conclusion that assigning priority to keypoints would increase heading estimation accuracy in the application phase.

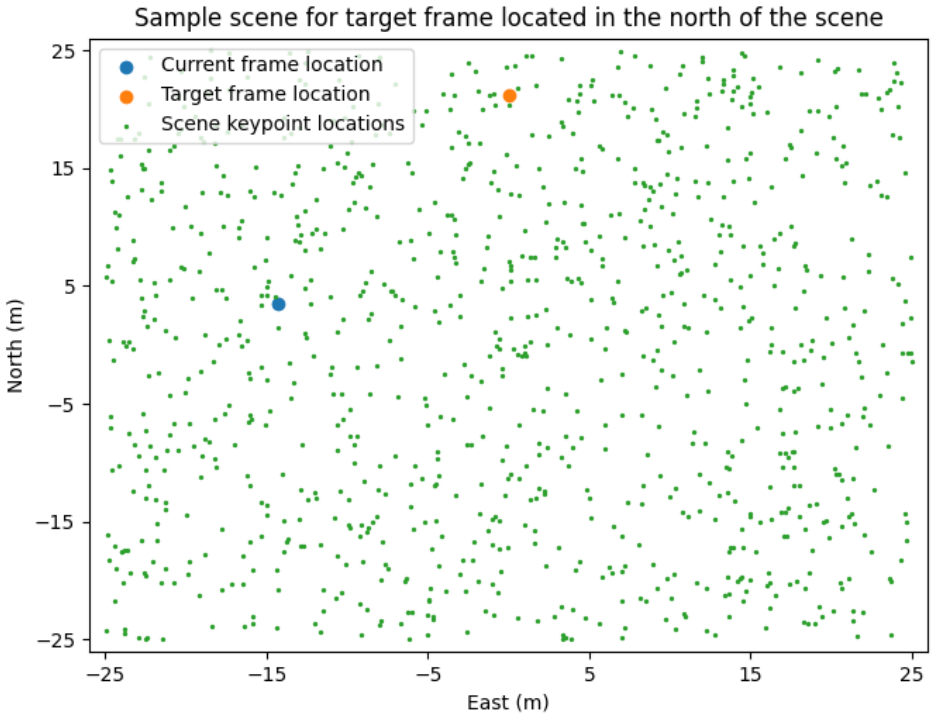


Figure 3.13: Sample Monte Carlo Simulation Scene

3.2.1.3 Target Frame, Located In the North-East of the Scene

In this section of the analysis, the target frame is located in the north-east of the screen in order to limit both keypoint distribution around the frame in [-180°, -90°] and heading between robots in [0°, 90°].

It is easy to catch from both Figure 3.39 and Figure 3.40 that method-3 has the worst performance on AHAE since it has the highest mean and standard deviation for almost all the number of matched keypoints. Also, it is detected from the Figure 3.39 that AHAE standard deviations result in a steeper slope in method-4 when the number of matched keypoints is decreased below 34. This means that method-4 tends to vary more than other methods, hence; in the case of bounded keypoint distribution, method-4 needs a high number of matched keypoints to make a comparable heading estimation. Moreover, parallel results are yielded as specified in the analysis Target Frame, Located In the North of the Scene in terms of AHAE means. Although keypoint distribution focuses on an interval and keypoints are not evenly distributed

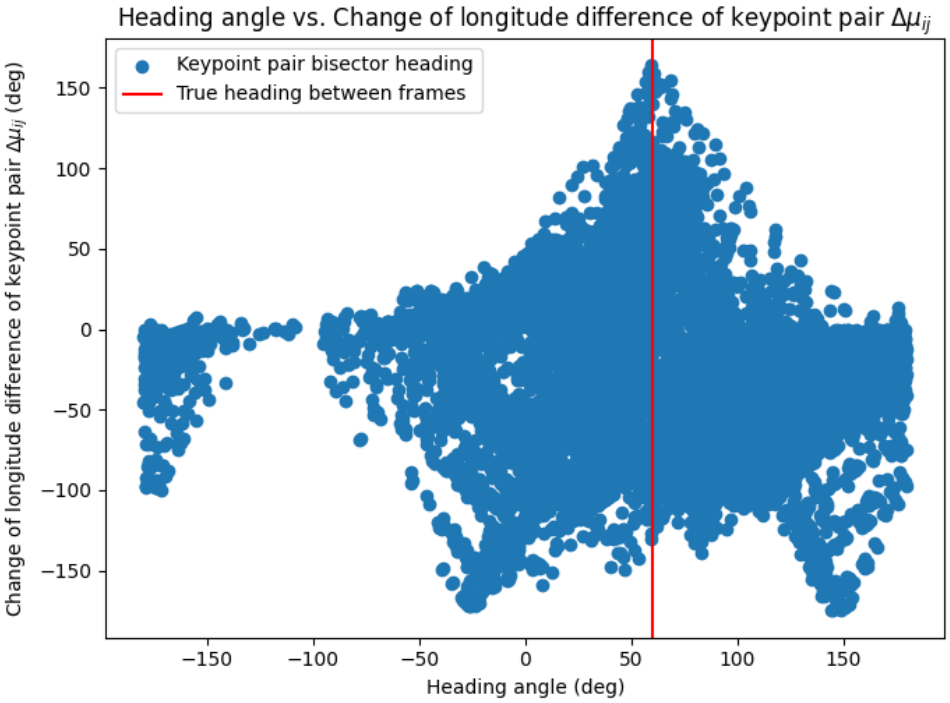


Figure 3.14: Longitude Difference and Heading Relation

around the true heading, methods with assigned keypoint priorities have lower AHAE means as seen in Figure 3.40. However, since keypoint distribution is more bounded to a region when the target frame is placed in the north-east of the scene, we realized an increment compared to the target frame placed in the north of the scene in the standard deviation of AHAE means in the method-4 and method-6. Further, the dashed blue line, which is representing the means of AHAE mean, is not passing inside the interquartile in Figure 3.34. It demonstrates the fact that the exponential function, which is used to assign priorities on keypoints, influences the AHAE means in most cases but does not diminish whole drawbacks. In other words, the figure illustrates high outliers that affect the mean considerably but does not occur frequently. Even though outliers do not occur regularly to shift the box plot upwards in this analysis, it is a good hint to keep in mind for the next investigations.

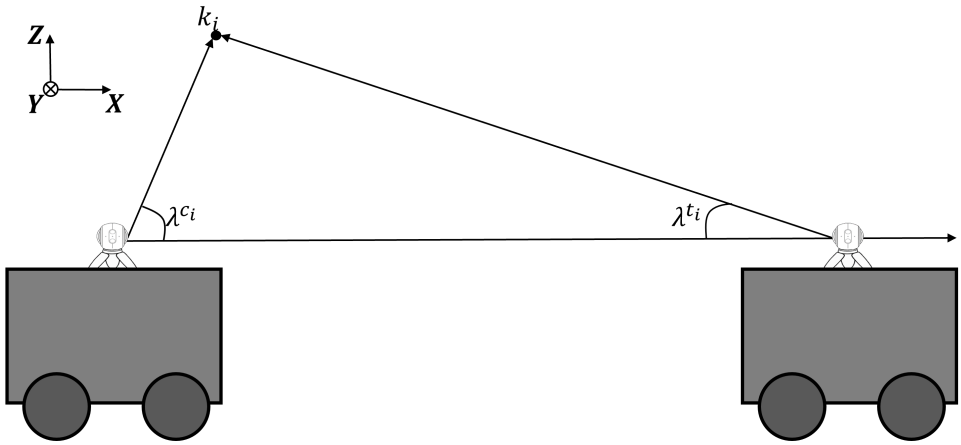


Figure 3.15: Sample Failure Case of Latitude Decrease

3.2.2 Heading Estimation Performance with Path Tracking Comparison

Since decreasing the number of keypoints that are extracted from a scene is crucial in terms of computational efficiency as also described in section 3.2.1, we aim to enlarge the optimum number of keypoint selection operation by creating an iterative path tracking process from current frame to target frame. In more detail, another Monte Carlo simulation is conducted by introducing sample scenes in which randomly generated keypoints, current frame, and target frame locations are specified as shown in Figure 3.13. In the scene creation, the methodology is kept the same as in section 3.2.1. Therefore, in a sample scene, 1000 random keypoints are created, the target frame is placed in a specific location, and the current frame is randomly located to test different keypoint location combinations. After the scene is created and frame locations are specified, random keypoints are selected to simulate matched keypoints between frames. An iterative process is started to be performed to estimate the heading to approach the target frame location, and the current frame location is moved towards the estimated heading with respect to selected speed and time-step. This iterative process is followed until the robot has reached the target frame. At this point, it is essential to say that selected keypoints that simulate matched keypoints between frames are not changed. Instead, their relative location to the current frame is computed at the end of each iteration. From a different point of view, a 2D simulation without any dynamic behavior is created to compare path tracking performances of heading estimation methodologies as shown in Figure 3.41. To investigate the performance, the distance covered by the robot is compared with the minimum distance

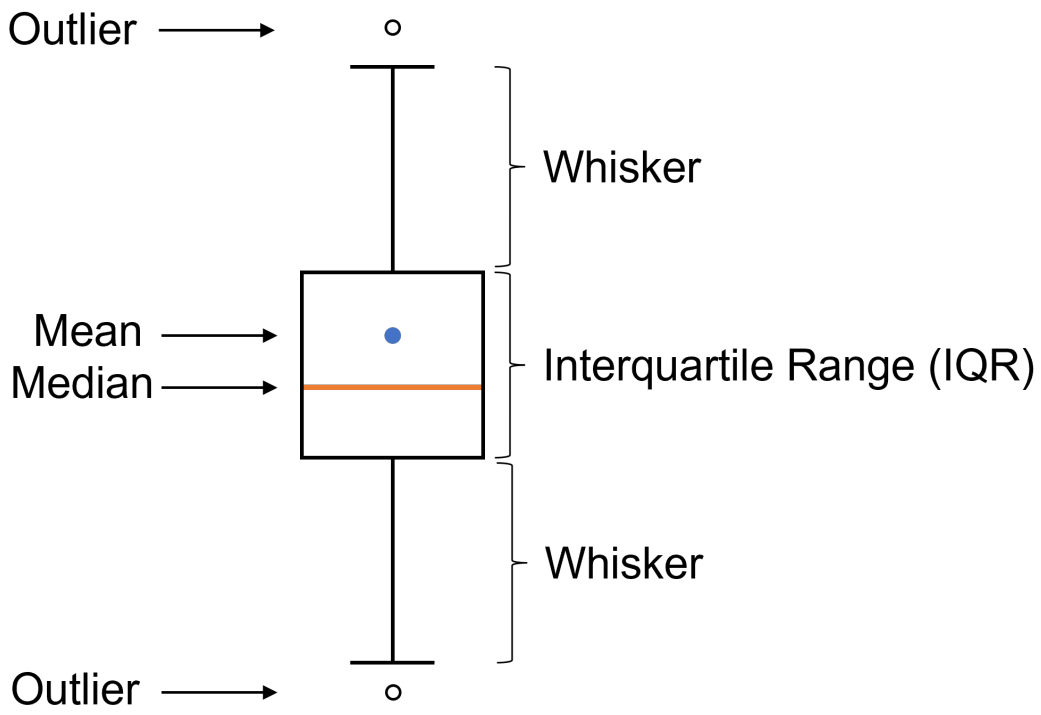


Figure 3.16: Parts of a Box Plot

between the current frame and the target frame. In more detail, the minimum distance between current frame and target frame d_m is subtracted from the distance covered d_c, and the difference Δd is divided by the minimum distance between current frame and target frame. Result of the division is multiplied by 100 to calculate the Distance Covered Percent Error (DCPE) Δd_p as in (3.6). Moreover, the same 6 methods investigated in section 3.2.1 are handled also in this study. As the final argument, we have limited the distance covered d_c with the twice of the minimum distance d_m to avoid infinite iteration loop. Hence, the maximum DCPE will be calculated as 100 in the analysis.

$$\begin{aligned} \Delta d &= d_c - d_m \\ \Delta d_p &= \left(\frac{\Delta d}{d_m}\right) 100 \end{aligned} \tag{3.6}$$

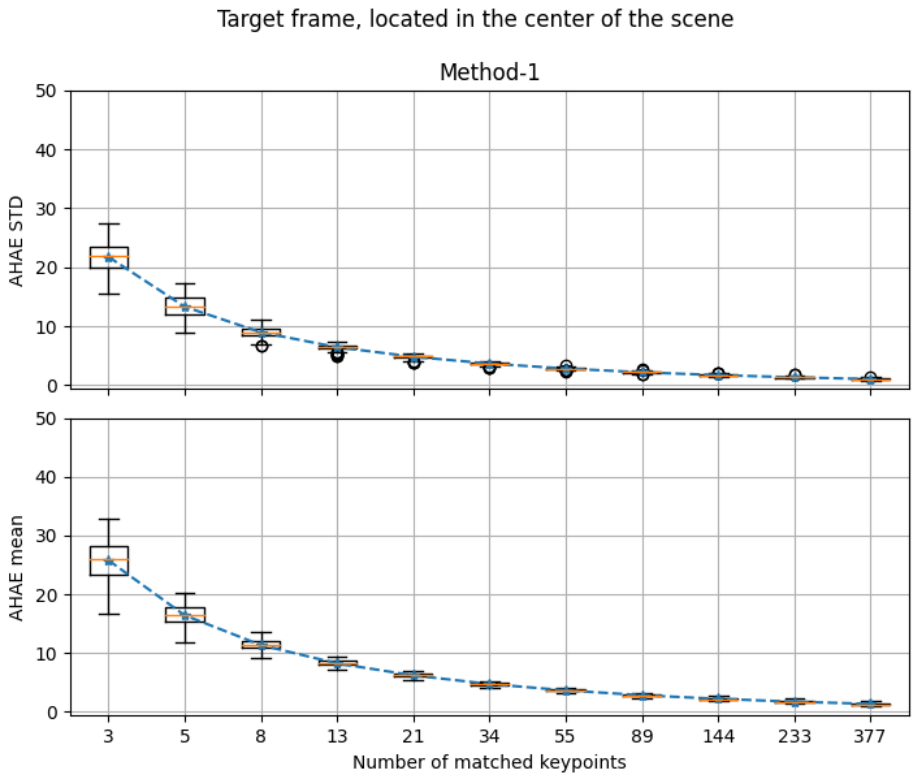


Figure 3.17: Method-1 AHAE Mean and Standard Deviations Box Plot

3.2.2.1 Target Frame, Located In the Center of the Scene

In this section of the analysis, target frame is located in the center of the screen in order to expand not only keypoint distribution around the frame and heading between robots to whole domain. Figure plotting and investigation procedure for the whole analysis is the same as the one in section 3.2.1. In more detail, while figures Figure 3.42, Figure 3.43, Figure 3.44, Figure 3.45, Figure 3.46 and Figure 3.47 are showing heading error standard deviations and means for each method, Figure 3.48 and Figure 3.49 dedicated to comparison of methods among each other.

It can be seen from Figure 3.48 and Figure 3.49 that standard deviations for both standard deviation of the DCPE and mean of DCPE for all scenes stay below 5 for all methods when the number of matched keypoints are increased. Moreover, combined methodologies method-5 and method-6 have better performance on the mean of DCPE standard deviation for all scenes when the number of matched keypoints is

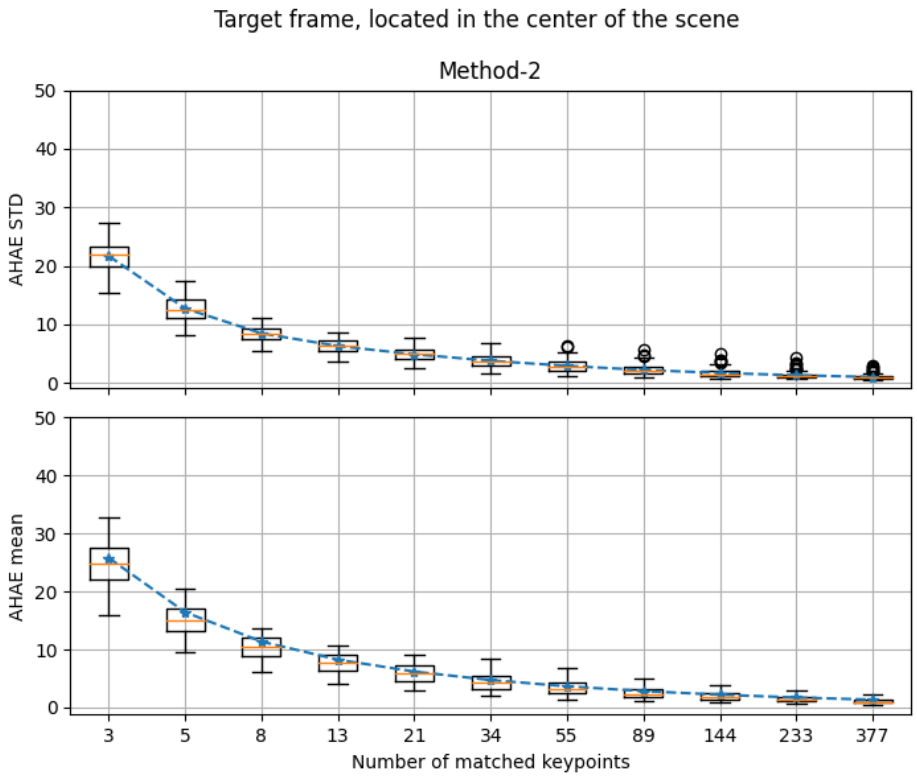


Figure 3.18: Method-2 AHAE Mean and Standard Deviations Box Plot

less than 13 as in Figure 3.48. In other words, combined methodologies heading error vary less than other methods at the low amount of matched keypoints. However, the gap between methodologies get closer to zero when the number of matched keypoint increases. Therefore, when the number of matched keypoints is higher than 34, the performances of algorithms are equivalent to each other when selected keypoints are distributed over the whole domain.

3.2.2.2 Target Frame, Located In the North of the Scene

In this section of the analysis, the target frame is located in the north of the screen in order to limit both keypoint distribution around the frame in [-180°, 0°] and heading between robots in [0°, 180°].

It can be seen from Figure 3.57 that method-4, method-5, and method-6 have better performance on the mean of the DCPE standard deviations for all scenes which is an

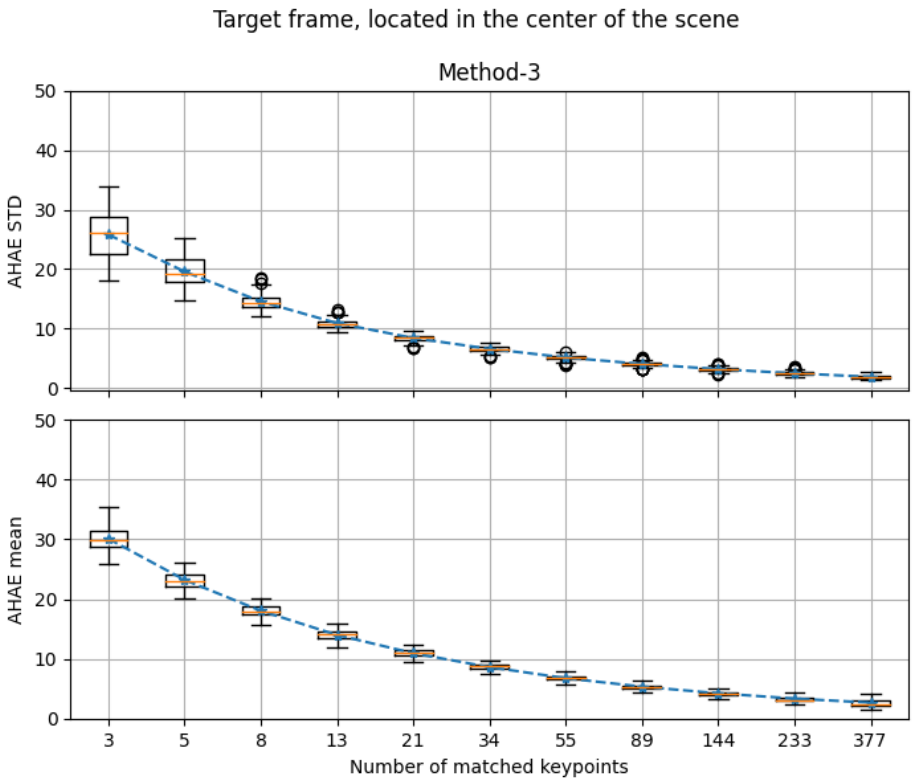


Figure 3.19: Method-3 AHAE Mean and Standard Deviations Box Plot

indication of less variation in terms of the DCPE. Furthermore, when keypoint distribution focuses on an interval and keypoints are not evenly distributed around the true heading, methods that have no priority on keypoints suffer from heading bias as in Figure 3.32 and this bias also affects the DCPE in Figure 3.58. In other words, if matched keypoints are bounded in an area, method-1, method-3 and method-5 are likely to address that area, and depart from the true heading as seen with wide boxes and whiskers in figures Figure 3.25, Figure 3.27 and Figure 3.29. However, even though the standard deviation of method-2 heading error in Figure 3.32 is one of the lowest methods, the DCPE which is labeled with a dashed orange line in Figure 3.58 does not reflect the same results. The main reason of this issue is explained in Figure 3.50. In the case of keypoints left in a region, the bisector heading vector might not route the robot towards the target frame. While the routing problem is compensated by assigning priority in method-2, priority assignment is not sufficient with the currently implemented exponential function when frame locations get closer to each other. This is because the change of longitude difference of keypoint pair is not visible as in Figure 3.14 when the current frame is in contact with the target frame. The

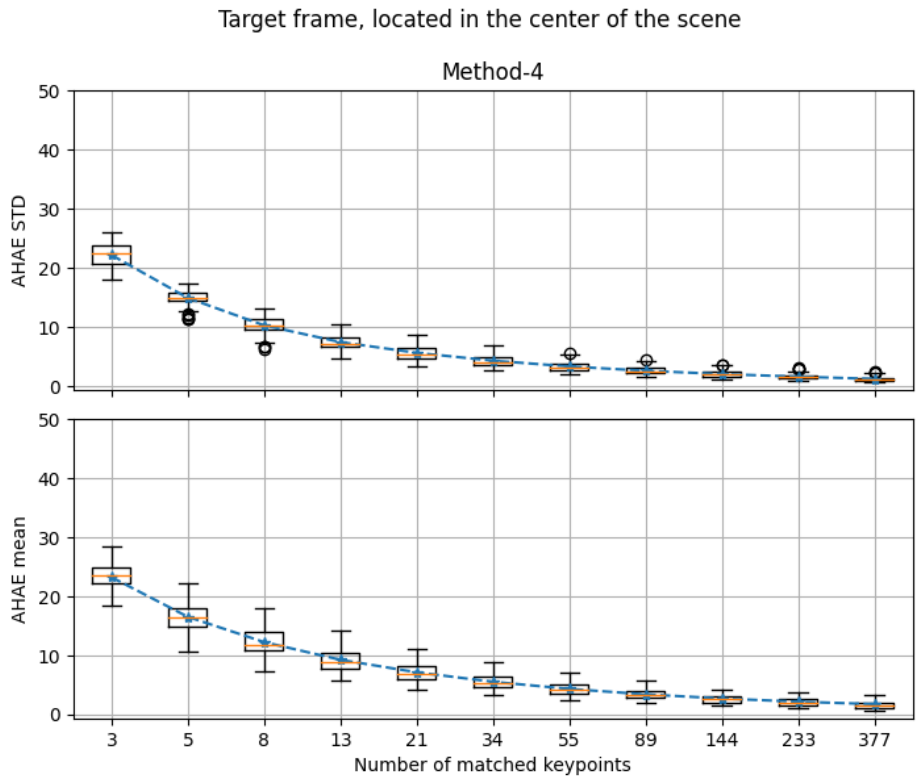


Figure 3.20: Method-4 AHAE Mean and Standard Deviations Box Plot

significance of this kind of issue increases with the limitation of keypoint distribution. In the previous analysis Section 3.2.1, where current frame iteration is not performed to reach the target frame, heading estimation is calculated for only one current frame location. Although an increase in the standard deviation of heading error means is detectable for method-1 which has no priority on keypoint pair heading vectors, it is not observable in method-2 results. Through the iterative process, all distance grids are covered between frames starting from the first distance, and failure case has been caught frequently when the current frame started arriving at the target frame.

3.2.2.3 Target Frame, Located In the North-East of the Scene

In this section of the analysis, the target frame is located in the north-east of the screen in order to limit both keypoint distribution around the frame in [-180°, -90°] and heading between robots in [0°, 90°].

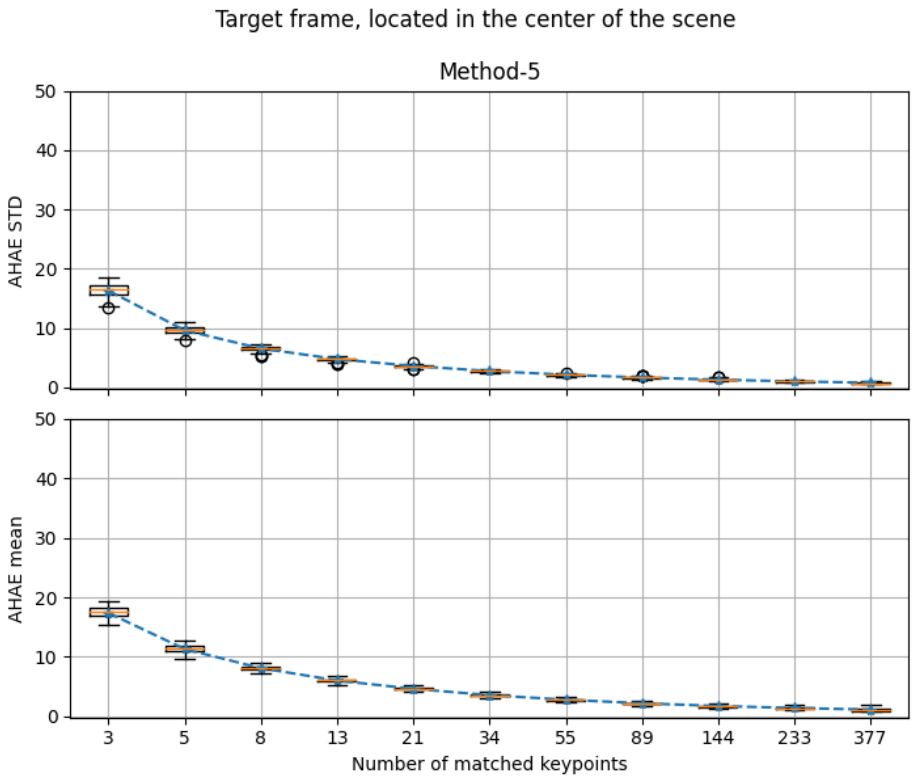


Figure 3.21: Method-5 AHAE Mean and Standard Deviations Box Plot

The most vital outcome that we detected in the results are the divergent behavior of method-1, method-2, method-5 and method-6 in both Figure 3.65 and Figure 3.66. When keypoint distribution is bounded in a narrow interval and keypoints are not evenly distributed around the true heading as in this case of the analysis, methods that have heading unit vector calculation based on paired keypoints suffer from the problem Figure 3.50. In more detail, in the case of having matched keypoints that are localized in an area, method-1, method-2, method-5, and method-6 tend to focus on that area, hence, deviating from true heading and resulting in high distance covered percent error. As an example, tracked path by the robot with the supervision of method-1 is drawn in Figure 3.67 and Figure 3.68. While overall performance is shown on the left side of the figures, the right side is dedicated to zooming in on the failure region. It can be easily detected that when keypoints are bounded in a region, paired keypoint bisector heading vectors reduce system performance. After some point, it might cause the robot to make only back-and-forth movements instead of moving towards the target frame as in Figure 3.50. Even though there is a performance downgrade in method-4 compared to other target frame location analyses,

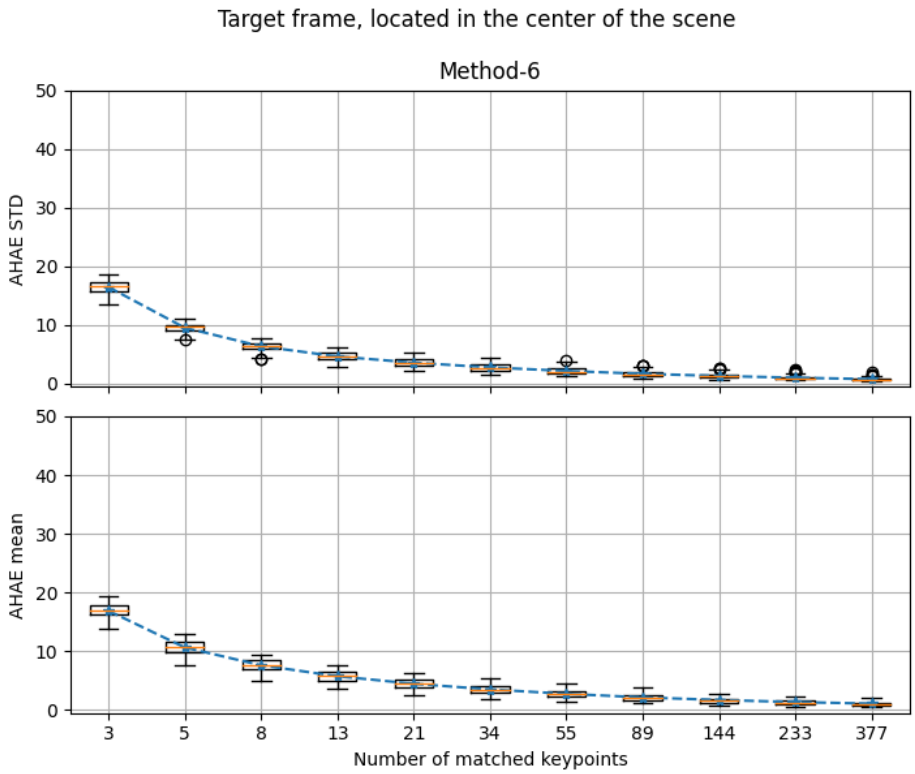


Figure 3.22: Method-6 AHAE Mean and Standard Deviations Box Plot

method-4 has significantly better performance when the target frame is placed at the north-east of the scene as seen in method-wise comparison figures Figure 3.66 and Figure 3.65. Therefore, it is sufficient to say that method-4 is the most robust heading estimation method in terms of keypoint distribution randomness. However, the only covered keypoint priority function for both method-2 and method-4 is the exponential function, and the analysis results might be improved by using a more appropriate one. In summary, method-4 will be the heading estimation methodology that will be used in the implementation phase of the thesis, and the matched number of keypoints will be kept above 60 after which heading estimation performance is not significantly affected. Since mAP results for SuperPoint are above 0.40 in most of the cases in Local Feature Matching Algorithm Comparison analyses results, the maximum amount of keypoints that will be generated in the keypoint extraction process will be limited to 150.

As a summary in this chapter, the local feature matching algorithm combination and the heading estimation method are determined for further usage. In addition, while

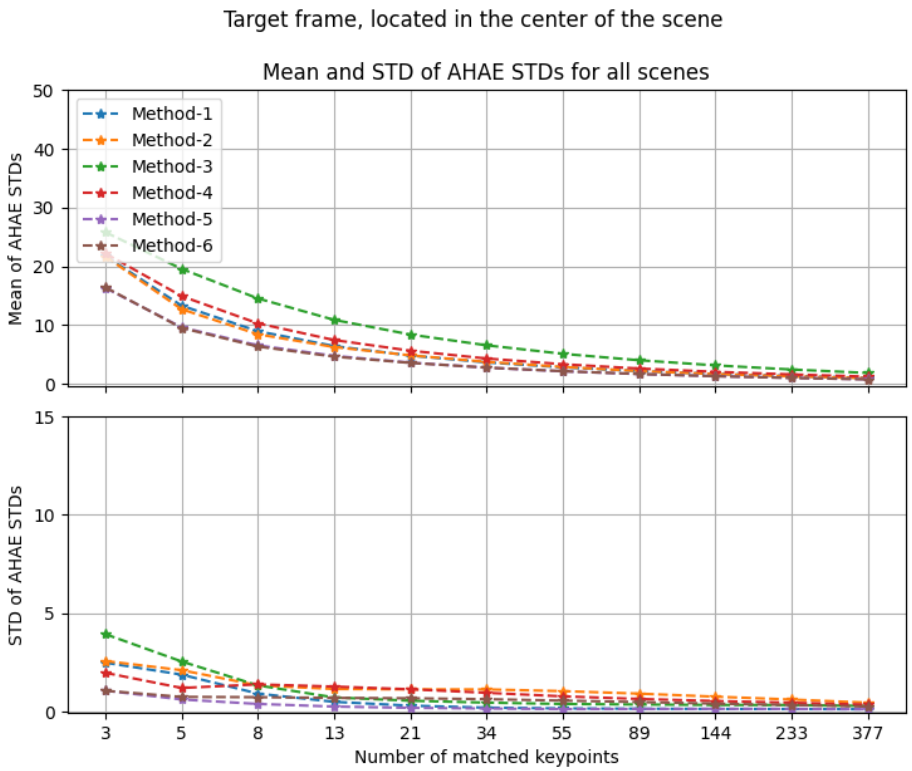


Figure 3.23: Mean and Standard Deviation of AHAE Standard Deviations Comparison

the maximum number of keypoints that will be generated from a frame is limited to 150, the optimum number of matched keypoints, after which no increment is achieved concerning the navigation performance, is specified as 60. In the next chapter, the problem will be investigated from the coding perspective and will be put into practice in more detail.

