University of California
Merced

Appearance-Based Navigation, Localization, Mapping, and Map Merging for Heterogeneous Teams of Robots

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy

in

Electrical Engineering and Computer Science

by

Görkem Erinç

2013

Abstract of the Dissertation

Appearance-Based Navigation, Localization, Mapping, and Map Merging for Heterogeneous Teams of Robots

by

Görkem Erinç

Spatial awareness is a vital component for most autonomous robots operating in unstructured environments. Appearance-based maps are emerging as an important class of spatial representations for robots. Requiring only a camera instead of an expensive sensor like a laser range finder, appearance-based maps provide a suitable world model to human perception and offer a natural way to exchange information between robots and humans. In this dissertation, we embrace this representation and present a framework that provides navigation, localization, mapping, and map merging capabilities to heterogeneous multi-robot systems using exclusively monocular vision. Our first contribution is integrating different ideas from separately proposed solutions into a robust appearance-based localization and mapping framework that does not suffer from the individual issues tied to the original proposed methods. Next, we introduce a novel visual navigation algorithm that steers a robot between two images through the shortest possible path. Thanks to its invariance to changes in the tilt angle and the elevation of the cameras, the images collected by another robot with a totally different morphology and camera placement can be used for navigation. Furthermore, we tackle the problem of merging together two or more appearance-based maps independently built by robots operating in the same environment, and propose an anytime algorithm aiming to quickly identify the more advantageous parts to merge. Noting the lack of any evaluation criteria for appearance-based maps, we introduce our task specific quality metric that measures the utility of a map with respect to three major robotic tasks: localization, mapping, and navigation. Additionally, in order to measure the quality of merged appearance-based maps, and the performance of the merging algorithm, we propose the use of algebraic connectivity, a concept which we borrowed from graph theory. Finally, we introduce a machine learning based WiFi localization technique which we later embrace as the core of our novel heterogeneous map merging algorithm. All algorithms introduced in this dissertation are validated on real robots.

The dissertation of Görkem Erinç is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

Ming-Hsuan Yang

David C. Noelle

Stefano Carpin, Committee Chair

University of California, Merced

2013

Dedicated to my father...

Table of Contents

List of Figures

List of Tables

Acknowledgments

After five years in Merced working towards a degree that I once set as a goal, it has come down to writing this one last section of my dissertation. Each sentence I add to this section not only takes me another step closer to finishing this document but also pushes me to this brand new life they claim to exist after grad school. Possibly it is the fear of this unknown that makes me hesitant to close this chapter. However, I know that the people who made this real by unconditionally guiding and supporting me throughout this rough journey of mine will be there to help me in the next chapter of my life.

Among these people, I would especially like to thank my advisor Stefano Carpin for his academic guidance but more importantly for his friendship. Working with him for a total of nine years at various levels has not only taught me many aspects of robotics but also exemplified how to be an excellent researcher. I would also like to thank him for his personal support when I needed the most and for not charging me for the long hours of our unofficial therapy sessions. Evidently, I would not be here today with my PhD (and MSc) without his support. I would also like to express my gratitude towards my committee members David C. Noelle and Ming-Hsuan Yang for their valuable comments and suggestions.

Additionally, I want to thank the UC Merced Graduate and Research Council and the Center for Information Technology Research In the Interest of Society (CITRIS) for financially supporting my research.

I would like to thank my friends and colleagues in the Robotics Lab at UC Merced including Andreas Kolling, Nicola Basilico, Derek Burch, Seyedshams "Shams" Feyzabadi, and of course Benjamin Balaguer with whom I spent long nights in the empty corridors of our engineering building convincing robots to work autonomously. Thank you, Ben, for all your constructive comments and our enlightening discussions. Other friends at UC including Ankur Kamthe, Gayatri Premshekharan, David Huang, Carlo Camporesi, Paola Di Giuseppantonio Di Franco, Fabrizio Galeazzi, Alicia Ramos Jordan, Marco Valesi, and Nicole and Joshua Madfis have helped keep me sane with their friendship and company along the way.

My house mates, Oktar Özgen and Mentar Mahmudi, helped me to study advanced probability during our traditional poker nights. We clearly shared more than a house and hopefully have built something permanent. Thanks for being the two constants in my life.

Most importantly, I would like to thank to my family. To my mother Meral, I thank you for your endless support and unconditional love reaching to a level that no one can ever comprehend. My little sister Gökce, thank you for reminding me that you will always be there whenever I am in need and making me laugh when I didn't even want to smile. To my miracle son Ada, you brought joy to all of us and taught us again how to be happy. Follow your dreams and be a free man, free from all sorrows. And the person who stood by me at every step along the way, encouraged me to proceed when everything fell apart, patiently but constantly supported me just to see me succeed, and shared my fears, hopes and dreams, my wife Gökce. With all my love... And my dad, the reason I am who I am today, the motivation behind my work, my derive to be a good man, the man that I hope one day my son will see me as. I miss you every day...

Vita

2000–2004	B.Sc., Mechatronics, Sabanci University, Turkey.
Summer 2003	Research Intern, Electronic R&D Department, Arcelik, Turkey.
Spring 2004	Teaching Assistant, Sabanci University, Turkey.
2004–2006	M.Sc., Computer Science, Jacobs University, Germany.
Spring 2007	Research Intern, Robotics Institute, Carnegie Mellon University, USA.
2007–2013	Graduate Student Researcher & Teaching Assistant, University of California Merced, USA.

Publications

G. Erinc, S. Carpin. Anytime merging of appearance based maps. *Autonomous Robots* (Accepted)

G. Erinc, B. Balaguer, S. Carpin. Heterogeneous Map Merging Using WiFi Signals. *The IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013 (Accepted)

G. Pillonetto, G. Erinc, S. Carpin. Online estimation of covariance parameters using extended Kalman filtering and application to robot localization. *Advanced Robotics*, 26(18), pages 2169-2188, 2012

B. Balaguer, G. Erinc, S. Carpin. Combining classification and regression for WiFi localization of heterogeneous robot teams in unknown environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3496-3503, 2012

G. Erinc, S. Carpin. Anytime merging of appearance based maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1656-1662, 2012

G. Erinc, S. Carpin. Evaluation criteria for appearance-based maps. In *Proceedings of Performance Metrics for Intelligent Systems* , 2010

G. Erinc and S. Carpin. Image-based mapping and navigation with heterogeneous robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5807-5814, 2009.

G. Erinc, G. Pillonetto, S. Carpin. Online estimation of variance parameters: experimental results with application to localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1890-1895, 2008

G. Erinc, S. Carpin. A genetic algorithm for nonholonomic motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1843-1849, 2007

CHAPTER 1

Introduction

Robotics evolving from an industrial driven technology into a science that creates solutions to ease every aspect of daily life has started shaping the future. From robotic museum guides, mine sweeping military robots, Mars explorers to urban search and rescue robots, a variety of robots have already been introduced into different areas. Within 30 years of time their physical capabilities have made a tremendous jump. In 1986 when Honda introduced *E0*, the first of their famous E-series humanoid robots, it was an outstanding achievement that finally a two-legged robot was able to walk in a straight line even if its motion was discrete and boringly slow. Today, the technology reached to a point that robots like *Asimo* or *BigDog* can perform many complex tasks with their fast, dynamic, and stable motions. Some may even argue that robots like *Petman* resemble a human more than a robot. While it is a pure speculation whether these robots would pass a hypothetical *Turing test* designed for physical appearance and behavior, it is evident that robotic platforms are getting better in every aspect of their performance. Not only their physical but also their computational capabilities have improved thanks to the advances both in computer science research and sensor technologies. Years after *Deep Blue*'s legendary win against the reigning World Chess Champion, Gary Kasparov, today scientists channel their energy to solve much more challenging problems than chess. *Watson*, for instance, was designed by IBM as a question answering machine with interconnected complex modules each focusing on subjects like information retrieval, natural language processing, and machine learning. Watson's capabilities were tested in *Jeopardy!* where it competed against two of the most successful contestants on the show and Watson left the stage with a clear victory.

These extraordinary pieces of technology are the indications, if not proofs, that we are coming to a future in which *Rosie* from *The Jetsons* will not be a fictional character anymore. However, despite all the efforts and advancements, the dream of robots becoming a part of our daily lives as shared by Bill Gates in his article *A Robot in Every Home* [54] does not seem feasible at least for the near future. One of the main obstacles to realize this envisioned future is financial. It should come as no surprise that sophisticated robots equipped with high-tech actuators and sensors like

the ones mentioned above are very expensive for an average consumer. Indeed, that is one of the main reasons why we only see this level of advanced technology either in research labs with substantial funding in place or as a part of a team of a defense contractor. In order for the general public to welcome coexistence with robots, first of all, robots need to have an affordable retail price tag. Therefore, inexpensive production becomes a necessity which entails inexpensive sensors, low computational power and inexpensive hardware. Today, due to these limitations consumer robotics follows a temporary trend that favors simple and specialized robots rather than highly sophisticated robots capable of solving many different tasks. With the success story of iRobot *Create*, it became clear that robots with simple functionality executed robustly have a higher chance of successfully dominating today's market.

Whether it is a robot designed to accomplish a specific task or a complex one integrating multi-functionality into one body, they need to be designed to be able to coexist with humans in human-made environments. More importantly, they should be able to interact with people and the environment. Perception and autonomy are the key concepts for this interaction and it is therefore very important that they perceive the world similarly and possess some level of autonomy.

Perception opens the doors to reasoning, acting and intelligence by defining the ways to collect information from the surroundings. Cameras providing the closest experience to human perception are the most suitable sensors that robots can use to extract similar information from the environment. With the increasing interest in vision-based robotics several studies showed that the robots can use the perception realized by stereo cameras [152, 115], omnidirectional cameras [16, 164] or monocular cameras [145, 160, 120, 81] for navigation, mapping, place recognition, object recognition and manipulation. Among those, monocular cameras fits our vision of affordable robots due to their simplicity and cheap price.

Domestic environments possessing highly complex and dynamic natures propose several challenges for the robots to operate autonomously. Full autonomy requires the robot to be able to explore an unknown environment, build a map of it in order to use the map for future planning tasks and localize itself in this partially explored map. Studies for this very well defined problem, Simultaneous Localization and Mapping (SLAM), focused on addressing challenging issues such as data association, loop closing, and noisy sensors as well as finding a better representation of the world, a better map. The significant portion of the research efforts have been devoted to represent the world by employing a metric model of the robot's workspace. In these approaches the robot creates an occupancy grid model of the world with metric information where each cell of the grid is labeled with the probability of being

occupied by an object. This model is mainly concerned about metric coordinates of itself and other objects in the environment. Laser range scanners are one of the most commonly used sensors in these SLAM solutions due to their high precision. Nevertheless, it suffers from its high price and does not fit into our vision of vast production of robots. On the other hand, one could argue that it is possible to perform most missions without the precise knowledge of the robot's position and orientation. This is, indeed, the case for many living beings when they navigate in complex environments. This point of view is captured by *topological maps*. The simplified representation of the world based on some topology is closer how humans interpret their environment, and therefore, it is easier to integrate common concepts into this world representation. It is also generally assumed that humans adopt a partially hierarchical representation of spatial organizations [122, 72]. Applying the same idea into robot localization and mapping problems brings several advantages. With additional layers of abstractions the resulting hierarchical topological world representation captures both necessary physical information for the robot to navigate and localize and also provides a cognitive framework enriched with semantic information which can be used for human-robot communication.

Appearance-based maps which have recently attracted attention of the robotics community due to their suitability for topological maps constitute the first steps towards this direction. An appearance-based map can basically, yet incompletely, be defined as an image network. These representations provide a more suitable world model to human perception and offer a natural way to exchange information between robots and humans due to their entanglement with imagery. Moreover, the only sensor required to create an appearance-based map is a camera. With current webcam prices dropping below $20 mark, this model aligns to our inexpensive sensor vision. These vision based solutions work on image space and do not necessarily need metric localization for mapping or navigation purposes. In other words, localization and mapping is realized in *appearance space*. Focusing on camera-only systems, odometry data is not used for localization and mapping purposes. This holds special importance for systems where there is no odometry information available or odometry is very difficult to estimate as in human motion. In situations where humans are involved such as first responders in urban search and rescue or children in child-care, SLAM can be addressed by appearance-based methods since they only focus on the sensor model and observations and not on the motion model.

It is evident that a complex task manageable by a single robot could be even more effectively executed by a collaborative multi-robot system. Robot teams are more robust to failures and can solve a task in less time. Moreover, there is an obvious need for collecting spatially distributed information acquired by teams of robots physically

operating in a shared environment. With respect to the SLAM problem, the spatial awareness can be greatly enhanced by fusing multiple maps independently created by each robot into a single spatial model. This fusion of data produced by different sources not only is more informative but also prevents redundancy in the team, e.g., mapping a part of the environment that is already mapped by another robot in the team. After multiple maps have been combined together and the result has been shared, each of the robots that contributed one of the partial maps is then equipped with a more comprehensive spatial model. The merged map can then enable robots to individually perform tasks they would otherwise not be able to accomplish, like for example safely navigating to a part of the environment they have not explored.

Information fusion with respect to map merging has been studied and solutions are proposed for topological maps [78, 140] and occupancy grid maps [12, 20, 135]. However, merging appearance-based maps is an outstanding problem and deserves attention. Furthermore, additional contributions should be done in order to make it possible to merge appearance-based maps from heterogeneous multi-robot systems because within the current appearance-based navigation frameworks in the literature it is not possible to integrate visual information collected by different cameras attached to different robots into one map and still keep the navigation feasible.

The value of merging independently generated maps together becomes especially apparent after devastating disasters like the earthquake in Japan in 2011 which took nearly 16000 lives and caused the meltdown of 3 nuclear reactors. Despite Japan's long-lasting technological leadership in the field of robotics, they did not have a robot capable of measuring the extent of the radiation leak from the nuclear plant. Luckily, QinetiQ and iRobot, two leading robotics companies, lent a small army of their robots to the cause. Robots with different morphologies, sensors, operational capabilities, knowledge representations, and spatial representations had to work towards the same goal and operate in the same environment. In these critical operations robots are usually used as tools to provide valuable information to humans who act as the decision maker and the actual consumer of the collected data and maps. It is expected that different robot teams will provide data and maps in different formats. Hence, an algorithm that can merge these multiple partial heterogeneous models into a single model can vastly improve the quality and utilization of the collected information especially when considering each model's strengths and weaknesses.

The inherent challenge of merging heterogeneous maps together arises from the fact that each of these maps, by definition, has its own spatial model and they lack a common representation. We believe that WiFi signals which are omnipresent in today's world can serve as the missing component in the attempt of solving heteroge-

neous map merging problem. These WiFi signals can be measured and recorded using a WiFi card. We believe that it is safe to assume that regardless of the type of map it generates, every robot carries a WiFi card on board to communicate with its team members and the base station. The idea is then whenever a robot creates any type of map (occupancy grid, topological, or appearance-based) it also records the WiFi signatures along with the data it stores in the map, i.e., producing an overlapping WiFi map of the environment it is mapping. With this respect, merging heterogeneous maps together boils down to finding partial correspondences between their WiFi maps and then fusing the heterogeneous information associated with matching WiFi signatures. Referring back to the nuclear reactor disaster scenario, this would allow the operators to send a small number of heavily equipped robust mobile robots with accurate occupancy grid map generating capabilities along with multiple cheap flying robots with cameras into the facility and visualize all the information in a single map.

1.1 Dissertation Contributions

In this dissertation, focusing on indoor environments we make contributions in five major research areas in the field of robotics. First, we introduce our novel visual navigation algorithm that steers a robot between two images through the shortest possible path. The algorithm presented in Chapter 3 targets mobile robots equipped with a monocular pan-camera and is invariant to changes in the tilt angle and the elevation of the camera. Due to these unique properties, images collected by another robot with a totally different morphology and camera placement can be used for navigation. This fact opens the doors to the possibility of gathering all images captured by an heterogeneous team of robots, and sharing them among all members of the team so that they not only have increased spatial awareness but also more candidate locations to navigate. Second, in Chapter 4 we focus on how these collection of images can be formalized into a spatial model, the appearance-based map. As we will discuss in Chapter 2, appearance-based maps are not novel and in the literature there are solutions proposed for appearance-based localization and mapping problem targeting single robots equipped with monocular vision. Our contribution here lies in demonstrating that these separately proposed methods can be integrated in a way that they constitute the basis of a robust system which does not suffer from the individual issues tied to the original proposed methods. We also integrate visual navigation algorithm into the framework and present navigation results in Chapter 4. Third, we propose a novel anytime algorithm to merge multiple

appearance-based maps together in Chapter 5. With this algorithm robots exploring the same environment can now merge their partial maps into a single map providing a more complete spatial model. Fourth, noting the lack of any evaluation criteria for appearance-based map, we introduce our task specific quality metric that measures the utility of the map with respect three major robotic tasks: localization, mapping, and navigation. Additionally, in order to measure the quality of merged appearance-based maps, and the performance of the merging algorithm, we propose the use of algebraic connectivity, a concept which we borrowed from graph theory. Fifth, in Chapter 6 we present our novel machine learning based heterogeneous map merging algorithm which uses WiFi signals as the common layer among different map types. We cast the idea of matching WiFi signals as a WiFi-based localization problem and present indoor and outdoor localization and map merging results in the same chapter. We then conclude the dissertation with final comments in Chapter 7 and discuss the possible directions that the research presented here may go.

CHAPTER 2

Related Work

2.1 Visual Navigation

With a vast literature on the subject, visual servoing, i.e., the use of computer vision data as feedback to control the motion of a robot, is one of the most studied problems in computer vision. More recently, the area has attracted significant attention as technology have made real-time execution of computationally demanding visual servoing algorithms possible. At the same time, robust methods for real-world scenarios have gradually enabled progress on realistic problems in terms of complexity. A comprehensive introduction on the topic is presented by Chaumette and Hutchinson [24, 25, 26].

There are two distinct situations to be considered. The camera can be attached to the robot end-effector observing the target (eye-in-hand) or fixed in the world observing both the robot and the target (eye-to-hand). These two schemes have technical differences and in general, one configuration is not better than the other since the choice depends on the application. The main difference is that eye-in-hand configuration has a partial but precise sight of the scene whereas the eye-to-hand camera has a less precise but global sight of it. Also different type of information extracted from perceived images such as points, lines, corners, or regions can be used as feedback, which leads to a varied spectrum of solutions. Among these vision-based navigation solutions, traditionally, most of them are dedicated to Autonomous Ground Vehicles (AGV), but recently, visual navigation is gaining more popularity among researchers working on Unmanned Aerial Vehicles (UAV). The variety of proposed solutions also extend to the areas they applied to, such as surveillance, patrolling, search and rescue, real-time monitoring, outdoor and indoor building inspection, and mapping. Desouza and Kak [35] and building on that work Bonin-Font et al. [15] published a detailed survey of visual navigation algorithms for mobile robots. Their taxonomy classifies algorithms mainly into two categories, map-based and mapless, and building on the definition of a map being either as metric or topological, appearance-based methods are categorized under mapless methods. However, as mentioned earlier in

Chapter 1 we advocate going beyond the traditional definition of a map and define any data structure that is used for localization and is build either online or offline and possibly updated during the navigation as a map. Within this respect, navigation algorithms using view sequences or appearance-based maps are considered map-based approaches. Hence, hereby we prefer to divide visual navigation algorithms into two main categories: position based visual servoing (PBVS) and image-based visual servoing (IBVS) as presented in [24]; while still borrowing some of the categories from their taxonomy.

The two main approaches in visual servoing, PBVS and IBVS differ in terms of their error definitions. In PBVS methods the vision sensor is considered as a 3D sensor so the 3D information captured from the scene along with the camera model is used to estimate the pose of the target in the Cartesian space with respect to the camera coordinate frame. In other words, keeping the focus on the 3D pose information, the error is defined as the difference between the desired and current robot or camera poses, while in IBVS, the vision sensor is treated as a 2D sensor since IBVS control objective and also control laws are directly expressed in the image feature parameter space. Within this respect, the latter is more robust with respect to uncertainties and disturbances on the robot model as well as the camera calibration error [81, 43]. Position based methods require online computation of target pose with respect to the camera. Computing that pose from a set of measurements in one image necessitates the camera intrinsic parameters and the 3D model of the object observed to be known. However, this is hardly a limitation of the approach since in the literature there are algorithms that estimate camera parameters online [151, 33] during navigation. IBVS algorithms, on the other hand, do not need a priori knowledge of the 3D structure of the scene. However, there are also some drawbacks to be considered for IBVS methods. One of the main disadvantages of IBVS is that when the displacement between the current and the target image is too large, the camera may reach a local minimum or may cross a singularity of the image Jacobian, also known as interaction matrix. Additionally, the feature depths are unknown in a pure IBVS framework, and must be estimated during servoing in order to compute the interaction matrix. Thus, only local stability can be guaranteed for most IBVS schemes [111].

2.1.1 PBVS methods

Position-based control algorithms use the pose of the camera with respect to some reference coordinate frame to compute the pose of the target and required actions to reach there. These algorithms generally fall into two categories: map-using sys-

tems and map-building systems. While map-using navigation systems need to be provided with a complete map of the environment before the navigation task starts, map-building navigation systems explore the environment and automatically build a map of it before they use it in the subsequent navigation stage. Other systems that fall within this category are able to self-localize in the local map they build simultaneously during the navigation. Typically, all of these algorithms utilize either 3D feature maps, occupancy grid maps, or a combination of them as the map representation.

The idea of extracting range data and depth measurements from images to be used for navigation and obstacle avoidance originates from the aim of using vision in an analogous way to ultrasound sensors. Once the obstacles are identified, the distance between the robot and the obstacle is computed in world coordinates. This approach, also called visual sonar, is adopted by Martens et al. [117] in their ARTMAP neural network based framework where sonar data and visual information from a single camera are fused together to obtain a more complete perception of the environment and actual metric distances to obstacles are computed based on the primitive yet very successful image-based distance computation algorithm by Horswill [75]. The distance computation algorithm is designed around the ideas that with the assumption that the carpet is always textureless, any area with visual texture should be an obstacle and if most obstacles touch the floor and the floor is roughly flat, then obstacles higher in the image are further away.

Veloso et al. [97, 45] presented a new visual sonar-based navigation strategy for the RoboCup [91] competition in mind where teams of Sony AIBO robots compete in the game of soccer. The vision module integrates domain specic knowledge and segments color images into known objects like floor, other robots, goals and the ball, and other unknown objects. A radial model of robot's vicinity is created after range and angle to all object are computed. Using this model and contour following techniques, the robot can avoid obstacles not only in its field of view but also where it is not currently looking.

Visual sonar concept is also used by Martin [118] to extract depth information from single camera images of indoor environments. A supervised learning framework is employed and images obtained in the training phase are labeled for the location of obstacles. Given labeled images, a genetic algorithm is used to automatically discover best monocular vision algorithm to detect ground boundaries of obstacles. These algorithms are then combined with a reactive obstacle avoidance algorithm originally developed for sonar.

All of these methods are designed with the assumption that the system will operate under constant floor color and lighting conditions. However, these assumptions severely limit the algorithms useability and performance in more realistic scenarios. Hence, more sophisticated methods use feature descriptors instead of color based obstacle boundary identifiers like the work presented by Se et al. [148, 147, 149]. The authors introduce a vision-based SLAM algorithm for a triclops camera system that used their own wide baseline matching technique. SIFT features extracted from images from these three cameras are combined with epipolar and disparity constraints. Matching features in subsequent views are identified by maintaining a database of located SIFT landmarks, which is is incrementally updated over time so that it adapts to dynamic environments. 3D world coordinates of these features are then estimated using the pixel positions of the matches, and the camera intrinsic parameters. Later, a least-squares minimization technique is applied on matching features in order to compute an accurate camera ego-motion which is used to correct possible localization errors. Elinas et al. [40] used the resulting 3D feature map and accompanying underlying occupancy grid map to autonomously navigate a robotic waiter in an indoor setting. With the goal of finding the shortest, but also the safest path, the navigation algorithm is designed as a combination of two path planning algorithms to allow the robot to navigate efficiently without getting stuck in cluttered areas. In clear areas, a shortest path planner algorithm is executed with a fixed distance constraint from obstacles, while in cluttered areas, a potential field planner takes over to avoid getting stuck.

More recently, Tomono [158] proposed a high density visual navigation system targeting indoor environments. Based on stored object models created offline, the algorithm performs online shape reconstruction and object recognition to estimate 3D locations of the objects during the navigation stage using monocular camera imagery accompanied by data captured by the on-board laser range finder. Assuming known camera intrinsic parameters, the author estimates the camera motion from the image sequence based on a structure-from-motion technique presented in [157].

One of the classic vision based localization, mapping and navigation approaches is presented by Wooden [166] for Learning Applied to Ground Robots (LAGR). After stereo depth maps are generated by matching small patches in two images from two calibrated cameras, the actual 3D position of image points are calculated through the transformation matrix deduced from the geometrical characteristics of the camera. The resulting stereo depth map provides a good estimate of the terrain, but needs some post-processing for safe navigation. First, the derivative of the map is computed in order to avoid too steep terrain and detect abrupt changes in slope, which may indicate trees or rocks. Then, to smooth some of the variation and

decrease the resolution of the map, the derivative map is transformed into a cost map, which is the average of the transformed derivative measurements over a fixed sized square region. Places where there is no derivative measurement, a high cost is assigned causing the robot to tend towards flatter parts of the environment. The robot is navigated by following a sparse set of waypoints provided by the $A*$ based path planning algorithm applied on the resulting cost map. Since running a global search using the planner may be too slow for real-time requirements, the planner only performs its search over the square region bounded by where the robot has terrain information so that the robot can start navigating using the initial waypoints. The list of waypoints are revised as planner broadens its search and updates the final path.

The commercial vSLAM system by Karlsson et al. [86] builds a connected map of recognizable locations using the odometry and extracted SIFT features. Extracted SIFT features are transformed into 3D landmarks using the visual front-end by Goncalves et al. [60]. A motion model via a combination of a particle filter [165, 61] and Extended Kalman Filter (EKF) [84] is utilized to continuously track the location of the robot in a two-bedroom apartment setting, while the estimated location of the robot is used for navigation. Similarly, Danesi et al [32] proposed a visual servoing algorithm also targeting 3D feature maps. 3D coordinates of extracted local invariant features are estimated via EKF and computed values are merged into a general 3D feature map which is used by the vision-based controller presented in [86] to steer the vehicle between frames.

2.1.2 IBVS methods

One early work in IBVS was proposed by Matsumoto et al. [119] where they proposed a view-based navigation method using the feedback from a monocular camera. The approach is motivated by sequence of images memorized during a training run and recorded directional relations between stored images. The memorized view sequence can be only used to follow the exact same trajectory by navigating between image pairs by executing recorded actions. However, the actions are specifically designed for corridor like environments and are too primitive. Moreover, due to the simplicity of the method and the assumption that the error between views should be a smooth and monotonic function, the algorithm does not generalize and fails to address navigation problems in complex environments. Building on this work, Jones et al. [3] defined a series of control rules based on the correlation peak-patterns. However, the algorithm still suffers from the simplistic and error-prone pixel by pixel view comparison step. The solution presented by Ohno et al. [133] is similar but faster than Jones' since

it saves memory and computational time by using only vertical lines from templates and online images to do the matching.

An interesting approach developed by Santos-Victor et al. [144] emulates the bees' flying behavior. The system moves in a corridor using two cameras to perceive the environment, one camera on each side of the robot, pointing to the walls. Just like bees the robot navigates centered in a corridor by computing the differences in optical flow from the images of both sides and measuring the difference of velocities with respect to both walls. If velocities are different, it moves to the wall whose image changes with minor velocity. Otherwise, goes straight in the center of the corridor. The main problem of this technique is that the walls need to be textured enough to present an optimum optical flow computation.

The paper by Kosnar et al. [93] attacking the problem of visual topological mapping for outdoor environments requires constant path color for visual navigation. Cross sections on the road are represented as vertices in the graph and vertex matching is realized by comparing the number of outgoing edges and their azimuths. With these assumptions this method is designed for very specific environments and the presented servoing method cannot be generalized for more realistic scenarios.

Inspired by the concept called Ego-Sphere first proposed by Albus [2], Kawamura et al. introduced the egocentric representations, Sensory Ego-Sphere (SES) and Landmark Ego-Sphere (LES) [82]. Authors defined the SES as the memory structure representing the robots perception of its current local environment and the LES as the expected state of the world at the target position. They both represent the objects and events around the robot by only using their angular distributions relative to the robot. A navigation algorithm based on these special Ego-Spheres is proposed [87, 88]. The presented algorithm used a simplified structure where 3D shape of the Ego-Sphere is projected onto the equatorial plane of the sphere, resulting in a 2D representation. At each step of the navigation, heading is computed by comparing the two egocentric representations without explicitly requiring range information.

More advance IBVS control implementations involve the computation of the image Jacobian which represents the differential relationship between the scene frame and the camera frame. The most common approach to generate the control signal for the robots is the use of a simple proportional control where the error is premultiplied with the pseudo-inverse of the Jacobian matrix before it is fed into the controller [136, 68].

In general, IBVS algorithms suffer from the singularity problem in the computation of image Jacobian matrix like in the algorithm presented by Hoffmann et al. [74]. Hence, to compensate these shortcomings algorithms entirely based on epipolar

geometry are introduced [41, 28, 113, 32, 150].

Maohai and colleagues [112] proposed an algorithm that uses the coordinates of image features and epipolar constraints which are well studied in the field of multi-view geometry. The algorithm employs the Random Sample Consensus (RANSAC) [52] method to find the robust estimation of the fundamental matrix which can be calculated with only five matches thanks to their assumption that the motion of the robot is restricted to a plane. The disadvantage of this method is that due to the loss of depth information in the image capturing process, translation can only be recovered up to a scale factor. Therefore, odometry information is used to estimate the ratio constant. The same epipolar geometry based approach is used by Booij et al. [16] to navigate a robot with an omnidirectional camera in an appearance based topological map. The prior knowledge that positions of the cameras do not differ in height and the relative rotation only occur around the vertical axis is incorporated by restricting the essential matrix which enables its computation with only four matchings. The desired heading to navigate from the current image to the target image is computed from the translational information extracted from the essential matrix. The experimental results showed that the robot did not drive the exact trajectory that was driven while taking the dataset; indeed, it used a shorter path.

Lin et al. [100] presented a homography matrix based visual servoing method. Homography matrix is approximated by a simpler form of affine transformation and motion control rules for forward translation and rotation are defined on the affine transformation matrix. A similar study was performed by Santos-Victor et al. [145] whose work is restricted to navigating in corridors and environments alike. Since the presented visual navigation algorithm is based on vanishing point, the method fails in environments with large open spaces. Other examples of homography based servoing were reported in [142, 169, 137, 64].

In [28, 114, 113] Mariottini and colleagues solve the the navigation problem for a non-holonomic robot based on epipolar geometry. Their work focuses on a single robot approach and assumes the robot navigates using a data structure based upon formerly collected data. Due to the non-holonomic constraints, the produced path significantly deviates from the shortest one between current and goal positions.

Similarly, epipolar geometry based navigation algorithms for holonomic [29] and nonholonomic [104] robots are proposed. These approaches do not require any 3D distance computations since the control is based directly on the trajectory of the epipoles. Specifically, in [29] the authors introduce mobile robot control laws as a part of their visual servoing framework exploiting the epipolar geometry from object profiles and epipolar tangencies but without solving any correspondence problem.

Finally, there are also methods that combine the advantages of both worlds like the hybrid switched system approach proposed by Gans and Hutchinson [53]. The proposed algorithm switches between two controllers designed for IBVS and PBVS based on the availability of visual features. Comparing to IBVS approaches, the algorithm needs more time to converge due to the fact that the image error may increase during the time the controller is in PBVS mode minimizing position based error. The biggest problem with this method is that the global stability is not guaranteed and the potential issues due to the local minima as pointed out by Chaumette [23] are not considered in this approach.

2.2 Vision-based Localization and Mapping

Early appearance-based approaches like the one presented by Matsumoto et al. [120] modeled the environment as a sequence of images where localization and mapping problems are solved independently. The map is created in a supervised setting and given a map, a query image is localized to a recorded view that has the minimum accumulated error computed by pixel-by-pixel image intensity level comparison. However, the proposed simple similarity function is not invariant to scaling, orientation, or partial overlaps.

The papers by Se et al. [148, 147, 149] present a vision-based SLAM algorithm for a three-camera system. 3D coordinates of extracted SIFT features are used for mapping and localization. In order to eliminate false feature matchings, each feature in the database carries statistical information regarding its lifetime, number of occurrences, etc. and this information is used to measure the validity of that feature. Similarly, there are several studies [112, 86] representing the world by a spatial map that contains 3D feature coordinates. The work proposed by Danesi et al. [32] relies on a hybrid (metric and topological) map built on visual cues. Within the topological map, images represent nodes in the graph and edges represent feasible paths that the robot can traverse by the PBVS controller presented in [86]. The 3D coordinates of the extracted features are estimated via EKF and encoded in the metric map.

After Lowe [105] introduced SIFT in 2004 and successfully demonstrated that they can be used to robustly differentiate images from one another, more sophisticated appearance-based algorithms has started to emerge. Valgren and colleagues [164] proposed an incremental topological mapping algorithm for robots with omnidirectional cameras. In this approach, a node in the map is a collection of sequential images that are considered similar enough where the similarity is defined as sufficient

number of feature matches. However, due to the lack of geometrical constraints the algorithm carries a high risk of misclassification. The links, on the other hand, are created after processing the affinity matrix that theoretically stores every possible image comparisons. In order to prevent the quadratic growth of the search space they proposed a random sampling algorithm with a linear complexity. Nevertheless, the paper does not provide any solutions how to efficiently store and use the affinity matrix with very large number of images. Differently, Mulligan and Grudic [131] proposed a semi-supervised learning technique to segment image sequences into a topological map by clustering on low dimensional manifolds in sensor space.

Jeong and Lee [83] proposed a new vision-based SLAM technique where salient image features are detected and tracked through the image sequence captured by a single camera looking upward. Compared with the conventional frontal view system, the ceiling vision has advantage in tracking, since it involves only rotation and affine transform without scale change. Perceptual aliasing, however, is the main issue since ceiling imagery mostly is featureless and has substantial repetitions.

More recently, Lin et al. [100] proposed a two-stage place recognition system in which an offline learning stage should take place before online recognition can be used. Focusing on the localization aspect, the paper does not provide any mapping algorithm. Similarly to the other works in the literature [112, 162] localization to one of the images stored in the learning stage is implemented by comparing the features from the current image to the features in the database using a kd-tree based approximate nearest neighbor search algorithm.

Zivkovic and colleagues [16, 173, 175, 176] presented an appearance-based topological mapping algorithm. Without any metric information the method builds an appearance graph using SIFT features and uses geometric constraints for image comparison. In addition, they use a graph partitioning method to cluster vertices in the appearance graph and construct a high-level map. However, the algorithm requires the number of clusters to be known in advance. Besides this assumption, they use a fixed assignment of vertices to the clusters that limits the algorithm's adaptivity to the changes in the environment.

One of the few systems where vision is used in the context of metric and topological mapping is described in a few recent papers by Kŕose and colleagues [16, 174, 177], where the robot however relies on an omnidirectional camera. Comparing to the monocular vision, the use of omnidirectional images greatly simplifies the problem, because thanks to their rotational invariance omnidirectional images can be associated with the position where they have been acquired, disregarding the robot heading.

2.3 Map Merging

In this section we will cover the proposed solutions for the map merging problem. Map merging research in general has been limited and most related works appeared relatively recently when compared to the mapping literature. In particular, most of the research efforts have been devoted to solve the problem of merging metric maps, either feature-based or occupancy grid. Carpin et al. [22, 12] proposed a map merging algorithm that utilizes an optimization framework to compute the optimal transformation between two occupancy grid maps. This approach is later ameliorated into real-time merging algorithm that using Hough spectrum tracks multiple hypothesis to cope with ambiguities [20, 21]. Similar ideas have been recently proposed in [56, 55].

Former research is much more limited when it comes to topological map merging or appearance-based map merging. A general solution to the problem of topological map merging is proposed by Huang and Beevers [78, 79, 11]. The algorithm identifies common subgraphs using a vertex clustering method based on the SVD of the covariance matrix. Each pair of common subgraphs becomes a putative solution candidate for merging. Exploiting the knowledge integrated into the graph structure such as the degree of vertices and edge properties, the candidates that pass a geometrical test are considered as valid solutions. The main problem with this approach is its computational complexity since it has been shown that maximal common subgraph problem is computationally intractable [85]. Additionally, this approach due to its generality, does not provide methods to use it for appearance-based graphs which have some specific properties that should be preserved such as the feature correlations used for visual servoing.

Another study published by Ferreira et al. [46] addresses the problem for merging topological paths annotated with images and range scans. In particular, they proposed a two-step approach to find overlaps in view sequences and stitch them together into a generic topological map. Tentative overlaps are first discovered from the similarity matrix built by pairwise comparisons of all images and identified overlaps are merged together after they are verified based on the idea that matched similar vertices should also have similar neighbors. One disadvantage of the method is that merging procedure does not start until after the similarity matrix is built by pairwise comparisons of all images and local alignments are found. Hence, the algorithm is clearly offline and its performance suffers as the number of images increases.

Ho and Newman [73] proposed a system in which an algorithm to identify matching subsequent images between two maps is implemented. They also process the

similarity matrix and apply a modified version of Smith-Waterman algorithm [154] to find local alignments. The proposed method similarly does not scale well with respect to the number of images due to the dependency on the creation of full similarity matrix. One of the few multi-robot visual localization and mapping studies is presented by Hajjdiab and Laganiere [64]. In this paper each robot equipped with a monocular camera starts from an unknown location and incrementally builds a local visual map of the environment with the ability to localize itself in the map. In the case of an overlap between any two robots, images from both maps are stitched together using inter-image homography and the resulting joint map is utilized by both robots. However, the merged map is not suitable for a visual servoing approach that is based on the movements of epipoles like the ones presented in [41, 113].

The problem of merging appearance-based maps is not addressed yet. The very first solution to this problem we proposed in [42] will be presented in Chapter 5. Similarly, all former attempts have considered instances where all maps to be merged are of the same type. Heterogeneous maps, on the other hand, in general have scarcely been considered, although never in the context of merging, in [32, 132, 167]. In Chapter 6 to the best of our knowledge we will introduce the first heterogeneous map merging algorithm.

2.4 Wifi Localization

As mentioned in Chapter 1 our heterogeneous map merging algorithm uses WiFi maps as a common layer and the correspondences between maps are identified by *localizing* WiFi signatures from one map in another. We will present the details of our proposed WiFi localization algorithm in Section 6.3. Hence, in this section we will review the state of the art in WiFi localization. The utilization of wireless signals transmitted from Access Points (APs) or home-made sensors has enjoyed great interest from the robotics community, in particular for its applicability to the localization problem. In this area, the application of data-driven methods has prevailed thanks to two distinct but popular approaches. First, the *signal modeling* approach attempts to understand, through collected data, how the signal propagates under different conditions, and the goal is to generate a signal model that can then be exploited for localization. In some sense, the *signal model* provides an estimation that can be compared to current signal readings to infer a robot's position. Second, the *signal mapping* approach directly uses collected data by combining spatial coordinates with wireless signal strengths to create maps from which a robot can localize. The resulting WiFi map can then be utilized by robots to localize based on current wireless

readings. Since signals are distorted due to "typical wave phenomena like diffraction, scattering, reflection, and absorption" [17], practical implementations of *signal modeling* are not yet available for unknown environments since they need to be trained in similar conditions to what will be encountered (i.e., they require at least some a priori information about the environment) [171]. Moreover, it has been shown in [76] that a signal strength map should yield better localization results than a parametric model. Consequently, the rest of this section highlights related works regarding the mapping approach, which we employ for our algorithm. A comprehensive study on signal modeling techniques is presented by Goldsmith [59].

Signal mapping can be divided into a training and a localization phase. During the training phase, *signal mapping* techniques tie a spatial coordinate with a set of observed signal strengths from different APs, essentially creating a signal map. This mapping process is repeated until the entire environment, or a particular region of interest, is covered and results in a WiFi map. The training phase is generally performed offline and can be acquired by a human instead of a robot [95, 47]. In fact every cited publications in this section performs the training phase offline. Given a new set of observed signal strengths acquired at an unknown position, the goal of WiFi localization is to use the map acquired during training to retrieve the correct spatial coordinate. A variety of methods have been devised to solve this problem, with two of the earliest solutions being nearest neighbor searches [7] and histograms of signal strengths for each AP [168]. Analysis of the distribution of signal strength readings suggested that they are normally distributed [76, 51], paving the way to an abundance of Gaussian-based techniques that attempt to model the inevitable variance in signal strength readings.

The first Gaussian-based solution to WiFi localization [76], which we subsequently call the *Gaussian Model*, consists in fitting a Gaussian distribution for each location and each AP, using data acquired during the training phase. Once the Gaussian distributions have been computed, an unknown location described by a new set of observed signal strengths can be determined using the Gaussians' Probability Density Function (PDF). Its simplicity, speed, and good accuracy have made it, even to this day, one of the most popular WiFi localization techniques [76, 95, 13]. Other popular Gaussian-based algorithms for WiFi localization involve Gaussian Processes (GP), whether they are used directly [48, 38], within a Latent Variable Model framework [47], or improved upon to create a new algorithm called WiFi GraphSLAM [77]. Gaussian Processes and their variants are very promising since they attempt to establish the correlation's strength between training data points (e.g., points close together should have similar observations), making them especially

suited for regression. Unfortunately, they require a long parameter optimization step [38] that makes them difficult to use in scenarios where real-time operation is required and computational power limited.