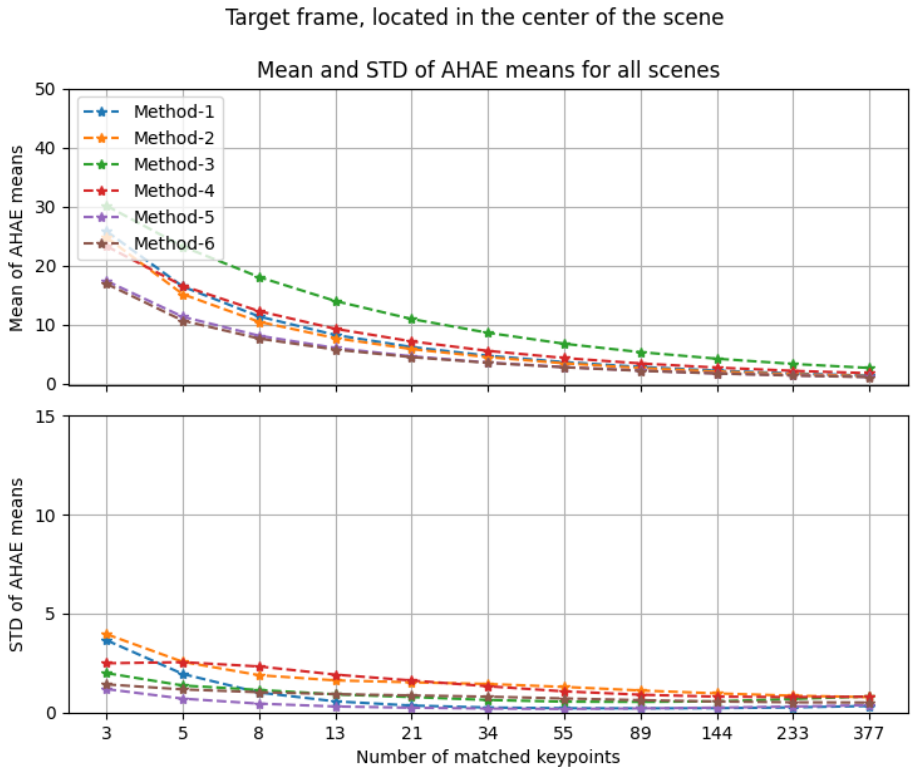


Figure 3.24: Mean and Standard Deviation of AHAE Means Comparison

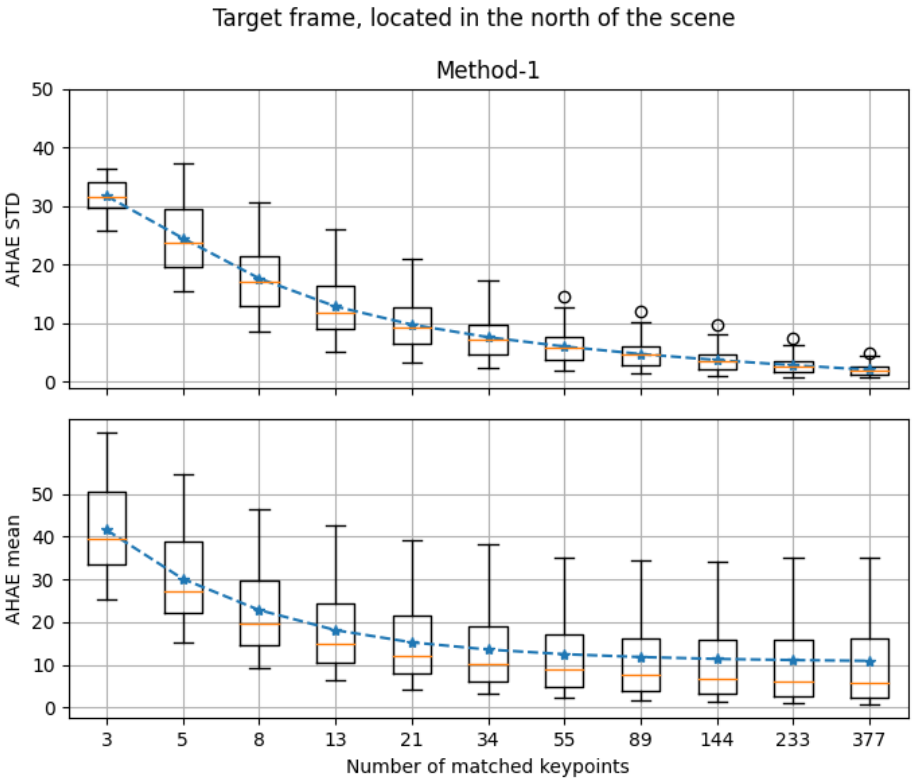


Figure 3.25: Method-1 AHAE Mean and Standard Deviations Box Plot

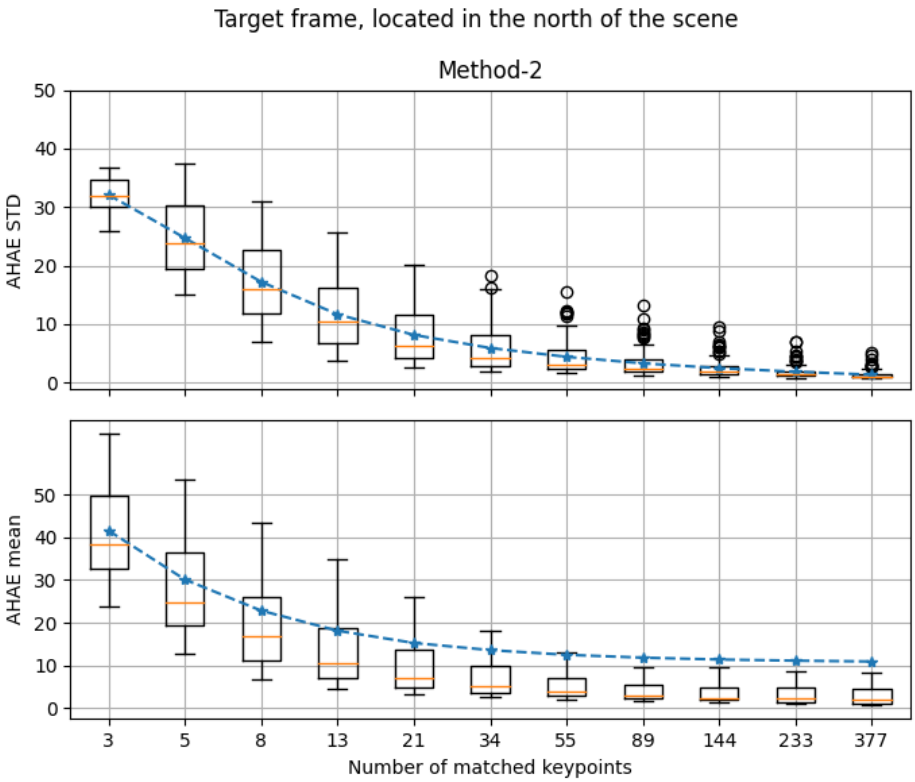


Figure 3.26: Method-2 AHAE Mean and Standard Deviations Box Plot

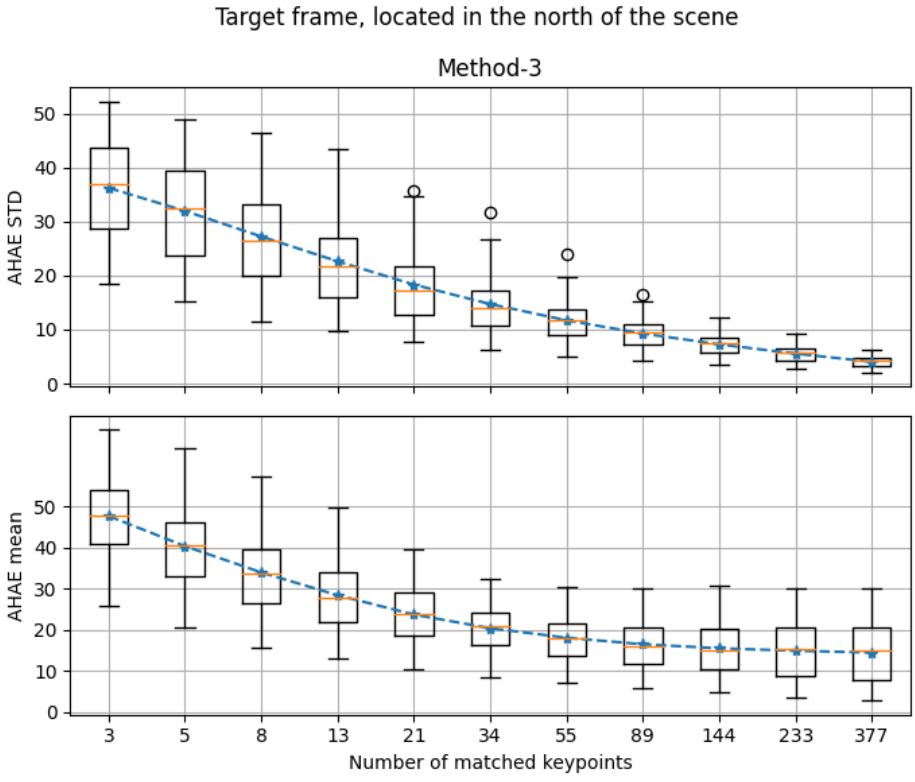


Figure 3.27: Method-3 AHAE Mean and Standard Deviations Box Plot

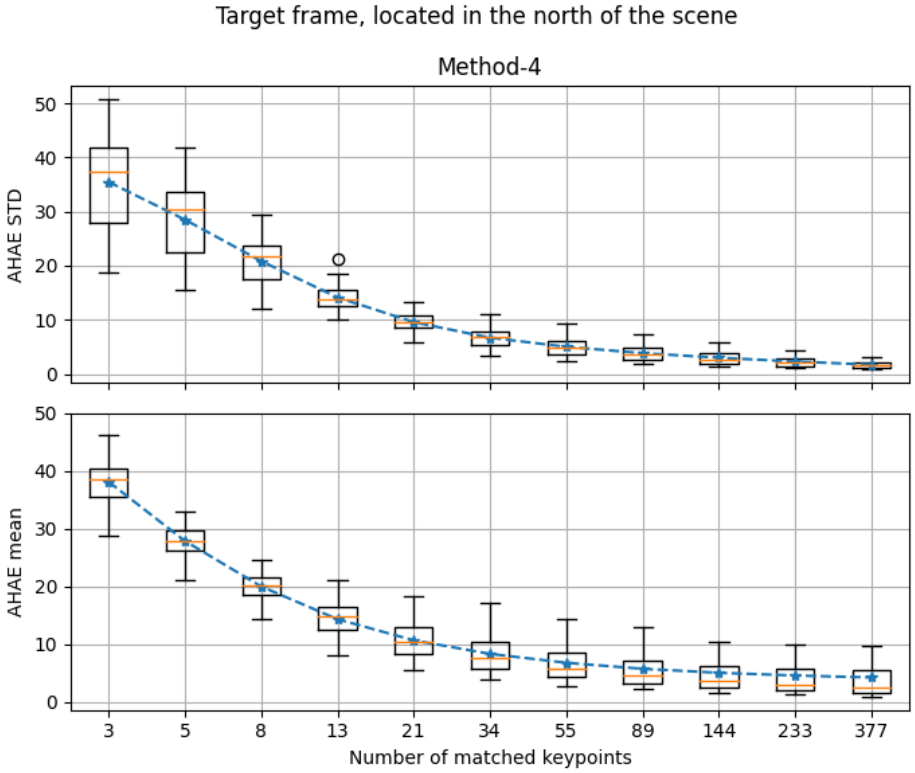


Figure 3.28: Method-4 AHAE Mean and Standard Deviations Box Plot

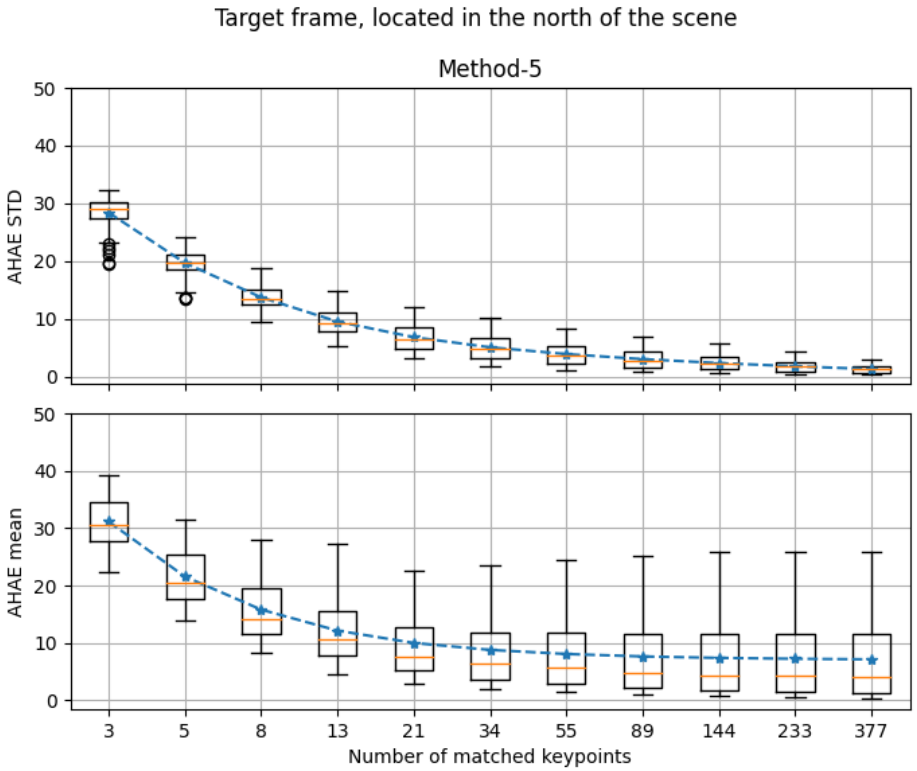


Figure 3.29: Method-5 AHAE Mean and Standard Deviations Box Plot

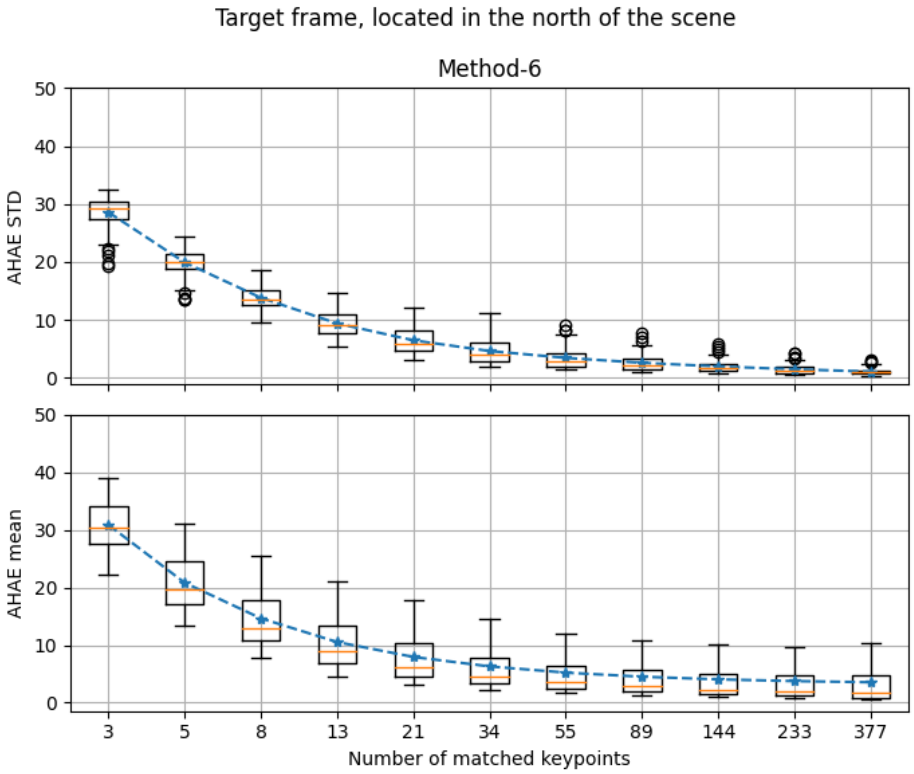


Figure 3.30: Method-6 AHAE Mean and Standard Deviations Box Plot

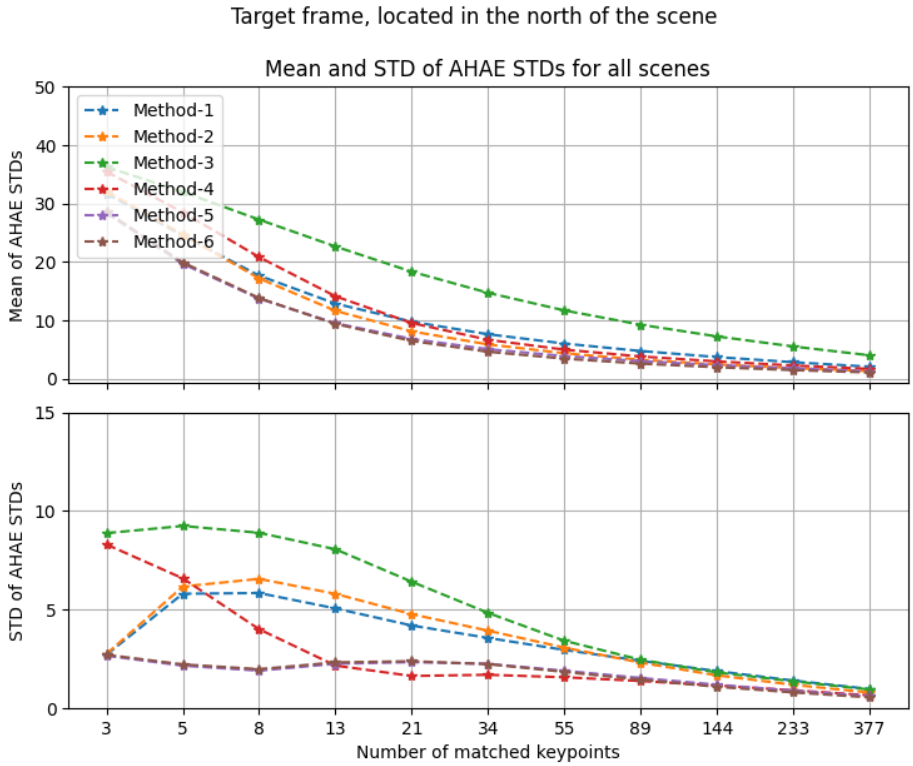


Figure 3.31: Mean and Standard Deviation of AHAE Standard Deviations Comparison

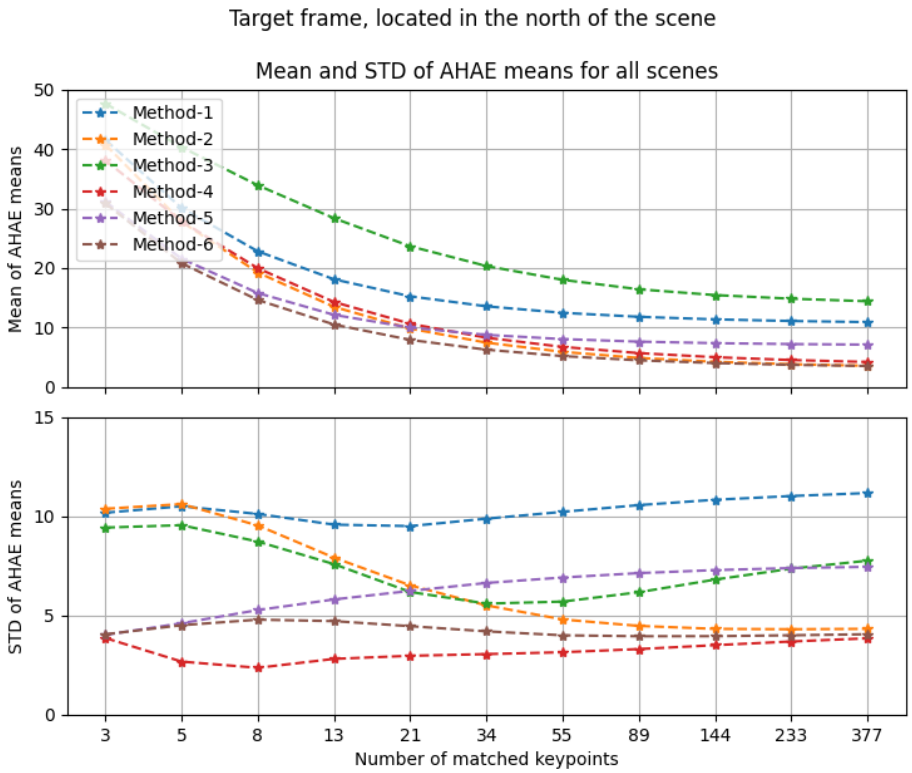


Figure 3.32: Mean and Standard Deviation of AHAE Means Comparison

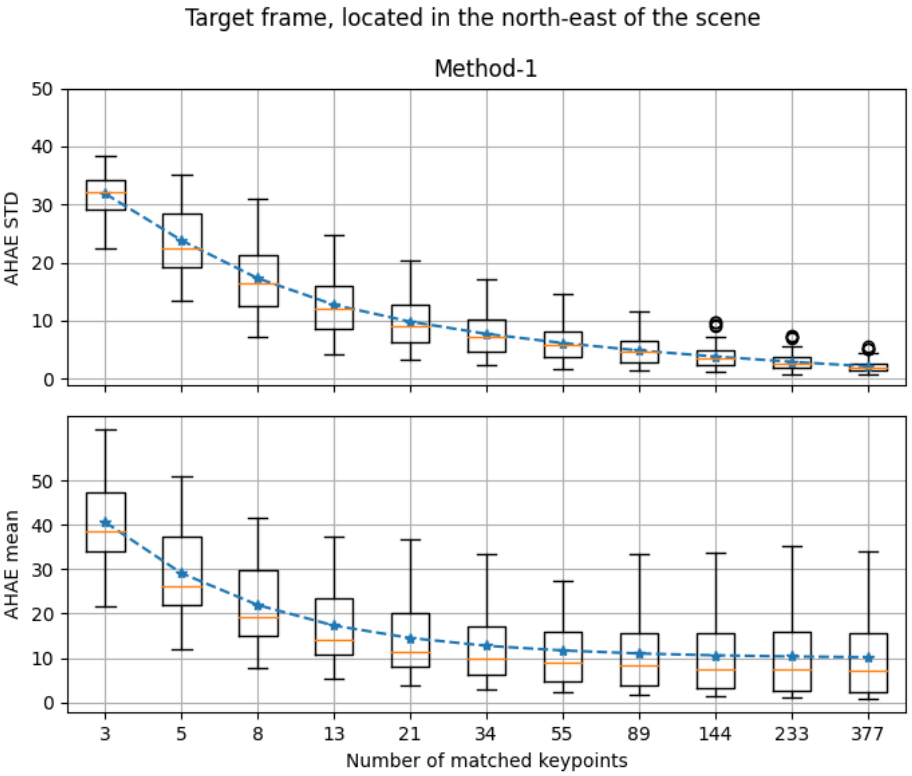


Figure 3.33: Method-1 AHAE Mean and Standard Deviations Box Plot

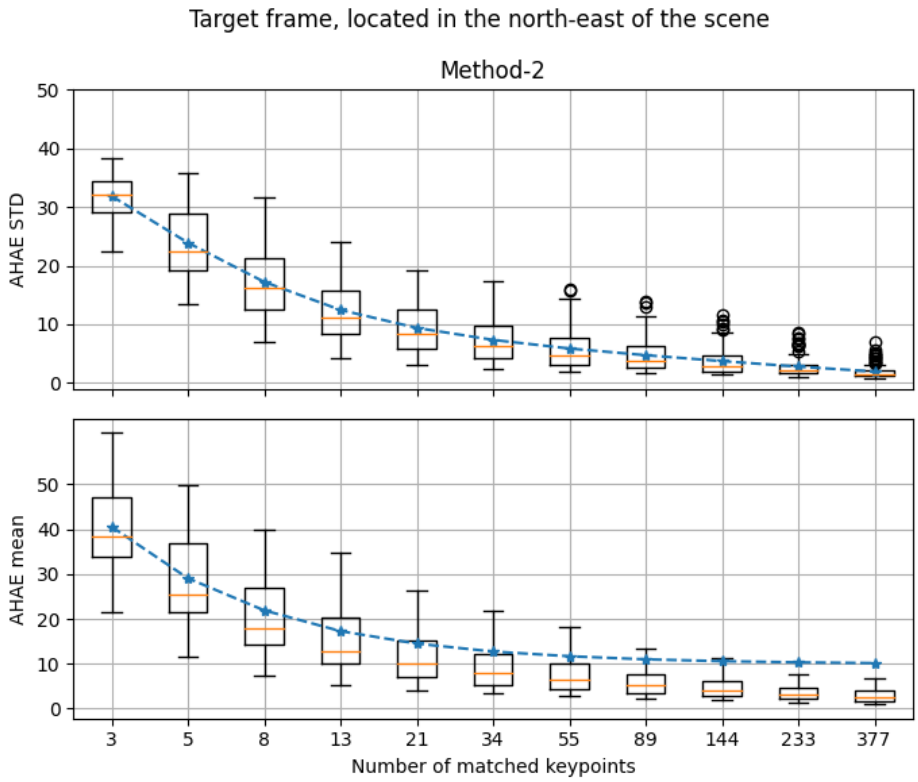


Figure 3.34: Method-2 AHAE Mean and Standard Deviations Box Plot

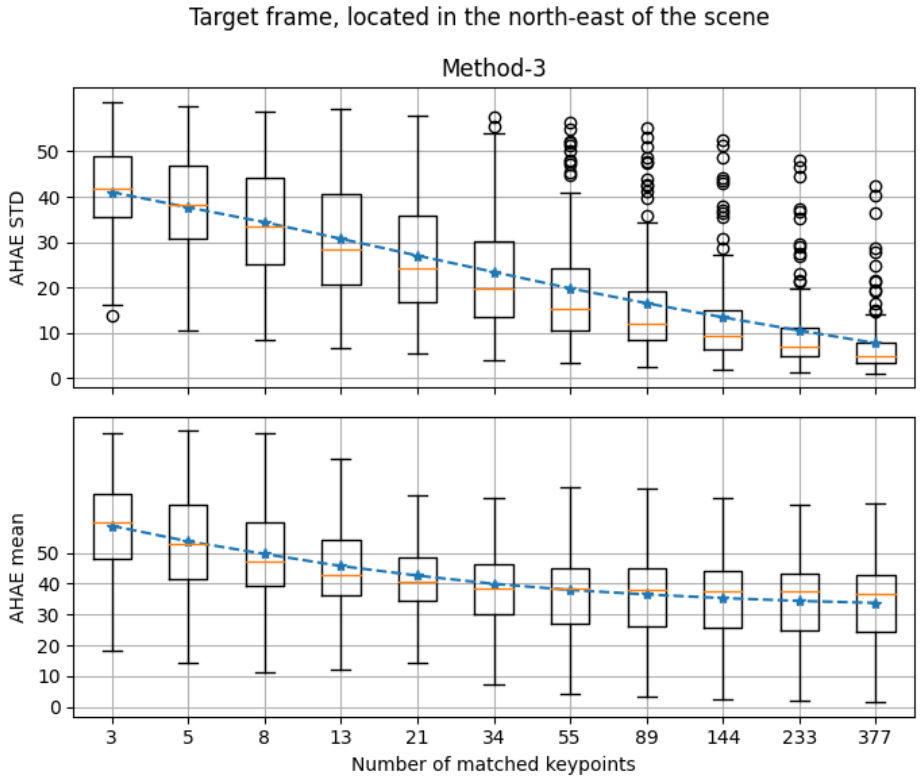


Figure 3.35: Method-3 AHAE Mean and Standard Deviations Box Plot

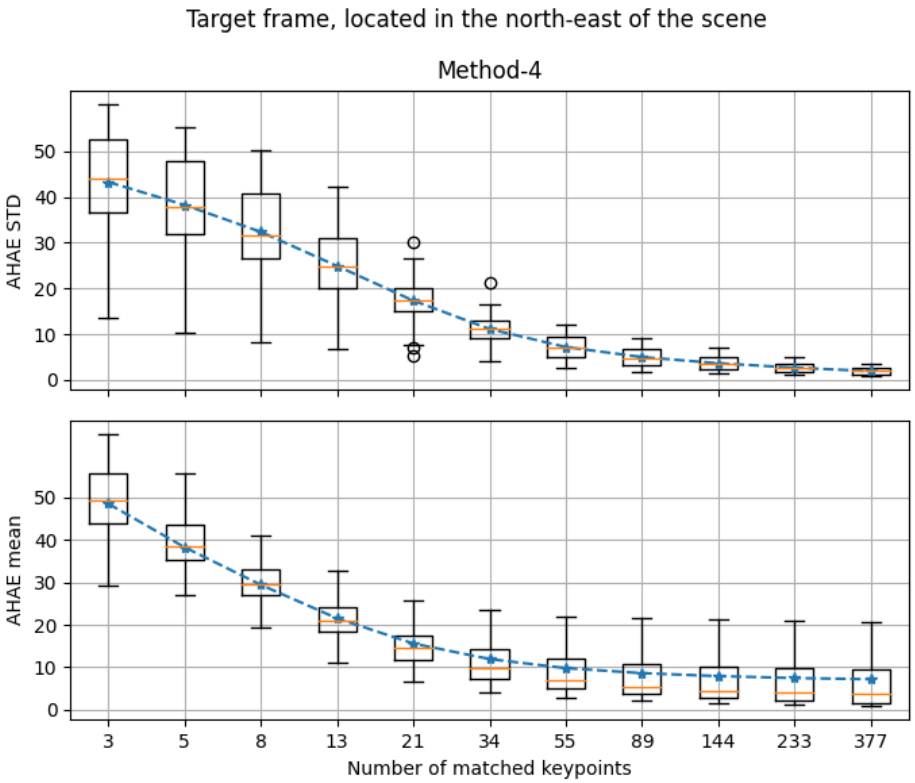


Figure 3.36: Method-4 AHAE Mean and Standard Deviations Box Plot

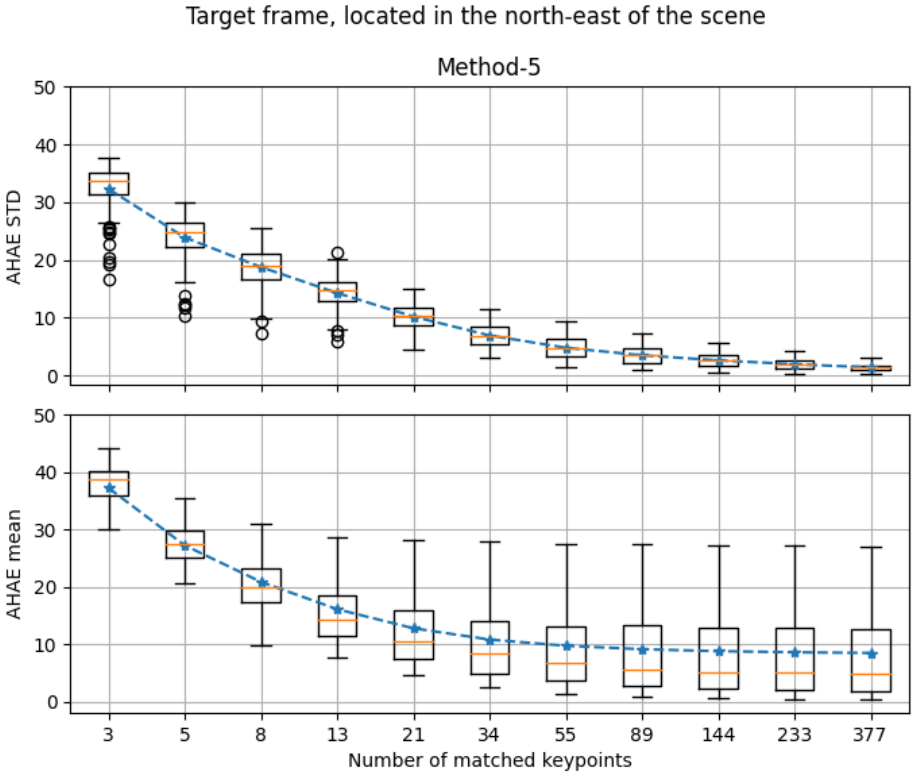


Figure 3.37: Method-5 AHAE Mean and Standard Deviations Box Plot

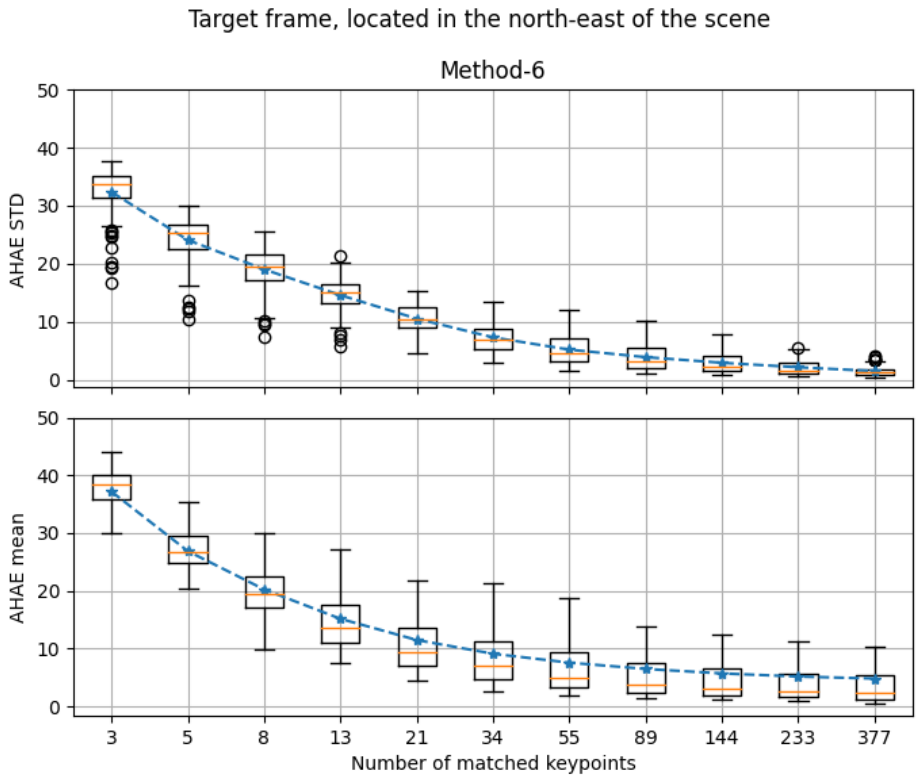


Figure 3.38: Method-6 AHAE Mean and Standard Deviations Box Plot

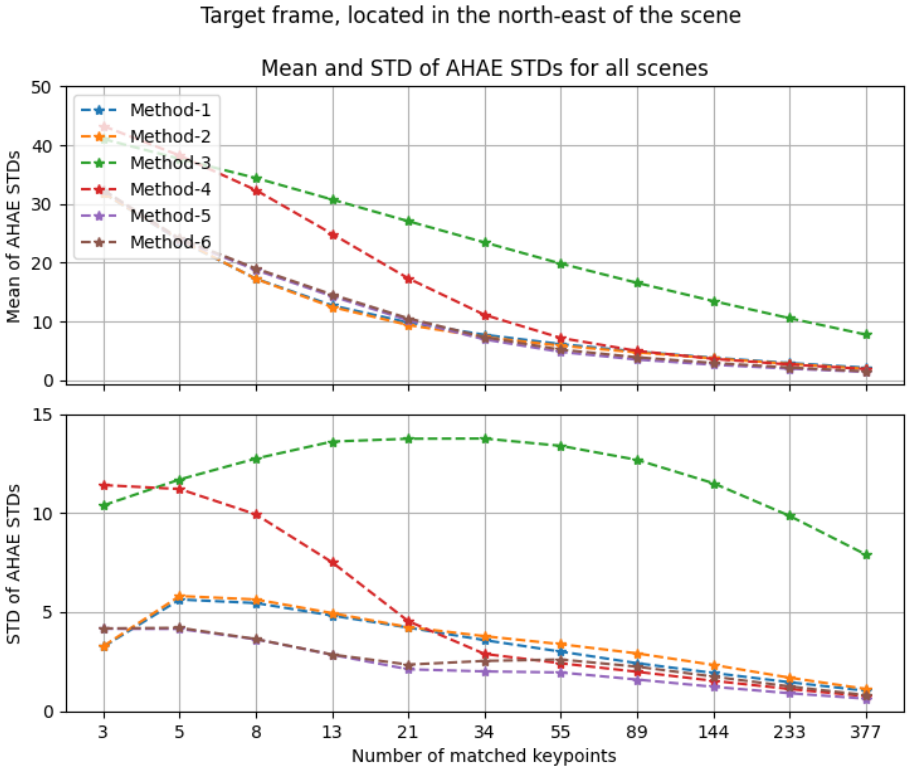


Figure 3.39: Mean and Standard Deviation of AHAE Standard Deviations Comparison

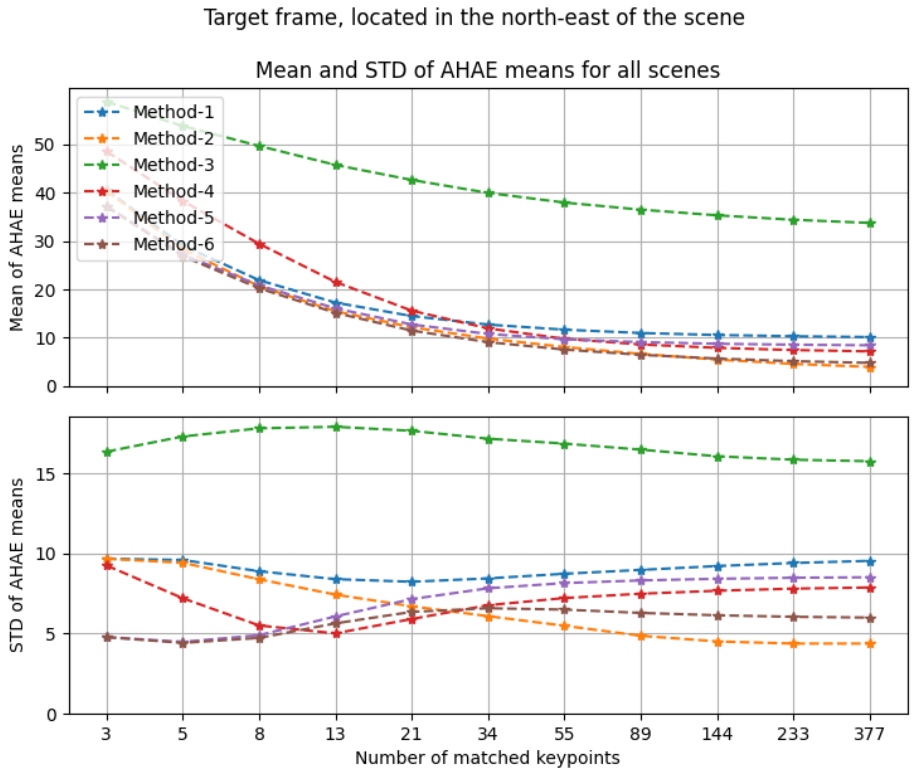


Figure 3.40: Mean and Standard Deviation of AHAE Means Comparison

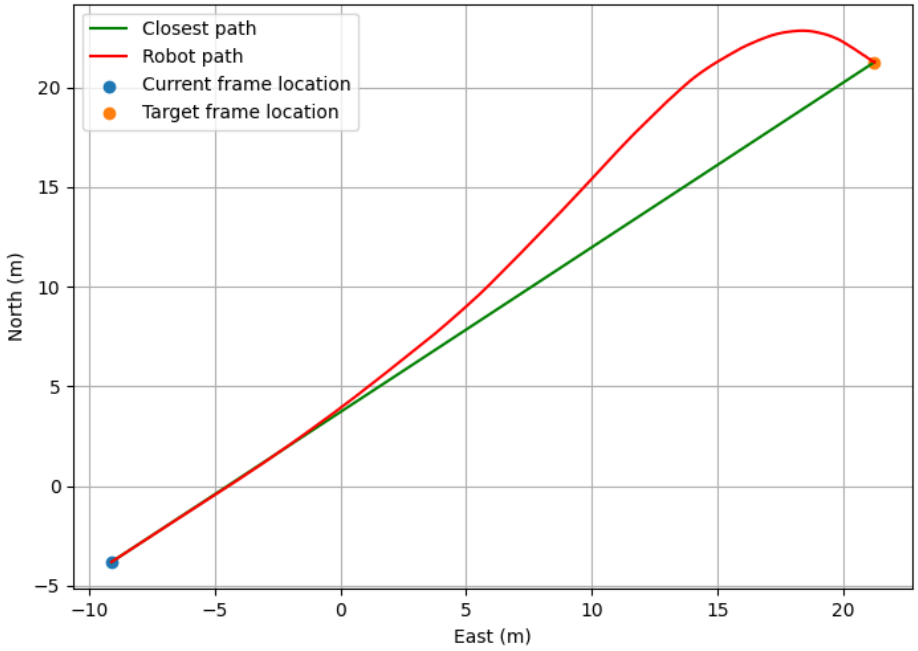


Figure 3.41: Sample Monte Carlo Simulation Path Tracking

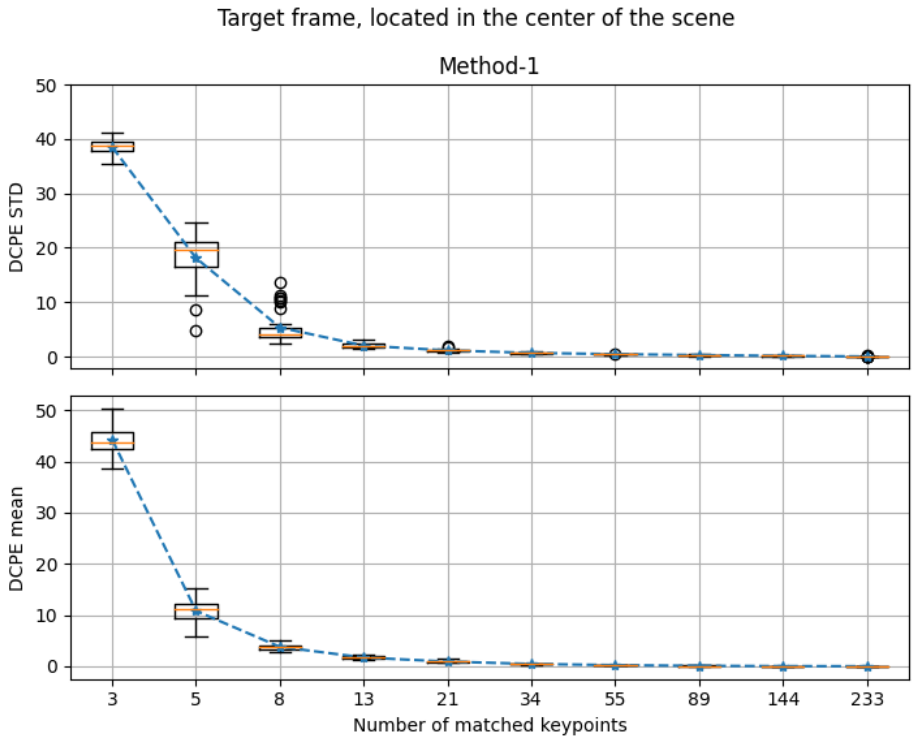


Figure 3.42: Method-1 DCPE Mean and Standard Deviations Box Plot

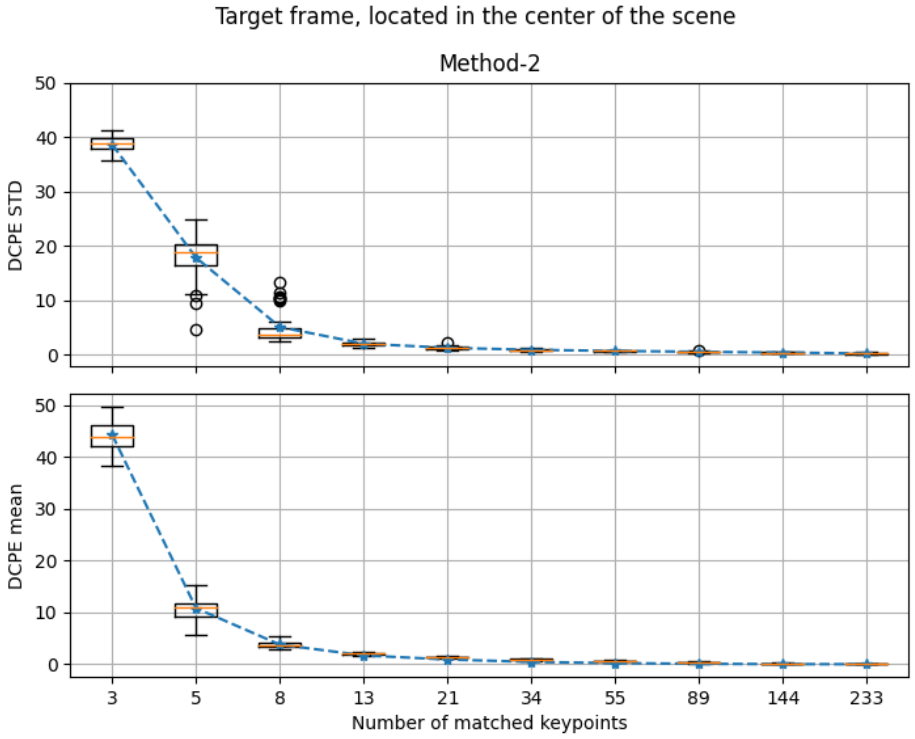


Figure 3.43: Method-2 DCPE Mean and Standard Deviations Box Plot

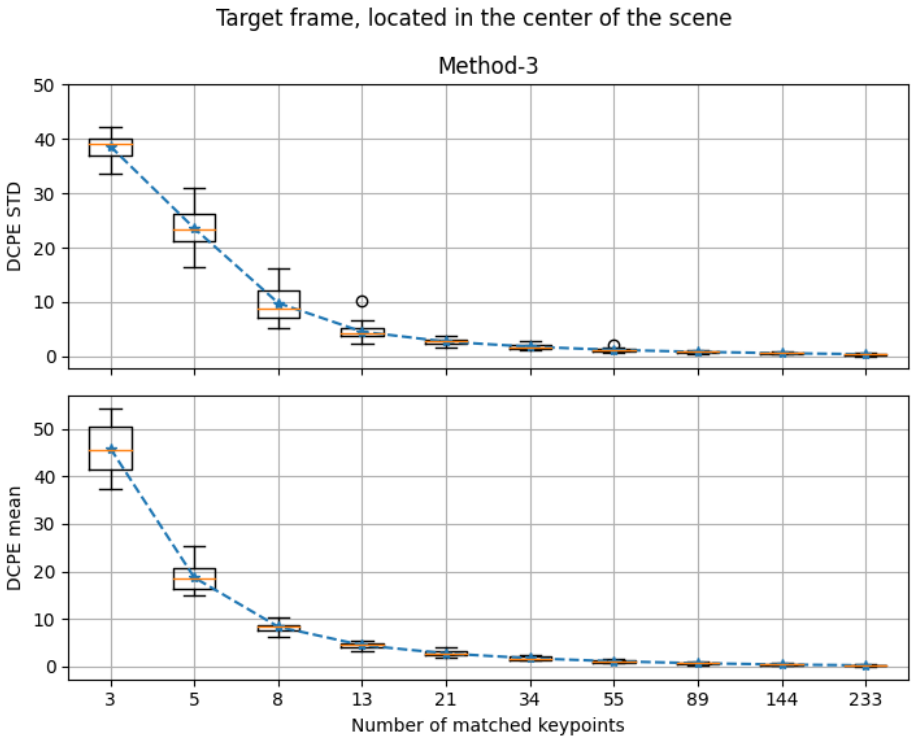


Figure 3.44: Method-3 DCPE Mean and Standard Deviations Box Plot

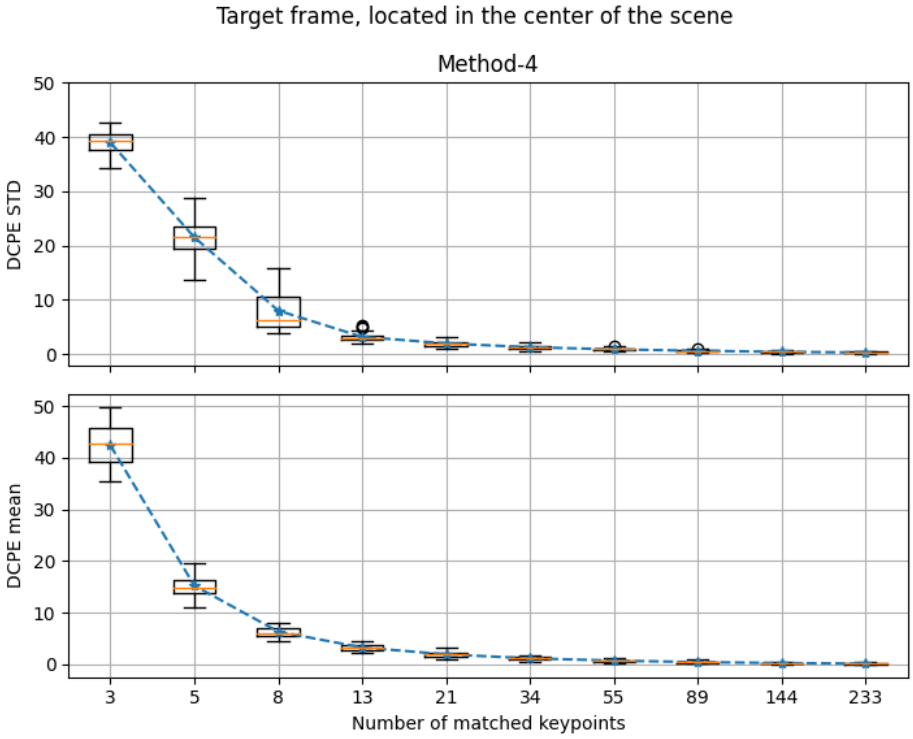


Figure 3.45: Method-4 DCPE Mean and Standard Deviations Box Plot

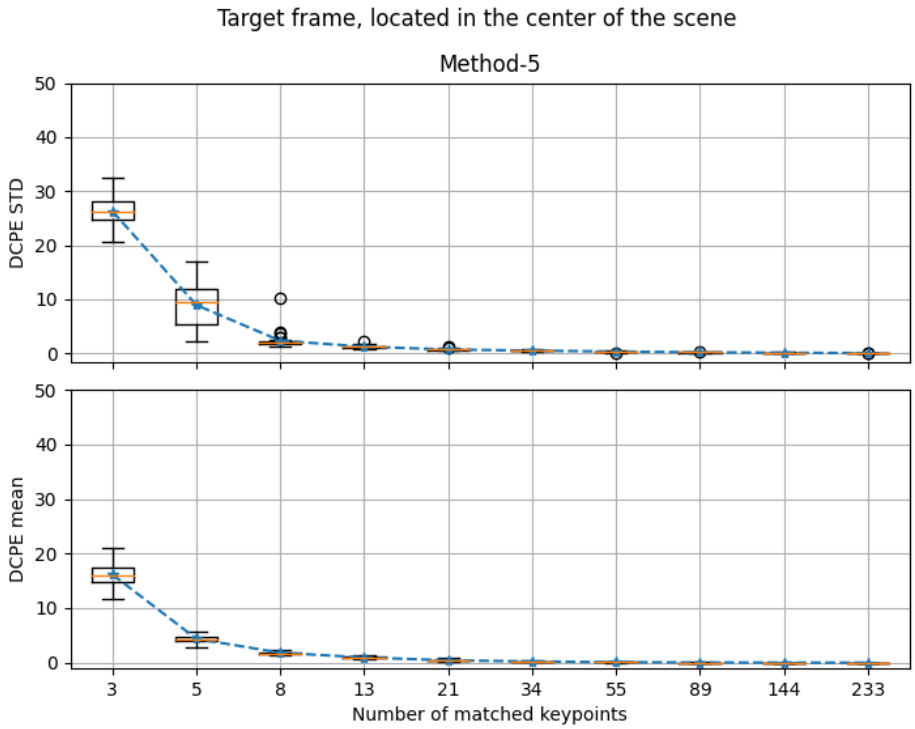


Figure 3.46: Method-5 DCPE Mean and Standard Deviations Box Plot

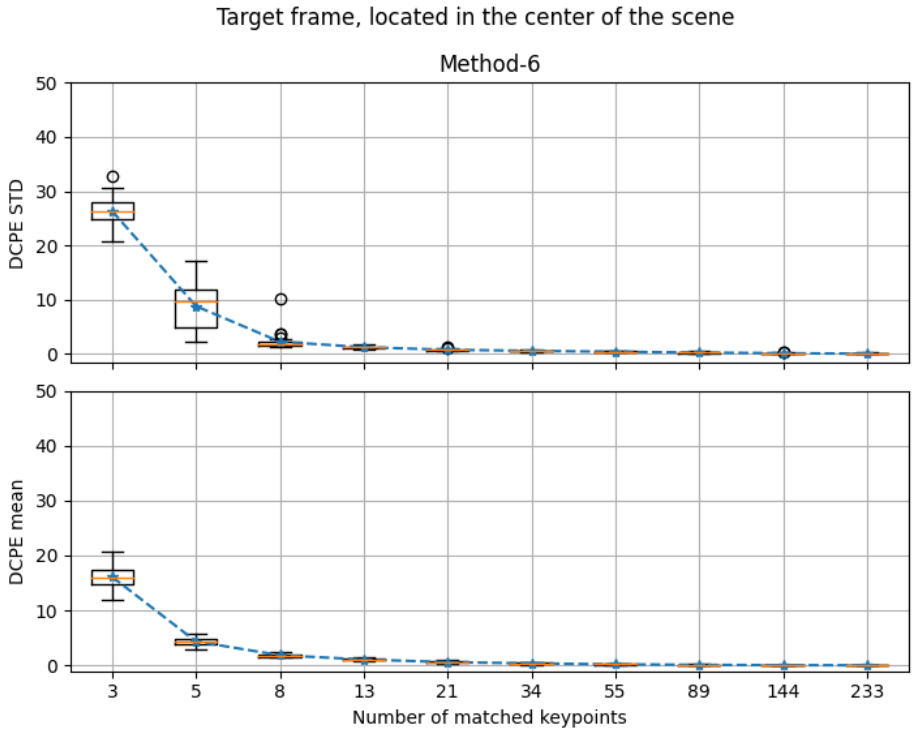


Figure 3.47: Method-6 DCPE Mean and Standard Deviations Box Plot

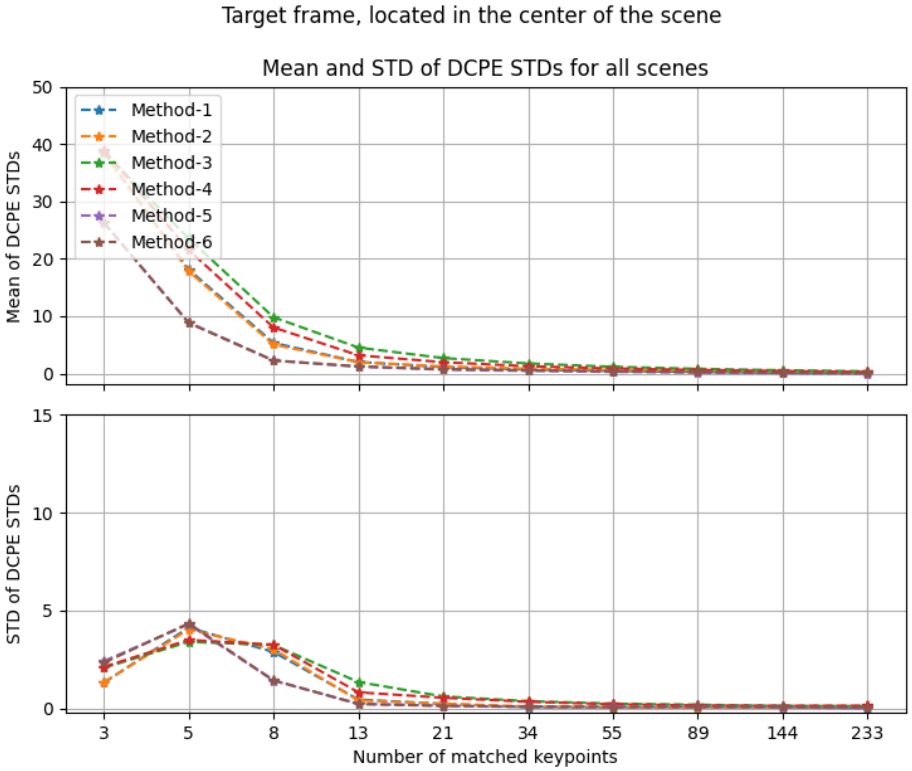


Figure 3.48: Mean and Standard Deviation of DCPE Standard Deviations Comparison

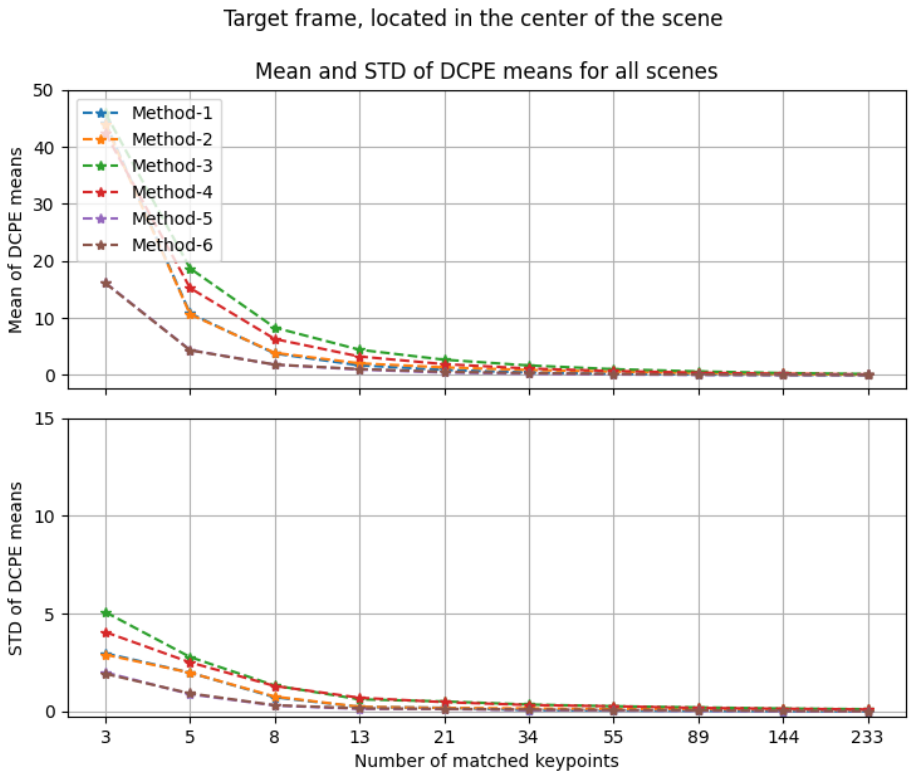


Figure 3.49: Mean and Standard Deviation of DCPE Means Comparison

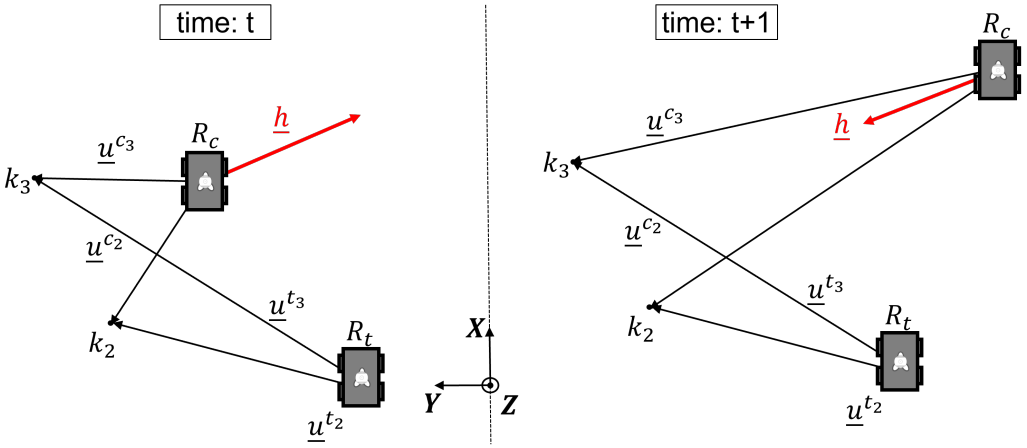


Figure 3.50: Heading Estimation Based on Pairing Sample Failure

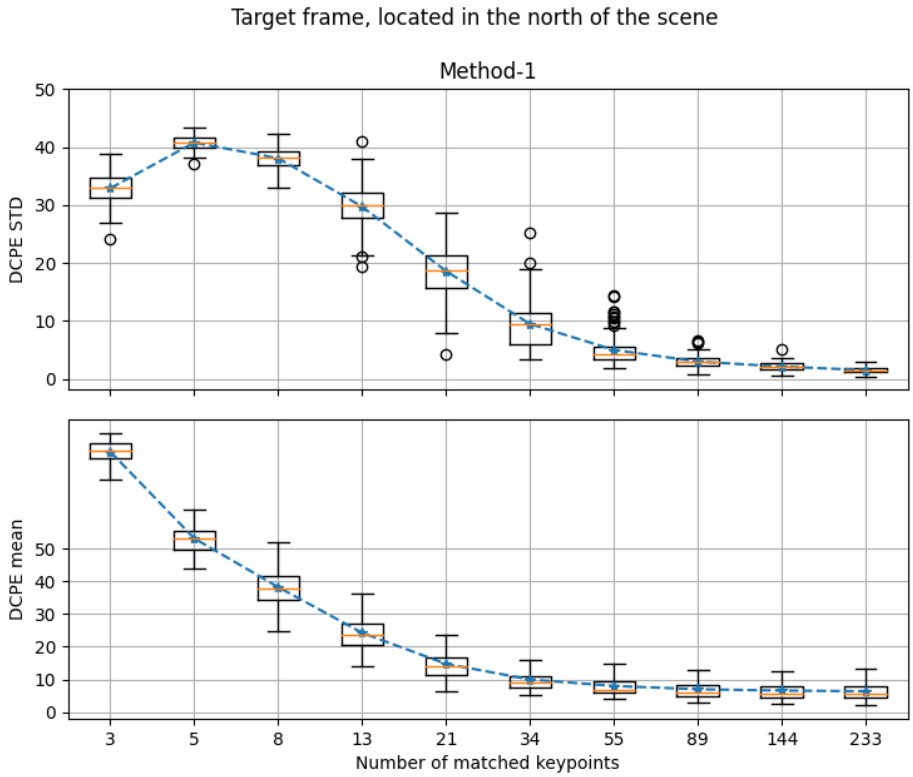


Figure 3.51: Method-1 DCPE Mean and Standard Deviations Box Plot

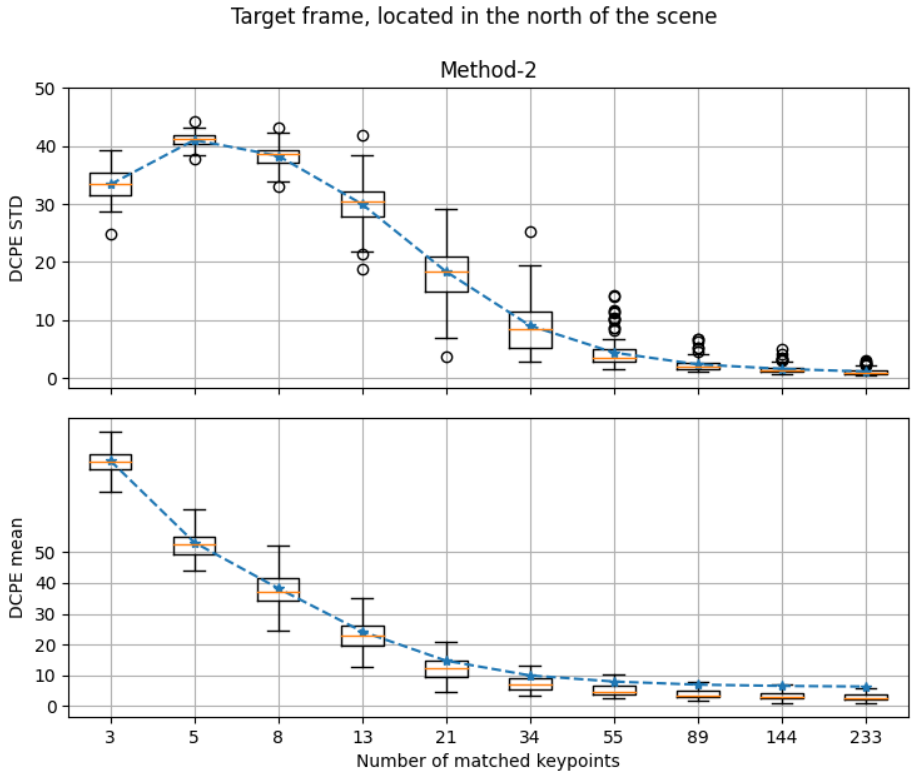


Figure 3.52: Method-2 DCPE Mean and Standard Deviations Box Plot

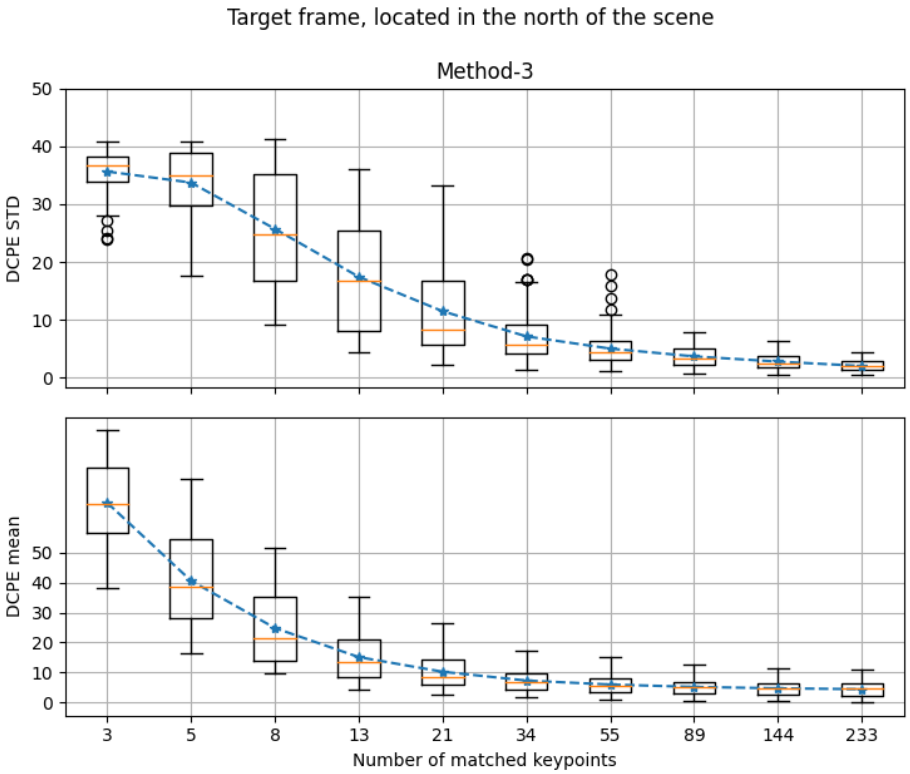


Figure 3.53: Method-3 DCPE Mean and Standard Deviations Box Plot

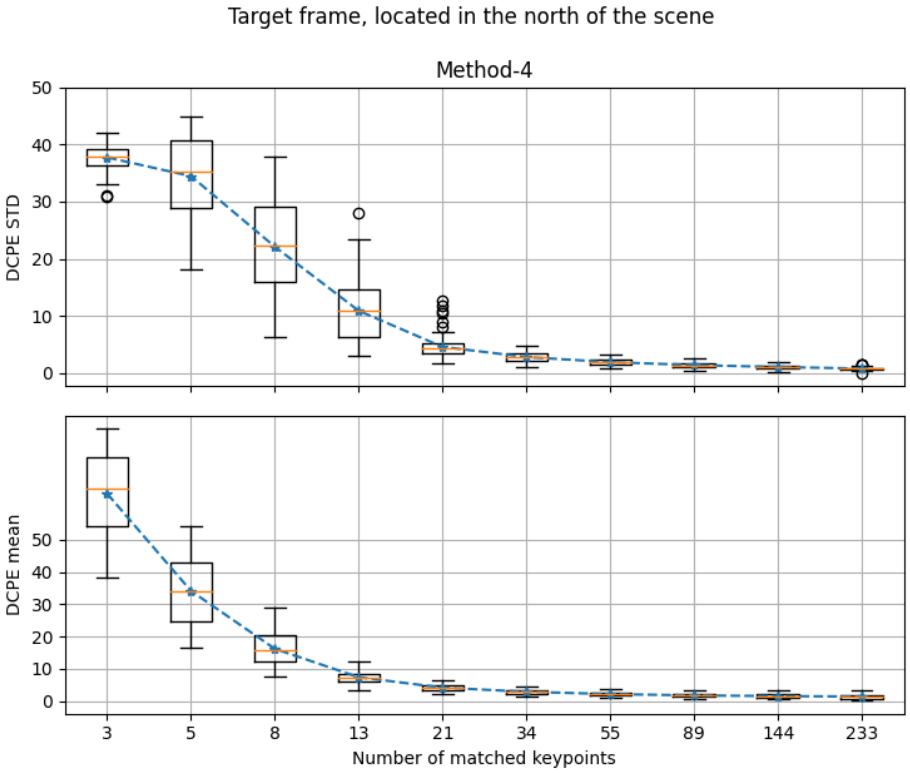


Figure 3.54: Method-4 DCPE Mean and Standard Deviations Box Plot

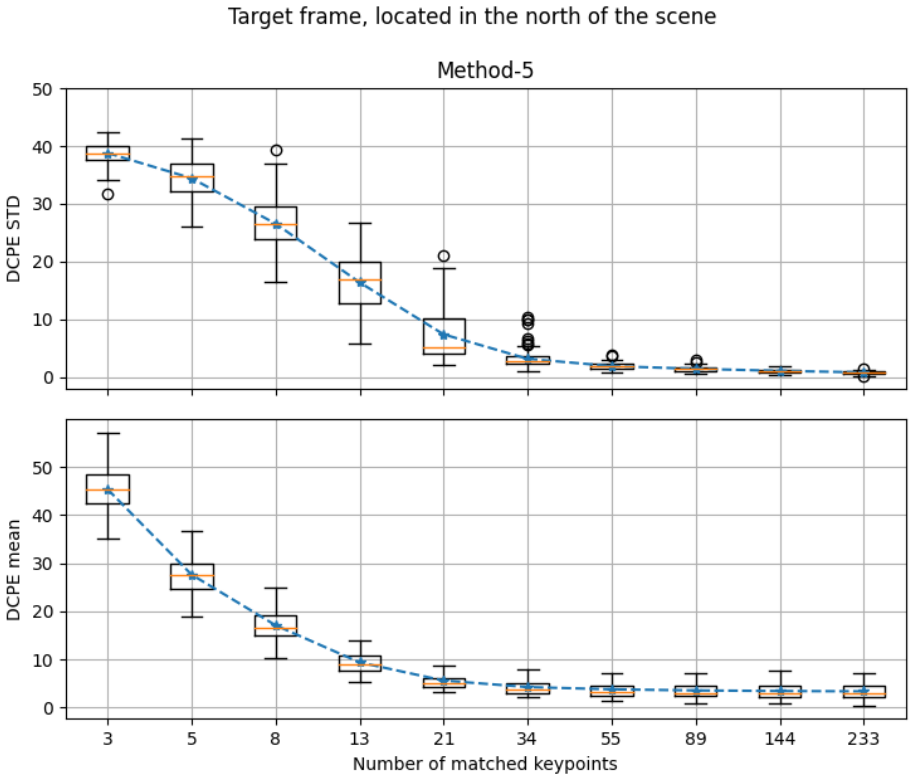


Figure 3.55: Method-5 DCPE Mean and Standard Deviations Box Plot

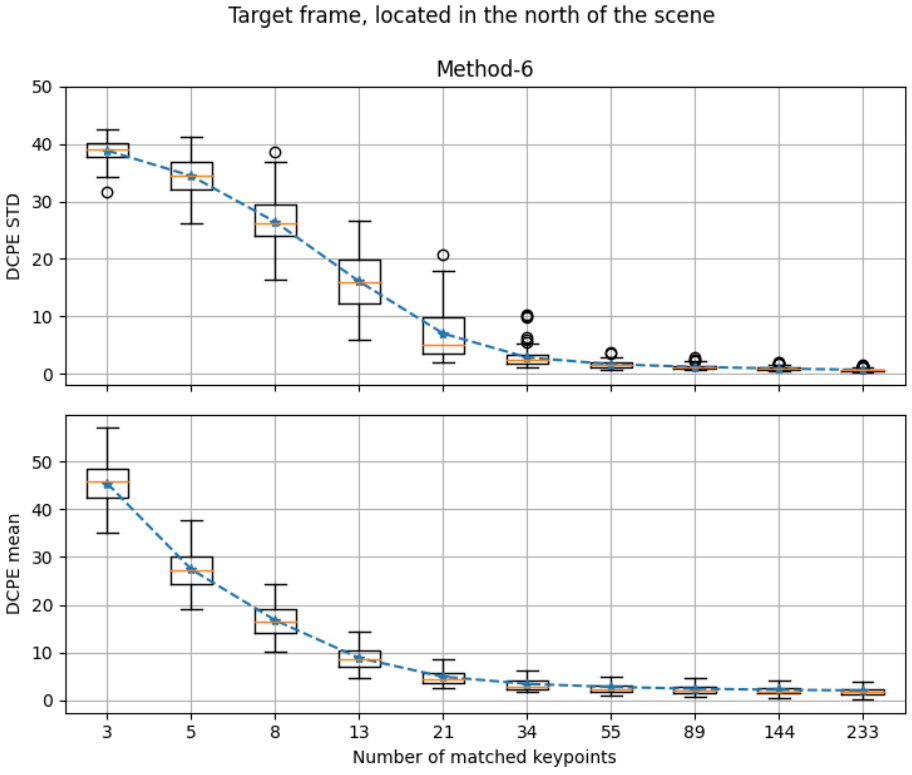


Figure 3.56: Method-6 DCPE Mean and Standard Deviations Box Plot

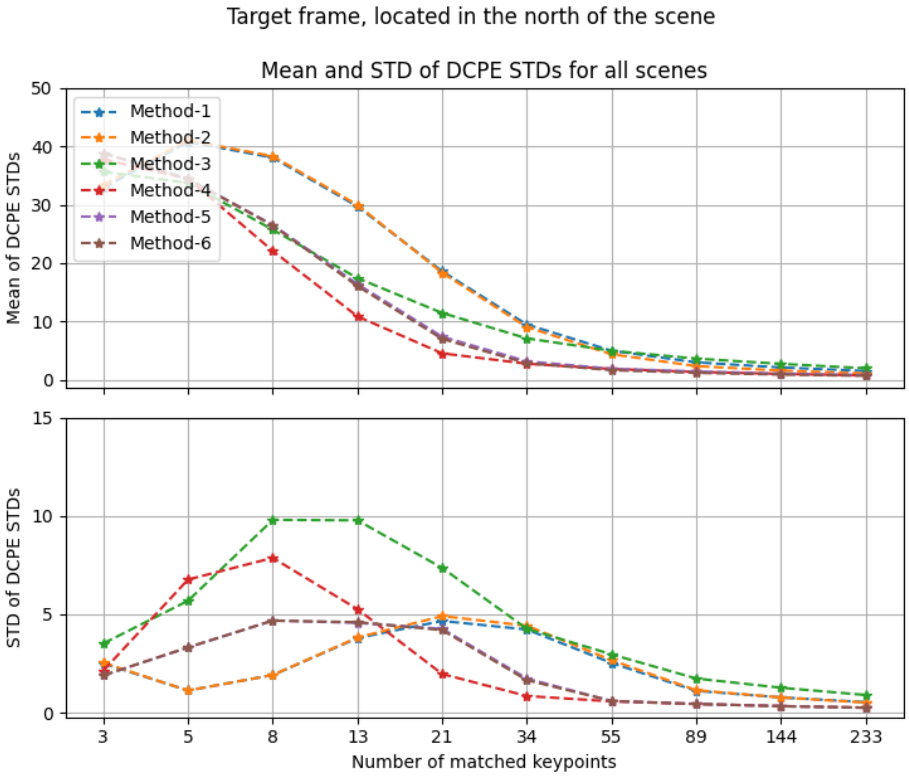


Figure 3.57: Mean and Standard Deviation of DCPE Standard Deviations Comparison

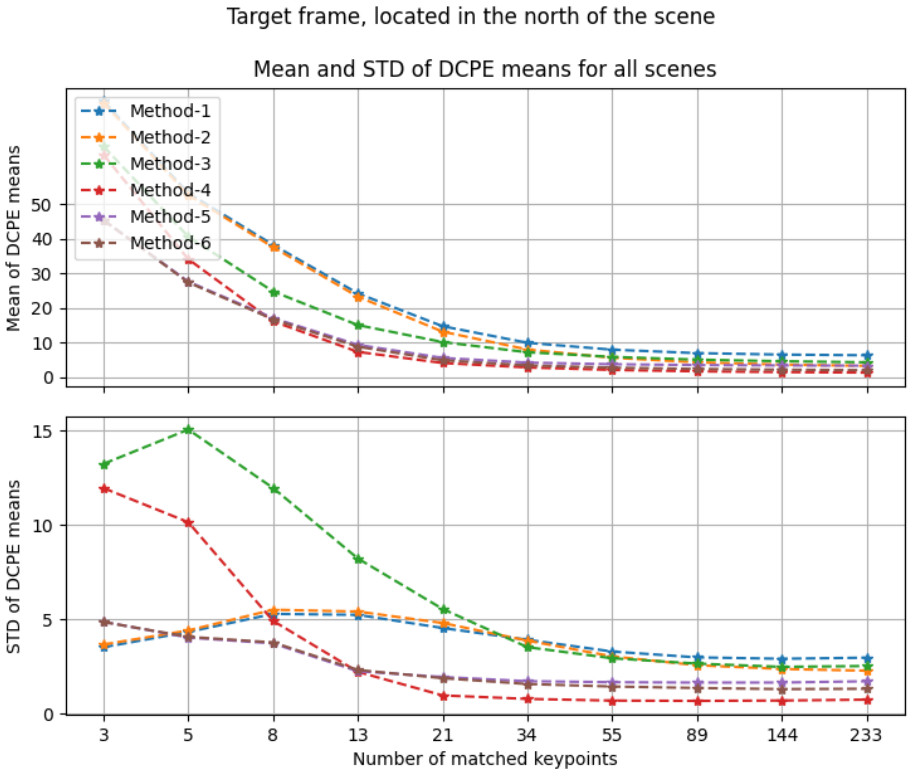


Figure 3.58: Mean and Standard Deviation of DCPE Means Comparison

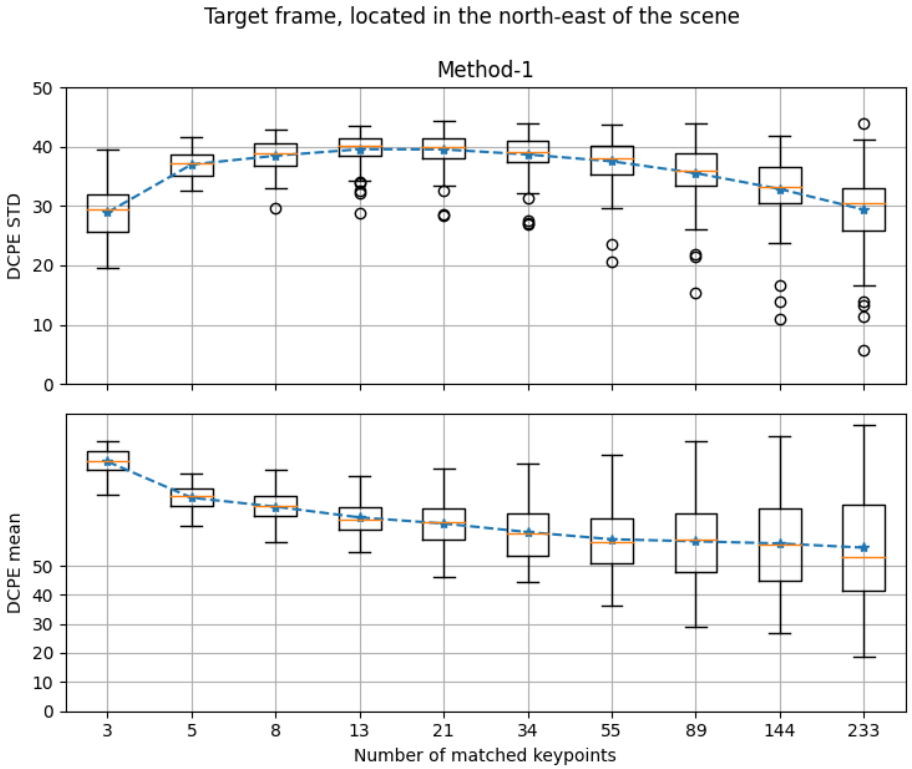


Figure 3.59: Method-1 DCPE Mean and Standard Deviations Box Plot

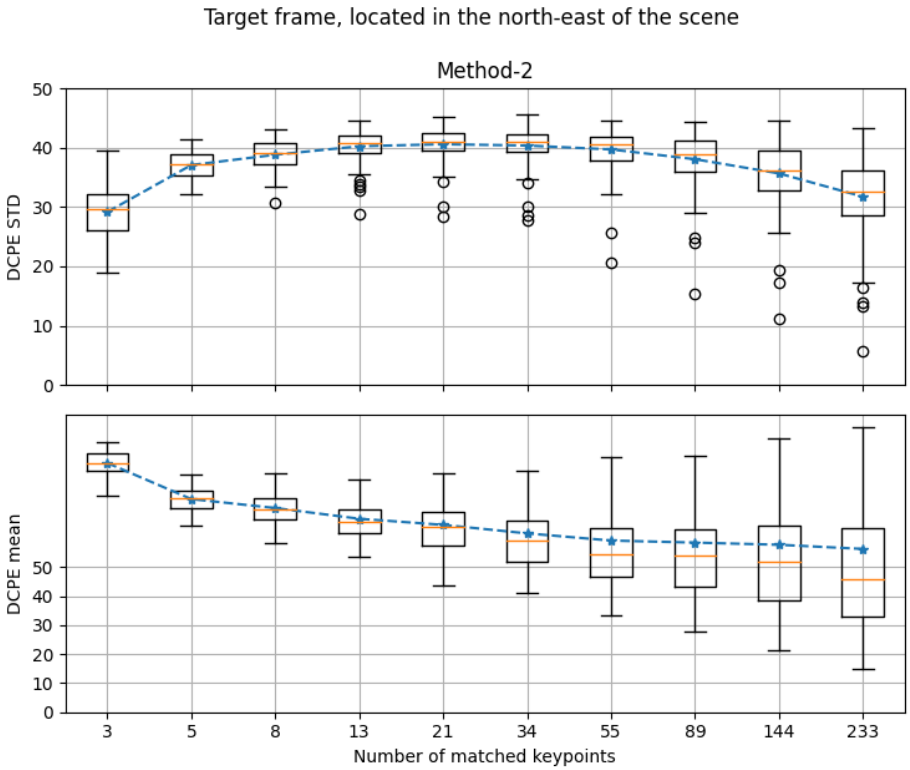


Figure 3.60: Method-2 DCPE Mean and Standard Deviations Box Plot

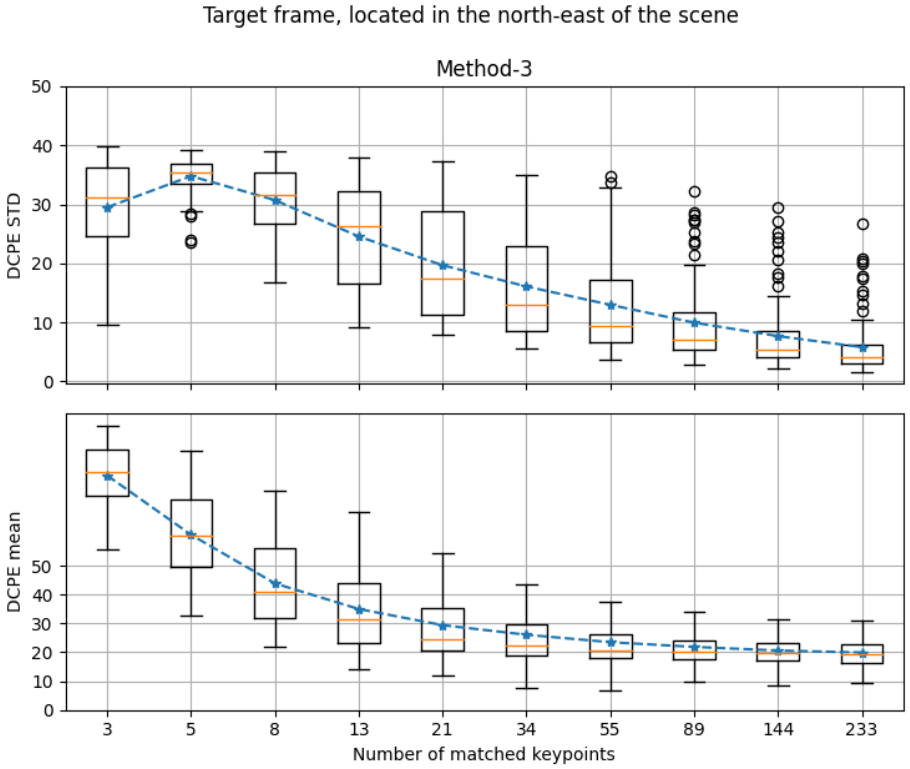


Figure 3.61: Method-3 DCPE Mean and Standard Deviations Box Plot

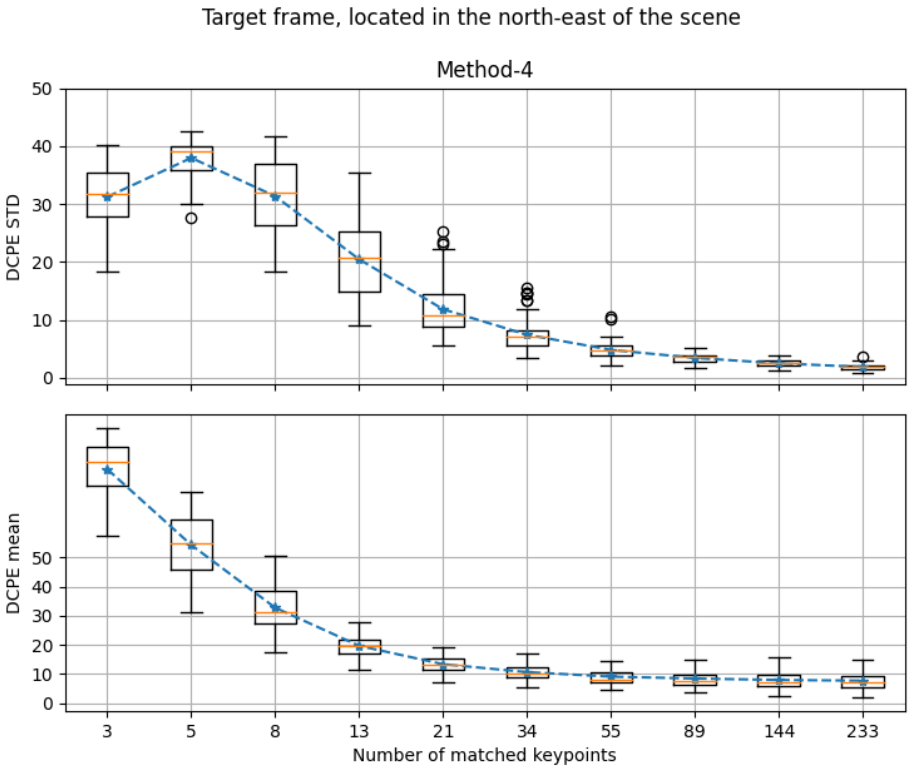


Figure 3.62: Method-4 DCPE Mean and Standard Deviations Box Plot

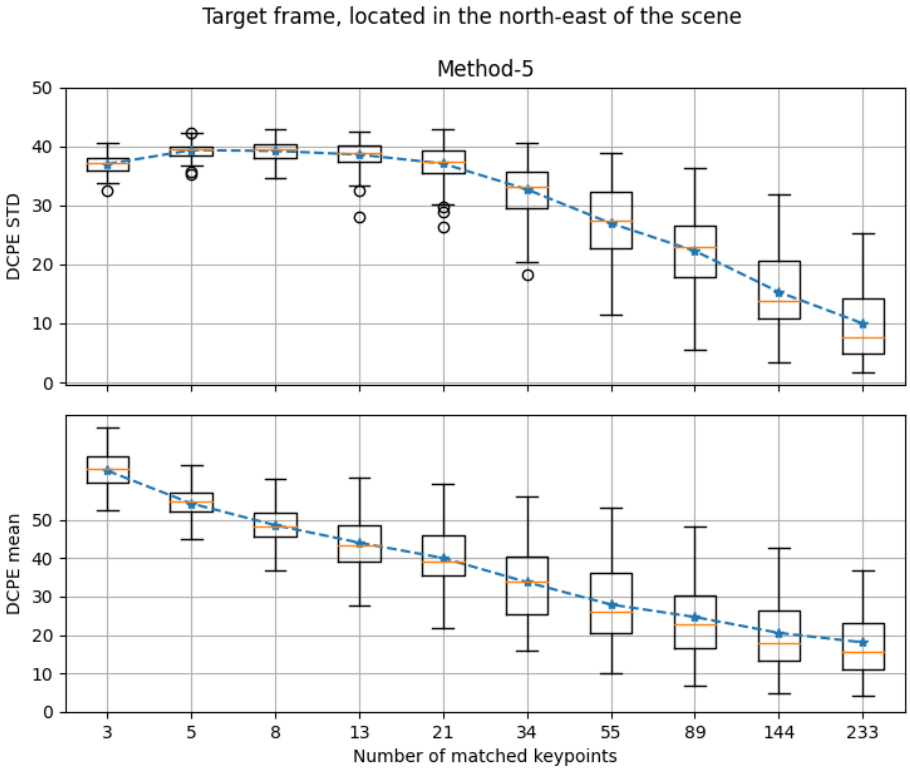


Figure 3.63: Method-5 DCPE Mean and Standard Deviations Box Plot

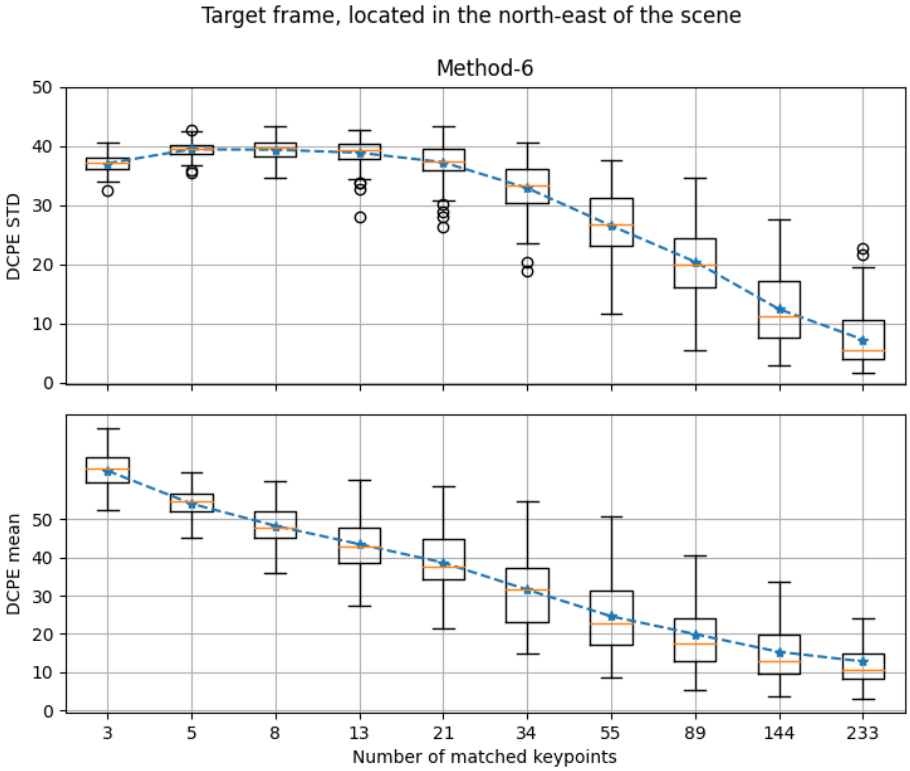


Figure 3.64: Method-6 DCPE Mean and Standard Deviations Box Plot

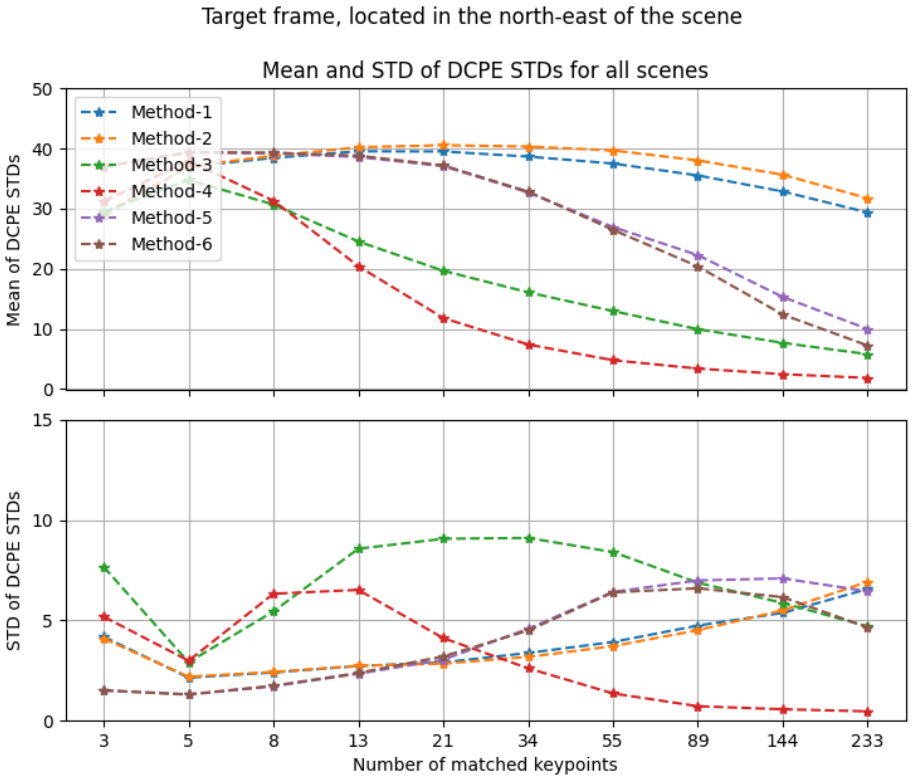


Figure 3.65: Mean and Standard Deviation of DCPE Standard Deviations Comparison

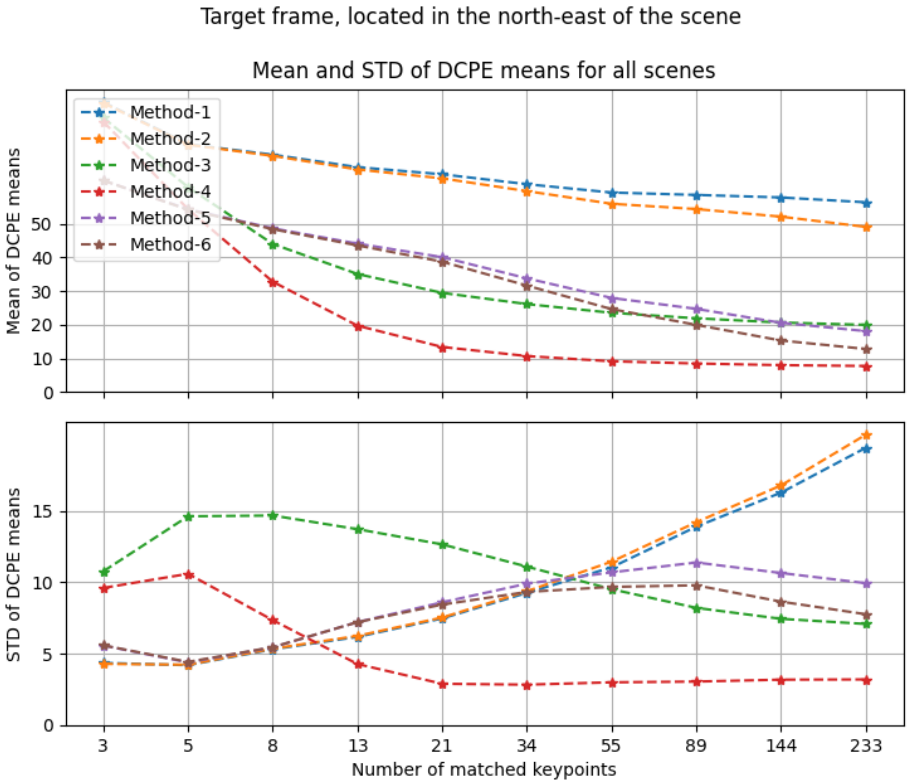


Figure 3.66: Mean and Standard Deviation of DCPE Means Comparison

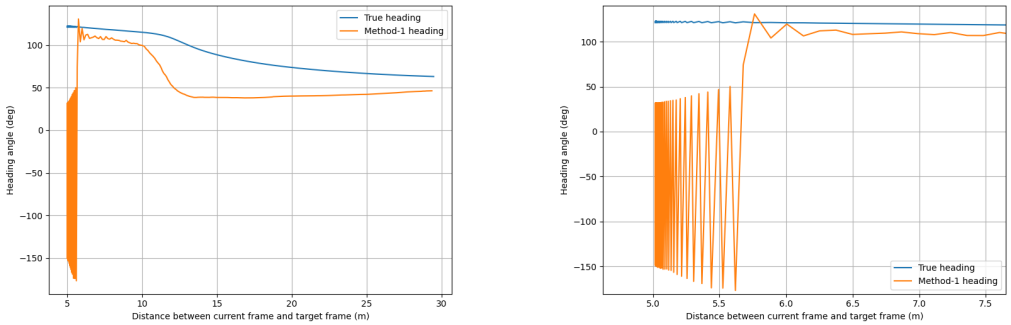


Figure 3.67: Heading Angle Change for Sample Failure Case of Method-1

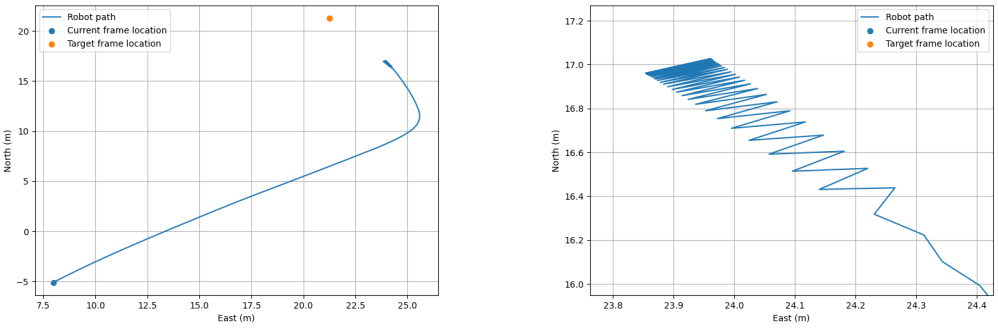


Figure 3.68: Path Change For Sample Failure Case of Method-1

CHAPTER 4

IMPLEMENTATION

In this chapter, we will be explaining the developed algorithm in more detail to increase awareness of the designed procedures and the whole application. In other words, since catching the reader's interest is crucial to be able to express the core idea, explaining the application structure in its most explicit shape has the maximum effectiveness to achieve the attention goal. As algorithm development is divided into three steps, namely, Localization and Path Planning, Mapping and Exploration and Topological Navigation, this chapter will individually follow these steps to touch into details. Firstly, a part of the topological map operations, in which the location of the robot in an already constructed map and planning of the path to reach the target frame is determined, will be clarified. Secondly, the early part of the topological map operations, unknown environment exploration, and map construction will be explained. Lastly, the chapter will be closed with topological navigation which is based on local feature matching and reactive behavior.

4.1 Localization and Path Planning

Localization is needed when new action will be taken according to the current state of the robot. In other words, if the relative location of the robot is not known concerning map nodes and corrective movement has to be taken in accordance with surrounding map nodes, localization is performed to extract the nearest map nodes to the present position of the robot. Since the primary input that is used in all operations is a spherical camera image, localization is also visually executed with the help of spherical

camera frames and their comparison with each other. This process is mainly performed in two main circumstances:

- When a new node is added to the map, but it is not connected to other nodes, i.e. edges are not computed.
- When a target frame is set and the path is not defined to follow starting from the current robot position.

The first bullet is needed in Mapping and Exploration phase in which the robot is analyzing an unknown environment to enlighten its foreign regions. At that stage, localization of the robot is needed to find out the nearest map nodes to estimate both the robot's heading direction to achieve more exploration and to create edges to the currently added node which are explained in detail in Section 4.2. The second bullet is used in Topological Navigation where mapping is already performed, and the robot is located in a known environment. As described specifically in Section 4.3, when a target frame is assigned for the robot to reach by following map nodes, the current robot location and target frame location are localized to define the path between the start point and end point.

The localization process is primarily focused on finding the nearest map nodes to the compared frame. As expressed in Section 2.2.2.1 Mapping and Exploration, when a frame is added to the map, its keypoint locations and descriptor results, which are extracted from SuperPoint [13] algorithm, are also appended as visual bag of words. When local feature matching is performed, descriptor results are compared with each other with the help of SuperGlue [59] as shown in Figure 4.4. The number of matches between the two frames is named similarity, and if the similarity is higher than the highest similarity calculated so far, the current map node is assigned as the nearest node to the compared frame. Also, the previous nearest node is taken to the second nearest map node. This procedure is followed until the last map node is controlled in terms of similarity. Then, the nearest two map node properties are set to finish the whole localization activity. In addition to these steps, if the robot is at the Mapping and Exploration stage, calculated similarity S_e between the compared frame and the current map node is also checked with the minimum similarity threshold. Minimum

similarity threshold $S_{th,min}$ is the minimum similarity value between two frames to assign them as similar enough to perform topological navigation. In other words, two frames are connected and topological navigation is made available when the similarity between them is higher than the $S_{th,min}$.

In the case of having a target frame set as a final destination, Topological Navigation application started to be performed to achieve the goal. As explained explicitly in section 4.3, the global operation of reaching from the current frame to the target frame is divided into local successive operations that are based on following connected map nodes through the target frame. To make these local successive operations available, path planning has to be performed. In other words, path planning is the task of defining intermediate map nodes, called via points, to associate a continuous network between the current frame and target frame. The preliminary concern of path planning is having an already constructed topological map as explained in Mapping and Exploration. Then, at the start of Topological Navigation algorithm as specified also at the flowchart in Figure 4.5, path is initiated if it is not already defined. Before the path is established, localization of both the current frame and target frame is accomplished to extract the nearest node information for each frame. After that, the target frame is added to map nodes to treat it as the final node of the path. In the final step, via points are determined with the help of the Dijkstra's algorithm. In the Dijkstra's algorithm, the cost of the generated path tried to be minimized, i.e. the path includes the lowest number of nodes. Even though we refer to the constructed path as the shortest one, it does not mean that the path is the shortest in terms of metric information. In other words, we indicate the map as only visually shortest due to the lack of metric data in the whole navigation framework. Moreover, a sample path upon topological map is visualized in Figure 4.1. In the figure, the start node is colored green, via points are colored purple and the finish node is colored red. Other map nodes that are visualized with blue are not included in the planned path. Therefore, it is expected from the robot to follow the path starting from the green to reach the red with the help of purples.

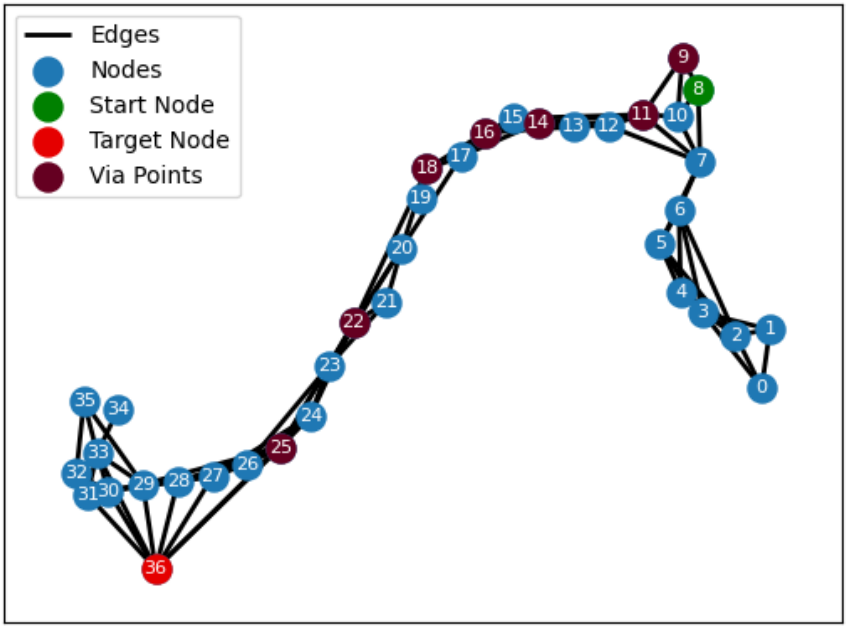


Figure 4.1: Planned Path on Topological Map

4.2 Mapping and Exploration

At the first start of the robot, the operation environment might be unknown. Therefore, the initial algorithm that will be performed is mapping of the environment as proposed frequently in a normal SLAM application. The main purpose of the mapping and exploration phase is to identify environmental features and components to perform planned exercises without any need for further exploration. In other words, after mapping and exploration is performed, the robot will be familiar with the environment to achieve any topological navigation task without the need for additional information. In our case, the constructed map will be an undirected graph that has no specific direction among its nodes. As shown in the flowchart of topological navigation and mapping in Figure 4.5, mapping and exploration is started if the mapping is active after taking current frame properties. In the mapping and exploration phase, the current frame is treated as one of the map nodes and added to the map without controlling similarities. When the current frame is added to the map, it is not directly saved as the map node. Since a raw equirectangular image dimension that is used

as input to our methodology is 1024x2048, it would consume too much memory as the number of map nodes increases. Therefore, instead of directly saving images, we first perform SuperPoint [13] to extract its keypoint locations and descriptor results which have Nx2 and Nx256 dimensions respectively, where N is the number of keypoints that are extracted from a frame. It is limited with S_{max}, the maximum number of keypoints that are determined in the Section 3.2 Heading Estimation Algorithm Comparison. Furthermore, to create a visual bag of words to free up memory and ease the node-saving process, a keypoint is handled as a visual word, and the number N is treated as the maximum number of visual words for a frame. A sample of output keypoints, which are extracted from the related figure image, are represented in Figure 4.3 and labeled with yellow scatter. In addition to these visual words, a unique ID is added to each node to identify it in further needs.

As another important point in the map node amount extension process, the similarity between the current frame and each map node is computed. We define similarity as an approaching metric to analyze the status of the local feature matching result of the two input frames. Since our algorithm framework is principally based on the number of matches reported after operating local feature matching with the help of SuperGlue [59] as expressed with colored lines in Figure 4.4, the similarity is calculated after each local feature matching is accomplished between the current frame and a map node. Considering that maximum number of keypoints that will be extracted from each frame S_{max} is already specified, similarity of compared frames S_e, i.e. indicated edge, is decided by assigning new thresholds by means of S_{max}. Therefore, two similarity thresholds, which are minimum similarity threshold $S_{th,min}$ and maximum similarity threshold $S_{th,max}$, are defined to control if the amount of matches is in the interested region for the current frame and the map node. While minimum similarity threshold $S_{th,min}$ is defined to control if two frames are similar enough to maintain the navigation process, maximum similarity threshold $S_{th,max}$ is defined to monitor if two frames are too close to each other. More specifically in terms of usage, edge addition decision for the current frame and specific map node, node placement decision for the current frame, and the heading estimation methodology for the next algorithm step is determined after threshold investigation of similarities of map edges. As the first step, at the localization process of the current frame among map nodes,

the minimum similarity threshold $S_{th,min}$ is compared after each similarity calculation for the specified edge. If the edge similarity S_e is higher than the minimum similarity threshold $S_{th,min}$, then the edge is added to the map to allow path creation as expressed in (4.1). Moreover, while looping through each map node to compare with the current frame, similarities of the two nearest map nodes to the current frame are saved to use in further cases. Current frame and nearest map node similarity $S_{e,n}$, i.e. nearest edge similarity, will be used to determine if the current frame is too close or too far away from the map. Hence, nearest edge similarity is compared with both $S_{th,min}$ and $S_{th,max}$ to determine if $S_{e,n}$ is inside those limits. In other words, if $S_{e,n}$ is inside $S_{th,min}$ and $S_{th,max}$, then nearest edge is not similar enough to say that nearest map node and current frame are equivalent or disconnected enough to conclude that they are completely mismatched. Therefore, according to the threshold result of the nearest edge, the decision is made whether to place the current frame as a map node or not as shown in (4.2). The second nearest map node is used to determine the heading strategy that will be operated for better exploration according to the current status of the robot. In detail, if the nearest edge similarity satisfies thresholds for node placement, then the second nearest edge similarity $S_{e,sn}$ is compared with the same thresholds. If $S_{e,sn}$ is in between $S_{th,max}$ and $S_{th,min}$, then we come up with the same arguments as we specified for $S_{e,n}$. As the final claim, if both similarities $S_{e,n}$ and $S_{e,sn}$ are inside $S_{th,min}$ and $S_{th,max}$ thresholds, then heading estimation is performed with exploration strategy, if only $S_{e,n}$ or none of them satisfies thresholds, then heading estimation is conducted with admissible heading strategy as can be seen from topological navigation and mapping algorithm flowchart in Figure 4.5.

$$\zeta_e = \begin{cases} 1 & \text{if} \quad S_e > S_{th,min} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where:

$\zeta_e =$ Edge addition decision

$$\zeta_n = \begin{cases} 1 & \text{if} \quad S_{th,max} > S_{e,n} > S_{th,min} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where:

$\zeta_n =$ Node placement decision

While deciding on the heading estimation methodology, the existing map is controlled if it has at least two nodes or not to perform heading estimation based on the current state of mapping and exploration. In more detail, if the map does not have a previous node, i.e. current frame is the only node of the map, the robot starts exploration by assigning a random heading since there is no information available regarding the environment. In the case of having more than one node on the map, after localization of the current frame inside the map with the help of similarity computation between the current frame and nearest map nodes, heading estimation is performed to explore unknown portions of the environment. Since two nearest map frames are extracted through the localization of the current frame inside the map as illustrated in the frame localization algorithm flowchart Figure 4.2, their similarities are checked to decide on the status of the current frame. If one of them is too similar to the current frame, meaning that the current frame already exists on the map, or both of them are completely disconnected from the current frame, meaning that the robot is in a separated environment or traveling inside a covered region such as tunnel, heading estimation methodology called admissible heading is performed to move the robot. In the admissible heading method, there are two main parts. The first one is dedicated to situations where only one map node satisfies similarity thresholds. Since only one map node is available to localize the robot and decide on a heading for more exploration, the heading is estimated to move in the reverse direction from the map node. Additionally, a random angle limited with $[-90°, 90°]$ is added to generate irregular motion in the movement direction as can be seen in the left side of Figure 4.7. The second one includes situations in which there are no map nodes found to localize the robot. These kinds of conditions are expected when the robot is covered with surrounding objects such as a tree or the robot is passing through a closed region such as a tunnel. In these cases, previous heading h_p will be used to keep the robot in the track and a random angle limited with $[-90°, 90°]$ is added to generate irregular motion in the movement