For completeness, we mention that WiFi localization can be cast as a machine learning problem and, consequently, any algorithm capable of classification or regression could be exploited for WiFi localization. For example, logistic regression, multinomial logit, neural networks, and boosting could all be utilized to solve the WiFi localization problem. In the literature, however, only Support Vector Machines (SVMs) [161] and Random Forests [8] were shown to perform well in terms of accuracy. Whereas SVMs suffer from slow training sessions, Random Forests are fast to train, can localize quickly, and are very accurate, currently making them the best candidate for online WiFi localization in unknown environments [8].

The accuracy of all the methods described in this section has been significantly increased by taking into account a robot's inherent temporal and spatial coherence over time using Hidden Markov Models (HMMs) [95, 37] or MCL [76, 13]. Using its state transitions, an HMM places probabilistic restrictions regarding which states a robot can move to. The HMM essentially corrects potentially wrong WiFi localization that would result in impossible robot motion for a given time step (unless the robot was kidnapped). Although HMMs for the purpose of WiFi localization work both in theory and practice, they are often created manually due to the difficulty of accurately inferring them automatically. MCL combines a robot's motion model with a measurement model derived from one of the aforementioned WiFi localization techniques. Similarly to HMMs, the robot's motion model adds robustness to the WiFi localization by reducing the probability for robot locations that cannot be predicted by its motion model. Conversely to HMMs, however, MCL does not need environment-dependent information or a human in-the-loop. The manual creation of HMMs is evidently not suitable for online WiFi localization and we consequently prefer the MCL approach to correct WiFi localization errors as a robot moves through an environment.

CHAPTER 3

Visual Navigation

Visual navigation is evidently one of the most fundamental building blocks of a heterogeneous multi-robot system operating in an unknown environment relying solely on visual sensors. In this chapter building upon different contributions made in the past in the fields of computer vision, mapping, and visual servoing, we present the first steps towards the implementation of this vision. Focusing our attention to domestic environments and mobile platforms with limited computational power, we describe how it is possible to implement an image-based visual servoing strategy that steers a non-holonomic mobile robot equipped with a monocular camera so that its currently perceived image matches a given target view. In addition to proposing this novel algorithm, we extend the single robot visual navigation framework to multi-robot heterogeneous systems. Even though this step is conceptually the last one, i.e., it is executed after a set of images is collected and organized into an appearance-based map, its discussion is presented first since it introduces some concepts related to camera geometry that will be used later on. The results presented in this chapter have been published in [41].

3.1 Problem Formulation

In this section we first describe the system where our vision-based controller is used. After explaining the camera model we utilized, we overview some topics in multi-view geometry that constitute the fundamentals of the presented navigation algorithm.

3.1.1 System Model

We consider a set of N robots moving on a plane with with non-holonomic motion constraints which we model each robot as a unicycle. It is assumed each robot in the system is equipped with a pan actuated monocular camera that can be turned to a desired direction. The state of each robot is a vector in $\mathbb{R}^4$ and is defined as $[x\ y\ \theta\ \phi]^T$, where x and y are the Cartesian coordinates of the center of the robot, θ is

the orientation of the robot with respect to x axis of the world coordinate frame, and ϕ is the orientation of the camera with respect to the robot's heading. Without loss of generality, we assume that the camera is placed concentrically with the robot at a prefixed elevation, and is tilted by an arbitrary known angle. The general formula for the kinematic model of a differential drive robot is the following.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & 0 \\ \sin\theta & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{3.1}$$

The input vector for the system is $U = [u_1 \ u_2 \ u_3]^T \in \mathbb{R}^3$ specifying translational speed (u_1), rotational speed (u_2), and the rotational speed for the camera (u_3).

3.1.2 Camera Model

In our setup all robots are equipped with monocular cameras with a limited field of view and we adopt the ideal pinhole camera model as the imaging model due to its simplicity. It is an idealization of the thin lens model in a way that it describes the camera aperture as a point and defines a mapping between the coordinates of a 3D point and its projection onto the image plane. Let the center of projection be the origin of an Euclidean coordinate system which we denote as the *optical center* or *camera center* and its z axis as the *optical* or *principal axis*. The plane $z = f$ perpendicular to this axis is the image plane and the intersection point of the image plane with the optical axis is called the *principal point*.

Under the pinhole camera model, a point in space with coordinates $\mathbf{X} = [X, Y, Z]^T$ is mapped to the point $\mathbf{x} = [x, y]^T$ on the image plane where a line joining the point $\mathbf{X}$ to the optical center meets the image plane. As can be seen in Figure 3.1 the coordinates of X and $\mathbf{x}$ are related by the so-called ideal *perspective projection*

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{3.2}$$

where f denotes the focal length of the camera. In homogeneous coordinates, it can be written as

$$Z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.3}$$

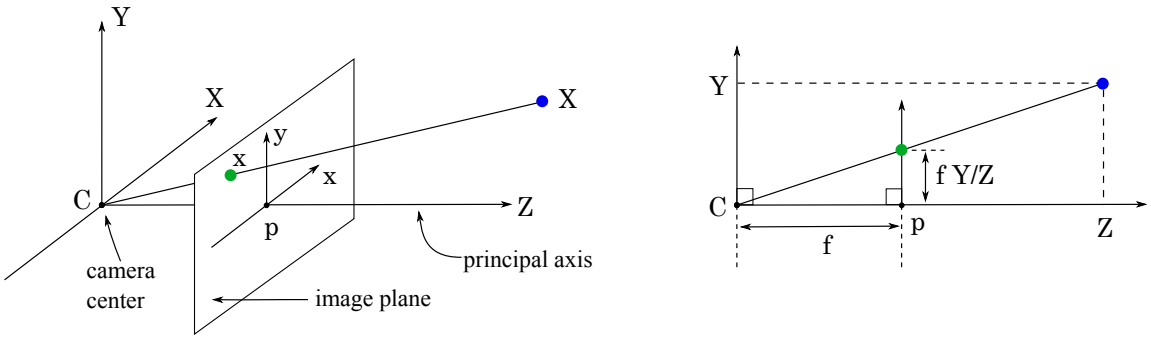


Figure 3.1: Pinhole camera geometry is illustrated where C represents the optical center and p the principal point. The image of the point X is the point x at the intersection of the ray going through the optical center and the image place at a distance f from the optical center.

Since the depth of the point $\mathbf{x}$, Z, is usually unknown, it can be simply written as an arbitrary positive scalar $\lambda \in \mathbb{R}^+$. With the decomposition of the above matrix, we get

$$\lambda \mathbf{x} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X} \tag{3.4}$$

We define two matrices as

$$K_f = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3x3}, \quad \Phi_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{3x4} \tag{3.5}$$

To summarize, using the above notation, the overall geometric model of an ideal camera reduces to

$$\lambda \mathbf{x} = K_f \Phi_0 \mathbf{X} \tag{3.6}$$

The equation 3.6 is derived based on a specific choice of reference frame centered at the optical center with one axis aligned with the optical axis. However, in practice digital cameras obtain their measurements in terms of pixel values (i,j) with the origin of the image coordinate frame located at the upper-left corner of the image. Therefore, the relationship between the pixel array and image coordinate frame should be specified. Due to the unit differences between two coordinate systems, we first need to find the scaling factor for each axis to convert metric units to pixel coordinates. If $[x_s, y_s]$ is the scaled version corresponding to pixel coordinates, and

s_x and s_y are the computed scaling factors, then the transformation can be written as the following

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.7}$$

For most of the cameras, each pixel in the sensor array is square so we can usually assume $s_x = s_y$.

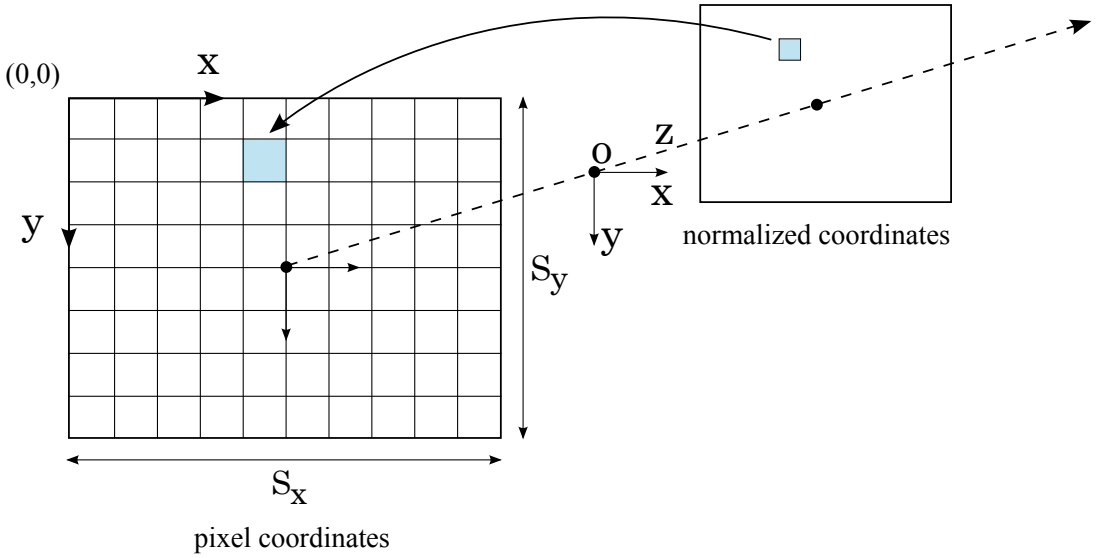


Figure 3.2: Transformation from normalized coordinates to coordinates in pixels.

Next, we need to translate the origin of the reference frame to the upper-left corner of the image since x_s and y_s are still specified relative to the principal point. Given o_x and o_y are the pixel coordinates of the principal point relative to the image reference frame, the actual image coordinates $[x', y']^T$ are described as following

$$\begin{aligned} x' &= x_s + o_x, \\ y' &= y_s + o_y \end{aligned} \tag{3.8}$$

In practice, pixels may not be exactly rectangular so a more general form of the scaling matrix includes a skew factor, s_θ. The overall transformation then takes the general form

$$x' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.9}$$

Finally, if we combine the projection model described in Equation 3.4 with the translation and scaling, we get the complete transformation formulation in homogeneous coordinates between a 3D point in camera frame and its corresponding reflection in the image frame in pixel coordinates.

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.10}$$

Here the standard projection matrix Φ_0 defines a perspective projection with respect to a normalized coordinate system as if the focal length was equal to 1. Then, a pair of transformations that depend on the parameters of the camera is applied to get the final pixel coordinates. These camera parameter related transformations are combined into a 3×3 *intrinsic parameter matrix* K, also known as the *calibration matrix*.

$$K = K_s K_f = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Currently, most of the high-end cameras used in robotics applications come with all the required information about their intrinsic parameters to fill the calibration matrix. However, due to the small differences occurring during the camera and lens production processes, this information may not be accurate enough for some vision applications. In the most extreme case, none of these parameters are known in advance which is the case with most of the webcams available in the market. Nevertheless, the intrinsic parameters of a camera can be estimated through a procedure called *calibration* as described in [70]. Given a set of images captured by the camera, the calibration can be done either manually by identifying the projection of the same 3D point in all images or automatically by employing a feature extraction and matching algorithm as demonstrated in [101]. In our system, for simplicity, we assume that all the robots have identical cameras and their calibration matrices are known and equal.

3.1.3 Epipolar Geometry

As mentioned in the previous section, the goal of any visual navigation algorithm is to steer a robot between two images. More precisely, given an initial and a target

image, a sequence of waypoints that would lead the robot from the initial camera pose to the final camera pose needs to be determined. If the initial and target camera poses could be extracted, assuming the robot has other means of localization, the whole problem would reduce to a path planning/tracking problem. Hence, it is essential to understand the geometric relationship between two images and study the properties of multi-view geometry some of which are exploited by our navigation algorithm. Here we will describe some of the building blocks of the geometry of two views, known as *epipolar geometry*.

With reference to Figure 3.3, let us consider two cameras whose relative displacement is given by a rototranslation (R, T), where $R \in \mathrm{SO}(3)$ is a rotation matrix, and $T \in \mathbb{R}^3$ a translation vector. Let K be the common intrinsic camera parameter matrix. The first camera position will be indicated as the *actual* camera position, while the second is the *desired* camera position (hence the subscripts used in the following).

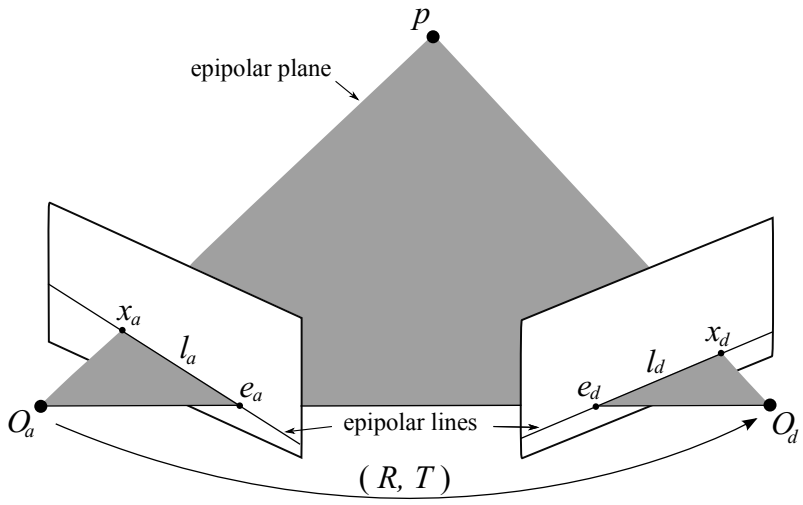


Figure 3.3: Two projections x_a and x_d of a 3D point p from two perspective points. The frame on the left is associated with the actual camera position, while the frame on the right indicates the desired camera position. The intersection of the line connecting camera optical centers, O_a and O_d, with the two image planes determines the two epipoles $\mathbf{e}_a$ and $\mathbf{e}_d$.

The line connecting the centers of the reference frames associated with the two cameras is called *principal line* or *baseline*, and let $\mathbf{e}_a$ and $\mathbf{e}_d$ be the intersections of the principal line with the image planes of the first and second camera, respectively. Points $\mathbf{e}_a$ and $\mathbf{e}_d$ are called *epipoles*. The horizontal coordinate of $\mathbf{e}_a$ in the actual

image plane will be indicated as $\mathbf{e}_{au}$, and a similar notation is used for $\mathbf{e}_d$. The line l connecting the epipole to the projection of point p on the image plane x is the *epipolar line*. By definition it is evident that the point on the image plane corresponding to p is restricted to the epipolar line. In other words, the epipolar line in the desired image l_d is the projection of the ray from the point p through the camera center of the actual camera O_a. The projective mapping $x_a \mapsto l_d$ from points to lines is represented by the *fundamental matrix* F which is computed as the following

$$F = K^T \widehat{T} R K^{-1} \tag{3.12}$$

where $\widehat{T}$ is the skew-symmetric matrix associated with the translation vector T. The geometric and arithmetic derivation of the fundamental matrix can be found in [65]. For any point x_a in the actual image, the corresponding epipolar line is $l_d = Fx_a$. Similarly, $l = F^T x_d$ represents the epipolar line corresponding to x_d in the desired image. Since x_d lies on the epipolar line $l_d = Fx_a$, we get $x_d^T l_d = 0$. In other words, for two cameras with non-coincident centers, the fundamental matrix is the unique 3×3 rank-2 homogeneous matrix that satisfies

$$x_d^T F x_a = 0 \tag{3.13}$$

for any pair of corresponding points $x_a \leftrightarrow x_d$ in the two images.

The epipolar line $l_d = Fx_a$ also contains the epipole $\mathbf{e}_d$. Thus $\mathbf{e}_d$ satisfies ${\mathbf{e}_d}^T l_d = ({\mathbf{e}_d}^T F)x_a = 0$ for all x_a. Similarly, $x_d(F\mathbf{e}_a) = 0$. It follows that the epipoles are in the left and right null spaces of the fundamental matrix.

$$\begin{aligned} {\mathbf{e}_d}^T F &= 0 \\ F\mathbf{e}_a &= 0 \end{aligned}$$

Hence, given F the epipoles can be computed. It is however immediate to realize that knowledge of F, or of the epipoles, does not imply that the transformation (R, T) can be retrieved. As evident from Figure 3.3, epipoles are invariant for translations along the principal line.

The fundamental matrix F can be computed using various algorithms known in literature like the ones described in [65, 109] and it requires knowledge of a certain number of matching features between two images taken by the cameras. The simplest method among them is the eight-point algorithm introduced by Longuet-Higgins [103] which requires eight point correspondences between two images. The problem with this approach is that the least-squares solution of the homogeneous linear system of equations is suboptimal in the presence of noise in image coordinates. In theory,

the matrix formed from the corresponding points should have one singular value equal to zero with the rest of them being non-zero. In practice, however, if more than eight correspondences are used to solve for F or the image coordinates are affected by various types of noise, some of the non-zero singular values can become small relative to the larger ones and there may not be a well-defined singular value which can be identified as approximately zero. Hartley [66] showed that this problem is mainly caused by the poor distribution of the homogeneous image coordinates in their space and proposed the normalized eight-point algorithm that produces a better estimate of the fundamental matrix than the one would have been obtained by estimating from the un-normalized coordinates. There are also other more complex methods which due to their nonlinear minimization may produce better results such as Levenberg-Marquardt algorithm [98, 116] based algebraic error minimization technique [67] or the Maximum Likelihood Estimation (MLE) based the Gold Standard Method [109] that minimizes the geometric distance which corresponds to the reprojection error.

The quality of the estimated fundamental matrix can be measured using an error metric. One obvious choice is to compute the reprojection error which corresponds to the geometric distance

$$\sum_i \|x_i - \hat{x}_i\|_2^2 + \|\hat{x}'_i - x'_i\|_2^2 \tag{3.14}$$

where $x_i \leftrightarrow x'_i$ are the point correspondences, and $\hat{x}_i$ and $\hat{x}'_i$ are the estimated correspondences that satisfy $\hat{x}_i'^T F \hat{x}_i = 0$ for a rank-2 matrix F. However, instead of computing a set of subsidiary variables, namely the estimated correspondences, we can take a first-order approximation to the reprojection error, called the *Sampson distance* which only involves the elements of F. Sampson distance is defined as

$$\sum_i \frac{(x_i'^T F x_i)^2}{(Fx_i)_1^2 + (Fx_i)_2^2 + (F^T x'_i)_1^2 + (F^T x'_i)_2^2} \tag{3.15}$$

where $(Fx_i)_j^2$ represents the square of j-th element of the vector Fx_i.
Another alternative is to use symmetric epipolar distance defined as

$$\sum_i d(x'_i, Fx_i)^2 + d(x_i, F^T x'_i)^2 \tag{3.16}$$

where the function d returns the distance in pixels between a point and a line. The error is the sum of squared distances between every single matching point and its corresponding epipolar line in the other image. However, as shown by Zhang [170]

minimizing Sampson distance outperforms this cost function. More detailed discussion on different choices of cost functions and algorithms to estimate the fundamental matrix can be found in [159, 109, 65].

Independent of the method used to compute the fundamental matrix, the problem of noise affecting the image coordinates of the correspondences plays a vital role in the accuracy of the estimation, and in practice, the source of inaccuracy is not only the noise, but also possible false point correspondences used as the input to these estimation algorithms. Thus, to minimize the effects of these false point matches, RANdom SAmple Consensus (RANSAC) [52] method is utilized. The basic assumption behind the RANSAC algorithm is that the data we try to explain with a mathematical model contains inliers (data that fit the model) along with some outliers (data that cannot be explain with the model) cause either by noise or faulty observations. Being a non-deterministic algorithm, RANSAC iteratively subsamples the data and computes the error between the (usually small) data and the model. The idea is that as more iterations are executed, the probability of getting a dataset containing only inliers increases which results in a better estimation of the parameters of the mathematical model that fits this data.

Algorithm 1 estimateFundamentalMatrix()

```
for n = 1 → N do
    X_s ←RandomSample(X, 8) {randomly selects 8 correspondences}
    F ←NormalizedEightPointAlg(X_s)
    S ← ∅, S_best ← ∅
    for all (x, x') ∈ X do
        d ←SampsonDistance((x, x'), F)
        if d ≤ ε then
            S ← S ∪ (x, x')
        end if
    end for
    if size(S) > size(S_best) then
        F_best ← F
        S_best ← S
    end if
end for
F ←NonlinearEstimation(S_best)
```

We integrate RANSAC with normalized eight-point algorithm to get a more robust estimation of the fundamental matrix which, in pseudo-code, presented in

Algorithm 1. At every iteration of RANSAC, first, a random sample of 8 correspondences are selected. Then, using normalized eight-point algorithm the best fitted fundamental matrix is computed and the error between the estimated F and this subset is calculated using Equation 3.15. Next, inliers which are point correspondences supporting F are identified among all tentative point matches. A match (x_i, x_i') is said to be supportive of the F if its Sampson distance is less than ϵ, a preset pixel threshold. These steps are repeated for a number of iterations which can be determined adaptively and the F with the largest number of supporters is chosen as the best fundamental matrix and its supporting matches are identified as final inliers. Finally, the F is re-estimated from all correspondences classified as inliers. For this last step, in order to make use of all the correspondences we iteratively minimize the Sampson distance using a more sophisticated nonlinear approach like the Levenberg-Marquardt algorithm.

3.2 Navigation Algorithm for Heterogeneous Multi-Robot Systems

3.2.1 Single Robot Navigation

Mariottini et al. [113] proposed a two-stage method that controls a non-holonomic robot equipped with a fixed camera. Given the actual camera position, and a desired camera position specified as a target view, their control schema moves the robot so that the actual image eventually matches the desired image solving the visual navigation problem illustrated in Figure 3.4. Due to the non-holonomic constraints, the produced path significantly deviates from the shortest one between the current and goal poses. Given that the system model we formerly assumed allows for rotations in place and includes an additional degree of freedom for the camera that allows us to rotate the camera around its vertical axis, we extend their approach into a four-stage control schema which is more articulated but produces the shortest possible motion.

The four-step visual navigation strategy targeting pan camera equipped robot systems is described in the following. Due to the simplicity of the kinematics of the system and the fact that the overall navigation algorithm is decomposed into four independent steps, without loss of generality, we employed proportional controllers and their parameters are tuned manually. In practice one can use more sophisticated approaches, such as PID controllers, and their parameters can be tuned using Ziegler-Nichols method [172]. The k_i factors are the proportional gains of the proportional controllers used in the various steps. The algorithm proceeds to next step when the

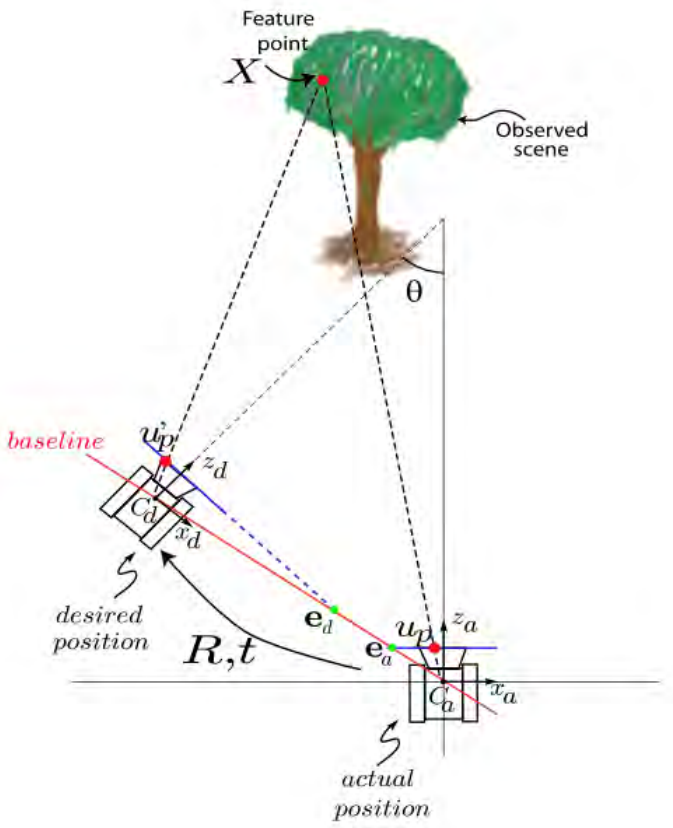


Figure 3.4: Illustration of the visual navigation problem.

error defined for the current step falls below a preset value. Figure 3.5 provides a graphical illustration of the four step approach. Four steps are listed as following.

1. From the initial configuration, applying the control law $U = [0, -k_1\mathbf{e}_{au}, 0]^T$ the actual epipole's horizontal coordinate $\mathbf{e}_{au}$ is brought to 0. The desired behavior is that the robot rotates around its z axis towards the target configuration. At the end of this step, the robot's heading will be aligned with the principal line.

2. Keeping the robot fixed in place, the camera will be rotated to align the actual camera coordinate frame with the desired camera configuration. The control law brings the horizontal coordinate of the actual epipole $\mathbf{e}_{au}$ to the horizontal coordinate of the desired epipole $\mathbf{e}_{du}$: $U = [0, 0, -k_2\,(\mathbf{e}_{au} - \mathbf{e}_{du})]^T$.

3. In the first two steps the epipoles are locked to appropriate values to produce

the desired behaviors. Additionally, a relative translation of the image planes along the baseline cannot be derived from the epipoles. Thus, an additional source of feedback is needed. In this step we use feature match based error as feedback. The error Err_{pix} is defined as the maximum error between the horizontal image coordinates of matched features. The control law for this step is the following: $U = [-k_3 Err_{pix}, 0, 0]^T$. As the error goes to zero, the robot approaches the desired robot position while moving along the baseline. During this motion the actual camera heading is kept aligned with the desired camera configuration. At the end of this step the robot will be at the desired location and the camera will have the same view.

4. If it is desired to match the robot's heading with the camera heading, then in the last step the robot should be rotated so that the angle between its heading and the camera orientation goes to zero. Since the camera is attached to the robot, its rotation also affects the camera. Thus, the camera rotates simultaneously in order to compensate the rotation of the robot. The resulting control law is defined as $U = [0, -k_4 Err_{pix}, -k_5 \phi]^T$.

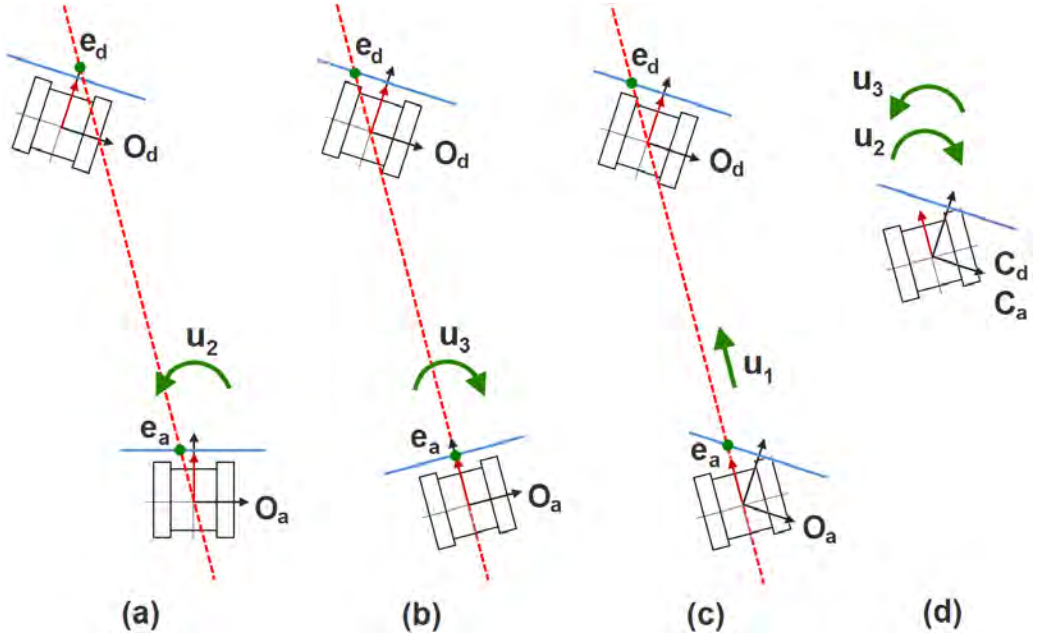


Figure 3.5: Four steps needed to bring the robot to the desired position specified by a target view is described.

The proposed approach works well if the target robot configuration is in the field of view of the initial camera configuration. Otherwise, in step-1 when the robot

turns toward the target position, the cameras may lose the overlap in their observed scenes and the number of common features therefore may fall below the required number to robustly estimate the epipoles. For instance, consider the inverse problem of navigating from the goal position to the initial position in Figure 3.5. Please note that the omni-directional camera based systems are not affected by this problem since their field of view is 360° so their acquired observation through the camera is independent from the heading of the robot. On the other hand, our epipole-based visual servoing approach working on a monocular camera suffers from this issue since it does not directly consider the area of the overlap between two images or the number of feature matches. Hence, in the current implementation, we first test the target position to see if it lies in the field of view of the initial camera configuration. If it is in the field of view the described algorithm is applied which results in the near-optimal robot motion. Otherwise, the method presented by Mariottini [113] can be applied which keeps the scene under common surveillance with a cost of increasing the path traveled by the robot.

3.2.2 Multi-Robot Navigation

Up to now we assumed that the target image is taken from the same camera by the same robot that tries to reach it. Even though this assumption is valid for most of the current robotics applications like a robot building and using its own map, the value of integrating robots with different capabilities to create a multi-robot heterogeneous system is incontrovertible. Based on this idea, we extend our visual navigation algorithm so that it provides a viable control strategy that is independent of utilized camera configuration. In other words, a target image captured by a certain robot can be used by a different mobile platform for navigation purposes. This ability greatly enhances the utility of the system. For the moment we assume the robots are equipped with same cameras, i.e. same intrinsic camera parameters, however, full heterogeneity with the usage of different cameras can be achieved by utilizing an automatic intrinsic parameter estimation algorithm like the one described by Liu and Hubbold [101]. Changing the elevation of the camera only shifts the projection of the observed scene along the vertical axis of the image plane and does not have any effect on horizontal coordinates of the image pixels. Since the navigation algorithm is designed based on the u-coordinates of the epipoles, the elevation of cameras does not affect the movement of e_{au} and e_{du}. Thus, the proposed navigation algorithm can be used for heterogeneous robot teams having cameras at different heights. Another factor that plays an important role on how the scenery is perceived is the tilt angle of the camera. In a heterogeneous team of robots some robots may have cameras

tilted to a different angle than the rest of the team, or a single robot may capture images by tilting its camera to get better angles of perception during the course of its navigation. However, as the tilt angle between two cameras differs, the locations of the epipoles change. Hence, we introduce another layer to our control strategy to make navigation possible between images taken by cameras with different tilt angles. The intuition behind this extension is that we want to know how the image would look like if the actual camera position had the same tilt angle as the desired camera position (or viceversa). In that case, the estimated image could be used as the target image and the navigation algorithm could be applied without much change since the tilt angles of both cameras would be the same. In order to create this image we first back-project each image into normalized image coordinates from pixel coordinates, then virtually rotate the desired camera to the same tilt angle of the actual camera, and finally project the image again to pixel coordinates as observed from the rotated camera. The idea is explained in more detail below as a 3-step procedure. Since we are only interested in the extracted features, we slightly abuse the terminology and refer to the union of the extracted features as the image. Let I_t be the target image defined as $I_t = \bigcup p_i$ where p_i is an extracted feature from the image with coordinates $[u_i \ v_i]^T$ in the image plane. We recall that the intrinsic parameter matrix K can be decomposed as $K = K_s K_f$ as shown in Equation 3.11. For each feature p_i in the target image we apply the following transformations:

1. The feature $p_i = [u_i \ v_i]^T$ is first transformed into $p_{pix} = [u_i \ v_i \ 1]^T$ defined in homogeneous pixel coordinates. Then, we back-project it to normalized image coordinates by using K_s

$$p_{img} = K_s^{-1} \, p_{pix}. \tag{3.17}$$

   After the feature is transformed into normalized image plane, the z coordinate is set to the focal length of the camera

$$p'_{img} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f \end{bmatrix} p_{img}. \tag{3.18}$$

2. Next, in order to simulate the perception from the virtually rotated camera, we rotate the point around the camera center with R_ψ where R is the rotation matrix defined around the camera's x axis and ψ is the tilt angle between the two cameras

$$p_{rot} = R_\psi \, p'_{img}. \tag{3.19}$$

At the end of the this step we achieve the normalized image coordinates of the feature as it would be perceived from the target camera if it had the same tilt angle of the actual camera.

3. In the last step we project the feature onto the CCD of our virtual camera and get the pixel coordinates we look for. The projected feature should be normalized in order to get the correct pixel values

$$p'_{pix} = K_s \, K_f \, p_{rot}. \tag{3.20}$$

The tilt-correction process is summarized in Figure 3.6. Once all features are transformed into their new pixel coordinates, the resulting image can be used as the target image and the navigation algorithm can be applied.

3.3 Results

3.3.1 Simulation

The four step navigation algorithm described in the previous section is first simulated in MATLAB. The environment is modeled by random 3D points posing as actual correspondences of visual features. The desired camera configuration is also randomly chosen with the constraint that enforces a minimum number of visible features. Among the places from which the desired camera configuration is in the field of view, the actual camera configuration is randomly selected in a way that at least 50% of the features are seen by both cameras. Both cameras take 320×240 pixel images and their configurations vary in terms of 3D coordinates and pan and tilt angles.

The 3D scene is projected into virtual image planes and the feature correspondence problem is assumed to be solved. Image coordinates of the features are projected to zero tilt angle configuration for both cameras using the projection algorithm introduced in Section 3.2.2. The resulting feature sets are used to compute the fundamental matrix and the epipoles. The necessary inputs are calculated by the algorithm described in Section 3.2.1. A sample run representing the overall error behaviors is shown in Figures 3.7 and 3.8. In this run two cameras share 43 feature correspondences and the target camera configuration differs from the actual camera configuration by 30° tilt and 12.5° pan angle. As can be seen in Figure 3.7, in the first two steps of the algorithm both epipoles converge to zero. Then, as the robot moves along the baseline, the pixel error defined as the maximum distance between

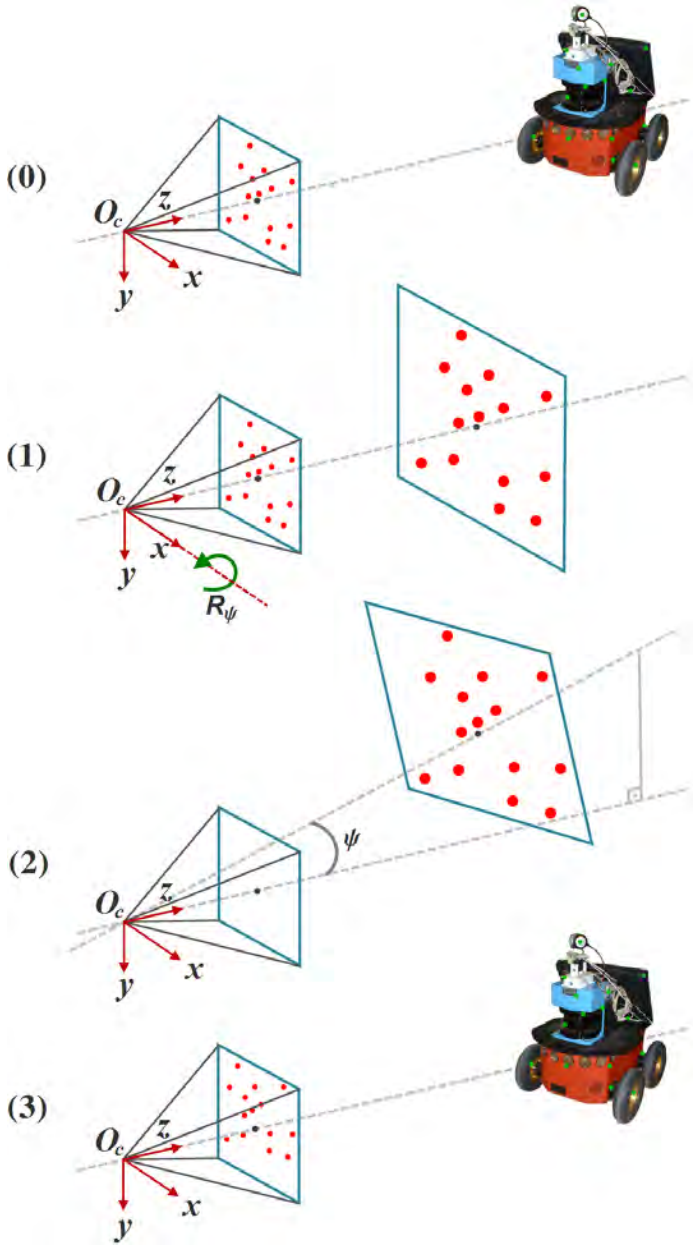


Figure 3.6: The first row represents the feature extraction process. In the first step feature points are back-projected and p_{img} is computed. Then, points are rotated by ψ around x axis of the camera as illustrated in the second step and projected back to achieve the desired effect.

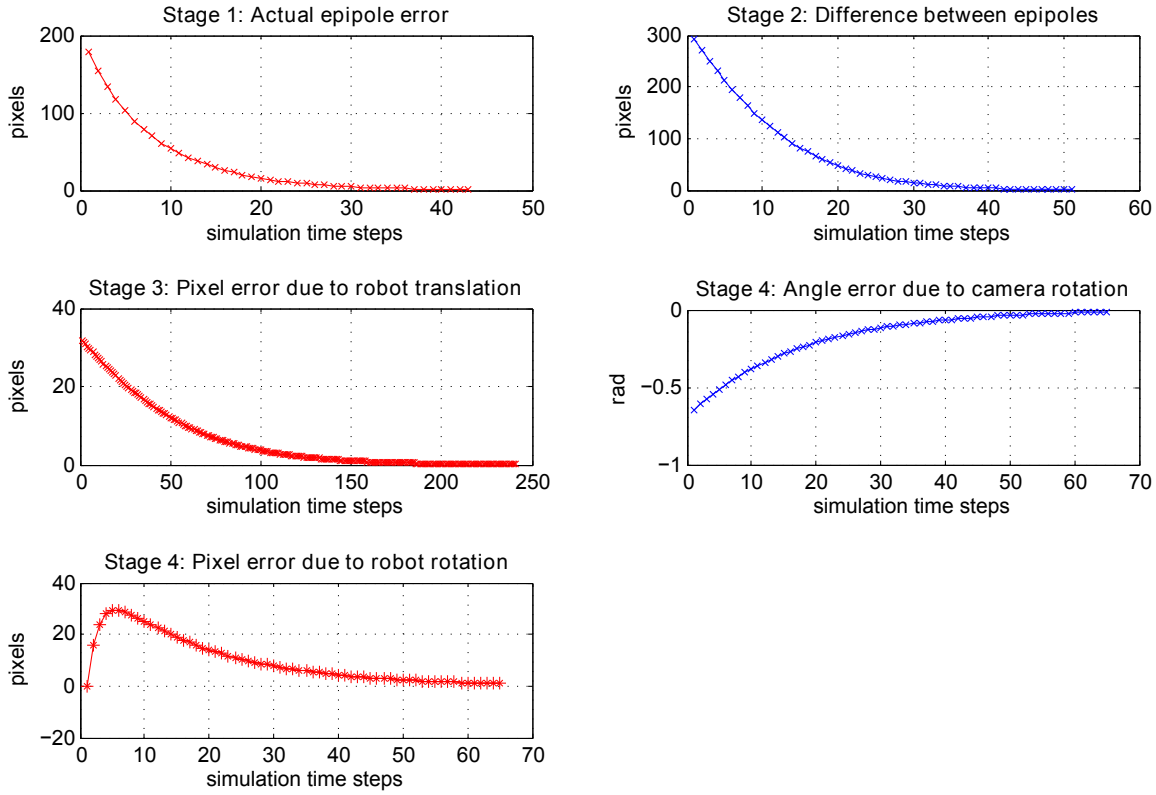
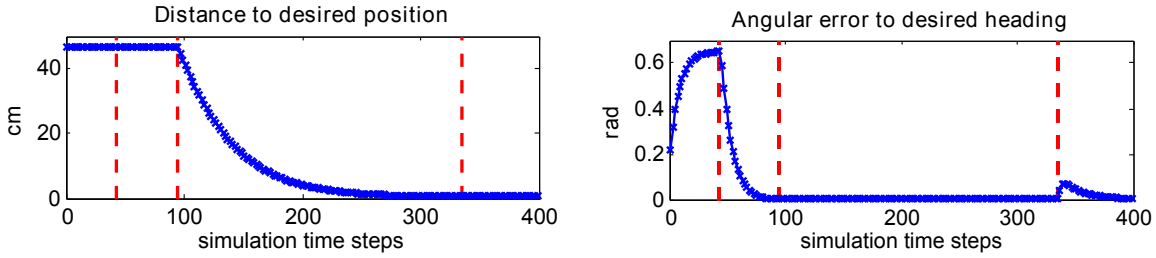


Figure 3.7: Simulation results of a random run shows the error profiles for each stage of the navigation algorithm.

feature matches in pixel coordinates also zeros out. In the fourth step both the robot and the camera rotate to minimize different error functions. It can be noted that in the sub-figure showing the pixel error due to robot rotation, the error first increases and then converges to zero. The reason for this overshoot is the fact that k_4 and k_5 setting the reaction speed for the robot and the camera, respectively, are not tuned perfectly. However, in none of our simulations the error reached to a critical value causing to lose the overlap between the target and perceived images. With a faster controller cycle and better parameter tuning, this overshoot can be minimized.