direction as illustrated in right side of Figure 4.7. In addition, if similarity thresholds are not satisfied by any of the map nodes, the current frame is deleted from the map as it is already added at the first stage without checking similarities. However, to prevent recurring admissible heading with no similar map nodes around and moving the robot towards an unknown area, a counter called saturation counter is assigned to count the number of repeated admissible heading steps with no map nodes that satisfy similarity thresholds. If the counter reaches the threshold, a special mode called lost mode is activated. In the lost mode, the main objective of the robot is to arrive at the lastly added map node to continue the mapping process from a safer point. When the lastly added map node is approached, the lost mode is deactivated and the mapping process is continued from where it is left. In cases where similarity thresholds are satisfied for both the nearest map node and the second nearest map node, the robot starts the exploration strategy by moving in the opposite direction with the bisector heading vector of the two nearest map nodes as illustrated in Figure 4.6. Moreover, to add randomness in map generation and to avoid looping through the same headings in each exploration step, a random exploration noise is generated and summed with the calculated exploration heading.

4.3 Topological Navigation

We refer to topological navigation as the process of following a path that is created to reach the goal of finding the given target. In our case, the given target is an equirectangular image on a topological map that is constructed in the Mapping and Exploration. When the target frame is set as the final destination, our algorithm starts with the localization of both the current frame and the target frame and plans the path to reach the latter as explained in detail in section Localization and Path Planning. Since our dominant sensor for the navigation process is a spherical camera, routing the robot directly from the current frame to the target frame is not possible due to the camera range. Therefore, the planned path is divided into successive subtasks with the help of nodes and edges of the constructed topological map. Since these subtasks are already named via points in the Localization and Path Planning section, following via points is the primary objective of the topological navigation operation. As can be

seen from the flowchart of topological navigation and mapping in Figure 4.5, heading estimation is performed until the algorithm is satisfied as the current via point is reached. Moreover, the via point index increment decision is taken into consideration in two steps. The first one is dedicated to accomplishing the similarity of the current via point $S_{e,cv}$. When the current via point similarity to the current frame is higher than the maximum similarity threshold $S_{th,max}$, then we are safe to say that the robot is close enough to the current via point to start its next local task. However, since metric information does not exist in the setup and the similarity threshold is satisfied before reaching the current via point in most of the cases, the next via point might not be visible from the location where the via point index is increased. This is the point where the second consideration step is taken into account. In detail, after the current via point similarity is satisfied, the robot starts to control the similarity of the next via point before increasing the via point index to switch over to the next local task. When the next via point similarity $S_{e,nv}$ meets the criterion of being higher than the minimum similarity threshold $S_{th,min}$, visibility concern of the next via point is diminished. As illustrated in the (4.3), when both the current via point and the target via point similarity requirements are satisfied, the via point index is increased by one, otherwise kept the same. It would be necessary to highlight at this point that the target frame is treated as the last via point since it will be the last subtask of the path planning. Therefore, this iterative process is followed until the last via point, i.e. target frame, is decided to be close enough to the robot's location. Furthermore, besides following the current via point according to the state of the robot, the robot has to act in accordance with the environmental conditions. The primary concern in terms of environmental conditions is obstacles. Hence, the robot always controls surrounding obstacles in order not to hit any of them. As can be seen from the flowchart of heading estimation Figure 4.8, the first thing that is controlled is the obstacle existence. If the robot detects any obstacles around, the next action is taken to avoid them regardless of the current state of the robot. When the obstacle heading h_o is specified, corrective movement will be moving in the opposite direction of the obstacle heading direction as also shown in Figure 4.9. In addition to the obstacle avoidance heading, the estimated heading to perform successful topological navigation is taken into account in obstacle avoidance. To be more specific, the weighted sum of the obstacle heading and the estimated heading is calculated by generating a

random weight $w_1 \in (0.5, 1.0)$. This way, the robot is able to keep the track of the current topological navigation state while escaping from the obstacle. Even though our decision process takes obstacles into account in the heading estimation, the creation of obstacle heading is not taken into account in this thesis. Thus, the obstacle heading h_o is assumed to be fed from an outer source.

$$i_{vp} = \begin{cases} i_{vp} + 1 & \text{if } (S_{e,cv} > S_{th,max} \; and \; S_{e,nv} > S_{th,min}) \\ i_{vp} & \text{otherwise} \end{cases} \tag{4.3}$$

where:

$i_{vp} =$ via point index

While closing the chapter, we would like to summarize the fundamental processes that allow the whole algorithm framework to work in harmony with each other. After a spherical image is taken from the only input source spherical camera, the most crucial part of the algorithm that makes the images meaningful from the topological navigation point of view, local feature matching is performed with the help of SuperPoint[13] and SuperGlue[59] to extract matched keypoint locations between two frames. Since matched keypoint locations are already represented in the spherical grid as a result of the spherical grid of the spherical camera, latitudes λ and longitudes μ of the matched keypoint locations are directly used in the heading vector estimation that will determine the robot's motion until the next computation. After the extraction of matched keypoint locations, heading vector is estimated by using method-4 stated in Section 2.2.1.3 Heading Estimation. The main reason for the selection of the method-4 are robustness of the method to keypoint location distribution around frames, and its computational efficiency compared to other proposed methods due to the elimination of keypoint pairing mentioned in method-1. In the heading estimation process, latitudes are taken into account such that if the latitude of a matched keypoint is increasing from the current frame to the target frame, then the keypoint is classified as a front keypoint, and the robot is decided to move in that matched keypoint longitude direction, otherwise, the keypoint is classified as rear keypoint, and the robot is decided to move in the inverse direction. After each matched keypoint is evaluated according to the criterion, all keypoint heading vectors $\underline{u_i}$ are summed up to generate

the heading vector $\underline{h}$ by assigning a priority on the front keypoints by taking exponential function of latitude difference in magnitude $\Delta\lambda_i$. Equations for the heading estimation are explained deeply between equations (2.11) and (2.18). At this point, it is essential to say that even though explanation of the proposed algorithm is divided into three main parts; Localization and Path Planning, Mapping and Exploration and Topological Navigation, fundamental flowcharts in Figure 4.2, Figure 4.5 and Figure 4.8 are drawn to simplify major processes as uncoupled as possible. However, it is vital to keep in mind that the main parts of the algorithm frequently intersect with each other and work in a united regime due to the nature of any SLAM application. In the next chapter, we will explain constructed simulation environment concerning the mobile robot and the worlds used in experiments by visualizing and detailing their purposes.

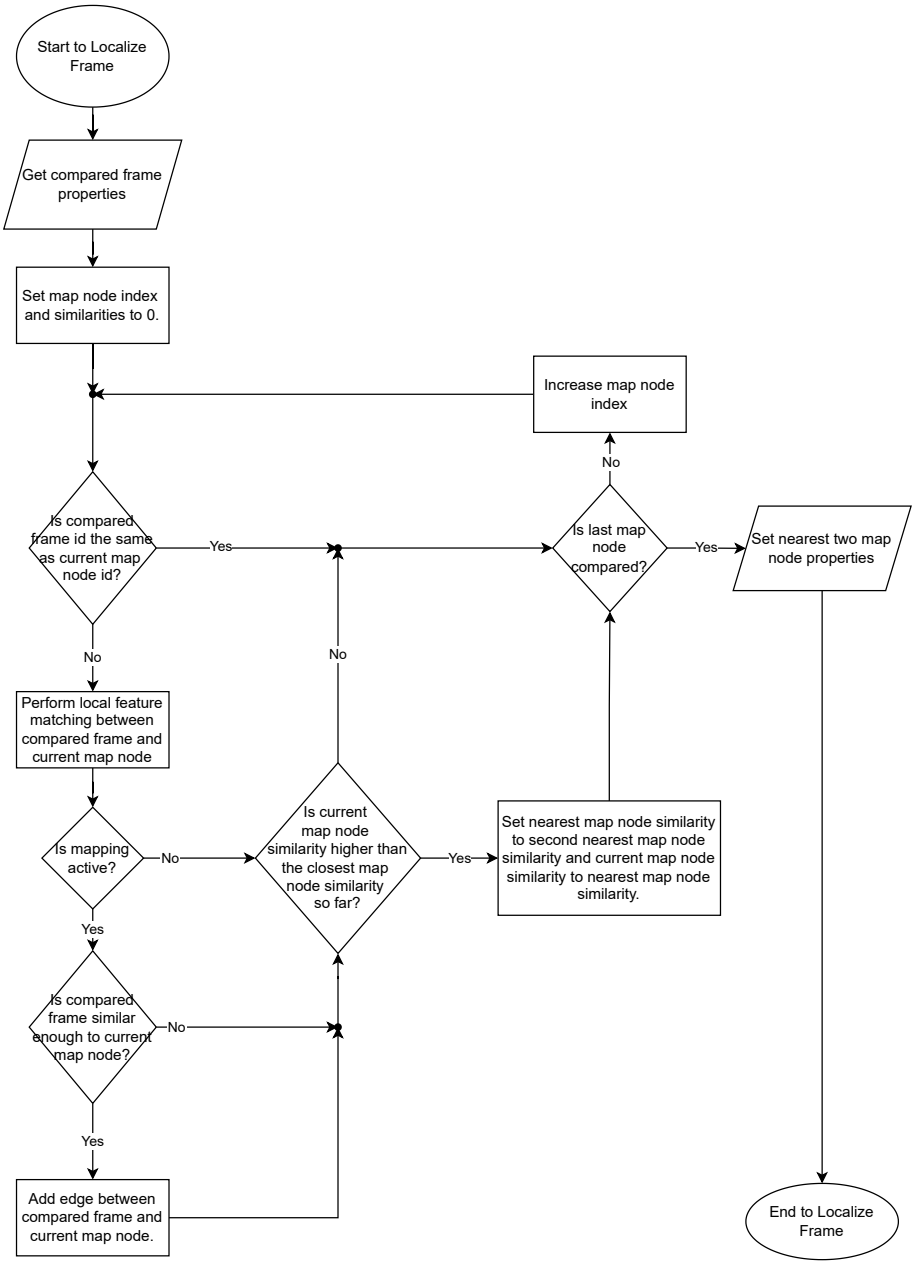


Figure 4.2: Frame Localization Algorithm Flowchart

Figure 4.3: Sample Image with Extracted Keypoints

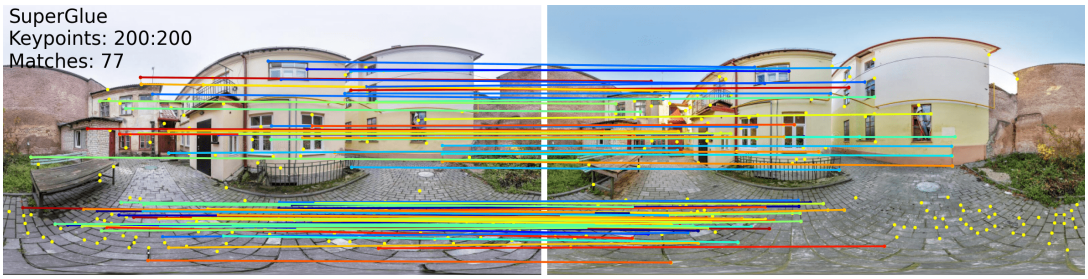


Figure 4.4: Sample Feature Matching Between Two Images

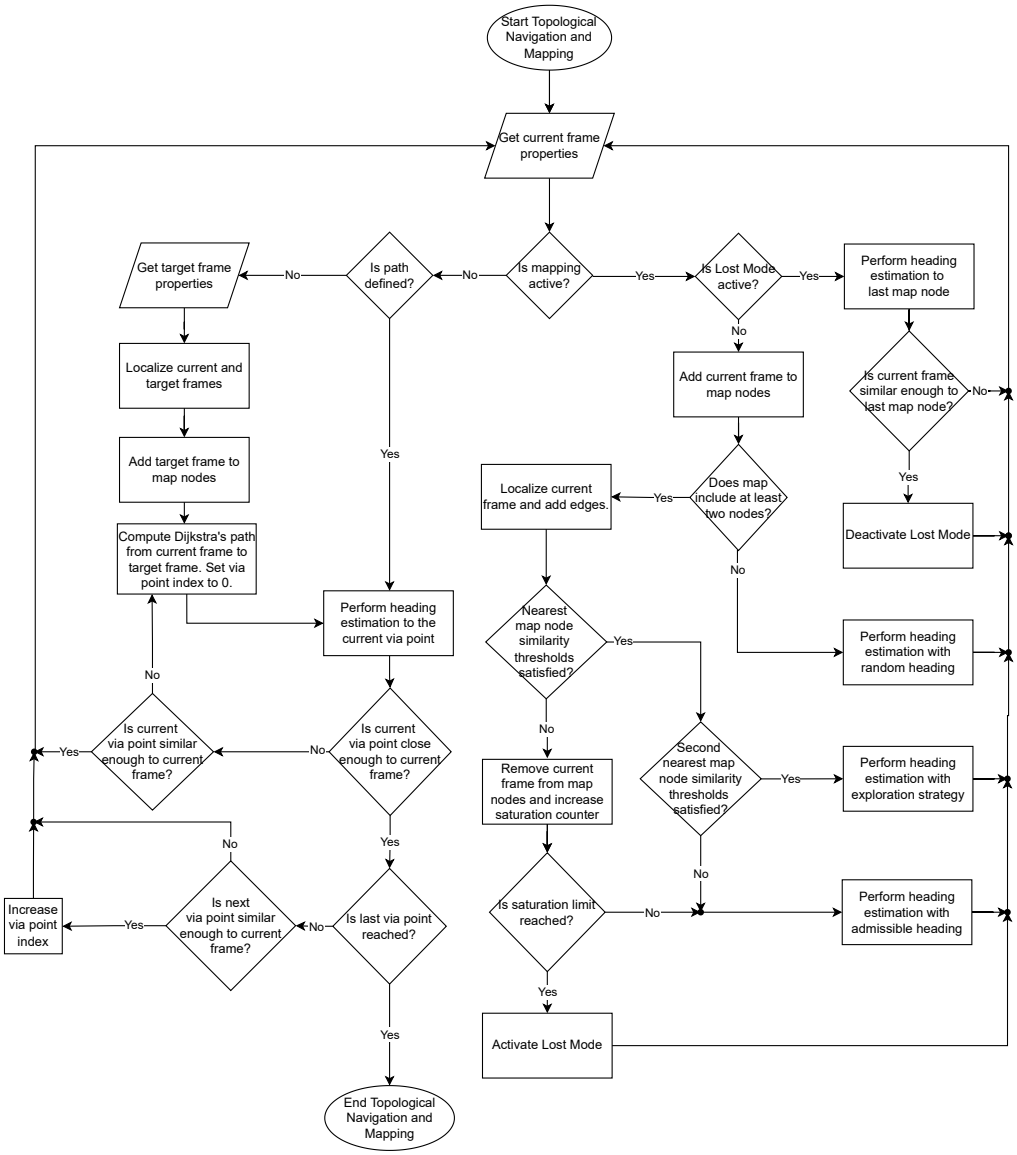


Figure 4.5: Topological Navigation and Mapping Algorithm Flowchart

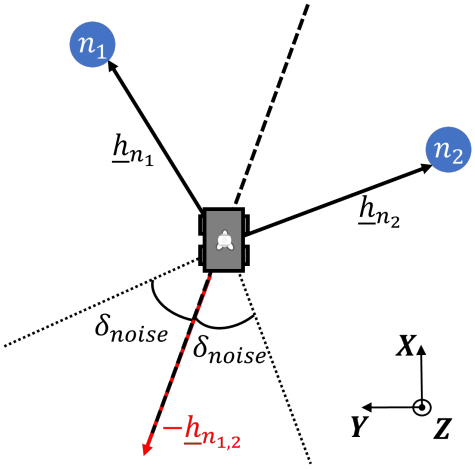


Figure 4.6: Heading Estimation with Exploration Strategy

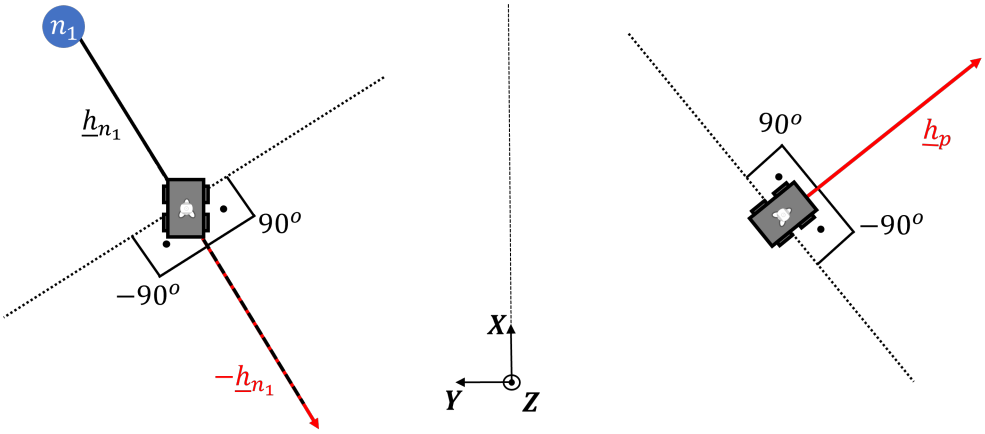


Figure 4.7: Heading Estimation with Admissible Heading

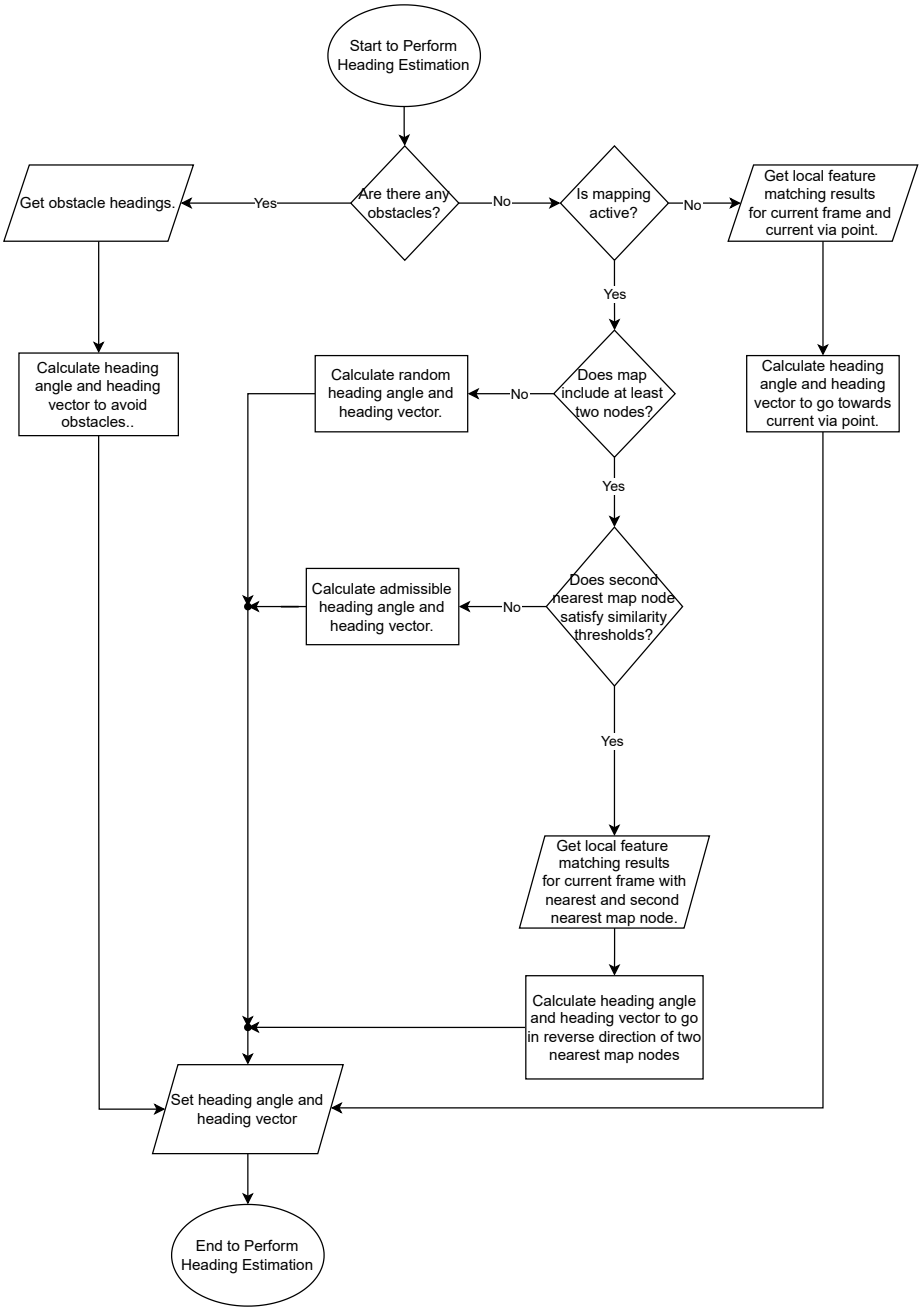


Figure 4.8: Heading Estimation Algorithm Flowchart

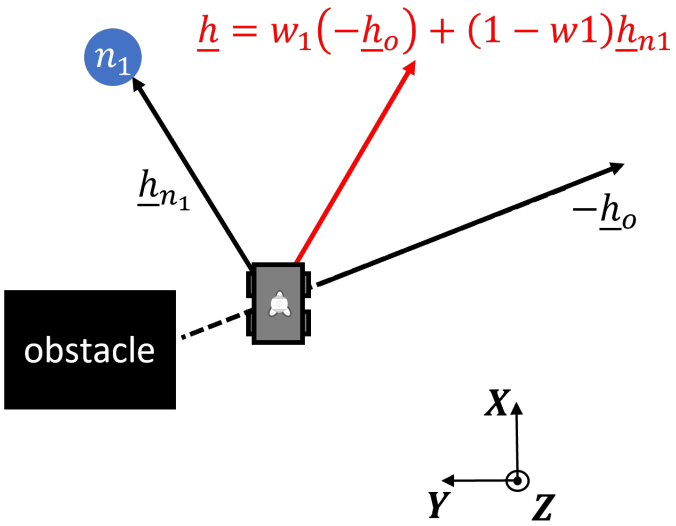


Figure 4.9: Obstacle Avoidance Strategy

CHAPTER 5

SIMULATION ENVIRONMENT

In this chapter, the developed simulation environments, which are used extensively while examining the algorithm performance, will be explained. The developed worlds are constructed in the open-source robotic simulator Webots [45] due to its simplicity and availability for a wide range of usage.

In a robotic application process, trial and error are frequently needed to overcome possible error situations. However, reporting and solving those error cases are not evident due to the different side effects of testing in a real environment. The first reason is that the environment may not be found as it is previously. In other words, as a consequence of dynamic changes in habitat such as moving objects, weather, illumination, and shadow, the recreation and pinpointing of the error case may not be possible. Therefore, the creation and finalization of a single scenario may take a significant amount of time. If an operation is defined with a backstory such as a kidnapped robot in an unknown environment like a forest, the foundation and maintenance of the environment can take days, and keeping the environment standstill may be impossible due to seasonal changes. However, if the environment is built in a robotic simulator, the environment will stay the same as it is left. Hence, it will save a considerable amount of time that is spent fixing the real-life testing environment. The second reason is that the delay length which is dependent on situations other than algorithm development can be high with real-life testing. In detail, if an algorithm development process is considered, a developer needs to deal with the hardware alongside the software. Therefore, any complex hardware issue will delay the testing due to the learning curve that must be encountered before mastering the root

cause. The third reason is that the algorithm robustness can be tested considering the uncertainties. Various uncertainties occur during any robotics application, hence; the robustness of a developed algorithm is crucial for the robot to continue operating. Since any parameter can be controlled in the world in a robotic simulator, controlled testing is accomplished efficiently. Moreover, if multiple runs need to be executed to generate a statistical analysis of the effect of a specific parameter, testing can even be automated. Thus, due to stated reasons, we have decided to start our experimental process in the Webots robotic simulator.

The chapter is divided into two parts. In the first part, the mobile robot, which is designed in Webots, to use in applications will be explained. Finally, the chapter will be concluded by clarifying operated worlds in various experiments.

5.1 Mobile Robot

The whole framework explained in this thesis is designed for a mobile robot platform. Therefore, before going into the hands-on part, a mobile robot that can handle the proposed topological navigation and mapping has to be defined. The mobile robot as shown in Figure 5.1 with 4 legs is designed in Webots and placed into environments to perform the topological navigation process efficiently.

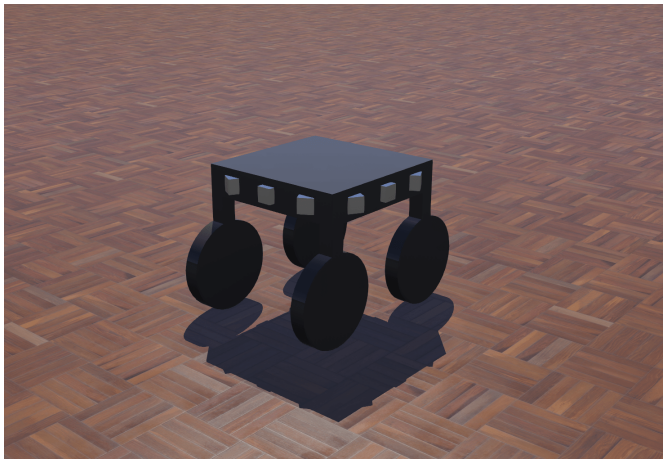
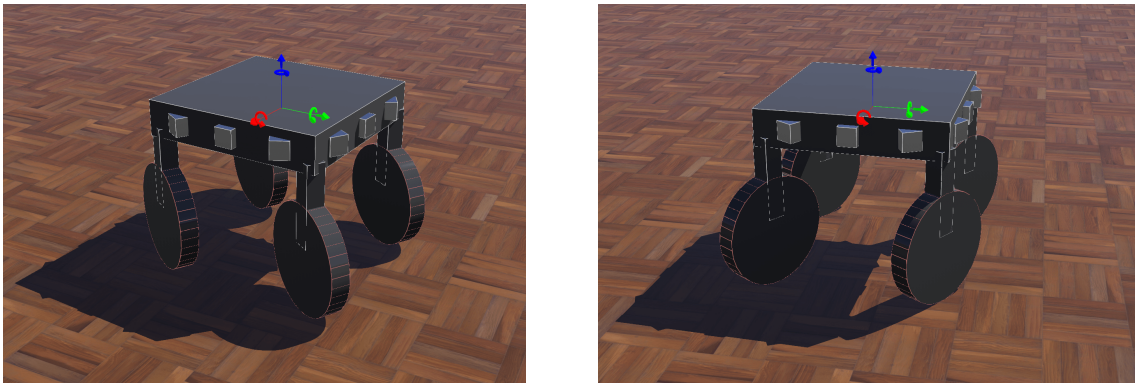


Figure 5.1: Mobile Robot Designed in Webots

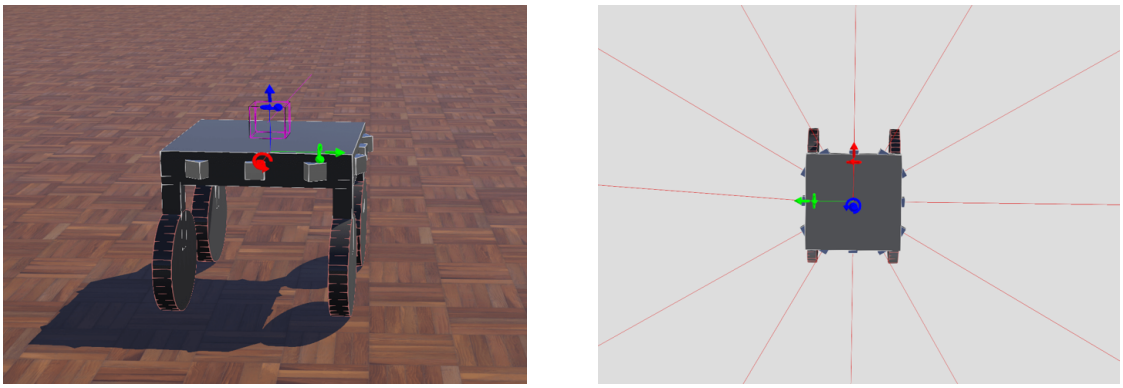
The robot is equipped with two servo motors for each of its legs to execute both movement and rotation of the wheel. Therefore, when a heading vector is computed



(a) Wheel Rotated to $60°$ (b) Wheel Rotated to $-30°$

Figure 5.2: Mobile Robot Wheel Rotations

in a sample topological navigation step, wheels are rotated as can be seen in Figure 5.2 along with the heading vector to move towards the goal. Moreover, a spherical camera shown in Figure 5.3a is attached to the robot, and the camera is highlighted with a pink cube to show its frustum. Also, 12 equivalent distance sensors are attached to the robot at $30°$ intervals to create the external source of the obstacle heading. Line of sights of each sensor are shown in Figure 5.3b with red.



(a) Spherical Camera (b) Distance Sensors

Figure 5.3: Mobile Robot Sensors

5.2 Simulation Worlds

Four Webots worlds were constructed to employ four distinct purposes. Since the main purpose of topological navigation is achieving the navigation task in indoor environments or narrow outdoor spaces, each world is dedicated to a specific purpose. In the following sections, these four worlds and their representation purposes will be

explained.

5.2.1 Village Realistic

In the simulator, we have selected an existing world named realistic village, and its general view is placed in Figure 5.4. The world is dedicated to representing a small village with details. However, since the goal of topological navigation is achieving the navigation process in a region where it is challenging to have accurate location estimation such as narrow streets, a specific neighborhood of the world is used in experiments. Therefore, borders of the specified neighborhood are defined as shown in Figure 5.5. In the figure, borders are highlighted with yellow on purpose to ease identification. Nevertheless, borders are defined with the help of natural resources such as a sidewalk as in Figure 5.6a, barrier as in Figure 5.6b and fence as in Figure 5.6c. This way, the obstacle avoidance of the robot is also tested to check if it escapes colliding with borders or not. However, it is essential to keep in mind that the obstacle heading will be supplied with the help of an external source other than a spherical camera since it is not included in the topological navigation and mapping framework.

Figure 5.4: General View of Webots Village Realistic World

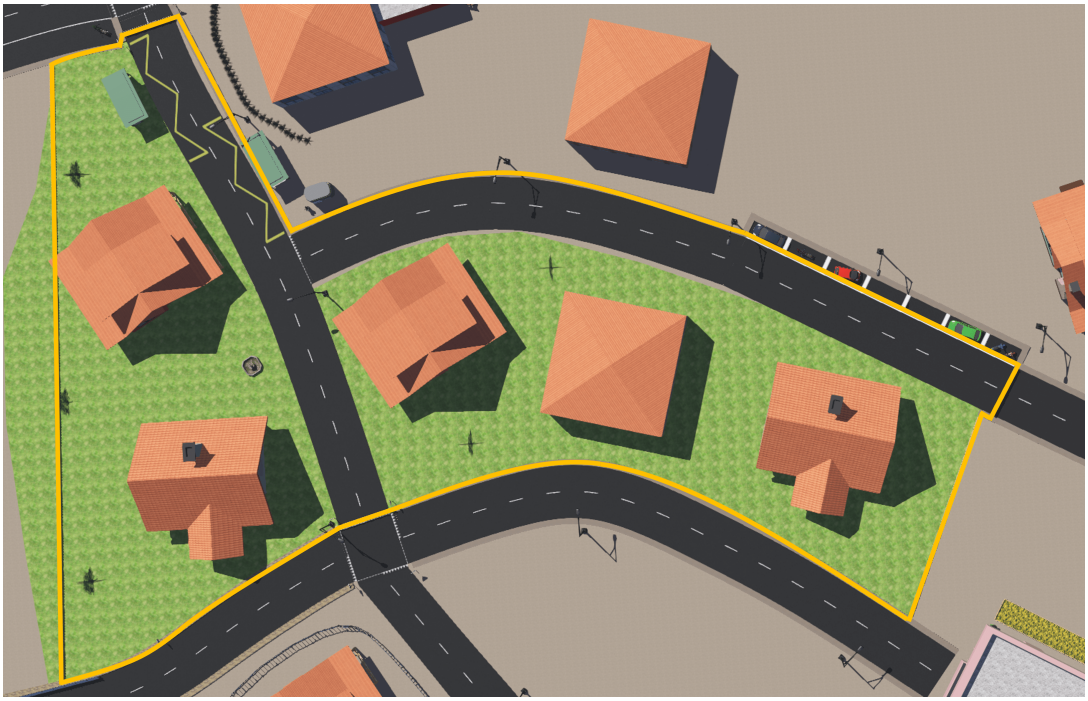
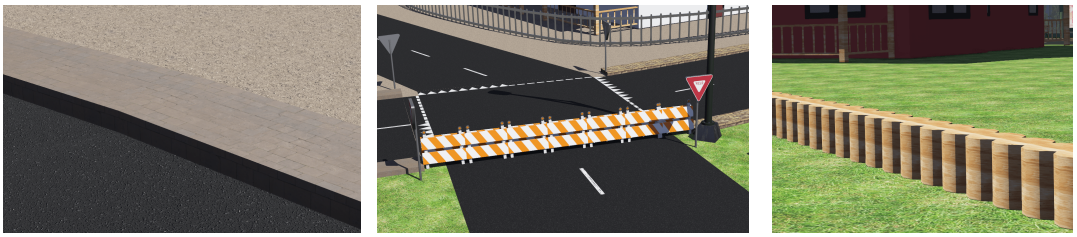


Figure 5.5: Village Realistic World Neighborhood Used in Applications



(a) Sidewalk (b) Barrier (c) Fence

Figure 5.6: Natural Border Resources

5.2.2 Complete Apartment

The second world that will be used in the application process is the existing Webots world named complete apartment as shown in Figure 5.7. The main usage of the world is simulating an indoor environment with household furniture such as a couch, a bookshelf, and a dining table. Therefore, rooms of the apartment are kept closed so that the living room and the corridor are used in experiments as highlighted with yellow borderlines in Figure 5.8.

Figure 5.7: General View of Webots Complete Apartment World

5.2.3 Break Room

The third world chosen to be used is the existing world named break room as shown in Figure 5.9. As the name suggests, the world represents a break room in an office environment. In detail, the world is filled with office furniture that will allow testing the algorithm concerning an indoor office atmosphere.

5.2.4 Factory

The fourth and final world is the extended form of the existing Webots world named hall. The world is enlarged with an additional outdoor environment to represent a hybrid factory world that includes both indoor as in Figure 5.11 and outdoor as in Figure 5.12. Our main intention to use the factory world is to simulate an environment with industrial types of equipment as can be seen in Figure 5.10. Moreover, as the world includes both indoor and outdoor spaces, drawbacks that might occur while moving through spaces are tried to be localized.

Figure 5.8: Complete Apartment World Rooms Used in Experiments

In summary in this chapter, the designed mobile robot and constructed worlds in Webots robotic simulator are demonstrated in detail. In the next chapter, we will explain generated algorithm results by diving into the hands-on part.

Figure 5.9: General View of Webots Break Room World

Figure 5.10: General View of Webots Factory World

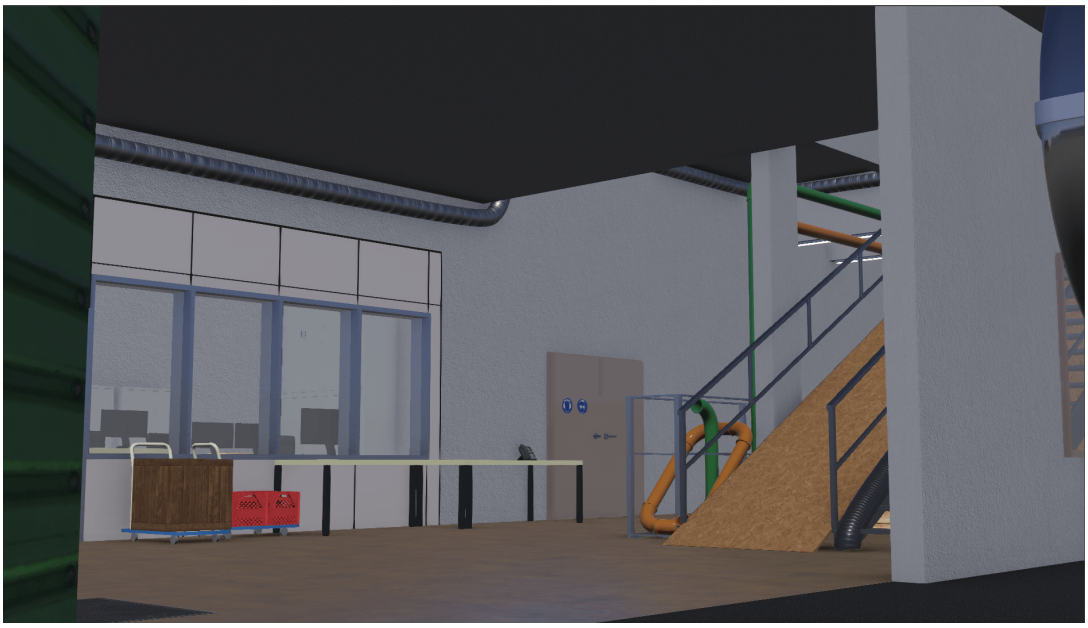


Figure 5.11: Webots Factory World Indoor

Figure 5.12: Webots Factory World Outdoor

CHAPTER 6

RESULTS AND DISCUSSION

Methods that will be used for local feature matching and heading estimation are decided in Chapter 3 and the whole framework for the topological navigation and mapping is explained in Chapter 4. Therefore, this chapter is dedicated to the implementation of algorithms and decision processes. The final structure of the framework is tried out interactively to criticize the algorithm results in environments that are created in the open-source robotic simulator Webots as also explained in Chapter 5.

Even though uncertainties that occur due to environmental changes such as brightness, shadow, weather, moving objects, and pedestrians are not easy to manipulate in real-life testing, controlled experiments can be easily performed by handling a certain environmental parameter in a simulated scene testing. Moreover, in Webots, sensor and movement effects such as noise and motion blur can also be specified concerning the needs. Therefore, example Webots worlds are created to examine the performance of the proposed methods. After the worlds are introduced, both topological navigation and mapping processes will be executed to investigate the algorithm's performance along with its drawbacks.

6.1 Mapping and Exploration

After the construction of the simulation worlds in Webots, the first thing that the robot has to accomplish is the identification of the environment features. In other words, the story starts with a robot placed in an unknown environment. To generate later tasks successfully in the environment, the robot has to analyze the world concerning

its local features in different locations. Therefore, we have started the mapping and exploration phase from a random point to identify as many places in the world as possible. In the next subsections, the mapping and exploration process for each world will be investigated individually. In all subsections, a sketch of the Webots worlds as in Figure 6.1, Figure 6.9, Figure 6.14 and Figure 6.16 with their gathered obstacles is created to simplify the explanation process of the algorithm operations. In the figures, the border of the world is highlighted with yellow, and the robot is not able to go beyond those limits on account of natural borders in the environment. World properties that cover a large number of areas such as houses, fountains, couches, tables and bus stops are shown with dark gray regions and their classifications are written nearby. Obstacles that overlap a smaller amount of regions such as trees and street lights are scattered with specific markers as labeled in the legend. Moreover, the robot start location, i.e. the first node, is specified with the blue circle. As the robot learns the environment, the number of blue circles that specify map nodes will increase and the nodes will be connected via edges to show the existence of a path between connected two nodes. The mapping and exploration phase is executed as stated in the Chapter 4. Firstly, the first robot location is added to the map, and the robot generated a random heading vector to start the mapping. With the addition of another node to the map, the robot started executing either admissible heading or exploration strategy concerning the current step of the mapping and exploration phase as stated in Figure 4.8. Since the maximum number of keypoints that will be generated from a frame is specified in section Section 3.2 Heading Estimation Algorithm Comparison as 150, the maximum similarity that can be achieved between two frames S_{max} is assigned as 150 as well. As also mentioned in the same section that the number of matched keypoints to keep topological navigation in a certain performance is classified as 60. However, to increase robustness and to be able to execute the navigation process even under dynamic effects that occur due to environmental changes the mapping safety factor γ_m is multiplied by the number in the mapping and exploration phase. Hence, the minimum similarity threshold $S_{th,min}$ is assigned as 80 in the mapping and exploration phase only. On the other hand, the main purpose of the mapping and exploration phase is to create connected, but distinct map nodes, hence; the maximum similarity threshold $S_{th,max}$ that will be controlling if two map nodes are too close to each other is defined as 110 based on empirical results in the Webots world. As summarized

in equations (6.1) and (6.2), a node is placed if the current frame and nearest map node similarity $S_{e,n}$, i.e. nearest edge similarity, is inside the similarity limits, and an edge is added between two nodes if the current edge similarity S_e is higher than the minimum similarity threshold.

$$\zeta_n = \begin{cases} 1 & \text{if} \quad S_{th,max} > S_{e,n} > S_{th,min} \\ 0 & \text{otherwise} \end{cases}$$

where:

$$\zeta_n = \text{Node placement decision}$$
$$S_{th,max} = 110, \text{ The maximum similarity threshold}$$
$$S_{th,min} = 80, \text{ The minimum similarity threshold} \tag{6.1}$$

$$\zeta_e = \begin{cases} 1 & \text{if} \quad S_e > S_{th,min} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

where:

$$\zeta_e = \text{Edge addition decision}$$

6.1.1 Village Realistic

The hyperparameters of SuperGlue are selected as suggested in [59]. Therefore, the confidence threshold is assigned as 0.2 so that matches between two frames that have higher confidence than 0.2 are specified as correct matches and ignored otherwise. The mapping and exploration start location, i.e. the first map node, for the village realistic world is shown Figure 6.1. After the first time-step, heading is generated by following steps mentioned in Figure 4.8.

According to explained strategies so far, the mapping and exploration phase is continued to see if the robot accomplishes building a sufficient map inside the Webots world by avoiding obstacles. At this point, it is essential to say that metric locations for both robot and node are saved simultaneously for debugging and investigation purposes. When the robot reaches a corner inside the map, the robot is not able to

Figure 6.1: Mapping and Exploration Start Location for the Village Realistic

proceed with the exploration anymore since regions that are close to the border are already identified and the exploration strategy wants the robot to go towards borders as in Figure 6.2. At that stage, our setup lacked an additional step such as navigating to a random map node to continue exploration. Therefore, instead of leaving the robot making back-and-forth motions around the border without any map node addition, the robot is moved to an unidentified region to continue mapping. Thus, the map generated in the first mapping and exploration phase can be appended and enlarged with additional mapping phases. Exhaustive mapping and exploration is kept outside of this thesis and left as future work.

After we have satisfied with the constructed map, the mapping and exploration phase has been ended, and the map is illustrated in Figure 6.3. It is noticeable from the figure that there are long-distance edges between nodes that are not close enough to each other to be connected. The main reason for the issue is the common features that exist even if two frames are not adjacent. To better illustrate, two map nodes, node-14 and node-82 are taken into account and their locations are shown in the map

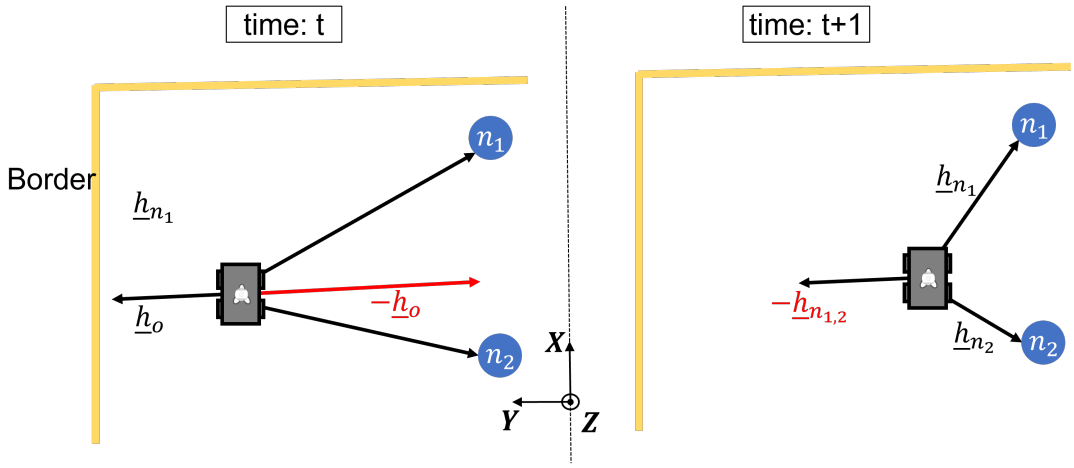


Figure 6.2: Sample Failure of Exploration Strategy

in Figure 6.4. If the node locations are analyzed qualitatively at the first look, their connectivity is not feasible due to blocking houses between the two nodes. However, if the two images are investigated together in terms of common features as in Figure 6.5, the frames look alike concerning surrounding objects even if most of them such as houses are not the same. After the matching is performed between the stated map node frames, the matching results are represented in Figure 6.6. We have realized from the matching results that since a part of the robot is always visible at the bottom side of a frame, it creates false positive matches as our primary purpose is to match environmental features. To overcome this problem, a crop degree will be assigned to crop the frame with a specific amount so that part of the robot will be deleted from any frame. Another issue that is visible in the matching is that common house features such as chimneys, windowpanes, and roofs are the main matched keypoint locations even if they don't correspond to the same house. The issue will be overcome by increasing the confidence score of SuperGlue from 0.2 to 0.4. As can be seen from the local feature matching result after the adjusted settings in Figure 6.7, the number of matched keypoints is decreased and the robot is not visible in frames anymore. Thus, we will continue with the matching confidence threshold of 0.4 and crop degree of 25°. However, it is still sufficient to keep in mind that common environmental features such as chimneys are matched even if they don't correspond to the same house. Hence, if a geometric verification method like RANSAC is adopted for spherical images, then it would discard the common features if the geometries do not match with each other in two images. Further, to prevent long-distance nodes from

being connected in general, the mapping process could also be handled episodically. In detail, a sliding window of possible node connectivity range could be defined to add an edge from the current frame. This way, the current frame is only connected to a limited range of map nodes.

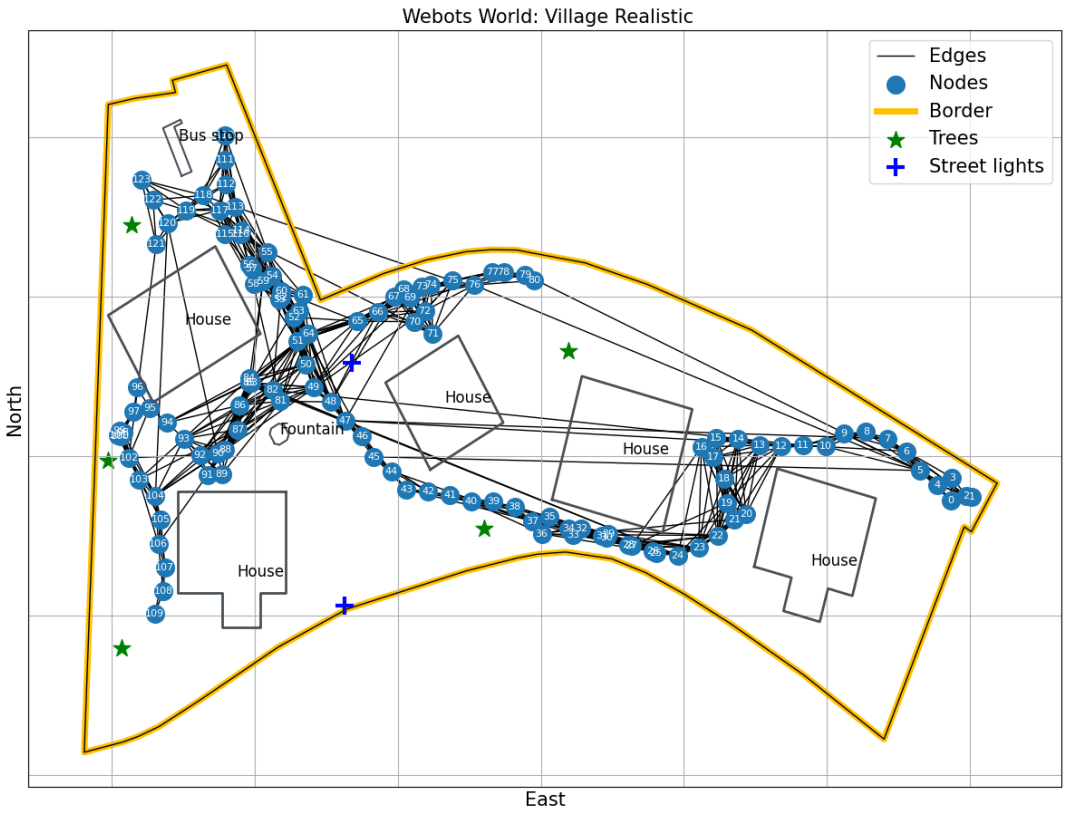


Figure 6.3: Constructed Topological Map for the Village Realistic

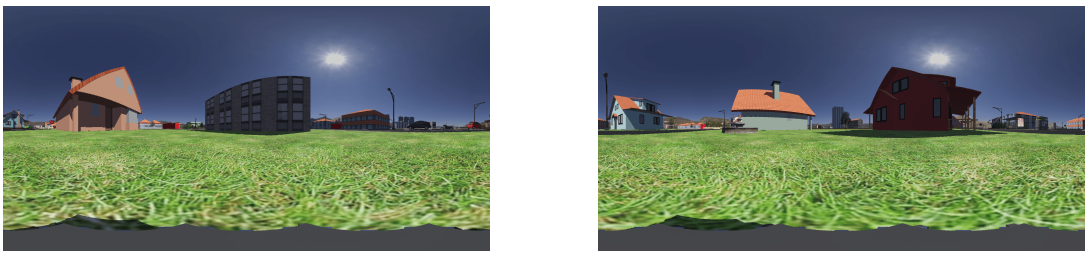
The mapping and exploration phase is restarted from the first node in the Figure 6.1 and the regenerated map in the village realistic world is drawn in Figure 6.8. As can be seen from the figure that the long-distance edges caught in the previously constructed topological map in Figure 6.3 are not valid anymore, and all edges are created within the visible range of the robot.

In summary, even though the mapping and exploration phase is not perfectly finalized, an efficient topological map with distinct nodes that cover most of the parts of the world is generated at the end of the process.

Figure 6.4: Map Nodes 14 and 82 Locations

6.1.2 Complete Apartment

The mapping and exploration phase for the complete apartment world is started from the node location shown in Figure 6.9. In an indoor environment, the upper side of the frame is covered by the ceiling. Further, a house in general consists of several repeatable features such as equivalent doors, ceiling lights, and ceiling corners. Since a door, for example, is not unique in the house, false positive matches are likely to be extracted from those features. Even though these features might exist in any



(a) Map Node-14 Frame

(b) Map Node-82 Frame

Figure 6.5: Map Nodes 14 and 82 Frames

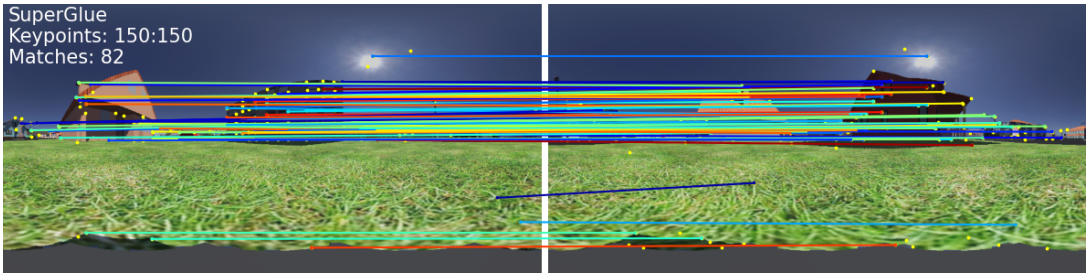


Figure 6.6: Matching Result of Map Nodes 14 and 82 Frames

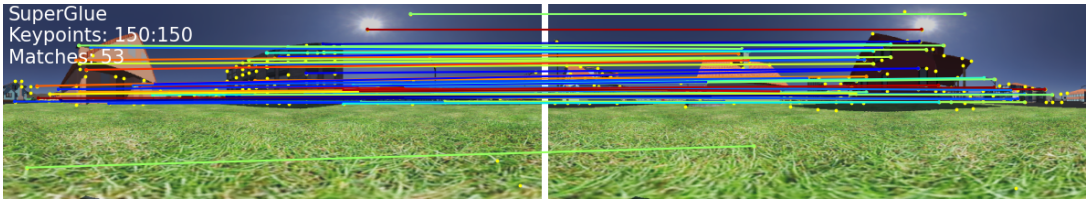


Figure 6.7: Matching Result of Map Nodes 14 and 82 Frames with Adjusted Settings

frame, the successful connection of those two frames is only applicable when the common features are salient in the frame such as a narrow corridor. From a different perspective, two nodes created in the complete apartment map in Figure 6.10 are selected to show the process in detail. As can be seen in Figure 6.11 that there are identical doors in different locations to represent different rooms in the corridor. A successful match is created when a keypoint is generated from a similar location of different doors. If the latitude change in magnitude is high for those false positive keypoints as in Figure 6.11, method-4 tends to create a faulty movement due to the high priority of false positive matches. Therefore, a high chance of failure is detected for method-4 in case of false positive matches with high latitude change in magnitude.

As revealed in the Section 3.2 that method-1 has defect on estimating accurate heading when the keypoints are bounded in a region. Since found out in this section that method-4 creates failing headings when there are false positive matches with high latitude change in magnitude. To overcome drawbacks of both methods, our heading estimation algorithm output in indoor worlds is interactively switched between method-4 and method-1 based on matched keypoint distributions as shown in Figure 6.12. In detail, in an indoor environment heading is estimated using method-1 when the keypoint longitude distribution is not limited in a region over the circle as in Figure 6.12. In the figure, the matched keypoint longitude distribution is represented with a polar histograms. The red histogram bars are dedicated to show the number

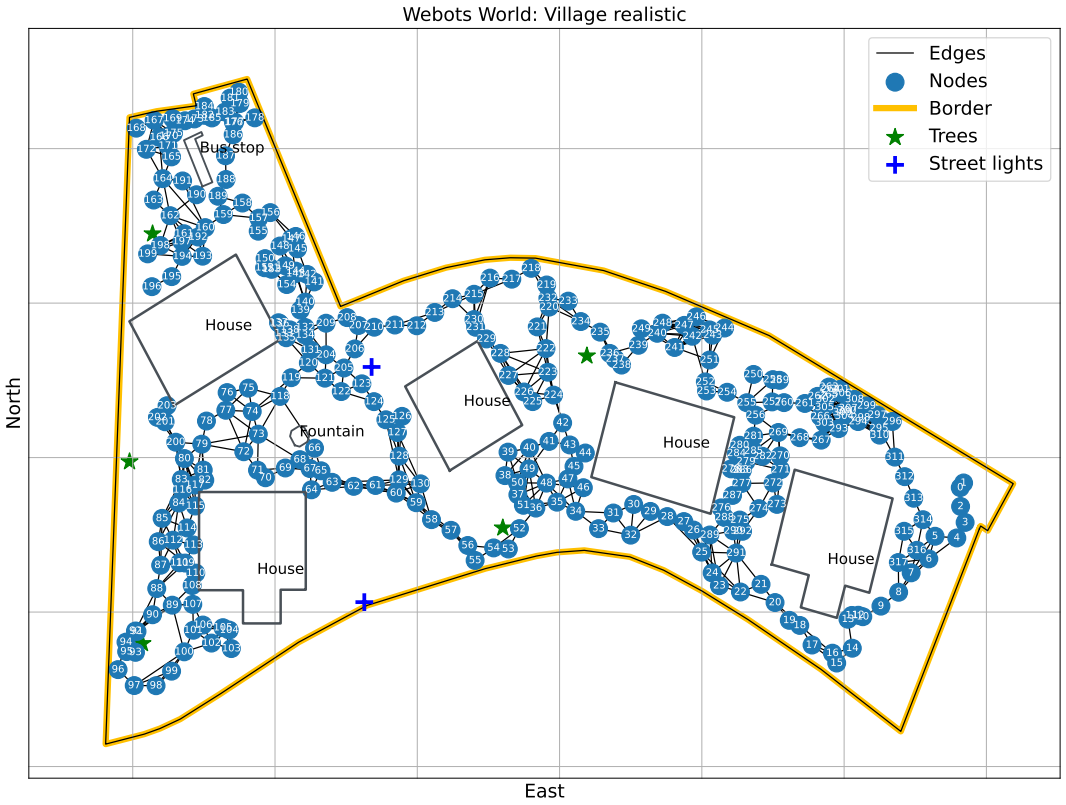


Figure 6.8: Constructed Topological Map for the Village Realistic with Adjusted Settings

of matched keypoints for the specified piece. Grid lines are drawn concerning the circle radius for each number of matched keypoints starting from 1 to 7. If a limitation is caught concerning the keypoint longitude distribution, method-4 is applied to create a more accurate heading. To do so, the longitude circle is divided into 18 pieces and the number of matched keypoints are computed for each piece. When zero matched keypoints are extracted from 9 consecutive pieces, method-4 started to be used as the heading estimation methodology, method-1 used in other situations. As a future improvement, the keypoint priority assignment function, which is assigned as the exponential function through the thesis, could be tuned to compensate more generic domain.

After the mapping and exploration is started with the addition of above methodologies, the constructed topological map for the world is shown in Figure 6.13. As can be seen from the figure that the map covers most of the world to perform the topological navigation task.

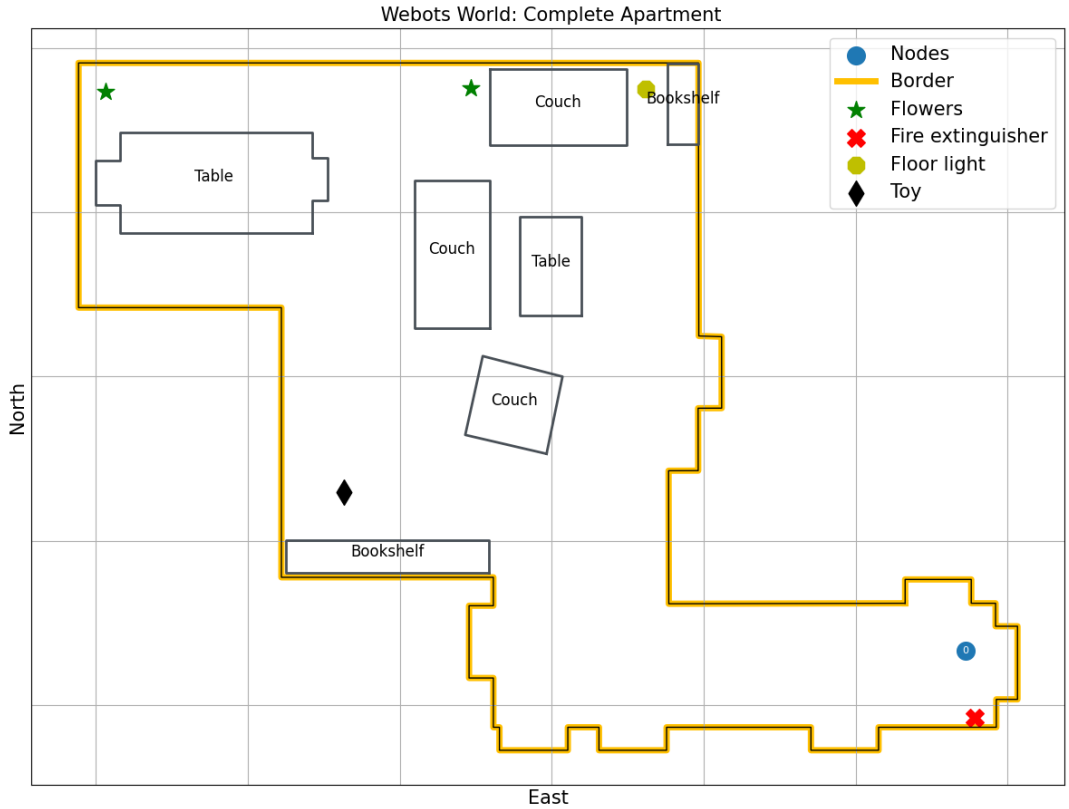


Figure 6.9: Mapping and Exploration Start Location for the Complete Apartment

6.1.3 Break Room

The mapping and exploration phase for the break room world is started from the node location shown in Figure 6.14. Since the break room is an indoor world, the methodology introduced to extract heading in Section 6.1.2 is directly used to construct a topological map. It is safe to say that no extra issues are detected in the mapping and exploration phase of the break room world and the constructed topological map is represented in Figure 6.15.