The overall position and angular errors between the actual and desired camera configurations are plotted during four-steps of the navigation algorithm in Figure 3.8. Since the robot's translational speed stays at zero during the first two steps, the distance error does not change. The error then zeros out in the third step when the robot moves along the baseline. On the other hand, angular error reaches its desired value at the end of second step.



(a) (b)

Figure 3.8: Each dotted vertical line corresponds to a transitions to the next stage. a) Distance error profile through four stages of navigation algorithm is plotted. b) Angular error between the actual and desired camera headings is shown.

3.3.2 Implementation on a Multi-Robot System

The proposed system has been implemented on a multi-robot system consisting of two platforms shown in Figure 3.9. The first one is the P3AT platform produced by Activemedia. The other is the iRobot Create robotic platform designed as an in-house autonomous robot. Onboard the Create we have installed an eBox 3854 compact PC. The platform is equipped with an 800MHz fanless VIA EDEN-N processor, a 4GB flash drive, 256Mb of RAM, and an 802.11 wireless board. The PC runs a standard Linux distribution, and is directly powered from the robot battery. On the other hand, P3AT robot is commanded by a standard laptop equipped with an 1.8GHz CPU and 512MB of RAM. Both robots are equipped with a Philips webcam operating at a resolution of 320×240 pixels mounted on a Phidget servo providing the needed additional degree of freedom for rotating the camera. It is evident that the two robots have a very different morphology and cameras are placed at different elevations and different tilt angles. The robustness of the visual navigation algorithm has been demonstrated in a multi-robot framework in an office environment.

Initially each robot is driven through an arbitrary trajectory, and a set of images are automatically captured by the robots' monocular cameras. A random image is selected among collected images and used as the target view. Each robot is then placed on an arbitrary location close the place where the target view is captured and takes an image from this location which is treated as the initial image. Given these initial and target images, the visual navigation algorithm generates the control inputs and steers the robot to the target location. This procedure is repeated a total of 40 times and a representative set of results of this experiment is shown in Figures 3.10 to 3.13. The top row corresponds to the initial image whereas the target image is

shown in the middle row. The bottom row presents the final image that the robot captured when the navigation algorithm declared that the target location is reached. Figures 3.10 and 3.11 refer to the first part of the experiment where the target image is captured by the same robot executing the navigation algorithm, P3AT and Create respectively. Even though most of the trials resulted in remarkable results, some of these trials converged to non-optimal solutions as can be seen in second column of Figure 3.10 and first column of Figure 3.11. This behavior was not observed in the simulation results and is caused by the false feature correspondences computed by the feature matching algorithm which will be described in Section 4.4. Due to these false matches, the fundamental matrix sometimes could not be calculated correctly even though RANSAC algorithm is used to identify and discard these outliers as previously described in Algorithm 1. Hence, the epipoles computed as the right and left null vectors of F carried some error which resulted in providing the robot with the wrong control inputs.

Figure 3.9: The two robots used to test the navigation algorithm in a heterogeneous framework. On the left: an iRobot Create platform; on the right: a P3AT robot.

In the second part of the experiment we tested the capabilities of the navigation algorithm for the case where the target image is captured by a different robot than the one using it for navigation. In Figure 3.13 P3AT is navigating to a target image captured by the Create robot while Figure 3.12 presents the results of the opposite setup, i.e., Create navigating towards an image taken by P3AT. With its tilt angle

Figure 3.10: Results of the visual navigation algorithm tested on the Create robot. Top row: Initial image taken before starting to navigate. Middle row: Target image captured in an independent run by the Create robot. Bottom row: Final image taken after the navigation algorithm declares that target view is reached.

correction mechanism summarized in Figure 3.6 the navigation algorithm minimizes the error defined in the horizontal component of the epipoles which accounts for tilt and elevation differences, respectively. However, due to the difference in elevation and tilt angle of cameras, it is not always possible to reach the same perspective of the target image. Some of these cases where the final image is not as close to the target view as the rest of the results can be seen in Figure 3.12, especially in the trial shown in the second column. In order to perceive the boxes on the right bottom corner of the target image as the same size, Create needs to come closer to those boxes. However, since the Create robot is much closer to the ground than the P3AT, moving towards them would put those boxes out of the field of view of the camera.

3.4 Conclusions

We presented a novel visual servoing algorithm based on epipolar geometry extracted from images perceived by the robot. Focusing on mobile robots equipped with monocular cameras, the algorithm solves the navigation problem in four steps through which it steers a robot towards a target image using the shortest possible path between its current and target pose. This novel algorithm by design supports

Figure 3.11: Results of the visual navigation algorithm tested on the P3AT robot. Top row: Initial image taken before starting to navigate. Middle row: Target image captured in an independent run by the P3AT robot. Bottom row: Final image taken after the navigation algorithm declares that target view is reached.

heterogeneity with respect to the pose and configuration of the camera. In other words, a robot can navigate to an image captured by a different robot even if two robots have a very different morphology and cameras are placed at different elevations and different tilt angles. This important feature of the proposed method extends the utility of the algorithm beyond a single robot visual navigation framework and allows it to be used by heterogeneous multi-robot systems. The proposed approach has been implemented and validated both in simulation and on a team of two robots, effectively demonstrating the power of the proposed system. The idea of servoing to an image captured by another robot can be generalized to using a visual map created by another robot for navigation which we will focus on in the next section.

The proposed system can be extended in various directions. Firstly, on the practical side, the tradeoff between speed of execution and precision should be investigated to evaluate the effects of moving the robots at higher velocities at the cost of occasionally losing localization or missing some of the steps in the navigation strategy. Secondly, the stability of the proposed four-step servoing algorithm should be studied and how to choose the control parameters to guarantee the convergence of designed errors at each step should be explored. Finally, we believe that the last missing ele-

Figure 3.12: The Create robot is navigating towards an image taken by the P3At robot. Top row: Initial image taken by Create before starting to navigate. Middle row: Target image captured in an independent run by the P3AT robot. Bottom row: Final image taken by Create after it declares that target view is reached.

ment is the ability to sense if the robot is going to lose the overlap between current and the target images during the first servoing step, i.e., rotating the camera towards the target camera location. In that case the system should switch from its servoing algorithm that is optimal in terms of travel distance to an algorithm that focuses on maintaining the overlap at the expense of increased path length. With the addition of these mission critical features the robustness of the proposed navigation algorithm can be increased substantially.

Figure 3.13: The P3AT robot is navigating towards an image taken by the Create robot. Top row: Initial image taken by P3AT before starting to navigate. Middle row: Target image captured in an independent run by the Create robot. Bottom row: Final image taken by P3AT after it declares that target view is reached.

CHAPTER 4

Appearance-Based Localization and Mapping

The Simultaneous Localization and Mapping (SLAM) algorithms furnish robots with a vital feature that allows them to localize and navigate in unknown environments. One of the biggest problems in SLAM is loop closure, i.e., revisiting a previously mapped location and associating it with another one already visited. When metric maps are used, loop closure usually requires a computationally expensive backpropagation process. Appearance-based maps, on the other hand, provide a simple but powerful solution to this problem. In contrast to metric SLAM techniques, without the need of explicit positioning of the robot, places can be recognized via their appearances. In other words, matching and connecting two parts of the map is as simple as creating an edge in between. Appearance-based maps are also suitable for resource-limited robots which need to solve the SLAM problem without the aid of good sensors like laser range scanners. Since no metric information is required to build or utilize appearance-based maps, no other sensor than a simple camera is needed. Even though the proposed framework can also be used with other type of cameras like omni-directional cameras, here we focus our attention to on monocular cameras to integrate the visual navigation algorithm presented in Chapter 3.

After we give a formal definition of appearance-based maps in Section 4.1 and describe the adopted image representation in Section 4.2, we present the data structures used in our technique in Section 4.3. The proposed appearance-based framework requires an offline training phase which will be presented in the same section. Even though it seems like a drawback of the approach, training is a onetime procedure and is independent of the robot utilizing it or environment being mapped. Appearance-based localization and mapping techniques are discussed in detail in Section 4.4 and 4.5, respectively. The basics of the proposed appearance-based localization and mapping system as well as the prerequisite training process are illustrated as a block diagram in Figure 4.1. Next, in Section 4.6 we demonstrate how appearance-based maps can be used for path planning and visual navigation. Finally, we conclude the chapter after we introduce our task specific quality metric in Section 4.7 which measures the utility of the built appearance-based map with respect to three major robotic tasks: localization, mapping, and navigation.

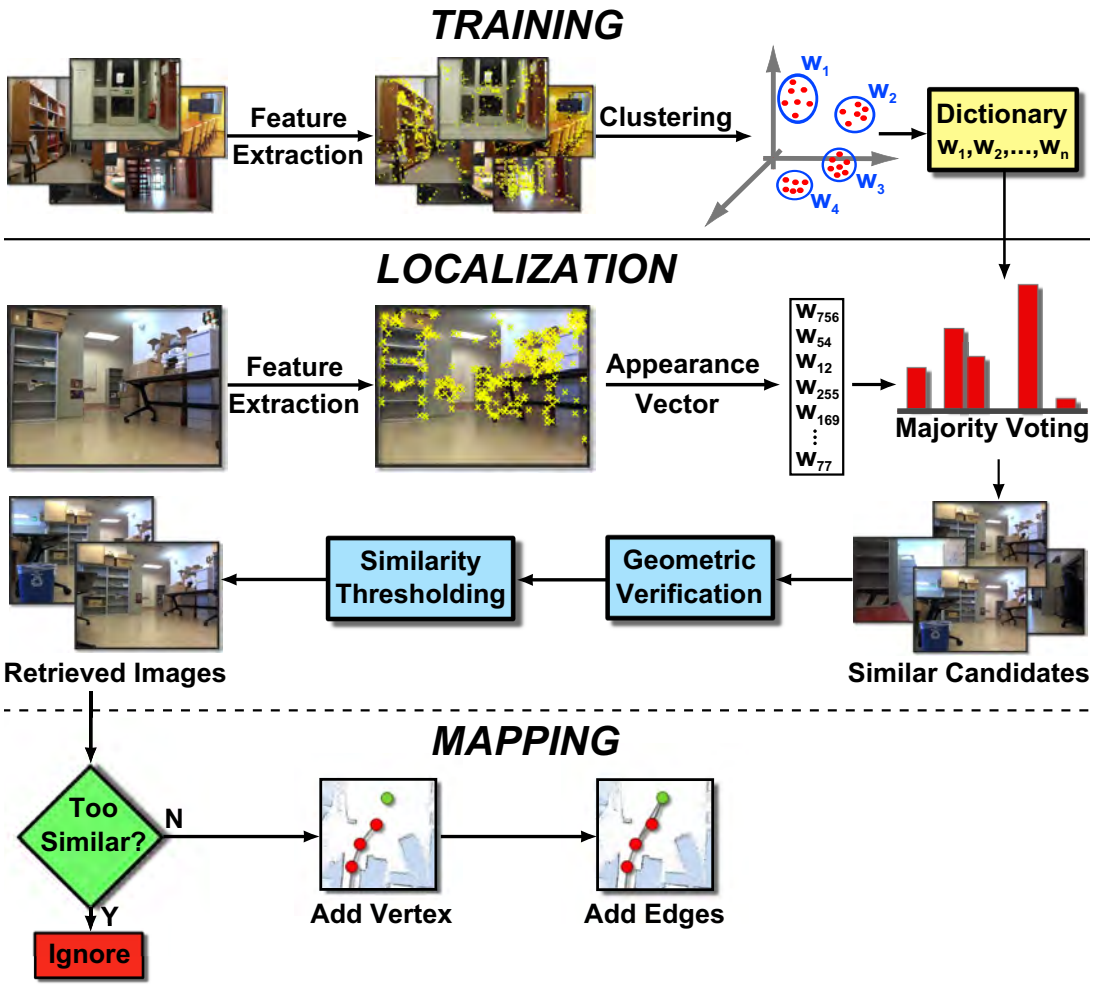


Figure 4.1: Overview of dictionary learning, map building and localization procedures are presented.

4.1 Definition

In formerly published work there exists no unified definition of appearance-based map. The one we embrace here is aligned with most literature, but includes also some ad-hoc features aiming to enable autonomous navigation via image-based visual servoing algorithm described in Section 3.2. We define an appearance-based map as an undirected weighted graph $M = (V, E, w)$ in which every vertex $v \in V$ represents an image acquired by a camera at a certain position in the workspace. In systems using omnidirectional cameras, thanks to the rotational invariance of the

images, the configuration space of the camera is two dimensional, i.e., only x and y coordinates are relevant. For the case of monocular cameras, instead, the orientation of the camera is also important. Therefore, each vertex is intrinsically related to the position and orientation of the camera capturing the image. However, it is important to note that this or any other any metric information is not encoded in the appearance-based map structure. An edge $e_{ij} \in E$ connects two distinct vertices $v_i, v_j \in V$ whenever the associated images are sufficiently similar according to a given similarity metric $S : V \times V \to \mathbb{R}$. Different metrics can be used and edges are therefore parametric with respect to the used metric. In Section 4.4 we will describe the image similarity metric we use in our implementation. An edge is added between v_i and v_j whenever $S(v_i, v_j) > T_{min}$, where T_{min} is a metric dependent threshold. The weight w of an edge is set to the similarity between the vertices it connects, $w(e_{ij}) = S(v_i, v_j)$.

It is worth noting that T_{min} not only plays a role in defining when an edge should be added between two vertices, but it will also influence the behavior of the merging algorithm, as explained in Chapter 5. However, according to our experience, the choice of its value should be mainly driven by the requirement of providing enough features to ensure robust navigation between two images using the visual servoing algorithm we discussed in Chapter 3. The T_{min} value we outlined above reflects the characteristics of the test environment, of the cameras we used, of the resolution of the captured images, of the chosen feature descriptor, and of the robotic platform used to navigate in the environment. Evidently, in a different scenario the user should adjust the value based on preliminary experiments aimed at identifying the sensitivity of T_{min} to the navigation algorithm used.

Figure 4.2 shows an illustrative example of appearance-based maps.

4.2 Image Representation

It is obviously impractical to store raw images to determine similarities between vertices and then ultimately build the graph. Therefore, each image is represented as a set of robust local image features characterizing the scene captured by the image. In the literature there are several feature detector algorithms such as Scale Invariant Feature Transform (SIFT) [105], Harris affine region detector [124], KanadeLucas-Tomasi (KLT) [107], Speeded Up Robust Features (SURF) [10]. Among them we have chosen SIFT descriptors due to their outstanding performance on distinctiveness and robustness against competing algorithms [125, 9]. SIFT descriptors are the collection of feature vectors extracted from an image where each feature is invari-

Figure 4.2: A simple appearance-based map with edges inserted between sufficiently similar images.

ant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion. These descriptors are created as a result of a four stage filtering algorithm. First, all scales and image locations are searched by using a difference-of-Gaussian functions to identify scale-space extrema which correspond to potential interest points. At each candidate location, a detailed model is fit to determine location and scale. After low contrast candidate points and edge response points along edges are discarded, remaining candidates are assigned as keypoints. This step ensures stability of keypoints for matching and recognition. Then, dominant orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations. Finally, the local image gradients' magnitudes and orientations are measured at the selected scale in the region around each keypoint. These are weighted by a Gaussian window and then accumulated into orientation histograms summarizing the contents over subregions. The resulting orientation histograms represent the SIFT descriptors and allow for signicant levels of local shape distortion and change in illumination. In his seminal paper Lowe [105] experimentally shows that orientation histograms of size 128 performs the best. Therefore, in our implementation we employ an open-source library by Hess [71] and use 128 dimensional SIFT features. Further details on construction process of SIFT descriptors and their characteristics can be found in [105]. A sample image with extracted SIFT features is shown in Figure 4.3.

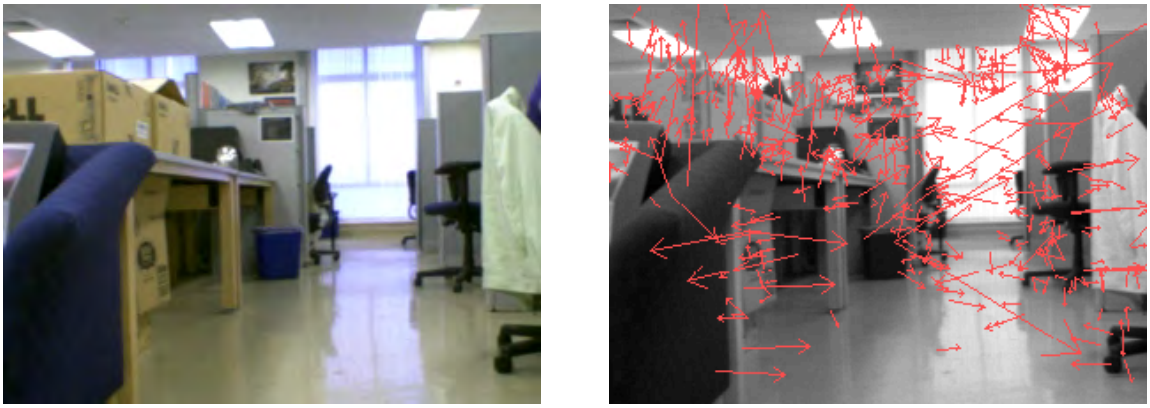


Figure 4.3: Sample image with extracted SIFT features

4.3 Data Structures and Training

Due to its efficient matching process and scalability, we use the image search engine developed by Sivic and Zisserman [153] as the base of our Bag-of-Words (BoW) method. An image in BoW approach is modeled as a document with some words. After the feature extraction process described in the previous section, each image is already converted into a collection of SIFT features. However, using these features directly as words has several disadvantages like the increased size of the dictionary, decreased robustness due to the inclusion of noise, and increased redundancy. Therefore, they are quantized into visual words. Quantization is obtained by k-means clustering performed on the descriptors from a number of training images. Focusing on indoor environments, we extracted over 20 million SIFT features from random indoor images obtained from online image repositories [69, 108, 138, 139, 4]. Some examples of these images can be seen in Figure 4.4. After various tests we determined that 50K words (clusters) offer a good compromise between speed and accuracy, so we fixed the number of clusters to this value. The centers of the learned clusters are then defined as the *dictionary*, also known as the codebook, where the number of clusters corresponds to the size of the dictionary. This computationally expensive offline training procedure is only applied once and the resulting static dictionary is preloaded to all robots performing appearance-based localization and mapping. Robots can then encode their visual perception with words from the same dictionary. This aspect plays an important role in the map merging algorithm, as we will describe in the next chapter. It is also worth noting that no images from the environment used to test our algorithm were included in the training set. Despite this, robots preloaded with this static dictionary perform mapping and localization tasks

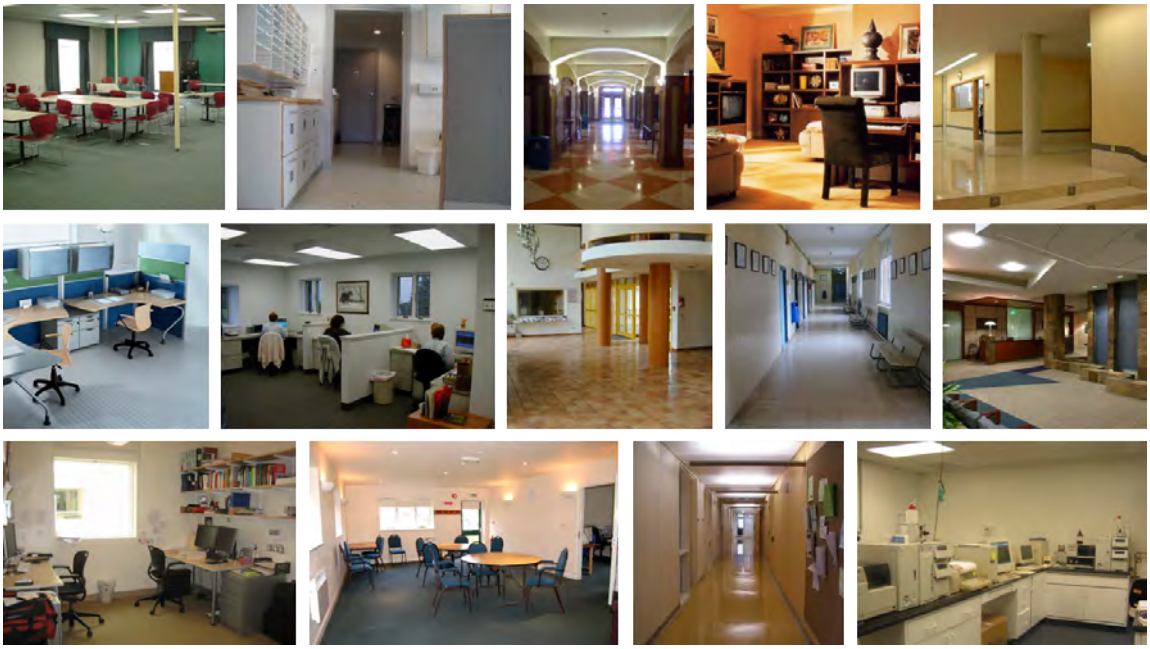


Figure 4.4: A representative set of random training images collected from online repositories

successfully, as we will discuss in Section 4.6. With the idea that words that are too common among the training images will not help differentiating query images, the top 5% of the most common words is discarded. The dictionary includes also an *inverted index* that is incrementally updated when new images are added to the map. The inverted index associates every word to the list of images in which the word is seen, together with the x, y coordinates in the image plane.

4.4 Localization in Appearance-Based Maps

Given an appearance-based map $M = (V, E, w)$ and a query image I_q, we want to find the most similar image in V. This step is needed for both map building and map merging. The algorithm should either return the most similar image, or determine that no sufficiently similar image exists in the map. Localization proceeds as follows. Starting from I_q we extract the set of SIFT features F_q. For every feature $f_i^q \in F_q$, the closest word in the dictionary is determined. Given that we opted for a SIFT implementation, matching is performed using the L_2 norm in $\mathbb{R}^{128}$. Kd-trees do not provide speedup over exhaustive search for spaces with 10 or more dimensions for

exact solutions, therefore striving for real time performance we use an approximate nearest neighbor solution [130]. After every feature has been matched to the closest word, a sparse *appearance vector* ψ_q is built. The appearance vector ψ_q records every word in the dictionary matched to the features extracted from I_q. Then, by using ψ_q and the inverted index each word appeared in the query image casts a vote for all images in the map that also contain this word. When normalized, the histogram of accumulated votes defines a probability distribution over all images in the map. As a result of this voting procedure, all images sharing one or more features with I_q are identified as candidate matches. The number of votes indicating the number of common features defines the formerly introduced similarity metric S.

One of disadvantages of BoW approach is that it ignores the spatial relationships among the visual words, which is very important in image representations. Moreover, the number of feature matches computed through the voting scheme can be affected by outliers due to quantization effects in word clustering and to the approximation in finding the nearest neighbor. Therefore, as final step a robust estimation of the multi-view geometry that links these images is performed by utilizing the RANSAC based algorithm presented in Algorithm 1. For each candidate image, the best fitting fundamental matrix is computed and matching feature pairs supporting this matrix are identified. These supporting feature matches are also tested for spatial consistency as described in [153]. Based on the idea that matching regions in compared images should have a similar spatial agreement, the algorithm eliminates feature matches that do not comply with the spatial layout of the neighboring matches in the query and target images. If the number of remaining feature matches still exceeds a threshold T_{min}, the image is considered as a match. We set this threshold to 15, i.e., the minimum number of matches required to robustly navigate between two images using the navigation algorithm we presented in Section 3.2. Among all candidate images that pass this geometric verification test, the image with the largest number of matches is chosen to be the most similar and considered as the most likely location in the map. If there exist no image candidate with at least T_{min} matches, the algorithm terminates indicating no similar image was found. The overall localization procedure is described in Figure 4.1.

The described majority voting schema is compared with the common pairwise image matching method [16] and the results are shown in Figure 4.5. The pairwise method, instead of using a centralized dictionary or a database of features, extracts features from each image and compares them with the features from every other image in the appearance-based map. In order to reduce the computational cost of matching, a kd-tree is constructed from the features of each image. In this performance comparison analysis, appearance-based maps with different number of images

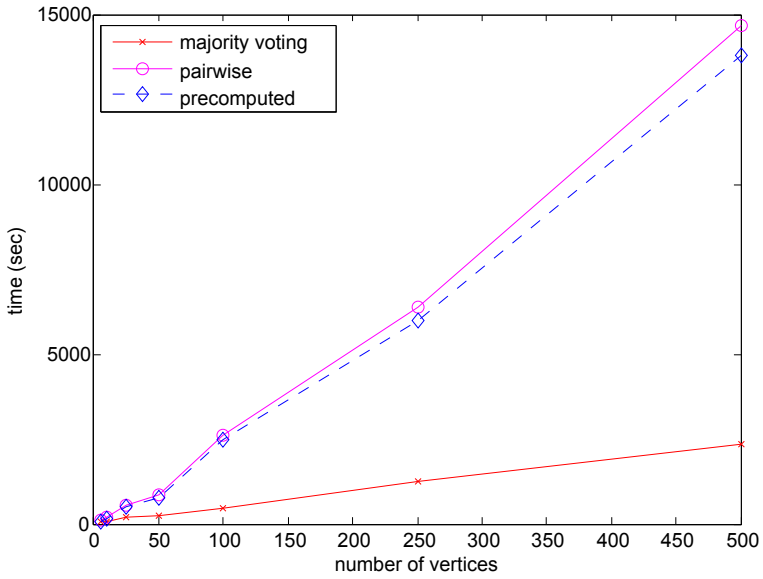


Figure 4.5: Appearance-based graphs constructed for different number of images are used to run 500 localizations to random query images. The graph shows the overall time spent by the presented majority voting schema and two pairwise image comparison methods, *pairwise* where for each localization instance kd-trees for all images are constructed from scratch and *precomputed* where kd-trees are precomputed and used throughout all localization calls.

are generated and a random query image is localized in the resulting map. The plots in Figure 4.5 correspond to the overall time to compute 500 localizations which includes the kd-tree construction time for pairwise matching approach. Additionally, we also show the overall time for pairwise method in case where kd-trees are precomputed. As can be seen in the figure, the difference between storing the kd-trees in advance or re-constructing them for each localization attempt takes only a small percentage of the overall time required. In other words, the performance of the pairwise approach suffers more from the localization step of high number of image-image comparisons. The figure also concludes that the majority voting approach outperforms pairwise matching methods.

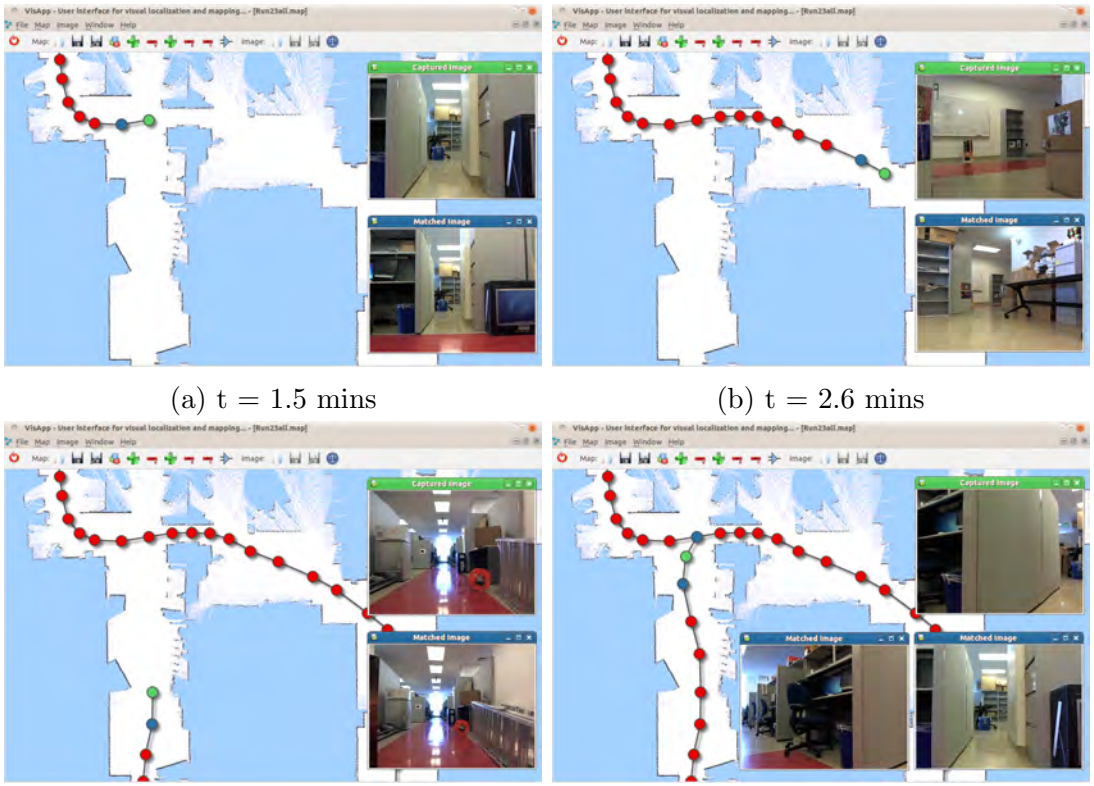
4.5 Appearance-Based Map Building

Mapping is the process of iteratively acquiring new images and updating the appearance-based map, if needed. Given a new image I_n, the process starts by running the formerly described localization algorithm. Localization returns either a set of similar images, or an empty set if no sufficiently similar image is found in the map. If no image is returned, then I_n is inserted into the map because the robot has discovered a new location that has no connection with any previous image. If localization returns a set of similar images, then image similarity is used to determine whether I_n should be inserted. If I_s is one of the returned images and $S(I_n, I_s) > T_{max}$, then I_s is considered too similar to I_n and it will not be inserted because it is not sufficiently informative. Otherwise, the image is added to the map and edges are added between the new vertex and all similar images identified by the localization algorithm. By rejecting the insertion of similar images into the map, we prevent the map from growing too much (in terms of images) when the robot revisits the same area numerous times. Figure 4.6 shows some snapshots of the mapping process along with the Graphical User Interface (GUI) we designed. The system can build appearance-based maps in real-time. Using a non-optimized C++ implementation it takes around 0.6 seconds to process a single image of size 320×240, including image capturing, feature extraction, global localization, and map update.

4.6 Planning and Navigation on Appearance-Based Maps

Appearance-based maps with their underlying weighted graph structure provide a suitable framework for most path planning algorithms such as depth-first search, Dijsktra, and A*. Given the current image and a goal image, the planning algorithm returns a series of vertices, when followed, guides the robots from its current vertex to the goal vertex. In this section we focus on the utility of appearance-based maps from path planning and navigation perspective and present our path planning algorithm supported with previously introduced visual navigation method in Algorithm 2.

Given a goal image I_g the algorithms first searches that image in the appearance-based map using the previously described localization procedure (line 3). If it fails in locating the goal image in the map, the image is inserted into the map as a new vertex and that vertex is set as the target location (line 5). In order to determine its current position with respect to the appearance-based map, the robot localizes itself in the map by searching for the vertex with highest similarity to its actual view I_a (line 4). Similarly, if there exist no image that is sufficiently similar to the actual



(a) t = 1.5 mins (b) t = 2.6 mins

(c) t = 8.3 mins (d) t = 9.2 mins

Figure 4.6: Some snapshots taken while the robot builds an appearance-based map using the BoW method are presented. The last captured image is displayed at the top right corner of the GUI while matched images are shown at the bottom right corner. Vertices corresponding to query and matched images are shown in green and blue, respectively. Note that the occupancy grid map overlaid with images is shown for display purposes only and not used by the robot.

image, the actual image is added to the map as a new vertex (line 6). If either the goal image or the actual image is captured at a location not covered by map and hence fails to create any edges with the vertices in the map due to non-sufficient similarity, the path planning algorithm terminates by declaring that the map does not contain a valid path between these two images (line 8). Otherwise, once the vertices corresponding to both actual and goal images are identified in the map, the optimal path connecting these vertices is determined using Dijsktra's algorithm (line 10). Since Dijsktra's algorithm minimizes the cost of the path, edges in the graph should be labeled with a distance measure. The distance between vertices

is defined as dissimilarity between images associated with them where non-existing edges indicate infinite dissimilarity. The distance is calculated as reciprocal of the edge weight which represents the similarity, $\frac{1}{w_{ij}}$. Due to the dissimilarity-based cost function, the computed path favors navigating through images with high number of feature matches and avoids places where features change rapidly. It is important to note that the computed path is not necessarily the shortest path in Euclidean space since the planning takes place in the appearance space. If during the planning process the overall distance or the time it takes to navigate the path needs to be optimized, the weights need to be assigned accordingly.

After the path is computed as a sequence of vertices (line 10), the planner assigns the next vertex in the path as the waypoint and executes the visual navigation algorithm that steers the robot to that vertex (lines 11-16). In other words, the robot advances in the path one vertex at a time until it reaches the goal vertex.

Algorithm 2 Navigate(I_a,I_g)

```
1:  F_g ← extractFeatures(I_g)
2:  F_a ← extractFeatures(I_a)
3:  v_g ← localize(F_g)
4:  v_a ← localize(F_a)
5:  if v_g = NULL then v_g ← insertToMap(F_g)
6:  if v_a = NULL then v_a ← insertToMap(F_a)
7:  if v_g = NULL or v_a = NULL then
8:     return FAILURE
9:  else
10:    P ← Dijkstra(v_a,v_g)
11:    v_i ← popFront(P)
12:    repeat
13:       v_j ← popFront(P)
14:       visualNavigate(v_i, v_j)
15:       i ← j
16:    until P not empty
17:    return SUCCESS
18: end if
```

The proposed algorithm is tested in an indoor scenario using the same team of robots with different morphologies presented in Section 3.3. Each robot is teleoperated through an arbitrary trajectory, and collected a sequence of images. For each set of images an appearance-based map is built using the map building algorithm

described in the previous section. Each robot is then placed on an arbitrary location in the vicinity of the area covered during the map building phase so that the initial view of the robot will be similar to at least one of vertices in the map. The target view, on the other hand, is assigned by randomly choosing an image from the map. Given these initial and target images, the planner algorithm returns the sequence of images each robot needs to follow in order to reach the target location. The resulting path is followed by navigating from one image to the next using the visual navigation algorithm presented in Section 3.2. However, due to the noise in feature coordinate estimations and outliers in feature matches, the computed actions to steer the robot between images are subject to error and the computed epipoles are not as smooth as observed in simulation. Thus, by applying error-prone actions computed based on the noisy data, the robot might get lost, i.e., the number of common features between the current view and any image in the map falls below the minimum similarity threshold T_{min}. Whenever this phenomenon happens, the robot first assumes the failure is due to the delay in communication between the controller and the actuators, and scans its local environment for re-localization by rotating its camera. In the trials with Create, where this type of failure is commonly observed, this method successfully re-localized the robot in the graph. On the contrary, the P3AT robot never lost its sight in our trials. Only in one occasion where sufficient overlap between the actual and the goal image could not be recovered by the camera rotation, the Create declared itself lost and started a global localization procedure. The robot successfully localized itself in the graph by creating an edge to the next waypoint in the path and reached the target image by navigating through the remaining waypoints.

In these experiments robots not only navigated in their own maps but also in maps created by other morphologically different robot. Figure 4.7 presents the path followed in a sample run where the traversed distance is roughly 30 meters. The resulting appearance graph consists of 42 vertices. We remind that the path is only for visualization purposes since the proposed method does not use any metric information. Some of the images captured by P3AT and used to build an appearance-based map are shown in the top row of Figure 4.8. More precisely, those are the images associated with the six vertices outlined along the path. The middle row shows the corresponding view obtained by the P3AT robot while navigating to reach in sequence those waypoints. The last row instead shows the final images captured by the iRobot Create when it thinks the corresponding waypoint is reached. As can be seen in the last images of both sequences, despite their perceptual differences both robots successfully reached the target location.

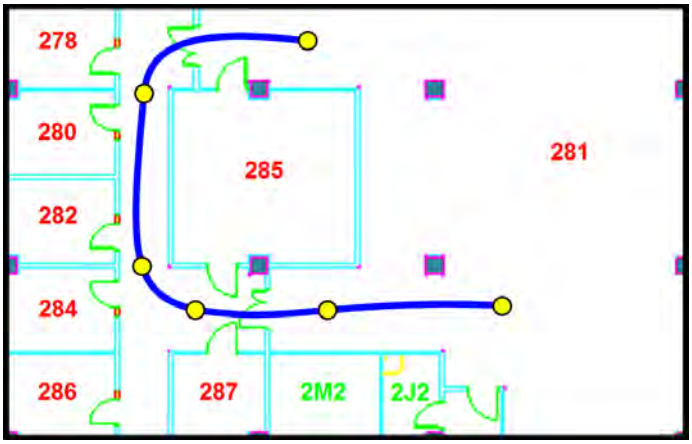


Figure 4.7: The path illustrates the followed trajectory during the map building process. Circles indicate some of the vertices created in the appearance graph and the associated images images are presented in Figure 4.8. Robots start from the bottom vertex on the right and navigate to the top vertex in the middle along the depicted path.

4.7 Quality Assessment of Appearance-Based Maps

In the previous section we presented our appearance-based map building algorithm which consisted of several components such as feature extraction, feature matching, vertex matching, similarity detection, and edge insertion. The map building algorithm is designed to be modular with respect to these independent components. Evidently, one module can be replaced with a completely different one as long as its input-output relationship is kept intact. For instance, one may prefer SURF descriptors over SIFT descriptors due to their computational advantages. In that case even if the same set of images are used to build the map, the features extracted from these images as well as the number of feature matches will be different which will result in a totally different edge set, hence a different map. In order to analytically determine the effectiveness of that module in the map building algorithm, a map can be build for each alternative method by only replacing that specific module, keeping the rest of the building framework the same, and inputting the same image set as illustrated in Figure 4.9. The quality of the resulting maps needs to be evaluated and the module creating a map with higher quality should be chosen. Similarly, by using the same set of images different appearance-based maps can be built by changing only the parameters used in the map build procedure and an evaluation criteria is needed to measure the quality of these maps in order to establish the optimum

Figure 4.8: With reference to Figure 4.7, the top row shows the images collected by P3AT while building the appearance-based map. Then middle row shows the corresponding view of the P3AT robot upon reaching those waypoints, while the bottom row shows the corresponding views for the iRobot Create.

parameter set. Alternatively, this map benchmark can be used to evaluate the maps built by different agents equipped with the same map building algorithm navigating in the same environment as shown in Figure 4.10.

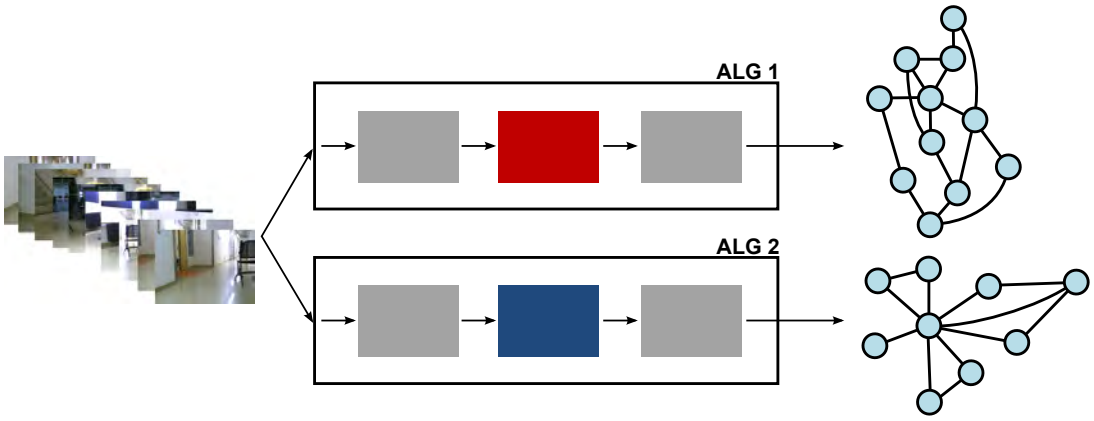


Figure 4.9: The effectiveness of one part of the map building algorithm can be measured by evaluating the resulting maps.

The topic of objective quality evaluation has been one of the most neglected areas in robotics community even though it is generally agreed that robotics researchers are in great need of objective quality measures to evaluate the impact of the count-

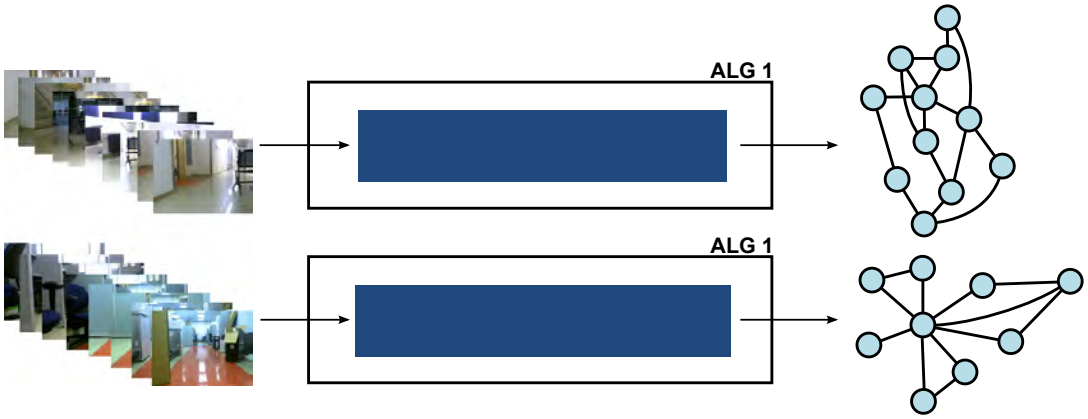


Figure 4.10: Different agents using the same map building algorithm can generated different maps in the same environment and a good map evaluation metric should be able to measure the quality of these maps.

less ideas proposed in this quickly growing field. To date, for most problems it is impossible to compare two different solutions according to widely accepted criteria. The peculiar nature of robotics research also undermines the experimental replication of published results. However, given the recent dramatic changes in information technology, it is nowadays possible and needed to converge towards accepted performance measures, and to disseminate experimental data so that these assessments can be performed and repeated by an arbitrary third party.

The intrinsic purpose of a map evaluation metric is to measure the quality of different maps and determine which map is the best in general or in terms of some specific criteria. As previously mentioned the maps to be evaluated can be generated by different agents using different algorithms. Focusing on the goal of creating a generic appearance-based map benchmark, we present evaluation criteria measuring the quality of a map independently of the algorithm used to build it. Hence, during the evaluation of a map the building blocks of the map creation algorithm are treated as black boxes, and the metric is designed in a way that minimizes the effects of these building blocks. Furthermore, instead of aiming to measure a map's overall goodness, we introduce a set of task-based performance evaluation criteria to measure the usefulness of the map with respect to each individual task and its successful execution.

4.7.1 Evaluation of Visual Maps

The ideal quality-assessment of a map would be performed by comparing it to the ground truth, i.e. the real value of the variable to be estimated. However, in most cases ground truth might not be easily accessible, or it may be time consuming to acquire. This is a well-known problem for any kind of map. Even if it can be measured, since every device is entailed with a non-zero error, only the most accurate estimate can be used as the ground truth. In other words, it is not always possible to acquire a good model of the environment to be used as ground truth. Depending on the type of the map different kind of data have been used for the estimation of the ground truth. For instance, the coordinates of features are used as ground truth in the comparison of feature maps; whereas, occupancy grid maps mostly utilized blue prints or hand-drawn sketches, and more recently post-processed point clouds of laser range finder sensors. Approaches based on appearance graphs, on the other hand, sample 3D environments with 2D images, and it is therefore impractical to generate a map of all possible views serving as ground truth. In fact, the real environment is the only source which can be used as ground truth. Hence, the quality of an appearance graph can be best measured by evaluating how well it captures the desired properties of the environment it models.

A map by definition is a representation of the environment, and therefore, it may be natural to conclude that the map with most resemblance has the utmost quality. However, this point of view skips the basic motivation behind the need of a map, i.e. its utilization for the successful completion of a given task. A robot creates some form of an internal representation of the environment as a tool to successfully achieve its assigned mission. Thus, for the map to be most useful to the robot, it has to offer enough information to complete the assignment. Consider for example a robot performing inside a warehouse whose task is to quickly navigate between target locations revealed during the mission. One cannot rule out the possibility that a robot using a map with good geometric accuracy, but possibly cumbersome, may be outperformed by one using a model with an inferior geometric accuracy but easier to process.

With this motivation, we advocate that the assessment shall not be detached form the task at hand. Despite the fact that this statement may seem fairly trivial, in the context of occupancy grid maps one faces a significant amount of scholar work where maps are most often treated as images, and therefore contrasted and evaluated using algorithms and metrics that have little to do with the ultimate task the maps are needed for. On the contrary, we propose three task-centric evaluation criteria, namely localization, navigation, and planning. The performance of a map in terms of

localization is measured by the amount of information captured from the environment and the accuracy of this information. The planning metric favors instead maps with high connectivity and measures the validity of these connections. The navigation criterion, on the other hand, computes the robustness and stability associated with the paths that a robot will extract from the map. In the following sections we explain each of these metrics in more detail and exemplify how the proposed metric work on a set of appearance based maps. In order to capture a broader spectrum of maps in this comparative analysis, a P3AT robot equipped with a monocular camera is teleoperated in our research lab at UC Merced twice resulting in image sequences from two independent runs. These images are fed as the input to our map creation algorithm described in Section 4.5. The paths followed during these runs and the test runs are shown in Figure 4.11. A total of 1064 test images are collected during these runs. The test image sequence is divided into three subpaths with different colors as shown in Figure 4.11.

(a) Image collection for map building (b) Image collection for testing

Figure 4.11: a) The paths followed by the robot during the map building processes are shown. b) Three paths followed to collect test images are shown.

In order to increase heterogeneity, different parameters are used while building the two maps. In the first run (represented in blue in Figure 4.11), a higher image capturing frequency is used resulting in map M_1 with a larger number of images. Furthermore, comparing to the second map M_2, a more conservative approach is taken in the image retrieval algorithm by increasing the required number of feature

matches for an image in the map to be matched with a query image. As a result two very different appearance graphs are created.

4.7.1.1 Localization

The ability to estimate its own position is one of the very fundamental robot abilities enabling the successful completion of a variety of tasks. Hence, it is very important that an evaluation algorithm measures the quality of a map with respect to its usefulness for localization. Unlike in metric maps, localization in appearance graphs is realized by finding within the map the image most similar to the one perceived by the robot when it needs to localize itself. The robot is declared lost if it cannot localize itself to any image in the map.

The utility of the map with respect to localization is measured by the robot's localization performance using that map. A good metric should assign high utility to maps providing good localization. The goodness of localization in appearance graphs can be defined in two ways: 1) coverage 2) accuracy.

Coverage: The *coverage* metric measures the amount of information in the environment captured by the images in the appearance graph. It is evident that the coverage grows as the number of vertices in the map increases. For a map fixed number of vertices, however, maximum coverage can be achieved by minimizing the perceptual overlap between images. In other words, we prefer maps with vertices spread in the environment as much as possible as shown in Figure 4.12.

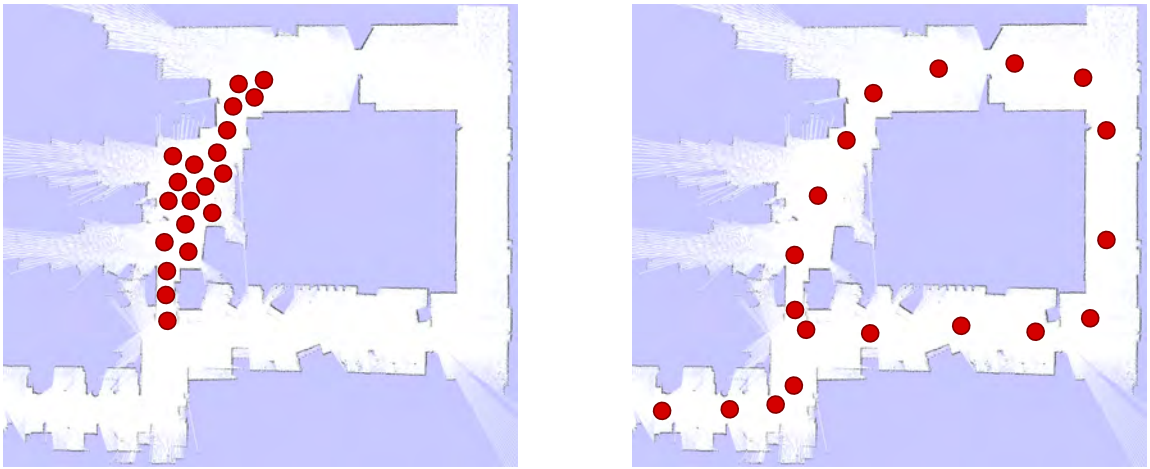


Figure 4.12: Vertices of two appearance-based maps with low (left) and high (right) coverage are overlayed on the occupancy grid map of the environment.

To this end, we propose to use a large set of pictures captured at random locations in the environment where the map is created as inputs to the localization algorithm where each of these images is localized to the most similar location in the appearance-based map. It is important to recall that if the similarity between the query and the most similar image in the map falls below a preset similarity threshold as described in Section 4.4, the localization attempt is declared as unsuccessful. The idea is that if the environment is covered very well by the images in the map, the algorithm should be able to localize successfully most of the query images which then translates into a high coverage score. On the contrary, if there are no images capturing some part of the environment, the query images from that part of the environment cannot be localized to any of the images in the map and coverage score is decreased.

It can be easily concluded that this method of evaluation of a map's merit depends on the performance of the localization algorithm, and more specifically, the image matching function as its core component. However, even the most successful image retrieval algorithms have suboptimal performances. In other words, the image retrieval algorithm used for localization can return an invalid image/feature matching resulting in wrong localization. The erroneous localization ratio, therefore, depends on the performance of the image retrieval algorithm and less on the quality of the map being evaluated. In order to minimize the effect of the image retrieval algorithm on the quality score of the map, false image matches should be identified and discarded for each localization attempt. Since there exist no ground truth data stating which image matches are valid, each image match has to be visually verified by the person evaluating the map. Due to the simplicity of an image match verification task to humans, we believe at this point no subjectivity is introduced into the evaluation. This way false positives are eliminated. In order to also account for false negatives, a human should verify if a query image has actually a corresponding image in the map that the image retrieval algorithm could not locate. This task, however, is not trivial and unlike comparing just two images to detect a false positive, it requires to go through all test images and the images in the map. Hence, we choose the best possible image retrieval algorithm and think it as a black-box that behaves like an independent external source introducing this noise in terms of false negative image matches. Due to the fact that this algorithm affects both of the maps being evaluated, no bias is introduced into quality evaluation and this impractical human verification step can be avoided.

Having set the scene, the coverage of a map is defined as the percentage of successful localizations:

$$l_{cov} = \frac{L_{success}}{L_{total}}. \tag{4.1}$$

Accuracy: Localization accuracy, on the other hand, describes the closeness of the estimated location to the real location of the robot. From the appearance graph perspective the robot will benefit from an accurate localization when the returned and query image look alike. Since similarity between images is encoded by the number of matched features, the accuracy has a positive correlation with the number of correct feature matches. Besides sharing features originating from the same objects in the environment, in similar images these objects should appear in close proximity in terms of image coordinates. In order to capture this idea, the localization accuracy of a query image is defined as the reciprocal of the localization error which is the average distance between corresponding features' image coordinates

$$err_{acc} = \frac{\sum_{i=1}^{N} d\left(a_i, q_i\right)}{N} \tag{4.2}$$

where d is the Euclidean distance function, a and q are corresponding features from the retrieved and query image respectively, and N is the number of feature matches between these two images. Then, the reciprocal of the average accuracy error, err_{acc}, of all queries is assigned as the overall localization accuracy of the map, l_{acc}. Figure 4.13 and 4.14 illustrates examples of high and low accuracy, respectively. As can be seen in these figures, for similar pair of images the distance between matched features in image space is lower than the ones that are less similar.

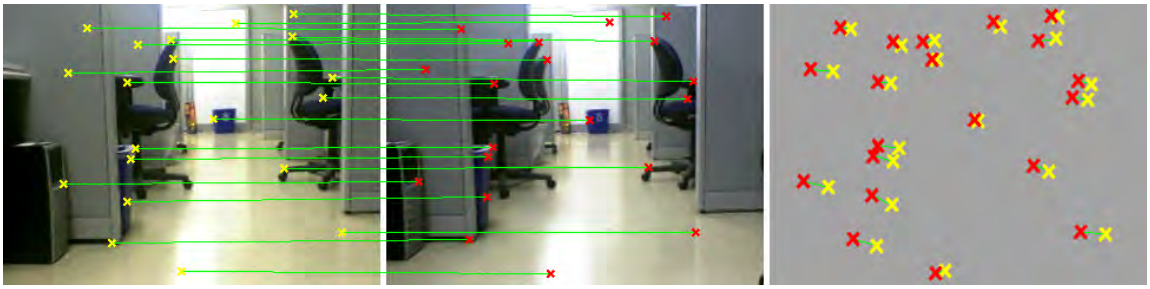


Figure 4.13: A matched image pair with high visual accuracy

In order to compare two maps, M_1 and M_2, with respect to coverage and accuracy, $n = 10$ images are chosen randomly for each test run. Each image is fed as the query image into the image retrieval algorithm and successful localization in the map is declared if the algorithm can match that image within the map and the user visually verifies it. For each successful localization, coverage ratio is increased and the accuracy of the match is calculated. Some examples of successful localizations along with their accuracies are shown in Figure 4.15. This procedure is repeated $m = 10$

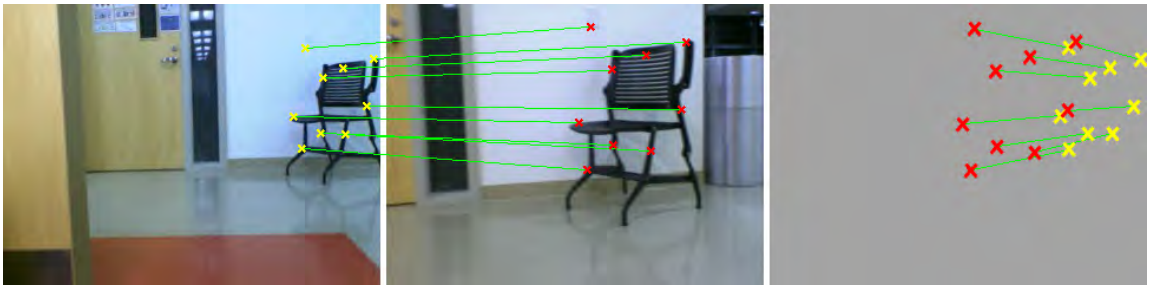


Figure 4.14: A matched image pair with low visual accuracy

times and the average scores are returned as the result of the evaluation of appearance maps in terms of localization task which are presented in Table 4.1. As expected, M_1 with 148 vertices incurred in higher localization accuracy when compared to M_2 with 99 vertices due to the higher probability of containing an image similar to a random image. On the other hand, in terms of localization coverage, M_1 scored almost the same in the first two test runs whereas M_2 scored almost twice as M_1 in the third test run. As it can be seen in Figure 4.11a, the two paths followed to collect images to build maps are almost identical except the small loop in the second map. The images in the third run were collected while the robot was traversing in the middle of the lab following the opposite direction of the map generation runs. Therefore, images pointing the opposite direction did not match to any of the images in M_1, whereas they matched the images captured while the robot was making a full turn.

Table 4.1: Localization quality of 3 sample runs

	Run1		Run2		Run3	
	l_{cov}	l_{acc}	l_{cov}	l_{acc}	l_{cov}	l_{acc}
M_1	0.52	0.037	0.78	0.029	0.22	0.046
M_2	0.50	0.023	0.77	0.030	0.37	0.027

Both the localization accuracy and coverage metrics favor appearance graphs with large number of vertices since the probability of having a similar image for a random query image increases with the number of images in the map. On the other hand, if the number of vertices in the map are limited due to the amount of data that can be stored, then a map consisting of images taken all around the environment will be preferred since it will reduce the number of times the robot gets lost. However, when the robot localizes itself, the localization will have less accuracy for a random query image since the images are spread around and perceptual overlap is minimal. Thus,

Figure 4.15: Sample image matches are shown. The top row contains query images whereas the retrieved images are shown in the bottom row. Localization accuracy computed for these matches are: 0.052, 0.020, 0.014 (left-right)

for a fixed number of vertices allowed in the map, there is a balance between accuracy and coverage. It is also important to note that neither accuracy nor coverage metrics consider the edges in the graph, but they both focus exclusively on the amount of information captured by the images encoded in the vertices.

4.7.1.2 Planning

In order to reach full autonomy, robots need to choose their actions by themselves. Planning can be realized using the so-called state space which provides an abstraction of the overall system. Similarly, an appearance graph with vertices corresponding to states and edges corresponding to actions can be used for path planning, i.e., finding the shortest path in the appearance space between two nodes in the graph. It is important to recall that the resulting path will not be the shortest path in Euclidean space, but instead in appearance space. In the literature there are several search algorithms which work directly on undirected weighted graphs such as Dijkstra's algorithm [36]. The time complexity of any planning algorithm working on graphs will increase with the number of vertices and edges. Hence, from the perspective of planning efficiency, maps with less vertices and edges will be preferred. Furthermore,

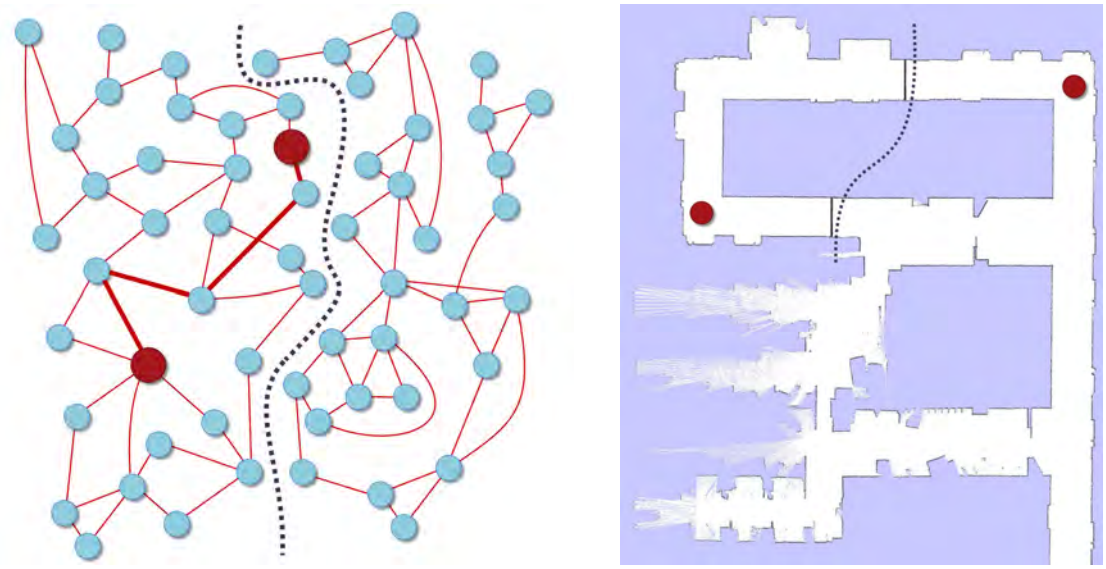
the number of extracted features are correlated with the number of vertices and have a direct effect on the running time of the planner since most of the image matching algorithms utilize nearest neighbor search in the high dimensional feature space. Hence, it can be concluded that the amount of data required to store the map is an indication of its performance in planning tasks, and data size comparison should be a part of the general map evaluation process. Table 4.2 summarizes the elements of the maps we are comparing where $\|V\|$,$\|E\|$, and $\|D\|$ are the number of vertices, edges, and features, respectively and the amount of data in megabytes required to store them into the disk. As mentioned in the previous section, the first map benefits from having more vertices and obtained a higher score in localization accuracy. Theoretically, a map with infinite number of images should have the utmost accuracy. On the other hand, this map gaining high scores from one side loses performance in planning tasks due to high number of vertices and associated features.

Table 4.2: Amount of data captured within maps

	$\|V\|$	$\|E\|$	$\|D\|$	dataSize(MB)
M_1	148	1396	24487	2.43
M_2	99	408	15824	1.56

Nevertheless, the reduction in the number of vertices and edges may cause the planning algorithm to compute a suboptimal solution while also negatively affecting localization and navigation. One solution for this problem is to define additional layers of abstractions on the appearance graph by adding different hierarchies in which meta vertices consist of partial graphs from lower layers. This way the planning can take place in higher levels with less vertices and edges and the solutions can be followed to lower layers during the implementation stage. However, here set our focus on single layer appearance graphs.

From a connectivity point of view for planning tasks, it is important that a map is well connected and contains a good representation of passages between places. Such a map will have a higher utilization rate than a map which reflects an accurate and detailed representation of the environment in general, but also includes an edge between two images of two places that are in fact physically separated. Hence, a metric to measure the usefulness of a map with respect to path planning should consider the ability to plan paths that are valid in the real environment. Inspired by the work of Collins et al. [31], we would like to measure the validity of the paths in a map by comparing paths generated within both the appearance graph and the actual environment.



(a) Appearance-based map (b) Grid map of the actual environment

Figure 4.16: The path connecting two images of interest denoted with red circles is shown in the appearance-based map (a) while the same images are shown in the occupancy grid map of the actual environment (b). The dotted line shows the separation of two physically disconnected regions of the environment. This figure an example of a valid path in the appearance-based map which, however, is indeed invalid in the actual environment, i.e., connecting two physically disconnected places.

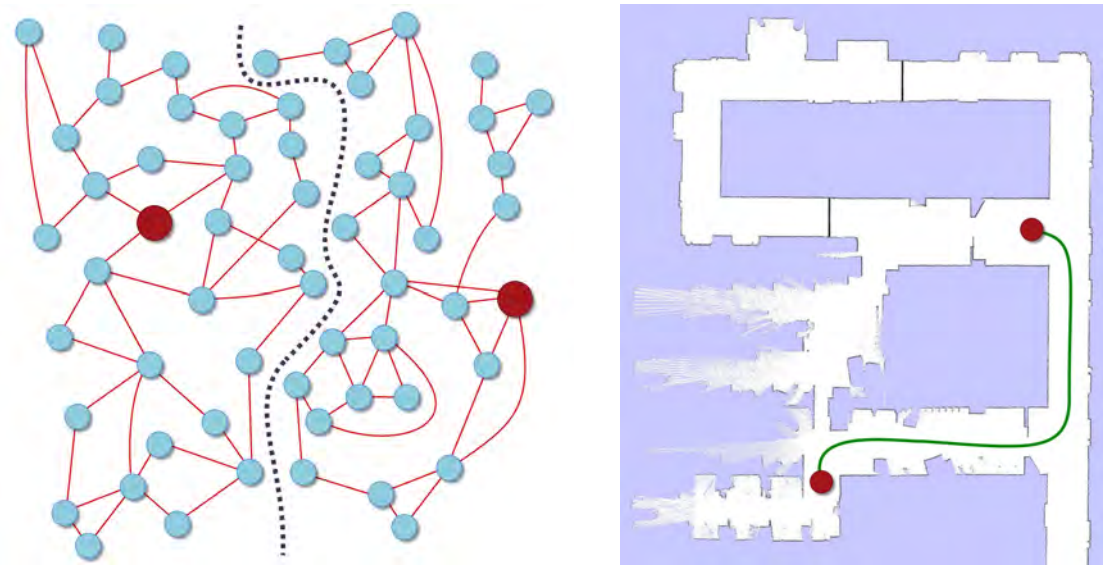
In this test, representing the *exactness* of the map, the ratio of paths that are valid in the map, but invalid in the real environment will be calculated. An example of such a case is illustrated in Figure 4.16. To measure the ratio of these paths, also known as *false positives*, two connected vertices are randomly selected from the appearance-based graph. Then, it is tested whether two places identified from these images are actually connected in the real environment. The physical connectivity assessment can be performed by teleoperating the robot which constructed the map from one place to another. If, however, the robot cannot be teleoperated due the limited physical access to the environment, this verification step can be simulated using the kinematic equations of the robot and the blue prints or the occupancy grid map of the environment. Visual inspection emerges as a good alternative in case none of these options are possible. Even though one can argue that the visual inspection introduces some subjectivity into the metric, for robots with well-known kinematics and structured environments it is expected that it will provide an accurate

approximation. A generated path failing this test is identified as a false positive. The false positive ratio is then estimated by repeatedly generating random paths in the appearance-based map and counting the ones that fail the connectivity test. The problem is that in order to detect a false positive path the map should contain images captured in disconnected parts of the environment. However, a map generated from a single run will not have such images since the robot will only be able to travel between physically connected places. Hence, there will be no false positives for such maps.

The value of this metric becomes apparent only when evaluating maps that are the results of a multi-robot map merging process. In a scenario where robots create local maps of different floors of a building, they can exchange information wirelessly without the necessity to share the same physical environment. The merged map, on the other hand, may contain links between similar images, even though they might belong to physically disconnected parts of the environment. This metric, measuring the ratio of such paths, i.e. false positives, can be used to evaluate the exactness of these merged maps from the planning point of view.

Alternatively, as proposed in [31] to measure the quality of metric maps, a map can also be evaluated in terms of *completeness* by the ratio of the paths that actually exist in the environment, but are not captured in the map, i.e., *false negatives* as shown in Figure 4.17. Borrowing the same idea that we used to compute false positive paths, two images from the map are randomly selected and a graph search algorithm is used to find a path connecting these two images. If this path cannot be computed due to the fact that two images lie in disconnected components of the graph, but the path exists in the real environment, the path will be declared as a false negative. The ratio of false negatives will reflect the inability of the planner to find a valid path using the map.

One could argue that two random images should be randomly selected from the environment instead of the map if the completeness of the map is to be measured. However, in that case, first, each random image sampled from the environment should be localized in the map. As stated in Section 4.7.1.1, this localization procedure measures the coverage of the map and may result in failure either due to low coverage score or due to the failure of the localization algorithm to find the valid match in the map. Therefore, to measure the utility of each task independently, query images are sampled from the images in the map. Applying this criterion in the selection of query images, this metric, however, will return zero false negative scores for any appearance graph consisting of only one connected segment since there will always be a path between any two vertices of the graph.



(a) Appearance-based map (b) Grid map of the actual environment

Figure 4.17: Two images of interest denoted with red circles are shown in the appearance-based map (a) while the path connecting these two images shown in the occupancy grid map of the actual environment (b). The dotted line shows the separation of two physically disconnected regions of the environment. This figure an example of a valid path in the actual environment which is not captured by the appearance-based map covering the same environment.

The ratio of the number of false negatives to the total number of path computing requests also shows the connectivity of the map. Instead of counting the number of disconnected components in the map, this score considers the size of each disconnected component. For instance, failing to create an edge between second and third vertex has much less effect than failing it between two vertices from two large disconnected graphs.

In order to measure the completeness of the maps with respect to their potential planning utilization, $n = 10$ image pairs are randomly selected from the maps. Then, Dijkstra's algorithm is applied to find a path connecting these two images where the reciprocal of the number of matching features is used as the cost for each subsequent image pair along the path. Repeating this procedure $m = 10$ times, the total number of false negatives are counted. The second map, M_2, is fully connected and therefore gets a full completeness score. On the other hand, M_1 has multiple connected components and therefore gets non-zero false negatives. In order to demonstrate how

this metric behaves in the presence of only few edges, the parameter T_{min} is used to set the minimum number of feature matches required to declare two images as a match. The increase in the threshold results in a map with edges only between very similar images. Due to this decrease in connectivity the false negative ratio increases as shown in Figure 4.18. The sudden jump between $T_{min} = 30$ and $T_{min} = 40$ is the result of losing connectivity in the middle of the path causing the map to split into two large components. Therefore, the probability of randomly selecting both vertices from the same connected subgraph decreases drastically.

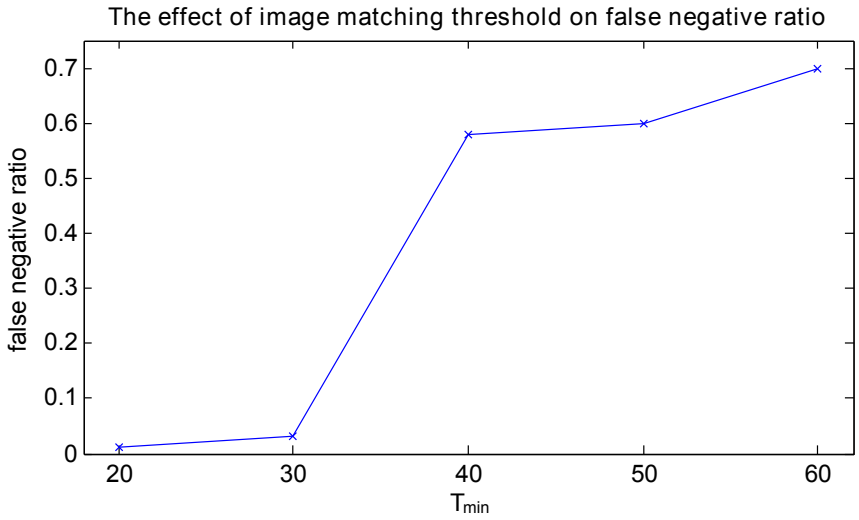


Figure 4.18: The effect of the number of feature correspondences to declare an image match, T_{min}, on the number of false negatives of path validity test is shown.

4.7.1.3 Navigation

Navigation, defined as the ability to reach an assigned target location from the robot's current position, is another fundamental level of competence that is greatly influenced by the available spatial model.

The navigability of a map can be measured by computing the stability and robustness of its internal paths from a robot navigation perspective. However, assessing the quality of all internal paths within a map is feasible due to the high number of possibilities. Thus, we embrace a randomized approach and apply the metric only to a randomly selected subset of internal paths. Since most of the proposed visual navigation algorithms as presented in Chapter 2 base their servoing algorithm on

multi-view geometry or more specifically on the fundamental matrix, we grasp the idea of measuring the quality of the fundamental matrix, F, between consecutive images in each path. Given a pair of consecutive images their corresponding features are extracted. Then, a RANSAC algorithm as described in Section 3.1.3 is utilized to compute the fundamental matrix based on a number of randomly selected tentative feature matches.

The average error of the fundamental matrix along the path is computed and returned as navigation error of the path. The overall navigation score of the map is then assigned as the reciprocal of the average navigation error of all randomly created paths. By design this metric favors maps which use a more conservative approach by only creating edges between very similar images which indeed fits our initial goal of rewarding such maps since they provide a safer medium to navigate in the environment. In order to show this effect, navigability metric is applied on the maps built in the previous section by changing the threshold responsible for determining when an edge is created. Figure 4.19 presents the trend of this error as the edge selection procedure gets more selective. As expected, the more similarity is required to create an edge, the lower navigation error is obtained for that map.

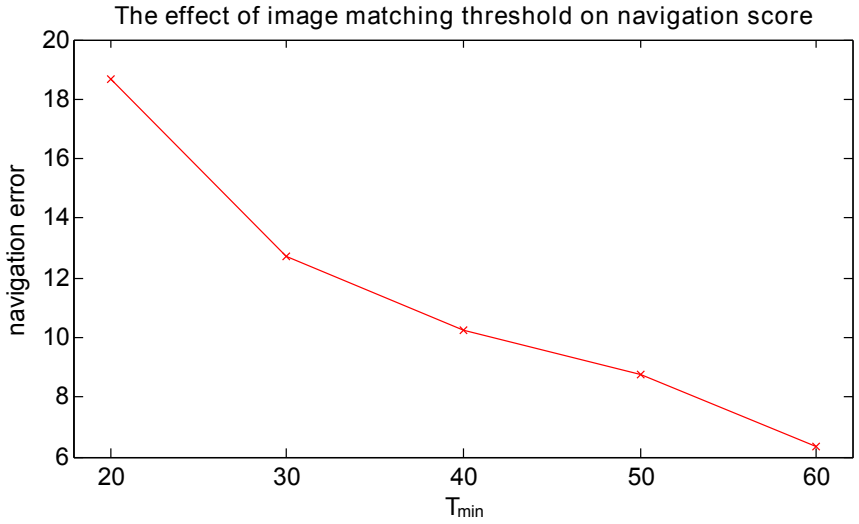


Figure 4.19: The effect of the number of feature correspondences to declare an image match, T_{min}, on the average navigation error of randomly selected paths is shown.

4.8 Conclusions

We have presented a system capable of incrementally building an appearance-based map from scratch. The system is designed based on the idea that robots can map an environment and localize in it simultaneously. This is due to the fact that localization and mapping procedures are designed to be entangled to each other. At every step when robot captures an image to localize itself, the results of its localization call can be used to update the map as described in Section 4.5. Both localization and mapping algorithms run in real-time using a laptop with standard computational power. It is evident that the computational time is dependent on the resolution of the images used to generate the map and for high resolution images of feature rich environments more computational power may be required to run the system in real-time.

The training procedure entangled with the BoW approach is shown to be non-sensitive to the training images since we used visual words extracted from random indoor images to create the dictionary which we later used to map the Engineering Building of UC Merced and successfully localized in it. It is once again important to note that no images from the environments we mapped and localized are included in the training images.

An appearance-based map, in general, can be used for localization and also to plan a path that can be followed by a robot using an image based visual servoing algorithm. An interesting feature of the system we propose is that the map can be utilized by a set of heterogeneous robots, i.e., it can be shared among the members of an heterogeneous multi-robot team. The proposed approach has been implemented and validated both on a team of two robots, effectively demonstrating the power of the proposed system.

In this chapter we also introduced a metric to measure the performance of appearance-based maps with respect to different tasks. These metrics focusing on different aspects of the map may not agree on one final map being the best for all tasks, but will measure the map's specific task-based performance. However, these objective measures should be repeatable so that other researchers can reproduce the same experiments to compare published results with their own algorithms. Since the real environment is used as ground truth and providing access to it is not feasible, the data that is used in the computation of the performance metrics should be shared in an online public repository. First of all, images that are captured during the navigation of the robot are needed in order to generate a map of the environment. Different map creation algorithms can be tested on this image set and then

evaluated by the task based measures proposed in this chapter. In addition to this image set, the computation of the localization metric needs images captured at random locations in the environment. The planning metric, on the other hand, requires the information whether there is actually a physical path in the real environment between the locations encoded in the randomly selected images from the map. This information can be captured in a binary *connectivity matrix* which stores all possible image combinations. Each element in this matrix stores true if two images corresponding to this element are physically connected, and false otherwise. Similarly, in order to count false negatives to calculate the planning metric, we need to know whether random images from the environment are physically connected. In summary, in order to make visual map generation and benchmarking possible even without the privilege of accessing the real environment, we believe that the following elements should be made available to third parties: 1) the image set collected by the robot along its path; 2) random test images captured in the same environment; 3) their connectivity matrices. Only then the experiments can be repeated and the performance of map generation algorithms can be compared.

CHAPTER 5

Appearance-Based Map Merging

We consider the problem of merging multiple appearance-based maps independently created by robots exploring the same unknown environment. Spatial awareness is one of the fundamental abilities for mobile robots, and the ability to fuse multiple maps into a single spatial model greatly enhances the utility of multi-robot systems. While autonomous map building has attracted enormous attention in the last two decades, map merging has emerged only recently and fewer results are available. Map merging is here defined as the process of combining multiple maps that have been independently built by different robots. This is different from cooperative mapping, where multiple robots concurrently and continuously contribute their data to a single map. Note that map merging is an incremental and online process, i.e., in general maps are not necessarily merged together when the mission is over, but rather while they are still being built. For example, robots exchange and merge their maps when they are within communication range [92], but during the remaining part of their mission they operate independently and possibly continue to improve their spatial models individually. Previously, solutions for merging occupancy grid maps [12, 20, 21, 56, 55], feature maps [5, 6, 135, 96, 34], and topological maps [46, 80] have been proposed. However, the problem of combining multiple appearance-based maps has not been considered before, and here we present the first solution to this problem. Given that appearance-based maps are represented by graphs, our method aims to identify edges connecting vertices belonging to different maps. Stated differently, it aims to find pairs of similar images in disjoint maps, where the concept of similarity is defined in Section 4.

A problem inherently related to map merging is the evaluation of results. Two maps can be merged in countless ways, so the problem of quantitatively measuring the quality of a merged map immediately arises. Unsurprisingly, this is an open problem and no well-accepted metric exists, as witnessed by recent special issues and projects devoted to robot benchmarking [110]. We therefore put forward a criterion based on algebraic connectivity specially developed for appearance-based maps. The original contributions we offer in this chapter are the following.

- We propose and analyze the use of algebraic connectivity as a metric to measure the quality of multiple appearance-based maps merged together.
- We introduce an anytime appearance-based map merging algorithm that quickly identifies edges leading to the largest gain in terms of algebraic connectivity. Given that the problem of determining edges leading to the largest increase in algebraic connectivity is known to be NP-hard [129], the algorithm we propose necessarily produces a suboptimal solution. The algorithm is anytime. In other words, it iteratively discovers and adds new edges, and outputs a partial, yet valid solution if stopped before completing its computation which eventually yields to a complete solution.
- The algorithm and the metric are experimentally validated by merging appearance-based maps autonomously built by mobile robots navigating indoor. Experimental findings reveal that our proposed method quickly determines the most convenient edges to add, and according to an anytime paradigm, eventually determines all edges that could be added.

The results presented in this chapter have been published in [42].

5.1 Problem Formulation

In this section we will define the problem description of the map merging problem. Given two appearance-based maps $M_1 = (V_1, E_1, w_1)$ and $M_2 = (V_2, E_2, w_2)$, determine a merged map $M_m = (V_m, E_m, w_m)$ with $V_m = V_1 \cup V_2$, and $E_m = E_1 \cup E_2 \cup E_c$, where $E_c \subseteq V_1 \times V_2$ is the set of edges connecting images in V_1 with images in V_2. Evidently, w_m are the weights induced by the image similarity function S. A simple illustration of map merging concept is shown in Figure 5.1. We require that every edge in the merged map still satisfies the condition $S(v_i, v_j) > T_{min}$ introduced in Chapter 4. This is trivially true for edges coming from E_1 and E_2, but must also hold for edges in E_c discovered during the merging process. From our point of view the very purpose of map merging is to enable robots to navigate and reach areas discovered by other robots (for example, using the image-based methods described in [41, 113]). Therefore, our emphasis is on finding edges in E_c connecting the two maps and enabling navigation. From the definition it is evident that the merged map M_m retains all images originally included in M_1 and M_2 as well as their edges. The value of merging is given by the novel edges discovered in E_c that link the two graphs together. Given the definition of E_c and its constraints, it is apparent that one could

evaluate image similarity for every couple of images in V_1, V_2 and add edges whenever their similarity exceeds T_{min}. This brute-force approach will discover the largest possible set, E_c^*, and will be further discussed in an later section. However, brute-force search is clearly time consuming and we are interested in faster alternatives that will output sets of edges that are subsets of E_c^*.

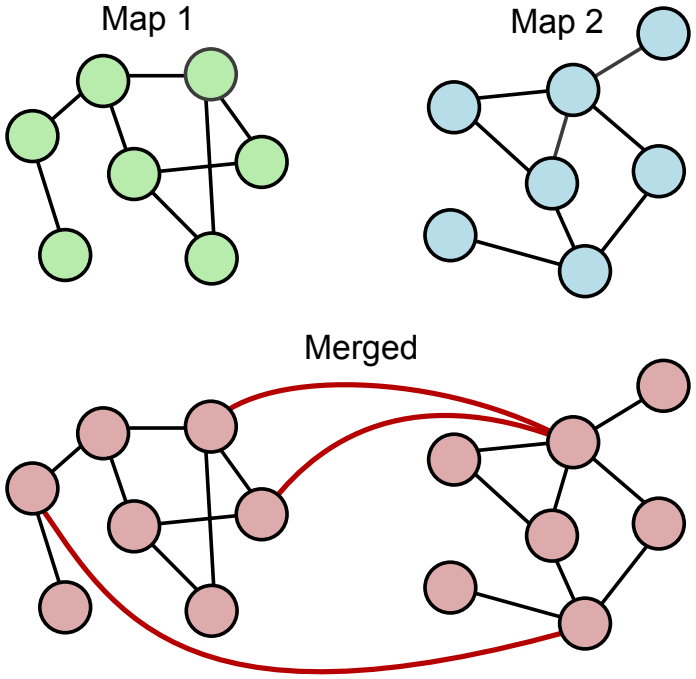


Figure 5.1: Given two simple maps, the merged map is created by inserting new inter-map edges to the union of both maps.

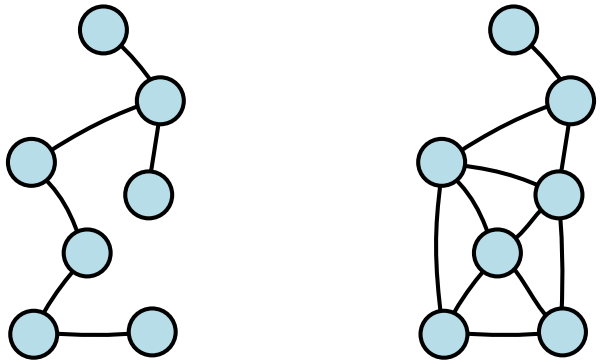
We conclude this section noting that the problem statement considers only the case where two maps are merged, but the idea can naturally be extended when more than two maps need to be combined. Indeed, after introducing our appearance-based map merging algorithm focusing on pairwise merging in Section 5.3.1, we present the extended version of the algorithm that merges multiple maps in Section 5.3.2. In order to assess the performance of the proposed method and prove that it works better than the state-of-the-art methods, a metric that measures the quality of the merged maps needs to be defined. Hence, before continuing to the discussion on the merging algorithm, in the next section we propose a method to measure merging quality that relies exclusively on graph properties.