6.1.4 Factory

The mapping and exploration phase for the break room world is started from the node location shown in Figure 6.16. Since the environment includes both indoor and outdoor fields, it can be classified as an hybrid. However, the topological navigation and mapping application is performed in a controlled outdoor area and the heading estimation methodology is kept as applied in indoor worlds. Extra issues are not

Figure 6.10: Map Nodes 1 and 7 Locations

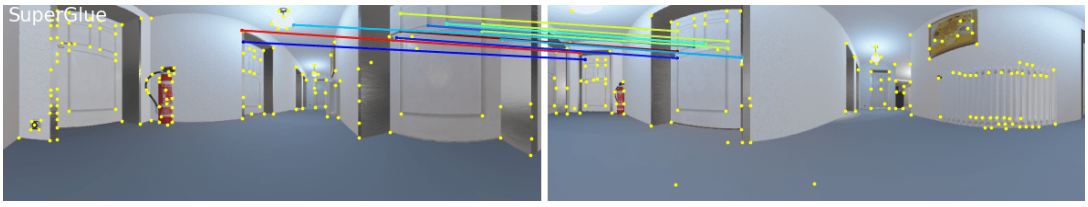


Figure 6.11: False Positive Matches due to General House Features

detected in the mapping and exploration phase of the factory world, and the transition from outdoor to indoor is safely executed.

In overall, in the mapping and exploration phase, it is detected from all maps that node spacing differs directly proportional with the surrounding object count and their spacing. In terms of field change, node spacing in indoor is denser than outdoor due to less spacing among surrounding objects. When there are more objects with less spacing in the same field, the map node spacing decreases as well due to high amount of feature change with the robot movement. The feature change directly affects the keypoint repeatability, hence, the similarity between two frames are affected significantly at each robot movement.

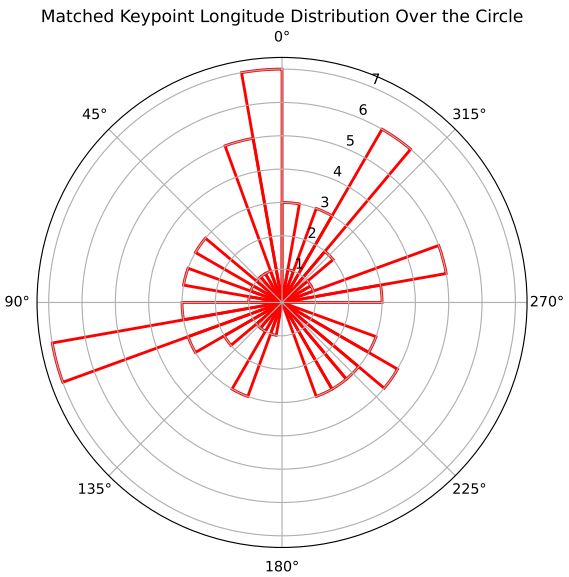


Figure 6.12: Polar Histogram of Keypoint Distribution

6.2 Topological Navigation

After a successful mapping and exploration phase, the next main task for the robot to perform is topological navigation. Topological navigation starts when a target frame is set to the algorithm, and if the target frame is close enough to a map node to perform the topological navigation. As stated in the topological navigation flowchart Figure 4.5, the first task is the localization of both the current and the target frame to find their nearest map nodes as shown in Figure 4.2. After they are localized, the path is defined to move the robot towards the assigned goal, which is finding the target frame. Since our topological navigation framework is divided into local tasks due to the sensor range of spherical cameras, intermediate locations, i.e. via points, between the current frame and the target frame are defined in the path planning to ensure continuity. Then, a heading vector is computed to move towards the current via point. In topological navigation, changing the local task is established concerning the similarity of both the current via point and the next via point with the current frame. In addition, different similarity thresholds are assigned compared with the ones in the mapping and exploration phase. The minimum similarity threshold $S_{th,min}$ is taken from the Section 3.2 Heading Estimation Algorithm Comparison as 60. Since the maximum similarity threshold $S_{th,max}$ is specified as 110 in the mapping and

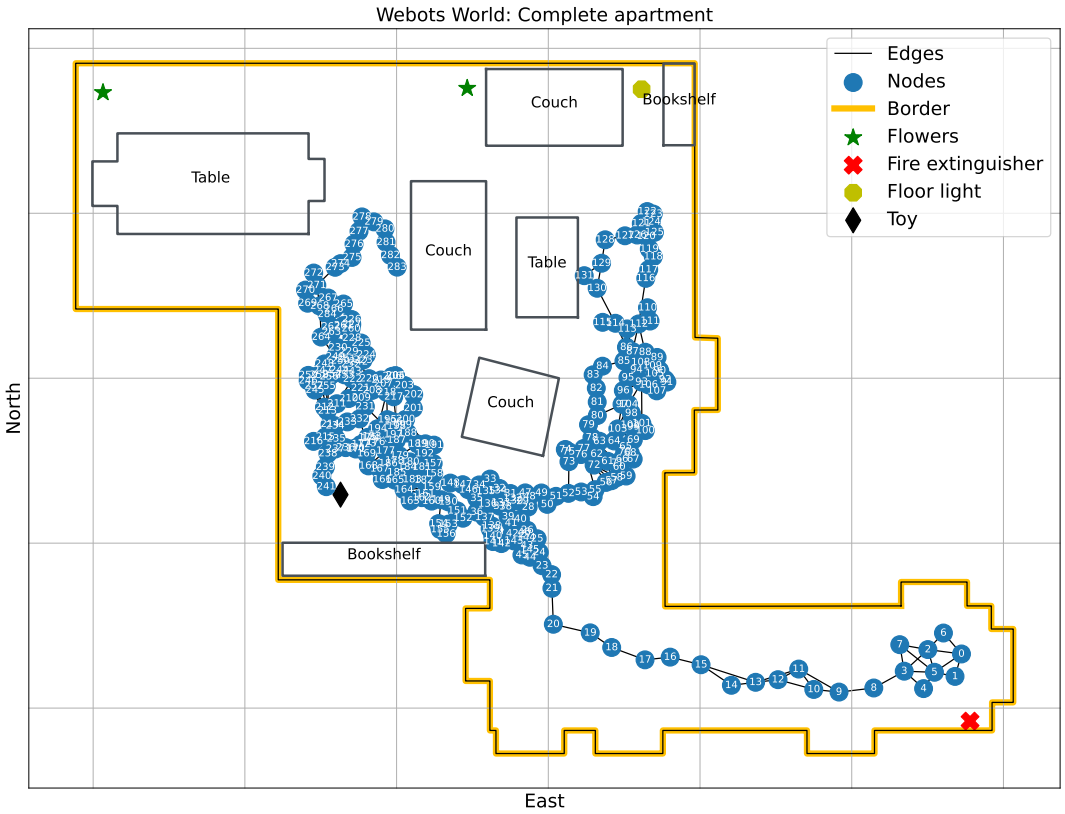


Figure 6.13: Constructed Topological Map for the Complete Apartment

exploration, it is kept smaller in the topological navigation phase to increase the via point index even if the current via point is not reached completely, but the next via point is visible enough to continue performing the navigation. Therefore, $S_{th,max}$ is set as 90 for the navigation period of the framework. However, to finish the navigation process as close as possible to the target, $S_{th,max}$ is set as 110 when the target node is started following. As summarized in the equation (6.3), if the similarity of the current via point $S_{e,cv}$ is higher than $S_{th,max}$, i.e. current via point is close enough to the current frame, and the similarity of the next via point $S_{e,nv}$ is higher than $S_{th,min}$, i.e. next via point is similar enough to the current frame to perform navigation, then via point index is increased so that the robot starts moving towards the next via point. On the other hand, even if $S_{e,cv}$ is high enough to continue the navigation, it may drop below the threshold in certain situations. For instance, if an unfeasible edge is added to the map due to false positive matches in the mapping and exploration as shown in Figure 6.6, then the navigation for that edge will be noisy and will never be finished. Therefore, $S_{e,cv}$ will go below the threshold due to the misguided movement of the robot. If the $S_{e,cv}$ is lower than $S_{th,min}$ in any moment of the navigation, the

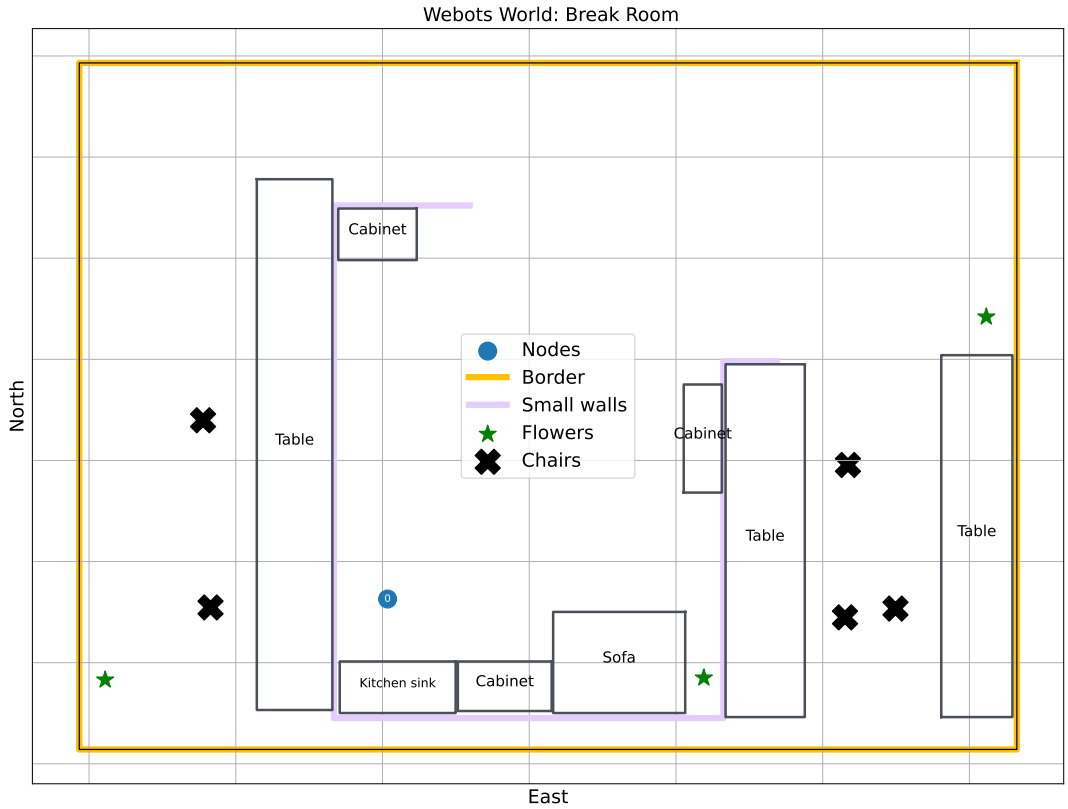


Figure 6.14: Mapping and Exploration Start Location for the Break Room

current edge is deleted and the path is reset as shown in (6.4). While performing the navigation, the velocity of the robot servo motors are assigned inversely proportional to the $S_{e,cv}$. Firstly, two velocity limits, the maximum velocity V_{max} and the minimum velocity V_{min}, are assigned as 10 rad/s and 6 rad/s respectively to keep the velocity in a certain range in outdoor. Since the indoor worlds are smaller than outdoor worlds, the velocity limits are divided by two in indoor and are assigned as 5 rad/s and 3 rad/s respectively. Then, a velocity gain K_v is assigned as in the equation (6.5) such that when the similarity is high between two frames, the robot is moved with a lower velocity vice versa. This is achieved by multiplying the V_{max} with the K_v as stated in the equation (6.6).

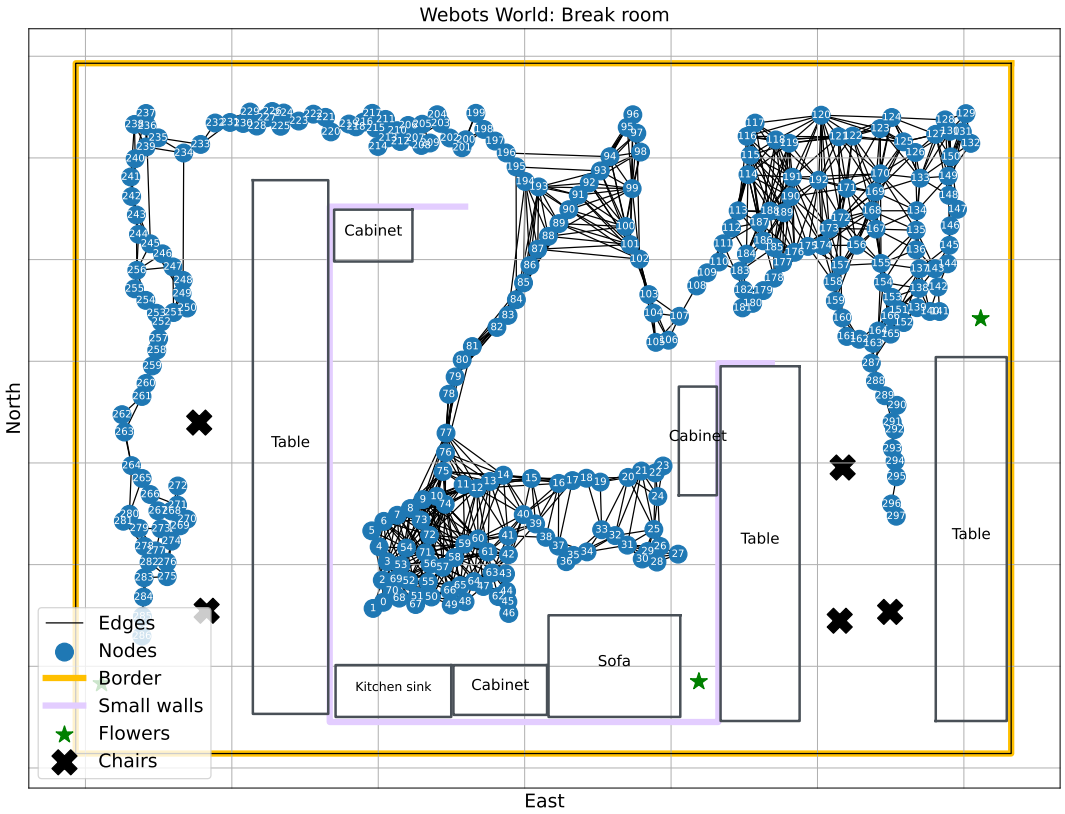


Figure 6.15: Constructed Topological Map for the Break Room

$$i_{vp} = \begin{cases} i_{vp} + 1 & \text{if} \quad (S_{e,cv} > S_{th,max} \; and \; S_{e,nv} > S_{th,min}) \\ i_{vp} & \text{otherwise} \end{cases}$$

where: (6.3)

$i_{vp} =$ via point index

$S_{th,max} = 90,$ The maximum similarity threshold

$S_{th,min} = 60,$ The minimum similarity threshold

$$\zeta_p = \begin{cases} 1 & \text{if} \quad S_{e,cv} < S_{th,min} \\ 0 & \text{otherwise} \end{cases}$$

(6.4)

where:

$\zeta_p =$ Path computation reset decision

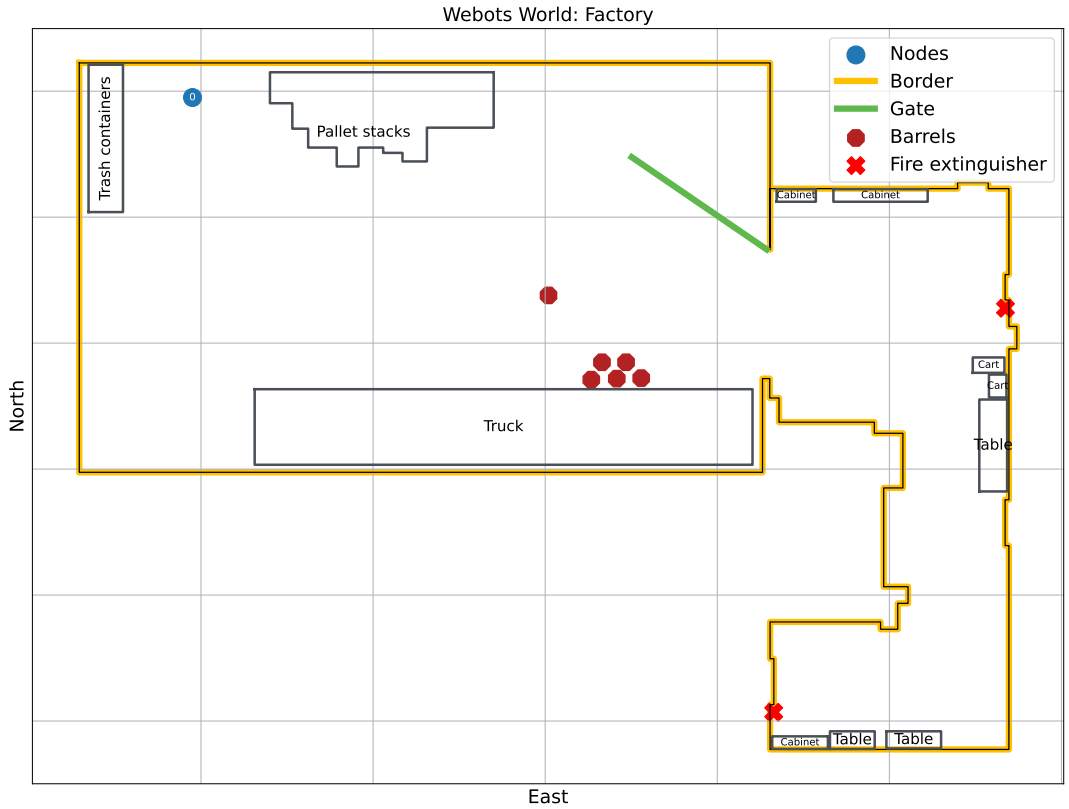


Figure 6.16: Mapping and Exploration Start Location for the Factory

$$K_v = \begin{cases} 1 - \frac{0.4(S_{e,cv} - 0.6(S_{max}))}{0.4(S_{max})} & \text{if} \quad S_{max} > S_{e,cv} > (S_{max})0.6 \\ 1 & \text{otherwise} \end{cases} \tag{6.5}$$

where:

K_v = Velocity gain

$$V = K_v V_{max}$$

where:

$$V_{max} = \begin{cases} 10\ rad/s & \text{for outdoor} \\ 5\ rad/s & \text{for indoor} \end{cases} \tag{6.6}$$

= The maximum rotational velocity of the robot servo motor

While performing the topological navigation in the Webots world, two robots are created as in Figure 6.18 to play a different version of hide and seek. While one robot, i.e. the hider, is hidden in a random place on the map, the other one, i.e.

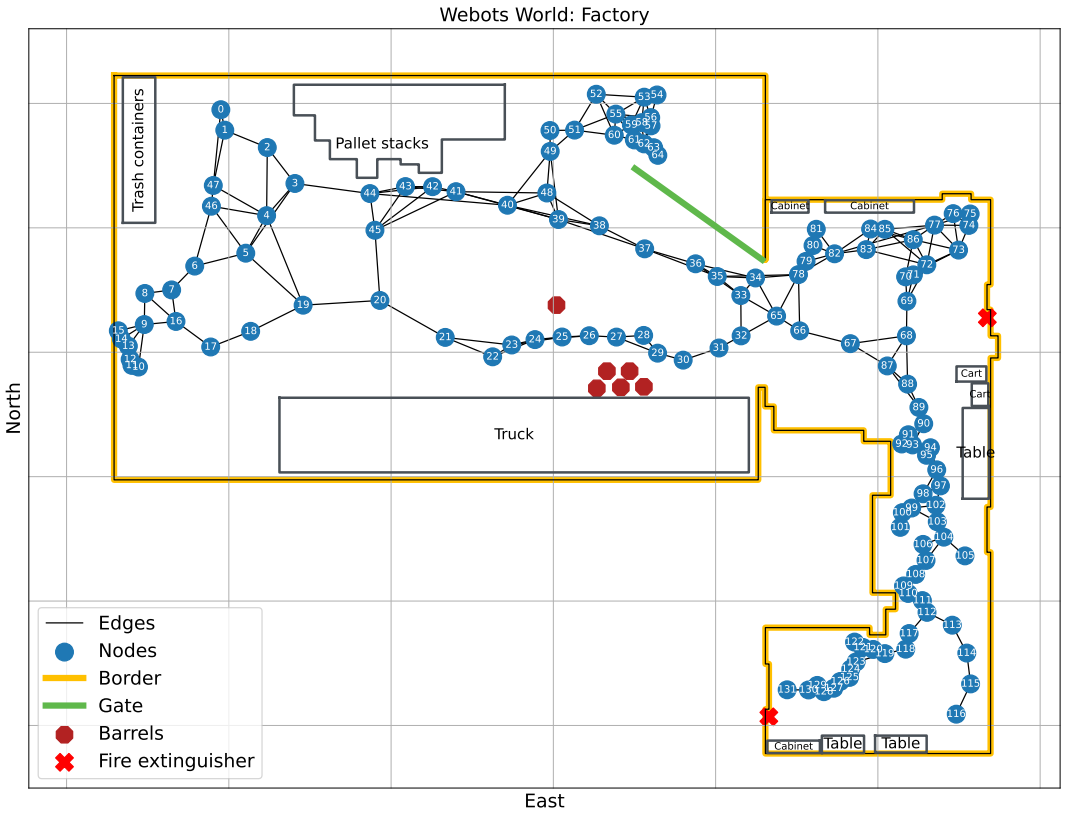


Figure 6.17: Constructed Topological Map for the Factory

the seeker, will try to find the hider by performing topological navigation. In the following sections, the hide and seek will be played several rounds to investigate algorithm performance in different Webots worlds.

6.2.1 Village Realistic

The hide and seek play consists of five rounds for the village realistic world. In the first round, the start node and the target node locations are illustrated in Figure 6.19 with a green circle and a red circle, respectively. In the figure, while the start node specifies the closest map node location to the current frame, the target node is directly located at the target frame location. The main idea behind the representation is to show each local task of the topological navigation by highlighting the first and the last one with special colors. As already mentioned in the algorithm flowchart in Figure 4.5, the next steps will be the path computation and the via-point navigation. The followed path and the via points are also added in the Figure 6.19. While the via points are highlighted with purple, the followed path by the seeker through the

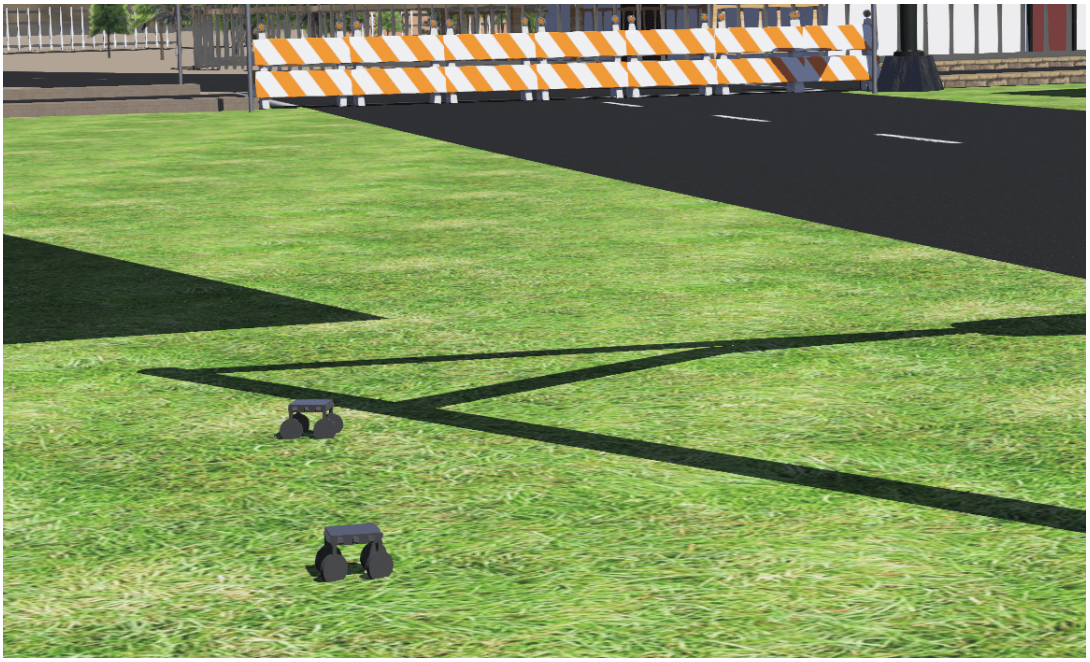


Figure 6.18: Two Robots Created to Use in Topological Navigation

navigation process is drawn with orange. The grid lines in the figure are generated with equal spacing of 5 meters to ease the investigation. As also seen from the path that the navigation is finished earlier than the target node point. Hence, topological navigation is not so-called accurate concerning the metric properties, but it is a framework that achieves moving a robot close enough to the set target. To better illustrate the final moment of the first round of the navigation process, Figure 6.20 is attached. In the figure, while the top left frame is showing the spherical camera output of the seeker, the top right frame displays the spherical camera output of the hider. Even though robots do not overlap each other, they are neighboring enough to say that the goal is accomplished.

When the second round is started, the seeker is kept at the location where it finished the first round, but the hider is placed in a different location as shown in Figure 6.21. Further, the followed path and via points are also labeled in the figure Figure 6.21. As can be seen from the followed path, the process is executed successfully and the seeker has reached the target frame without overlapping with the hider.

The same procedure is applied while switching to the third round so that the hider is moved to another location and the seeker is kept at the place where it finished the second round. The start node and the target node are shown along with the followed

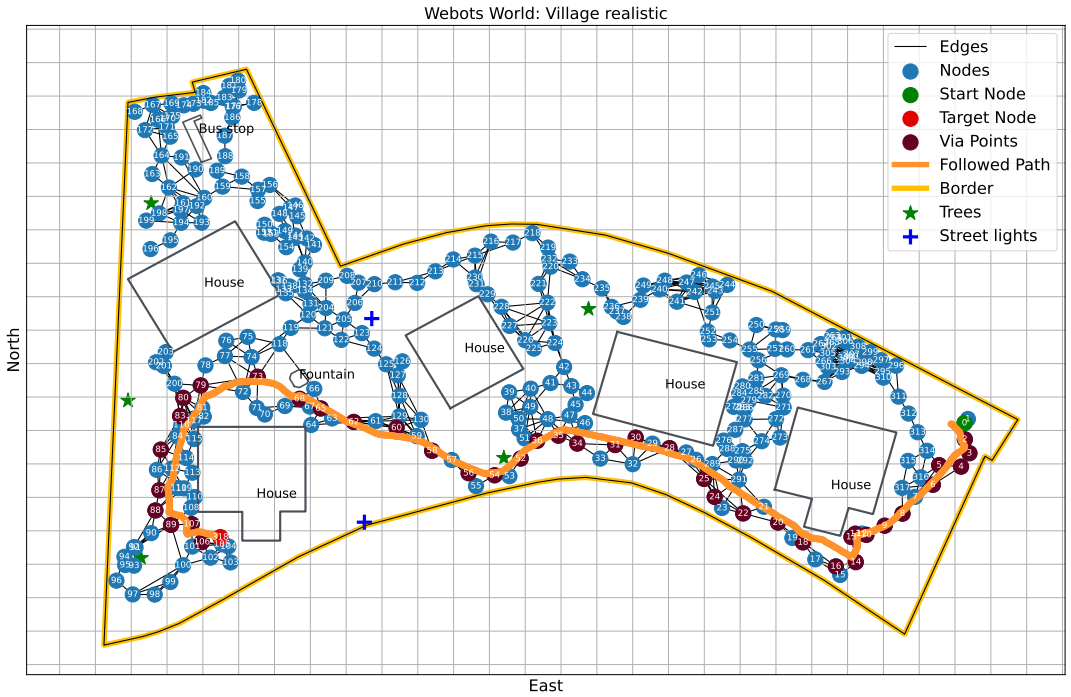


Figure 6.19: Village Realistic Followed Path for the First Round

path in Figure 6.23. The seeker successfully found the target without any faulty motion as seen in the figure. However, at the point where topological navigation ends, the seeker is relatively closer to the hider compared to the first two rounds.

In the fourth round, the hider is placed at the location labeled with the red circle in Figure 6.25, and the seeker is held where it finished its navigation process in the previous round. While no issues are found during the topological navigation, the navigation operation finished as wanted as seen in Figure 6.26.

In the last and fifth round for the village realistic world, the hider is carried to the target node in the Figure 6.27. As can be noticed also from Figure 6.27 that the navigation is finished after the seeker is overlapped with the hider, and the final scene is added to the Figure 6.28 for better illustration. Thus, the metric distance of the robots when the navigation goal is satisfied might differ from each other depending on the situation. Therefore, it is the validation point to say that topological navigation is not the process to execute precise metric operations and create the shortest path regarding metric properties.

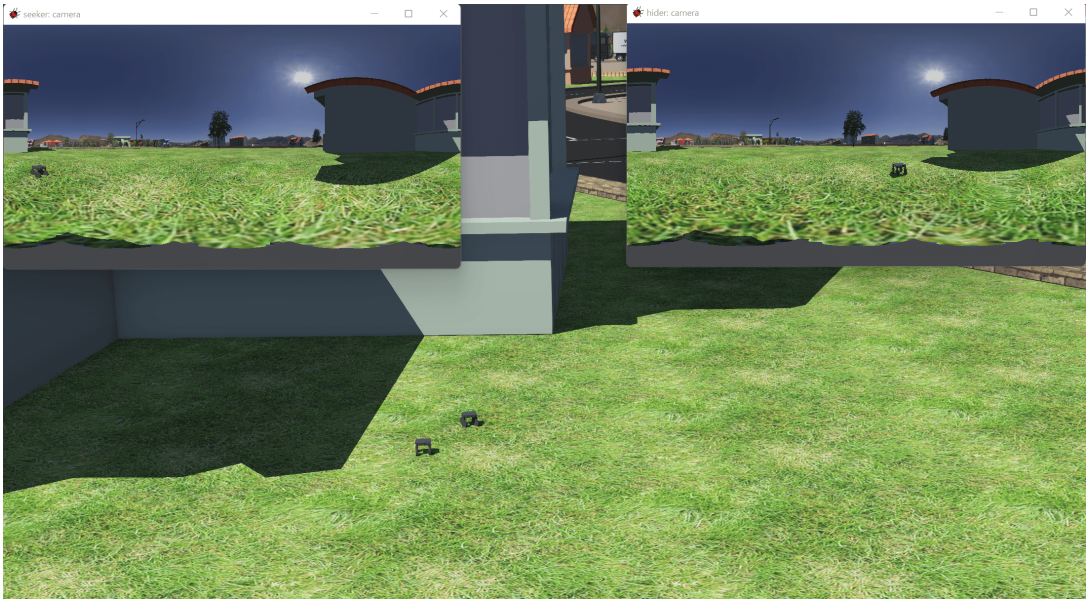


Figure 6.20: Village Realistic Final Scene for the First Round

6.2.2 Complete Apartment

The hide and seek is accomplished with three rounds in the complete apartment world. The first round is successfully completed concerning the start and target nodes specified in Figure 6.29. The grid lines in the figure are drawn with 0.5 meters of equal spacing for a better illustration of metric properties. The screenshot taken from the moment, in which the navigation task is completed, is demonstrated in Figure 6.30. As seen from the figure, the hider and the seeker overlap with each other at the final moment.

In the second round, while the hider is moved to the location specified with the red node in Figure 6.31, the seeker is kept at the point where it finished the first round. After the round is finalized, the followed path is also shown in Figure 6.31 and the final snapshot concerning robot positions and their spherical camera output is shown in Figure 6.32.

In the third round, same as in previous rounds, the hider is taken to another location shown with the target node in Figure 6.33. Since via point nodes 138 and 142 are close to the bookshelf, an oscillatory robot motion is detected due to the obstacle. Even though topological navigation is affected by the oscillatory motion due to the obstacle, the process is successfully executed. However, the obstacle avoidance algo-

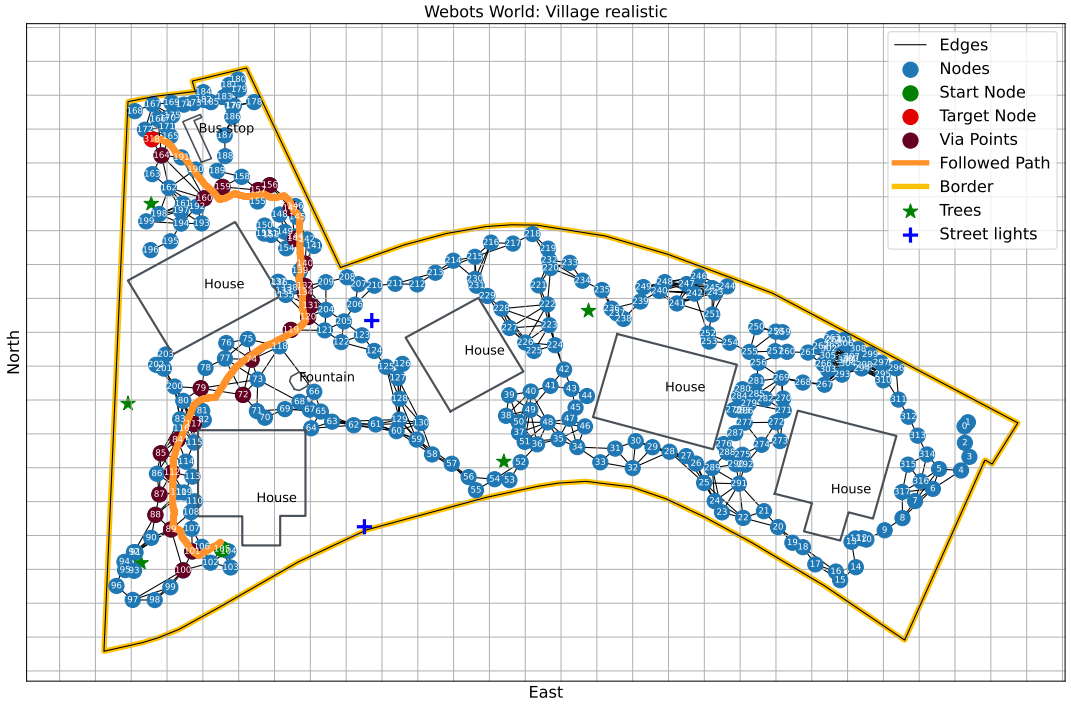


Figure 6.21: Village Realistic Followed Path for the Second Round

rithm could be reinforced with a memory-allocated candidate instead of momentary estimation as used in this thesis. The improvement opportunity for obstacle avoidance is kept as future work of this thesis.

6.2.3 Break Room

The hide and seek is executed for three rounds in the break room world. The first seeker and hider locations are highlighted with green and red nodes in Figure 6.35. Grid lines are generated with 0.5 meters of equal spacing for better illustration of metric properties. As can be seen from the figure in terms of the created map that when the robot moves in more open space, i.e. there are fewer obstacles around, the distance among map nodes gets closer, or more distant edges are created among map nodes. The first round is performed well as in Figure 6.35 and verified with the last snapshot in Figure 6.36.

In the second round, the exact same procedure was applied in the previous hide and seek rounds. The hider is taken to another place as shown with the target node in Figure 6.37, but the seeker is stood still at the point where it finished the first round.

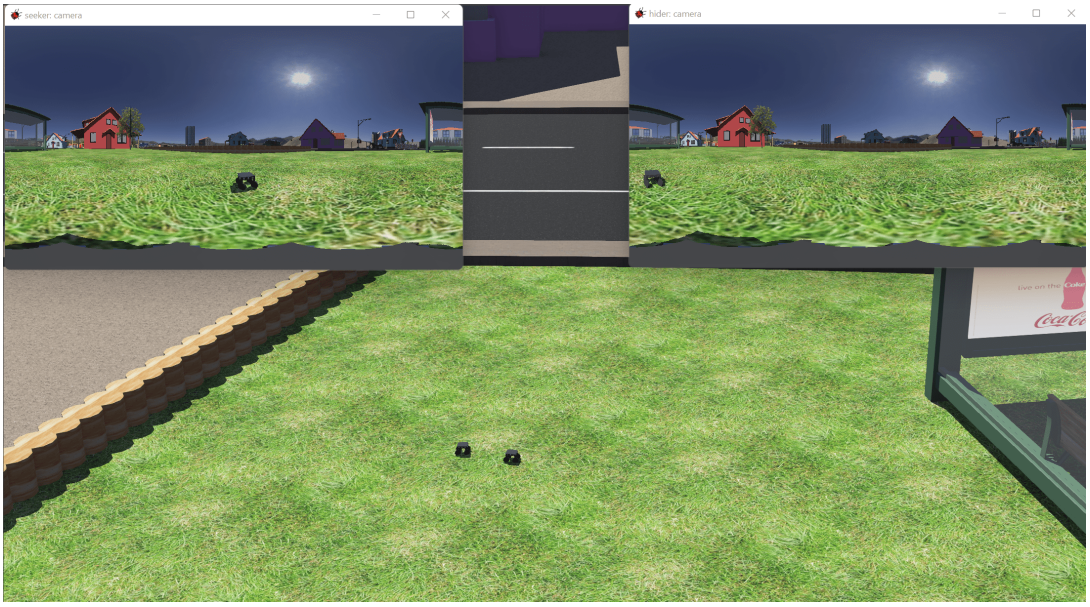


Figure 6.22: Village Realistic Final Scene for the Second Round

The second round is finished correctly and the robots overlapped each other at the final moment in Figure 6.38.

In the third and final round for the break room world, the hider is carried to the location shown with the red circle in Figure 6.39. The followed path labeled with orange scatter in the figure shows that the seeker was able to track the via points to reach the hider. As seen in the Figure 6.40 the hider and the seeker are over the top of each other, hence; the topological navigation process is concluded as intended.

6.2.4 Factory

The hide and seek is implemented for three rounds in the factory world. The first round is started from the start node to the target node as emphasized in Figure 6.41. According to the flow chart of topological navigation Figure 4.5, the next via point is started following when the current via point is close enough to the current frame and the next via point satisfies the minimum similarity threshold. Therefore, the followed via point by the robot could be changed before reaching the current via point location if the next via point is similar enough to perform topological navigation. At the path following process in the first round, a faulty robot motion is detected when node-49 is started to be followed by the seeker, and the start and end points of this false

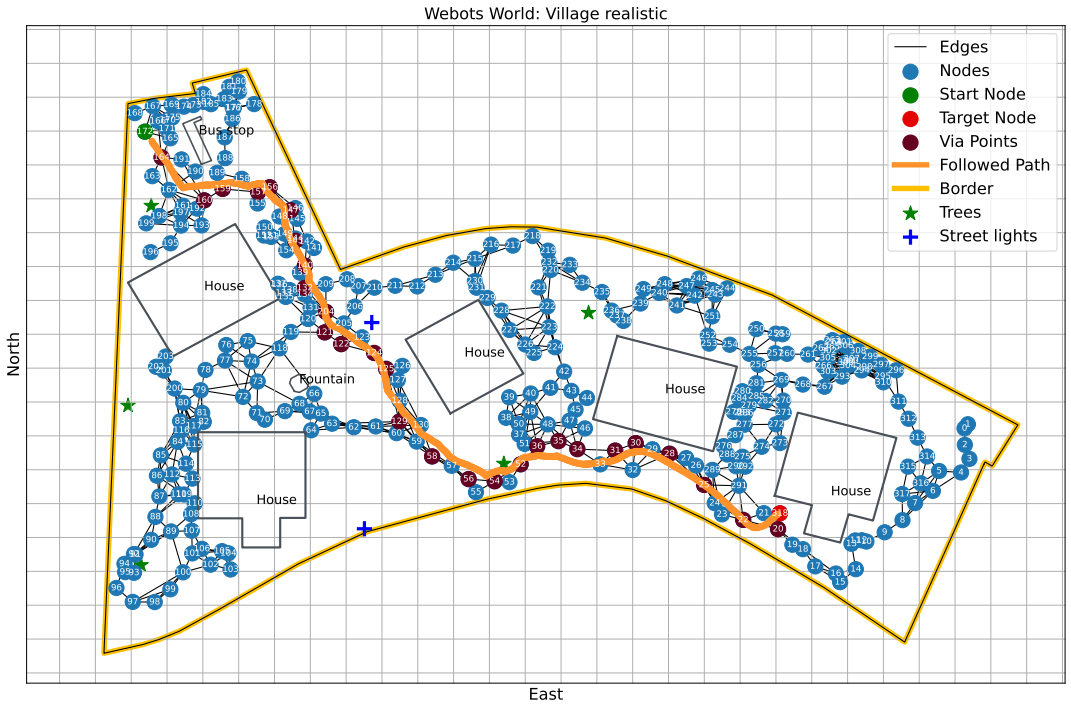


Figure 6.23: Village Realistic Followed Path for the Third Round

movement are labeled with pluses in Figure 6.41. The reason for the issue is due to the truck that covers most of the spherical frame and creates the only matched keypoints as can be seen from Figure 6.42. However, when the seeker starts seeing the pallet stacks and the gate from a similar perspective with the node-49, the heading estimate is corrected towards the node-49 as in Figure 6.43 and the topological navigation is concluded by finding the hider as in Figure 6.44.