5.2 Measuring the Quality of Merged Maps

In Section 4.7, we addressed the problem of defining mapping metrics particularly for appearance-based maps and proposed a set of task-based performance evaluation criteria to measure the quality of appearance-based maps independently from the algorithm used to build them. Here, we are concerned with a slightly different problem, i.e., assessing the quality of merged maps. Referring to the problem stated in the previous section, for a given instance of the map merging problem two different algorithms may produce two sets of connecting edges, say $E'_c \subset E^*_c$ and $E''_c \subset E^*_c$. We are therefore interested in a criterion to establish in a quantitative way which one is better. It is important to note that when two or more maps are merged together, the resulting map is also an appearance-based map and still possesses all the properties of appearance-based maps. Thus, its quality with respect to a particular task can be measured using the evaluation criteria presented in Section 4.7. The traits that define a good appearance-based map, however, do not necessarily determine the quality of the merged map, the merging process, or the performance of the merging algorithm. The quality of the merging process becomes especially important when maps are merged using anytime algorithms. In order to compare the performance of map merging algorithms, the quality of intermediate maps need to be measured. So that even if the final merged maps generated by two anytime algorithms are the same, i.e., $E'_c = E''_c$, we can choose the algorithm that creates a *better* map in the early stages of the merging process. We maintain that the main attribute that defines the goodness of a merged appearance-based map is *entanglement*. Informally speaking, entanglement is the amount of effort needed to split the merged map back into two separate maps. Given two solutions to the merging problem, we prefer the one with higher entanglement.

5.2.1 Defining Entanglement in Terms of Connectivity

The concept of entanglement can be formalized in different ways. Edge connectivity $e(G)$ is a commonly used metric to measure the connectivity of graphs and is defined as the minimum number of edges to be removed to get two separate components. In a map merging scenario where vertices are preserved and interconnecting edges are added this metric offers poor results. For example, for a well connected graph with just a single vertex of degree 1, $e(G) = 1$. Even tough inserting edges will improve the graph's overall connectivity (and then utility in terms of navigation), its edge connectivity will not change until one of the inserted edges is connected to the vertex with degree 1 as illustrated in Figure 5.2. Based on this and similar observations,



(a) (b)

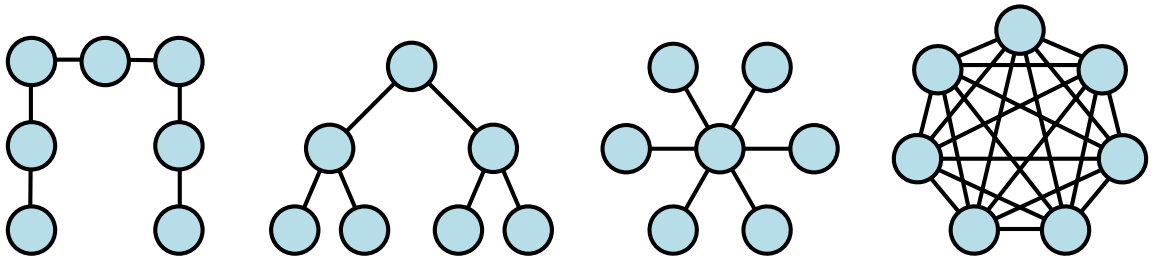
Figure 5.2: a) A sample graph with an edge connectivity of 1 is illustrated. b) The connectivity of the graph is bolstered with the addition of few edges, but its edge connectivity remains unchanged.

we maintain that *algebraic connectivity* is a better measure to assess the quality of map merging in the appearance-based domain.

Introduced in the seminal work by Fiedler [49], algebraic connectivity is a spectral property of the graph widely used to measure robustness and connectivity. Algebraic connectivity carries more information about the structure of the graph and is a more useful measure than edge connectivity. For instance, edge connectivity of all trees is equal to one, whereas the algebraic connectivity of a star is higher than that of a path as can be seen in Figure 5.3 which displays some examples of simple graphs and substantiates our preference over edge connectivity. In the following we therefore recall its formal definition, describe some of its important properties which substantiate our preference over edge connectivity. We emphasize that our choice for this metric is based on some of its well known properties, and we are not aiming at unveiling new properties for this well studied mathematical object.

5.2.2 Algebraic Connectivity

Let $G = (V, E)$ be an undirected graph with n vertices. The adjacency matrix, A, of G is a symmetric $n \times n$ matrix where n is the number of vertices, and $A_{ij} = 1$ if there is an edge between v_i and v_j and $A_{ij} = 0$ otherwise. The Laplacian matrix $L(G)$ is defined as $L = D - A$ where D is the $n \times n$ diagonal matrix of vertex degrees. The second smallest eigenvalue of L, $\lambda_2(L)$, is called *algebraic connectivity* and is usually indicated as $\alpha(G)$. The algebraic connectivity and its properties play a very



$e(G) = 1$	$e(G) = 1$	$e(G) = 1$	$e(G) = 6$
$\alpha(G) = 0.1981$	$\alpha(G) = 0.2679$	$\alpha(G) = 1$	$\alpha(G) = 7$

Figure 5.3: The figure shows edge connectivity $e(G)$ and algebraic connectivity $\alpha(G)$ for different elementary graph topologies. It is evident that edge connectivity does not differentiates between significantly different graphs, like chain, star, and tree.

important role on solving a wide variety of problems both in graph theory, such as the expanding properties of graphs, isoperimetric number, genus and other invariants of graphs, and in combinatorial optimization such as the problem of certain flowing process, the maximum cut problem and the traveling salesman problem [50, 127, 128].

5.2.2.1 Properties of Algebraic Connectivity

Various properties of algebraic connectivity have been discovered [49, 50] and a substantial amount of research [1, 14, 44, 49, 50, 57, 63, 90, 94, 106, 123] has been devoted to finding tight upper and lower bounds on the algebraic connectivity in the case of simple graphs[1]. Here we will only cover a few of the important properties that are related to the underlying graph structure of appearance-based maps and the map merging process. For the rest of this section, unless stated otherwise, let $G = (V, E)$ be a simple connected graph with order of $n = |V|$ and size of $m = |E|$, d_j be the degree of the j^{th} vertex, and $\delta(G)$ be the minimum vertex degree of G.

The algebraic connectivity is related to the sparsity of cuts[2] in the graph. That is, a graph with large algebraic connectivity cannot have very sparse cuts and therefore requires great effort to be partitions, i.e highly entangled. The algebraic connectivity of a graph is also an indication of the number of connected components it contains.

[1] A simple graph, also called a strict graph [163], is an unweighted, undirected graph containing no graph loops or multiple edges [58].

[2] The sparsity of a cut that bipartitions a simple graph is defined as the ratio of the number of edges across the cut divided by the number of vertices in the smaller half of the partition.

Since $D_{ii} = \Sigma_j A_{ij}$, it follows that all rows and columns of the Laplacian sum to zero, and hence that the vector $\mathbf{1}$=(1,1,1...) is always an eigenvector of L with corresponding eigenvalue 0. More precisely, the multiplicity of the value 0 as en eigenvalue of L is equal to the number of connected components of G [30]. Thus, we have $\alpha(G) = \lambda_2(L) > 0$ if and only if G is connected.

Lemma 1. *[49] If $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are edge-disjoint graphs with the same set of vertices then*

$$\alpha(G_1) + \alpha(G_2) \leq \alpha(G_1 \cup G_2)$$

Hence, $\alpha(G)$ is a non-decreasing function for graphs with the same set of vertices, i.e., $\alpha(G_1) \leq \alpha(G_2)$ if $\alpha(G_1) \subseteq \alpha(G_2)$. Furthermore, $\alpha(G)$ is a monotonically increasing function of the edge set. That is if $E_1 \subseteq E_2$, then $\alpha(G_1) \leq \alpha(G_2)$. Therefore, the more edges the merging algorithm inserts, the more connected the graph, and the higher algebraic connectivity will be. This concludes that E_c^* is optimal when algebraic connectivity is the considered metric.

Lemma 2. *Following the previous property, let G_1 arise from G by removing k vertices from G and all adjacent edges. Then*

$$\alpha(G_1) \geq \alpha(G) - k$$

Lemma 3. *[49] For a complete graph[3] G_n with n vertices $\alpha(G_n) = n$*

Lemma 4. *[49] If G is a non-complete graph with n vertices, then $\alpha(G) \leq n - 2$*

Lemma 5. *[49] It is also known that for a non-complete graph G, algebraic connectivity defines a lower bound on both the vertex, $\upsilon(G)$, and edge connectivity, $e(G)$ where edge connectivity is upper bounded by the minimum degree of the graph, delta(G).*

$$\alpha(G) \leq \upsilon(G) \leq e(G) \leq \delta(G)$$

Moreover, as shown in [99, 126], $\alpha(G)$ is lower bounded by a function of edge connectivity.

$$\alpha(G) \geq 2e(G)\left(1 - \cos\frac{\pi}{n}\right)$$

[3] *A complete graph is a graph in which each pair of graph vertices is connected by an edge.*

Lemma 6. *In [63], Guixian shows that a simple connected graph $G = (V, E)$ order of n is bounded below both by a monotonic function of squared sum of all vertex degrees*

$$\alpha(G) \geq \frac{2m - \sqrt{(n-2)\left((n-1)\sum_{i \in V} d_i^2 + 2m\,(n - 2m - 1)\right)}}{n-1}$$

and the minimum degree of the graph.

$$\alpha(G) \geq 2\delta(G) - n + 2$$

Therefore, edges that bolster connections between weakly connected vertices and hence increase the minimum degree yield larger improvements in algebraic connectivity.

Lemma 7. *As proven by Mohar in [126], if G is a simple graph with n vertices and m edges and D is its diameter*[4]*, then we have*

$$\alpha(G) \geq \frac{4}{nD}$$

Similarly, Lu et al. [106] showed that

$$\alpha(G) \geq \frac{2n}{2 + (n^2 - n - 2m)\,D}$$

In other words, algebraic connectivity is bounded below by the diameter of the graph. Thus, algebraic connectivity may be elevated by creating edges between distant vertices and decreasing the diameter of the graph.

All these properties relate to the characteristics that a quality measure for map merging algorithms should have. It therefore appears that algebraic connectivity is a promising criterion and we embrace it to measure the quality of merged maps and evaluate the performance of merging algorithms. Figure 5.4 illustrates some examples as the embodiment of the properties presented here and aims at showing how algebraic connectivity changes when edges are added between different vertices of the graphs being connected. Graphs associated with the appearance-based maps considered in this chapter include thousands of vertices and edges and are evidently much more complex. However, similar effects in terms of variations of algebraic connectivity are observed when edges are added between vertices that are peripheral or more internal to the graph. These changes in algebraic connectivity justify the that will be later on observed in Figure 5.7 when discussing the experimental results.

[4] *The diameter of a graph is defined as the greatest distance between any pair of vertices.*

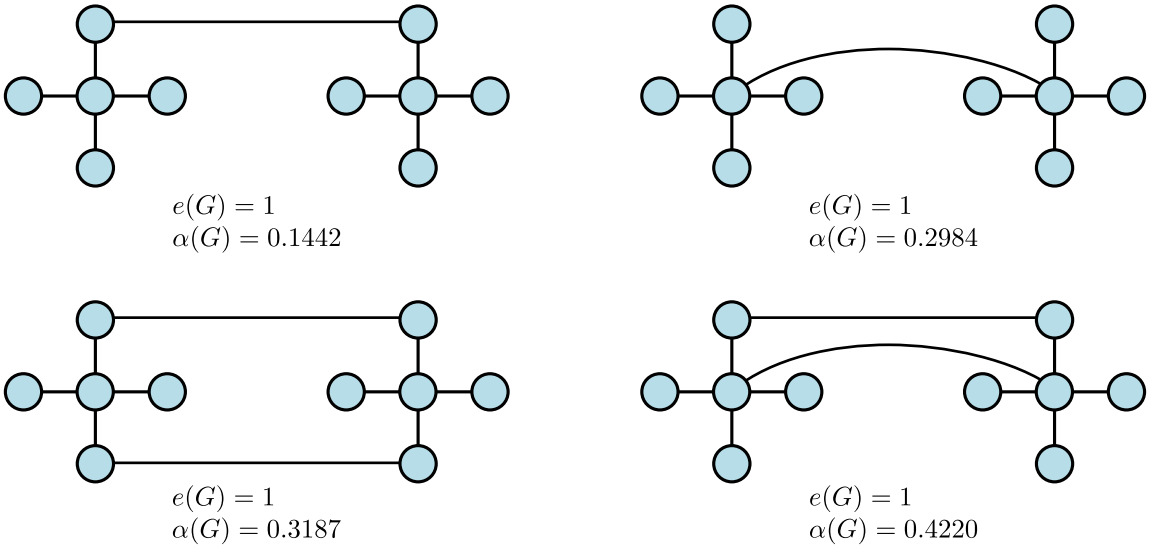


Figure 5.4: Variations in algebraic connectivity when different edges are added between two star shaped graphs being merged.

5.3 Merging Algorithm

5.3.1 Merging Two Maps

Let $M_1 = (V_1, E_1, w_1)$ and $M_2 = (V_2, E_2, w_2)$ be two appearance-based maps independently created by two robots running the algorithm described in Section 4.5. Without loss of generality let $|V_1| \geq |V_2|$. As explained in Section 5.1, our focus is on determining the set E_c, i.e., edges connecting vertices in V_1 with vertices in V_2. The motivation behind this idea is that these connections are needed to take advantage from maps created by other robots. In particular, with no connectivity between its own map and the new one, a robot will not be able to use image-based visual servoing to reach locations discovered by the other robot. In the following we present a centralized solution, i.e., we assume all computation is performed by one robot, but the proposed method can be easily distributed to share the workload.

To put our algorithm in perspective, it is useful to first consider a brute-force solution. Edges in E_c can be determined by brute-force by repeatedly calling the localization algorithm to localize every vertex in $v \in V_2$ inside map M_1. Localization returns the set of images whose similarity with v exceeds the given threshold T_{min}. This approach therefore identifies all edges that can be added between M_1 and M_2. Since algebraic connectivity is a monotonic function of the set of edges,

this algorithm is optimal with respect to the algebraic connectivity metric formerly introduced. However, its performance is evidently unsatisfactory from the point of view of required time. Nevertheless, it provides a useful yardstick to evaluate the performance of the solution we propose.

A fundamental aspect to consider is that algebraic connectivity will be maximized only after adding all possible edges in E_c. Therefore, if the goal is to optimize this performance metric there is not much one can do to outperform the brute force approach. But a key observation to develop a more efficient method is that most edges make only marginal contributions to algebraic connectivity, while just a few yield large gains [89]. Our goal is therefore to develop an *anytime* algorithm that quickly determines a subset of E_c yielding a large increase in algebraic connectivity and eventually reaching the optimal value returned by the brute-force method. The algorithm *QuickConnect* that we describe in the following aims to test and insert edges yielding large gains early in the process, and to postpone performing computationally expensive multi-view geometry and spatial consistency tests for edges giving only marginal increments. The algorithm is therefore *anytime* in the sense that even when it is stopped before it has processed all its input, it still produces a valid solution. Moreover, if it is allowed to run to completion, it determines the optimal solution and the quality of the solution it produces is a monotonic function of the time spent. The problem of identifying edges yielding the largest increase in algebraic connectivity is NP-hard [129] and the method we propose is therefore necessarily suboptimal.

Algorithm 3 sketches the pseudocode for QuickConnect. The algorithm consists of two phases, i.e., *exploration* (Line 1-11) and *refinement* (Line 12-15). Exploration works towards quickly identifying the most similar images from both maps and adding edges in a selective nature. We only add an edge if at least one of its source and target vertices is not connected by an earlier inter-map edge inserted during the exploration phase. The goal is to create only the essential connections between most similar vertices, and postpone the validation and insertion of the remaining edges. After most important edges are created between similar vertices, refinement starts and all edges not processed during exploration are considered. According to this idea, QuickConnect eventually discovers all possible connections, but ideally the most relevant ones are identified early on during the exploration process.

Exploration is based on the incremental construction of a similarity matrix R where r_{ij} stores a score (vote) between image $v_i \in V_1$ and $v_j \in V_2$. The score is the number of common features between v_i and v_j and R is incrementally built as follows. A priority queue W is initialized with all dictionary words appearing at least

Algorithm 3 QuickConnect($M_1 = (V_1, E_1, w_1)$,$M_2 = (V_2, E_2, w_2)$)

```
1: W ← InitializeQueue(M_2)
2: R, L ← null
3: E_c ← ∅
4: repeat
5:    d ← W.pop_front()
6:    P ← getVotes(M_1,M_2,d)
7:    E ← update(R,P)
8:    if !E.empty() then
9:       L ← processEdges(E,W,L,E_c)
10:   end if
11: until W.empty()
12: L ← sort(L)
13: for all e in L do
14:   insertEdge(e,E_c)
15: end for
16: return E_c
```

once among images in map M_2 (Algorithm 3, Line 1). All words are initially assigned the same priority. Then, all words in W are sequentially dequeued. For every word d a vote vector P is computed. Images in M_2 which contain this word cast a vote for all images in M_1 that also have this word in their appearance vectors (Line 6), and the similarity matrix R is then updated with the integration of new votes (Line 7). While updating R a list of candidate edges E is also created. A candidate edge between v_i and v_j is added to E as soon as r_{ij} exceeds T_{min}. Every candidate edge is then processed by the algorithm processEdges described in Algorithm 4.

Algorithm processEdges inserts a new edge in E_c only if at least one of its source and target vertices does not share an earlier discovered edge with a vertex in M_2 (Algorithm 4, Line 2). If the edge is added, then both vertices it connects are suspended from inserting any more edges until the end of exploration phase (Line 4 - 5). All edges connected to an already processed vertex are taken into a waiting list L that will be processed during the refinement stage (Line 10). In order to improve the performance of exploration, we also alter the priority of queue W based on the following *locality* observation. Due to the fact that only the images with high similarity are connected by an edge, a word seen by an image is likely to be visible by its adjacent images in the graph. Therefore, by casting votes from this word we can identify new edges originating from these neighbor vertices. To implement this idea,

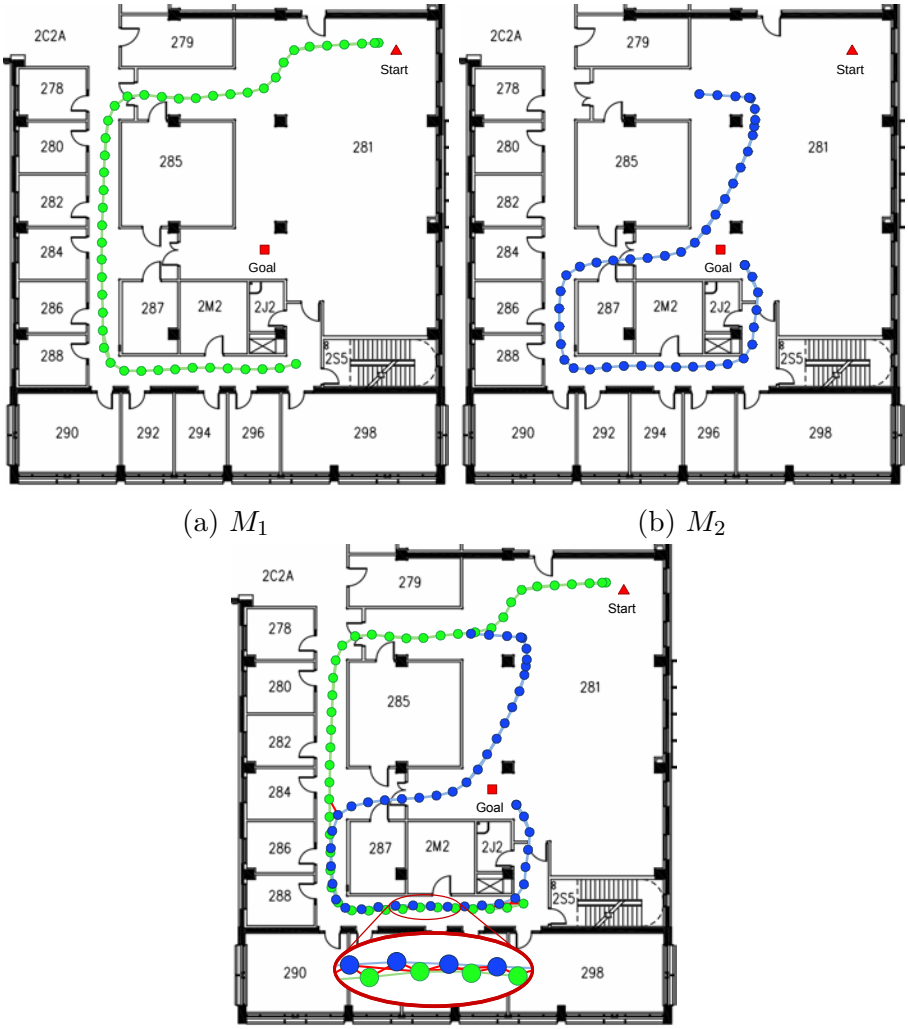
Algorithm 4 processEdges(List E, Queue W, List L)

```
1: for all e in E do
2:   if !isProcessed(e.source) OR !isProcessed(e.target) then
3:     insertEdge(e,E_c)
4:     setProcessed(e.source, true)
5:     setProcessed(e.target, true)
6:     Q1 ← getWords(e.source)
7:     Q2 ← getWords(e.target)
8:     W.reorder(Q1 ∪ Q2)
9:   else
10:    L.push_back(e)
11:  end if
12: end for
13: return L
```

once a similar image is found we increase the priorities of the words it contains (Line 8) so that if they are not processed yet, they move to the top of the word priority queue and become the next in the list to be processed.

When all the words are processed and the similarity matrix is completely filled, the refinement stage starts (Algorithm 3 Line 12 - 15). Exploiting the fact that a monotonic function of the minimum vertex degree and the squared sum of all vertex degrees provide lower bounds on algebraic connectivity as shown in Section 5.2.2, we aim to improve the minimum vertex degree (and then algebraic connectivity) early in the map merging process. Therefore, identified edge candidates are sorted with respect to the minimum degree of the two vertices it connects. Finally, edges are inserted sequentially starting from the ones connecting vertices with least number of adjacent vertices.

Figure 5.5 shows an example of result for the map merging process implemented by the algorithm just described. Figure 5.5a and 5.5b displays two maps independently built by two robots exploring different parts of the Science and Engineering building at UC Merced. Note that vertices of the graph have been correctly placed in the blueprint of the building because the robots used for this experiment are also equipped with a laser range finder, and we run a localization algorithm together with the known map to accurately determine where pictures are taken. This information is however used only for display purposes and for performance analysis, but is not available to the merging algorithm. Figure 5.5c shows the resulting merged map. In this case too, the merged graph M_m is overlaid to the blueprint to evidence the



(a) M_1 (b) M_2

(c) M_m

Figure 5.5: The top two panels show two appearance-based maps independently built by two robots exploring different parts of the same environment. The last figure shows the merged map and the detail shows some of the edges (red) discovered during the merging process and linking the two maps together.

coverage of the merged map, but this is just for display purposes. The reader will notice that if the maps are used for navigation via visual servoing the merged map is much more useful than the individual ones because thanks to its discovered edges (see detail in the bottom one) a robot will be able to go from the start to the goal location, whereas none of the individual maps would allow that.

5.3.2 Merging Multiple Maps

In the previous section we addressed the problem of merging two maps. The presented method can be easily extended to solve the problem of merging n appearance-based maps. The map merging problem description presented in Section 5.1 can be generalized as follows. Given a set of N appearance maps $M = \{M_1, ..., M_N\}$ where $M_i = (V_i, E_i, w_i)$, compute a merged map $M_m = (V_m, E_m, w_m)$ with $V_m = \bigcup V_i$, and $E_m = \bigcup E_i \cup E_c$, where $E_c = \bigcup E_c^{ij}$ with $1 \leq i, j \leq N$, and $E_c^{ij} \subseteq V_i \times V_j$ is the set of edges connecting images in V_i with images in V_j. Note that, as in the previous description, the merged map includes all vertices from all maps and we focus on creating new edges between images from different maps.

One approach to merging multiple maps is to merge two of them, merge the resulting map with a third, and so on, until all input maps are merged together. This *sequential pairwise merging* method merges a pair of maps at a time and creates in-between merged maps after each step. However, the intermediate solutions provided by this approach do not capture the connectivity of all maps until the very last step. In other words, during all but the last step there is at least one map not involved in the merging process, and because of this map the number of connected components within M_m is greater than one. Hence, the algorithm is not strictly anytime because algebraic connectivity remains zero until merging with the last map starts.

To overcome this limitation, and restore the desirable anytime feature, we introduce *QuickConnect-simultaneous*, an efficient parallel merging algorithm that overcomes the disadvantages of sequential merging approach. In the exploration phase, the algorithm incrementally constructs N_R similarity matrices in parallel, where $N_R = \binom{N}{2} = N(N-1)/2$, i.e., one similarity matrix for each pair of maps. For each word being processed, the algorithm first identifies images that contain this word from the first pair of maps and casts votes into their similarity matrix as described in Algorithm 3. Then, the same procedure is repeated for the remaining map pairs and the corresponding similarity matrices are filled with the votes for the same word. When all map pairs cast their votes into their similarity matrices, this word is labeled as processed and the next one in the priority queue is selected. A candidate edge is identified whenever the score of an image pair exceeds T_{min} in any of the similarity matrices, and the edge is inserted according to the processEdges routine described in Algorithm 4. Thanks to this parallel voting schema, from the very beginning edges between all map pairs are created simultaneously. All the remaining identified but not inserted edges from all map pairs are sorted by their vertex degree in an increasing order and inserted during the refinement phase starting with the one that has the minimum degree.

5.4 Results

In this section we present a comparative evaluation of the proposed map merging algorithm. First, we show results on merging two maps together and then at the end of this section we present the results of multi-map merging case. Brute-force merging sets the baseline for our comparisons because it eventually reaches the highest possible algebraic connectivity of the merged map. However, the trend to reach the maximum strongly depends on the sequence it follows in selecting couples of vertices to try adding edges between them. Algebraic connectivity may quickly increase if the merging algorithm finds good vertices and edges early in the process, but it may also be the case that good attempts are made only later on in the process. Therefore, if the goal is to assess the performance of brute-force from an anytime standpoint, the algorithm should be tested on a large set of highly diverse maps. On the other hand, having only a limited number of datasets available, we compare our algorithm against a randomized brute-force method, *BruteUniform*, that randomly selects couples of images from M_1 and M_2 using a uniform distribution, and we consider its average performance over repeated runs. As an additional term of comparison, we introduce *DegreeMin*, a variant of the randomized brute-force method that samples vertices from M_2 with a mass distribution inversely proportional to their degree. The rationale behind *DegreeMin* is trying to bias the search towards vertices with low degrees in order to possibly obtain large gains in algebraic connectivity, as outlined in section 5.3.1.

To test these algorithms, various appearance-based maps with a number of vertices ranging from a few hundreds to several thousands are built by a P3AT mobile robot equipped with a monocular camera using the map building algorithm described in Section 4.5. Various combinations of these maps are merged using these algorithms and a representative sample of these results are shown in Figure 5.6. The first 5 columns correspond to map pairs built in one side of the engineering building at UC Merced (research labs), whereas the remaining columns correspond to maps built on a different side of the building featuring a significantly different layout (administrative wing).

The goal is to evaluate the algorithm on realistic and heterogeneous datasets. For different map pairs we present the quality of merging (i.e., algebraic connectivity) after 10% of the time required by the brute-force approach to complete its processing. The rationale behind this evaluation criterion is to embrace an anytime viewpoint and assess how different algorithms will perform when stopped early. The results for the randomized algorithms (randomized brute-force and *DegreeMin*) are the average of 20 runs. The chart shows that *QuickConnect* outperforms the other two methods,

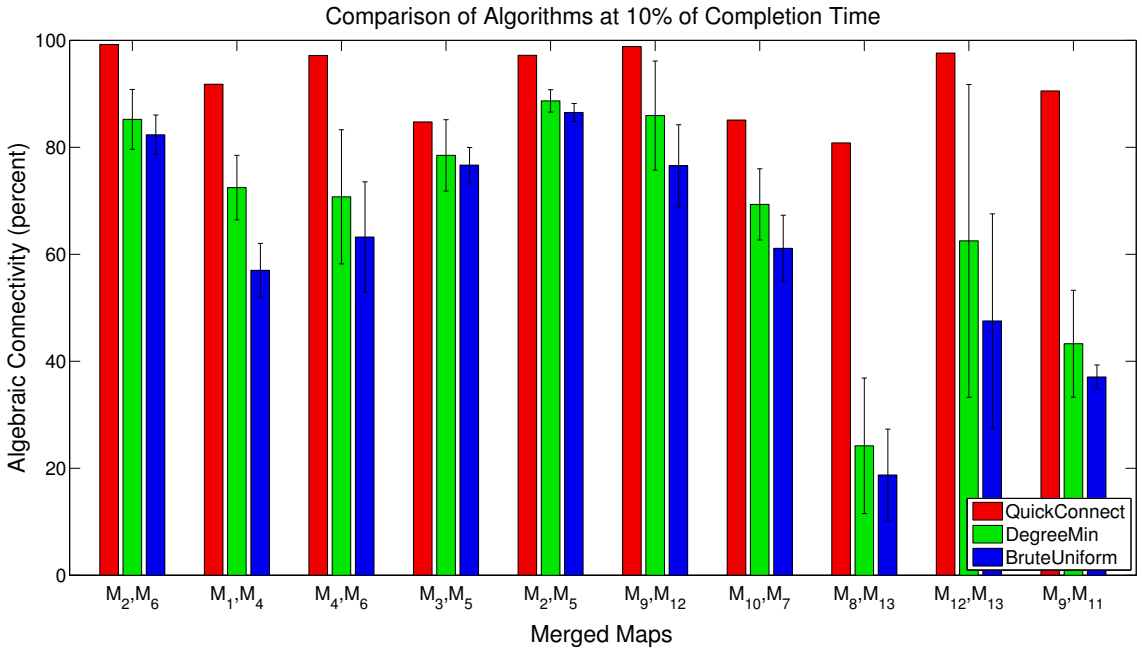
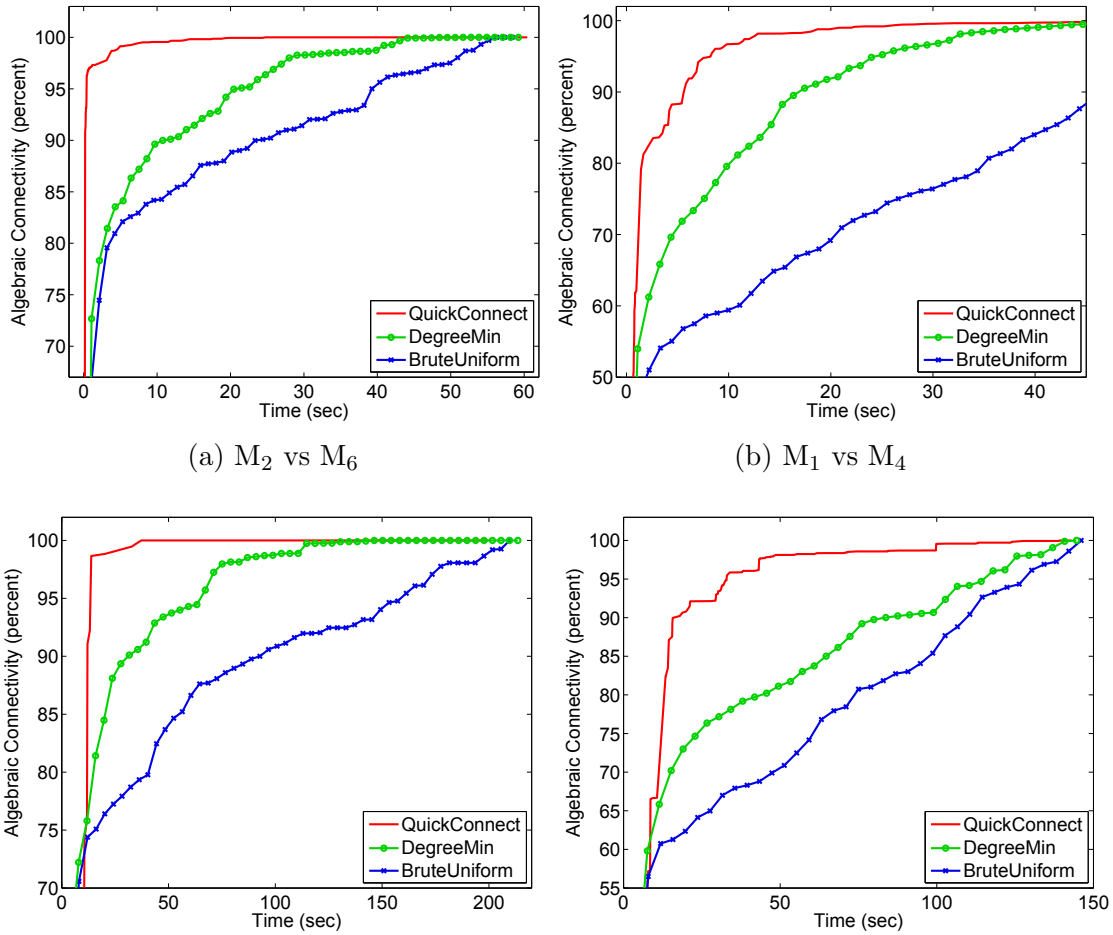


Figure 5.6: The chart contrasts algebraic connectivity when merging is stopped at 10% of the time required by brute-force algorithm to complete its exhaustive search. Results are shown for a representative selection of merged maps and three different algorithms are compared: QuickConnect, DegreeMin, and BruteUniform. Error bars representing one standard deviation are shown for randomized algorithms. Note that since algebraic connectivity varies for different map couples, we display a normalized value corresponding to the percent of the maximum value.

reaching above 90% of the maximum possible algebraic connectivity within 10% of total time. It is also notable that *DegreeMin* with its heuristic that favors minimum degree vertices performs better than randomized brute-force.

The temporal evolution of the merging process for a subset of representative map pairs from Figure 5.6 is shown in Figure 5.7. Note that all algorithms eventually reach the same maximum algebraic connectivity by adding all possible edges. However, QuickConnect achieves a steeper quality gain with its selective insertion process during the exploration phase. For instance, during the merging of the map pair shown in Figure 5.7a the exploration phase inserting only 247 edges in 1.5 seconds improves the quality up to 96% while it takes around 24 and 40 seconds and 2112 and 3989 edges to reach the same level of quality for *DegreeMin* and brute-force, respectively. On the other hand, the refinement stage adds almost 15 times more edges than



(a) M_2 vs M_6 (b) M_1 vs M_4

(c) M_9 vs M_{12} (d) M_{10} vs M_7

Figure 5.7: Normalized algebraic connectivity of four pairs of merged maps as a function of time for the three algorithms are presented.

the exploration phase to only improve the connectivity by 4%. Similar trends are observed in the rest of the dataset as can be seen in Figure 5.7. These charts experimentally support the claim that the exploration stage successfully identifies the small portion of edges that substantially increase algebraic connectivity.

In Figure 5.8 we show a further analysis of the changes in algebraic connectivity for the last map merging example shown in Figure 5.7. The figure analyzes the spikes in algebraic connectivity observed during the exploration stage of QuickConnect. For each of the three major increases we show the edges identified and inserted.

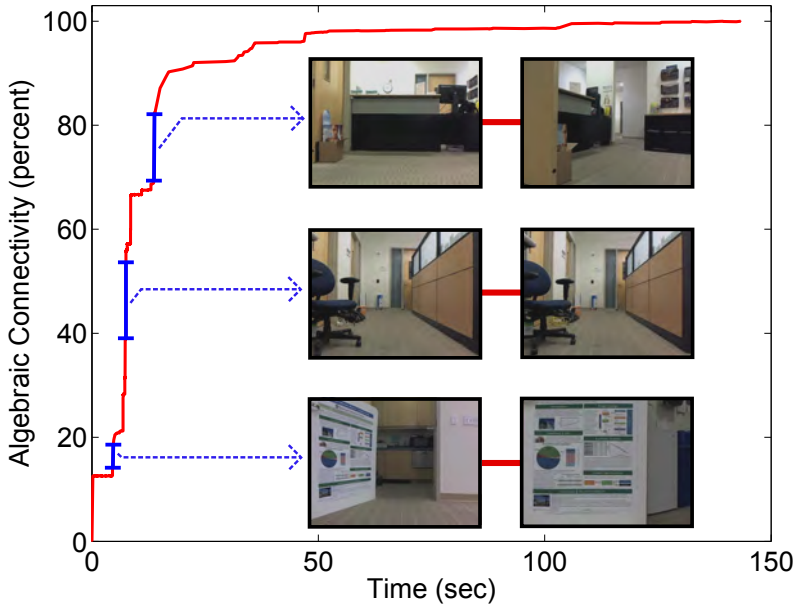
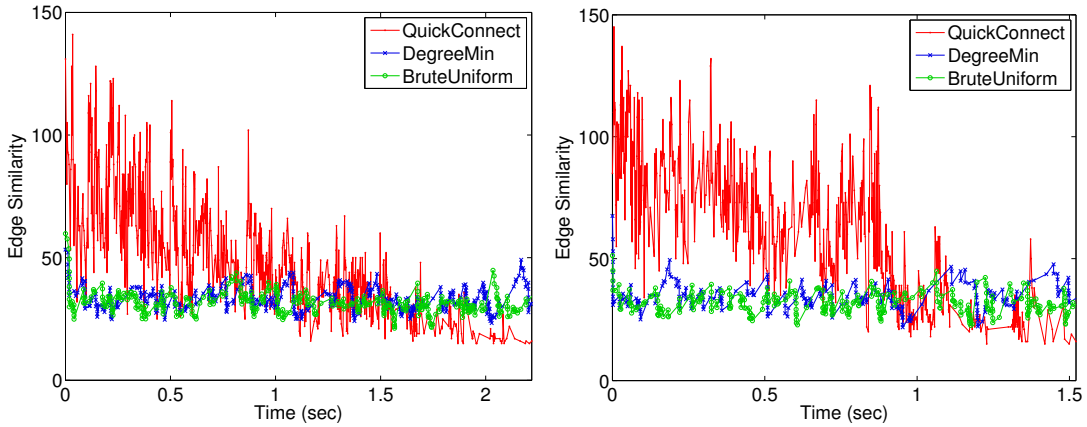


Figure 5.8: Inserted edges corresponding to three major spikes in algebraic connectivity are shown along with the vertices they connect.

It is also important to note that in its exploration stage, *QuickConnect* identifies the edges connecting the most similar images between two maps. This trend can be seen in Figure 5.9, where a closeup view of the aforementioned merging process is shown. This characteristic of the proposed algorithm is the main desired property of a compression algorithm that unifies similar vertices in both maps. Hence, this framework could easily be extended to a system that merges maps by not only adding new edges but also unifying vertices. Nevertheless, the effects of such unification on the quality and robustness of the map in terms of algebraic connectivity warrants much further investigation.

When multiple maps created by teams of robots need to to be merged, it is important to maintain the same performance displayed by the pairwise matching algorithm. Thus, we evaluated the performance of the generalized version of the proposed merging algorithm, QuickConnect-simultaneous, on the same datasets. For comparison purposes we also devised modified versions of previously presented methods: *BruteUniform-simultaneous* and *DegreeMin-simultaneous* which randomly sample vertices from the set of all vertices, V_m, uniformly and with a mass distribution inversely proportional to their vertex degree, respectively. Additionally, the standard sequential pairwise merging approach described in Section 5.3.2 is applied to all



(a) M_2 vs M_6 (b) M_1 vs M_4

Figure 5.9: Weights, i.e., similarity of connecting images, of the identified edges for two of the representative merging processes are plotted for three algorithms for the duration of the exploration phase of *QuickConnect*.

pairwise merging algorithms resulting in three new merging methods: *QuickConnect-pairwise*, *BruteUniform-pairwise*, and *DegreeMin-pairwise*. However, as mentioned in Section 5.3.2, in sequential pairwise merging the algebraic connectivity remains equal to zero until the very last map is merged. Therefore, presenting the quality of merging captured by each algorithm within 10% of total time as in Figure 5.6 would negatively affect sequential pairwise merging methods. To counter this fact, for each algorithm the time required to reach 90% of the overall quality is recorded and the results are shown in Figure 5.10. Again, for randomized methods the results are the averages of 20 runs.

As shown in Figure 5.10, QuickConnect-simultaneous, taking advantage of its parallel voting schema, creates edges between vertices from different map pairs, and therefore, outperforms all other algorithms for all merged maps. The ranking for the rest of the algorithms is not consistent. QuickConnect-pairwise in general achieves higher performance than its closest competitor DegreeMin-simultaneous with the exception of second and last group in Figure 5.10. However, its performance depends on the ratio of the duration of last merging step to the overall merging time, and cannot be generalized since until the last step the algebraic connectivity stays at zero due to the map being disconnected. Therefore, the performance of sequential pairwise merging algorithms suffers more when the number of maps to be merged increases, as in the last group where 4 maps are merged together.

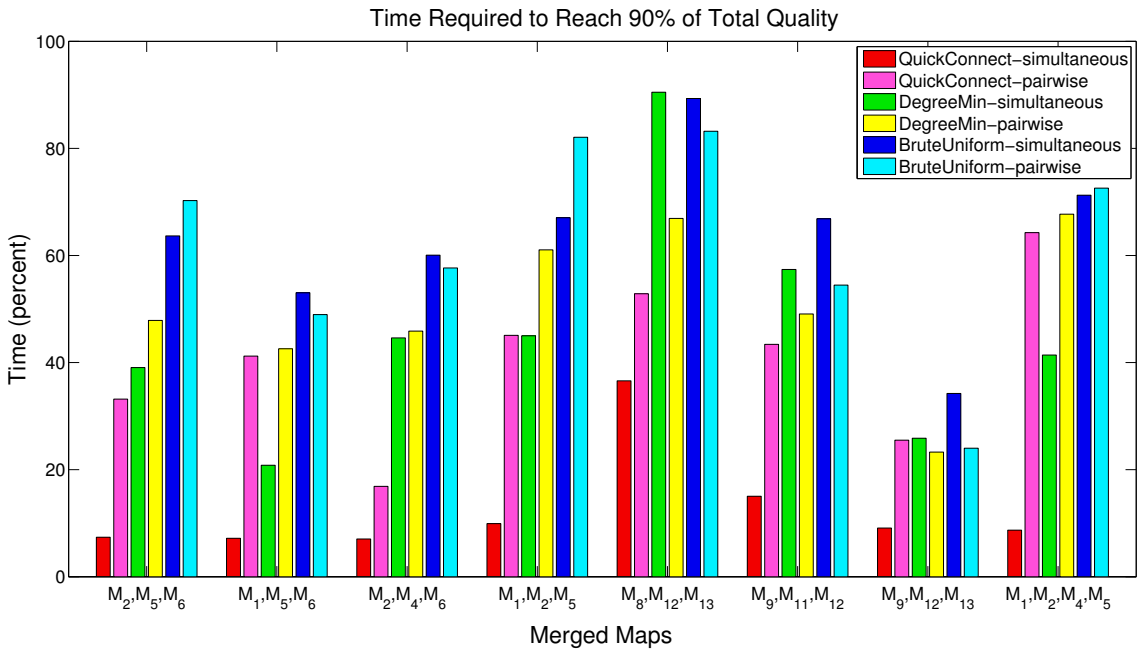


Figure 5.10: A representative selection of maps are merged together using simultaneous and pairwise merging algorithms. The cart shows the normalized time required to reach 90% of the total quality.

Furthermore, it can be noted that for some of the merged maps (4^{th}, 5^{th}, and 6^{th} group) pairwise versions of DegreeMin and BruteUniform performed better than the simultaneous versions of the same methods. The reason behind this phenomenon is that there was an isolated connection with high algebraic connectivity value, i.e., an edge between two vertices with no similarity among any of the neighboring vertices, that is discovered and created during the first pairwise merging step. Later, since this important connection is already created when this intermediate map is connected to the third map, the algebraic connectivity immediately reached very high values. On the other hand, in the simultaneous approach this edge may be identified and created anytime during the merging process which on average causes a lower quality increase comparing to the sequential pairwise merging methods.

Finally, it is important to note that we assumed that all computation is performed on one robot, but the proposed merging algorithm and the randomized methods can be easily parallelized by distributing the workload among all the robots with available computational power. Nevertheless, since all merging methods mentioned here will benefit from the same performance gain, even though the merging process

will be completed faster, the algebraic connectivity trends presented in Figure 5.7 and the relative performances of merging algorithms will not change. In other words, QuickConnect will still outperform the other two merging algorithms.

5.5 Conclusions

In this chapter we have investigated the problem of merging appearance-based maps, an underexplored topic that is very relevant in the area of multi-robot systems. Noting that there is a lack of well established criteria to evaluate the quality of a merging algorithm operating on appearance-based maps, we have put forward algebraic connectivity and explained why it is a promising criterion to assess the value of a map merging algorithm. Motivated by this metric, we have developed *QuickConnect*, an anytime algorithm that discovers important vertices and edges early on during the process. The method we have proposed has been integrated into the end-to-end system that creates appearance based maps in real-time using the bag of words approach as described in Chapter 4. The goodness of the method we proposed has been experimentally assessed by processing various maps we produced with the aforementioned method, and we verified that even if stopped early during the process, *QuickConnect* offering a convenient tradeoff yields better results when compared with the brute-force algorithm. More precisely, we observed a 10-90 ratio, i.e., by spending only 10% of the time it is possible to get about 90% of the benefits. We have also explored how pairwise merging can be extended to handle the case when multiple maps need to be combined together and introduced *QuickConnect-simultaneous*, an efficient parallel merging algorithm that overcomes the disadvantages of sequential merging approach. This more generic merging algorithm while preserving the anytime nature of its predecessor, overcomes the limitations of the brute-force method and yields the best results.

CHAPTER 6

Heterogeneous Map Merging

In the previous chapter we emphasized the importance of merging multiple maps into a single spatial model to increase spatial awareness of multi-robot systems, and proposed a map merging algorithm that is tailored to appearance-based maps and focused on their graph properties. While merging same type of maps provides valuable information especially to robots when they are the ones utilizing those maps, in situations requiring the acquisition of spatial models by a coordinated heterogeneous robot team such as urban search and rescue, patrolling, and explorations tasks, the need for heterogeneous map merging becomes apparent. Each robot member of these teams is usually manufactured differently, with different operational capabilities, sensors, and data output which makes it much harder to fuse data and acquire a single spatial model. In these tasks, robots are often used as tools to provide information to humans, who then decide the proper course of action given that information. Evidently, the ability to merge together multiple heterogeneous partial models can greatly improve the overall utility of the collected information.

The heterogeneous map merging problem has the inherent challenge that different spatial models lack, by definition, a common representation. To remedy this problem, we rely on the ubiquitous presence of WiFi signals in today's environments and assume that every robot, regardless of the type of map they produce, is equipped with a WiFi card. We believe this is a safe assumption considering that robot teams will need to communicate amongst each other and the base station. As each robot explores the environment, the Wireless Signal Strength (WSS) of all Access Points (APs) in range is recorded. This data is then used to create a WiFi map online and update it with every new WSS reading as will be described in the next section. Our heterogeneous map merging algorithm uses WiFi maps as a substrate to combine models relying on different representations. Maps of different types can then be merged into a single model by matching one portion of a map in another map using our WiFi classification-based localizer which we will present in Section 6.3. Although the algorithm we propose extends to other map types, we focus our discussion on occupancy grid and appearance-based maps. Our solution consists of three main steps. First, the overlap between the heterogeneous maps being merged is deter-

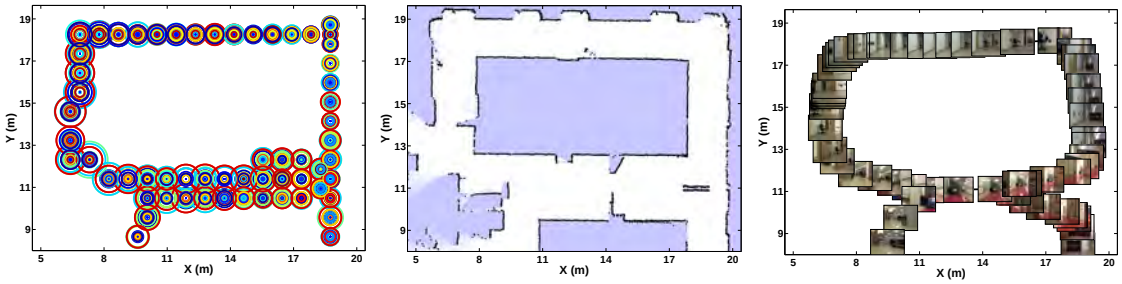
mined. Second, metric correspondences between overlapping parts are established. Third, the merging is improved by exploiting the structural properties inherent to graph-based maps. In this chapter we offer the following contributions.

- we propose a novel clustering algorithm that allows the acquisition of the WiFi map to be performed online, without having to stop the robot to collect WiFi readings or having a priori access to the environment;
- we introduce, to the best of our knowledge, the most complete and accurate WiFi localizer which we then utilize at the core of our map merging algorithm;
- we compute the amount of map overlap between multiple heterogeneous maps by casting the problem as One Class Classification;
- we develop, to the best of our knowledge, the first system capable of merging heterogeneous maps;
- we outline crucial design choices with respect to the algorithm's applicability and performance;
- we evaluate and compare numerous algorithms across different datasets.

6.1 Problem Formulation

As aforementioned, we focus on merging occupancy grid and appearance-based maps, but the approach is generic and supports merging different types of maps as long as WSS are collected during the map building process. Regardless of the map's type being considered, an observation $Z_i = [z_i^1, z_i^2, \ldots, z_i^a]$ is acquired using a WiFi card, where a denotes the total number of APs seen throughout the environment. Each z_i^k is the WSS of the k-th AP, measured in the number of decibels relative to one milliwatt (dBm), unless the AP cannot be seen from a particular location, in which case we set $z_i^k = -100$. Although we measure the signal strength as dBm, other measures such as the signal-to-noise ratio have been shown to work well for WiFi localization [95, 8]. Each observation Z_i is linked to a label $\hat{L}_i$ representing either a Cartesian point $\hat{C}_i = [X_i, Y_i]$ in the case of an occupancy grid map or an image I_i for an appearance-based map. All acquired observations and their labels are then collected into a set $\hat{T} = \cup_{i=1}^{m} \{\hat{L}_i, Z_i\}$ where m is the total number of observations.

In an attempt to model the noise in WSS readings and increase the robustness of the algorithm, we partition the m WiFi readings using their labels $\hat{\mathbf{L}} =$



(a) Wifi map (b) Occupancy grid map (c) Appearance-based map

Figure 6.1: Illustrative examples for the three different types of maps presented in this paper. a) WiFi map: illustrative representation of the WiFi readings for 57 APs, superimposed on the map. The size of each ring represents the AP's signal strength (the bigger, the better) and each color indicates a unique AP. b) occupancy grid map c) appearance-based map

$\{\hat{L}_1, \hat{L}_2, \ldots, \hat{L}_m\}$ into c clusters $\mathbf{L} = \{L_1, L_2, \ldots, L_c\}$ with $(c \leq m)$. In appearance-based maps, labels that represent vertices in the graph need to be clustered. For each image $\hat{L}_i = I_i$, a cluster labeled as $L_i = \hat{L}_i$ is created $(c = m)$ centering at the vertex it corresponds to and its observation Z_i along with all observations from its 1-hop neighbors are assigned to this cluster $\mathbf{Z_i} = \{Z_i \cup Z_j \;\; \forall_j \mid e_{ij} \in E\}$. Consequently, each cluster contains WiFi readings collected from locations where similar images can be captured. On the other hand, for occupancy grid maps, WiFi observations are clustered based on their Cartesian coordinates using the clustering algorithm described in the next section.

For each cluster we store its label L_i along with a vector of observations located inside the cluster $\mathbf{Z_i} = \{Z_i^1, Z_i^2, \ldots, Z_i^{s_i}\}$ where s_i is the total number of observations inside cluster i. Put differently, s_i observations are made for cluster i, whose label is L_i. It is worthwhile to note that the clusters may not and do not need to be uniform (i.e., s_i is not necessarily equal to s_j $\forall_{i,j} : i \neq j$). In other words, the number of observations for one cluster can be different than the number of observations for another cluster. The WiFi map can mathematically be represented as a set T of observations and their labels, $T = \cup_{i=1}^{c}\{L_i, \mathbf{Z_i}\}$. Figure 6.1 shows examples of the different types of maps presented herein.

Finally, we define the heterogeneous map merging problem as follows. Given two maps M_1 and M_2, their corresponding WiFi maps T_1 and T_2 with the list of corresponding labels $\mathbf{L_1} = \{L_1^1, L_2^1, \ldots, L_{c_1}^1\}$ and $\mathbf{L_2} = \{L_1^2, L_2^2, \ldots, L_{c_2}^2\}$ where c_1 and c_2 are the number of clusters in T_1 and T_2, determine the mapping $f : \{\mathbf{L_1} \cup \emptyset\} \leftrightarrow$

$\{\mathbf{L_2} \cup \emptyset\}$ so that each label in $\mathbf{L_1}$ matches a label in $\mathbf{L_2}$. For the sake of simplicity, the problem statement considers only the case where two maps are merged and, without loss of generality, we describe the process for merging an occupancy grid map $M_1 = M_{OCC}$ with an appearance-based map $M_2 = M_{APP}$. The idea can naturally be extended to multiple maps by merging them in pairs, other types of map (e.g., topological), and identical map types (e.g., merging two appearance-based maps). In the case considered in this chapter, merging these two maps narrows down to assigning a Cartesian coordinate to each image in the appearance-based map using the WiFi localization technique which will be described in Section 6.3. The merged map will consequently consist of images overlaid on the occupancy grid map with edges connecting the similar ones. Please note that the list of labels extracted from the WiFi maps are extended with the empty set. The reason is that the maps do not necessarily fully overlap and, therefore, some labels in one map do not necessarily match a label in the other map.

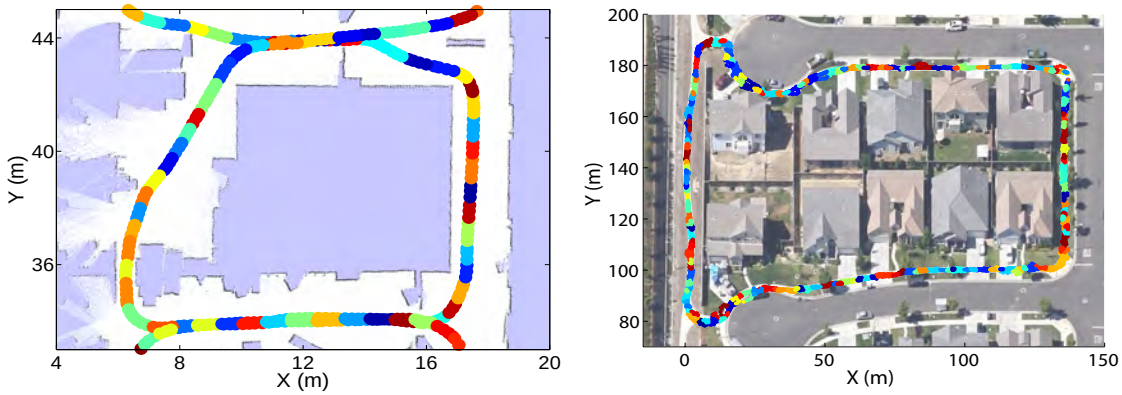
6.2 Clustering

Focusing on partitioning WiFi readings with Cartesian labels we need a clustering algorithm that groups spatially close points together which then will be used by the WiFi localization algorithm to determine matching labels. One of the most common clustering methods is k-means [102]. This method aims at creating optimal cluster boundaries such that each cluster contains all the points closest to its centroid. This methodology fails to address some vital aspects required for successful WiFi signal classification. First, the algorithm does not guarantee a minimum number of points within each cluster. The number of observations per location encompasses an important tradeoff since the higher the s_i, the better the noise model but longer the mapping process. The minimum number of observations per location s_{min} should not be less than a preset value in order to reach the targeted average classification error. Furthermore, we do not want observations acquired at locations that are far away from each other to be grouped in the same cluster. By increasing the number of clusters k the average cluster diameter can be decreased. The maximum cluster diameter d_{max}, however, cannot be controlled and depends on several other parameters such as the initialization method and data distribution. The total number of clusters c is another parameter that is crucial for successful localization, hence successful merging. Evidently, the number of clusters is proportional to the size of the explored area and the WiFi readings' density. It also defines the average distance between the clusters' centers, which has a direct effect on the localization accuracy that we analyze in Section 6.3.1.

The non-uniform binary split algorithm [146] (also known as bisecting divisive partitioning or hierarchical clustering) addresses these issues at the expense of optimality. Specifically, it is possible for the binary split algorithm to create clusters in which some points are actually closer to a neighboring centroid. The method starts with one cluster containing all of the data and recursively bipartitions the cluster with largest distortion into two sub-clusters until the required number of clusters is reached. The method essentially structures clustering into a hierarchical binary tree. Our modified version of the algorithm uses k-means clustering with $k = 2$ to split the data into two partitions. More importantly, instead of fixing the final number of clusters, we define a stopping condition for splitting a branch in the binary tree. According to this condition, we split a cluster c into two clusters c_1 and c_2 only if its diameter is greater than d_{max}. If both new clusters contain at least s_{min} points, they take their place in the binary tree and recursive partitioning continues. Otherwise, the new clusters c_1 and c_2 are ignored, the original cluster is kept, and the corresponding branch in the tree stops splitting. These conditions guarantee that after the final iteration of the algorithm we have at least s_{min} observations in each cluster and the diameter of a cluster only exceeds d_{max} in parts of the environment where not enough WSS readings were acquired.

Even though the algorithm is fast, taking less than 4 seconds to cluster over 1000 points, it is an offline clustering algorithm by design, i.e., it requires all observations available before starting the partitioning process. More importantly, observations acquired after the clustering is performed and WiFi map is created need to be processed and placed into appropriate clusters. These new observations need to be clustered without altering the cluster assignments of existing observations that have already been used by WiFi classification algorithm. Hence, we need an online clustering algorithm that makes hard decisions with respect to the clusters' boundaries. In other words, once a boundary is set between two clusters, it cannot be modified and the cluster is *fixed*.

Our online clustering algorithm is designed around this principle and is built based on the idea of growing multiple clusters until they encompass enough observations, in which case the clusters are fixed. Consequently, each cluster can be in one of two states describing whether the cluster's boundary is *fixed* or *growing*. For each new Cartesian-observation pair $\hat{C}_i, Z_i$, we identify the centroid C_p of its closest cluster p and compute the Euclidean distance g between $\hat{C}_i$ and C_p. There are two cases to take into account, depending on the cluster's state. First, if cluster c is *fixed* and the distance g is less than a pre-defined maximum radius g_{max}, then the observation Z_i is directly added to cluster c. Otherwise (if $g \geq g_{max}$), a new *growing* cluster is started with observation Z_i. Second, if cluster p is *growing* the observation Z_i



(a) Indoor (b) Outdoor

Figure 6.2: Illustrative representation of the clustering algorithm in indoor and outdoor environments, superimposed on corresponding occupancy grid map and Google map, respectively. The different clusters are represented by different colors and each encompass a varying number of WiFi observations.

is added to p. The *growing* clusters are converted to *fixed* clusters by performing a binary split on them. More specifically, *growing* clusters are split using k-means clustering with $k = 2$, resulting into two clusters c_1 and c_2. Both new clusters are *fixed* if they each encompass at least s_{min} observations within a radius of g_{max} from the cluster's center. When either of those conditions fail, the new clusters c_1 and c_2 are ignored and the original cluster p keeps *growing*. This algorithm guarantees that each *growing* cluster will have at least a minimum number of observations per cluster s_{min}. Figure 6.2 shows a visual representation of the resulting clusters for a path followed in indoor and outdoor environments.

6.3 WiFi Localization and Mapping

At the core of our heterogeneous map merging algorithm lies the ability to match labels. In other words, given a label L_i^2 we would like to find a matching label, L_j^1 from the other map using corresponding observations, i.e., WiFi readings. This problem is known as WiFi localization and as presented in Section 2.4 localization problems have been extensively studied and a variety of solutions have been proposed, each assuming different robotic platforms and scenarios. Given the sensory constraints imposed on robots due to various reasons such as weight, space, or budgetary limitations, WiFi localization solutions exploit wireless signals from APs that have become omnipresent

in today's urban infrastructure. In general, wireless signals are notoriously difficult to work with due to non-linear fading, obstructions, and multipath effects, but WiFi localization algorithm offers its share of advantages. Indeed, WiFi APs are uniquely identifiable, can be used indoor or outdoor, and are already part of the environment in which we would like to localize. Additionally, transmitted signals are available to anyone within range, allowing robots to exploit APs without ever connecting to them. The WiFi localization problem has seen some limited success over the last decade, but the lack of comparisons between solutions and strict constraints imposed by previously published algorithms render its application to real-world scenarios inadequate. In particular, the approaches are always offline, requiring earlier access to the environment for the WiFi map to be determined, and the robot often has to stop when collecting WiFi signal strengths at discrete locations, contributing to the reason the approaches are offline. From map merging perspective, offline algorithms require the robots to first stop and build their WiFi maps before attempting to merge them together, eventually delaying the creation of the merged map. In this section, after describing how WiFi maps are built, we present our solutions for these outstanding issues, providing the most complete and accurate WiFi localizer to date. More specifically, we offer the following contributions

- we localize using WiFi by casting the localization as a machine learning classification problem, the learning of which is solved in real-time thanks to an Online Random Forest;
- we contrast our method with six published algorithms, accounting for both real-time and offline performances;
- we propose a novel regression algorithm that builds upon the best classification algorithm;
- we develop an end-to-end Monte Carlo Localization (MCL) algorithm built upon the Online Random Forest and supported with odometry for robustness;
- we outline crucial design choices with respect to the algorithm's applicability to unknown environments and real-time performance;
- we establish the power of our WiFi localizer by assessing its strengths across different datasets that cover kilometers of indoor and outdoor environments.

Some of the results presented here have been published in [8].

6.3.1 Classification

6.3.1.1 Description

Figure 6.1a gives a good qualitative indication that it is possible to differentiate between different locations by only considering signal strengths received from APs. We start developing the WiFi localizer by casting it as a machine learning classification problem. Mathematically, given a WiFi map T and an observation $Z = [z^1, \ldots, z^a]$, we produce a function $f : Z \rightarrow p$ by using T as training data. Function f takes Z and returns the location p which corresponds to label L_p in T. We note that casting the problem in such manner is a simplification, since a robot needs to be positioned exactly at one of the locations in the training data in order for f to return the exact coordinate C_p (i.e., classification does not perform interpolation or regression). Solving the classification problem is nevertheless an important step since the effectiveness of various algorithms can be evaluated and applied to build a better localizer, as will be shown in the next sections. Computing f from T can be achieved using different techniques and to the best of our knowledge, no comparison has been made in the literature to find the best performing algorithm. Consequently, we implement a total of seven algorithms: Nearest Neighbor Search, Gaussian Model, Gaussian Processes, Support Vector Machine, Decision Tree, Random Forest, and Online Random Forest. Next we present each algorithm in the context of our problem's definition.

6.3.1.2 Classification Algorithms

Nearest Neighbor Search[7]: The nearest neighbor search does not require any processing of the training data, T. Instead, the distance between all the points in T and a new observation Z is computed, yielding to m distances. The location p that the observation returning the minimum distance belongs to is then assigned as the location of the new observation. The only necessary decision for this algorithm involves the choice of one of the numerous proposed distance formulas (e.g., Euclidean, Mahalanobis, City Block, Chebychev, Cosine). We use Euclidean distance to rigorously follow the algorithm presented in [7].

Gaussian Model[76, 95, 13]: The Gaussian model technique, proposed in the robotics literature, attempts to model the inherent noise of signal strength readings through a Gaussian distribution. For each location p and for each AP k the mean and standard deviation of the signal strength readings are computed from all observations acquired at that location, yielding to μ_p^k and σ_p^k, respectively. This means that a total of $c \times a$ Gaussian distributions are calculated. The location, p, of

a new observation, Z, is derived using the Gaussian's probability density function:

$$\arg\max_{p} \left[\prod_{k=1}^{a} \frac{1}{\sigma_p^k \sqrt{2\pi}} \exp\left(-\frac{(z^k - \mu_p^k)^2}{2(\sigma_p^k)^2} \right) \right]$$

Gaussian Processes[48, 38]: Conversely to the Gaussian Model, which describes a probability distribution in the space of signal strength readings, z_i^k, a Gaussian Process defines the probability distribution in the space of Cartesian positions, C_i. In other words, the Gaussian Model and Gaussian Process algorithms are different in that they essentially compute $P(C|Z)$ and $P(Z|C)$, respectively. Instead of directly computing the means and covariances for the Gaussian distributions, a kernel function κ is exploited to compute the covariances. Although there are many potential choices for the covariance kernel function, we replicate the works of Ferris et al. [48] and Duvallet et al. [38] by using the squared exponential kernel

$$\kappa(C_i, C_j) = \sigma_f^2 \exp\left(-\frac{|C_i - C_j|^2}{2l^2} \right)$$

where C_i and C_j are two Cartesian locations, σ_f^2 is the signal variance, and l is a scaling factor that determines the correlation's strength between C_i and C_j. In some sense, the kernel function determines the correlation between values located at different points. Assuming that the signal strength observations have additive noise represented by a zero-meaned Gaussian distribution whose variance is σ_n^2, we can compute the covariance matrix using the formula

$$cov(z_i^k, z_j^k) = \begin{cases} \kappa(C_i, C_j) + \sigma_n^2 & \text{if } i = j \\ \kappa(C_i, C_j) & \text{if } i \neq j \end{cases}$$

In matrix form, the covariance equation can be re-written as

$$cov(\mathbf{z}^k) = K + \sigma_n^2 I$$

where $\mathbf{z}^k$ is the linear combination of all signal strength readings for AP k and across all clusters, K is the matrix representation of the kernel function evaluated for all i and j, and I is the identity matrix. As the covariance equations suggest, a separate Gaussian Process needs to be computed for each AP k. When used to solve a classification problem, the location $C \in T$ corresponding to a new observation Z is computed from

$$\begin{aligned} C &= \arg\max_{C} \left[\prod_{k=1}^{a} \frac{1}{\sigma_c^k \sqrt{2\pi}} \exp\left(-\frac{(z^k - \mu_c^k)^2}{2(\sigma_c^k)^2} \right) \right] \\ \mu_c^k &= \mathbf{k_c}^T cov(\mathbf{z}^k)^{-1} \mathbf{z}^k \\ \sigma_c^k &= \kappa(C, C) - \mathbf{k_c}^T cov(\mathbf{z}^k)^{-1} \mathbf{k_c} \end{aligned}$$

where $\mathbf{k}_c$ is a vector of covariances between location C and each location C_i inside the training data T. Although the aforementioned equations solve a classification problem, Gaussian Processes are better suited for regression problems and fit very well with MCL algorithms. Indeed, an hypothesized robot location C (not necessarily in T) accompanied with an observation Z can probabilistically be evaluated using

$$P(Z|C) = \prod_{k=1}^{a} \frac{1}{\sigma_c^k \sqrt{2\pi}} \exp\left(-\frac{(z^k - \mu_c^k)^2}{2(\sigma_c^k)^2}\right).$$

It is worthwhile to note that Gaussian Processes rely on three parameters (i.e., σ_f^2, l, and σ_n^2) that greatly affect the prediction accuracies of the algorithm. Fortunately, these parameters can be learned directly from the training data T by maximizing the log likelihood of the predictions, using the trained observations and conditioned on the three parameters [141].

Support Vector Machine[161]: Support vector machines work by constructing a set of hyperplanes in such a way that they perfectly divide two data classes (i.e., they perform binary classification). Generating the hyperplanes is essentially an optimization problem that maximizes the distance between the hyperplanes and the nearest training point of either class. Although initially designed as a linear classifier, the usage of kernels (e.g., polynomial, Gaussian) enables non-linear classification. Since our training data T comes from c distinct locations, we use a support vector machine variant that works for multiple classes. We chose the one-versus-one approach due to its structure that allows us to train for new classes online. In this approach we build binary classifiers which distinguish between every pair of classes. This essentially creates a set of $c \times (c-1)$ binary classification problems, each of which is solved using the standard support vector machine algorithm with a Gaussian kernel. Once all the support vector machines are trained, we can localize given a new observation Z by evaluating its high-dimensional position with respect to the hyperplanes, for each support vector machine p. The class p is chosen by the support vector machine that classifies Z with the greatest distance from its hyperplanes.

Decision Tree[19]: A decision tree is a binary tree constructed automatically from the training data T. Each node of the tree corresponds to a decision made on one of the input parameters, z_p^k, that divides the node's data into two new subsets, one for each of the node's sub-trees, in such a way that the same target variables, p, are in the same subsets. The process is iterated in a top-down structure, working from the root (whose data subset is T) down to the leaves, and stops when each node's data subset contains one and only one target variable or when adding new nodes becomes ineffective. Various formulas have been proposed to compute the "best" partitioning

of a node's data subset, the most popular of which being the Gini coefficient, the twoing rule, and the information gain. After experimental assessments, we found the choice of partitioning criterion insignificant in terms of localization accuracy and use, arbitrarily, the Gini coefficient. In order to classify a new observation Z, the appropriate parameter z^i is compared at each node, the decision of which dictates which branch is taken - a step that is repeated until a leaf is reached. The target variable p at the leaf is the location of the robot.

Random Forest[18]: Random forests are an ensemble of decision trees built to reduce over fitting behaviors often observed in single decision trees. This is achieved by creating a set of F trees, as described in the previous paragraph, each with a different starting dataset, T', selected randomly with replacement from the full training data T. An additional difference comes from the node splitting process, which is performed by considering a random set of q input parameters as opposed to all of them. The partitioning criterion is still used, but only the q randomly selected parameters are considered. In order to classify a new observation Z, it is processed by each of the F trees, resulting in F output variables, some of which may be the same. In some sense, each decision tree in the forest votes for an output variable and the one with the most votes is chosen as the location of the robot. The number of trees d encompasses a tradeoff between speed and accuracy.

Online Random Forest: The RF unfortunately assumes that the entire training data T is available a-priori, an assumption that does not hold for real-time WiFi localization. Similarly to [143], we modify the RF algorithm, creating an Online Random Forest (ORF) that learns from incremental updates to the training data T, in an online fashion. We describe our ORF algorithm with respect to its offline counter-part. Specifically, a RF is an ensemble of F decision trees built to reduce over-fitting behaviors often observed in single decision trees. Each tree f is created from a subset T', selected randomly with replacement from the full training data T. Every time a new node is created, a random set of decisions $d(Z) > \theta$ are created, the best one of which is used to split the node. The process of generating decisions is the same for RF and ORF. As shown in Algorithm 5 we update the trees every time we receive a new observation-label pair (Z, L_i) where each label is linked to the Cartesian coordinate of corresponding cluster center.

Each tree is updated one by one (line 1) and a Poisson distribution with parameter $\lambda = 1$ models the incremental arrival of new data (line 2). The Poisson distribution essentially mimics the random selection with replacement of the RF training data. The integer k is drawn from the Poisson distribution and dictates the number of times that the tree will take into account the observation-label pair (Z, L_i). Please note

Algorithm 5 ORF-Update(Z, L_i)

```
1: for f ← 1 to F do
2:   for k ← 1 to Poisson(1) do
3:     l = navigateToLeaf(Z)
4:     updateLeaf(l, Z, L_i)
5:     if shouldSplit(l) then
6:       arg max_{d∈D} △Gini(l, d)
7:       createChildren(l, d)
8:     end if
9:   end for
10:  if k ← 0 then OOBE_t ← updateOOBE(t, Z, L_i)
11:  if x drawn from Bernoulli(OOBE_t) then rebuildTree(t)
12: end for
```

that k can be 0 or greater than 1, mimicking an observation that was not sampled or sampled multiple times, respectively. The choice of a Poisson distribution comes from Oza et al. [134] who proved convergence to the offline approach. Since each node does not have access to all of the training data, we use the observation Z to navigate through the tree, reaching a leaf l (line 3). We add the new observation-label pair Z-L_i to the leaf l (line 4). So far, the algorithm simply updates the amount of data encompassed by each node of each tree in the ORF. We decide when the leaves should split using the *shouldSplit* function (line 5), whose pseudo-code is given in Algorithm 6. When choosing whether or not to split a node, we take into account two parameters. First, the leaf needs to possess observations representing at least two different clusters. Since we know that the clusters will come in sequentially from the Online Clustering, we want to make sure that we do not split a node whose data only account for one cluster. Second, we only want to split the node if the splitting criterion gain with respect to a decision d, $\triangle\text{Gini}(l, d)$, has increased "significantly". The extent to which the gain should increase is dictated by parameter α. If the choice of splitting the node is made (line 5), the best decision $d \in D$ is determined (line 6) and exploited to split the leaf, creating left and right children nodes (line 7). The newly-created children are imparted with the data of their parent node so that they can classify directly, without new observations.

In order to make the ORF more robust, we add the capability of removing and reconstructing bad trees in the forest. In other words, we want to add the capability of unlearning old information that might not reflect any longer the overall data distribution. To do so, we exploit the observation-label pairs that were not trained

Algorithm 6 shouldSplit(l)

1: $b \leftarrow$ *true*
2: **if** difClusters(l) < 2 **then** $b \leftarrow$ *false*
3: **if** $\triangle$Gini(l, d) $< \alpha \quad \forall d \in D$ **then** $b \leftarrow$ *false*
4: **return** b

as part of the tree, called Out-Of-Box (OOB) samples. From these samples, which only occur when $k = 0$, we can compute the OOB Error (OOBE) for each tree (line 10). In some sense, the OOBE, which is analogous to cross validation, supplies a quantitative measure of a tree's performance. By evaluating this measure, we can infer when a tree becomes less and less useful, in which case we destroy it and rebuild it from scratch using the generic RF tree building algorithm. To decide whether or not a tree should be discarded (line 11), we sample x from a Bernoulli distribution whose parameter p is set to the OOBE. The sample x is either 1 or 0 and indicates if the tree should be discarded. The Bernoulli distribution essentially models the probability of discarding a tree, where the probability is inversely proportional to the OOBE. It is important to note that reconstructing one tree in the entire forest does not significantly damage the algorithm's performance, yet allows the forest to adapt over time.

6.3.1.3 Results

The experimental results of the WiFi localizer cast as a classification problem greatly influenced our algorithmic design choices for the end-to-end WiFi localization algorithm. Therefore, we present the results before continuing with the algorithm's description. In addition to the seven aforementioned algorithms, we run experiments on a set of six different datasets, each representing distinctive environments and operating conditions. Table 6.1 provides detailed information for each of the six datasets. As can be seen from the table, we acquired two datasets, UCM and Merced, representing indoor and outdoor environments. The remaining four datasets, RICE, CMU, USC, and HKUST, were made publicly available by the authors of [95], [27], [76], and [39], respectively. The datasets differ tremendously in terms of environment's type (e.g., indoor, outdoor), WiFi map acquisition process (e.g., offline, online), mapping agent (e.g., robot, human), localization technique used to infer the Cartesian coordinates C_i (e.g., SLAM, GPS, manual labeling, laser-based MCL), wireless signal metric (e.g., dBm, SNR), approximate size of the environment covered, total number of clusters (c), average distance between clusters, minimum number of observations

per cluster, and total number of APs discovered (a). As a group, these datasets exemplify a wide range of operating conditions and provide the most extensive set of WiFi localization experiments to date. Since this set of experimental results is purely meant to evaluate the Online Random Forest against other algorithms and across different datasets, all of the data presented in this section was acquired by a single robot and processed offline. The more realistic scenarios involving robots mapping and localizing in real-time are presented in the next section.

	UCM	**M**erced	**R**ICE	**C**MU	**U**SC	**H**KUST
Source	In-House	In-House	Public	Public	Public	Public
Environment	Indoor	Outdoor	Indoor	Indoor	Indoor	Indoor
Map Acquisition	Offline	Online	Offline	Offline	Online[1]	Offline
Mapping Agent	Robot	Robot	Human	Human	Robot	Human
Localization	SLAM	GPS	Manual	Manual	MCL	Manual
Signal Metric	dBm	dBm	SNR	SNR	dBm	dBm
Length Covered	150m	1.5km	1.5km	300m	100m	100m
No. of Clusters, p	156	720	506	209	120	106
Clusters' Distance	0.89m	1.96m	3.33m	1.58m	0.69m	1.00m
$\min s_i \in A_i$	20	7	100	32	20	25
No. of APs, a	48	231	57	152	4	144

Table 6.1: Detailed information about the datasets used for the classification experiments. Each column corresponds to a dataset, with each row representing a specific characteristic of the dataset.

We start by analyzing each algorithm's accuracy as a function of the number of readings taken at each location, s, a plot of which is shown in Figure 6.3. The training and classification data are randomly sampled 50 different times from UCM dataset for each experiment in order to remove any potential bias from a single sample. Presented results are averages of those 50 samples and we omit error bars in our graphs since the results' standard deviation were all similar and insignificant. Since our goal is to ultimately be able to map unknown environments in real-time, the data acquisition needs to be efficient. It takes on average 411ms to get signal strength readings from all the APs in range so we want to minimize s. As expected, Figure 6.3 shows a tradeoff between the algorithms' localization accuracy and the number of readings used during the training phase. Since we want to limit the time it takes to gather the training data, setting the number of readings per location to 3 ($s = 3$) provides a good compromise between speed and accuracy, especially since the graph shows an horizontal asymptote starting at or around that point for the best

algorithms. As such, the rest of the results presented in the paper will be performed using 3 readings per location. All the algorithms, except the Decision Tree, reached an average classification error below 1m with 3 readings per location. Furthermore, the Gaussian based techniques, as expected, performed badly when 1 reading is used per location since in that case the training data does not contain enough variance to model, i.e., $\sigma = 0$.

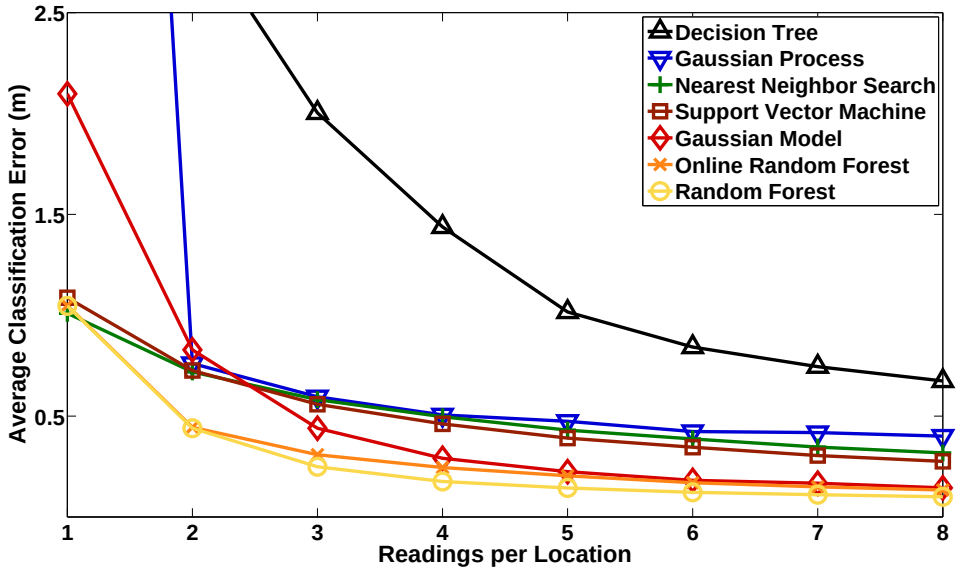


Figure 6.3: Average classification error, in meters, for each classification algorithm as a function of increasing number of readings per location.

Figure 6.4 shows the cumulative probability of classifying a location within the error margin indicated on the x-axis. This figure corroborates the findings of Figure 6.3, showing that the best algorithm is the Random Forest, which had never been exploited in the context of WiFi localization. Moreover, the Random Forest can localize a new observation, Z, to the exact location (i.e., zero margin of error) 88.58% of the time.

The average classification error of each algorithm and each dataset is shown in Figure 6.5. As can be seen from the figure, both Random Forests are consistently more accurate than the other algorithms, regardless of the dataset and, consequently, environment used. The Online Random Forest is only marginally worst than its offline counterpart; 7.39% on average. Compared with the other algorithms, the Online Random Forest is, on average, 15.96%, 103.24%, 301.26%, 475.99%, and 1400.71% better than the previously published WiFi localization algorithms (Gaussian Model,

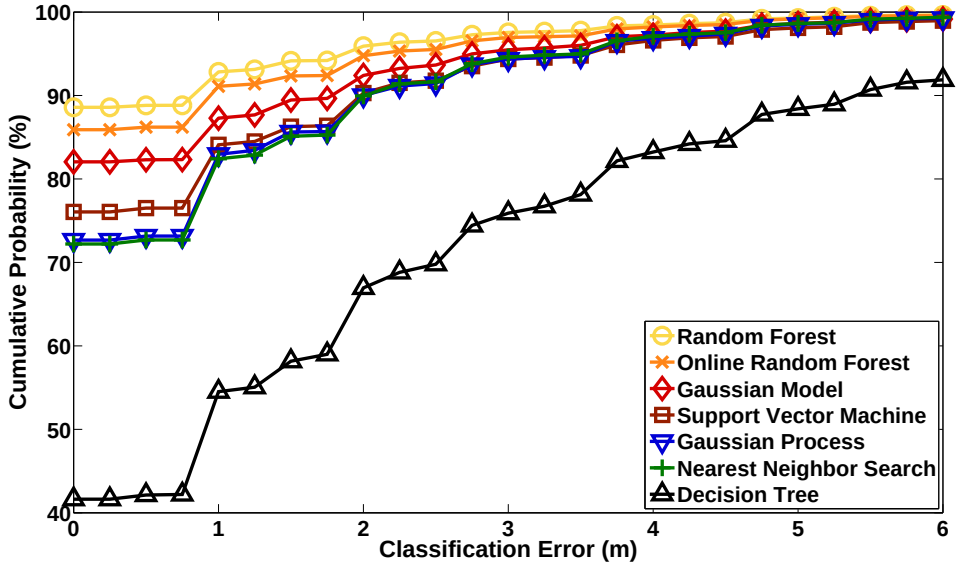


Figure 6.4: Cumulative probability of classification with respect to the error margin.

Nearest Neighbor Search, Support Vector Machine, Gaussian Process, and Decision Tree, respectively). It is also worth mentioning that the Rice dataset uses signal to interference ratios for their observations as opposed to signal strengths. Although both measures are loosely related, this indicates that the classification algorithms are both general and robust. These important findings indicate that the proposed Online Random Forest is data- and environment-independent and that we are not biased or over-fitting our datasets. Please note that for the indoor environments (i.e., UC Merced and Rice), the average classification accuracy is proportional to the average distance between clusters. As seen in Figure 6.5, the outdoor environment is the most challenging, due to the large distances between APs, prominent multi-path effects, and size of the environment.