In the second round, the hider is placed indoor part of the world, and the seeker is kept at the location where it finished the first round. As seen from the Figure 6.45, topological navigation for the factory world finished correctly and robots overlapped each other at the point where navigation is ended in Figure 6.46.

In the third and final round for the factory world, the hider is taken to an outdoor location close to the front of the truck as shown with a red circle in Figure 6.47. It is valid to say that the navigation process is handled successfully and the seeker was able to find the hider as verified in Figure 6.48.

In summary in this chapter, the proposed algorithms are put into practice to see the overall effectiveness of the topological navigation and mapping in a Webots world.

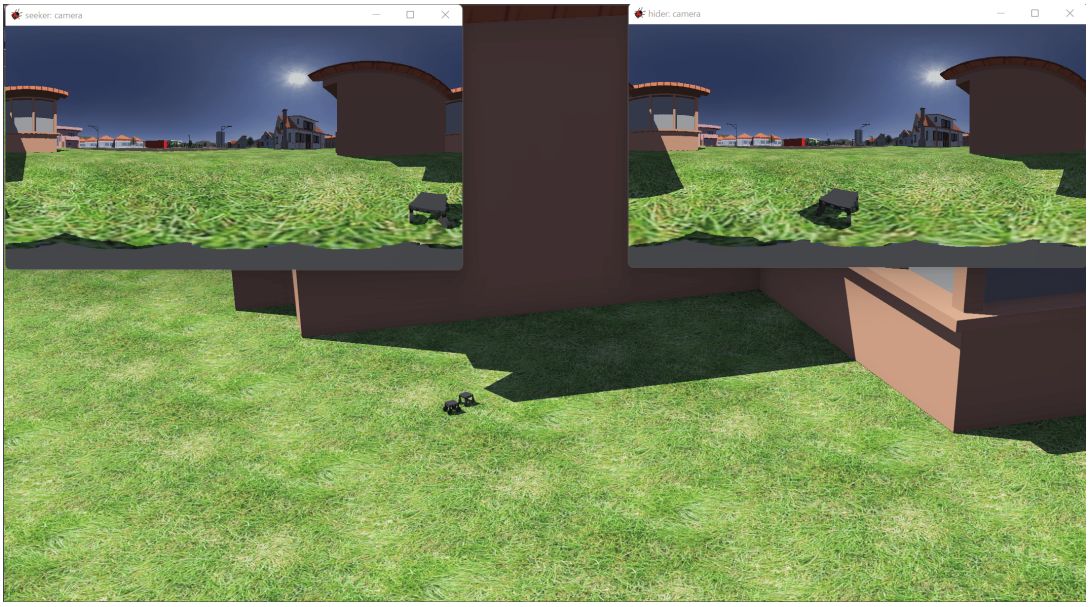


Figure 6.24: Village Realistic Final Scene for the Third Round

To monitor the main working structure, the algorithm performance is not investigated concerning the dynamic changes such as weather and illumination. However, it is detected that even though different drawbacks are identified, the proposed algorithms successfully performed both navigation and mapping phases in an unknown environment. In the next chapter, the thesis will be concluded by explaining the big picture and handicaps of the topological navigation and mapping framework.

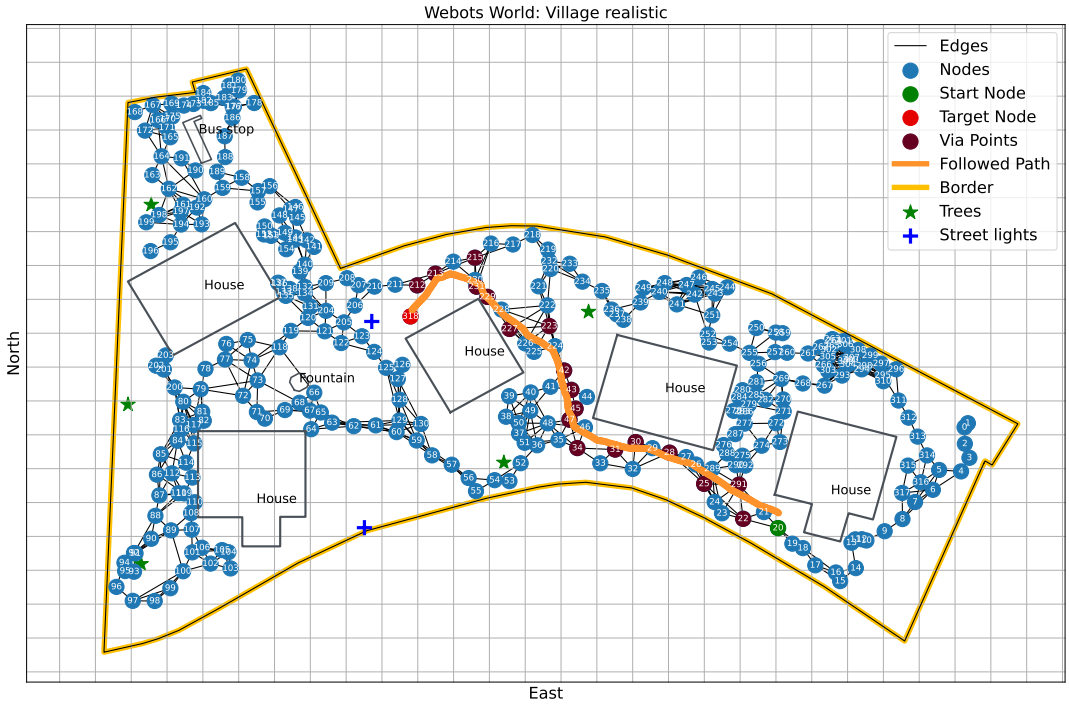


Figure 6.25: Village Realistic Followed Path for the Fourth Round

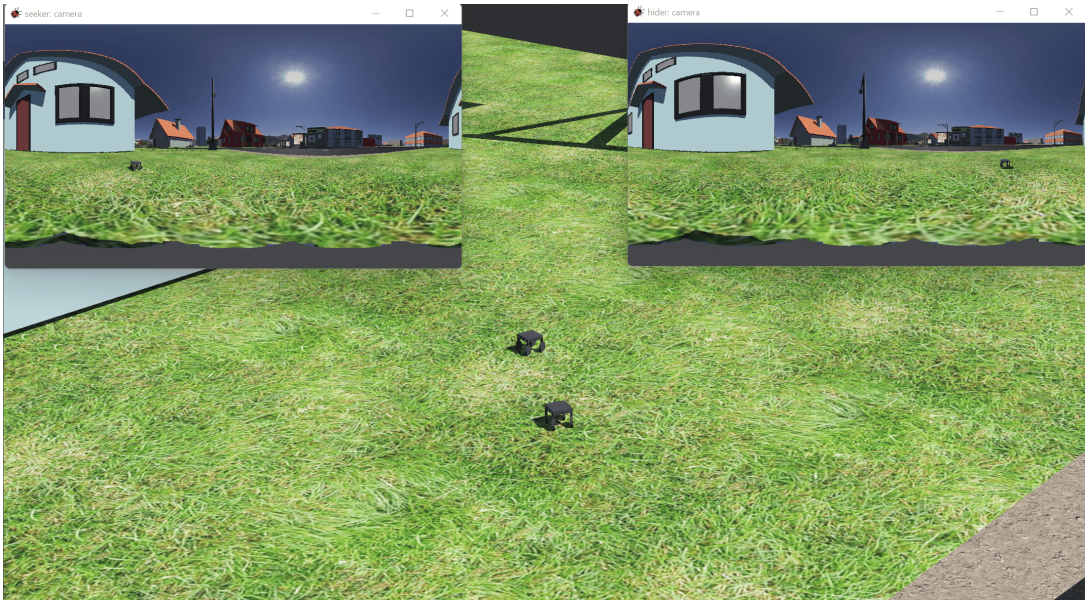


Figure 6.26: Village Realistic Final Scene for the Fourth Round

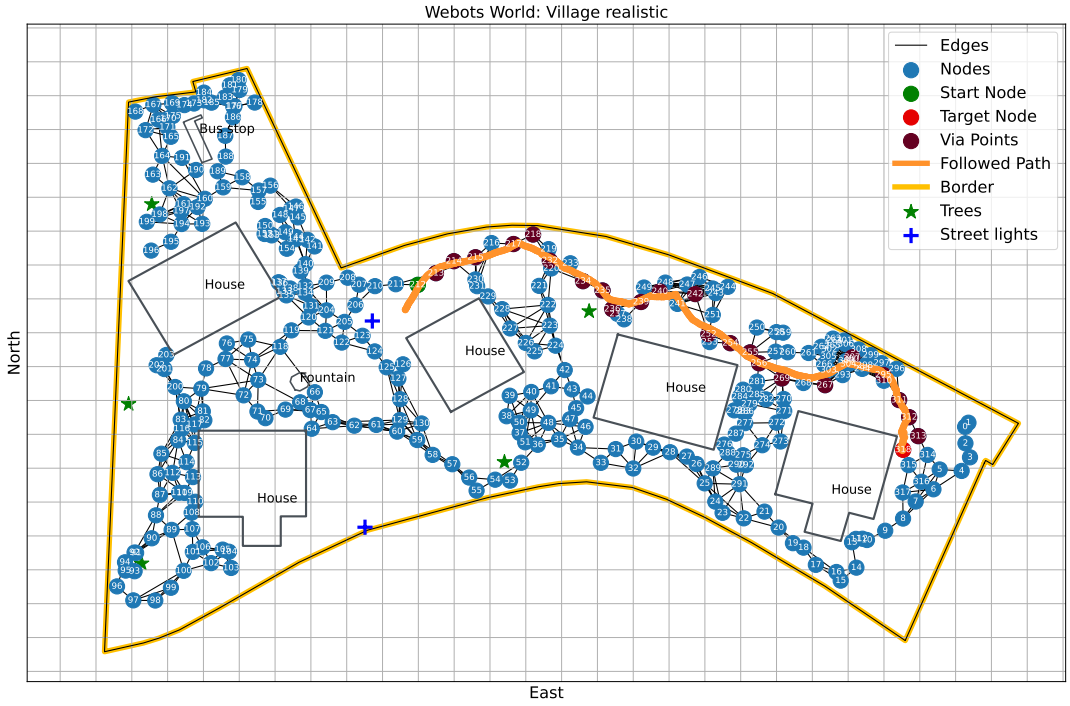


Figure 6.27: Village Realistic Followed Path for the Fifth Round

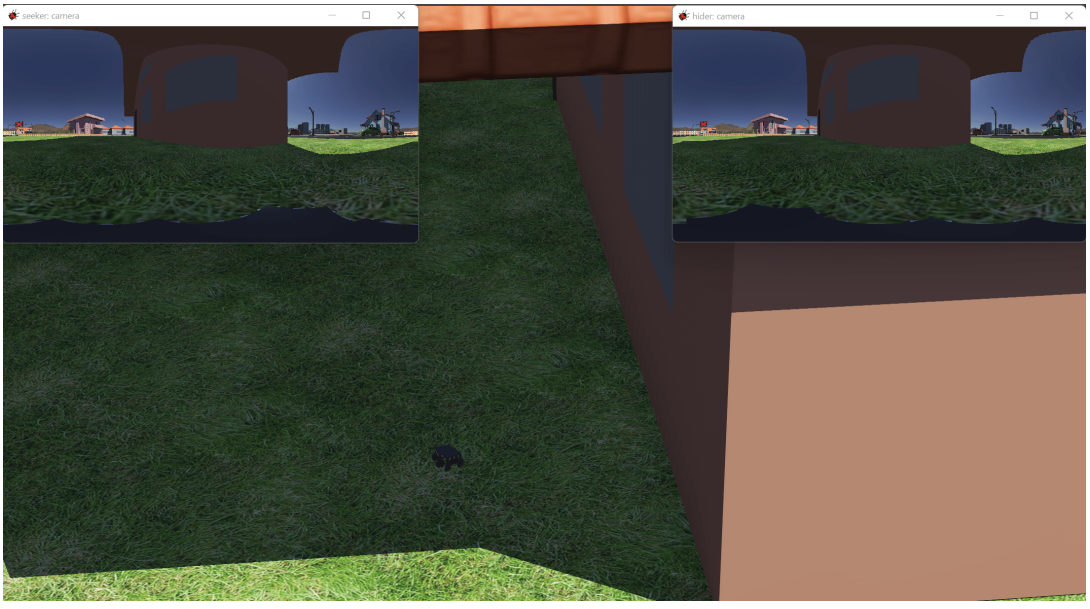


Figure 6.28: Final Scene for the Fifth Round

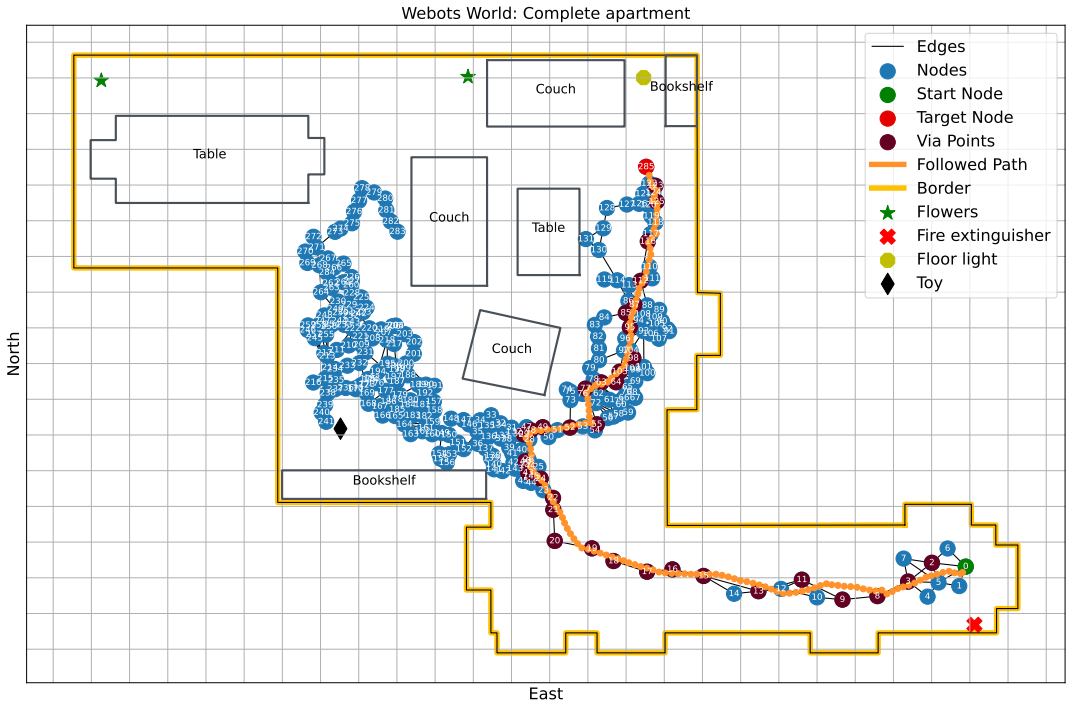


Figure 6.29: Complete Apartment Followed Path for the First Round

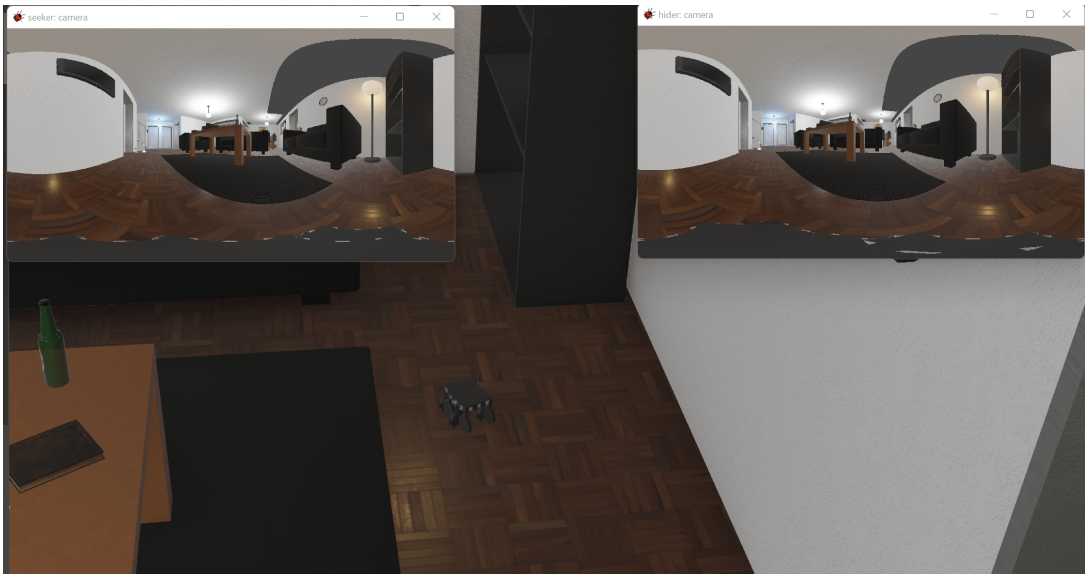


Figure 6.30: Complete Apartment Final Scene for the First Round

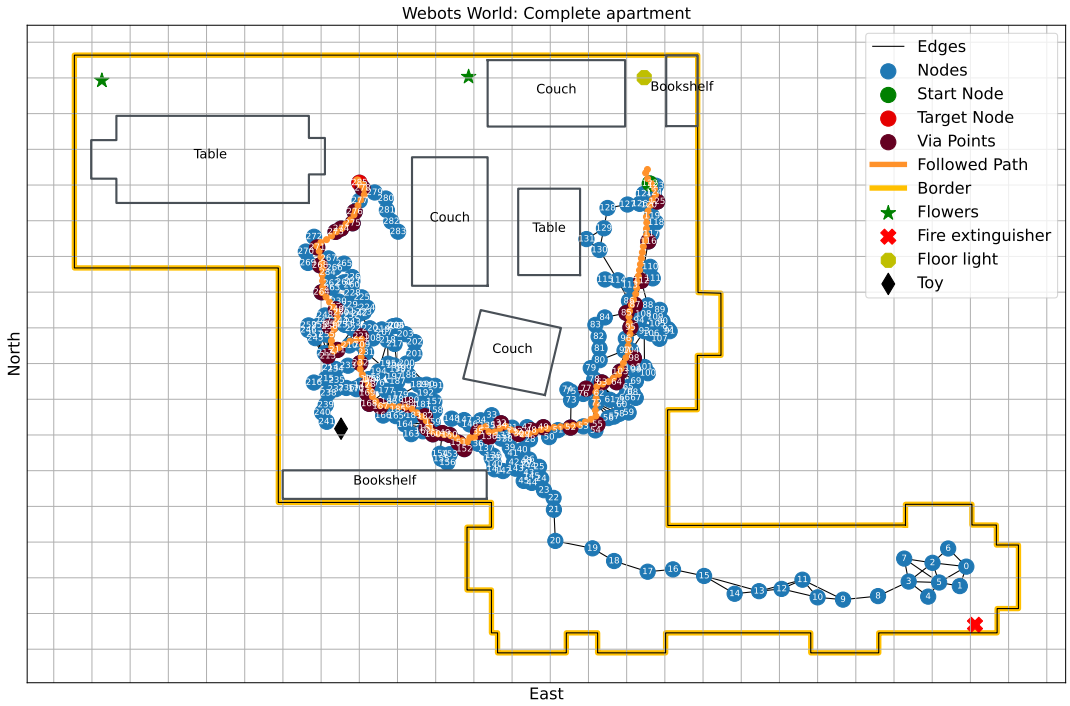


Figure 6.31: Complete Apartment Followed Path for the Second Round

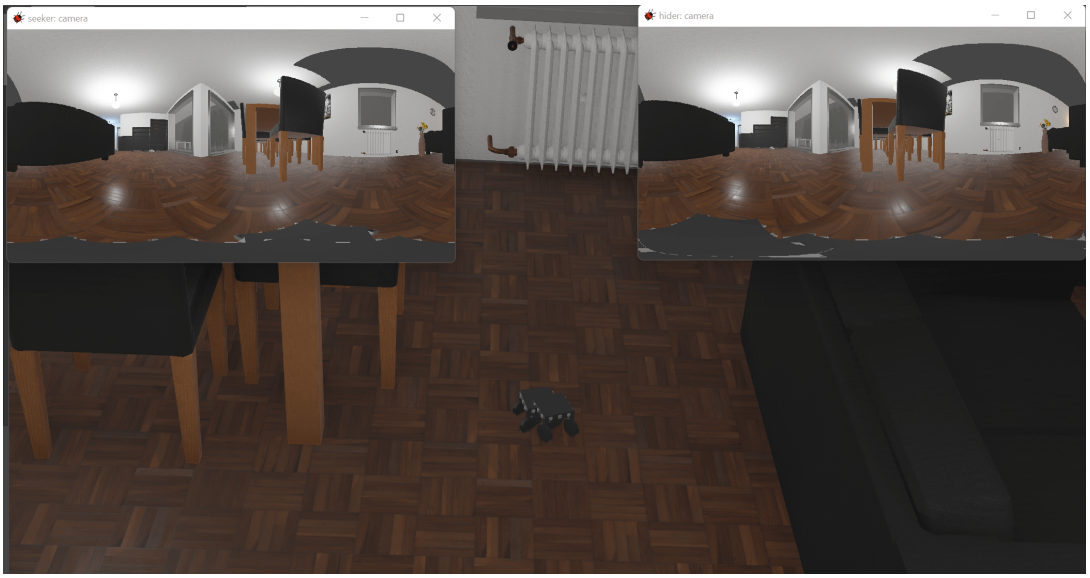


Figure 6.32: Complete Apartment Final Scene for the Second Round

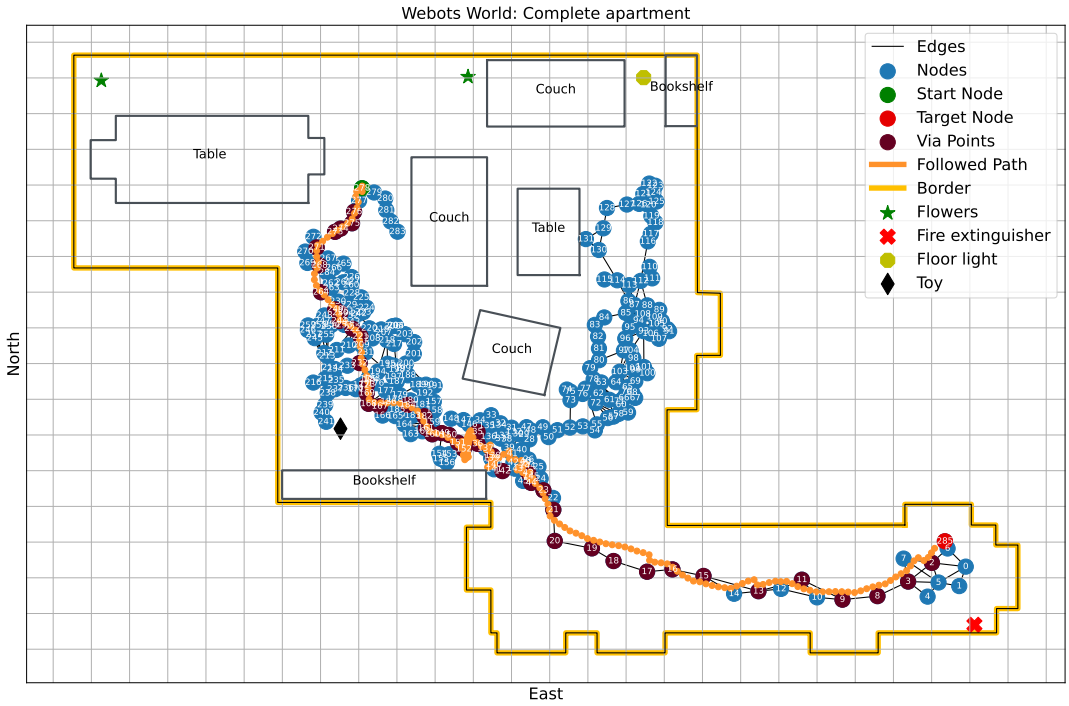


Figure 6.33: Complete Apartment Followed Path for the Third Round

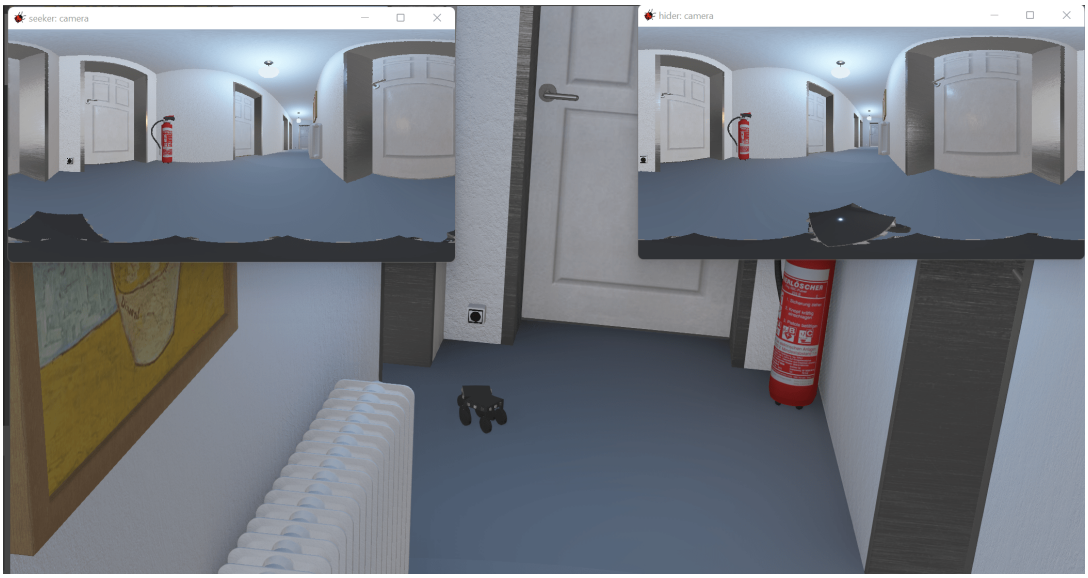


Figure 6.34: Complete Apartment Final Scene for the Third Round

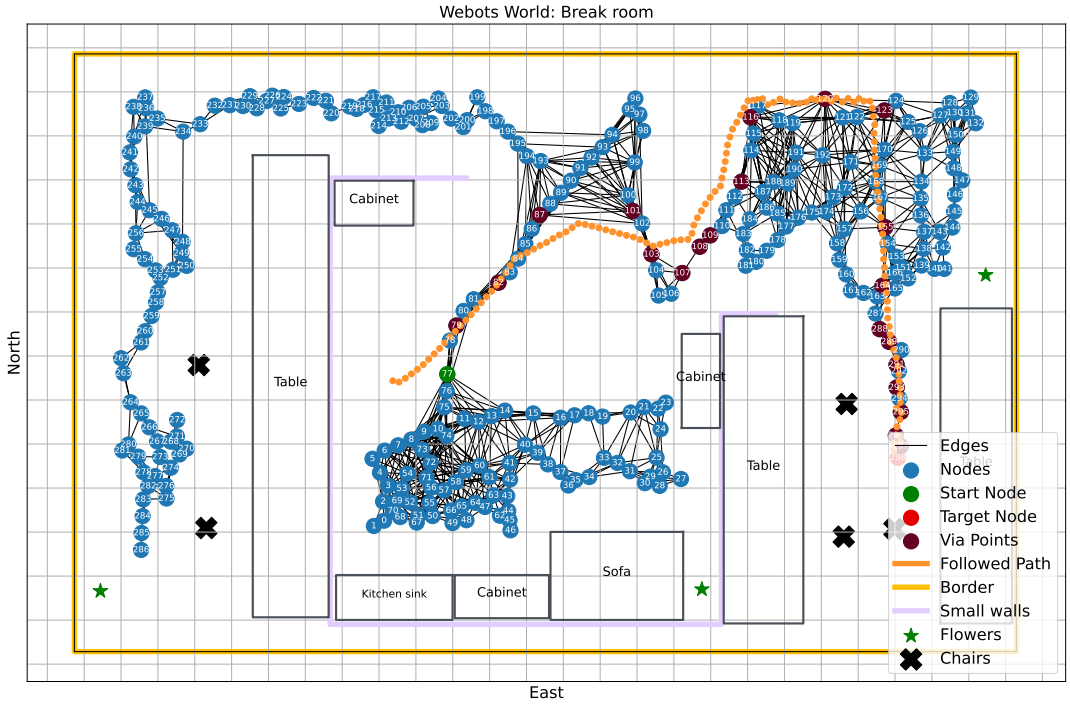


Figure 6.35: Break Room Followed Path for the First Round

Figure 6.36: Break Room Final Scene for the First Round

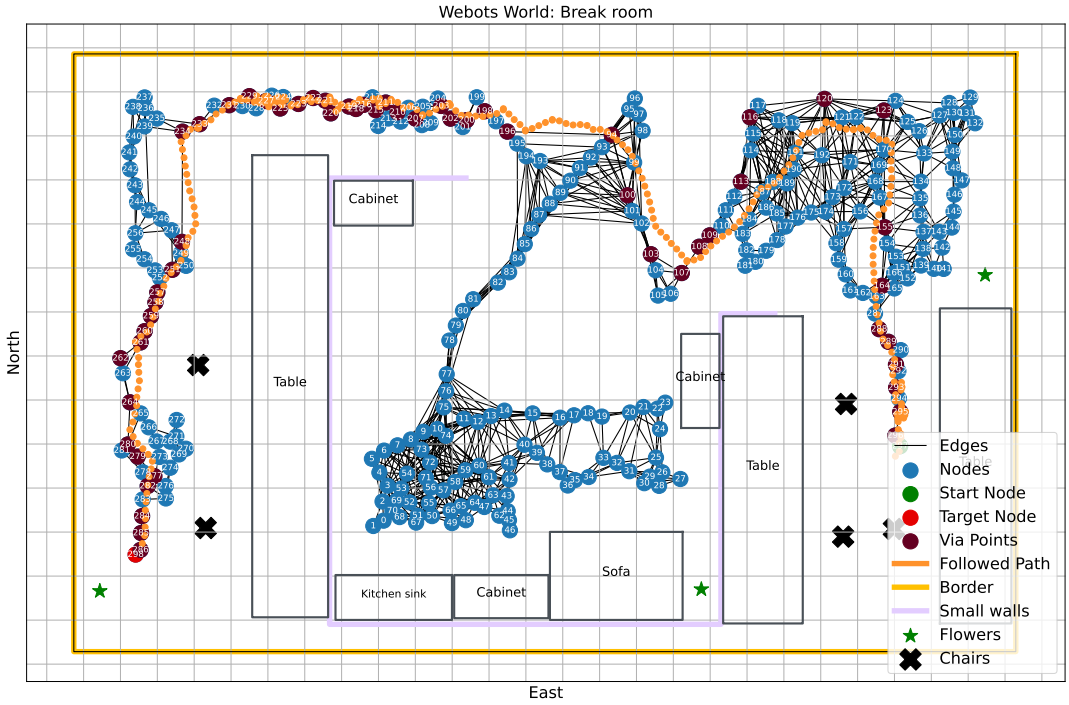


Figure 6.37: Break Room Followed Path for the Second Round

Figure 6.38: Break Room Final Scene for the Second Round

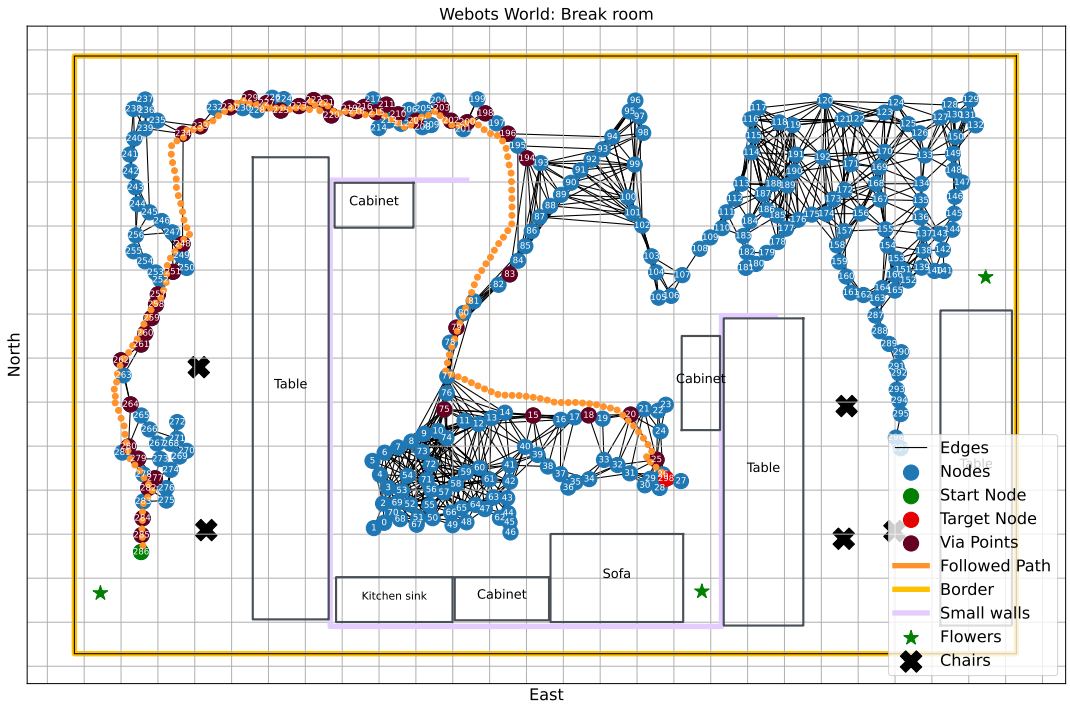


Figure 6.39: Break Room Followed Path for the Third Round

Figure 6.40: Break Room Final Scene for the Third Round

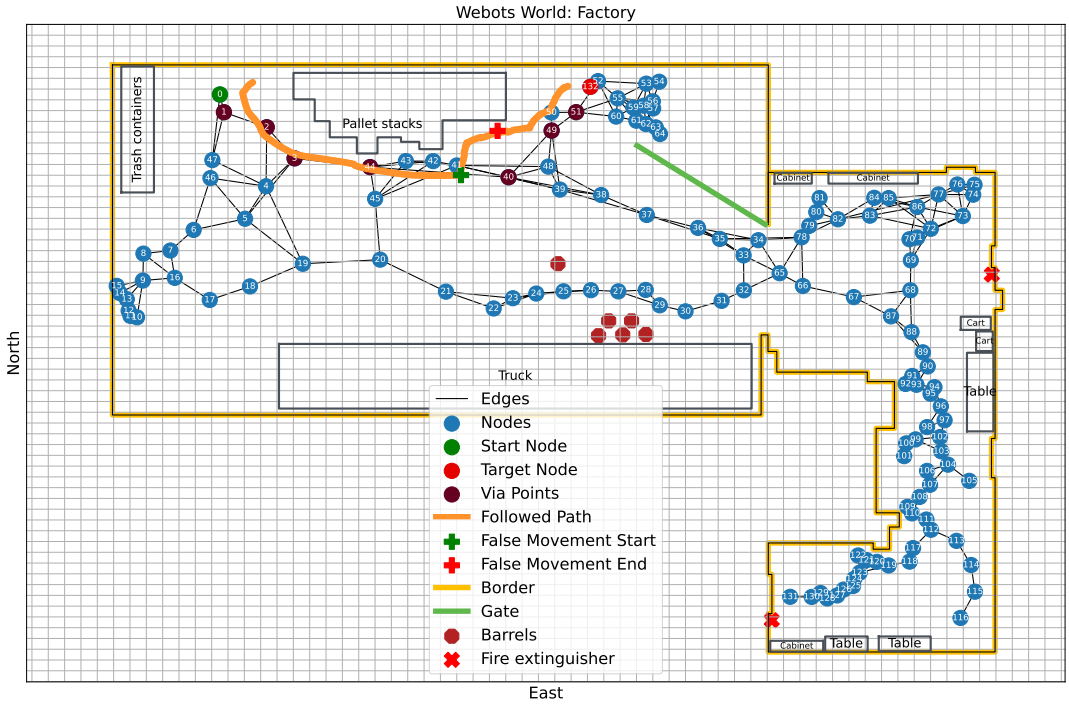
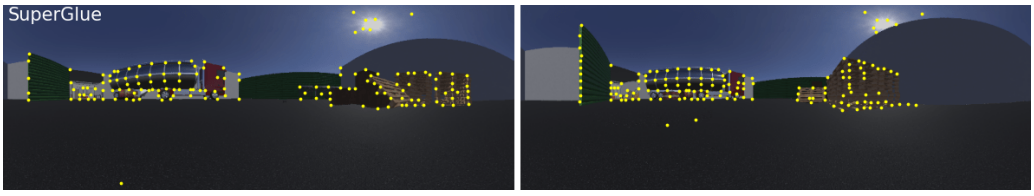
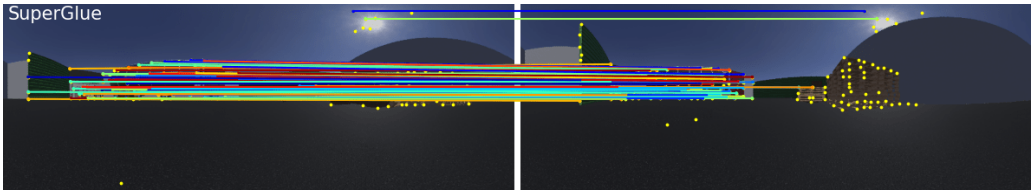


Figure 6.41: Factory Followed Path for the First Round

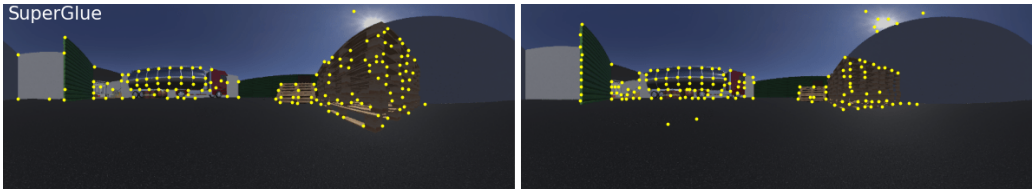


(a) Extracted Keypoints

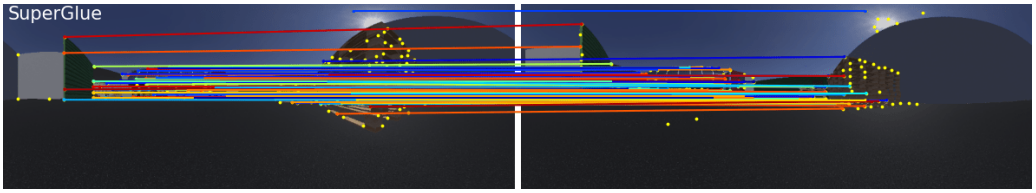


(b) Matched Keypoints

Figure 6.42: False Movement Start and Node-49 Local Feature Matching Results



(a) Extracted Keypoints



(b) Matched Keypoints

Figure 6.43: False Movement End and Node-49 Local Feature Matching Results

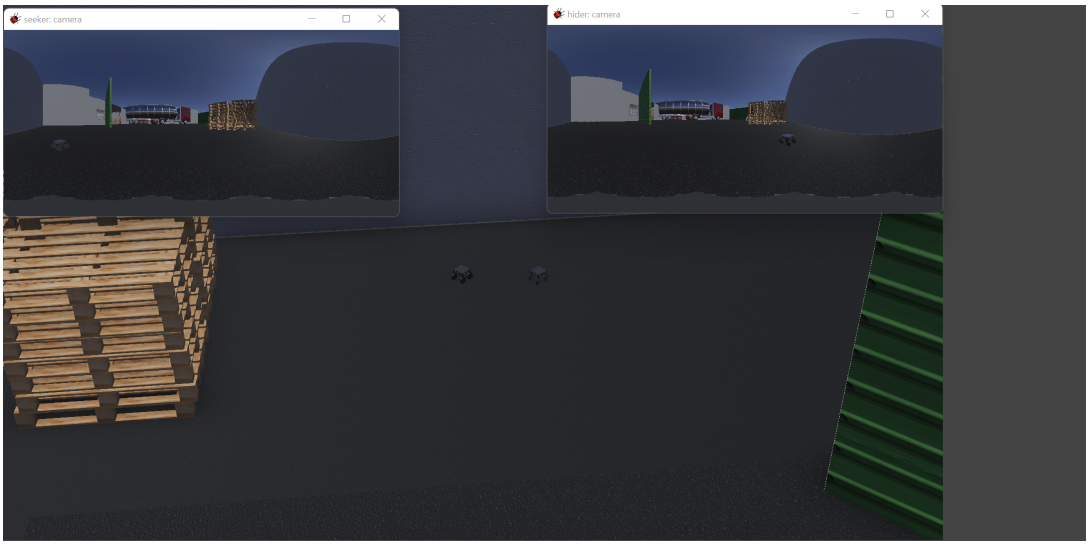


Figure 6.44: Factory Final Scene for the First Round

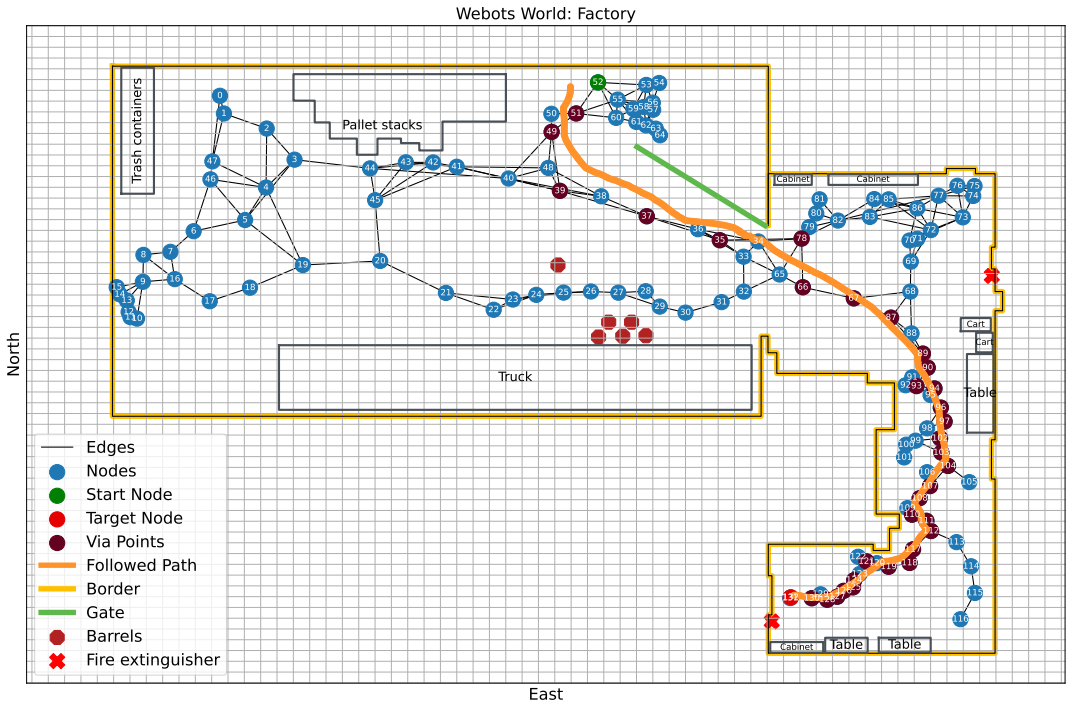


Figure 6.45: Factory Followed Path for the Second Round

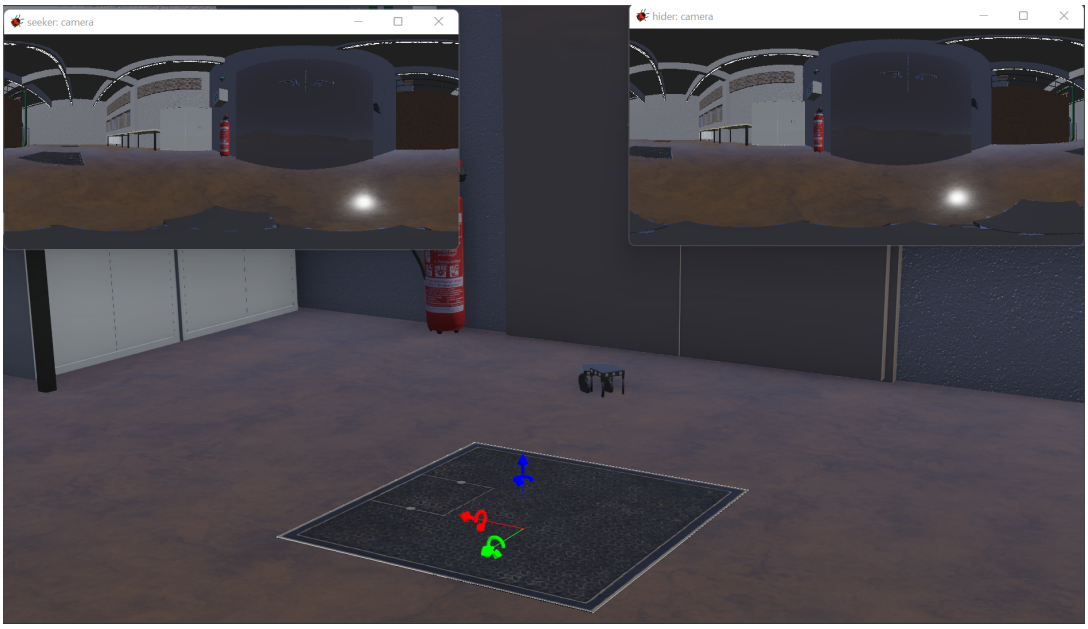


Figure 6.46: Factory Final Scene for the Second Round

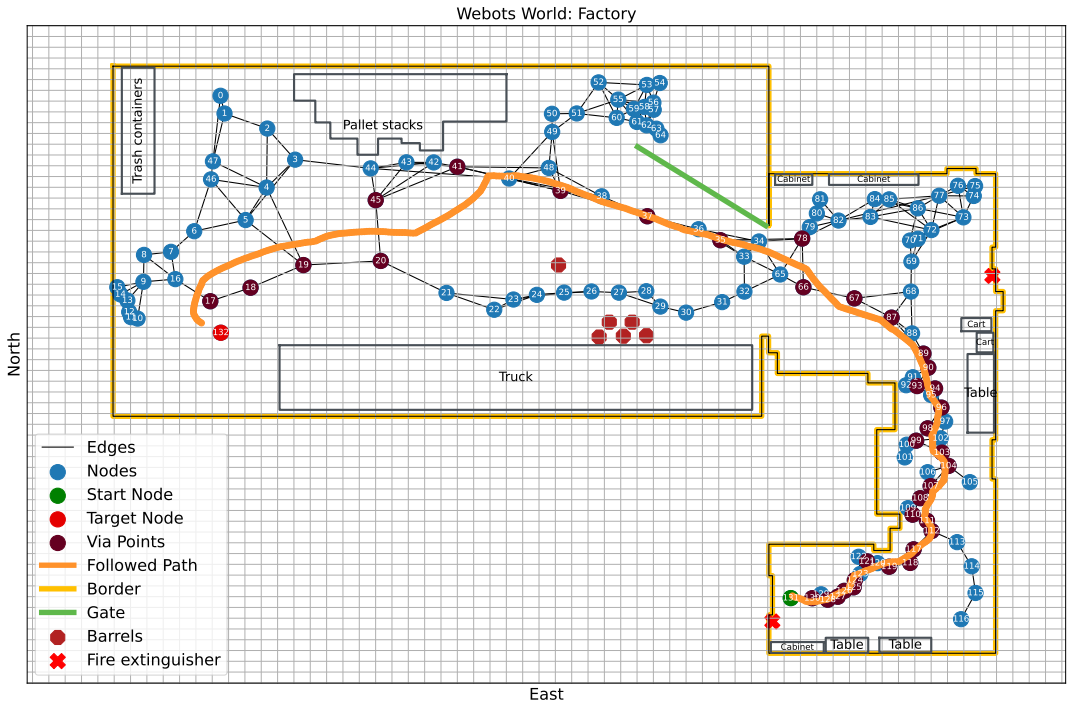


Figure 6.47: Factory Followed Path for the Third Round

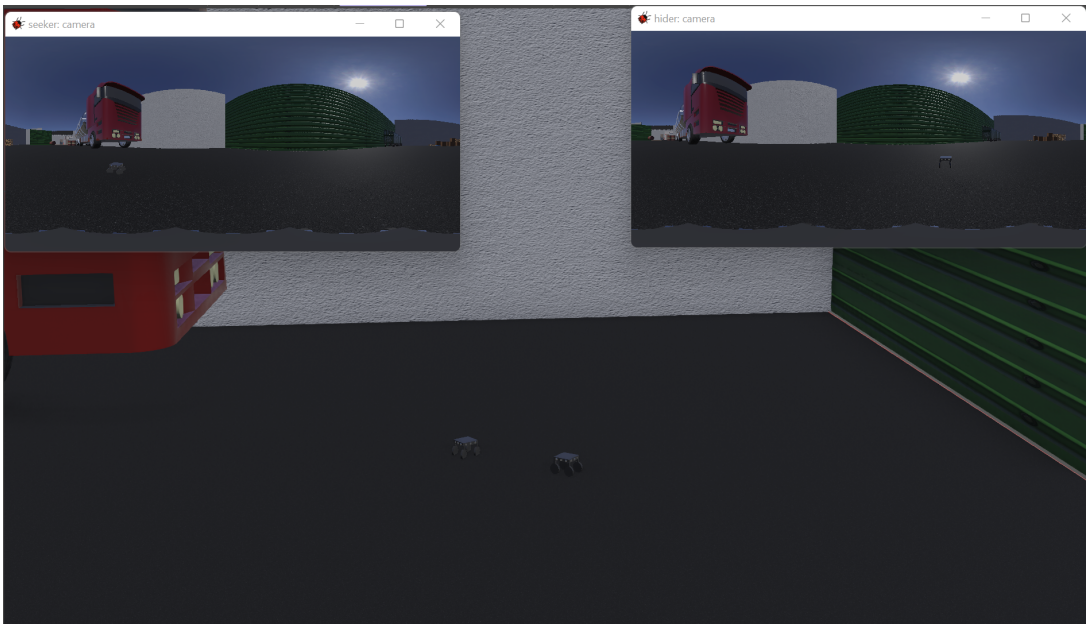


Figure 6.48: Factory Final Scene for the Third Round

CHAPTER 7

CONCLUSION

In this thesis, a topological navigation algorithm design and analysis has been explained by using spherical images as its primary source. Further, additional analyses are performed to find both the most convenient keypoint feature detector and feature matcher, i.e. local feature matching, algorithm combination, and the most practical heading estimation method. During the development, an object-oriented Python library is created to use in later tasks and to increase the automation of processes. The library consists of 9 different feature detectors, 3 distinct feature matcher algorithms, and 2 different image representation methods, and is compatible with additional improvements. In terms of heading estimation methods, the library includes 4 methodologies. In addition to the methodologies, the library is enlarged with the topological navigation and mapping framework in which both local feature matching and heading estimation selections are inherited. Since hands-on results are generated with the help of the Webots robotics simulator, a Python object that handles the whole Webots simulation automatically is also added to the library. At the end of the finalized scheme, the library is reinforced with proper getters and setters to specify public properties such as the maximum number of keypoints, which will be generated from a frame, and the heading vector, which is the main output of the topological navigation. The creation of an automatic testing framework, which enables remote testing and continuous improvement of developed algorithms and possible uncertainties with the help of Webots, is left as future work of this thesis.

The local feature matching algorithm selection study showed that spherical distortion affects the repeatable keypoint generation when the interested region is moved

towards the poles of the image in equirectangular representation. The issue is diminished by using a different representation method 'tangent images' that is based on dividing an image into its tangent planes. However, the image division process generated multiple smaller images from the spherical image and local feature matching is performed on all of them separately. Therefore, 'tangent images' representation increased the computational cost significantly compared to equirectangular image representation. Further, it has been detected that the SuperPoint and SuperGlue combination, based on graph neural networks, extracted comparable results from equirectangular images with the 'tangent images' representation. However, both the SuperPoint and SuperGlue are examined with their pretrained weights so that additional improvement could be achieved by training algorithms with spherical images. The heading estimation method selection study clarifies that the keypoint priority assignment improved heading estimation performance in all methodologies. If the keypoint distribution around the target frame is bounded in a region, a performance drop is realized in heading estimation for methods based on keypoint pairing. Thus, method-4, which estimates the heading vector based on keypoint latitude change and assigns priority to keypoints concerning the magnitude of the keypoint latitude change, is marked as the most robust method concerning keypoint distribution and computational effectiveness by shrinking keypoint pairing. However, methods are still open to improvement concerning the keypoint priorities. Since only an exponential function is covered to assign priority to keypoints, an additional analysis could be performed to investigate the performances of different functions.

In the hands-on part of the framework, it has been realized during the mapping and exploration phase that the current exploration heading estimation is stuck in corners since there is no farther region to search due to border blockade. Therefore, a new state is needed to recover from being stranded in corners when there are consecutive returns to the lastly added map node due to node addition failures. Further in the mapping and exploration, because of false positive matches in local feature matching, impractical edge additions are detected on the map. This results in incorrect path planning and false robot movement. Therefore, the SuperGlue confidence score is increased from 0.2 to 0.4 to output more confident matching results. In addition, it has been detected that the robot is visible in each spherical camera frame. To over-

come the problem, a crop degree of 25° is generated to crop 25° of the frame from both upper and lower sides. This way, the visibility of the robot inside the frame is removed and the high spherical distortion at the poles is shrunk. In addition, local feature matching could be reinforced with a geometric verification method such as RANSAC to create further geometrical robustness. From a different point of view, to prevent long-distance edge additions, edge addition is proposed to be made episodically so that similarity of the last added node is only compared with nodes that are inside a certain sliding window. The limited range of node comparison is also another opportunity to bypass unachievable edges in path planning.

In summary, a topological navigation and mapping framework based on spherical images is proposed for usage in an unknown environment. Even though drawbacks are identified during the process, the proposed methodology is tested interactively in the Webots robotics simulator and both topological navigation and mapping tasks are performed effectively. In detail, a robot is able to map an unknown environment and perform navigation to a random point in the environment without any need for metric information. However, the framework is not examined concerning the uncertainties due to dynamic changes such as illumination. To investigate the performance of algorithms in terms of uncertainties, additional analysis can be taken into account while changing the environmental properties of the already constructed Webots worlds. In addition, since the framework is not put into practice in a real-world application, the hands-on part can be enlarged by implementing the framework with a robot equipped with a spherical camera in a sample area.

REFERENCES

[1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, Speeded-up robust features (surf), Computer Vision and Image Understanding, 110(3), pp. 346–359, 2008, ISSN 1077-3142, similarity Matching in Computer Vision and Multimedia.

[2] G. Beutler, M. Rothacher, S. Schaer, T. Springer, J. Kouba, and R. Neilan, The international gps service (igs): An interdisciplinary service in support of earth sciences, Advances in Space Research, 23(4), pp. 631–653, 1999, ISSN 0273-1177, satellite Dynamics, Orbit Analysis and Combination of Space Techniques.

[3] J. Bian, W.-Y. Lin, Y. Matsushita, S.-K. Yeung, T.-D. Nguyen, and M.-M. Cheng, Gms: Grid-based motion statistics for fast, ultra-robust feature correspondence, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2828–2837, 2017.

[4] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*, Springer Publishing Company, Incorporated, 1st edition, 2007, ISBN 3540734287.

[5] J. Cech, J. Matas, and M. Perdoch, Efficient sequential correspondence selection by cosegmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9), pp. 1568–1581, 2010.

[6] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta, Neural topological SLAM for visual navigation, CoRR, abs/2005.12256, 2020.

[7] T.-y. Chuang and N.-H. Perng, Rectified feature matching for spherical panoramic images, Photogrammetric Engineering & Remote Sensing, 84, pp. 25–32, 01 2018.

[8] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, Spherical cnns, CoRR, abs/1801.10130, 2018.

[9] J. Cruz, I. Bogdanova, B. Paquier, M. Bierlaire, and J.-P. Thiran, Scale invariant feature transform on the sphere: Theory and applications, Technical Report TRANSP-OR 090426, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne, 2009, published as Cruz, J., Bogdanova, I., Paquier, B., Bierlaire, M., and Thiran, J. (2012). Scale Invariant Feature Transform on the Sphere: Theory and Applications, International Journal of Computer Vision98(2):217-241.

[10] Y. Şahin, Topological Navigation Python Library, `https://users.metu.edu.tr/kbugra/research/topological_navigation/`, 2022.

[11] 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, 29(6), pp. 1052–1067, 2007.

[12] S. S. Deshmukh and J. J. Knierim, Influence of local objects on hippocampal representations: Landmark vectors and memory, Hippocampus, 23(4), pp. 253–267, feb 2013.

[13] D. DeTone, T. Malisiewicz, and A. Rabinovich, Superpoint: Self-supervised interest point detection and description, 2017, cite arxiv:1712.07629Comment: Camera-ready version for CVPR 2018 Deep Learning for Visual SLAM Workshop (DL4VSLAM2018).

[14] D. DeTone, T. Malisiewicz, and A. Rabinovich, Toward geometric deep SLAM, CoRR, abs/1707.07410, 2017.

[15] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, A solution to the simultaneous localization and map building (slam) problem, IEEE Transactions on Robotics and Automation, 17(3), pp. 229–241, 2001.

[16] M. Eder and J. Frahm, Convolutions on spherical images, CoRR, abs/1905.08409, 2019.

[17] M. Eder, T. Price, T. Vu, A. Bapat, and J. Frahm, Mapped convolutions, CoRR, abs/1906.11096, 2019.

[18] M. Eder, M. Shvets, J. Lim, and J. Frahm, Tangent images for mitigating spherical distortion, CoRR, abs/1912.09390, 2019.

[19] A. Elfes, Sonar based real-world mapping navigation, Robotics and Automation, IEEE Journal of, RA-3, pp. 249 – 265, 07 1987.

[20] A. Elfes, Using occupancy grids for mobile robot perception and navigation, Computer, 22(6), pp. 46–57, 1989.

[21] J. Engel, T. Schöps, and D. Cremers, Lsd-slam: Large-scale direct monocular slam, in D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pp. 834–849, Springer International Publishing, Cham, 2014.

[22] M. Fischler and R. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM, 24(6), pp. 381–395, 1981.

[23] P. Foo, W. Warren, A. Duchon, and M. Tarr, Do humans integrate routes into a cognitive map? map- versus landmark-based navigation of novel shortcuts., Journal of experimental psychology. Learning, memory, and cognition, 31, pp. 195–215, 04 2005.

[24] S. Gillner and H. A. Mallot, Navigation and acquisition of spatial knowledge in a virtual maze, Journal of Cognitive Neuroscience, 10(4), pp. 445–463, 1998.

[25] T. Goedemé, M. Nuttin, T. Tuytelaars, and L. Van Gool, Omnidirectional vision based topological navigation, International Journal of Computer Vision, 74, pp. 219–236, 07 2007.

[26] R. Gomez-Ojeda, D. Zuñiga-Noël, F.-A. Moreno, D. Scaramuzza, and J. Gonzalez-Jimenez, PL-SLAM: a Stereo SLAM System through the Combination of Points and Line Segments, 2018.

[27] H. Guan and W. A. P. Smith, Brisks: Binary features for spherical images on a geodesic grid, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4886–4894, 2017.

[28] H. Hadj-Abdelkader, E. Malis, and P. Rives, Spherical Image Processing for Accurate Visual Odometry with Omnidirectional Cameras, in *The 8th Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras*

- *OMNIVIS*, Rahul Swaminathan and Vincenzo Caglioti and Antonis Argyros, Marseille, France, October 2008.

[29] C. Harris and M. Stephens, A combined corner and edge detector, in *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

[30] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2 edition, 2004.

[31] 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*, volume 1, pp. 345–351 vol.1, 1997.

[32] 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, 2016.

[33] Y. Hu, B. Subagdja, A.-H. Tan, and Q. Yin, Vision-based topological mapping and navigation with self-organizing neural networks, IEEE Transactions on Neural Networks and Learning Systems, pp. 1–13, 2021.

[34] K. Kawamura, A. Koku, D. Wilkes, R. Ii, and A. Sekmen, Toward egocentric navigation, International Journal of Robotics and Automation, 17, 01 2002.

[35] D. Kortenkamp and T. Weymouth, Topological mapping for mobile robots using a combination of sonar and vision sensing, in *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*, AAAI'94, p. 979–984, American Association for Artificial Intelligence, USA, 1994, ISBN 0262611023.

[36] B. Kuipers and Y.-T. Byun, A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations, Robotics and Autonomous Systems, 8(1), pp. 47–63, 1991, ISSN 0921-8890, special Issue Toward Learning Robots.

[37] J. J. Leonard and H. F. Durrant-Whyte, Simultaneous map building and localization for an autonomous mobile robot, Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91, pp. 1442–1447 vol.3, 1991.

[38] S. Leutenegger, M. Chli, and R. Siegwart, Brisk: Binary robust invariant scalable keypoints, in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2548–2555, 11 2011.

[39] 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, 2014.

[40] X. Li, K. Han, S. Li, and V. A. Prisacariu, Dual-resolution correspondence networks, CoRR, abs/2006.08844, 2020.

[41] G. LoweDavid, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision, 2004.

[42] J. Ma and J. Zhao, Robust topological navigation via convolutional neural network feature and sharpness measure, IEEE Access, 5, pp. 20707–20715, 2017.

[43] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, A constant-time efficient stereo slam system, in *Proc. BMVC*, pp. 54.1–54.11, 2009, ISBN 1-901725-39-1, doi:10.5244/C.23.54.

[44] M. Meng and A. 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 vol.2, 1993.

[45] O. Michel, Webots: Professional mobile robot simulation, Journal of Advanced Robotics Systems, 1(1), pp. 39–42, 2004.

[46] B. Micusík and J. Kosecka, Piecewise planar city 3d modeling from street view panoramic sequences, 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2906–2912, 2009.

[47] K. Mikolajczyk and C. Schmid, A performance evaluation of local descriptors, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(10), pp. 1615–1630, 2005.

[48] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas, Working hard to know your neighbor's margins: Local descriptor learning loss, CoRR, abs/1705.10872, 2017.

[49] H. P. Moravec, Sensor fusion in certainty grids for mobile robots, AI Magazine, 9(2), p. 61, Jun. 1988.

[50] D. W. Murray and G. Klein, Simulating low-cost cameras for augmented reality compositing, IEEE Transactions on Visualization & Computer Graphics, 16(03), pp. 369–380, may 2010, ISSN 1941-0506.

[51] J. Pan, J. Pack, A. Kosaka, and A. Kak, Fuzzynav: A vision based robot navigation architecture using fuzzy inference for uncertainty reasoning, in *[1995] Proceedings of the World Congress on Neural Networks*, pp. 602–607 vol.2, 1995.

[52] F. Perez Fontan, B. Sanmartín, A. Steingaß, A. Lehner, J. Selva, E. Kubista, and B. Arbesser-Rastburg, Measurements and modeling of the satellite-to-indoor channel for galileo, in *In Proceedings of the National Technical Meeting, Institute of Navigation Vol. 2004*, pp. 190–202, 01 2004.

[53] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, Pl-slam: Real-time monocular visual slam with points and lines, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4503–4508, 2017.

[54] R. Raguram, J.-M. Frahm, and M. Pollefeys, A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus, in D. Forsyth, P. Torr, and A. Zisserman, editors, *Computer Vision – ECCV 2008*, pp. 500–513, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, ISBN 978-3-540-88688-4.

[55] A. Ranganathan, E. Menegatti, and F. Dellaert, Bayesian inference in the space of topological maps, IEEE Transactions on Robotics, 22(1), pp. 92–107, 2006.

[56] I. Rocco, R. Arandjelovic, and J. Sivic, Efficient neighbourhood consensus networks via submanifold sparse convolutions, CoRR, abs/2004.10566, 2020.

[57] I. Rocco, M. Cimpoi, R. Arandjelovic, A. Torii, T. Pajdla, and J. Sivic, Neighbourhood consensus networks, CoRR, abs/1810.10510, 2018.

[58] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, Orb: an efficient alternative to sift or surf, in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 11 2011.

[59] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, SuperGlue: Learning feature matching with graph neural networks, in *CVPR*, 2020.

[60] D. Schleicher, L. Bergasa, M. Ocaña, R. Barea, and M. Guillén, Real-time hierarchical outdoor slam based on stereovision and gps fusion, IEEE Transactions on Intelligent Transportation Systems, 10, pp. 440–452, 09 2009.

[61] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer, Discriminative learning of deep convolutional feature point descriptors, in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 118–126, 2015.

[62] Y. Su and K. Grauman, Flat2sphere: Learning spherical convolution for fast features from 360° imagery, CoRR, abs/1708.00919, 2017.

[63] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou, Loftr: Detector-free local feature matching with transformers, CoRR, abs/2104.00680, 2021.

[64] I. Suárez, G. Sfeir, J. M. Buenaposada, and L. Baumela, Beblid: Boosted efficient binary local image descriptor, Pattern Recognition Letters, 2020, ISSN 0167-8655.

[65] 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, 7, pp. 84–88, 07 2015.

[66] S. Thrun, Learning metric-topological maps for indoor mobile robot navigation, Artificial Intelligence, 99(1), pp. 21–71, 1998, ISSN 0004-3702.

[67] Y. Tian, B. Fan, and F. Wu, L2-net: Deep learning of discriminative patch descriptor in euclidean space, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6128–6136, 2017.

[68] Y. Tian, X. Yu, B. Fan, F. Wu, H. Heijnen, and V. Balntas, Sosnet: Second order similarity regularization for local descriptor learning, CoRR, abs/1904.05019, 2019.

[69] C. University, *The Cambridge Dictionary*, Cambridge University Press, 1999.

[70] S. Urban and S. Hinz, MultiCol-SLAM - A Modular Real-Time Multi-Camera SLAM System, 2016.

[71] H. A. H. C. van Veen, H. K. Distler, S. J. Braun, and H. H. Bülthoff, Navigating through a virtual city: Using virtual reality technology to study human action and perception, Future Gener. Comput. Syst., 14, pp. 231–242, 1998.

[72] D. P. Vassileios Balntas, Edgar Riba and K. Mikolajczyk, Learning local feature descriptors with triplets and shallow convolutional neural networks, in E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 119.1–119.11, BMVA Press, September 2016, ISBN 1-901725-59-6.

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need, CoRR, abs/1706.03762, 2017.

[74] R. F. Wang and E. S. Spelke, Human spatial representation: insights from animals, Trends in Cognitive Sciences, 6(9), pp. 376–382, 2002, ISSN 1364-6613.

[75] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua, LIFT: learned invariant feature transform, CoRR, abs/1603.09114, 2016.

[76] Y. Zhang and F. Huang, Panoramic visual slam technology for spherical images, Sensors, 21(3), 2021, ISSN 1424-8220.

[77] Q. Zhao, W. Feng, L. Wan, and J. Zhang, Sphorb: A fast and robust binary feature on the sphere, International Journal of Computer Vision, 113(2), pp. 143–159, 2015.

[78] Y. Şahin, *Development of an Indoor Reactive Navigation Method Using Spherical Images*, Master's thesis, Middle East Technical University, 2021.

APPENDIX A

SAMPLE OBJECTS OF THE TOPOLOGICAL NAVIGATION PYTHON LIBRARY

The topological navigation python library can be found in [10].

```
class SelectKeypointDescriptor:
    def __init__(
            self,
            img_info,
            kp_descriptor="ORB",
            draw_keypoints=False,
            representation="Cartesian",
            crop_degree=0,
            max_keypoints=1024,
            device="cpu",
            matcher_selection="BruteForce",
            resize=[-1],
            kornia_apply_orinet=True,
            kornia_apply_affnet=True,
            ignore_border_kp=False,
    ):
        """
        This function initialize parameters used on the class
        Parameters
        ----------
        img_info : string
            Path to image.
        kp_descriptor : string
            Keypoint descriptor name.
        draw_keypoints : bool, optional
            keypoints drawing flag. The default is False.
        representation : string, optional
            image representation. The default is Cartesian
        crop_degree : int, optional
            image height padding degree of spherical image
        max_keypoints : int, optional
            Maximum number of generated keypoints.
            The default is 1024.
        device : str, optional
```

```
            Device currently in use. The default is "cpu".
        matcher_selection : str, optional
            selected matcher to load image
            with appropriate settings.
            The default is "BruteForce".
        resize : list, optional
            image resizing dimensions, -1 if unwanted.
            The default is [-1].
        kornia_apply_orinet : bool, optional
            OriNet application flag on kornia descriptors
            The default is True.
        kornia_apply_affnet : bool, optional
            AffNet application flag on kornia descriptors.
            The default is True.
        ignore_border_kp : bool, optional
            Ignore keypoints detected on image border on
            distorted image matching.
            The default is False.

        Returns
        -------
        None.

        """

    def try_descriptor(self):
        """
        This function calls the related descriptor
        from descriptor map
        Returns
        -------
        keypoint details object
            Properties of keypoint descriptor.
        """

    def update_kp_changes(self):
        """
        This function updates current descriptor parameter changes
        before running detection


        """

    @property
    def img_info(self):
        """
        Image information getter currently used in the object
        Returns
        -------
        Previously set image information
            can be image path, OpenCV image, Pytorch image
        """

    @img_info.setter
```

```
89      def img_info(self, new_img_info):
90          """
91          Image information setter to use in the object
92          Returns
93          -------
94
95          """
96
97      @property
98      def kp_descriptor(self):
99          """
100         Keypoint descriptor getter currently used in the object
101         Returns
102         -------
103         Previously set Keypoint descriptor
104             can be str
105         """
106
107     @kp_descriptor.setter
108     def kp_descriptor(self, new_descriptor):
109         """
110         Keypoint descriptor setter to use in the object
111         Returns
112         -------
113
114         """
115
116     @property
117     def representation(self):
118         """
119         Image representation getter currently used in the object
120         Returns
121         -------
122         Previously set Image representation
123             can be str
124         """
125
126     @representation.setter
127     def representation(self, new_representation):
128         """
129         Image representation setter to use in the object
130         Returns
131         -------
132
133         """
134
135     def compute_cv2_keypoints(self, img):
136         """
137         This function conduct keypoint detection & descriptor
138         computation process using built-in OpenCV functions
139         as both detector & descriptor
140
141         Parameters
142         ----------
```

```
143             img : array of uint8
144                 input image.
145             Returns
146             -------
147             img_details: Tuple
148                 Contains image keypoints [Mx4] and
149                 SIFT descriptors [MX128] Torch tensors
150                 BEBLID descriptors [Mx64] Torch tensors
151                 ORB descriptors [Mx32] Torch tensors
152
153             """
154
155         def compute_kornia_keypoints(self, img):
156             """
157             This function conduct keypoint detection & descriptor
158             computation process using built-in kornia functions
159             as both detector & descriptor
160
161             Parameters
162             ----------
163             img : array of uint8
164                 input image.
165             Returns
166             -------
167             img_details: Tuple
168                 Contains image keypoints [Mx4] and
169                 descriptors [MX128] Torch tensors
170
171             """
172
173         def compute_superpoint_keypoints(self, img):
174             """
175             This function conduct keypoint detection & descriptor
176             computation process using SuperPoint as
177             both detector & descriptor
178
179             Parameters
180             ----------
181             img : array of uint8
182                 input image.
183             Returns
184             -------
185             img_details: Tuple
186                 Contains image keypoints [Mx4] and
187                 SuperPoint descriptors [MX256] Torch tensors
188
189             """
190
191         def detect_equirectangular(self):
192             """
193             This function extracts only the visible keypoints
194             from equirectangular spherical image.
195
196             Parameters
```

```
        ----------

        Returns
        -------
        kp : torch tensor, visible keypoint coordinates
        desc : torch tensor, visible decriptors

        """

    def detect_tangent_images(self):
        """
        This function extracts only the visible keypoints
        from a collection of tangent images and transfers them
        to coordinates on the equirectangular image.
        That is, only returns the keypoints visible
        to a spherical camera at the center of the icosahedron.

        Parameters
        ----------

        Returns
        -------
        visible_kp : torch tensor, visible keypoint coordinates
        visible_desc : torch tensor, visible decriptors

        """


class TopologicalNavigation(SelectKeypointDescriptor):

    def __init__(self, device, crop_degree, max_keypoints,
                 min_matches, resize, exploration_noise=50,
                 mapping_stage=False, continue_mapping=False):
        # initializing inherited keypoint descriptor object
        SelectKeypointDescriptor.__init__(
            self,
            img_info="",
            kp_descriptor="SuperPoint",
            representation="Equirectangular",
            crop_degree=crop_degree,
            max_keypoints=max_keypoints,
            device=device,
            matcher_selection="SuperGlue",
            resize=resize,
        )

    @property
    def img_info(self):
        """
        Image information getter currently used in the object
        Returns
        -------
        Previously set image information
            can be image path, OpenCV image, Pytorch image
```

```
251             """
252
253         @img_info.setter
254         def img_info(self, new_img_info):
255             """
256             Image information setter to use in the object
257             Returns
258             -------
259
260             """
261
262         @property
263         def heading(self):
264             """
265             Heading angle getter currently used in the object
266             Returns
267             -------
268             Heading angle (deg)
269                 float
270             """
271
272         @property
273         def heading_vector(self):
274             """
275             Heading vector getter currently used in the object
276             Returns
277             -------
278             Heading unit vector
279                 numpy.ndarray
280             """
281
282         @property
283         def current_frame(self):
284             """
285             Current frame getter currently used in the object
286             Returns
287             -------
288             Previously set current frame information
289                 can be image path, OpenCV image, Pytorch image
290             """
291
292         @current_frame.setter
293         def current_frame(self, new_frame):
294             """
295             Current frame setter to use in the object
296             Returns
297             -------
298
299             """
300
301         @property
302         def target_frame(self):
303             """
304             Target frame getter currently used in the object
```

```
305             Returns
306             -------
307             Previously set target frame information
308                 can be image path, OpenCV image, Pytorch image
309             """
310
311         @target_frame.setter
312         def target_frame(self, new_frame):
313             """
314             Target frame setter to use in the object
315             Returns
316             -------
317
318             """
319
320         @property
321         def topo_map(self):
322             """
323             Topological map getter currently used in the object
324             Returns
325             -------
326             Currently used topological map
327                 Pickle object
328             """
329
330         @topo_map.setter
331         def topo_map(self, new_topo_map):
332             """
333             Topological map setter to use in the object
334             Returns
335             -------
336
337             """
338         def load_map(self, map_name=None):
339             """
340             This function loads constructed map from memory
341
342             Parameters
343             ----------
344             map_name: str
345                 Specified map name to load from the directory
346
347             Returns
348             -------
349
350             """
351         def save_map(self, map_name=None):
352             """
353             This function save constructed map to memory
354
355             Parameters
356             ----------
357             map_name: str
358                 Specified map name to save to the directory
```

```
359
360             Returns
361             -------
362
363             """
364         def draw_map(self, block=True):
365             """
366             This function draws the map as topo graph
367             with nodes and edges
368
369             Parameters
370             ----------
371             block: boolean
372                 Parameter specifying if plot stops
373                 code execution or not
374
375             Returns
376             -------
377
378             """
379         def draw_path(self):
380             """
381             This function draws the created path
382             upon the topological map
383
384             Returns
385             -------
386
387             """
388         def update_current_frame_properties(self):
389             """
390             This function is used to update
391             current frame properties when current_frame is set!
392
393             Returns
394             -------
395
396             """
397         def update_target_frame_properties(self):
398             """
399             This function is used to update
400             target frame properties when target_frame is set!
401
402             Returns
403             -------
404
405             """
406         def compare_current_frame_with_map(self):
407             """
408             This function compares current frame with map
409             and take necessary action according to robot stage
410
411             Returns
412             -------
```

```
413
414             """
415         def follow_robot_path(self):
416             """
417             This function follows the constructed robot path
418             according to target frame
419
420             Returns
421             -------
422
423             """
424         def check_obstacles(self):
425             """
426             This function checks obstacles when
427             distance_sensor_values are set!
428
429             Returns
430             -------
431             obstacles_found: boolean
432                 Parameter specifying if there is obstacle around
433
434             unit_obstacle_angle_vector: numpy.ndarray, (2,)
435                 Obstacle angle vector calculated
436                 with respect to distance sensor values
437
438             """
439         def perform_navigation(self, mapping_active=False,
440                                matched1_sp_closest=None,
441                                matched2_sp_closest=None,
442                                matched1_sp_second_closest=None,
443                                matched2_sp_second_closest=None,
444                                similarity_first=None,
445                                similarity_second=None):
446             """
447             This function performs navigation according to the
448             current stage of the robot &
449             local feature matching results
450
451             Parameters
452             ----------
453             mapping_active: boolean
454                 Parameter specifying if mapping stage is active
455             matched1_sp_closest: numpy.ndarray (Nx2)
456                 Local feature matching result of current frame
457                 to the most similar node
458             matched2_sp_closest: numpy.ndarray (Nx2)
459                 Local feature matching result
460                 of the most similar map node to current frame
461             matched1_sp_second_closest: numpy.ndarray (Nx2)
462                 Local feature matching result
463                 of current frame to the most similar second map node
464             matched2_sp_second_closest: numpy.ndarray (Nx2)
465                 Local feature matching result
466                 of the most similar second map node to current frame
```

```
            similarity_first: float
                Similarity score
                of the most similar map node to current frame
            similarity_second: float
                Similarity score
                of the most similar second map node to current frame

            Returns
            -------

            """
        def get_heading(self, matched1_sp, matched2_sp):
            """
            This function computes the heading vector
            to follow with respect to local feature matching results.
            Parameters
            ----------
            matched1_sp: numpy.ndarray (Nx2)
                Local feature matching result of the first frame
            matched2_sp: numpy.ndarray (Nx2)
                Local feature matching result of the second frame
            Returns
            -------
            heading_method5: float
                Calculated heading
            heading_xnorm_method5: float
                x-component of heading unit vector
            heading_ynorm_method5: float
                y-component of heading unit vector
            """
        @staticmethod
        def get_unit_heading_wrt_latitude(u_c, v_c, v_t, v_h):
            """
            This function computes unit heading vector
            for each keypoint by taking elevation into account.

            Parameters
            ----------
            u_c : numpy.ndarray (Nx2)
                Longitude unit vector array of keypoint
                for the current frame.
            v_c : numpy.ndarray
                Latitude unit vector array of keypoint
                for the current frame.
            v_t : numpy.ndarray (Nx2)
                Latitude unit vector array of keypoint
                for the target frame.
            v_h : numpy.ndarray
                Horizontal unit vector array.

            Returns
            -------
            u: numpy.ndarray (Nx2)
                Unit heading vector that is computed
```

```
            for each input keypoint pair.

        """
    @staticmethod
    def get_unit_heading(u_c, u_t):
        """
        This function computes unit heading vector
        for each keypoint pair.

        Parameters
        ----------
        u_c : list [u_ci, u_cj] [(N*(N-1)/2)x2, (N*(N-1)/2)x2]
            Unit vector list of keypoint pairs
            for the current frame.
        u_t : list [u_ti, u_tj] [(N*(N-1)/2)x2, (N*(N-1)/2)x2]
            Unit vector list of keypoint pairs
            for the target frame.

        Returns
        -------
        unit_heading: numpy.ndarray (Nx2)
            Unit heading vector that is computed
            for each input keypoint pair.

        """
    @staticmethod
    def get_keypoint_unitvector(keypoints_ang):
        """
        Generate unit vectors of keypoints.
        Rectangular decomposition of the latitude
        of the keypoint on a unit circle
        after discarding latitude results in its unit vector.
        Parameters
        ----------
        Returns
        -------
        numpy.ndarray
            Unit vectors that are drawn to each of the keypoints.
        """
    def compute_similarity(self, node1_details, node2_details):
        """
        This function computes similarity between node1 and node2
        by applying local feature matching.

        Parameters
        ----------
        node1_details: torch.Tensor (Mx4)
            Pytorch tensor of node1 descriptor details
        node2_details: torch.Tensor (Mx4)
            Pytorch tensor of node2 descriptor details

        Returns
        -------
        similarity: float
```

```
            Similarity score of the two nodes
        matched1_sp: numpy.ndarray (Nx2)
            Local feature matching result of the first node
        matched2_sp: numpy.ndarray (Nx2)
            Local feature matching result of the second node

        """
    def find_nearest_map_node(self, current_frame_name,
                              current_frame_number,
                              current_frame_details):
        """
        This function finds the nearest map node
        to the current frame with respect to similarity
        by searching through the map

        Parameters
        ----------
        current_frame_name: str
            Name of node
        current_frame_number: float
            Number of node
        current_frame_details: torch.Tensor (Mx4)
            Pytorch tensor of node descriptor details

        Returns
        -------
        closest_id: int
            The closest map node id to current frame
        similarity: float
            Similarity score of the closest node and
            the current frame
        similarity_previous: float
            Similarity score of the second-closest node and
            the current frame
        matched1_sp_closest: numpy.ndarray (Nx2)
            Local feature matching result
            of the current frame with the closest map node
        matched2_sp_closest: numpy.ndarray (Nx2)
            Local feature matching result
            of the closest map node with the current frame
        matched1_sp_second_closest: numpy.ndarray (Nx2)
            Local feature matching result
            of the current frame with the second-closest map node
        matched2_sp_second_closest: numpy.ndarray (Nx2)
            Local feature matching result
            of the second-closest map node with the current frame
        """
```

APPENDIX B

SAMPLE APPLICATION OF THE TOPOLOGICAL NAVIGATION PYTHON LIBRARY

```
from topological_navigation import TopologicalNavigation

# initializing TopologicalNavigation object
tp = TopologicalNavigation(device='cuda', crop_degree=25,
                           max_keypoints=150, min_matches=40,
                           resize=[400, 800], mapping_stage=False,
                           continue_mapping=False)
# loading specified map
tp.load_map(map_name=<name of the topological map .pickle>)
# drawing the map without any path
tp.draw_map()
# updating target frame
tp.target_frame = <path to target frame>
# updating current frame
tp.current_frame = <path to current frame>
# comparing current frame with map to perform navigation
tp.compare_current_frame_with_map()
# drawing path over the map for once
tp.draw_path()
# get estimated heading angle
heading_deg = tp.heading
```