Since we are focusing on online WiFi map building, we briefly describe each method with respect to its online capabilities. We "convert" the Random Forest to an online version by re-training it from scratch every time a new cluster is created. This is evidently time consuming, as will be shown in this section, but provides a very accurate baseline to compare against. The multi-class SVM can be trained in a one-against-one or one-against-all methodology. Since the one-against-all technique does not allow for online learning (unless training is performed from scratch for every new cluster), we use the one-against-one technique. Assuming we have discovered c clusters that were already trained one-against-one, a new cluster will require training

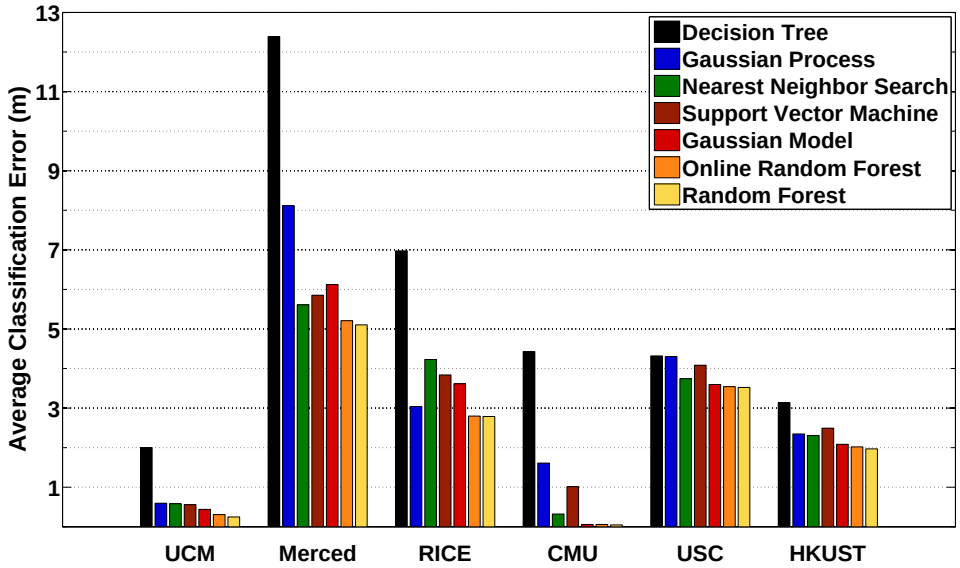


Figure 6.5: Average classification error, in meters, for each classification algorithm executed on the UCM, Merced, RICE, CMU, USC, and HKUST datasets.

c new SVMs. Evidently, the number of observations grows over time and so does the SVM training time. The Gaussian Model and Nearest Neighbor Search are already capable of running online. Since the Gaussian Model computes a set of Gaussian distributions for each cluster independently of any other clusters, we can simply wait to get a new cluster before computing its distribution. The Nearest Neighbor Search does not have a training phase and is consequently online by default.

The time that each algorithm takes to incrementally add a new cluster into the WiFi map is computed and the resulting training times are shown in Figure 6.6, where the number of clusters are progressively increased from 20 to 100 as the robot explores the map. It is important to note that 100 clusters accounts for a small map covering an area ranging from 100 to 300 square meters and that is the reason why the training process takes this little time in this example. Figure 6.6 does not include the Nearest Neighbor Search since it does not require a training phase. Similarly, training time for the Gaussian Process is omitted since it takes too long to train for online applications (i.e., an average of 4.75 minutes for 100 clusters). The figure clearly shows that both the Random Forest and Support Vector Machine take too much time to train, especially since they grow linearly and exponentially, respectively, as the number of clusters increases. On the other hand, the Gaussian Model and Online Random Forest add new clusters to a trained map in constant time, with speeds of 27.4ms and 2.5ms, respectively.

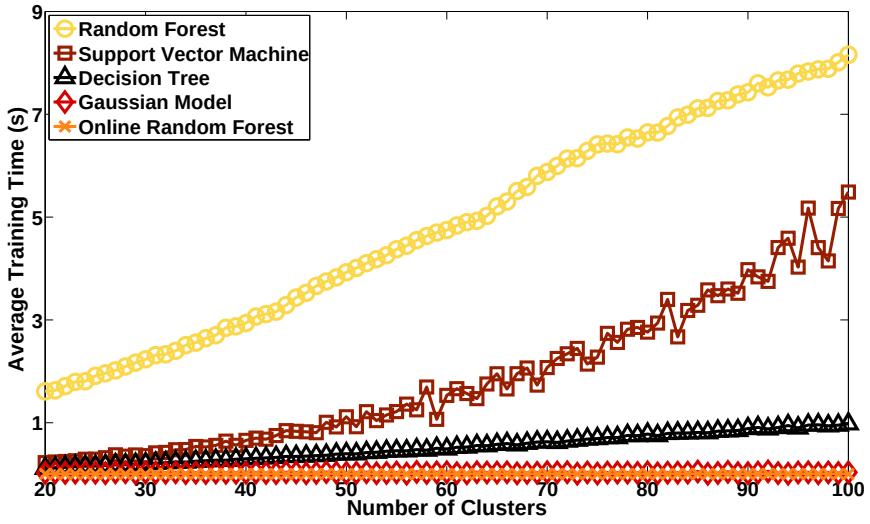


Figure 6.6: Average training time, in seconds, as the clusters increase during exploration. The Nearest Neighbor Search is omitted since it does not need training. The Gaussian Process is also omitted since it takes too long: 4.75 minutes on average for 100 clusters.

Not only is the training time important for real-world scenarios, but so is the classification time. Indeed, it does not matter how fast WiFi maps can be created, if they take minutes to be exploited. Figure 6.7 shows, for each algorithm, the average time to classify 10 observations. Both the Gaussian Process and Support Vector Machine algorithms are omitted due to their slow classification speeds of 3.05s and 9.07s, respectively, for maps containing 100 clusters. The trends for the remaining algorithms are approximately constant, except for the Gaussian Model which is linear. Once again, the Online Random Forest provides the fastest option, along with the Nearest Neighbor Search.

6.3.2 Regresion

Although the results of the classification algorithms are very encouraging, they do not depict a real world scenario, where locations explored by one robot, upon which the WiFi map is created, will surely be different than those explored by other robots. Consequently, we re-cast the WiFi localizer as a regression problem, where some inference is performed to generate Cartesian coordinates for locations that are not encompassed in the training data. Although typical off-the-shelf regression algorithms (e.g., Neural Network, Radial Basis Functions, Support Vector Regression)

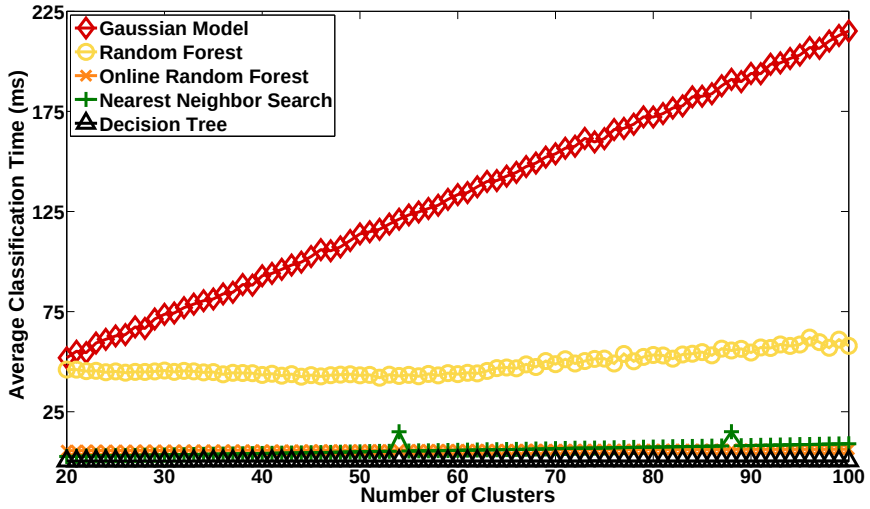


Figure 6.7: Average time, in milliseconds, it takes to classify 10 observations, as the number of clusters increases during robot exploration. The Gaussian Process and Support Vector Machine algorithms are omitted because they take too much time relative to the other algorithms (i.e., 3.05 and 9.07 seconds, on average, for a map size of 100 clusters).

might seem like a good choice initially, they get corrupted by the nature of our training data, the majority of which depicts unseen APs (i.e., $z_p^a = -100$). In addition, it would be wise to exploit the good results exhibited by the classification algorithms. We consequently design our own regression algorithm that builds upon the best online classification algorithm: the Online Random Forest. In addition to providing one of the best classification results, the Online Random Forest is appealing due to its voting scheme, which can be interpreted as $P(p|Z)$, for each $p \in T$.

Our regression algorithm is based on a Gaussian Mixture Model (GMM) [121] described in Algorithm 7. The GMM is introduced to non-linearly propagate, in the two-dimensional Cartesian space, results acquired from the Online Random Forest. More specifically, we build a GMM comprised of c (number of clusters) mixture components (line 1). Each mixture component corresponds to a cluster p and is constructed from a Gaussian distribution with a mean $\mu(p)$ (line 2), corresponding to the Cartesian coordinates of the cluster center of p, covariance $\Sigma(p)$ (line 3), and mixture weights $\phi(p)$ that are acquired directly from the Online Random Forest's voting scheme (line 4). Line 4 highlights the key difference between the classification and regression methodologies, which arises from the fact that taking the mode of the Online Random Forest's results, as is done in classification, discards valuable

information that is instead exploited in our regression algorithm. In some sense, the mixture weights are proportional to the Online Random Forest's belief of being at location p given the observation Z (i.e., $\phi(p) = P(p|Z)$).

Algorithm 7 Construct-GMM(Z)

1: **for** $p \leftarrow 1$ to c **do**
2: $\mu(p) \leftarrow C_p$
3: $\Sigma(p) \leftarrow \sigma^2 I$
4: $\phi(p) \leftarrow P(p|Z) \leftarrow$Online-Random-Forest-Prediction(Z)
5: **end for**
6: **return** $gmm \leftarrow$Build-GMM(μ, Σ, ϕ)

Algorithm 8 provides pseudo-code for the rest of the regression algorithm. Once the GMM is built (line 1), a couple of approaches are available, the most popular of which consists in taking the weighted mean of the model or drawing samples from the GMM with probability $\phi(p)$ and computing the samples' weighted mean. However, applying regression directly on the model puts too much emphasis on the Online Random Forest classification results and does not provide a fail-safe mechanism for misclassifications. Thus, instead of sampling from the GMM, our regression algorithm uses a k Nearest Neighbor Search (line 2) to provide k samples that are not only dependent on the observation Z, but also come from a different model than the Online Random Forest. This is a crucial step that adds robustness to the algorithm by combining two of the presented classification algorithms. In other words, where and when one algorithm might fail, the other might succeed. The choice of the Nearest Neighbor Search for this step (as opposed to the Gaussian Model or the Support Vector Machine) comes from the fact that s times more samples can be drawn from it (a total of $s \times c$, as opposed to c). The returned Cartesian coordinate (line 3) is finally calculated as the weighted mean of the k Nearest Neighbors, where the weight of each Nearest Neighbor is set from the Probability Distribution Function (PDF) of the GMM.

Algorithm 8 Regression(gmm, T, Z)

1: $nn \leftarrow$k-NN(T, Z, k)
2: $\hat{C} \leftarrow \dfrac{\sum_{i=1}^{k} nn_i \times \text{PDF}(gmm, nn_i)}{\sum_{i=1}^{k} \text{PDF}(gmm, nn_i)}$
3: **return** $\hat{C}$

The entire regression algorithm requires two parameters to be set, σ (Algorithm 7) and k (Algorithm 8). σ dictates how much the Gaussian components influence each other and should be approximately set to the distance between WiFi readings in the training data. k should be as high as possible in order to provide a lot of samples, yet low enough not to incorporate "neighbors" that are too far away. We have found setting k to 25% of all the observations in T (i.e., $k = 0.25 \times s \times c$) to be a good solution for this tradeoff.

Given a new observation Z, the regression essentially maps a three-dimensional surface to the X-Y Cartesian space of the environment, where the higher the surface's Z-value the more probable the X-Y location. A representative snapshot of the process is shown in Figure 6.8, highlighting an important behavior of the presented regression algorithm. Indeed, the algorithm is not only capable of generating Cartesian coordinates that were not part of the initial training data, but also takes into consideration neighboring votes from the Random Forest classification. In the figure, the highest Random Forest vote (represented by '*') is somewhat isolated and, consequently, its neighbors do not contribute to the overall surface. The region close to the actual robot's location (denoted by '+'), however, is comprised of many local neighbors whose combined votes outweigh those of the Random Forest classification, resulting in a better regression estimation (symbolized by 'x').

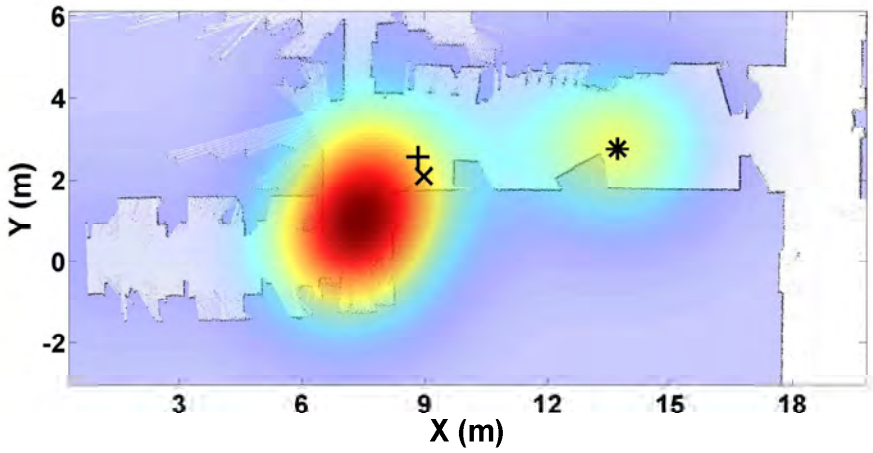


Figure 6.8: Regression example, showing the PDF of the GMM overlaid on top of a section the environment's map. The markers +, x, and * represent the ground truth, regression, and random forest classification locations, respectively.

6.3.2.1 Results

In order to evaluate the accuracy of the regression algorithm, it is necessary to gather new data that will mimic a robot exploring the environment not constrained to the locations covered by the training data T. Consequently, we perform 10 new runs (N_i) in the same environment covered by UCM dataset and utilize the same dataset for training, assuming no prior knowledge of the locations covered by T. In other words, while the training data (T) and the new data (N) approximately cover the same environment, they are not acquired at the same Cartesian coordinates. As such, this experiment provides a direct measure of the regression algorithm's strength. The two datasets, T and N, were taken at different dates and times making the regression problem even more difficult since additional noise is likely not to be modeled by the training data, T. For evaluation purposes, we manually record the robot's ground truth position at discrete locations so that we can quantitatively evaluate the regression algorithm. Figure 6.9 shows the average accuracy (from the 10 runs and 50 random samples used for training) of presented classification algorithms compared against the regression algorithm. Not surprisingly, the regression algorithm works very well by outperforming the Nearest Neighbor Search, Gaussian Process, Random Forest, Online Random Forest, Support Vector Machine, Gaussian Model, and Decision Tree algorithms by 39.33%, 40.86%, 42.38%, 43.27%, 51.67%, 61.16%, and 109.65%, respectively. We conclude this section by mentioning that the regression algorithm only takes 131ms to localize.

6.3.3 Monte Carlo Localization

Although the regression algorithm clearly improves the WiFi localizer's accuracy for real-world scenarios, further improvements can be obtained by taking into account the spatial and temporal constraints implicitly imposed by robot motion. For robot exploration, where the robot is unlikely to move drastically from one position to another, we want to exploit the fact that a robot's position and observation should be coherent within small time steps. In other words, the classification and regression algorithms discussed so far solve the localization problem without taking into consideration the robot's previous location, or, more precisely the probability distribution of the robot's previous location. Specifically, we use an Monte Carlo Localization (MCL), also known as particle filter localization, algorithm, built from the standard implementation presented by Thrun et al. [156], with a couple of modifications. Given a map of the environment, the algorithm estimates the pose of a robot as it moves and senses the environment. With its non-parametric represen-

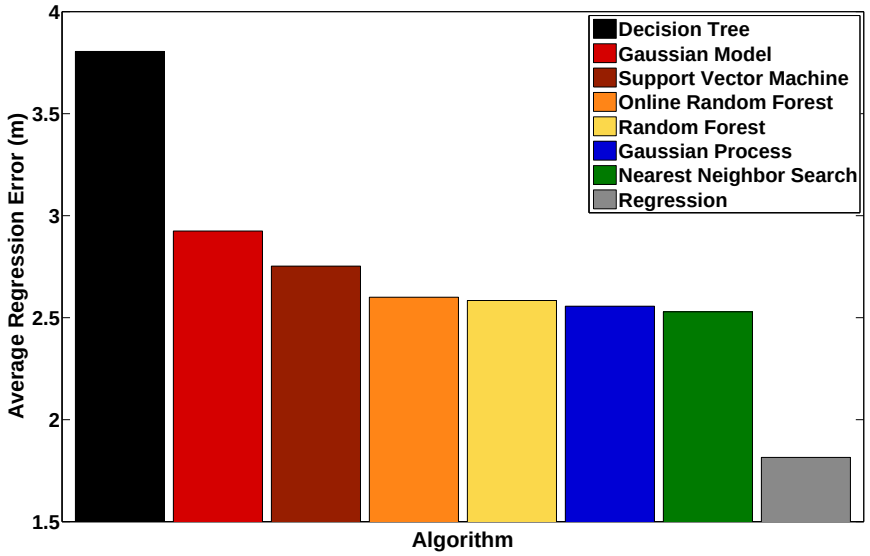


Figure 6.9: Average regression error, in meters, comparing all classification algorithms (left) and the proposed regression algorithm (right).

tation it approximates the probability distribution over the space of possible poses by a particle filter where each particle represents a possible state. When the robot moves and makes new observations, the particles are resampled based on recursive Bayesian estimation. MCL particularly performs well for situations where the belief is multimodal which usually is the case for indoor environments due to perceptual aliasing.

We adopt the same motion model, which exploits the translational and rotational velocities of the robot, in [156]. The measurement model is, evidently, different since it needs to take into account the aforementioned Online Random Forest based regression results. Given a particle with Cartesian coordinate C and a new observation Z, the measurement model computes the probability $P(C|Z)$. The pseudo-code is presented in Algorithm 9 and shows the effectiveness of our GMM implementation (line 1), which can seamlessly transition from regression to MCL. Indeed, each particle i (line 2) is assigned a weight proportional to the probability of being at the particle's state given a WiFi observation Z. Thanks to the GMM, the particle's weight is easily retrieved by using the PDF (line 3).

In addition to the measurement model, we modify the particles' initialization procedure. Instead of randomly or uniformly sampling the particles' state and giving each particle an equal weight, we force the robot to collect a new WiFi reading, Z,

Algorithm 9 Measurement-Model(Z)

1: $gmm \leftarrow$ Construct-GMM(Z) // See Algorithm 7
2: **for** $i \leftarrow 1$ to n **do**
3: $\quad i$.weight $\leftarrow$ PDF(gmm, i.state(X,Y))
4: **end for**

and initialize the particles using that observation, as shown in Algorithm 10. We start by constructing the GMM (line 1) and, for each particle in our filter (line 2), sample a data point from the GMM that will serve as the particle's X,Y state (line 3). Since WiFi signal strengths cannot infer the robot's rotation, we randomly sample the particle's θ state (line 4). The particles' weight is calculated from the GMM's PDF (line 5). In order to add robustness to the previously mentioned problem with robots' rotations, we pick the v best particles (line 7) and add y new particles that are the same as v but have y different θ states each randomly sampled between 0 and 2π (line 8). Since we have augmented the total number of particles, we finalize our particle initialization by keeping the best n particles (line 9).

Algorithm 10 Initialize-Particles(Z)

1: $gmm \leftarrow$ Construct-GMM(Z) // See Algorithm 7
2: **for** $i \leftarrow 1$ to n **do**
3: $\quad i$.state(X,Y) $\leftarrow$ sample(gmm)
4: $\quad i$.state(θ) $\leftarrow$ rand(0 to 2π)
5: $\quad i$.weight $\leftarrow$ PDF(gmm, i.state(X,Y))
6: **end for**
7: **for all** i within the top v weights **do**
8: $\quad$ Add y particles n^y where
$\qquad n^y \leftarrow m$, and
$\qquad n^y$.state(θ) $\leftarrow$ rand^y(0 to 2π)
9: **end for**
10: Select the $|m|$ particles with highest weights

6.3.3.1 Results

In this section, we evaluate our MCL algorithm, running with 5000 particles, on the indoor UC Merced and outdoor Merced datasets. Please note that since we do not have physical access to the areas where other datasets are generated, we cannot use

the MCL on the rest of the datasets.

The robots explored the UC Merced and Merced environments and localized strictly exploiting WiFi signal strengths readings within our algorithmic framework. In order to quantitatively analyze the MCL, an LRF and GPS receiver are mounted on the robots for the indoor and outdoor scenarios, respectively. Please note that these two sensors are only meant to generate pseudo-ground-truth and are not used as part of the WiFi localization. The indoor and outdoor environments are explored 10 and 5 different times, respectively. For each exploration, the MCL is performed 50 times to account for the random nature of the MCL. After each measurement model update, we localize the robot by computing the weighted mean of the particles and compute the error between the MCL and pseudo-ground-truth (SLAM or GPS). The average error for the 50 trials of the 10 indoor and 5 outdoor explorations are 0.61 meters and 1.02 meters, respectively, with a low average standard deviation of below 1cm for both datasets. The algorithm runs in real-time on a consumer laptop, where the motion and measurement models take, on average, 13.50ms and 195.19ms, respectively.

Comparing these results to an offline LRF SLAM algorithm [62], which produces an average error of 0.27m for indoor UCM dataset, we observe a compelling trade-off between accuracy and sensor payload (or price) when using the proposed WiFi localizer. Figure 6.10 provides a visualization of 3 outdoor runs while the other 2 runs are not omitted due to redundant visual overlap. An illustrative example of a partial run of the indoor experiment is shown in Figure 6.11.

6.4 Merging Algorithm

Our map merging algorithm consists of three main components, as shown in Figure 6.12. Given an occupancy grid and an appearance-based map, we first deduce the amount of map overlap by identifying and separating the labels that map to the empty set (i.e., $L_i \rightarrow \emptyset$). In the second stage, for each of the remaining labels in the appearance-based map $L_i \in \mathbf{L_2}$ the probably distribution over the set of labels in the occupancy grid map $\mathbf{L_1}$ is computed. The most similar Cartesian label can be determined using the probability distribution. Such an estimation, however, would be solely based on the WiFi readings linked to each vertex in the appearance-based map. In the last step, we use the additional information inherently encoded by the appearance-based map's edges to refine the probability distribution and improve the precision of the estimate by applying regression in the space of Cartesian labels.

Figure 6.10: Three non-overlapping localized outdoor runs showing the output of the WiFi localizer (line) and GPS receiver (crosses) on top of Google Maps.

6.4.1 Map Overlap Estimation

The issue of determining map overlap is a One-Class Classification (OCC) problem, whose aim is to distinguish one class of objects from all other possible objects. In the context of map merging using WSS, we assume that observations in T_1 represent one class of objects (i.e., the metric map) and we want to classify unknown observations inside T_2 (i.e., the appearance-based map) to see whether or not they belong to T_1. With principles related to outlier detection, OCC has stimulated the design of many algorithms. Since these algorithms are often data-dependent, we have implemented and contrasted five alternatives. Interested readers are directed to [155] for a more detailed survey.

6.4.1.1 OCC Algorithms

One-Class Support Vector Machine (OC SVM): OC SVM is a modified version of the two-class SVM where only one class is necessary for training. Similarly to SVM, a kernel function maps the training data into a feature space. The origin of that feature space is then treated as the only member of the "second" class and the techniques of a binary SVM classifier can be applied to produce a hyperplane that maximizes the functional margin. For the kernel of the OC SVM, we use a Gaussian Radial Basis Function that requires a parameter h describing the bandwidth of the

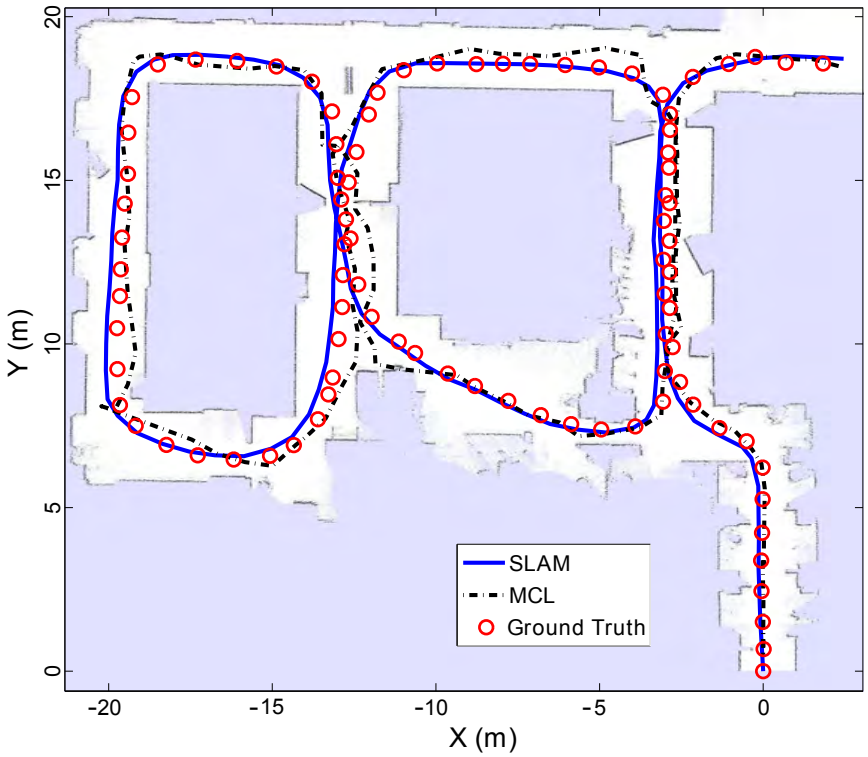


Figure 6.11: Localized run showing the output of LRF SLAM (straight line), the WiFi localizer (dotted line), and Ground Truth (circles).

kernel. Once the OC SVM is trained on the first map's data T_1, a new observation $Z_i \in T_2$ is classified by computing its relation to the hyperplane, $f(Z_i) = (w \cdot \Phi(Z_i)) - \rho$, where $f(x)$ represents the hyperplane equation, w is the normal vector to the hyperplane, $\Phi(x)$ describes the kernel function, and ρ corresponds to the hyperplane's offset. If $f(Z_i) > 0$ then $Z_i \in T_1 \cap T_2$ and if $f(Z_i) < 0$ then $Z_i \notin T_1$.

Principal Component Analysis (PCA): PCA orthogonally projects a dataset from one subspace to another while retaining as much variance into as few components as possible. PCA is generally utilized in the context of dimensionality reduction, but it can also be used as a classification and noise reduction technique. PCA in the context of OCC is performed as follows. First, the training data T_1 is used to compute the a components, of which only the $x < a$ with the greatest variance are retained. The x components are then used to project the training data into a lower dimensional subspace, resulting in $\tilde{T}_1$. A new observation to classify $Z_i \in T_2$ is projected into the same subspace (resulting in $\tilde{Z}_i$) and the Euclidean distance to the

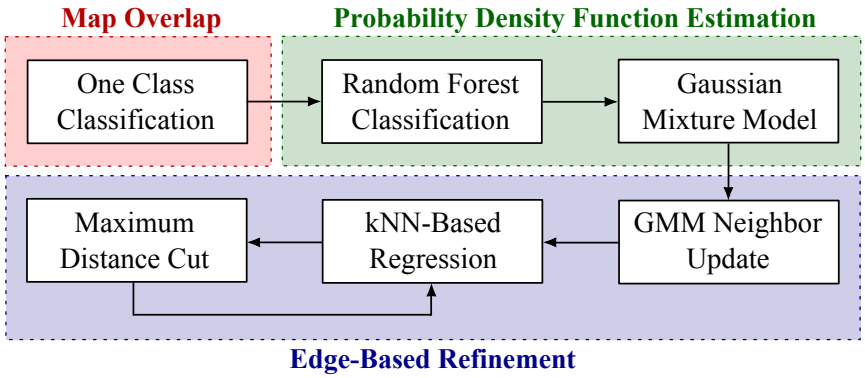


Figure 6.12: Flowchart describing three stages of heterogeneous map merging algorithm.

nearest point in $\tilde{T}_1$ is computed by $D(\tilde{Z}_i)$. If $D(\tilde{Z}_i) < d$, where d is a pre-computed distance threshold, we deduce that $Z_i \in T_1 \cap T_2$ (otherwise, $Z_i \notin T_1$).

k-Means: the k-means algorithm for OCC attempts to characterize the training data T_1 as a set of similar objects. The observations of the first map T_1 are partitioned into a set of k clusters that can then be exploited to classify unknown observations. The k cluster centers are consequently a-dimensional objects. An observation to classify $Z_i \in T_2$ is compared against its closest cluster center, where the Euclidean distance between the two is given by $D(Z_i)$. The new observation is considered part of the first map (i.e., $Z_i \in T_1 \cap T_2$) when $D(Z_i) < d$ (otherwise, $Z_i \notin T_1$), where d is a pre-determined distance threshold.

Nearest Neighbors Ratio (NN Ratio): the NN ratio is a modification of the simple NN search, where the distances of the NN and second-NN are compared. Since NN searches operate directly in the space of the training data T_1, they possess the distinct advantage of not requiring a training phase. To classify a new observation $Z_i \in T_2$, its two NN $Z_{N1}, Z_{N2} \in T_1$ are computed, along with their Euclidean distances $D(Z_{N1})$ and $D(Z_{N2})$. If $D(Z_{N1})/D(Z_{N2}) < d$, where d is a pre-computed ratio threshold, then we can assert that $Z_i \in T_1 \cap T_2$ (otherwise, $Z_i \notin T_1$).

Gaussian Model: this method models the density of the training data by fitting a Gaussian distribution. More specifically, the mean μ and covariance matrix Σ are computed from the occupancy grid map data T_1 to construct a multivariate Gaussian. The number of dimensions of the Gaussian is equal to a. To determine whether a new observation $Z_i \in T_2$ is part of the first map, we evaluate the Gaussian's Probability Density Function (PDF), $\varphi_{\mu,\Sigma}(Z_i)$, and compare the results against a PDF threshold,

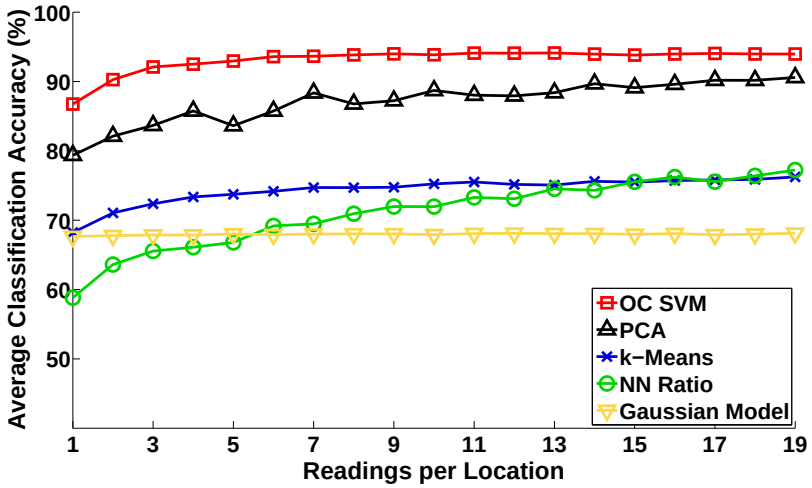


Figure 6.13: Classification accuracy for each OCC algorithm, as the number of observations per location is increased. The data presented is the average of 50 experiments.

d. We conclude that $Z_i \in T_1 \cap T_2$ when $\varphi_{\mu,\Sigma}(Z_i) > d$ and that $Z_i \notin T_1$ otherwise.

6.4.1.2 Results

We compare the classification accuracy of each algorithm to determine the best one to be used for map overlap estimation function. The results presented here are based on UCM dataset presented in Table 6.1. To investigate the potential effects of the number of observations per location on the algorithms' accuracy, the robot was stopped at pre-determined intervals and a total of 20 observations were recorded for each location. The entire map is divided in two, with 50% representing positive examples and the rest representing negative examples. Out of the positive examples, the algorithms are trained on 50% of the data so that the remaining 50% is used for classification, along with 50% of the data representing negative examples (i.e., we have the same number of positive and negative examples to classify). Due to the random nature of some OCC algorithms and potential bias towards certain datasets, we randomize the training and classification datasets and run the experiments 50 times.

Figure 6.13 shows the results of the experiments performed. OC SVM clearly separates itself from the other algorithms, with a classification accuracy of over 90% with very few readings per location. Comparatively, OC SVM is on average 6.49%, 20.17%, 23.74%, and 26.99% more accurate than PCA, k-Means, NN Ratio, and

the Gaussian Model, respectively. All of the algorithms are extremely fast, taking, in the worst case, 100ms and 20ms to train on the entire dataset and classify 100 observations, respectively.

Unfortunately, all these OCC algorithms rely on various parameters and classification results are highly-dependent on them. More specifically, it is necessary appropriately set the bandwidth kernel h for the OC SVM, the number of components x and the threshold d for PCA, the number of clusters k and the threshold d for k-means, the ratio threshold d for the NN Ratio, and the PDF threshold d for the Gaussian Model. To determine each of these parameters, we use process that relies on the cross-validation of the parameters by using a different dataset. As our cross-validation dataset T_O we used the outdoor Merced dataset with details presented in Table 6.1. Similarly to the process we described above to evaluate the performance of each algorithm, the cross-validation dataset is equally divided into positive and negative test cases and 50% of the positive test cases are reserved for training with the remaining 50% being allocated, along with 50% of the negative test cases, for classification. Using this data decomposition, we learn each parameter using a brute-force approach, where the parameters are sampled uniformly within their respective logical ranges. We train each OCC algorithm with a parameter sample on the training data of T_O and attempt to classify the data reserved for classification. Since we have ground truth, we can then compute the number of correct classifications. The process is repeated for all parameter samples and the one that yields the highest classification accuracy is retained. Table 6.2 summarizes computed parameters. This process, which only needs to be performed once, assumes that the dataset T_O is a good representation of the datasets that will be used by OCC algorithms. We have experimentally verified this assumption for the best performing algorithm, OC SVM. The classification accuracies for both indoor and outdoor datasets are computed for varying kernel bandwidth. As can be seen in Figure 6.14 they exhibit similar trends and the accuracy peaks around the same kernel bandwidth value. In other words, the parameter estimated using one dataset can be used on another dataset. The optimality of the computed parameter value can also be seen by the false positive and false negative ratio curves of the outdoor data classification results which is presented in Figure 6.15. Estimated parameter value corresponds to the point where the summation of false negatives and false positives reaches its global minimum.

OC SVM	PCA		k-means		NN Ratio	Gaussian Model
h	x	d	k	d	d	d
57.74	0.79	4.49	37	48.96	0.97	0.00

Table 6.2: Optimal parameters of OCC algorithms computed using outdoor Merced dataset.

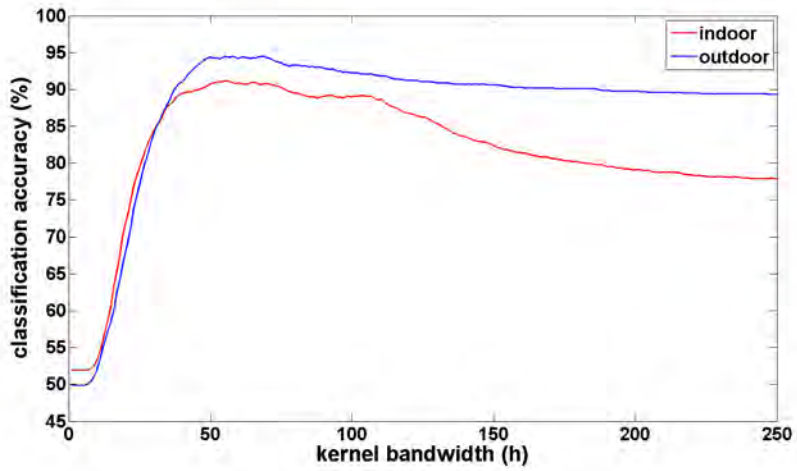


Figure 6.14: Classification accuracies of OC SVM performed on both indoor and outdoor datasets for varying kernel bandwidth.

6.4.2 Probability Distribution Function

Once the map overlap is estimated, using OC SVM, the best OCC algorithm exhibited in the previous section, the labels that belong to the overlapping regions of the maps, $L_i \not\rightarrow \emptyset$, are considered for merging. Building upon the results presented in Section 6.3 which shows that it is possible to differentiate between different locations by only considering WSS readings received from APs, these identified labels are matched via their associated WiFi readings using our WiFi localization algorithm. More precisely, we compute a function $f : Z_i \in T_2 \rightarrow L_j^1$ from the data in the WiFi map T_1. Function f takes an observation $Z_i = [z_i^1, \ldots, z_i^a]$ tied to L_i^2 in the appearance-based map and returns the matching label in the grid map, L_j^1, from which we can look up the Cartesian label C_j. In order to compute f from the training data T_1, we use the Online Random Forest (ORF), which is shown to provide the best localization results among all online classification algorithms. Using the ORF

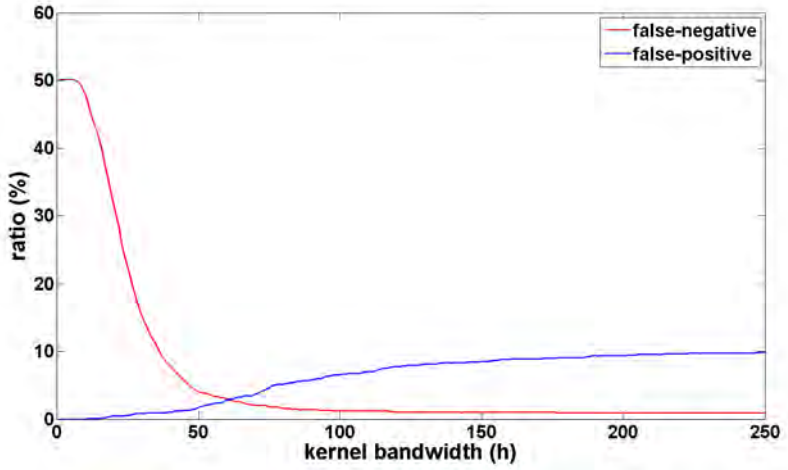


Figure 6.15: The percentages of false negatives (fn), false positives (fp), and their summation (fn+fp) are plotted for outdoor data as the kernel bandwidth for OC SVM algorithm is increased.

with 250 trees, WSS readings of all overlapping images in the appearance-based map $Z_i \in T_2$ are classified in T_1 and their matching labels L_j^1 in the grid map are determined. After the classification phase, we apply the regression algorithm described in Section 6.3.2 in order to define the probability distribution over the set of labels $P(L_j^1 = L_i^2 | Z_i)$ where $Z_i \in T_2$. A Gaussian Mixture Model (GMM) representing this probability distribution function is built using the procedure presented Algorithm 7.

6.4.3 Edge-based Refinement and Regression

After we generate a GMM for each image in the appearance-based map, a Cartesian coordinate in the occupancy grid map can be assigned each image. However, this single-shot localization approach only focuses on the WiFi readings associated with vertices and it can be greatly improved by including the extra information inherently encoded in the edges. As described in Section 4.1, edges of appearance-based maps symbolize similarity between connected images. Based on the idea that similar images should be captured from the same area in the environment, the position estimate of an image can be fine-tuned using its neighbors. The power of this local consistency approach is evident when one image is misclassified while the position of all its neighbors are correctly estimated. To address this and similar issues, we create, for each image I_i, an aggregated GMM $\tilde{g}_i$ that combines each of i's 1-hop

neighbor's GMMs g_i. The contribution of each neighbor is weighted with the edge's weight that encodes their image similarity. In other words, the more similar the neighbor, the higher its contribution to the aggregated probability distribution. The neighbor's GMM is combined with the GMM of image i and the resulting distribution is normalized.

While it is evident that including information from correctly localized neighbors improves the accuracy, the contribution of a misclassified neighbor to the aggregated GMM still creates a negative impact on regression results. With the same motivation that each image should not be localized far away from its similar neighbors, we introduce an iterative process that enforces local consistency among the images in the appearance-based map. The refinement process described in Algorithm 11 is based on the idea that any two correctly localized neighbors should lie within a maximum distance r_{max} of each other. To estimate this variable we compute the distances between every pair of images using initial estimations from the regression algorithm and set $r_{max} = 3\sigma$ where σ is the standard deviation of the distance distribution. The iterative algorithm is bootstrapped with the initial positions estimated by applying regression for each image I_i (line 2-4). The aggregated GMM $\tilde{g}_i$ associated with each image i in the appearance-based map (line 5) takes into account all of its neighbors (line 6). If any Gaussian mixture component's mean (line 7) is located at a distance of more than r_{max} away from i's estimated position, the mixture component of $\tilde{g}_i$ is removed (line 8). Once all GMMs are pruned, a new set of locations is computed by running the regression algorithm on the updated GMMs (line 2-4). This iterative refinement process continues until convergence (i.e., the mean difference between previous and new location estimations drops below a threshold ε).

6.4.4 Results

To evaluate the accuracy of proposed map merging algorithm two P3AT robots were used. One was equipped with a LMS200 laser range finder and other only with a Logitech C910 webcam. Both explored the first floor of the Engineering building at UC Merced in 3 independent runs covering a total distance of over 1 km. Each robot generated either an appearance-based map using the map building algorithm presented in Section 4.5 or an occupancy grid map by employing the SLAM algorithm [62] depending on the hardware they are equipped with. Both robots collected WiFi observations Z_i from a total of 75 unique APs ($a = 75$) using their WiFi cards. Appearance-based maps are composed of 234 images in average where a single WiFi reading is attached to each image. In the occupancy grid maps over 1000 WiFi readings per map are partitioned into an average of 229 clusters whose centers'

Algorithm 11 Refinement

repeat
  for $i \leftarrow 1$ to c_2 **do**
    $\hat{C}_i \leftarrow$Regression($\tilde{\mathbf{g}}_\mathbf{i}$, T_1, $Z_i \in T_2$)
  end for
  for $i \leftarrow 1$ to c_2 **do**
    for all $I_n \in$ Neighbors(I_i) **do**
      for $j \leftarrow 1$ to c_1 **do**
        if dist($\hat{C}_j$,$\hat{C}_n$) $\geq r_{max}$ **then** $\tilde{\mathbf{g}}_\mathbf{i}(\phi_j) = 0$
      end for
    end for
  end for
until convergence
return $\hat{C}$

Cartesian coordinates are used as WiFi map labels.

6.4.4.1 Full Map Merge

Each appearance-based map is merged with each occupancy grid map after the map overlap is computed using OC SVM algorithm with an accuracy of over 99% in average. For the images that are classified as a part of the occupancy grid map, a Cartesian coordinate is assigned as described in Section 6.4.3. An illustrative example of merged map with full overlap where all the images are correctly assigned to a position estimate is shown in Figure 6.16. To account for the randomness associated with RF training process, we merged each map pair 10 times. The average error of all 90 trials is 1.21 meters with $\sigma = 1.09$. The overall heterogeneous map merging algorithm including the map overlap computations takes in average 129 seconds to merge two moderate size maps.

6.4.4.2 Partial Map Merge

To demonstrate the true potential of the proposed merging algorithm, we fused an appearance-based map M_A with two smaller non-overlapping occupancy grid maps M_O^1 and M_O^2 created in addition to the ones described above. Two grid maps with an unknown transformation between their coordinate frames provide no additional information when used together. For instance, a robot using these maps cannot navigate

from one to another. However, when the appearance-based map is merged with these maps, it creates the needed link between them so that the merged map carries more value either to the operator or the robot utilizing it. This representative example is illustrated in Figure 6.17 where on the left the appearance-based map is drawn using the graphviz software and grid maps created by navigating in the top and bottom portions of the environment depicted in Figure 6.16 are presented on the right. Note that with no information on relative transformations between their local coordinate frames, the maps are placed at an arbitrary position while the orientations are set by their corresponding drawing software, gmapping and graphviz. To determine the overlap between maps, OC SVM is trained for each occupancy grid map and each image in M_A is classified as either in-M_O^1, in-M_O^2, or non-matching. The correct classification accuracies for M_O^1 and M_O^2 are 97.54% and 99.01%, respectively.The Cartesian coordinates of the images are then computed and they are placed to their estimated locations in their matching grid map while still being connected to their appearance-based map neighbors through their edges. The vertices of M_A shown in green in Figure 6.17 correspond to images captured in the hallway connecting two grid maps which is not visible in any of them. These vertices are successfully identified as non-matching and constitute a critical part of the merged map by holding all three individual maps connected. The accuracy of localized images is 1.06 meters in average and merging process takes less than 45 seconds.

6.5 Conclusions

In this chapter we have presented a system that, to the best of our knowledge, is the the first that can successfully merge together heterogeneous spatial models. Even though the presented framework is generic and can be used to merge other types of maps together, we channeled our attention specifically into merging occupancy grid maps with appearance-based maps. To remedy the lack of a common representation, we proposed to use WiFi signals as a common substrate. WiFi signals omnipresent in today's environments coupled with robots carrying WiFi cards to communicate with other team members or the base station makes it possible for robots to collect WSS readings while creating their maps whether it is an appearance-based or an occupancy grid map. Within this respect, maps accompanied with a WiFi map can be merged using an algorithm that, instead of correlating parts of these heterogeneous maps, matches WiFi signals and stitches their corresponding counterparts in the actual maps. We have proposed a state-of-the-art WiFi localization and mapping algorithm that is capable of localizing accurately and running in real-time as a solution to this

problem. Our algorithm solves major problems found in other WiFi localizers, since it does not require robots to stop when acquiring WSS readings and can build WiFi maps incrementally as part of an online process. The claims made throughout the chapter are substantiated by a comprehensive set of experiments analyzing 7 different algorithms across 6 datasets accounting for over 3.5 kilometers of indoor and outdoor exploration. Together, all of the experiments provide compelling evidence regarding the robustness of our approach.

The maps to be merged, however, does not necessarily hundred percent overlap with each other. Thus, we employed and tested contemporary machine learning algorithms to determine the overlap between maps. Our proposed map overlap estimation algorithm has been experimentally evaluated with various maps generated from aforementioned datasets.

Using these two modules, i.e., WiFi localization and map overlap estimation, overlapping vertices of appearance-based map can be matched to Cartesian locations in the occupancy grid map. In order to improve the accuracy we introduced an iterative local refinement process which exploits the valuable information encoded in the edges of the appearance-based maps. Our proposed system is validated using various occupancy grid and appearance-based maps built in real-world conditions, the results of which confirm its strengths. In the future, the system can be extended to include a wider variety of map types such as topological and feature based maps.

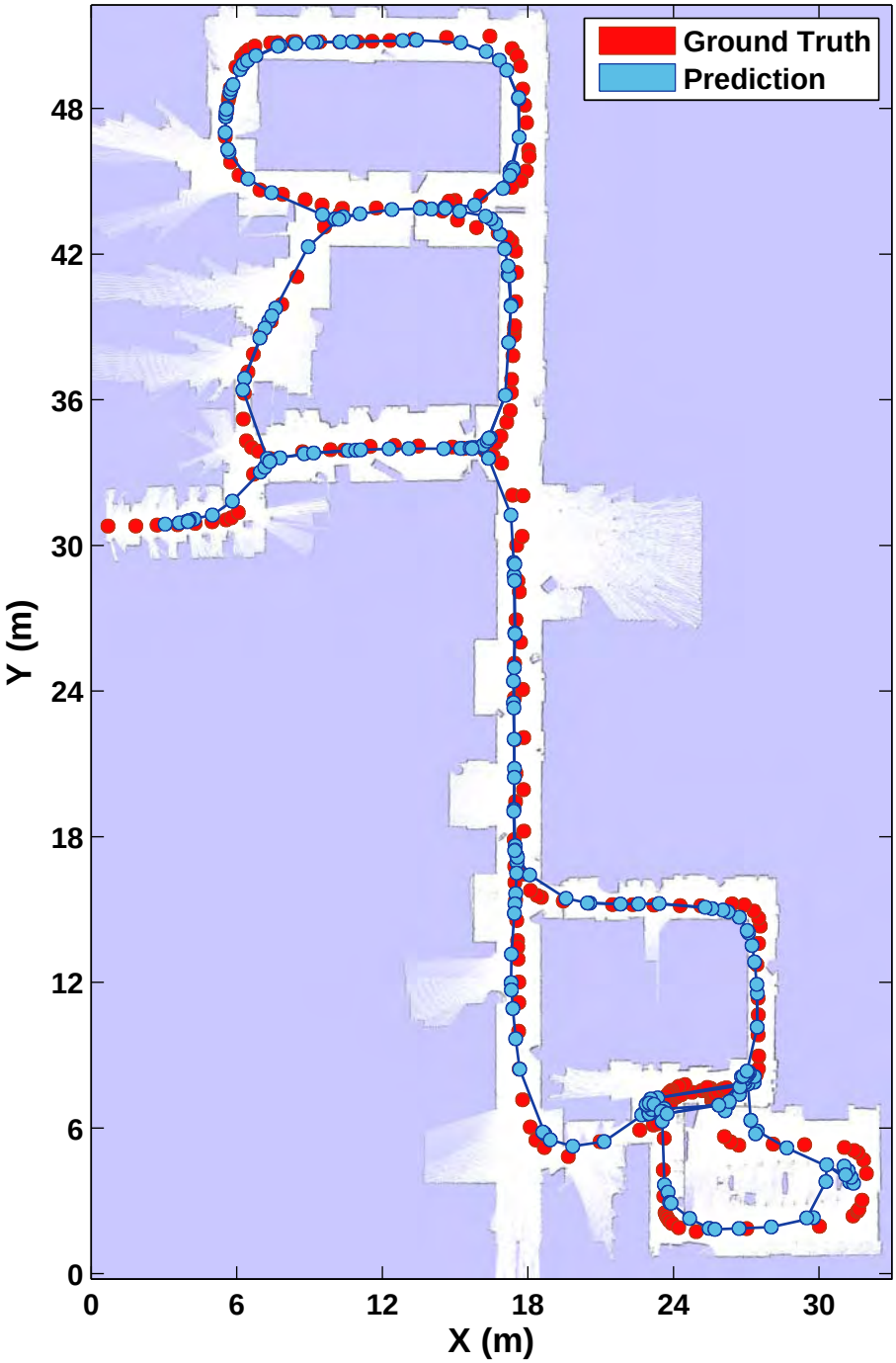


Figure 6.16: An appearance-based map is merged with an occupancy grid map after OCC algorithm correctly estimated 100% overlap between two maps.

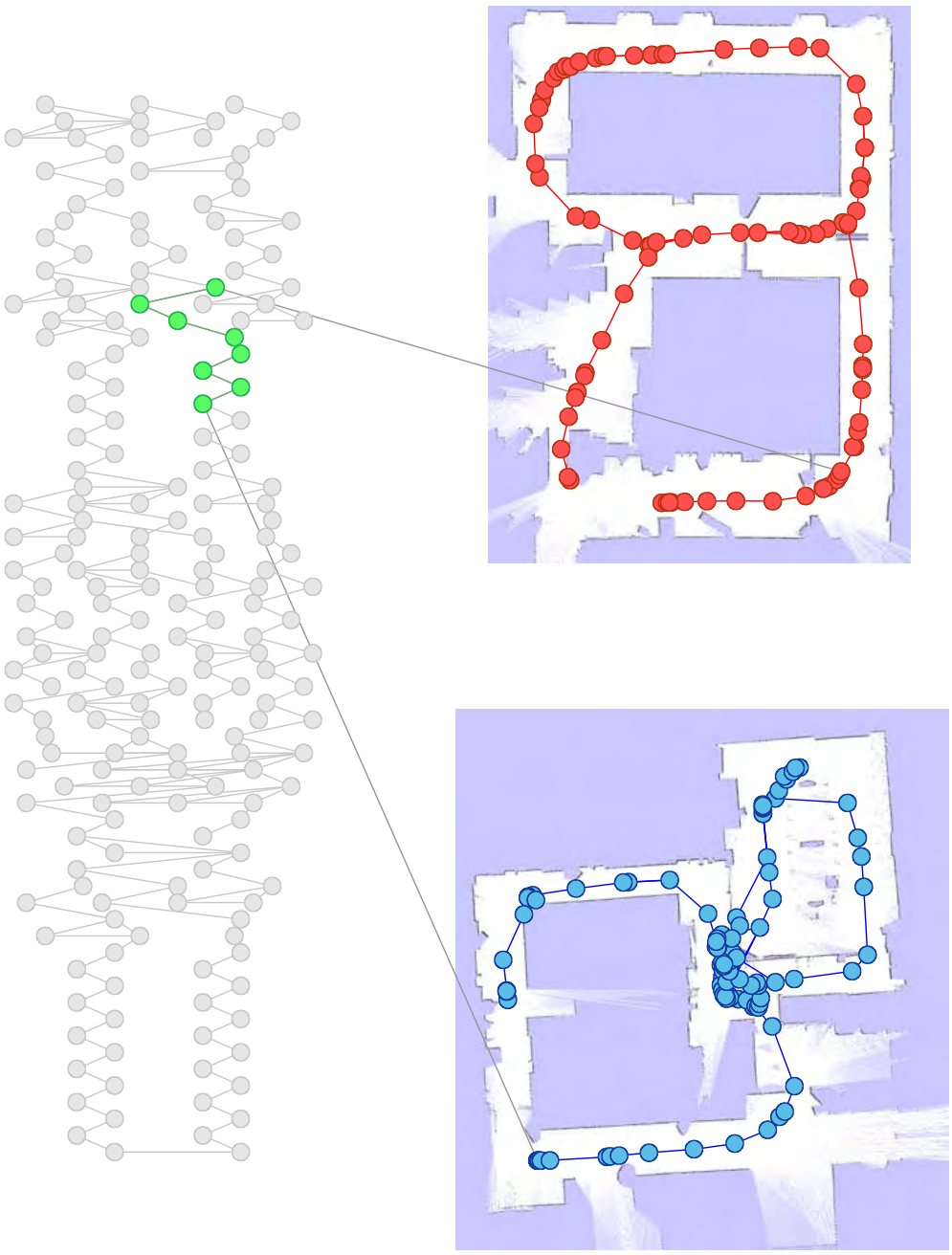


Figure 6.17: A representative example showing an appearance-based map (left) merged with two occupancy grid maps (right) is presented.

CHAPTER 7

Conclusions

In this dissertation, we presented an appearance-based localization and mapping framework and demonstrated both its localization and mapping capabilities using a heterogeneous team of sensor deprived robots only equipped with cameras. Sharing the dream of fully autonomous inexpensive consumer robots in every house, we advocate that along with localization and mapping, one of the most crucial tasks any robot should be able to perform is navigation. With this in mind, we proposed a novel visual servoing algorithm. The major advantage of the algorithm is that it can steer robots to target images captured by another robot which does not necessarily share the same morphology or the camera placement. This important feature of the proposed servoing technique enables the robots to navigate not only in appearance-based maps created by other robots but also in ones generated from a set of images collectively acquired by a team of heterogeneous robots. There are two main ways that a multi-robot system can create a map collectively: centralized and decentralized. We embrace the decentralized distributed approach and propose an appearance-based map merging algorithm. In a scenario where two robots are exploring an unknown environment, once they are in communication range, they can merge their partial maps in order to create and share a more complete spatial model of the environment. It is a complete anytime algorithm that creates *important* connections between maps earlier in the merging process so that if it gets interrupted, the gist of the overall connectivity will be already captured in the merged map. We also extended the algorithm to merge multiple appearance-based maps simultaneously which provides a substantial performance improvement over the baseline sequential pairwise merging algorithm. Furthermore, we proposed the use of algebraic connectivity, a well studied concept in graph theory, to measure the quality of merged appearance-based maps and the performance of merging algorithm.

In any robot team each individual member may be equipped with a different set of sensors and utilizing a different spatial model. In order to capture this heterogeneity in the spatial representations used among the members of the robot team we introduced a novel generic map merging algorithm which exploits the WiFi signals omnipresent in today's environments. More precisely, the algorithm requires

the robots to a create a WiFi map along with their own spatial representation and merges these heterogeneous models by using WiFi maps as the common substrate. The merging problem then boils down to merging WiFi maps by matching WSS readings. We presented a novel machine learning based WiFi localizer as the solution to this problem and demonstrated its accuracy and speed on a large set of datasets.

We are envisioning several possible directions that the presented framework can be extended to in the future. First of all, the visual servoing algorithm currently assumes that cameras used to capture initial and goal images are the same. However, it is very possible that even the robots in the same team may have different cameras. With this idea in mind, the algorithm can be generalized to accommodate different camera models and image resolutions. Another valuable addition would be in the appearance-based map merging algorithm. What concerns the merging of appearance-base maps, we disregarded merging of common vertices and primarily focused on creating interconnecting edges since without their insertion the maps would remain as separate entities with no utilization value. Nevertheless, by *merging* vertices common in both maps to be merged the resulting appearance-based map can be compressed and the redundancy in the map will be eliminated. This process can be as simple as identifying vertices that are very highly similar and removing all but one from the map. This type of compression may cause possible implications such as disconnectivity in the map and increasing the number of connected components, and therefore deserves some investigation.

The current appearance-based localization and mapping algorithm works based on the assumption that the environment is static. However, the inevitable dynamic nature of the environment causes several problems which a non-adaptive algorithm cannot cope with. Hence, in order to make it applicable to more realistic scenarios such as a robot navigating in a domestic environment for a long period of time, the map should be updated to reflect the dynamism in the environment. We believe this idea can be implemented by defining a *lifetime* for every vertex in the map which decreases as the robot navigates and re-explores the environment. The lifetime of a vertex will increase every time an image captured by the robot matches to it, meaning that the visual content encoded in the vertex still exists in the environment. Vertices with no lifetime left will be removed from the appearance-based map with the idea that they do not reflect the visual experience of the environment anymore.

The localization algorithm we presented processes each images individually and localizes it within the appearance map globally. This framework is more suitable for a scenario where random query images are needed to be localized. However, if

these images are captured sequentially like in the case of a robot taking images of the environment it is navigating in, then the temporal coherence between consecutive images can be used to eliminate false matches due to perceptual aliasing and consequently improve the localization accuracy. This idea can be realized by implementing a Bayes filter such as Monte Carlo Localization, also known as particle filter localization. The bag of words framework can be used as the measurement model along with a motion model devised from the robot's kinematics.

Another promising direction to investigate is the probabilistic hierarchical maps. The appearance-based maps handling lower level tasks like navigation, mapping, and localization, unfortunately, do not provide a minimalist structure that is optimized for computational complexity. A medium size map may contain thousands of vertices. In order to create a similar representation to how humans perceive the world and achieve an optimal graph complexity, a spatial representation that contains different levels of abstraction is required. Within this respect, the current framework can be extended to a hierarchical topological mapping approach that contains the appearance-based in its lowest level and additional levels of abstractions on top of that. Each abstract level will be created by clustering vertices in the lower level and assigning them to a meta-vertex, starting from the appearance-based map. An approximate solution based on spectral clustering can be implemented using a k-means clustering step to group the vertices in one layer into a meta-vertex in the next level. Additionally, in order to capture the dynamic nature of the environment, belong-to relationships of vertices between two hierarchical levels can be defined as probability distributions over all meta-vertices in the higher hierarchical level. Since each vertex in the lower level is linked to all meta-vertices in the higher level with a probability, the map can adapt itself to the changes in the environment by simply updating these probabilities. The resulting framework will enable the usage of probabilistic planning tools like Markov Decision Processes. Once the hierarchical topological map is constructed, the tasks executed in appearance-based maps such as localization and navigation can be performed in a more efficient way by exploiting the multi-level structure of the map.

References

[1] N.M.M. De Abreu. Old and new results on algebraic connectivity of graphs. *Linear Algebra and its Applications*, 423:53–73, 2007.

[2] J.S. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man and Cybernetics*, 21:473–509, 1991.

[3] C. Andersen, S. D. Jones, and J. L. Crowley. Appearance based processes for visual navigation. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, volume 2, pages 551–557, 1997.

[4] A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer. Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, 24(5):1027–1037, 2008.

[5] R. Aragüés, J. Cortés, and C. Sagüés. Distributed consensus algorithms for merging feature-based maps with limited communication. *Robotics and Autonomous Systems*, 59(3-4):163–180, 2011.

[6] R. Aragüés, J. Cortés, and C. Sagüés. Distributed consensus on robot networks for dynamically merging feature-based maps. *IEEE Transactions on Robotics*, 28(4):840– 854, 2012.

[7] P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *IEEE International Conference on Computer Communications*, pages 775–784, 2000.

[8] B. Balaguer, G. Erinc, and S. Carpin. Combining classification and regression for wifi localization of heterogeneous robot teams in unknown environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3496–3503, 2012.

[9] J. Bauer, N. Sunderhauf, and P. Protzel. Comparing several implementations of two recently published feature detectors. In *Proceedings of the International Conference on Intelligent and Autonomous Systems*, volume 6, 2007.

[10] H. Bay, T. Tuytelaars, and L.V. Gool. Surf: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 404–417, 2006.

[11] K.R. Beevers and W.H. Huang. Loop closing in topological maps. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 4378–4383, 2005.

[12] A. Birk and S. Carpin. Merging occupancy grids from multiple robots. *Proceedings of the IEEE*, 94(7):1384–1397, 2006.

[13] J. Biswas and M. Veloso. WiFi localization and navigation for autonomous indoor mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 4379–4384, 2010.

[14] T. Biyikoglu and J. Leydold. Graphs of given order and size and minimum algebraic connectivity. *Linear Al*, 436(7):2067–2077, 2012.

[15] F. Bonin-Font, A. Ortiz, and G. Oliver. Visual navigation for mobile robots: a survey. *Journal of Intelligent and Robotic Systems*, 3(3):263–296, 2008.

[16] O. Booij, B. Terwijn, Z. Zivkovic, and B. Krose. Navigation using an appearance based topological map. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3927–3932, 2007.

[17] W. Braun and U. Dersch. A physical mobile radio channel model. *IEEE Transactions on Vehicular Technology*, 40(2):472–482, 1991.

[18] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[19] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. CRC Press, Boca Raton, FL, 1984.

[20] S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 2008.

[21] S. Carpin. Merging maps via Hough transform. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1878–1883, 2008.

[22] S. Carpin, A. Birk, and V. Jucikas. On map merging. *Robotics and Autonomous Systems*, 53(1):1–14, 2005.

[23] F. Chaumette. Potential problems of stability and convergence in imagebased and position-based visual servoing. In D. Kriegman, G . Hager, and A.S. Morse, editors, *The confluence of vision and control*, volume 237, pages 66–78. LNCIS Series, Springer-Verlag, 1998.

[24] F. Chaumette and S. Hutchinson. Visual servo control, Part I: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, 2006.

[25] F. Chaumette and S. Hutchinson. Visual servo control, Part II: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, 2007.

[26] F. Chaumette and S. Hutchinson. Visual servoing and visual tracking. In *Handbook of Robotics*, chapter 24, pages 563–583. Springer, 2008.

[27] K. Chen and C. Guestrin. Wifi cmu. `http://select.cs.cmu.edu/data/index.html`, 2009.

[28] G. Chesi, G. L. Mariottini, D. Prattichizzo, and A. Vicino. Epipole-based visual servoing for mobile robots. *Advanced Robotics*, 20(2):225–280, 2006.

[29] G. Chesi, D. Prattichizzo, and A. Vicino. A visual servoing algorithm based on epipolar geometry. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 737–742, 2001.

[30] F. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[31] T. Collins, J.J. Collins, and C. Ryan. Occupancy grid mapping: An emprical evaluation. In *Proceedings of the Mediterranean Conference on Control and Automation*, 2007.

[32] A. Danesi, D. Fontanelli, and A. Bicchi. Visual servoing on image maps. In *Proceedings of the International Symposium on Experimental Robotics*, pages 277–286, 2006.

[33] T. Dang, C. Hoffmann, and C. Stiller. Continuous stereo self-calibration by camera parameter tracking. *IEEE Transactions on Image Processing*, 18(7):1536–1550, 2009.

[34] G. Dedeoglu and G. S. Sukhatme. Landmark-based matching algorithm for cooperative mapping by autonomous robots. In *Distributed Autonomous Robotic Systems 4*, pages 251–260. Springer, 2000.

[35] G.N. Desouza and A.C. Kak. Vision for mobile robot navigation: A survey. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002.

[36] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[37] F. Doshi-Velez, W. Li, Y. Battat, B. Charrow, D. Curtis, J. Park, S. Hemachandra, J. Velez, C. Walsh, D. Fredette, B. Reimer, N. Roy, and S. Teller. Improving safety and operational efficiency in residential care settings with wifi-based localization. *Journal of the American Medical Directors Association*, 13(6):558–563, 2012.

[38] F. Duvallet and A. Tews. WiFi position estimation in industrial environments using Gaussian processes. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2216–2221, 2008.

[39] L. Eddy and S. Wai. Lego robot guided by wi-fi devices. Technical Report 271-QYA2, The Hong Kong University of Science and Technology, 2010.

[40] P. Elinas, J. Hoey, D. Lahey, J.D. Montgomery, D. Murray, S. Se, and J.J. Little. Waiting with jose, a vision-based mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3698–3705, 2002.

[41] G. Erinc and S. Carpin. Image-based mapping and navigation with heterogeneous robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5807–5814, 2009.

[42] G. Erinc and S. Carpin. Anytime merging of appearance based maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1656–1662, 2012.

[43] B. Espiau. Effect of camera calibration errors on visual servoing in robotics. In *Proceedings of International Symposium on Experimental Robotics*, pages 187–193, 1993.

[44] S.M. Fallat, S. Kirkland, and S. Pati. On graphs with algebraic connectivity equal to minimum edge density. *Linear Algebra and its Applications*, 373:31–50, 2003.

[45] J. Fasola, P.E. Rybski, and M. Veloso. Fast goal navigation with obstacle avoidance using a dynamic local visual model. In *Proceedings of the Brazilian Symposium of Artificial Intelligence*, 2005.

[46] F. Ferreira, J. Dias, and V. Santos. Merging topological maps for localisation in large environments. In *Proceedings of the International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2008.

[47] B. Ferris, D. Fox, and N. Lawrence. WiFi-SLAM using Gaussian process latent variable models. In *International Joint Conferences on Artificial Intelligence*, pages 2480–2485, 2007.

[48] B. Ferris, D. Hahnel, and D. Fox. Gaussian processes for signal strength-based location estimation. In *Robotics: Science and Systems*, pages 303–310, 2006.

[49] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[50] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25:607–618, 1975.

[51] J. Fink, N. Michael, A. Kushleyev, and V. Kumar. Experimental characterization of radio signal propagation in indoor environments with application to estimation and control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2834–2839, 2009.

[52] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applicationsto image analysis and automated cartography articulated figures. *Communications of the ACM*, 24(6):381–395, 1981.

[53] N.R. Gans and S.A. Hutchinson. An experimental study of hybrid switched system approach to visual servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 3061–3068, 2003.

[54] B. Gates. A robot in every home. *Scientific American*, 296(1):44–51, 2007.

[55] S. Saeedi Gharahbolagh, L. Paull, M. Trentini, M. Seto, and H. Li. Efcient map merging using a probabilistic generalized voronoi diagram. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4419–4424, 2012.

[56] S. Saeedi Gharahbolagh, L. Paull, M. Trentini, M. Seto, and H. Li. Map merging using Hough peak matching. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4683–4688, 2012.

[57] A. Ghosh and S. Boyd. Upper bounds on algebraic connectivity via convex optimization. *Linear*, 418:693–707, 2006.

[58] A. Gibbons. *Algorithmic Graph Theory.* Cambridge, England: Cambridge University Press, 1985.

[59] A. Goldsmith. *Wireless Communications.* Cambridge Press, 2005.

[60] L. Goncalves, E.D. Bernardo, D. Benson, M. Svedman, J. Ostrowski, N. Karlsson, and P. Pirjanian. A visual front-end for simultaneous localization and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[61] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear non-gaussian bayesian state estimation. In *IEEE Proceedings F, Radar and Signal Processing*, volume 140, pages 107–113, 1993.

[62] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2006.

[63] T. Guixian, H. Tingzhu, and C. Shuyu. Bounds on the algebraic connectivity of graphs. *Advances in Mathematics*, 41(2):217–224, 2012.

[64] H. Hajjdiab and R. Laganiere. Vision-based multi-robot simultaneous localization and mapping. In *Proceedings of Canadian Conference on Computer and Robot Vision*, pages 155–162, 2004.

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

[66] R.I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

[67] R.I. Hartley. Minimizing algebraic error. In *Proceedings of DARPA Image Understanding Workshop*, pages 631–637, 1997.

[68] K. Hashimoto, A. Aoki, and T. Noritsugu. Visual servoing with redundant features. In *Proceedings of the IEEE Decision and Control*, volume 3, pages 2482–2483, 1996.

[69] V. Hedau, D. Hoiem, and D. Forsyth. Recovering the spatial layout of cluttered rooms. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1849–1856, 2009.

[70] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1106–1112, 1997.

[71] R. Hess. Sift feature detector. http://web.engr.oregonstate.edu/ hess/.

[72] S. C. Hirtle and J. Jonides. Evidence for hierarchies in cognitive maps. *Memory and Cognition*, 13:208–217, 1985.

[73] K.L. Ho and P. Newman. Multiple map intersection detection using visual appearance. In *International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2005.

[74] F. Hoffmann, T. Nierobisch, T. Seyffarth, and G. Rudolph. Visual servoing with moments of sift features. In *Proceedings of IEEE International Conference on Systems, Men and Cybernetics*, pages 4262–4267, 2006.

[75] I. Horswill. Visual collision avoidance by segmentation. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 2, pages 902–909, 1994.

[76] A. Howard, S. Siddiqi, and G. Sukhatme. An experimental study of localization using wireless ethernet. In *International Conference on Field and Service Robotics*, 2003.

[77] J. Huang, D. Millman, M. Quigley, D. Stavens, S. Thrun, and A. Aggarwal. Efficient, generalized indoor WiFi GraphSLAM. In *IEEE International Conference on Robotics and Automation*, pages 1038–1043, 2011.

[78] W. H. Huang and K. R. Beevers. Topological map merging. *The International Journal of Robotics Research*, 24(8):601–613, 2005.

[79] W.H. Huang and K.R. Beevers. Topological mapping with sensing-limited robots. In *Proceedings of the International Workshop on the Algorithmic Foundation of Robotics*, pages 367–382, 2004.

[80] W.H. Huang and K.R. Beevers. Topological map merging. *International Journal of Robotics Research*, 24(8):601–613, 2005.

[81] S. Hutchinson, G.D. Hager, and P.I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

[82] R.A. Peters II, K.E. Hambuchen, and K. Kawamura. The sensory egosphere as a short-term memory for humanoids. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, pages 451–459, 2001.

[83] W. Jeong and K.M. Lee. Cv-slam: A new ceiling vision based slam technique. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3195–3200, 2005.

[84] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[85] V. Kann. On the approximability of the maximum common subgraph problem. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*, pages 377–388, 1992.

[86] N. Karlsson, E.D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M.E. Munich. The vslam algorithm for robust localization and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 24–29, 2005.

[87] K. Kawamura, A.B. Koku, D.M. Wilkes, R.A. Peters II, and A. Sekmen. Toward egocentric navigation. *International Journal of Robotics and Automation*, 17(4):135–145, 2002.

[88] T. Keskinpala, D.M. Wilkes, K. Kawamura, and A.B. Koku. Knowledge-sharing techniques for egocentric navigation. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2469–2476, 2003.

[89] S. Kirkland. Algebraic connectivity for vertex-deleted subgraphs, and a notion of vertex centrality. *Discrete Mathematics*, 310(4):911–921, 2010.

[90] S. Kirkland, C.S. Oliveira, and C.M. Justel. On algebraic connectivity augmentation. *Linear Algebra and its Applications*, 435(10):2347–2356, 2011.

[91] H. Kitano, Y. Kuyinoshi, I. Noda, M. Asada, H. Matsubara, and E. Osawa. Robocup: A challenge problem for ai. *Artificial Intelligence Magazine*, 18(1):73–85, 1997.

[92] K. Konolige, D. Fox, B. Limketkai, J. Ko, and B. Steward. Map merging for distributed robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 212–217, 2003.

[93] K. Kosnar, T. Krajnik, and L. Preucil. Visual topological mapping. In *Proceedings of European Robotics Symposium*, volume 44, pages 333–342, 2008.

[94] R. Kumar. Bound for eigenvalues of a graph. *Journal of Mathematical Inequalities*, 4(3):399–404, 2010.

[95] A. Ladd, K. Bekris, A. Rudys, D. Wallach, and L. Kavraki. On the feasibility of using wireless ethernet for indoor localization. *IEEE Transactions on Robotics and Automation*, 20(3):555–559, 2004.

[96] H.-C. Lee and B.-H. Lee. Improved feature map merging using virtual supporting lines for multi-robot systems. *Advanced Robotics*, 25(13-14):1675–1696, 2011.

[97] S. Lenser and M. Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 886–891, 2003.

[98] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, 1944.

[99] X. Li and Y. Shi. Note on a relation between randic index and algebraic connectivity. http://arxiv.org/pdf/1012.4856.pdf, 2010.

[100] Z. Lin, S. Kim, and I. S. Kweon. Recognition-based indoor topological navigation using robust invariant features. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2309–2314, 2005.

[101] J. Liu and R. Hubbold. Automatic camera calibration and scene reconstruction with scale-invariant features. In *Proceedings of International Symposium on Visual Computing*, 2006.

[102] S.P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[103] H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, 1981.

[104] G. Lopez-Nicolas, C. Sagues, J.J. Guerrero, D. Kragic, and P. Jensfelt. Nonholonomic epipolar visual servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2373–2384, 2006.

[105] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[106] M. Lu, L.Z. Zhang, and F. Tian. Lower bounds of the laplacian spectrum of graphs based on diameter. *Linear Algebra and its Applications*, 420(2-3):400–406, 2007.

[107] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1981.

[108] J. Luo, A. Pronobis, B. Caputo, and P. Jensfelt. The kth-idol2 database. Technical report, KTH Royal Institute of Technology, CVAP/CAS, 2006.

[109] Y. Ma, S. Soatto, J. Kosecka, and S.S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2003.

[110] R. Madhavan, C. Scrapper, and A. Kleiner. Special issue on "characterizing mobile robot localization and mapping". *Autonomous Robots*, 2009.

[111] E. Malis and P. Rives. Robustness of image-based visual servoing with respect to depth distribution errors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 1056–1061, 2003.

[112] L. Maohai, H. Bingrong, and L. Ronghua. Novel method for monocular vision based mobile robot localization. In *Proceedings of International Conference on Computational Intelligence and Security*, volume 2, pages 949–954, 2006.

[113] G. L. Mariottini, G. Oriolo, and D. Prattichizzo. Image-based visual servoing for nonholonomic mobile robots using epipolar geometry. *IEEE Transactions on Robotics*, 23(1):87–100, 2007.

[114] G.L. Mariottini and D. Prattichizzo. Epipole-based visual servoing for nonholonomic mobile robots. In *Proceedinds of the IEEE International Conference Robotics and Automation*, volume 1, pages 497–503, 2004.

[115] T.K. Marks, A. Howard, M. Bajracharya, G.W. Cottrell, and L. Matthies. Gamma-slam: Using stereo vision and variance grid maps for slam in unstructured environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3717–3724, 2008.

[116] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[117] S. Martens, P. Gaudiano, and G.A. Carpenter. Mobile robot sensor integration with fuzzy artmap. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, Intelligent Systems and Semiotics, and Intelligent Control joint conference*, pages 307–312, 1998.

[118] M.C. Martin. Evolving visual sonar: Depth from monocular images. *Pattern Recognition Letters*, 27(11):1174–1180, 2006.

[119] Y. Matsumoto, I. Masayuki, and H. Inoue. Visual navigation using view-sequenced route representation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 83–88, 1996.

[120] Y. Matsumoto, K. Sakai, M. Inaba, and H. Inoue. View-based approach to robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1702–1708, 2000.

[121] G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, Inc., Hoboken, NJ, 2000.

[122] T. McNamara. Mental representations of spatial relations. *Cognitive Psychology*, 18(1):87–121, 1986.

[123] R. Merris. Laplacian matrices of graphs: a survey. *Linear Algebra and its Applications*, 197–198:143–176, 1994.

[124] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *Proceedings of the International Conference on Computer Vision*, pages 128–142, 2002.

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

[126] B. Mohar. Eigenvalues, diameter and mean distance in graphs. *Graphs and Combinatorics*, 7:53–64, 1991.

[127] B. Mohar. The laplacian spectrum of graphs. *Graph Theory, Combinatorics, and Applications*, 2:871–898, 1991.

[128] B. Mohar. Some applications of laplace eigenvalues of graphs. In *Graph Symmetry: Algebraic Methods and Applications*, volume 497, pages 225–275, 1997.

[129] D. Mosk-Aoyama. Maximum algebraic connectivity augmentation is np-hard. *Operations Research Letters*, 36(6):677–679, 2008.

[130] M. Muja and D.G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[131] J. Mulligan and G. Grudic. Topological mapping from image sequences. In *Proceedings of IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, 2005.

[132] J. Nieto, J.E. Guivant, and E.M. Nebot. The HYbrid Metric Maps (HYMMs): a novel map representation for DenseSLAM. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 391–396, 2004.

[133] T. Ohno, A. Ohya, and S. Yuta. Autonomous navigation for mobile robots referring pre-recorded image sequence. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, volume 2, pages 672–679, 1996.

[134] N. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics*, pages 105–112, 2001.

[135] N. Ergin Özkucur and H. Levent Akin. Cooperative multi-robot map merging using fast-SLAM. In J. Baltes et al., editor, *Robocup 2009: Robot Soccer World Cup XIII*, pages 449–460. Springer, 2010.

[136] N.P. Papanikolopoulos and P.K. Khosla. Adaptive robotic visual tracking: Theory and experiments. *IEEE Transactions on Automatic Control*, 38(3):429–445, 1993.

[137] N. Pears, B. Liang, and Z. Chen. Mobile robot visual navigation using multiple features. *EURASIP Journal on Applied Signal Processing*, 1:2250–2259, 2005.

[138] A. Pronobis and B. Caputo. The kth-indecs database. Technical report, KTH Royal Institute of Technology, CVAP/CAS, 2005.

[139] A. Quattoni and A.Torralba. Recognizing indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420, 2009.

[140] A. Ranganathan. *Probabilistic Topological Maps*. PhD thesis, Georgia Institute of Technology, 2008.

[141] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[142] R. Rodrigo, Z. Chen, and J. Samarabandu. Feature motion for monocular robot navigation. In *Proceedings of International Conference on Information and Automation*, 2006.

[143] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *Workshop on "On-line Learning for Computer Vision" at IEEE ICCV*, 2009.

[144] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi. Divergence stereo for robot navigation: Learning from bees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1993.

[145] J. Santos-Victor, R. Vassallo, and H. Schneebeli. Topological maps for visual navigation. In *Proceedings of International Conference on Computer Vision Systems*, pages 21–36, 1999.

[146] L. Schwardt and J. du Preez. Clustering. Technical Report PR414 / PR813 - 2005, University of Stellenbosch, 2003.

[147] S. Se, D. Lowe, and J. Little. Local and global localization for mobile robots using visual landmarks. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 414–420, 2001.

[148] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 2051–2058, 2001.

[149] S. Se, D. Lowe, and J. Little. Mobile robot localization and mapping with uncertainty using scale-invariantvisual landmarks. *International Journal of Robotics Research*, 21(8):735–758, 2002.

[150] A. Shademan and F. Janabi-Sharifi. Using scale-invariant feature points in visual servoing. In *Proceedings of International Society for Optical Engineering*, 2004.

[151] F. Shu, L. Toma, W. Neddermeyer, and J. Zhang. Precise online camera calibration in a robot navigating vision system. In *Proceedings of the IEEE International Conference on Mechatronics and Automation*, volume 3, pages 1277–1282, 2005.

[152] R. Sim and J.J. Little. Autonomous vision-based exploration and mapping using hybrid maps and rao-blackwellised particle filters. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2006.

[153] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, 2003.

[154] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[155] D. Tax. *One-class classification; Concept-learning in the absence of counter-examples*. PhD thesis, Delft University of Technology, 2001.

[156] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*, chapter 5,8. The MIT Press, Cambridge, Massachusetts, 2005.

[157] M. Tomono. 3-d localization and mapping using a single camera based on structure-from-motion with automatic baseline selection. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3342–3347, 2005.

[158] M. Tomono. 3-d object map building using dense object models with sift-based recognition features. In *Proceedings of IEEE International Conference of Intelligent Robots and Systems*, pages 1885–1890, 2006.

[159] P.H.S. Torr and D.W. Murray. The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3):271–300, 1997.

[160] A. Torralba, K.P. Murphy, W.T. Freeman, and M.A. Rubin. Context-based vision system for place and object recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, 2003.

[161] D. Tran and T. Nguyen. Localization in wireless sensor networks based on support vector machines. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):981–994, 2008.

[162] T.T.H. Tran and E. Marchand. Real-time keypoints matching: application to visual servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3787–3792, 2007.

[163] W.T. Tutte. *Graph Theory as I Have Known It*. Oxford, England: Oxford University Press, 1998.

[164] C. Valgren, A. Lilienthal, and T. Duckett. Incremental topological mapping using omnidirectional vision. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 3441–3447, 2006.

[165] M. West. *Bayesian Statistics 4*, chapter Modelling with mixtures, pages 503–524. Clarendon Press, London, 1992.

[166] D. Wooden. A guide to vision-based map building. *IEEE Robotics & Automation Magazine*, 13(2):94–98, 2006.

[167] K.W. Wurm, C. Stachniss, and G. Grisetti. Bridging the gap between feature- and grid-based SLAM. *Robotics and Autonomous Systems*, 58(2):140–148, 2010.

[168] M. Youssef, A. Agrawala, A. Shankar, and S. Noh. A probabilistic clustering-based indoor location determination system. Technical Report CS-TR-4350, University of Maryland, College Park, 2002.

[169] W. Zhang and J. Kosecka. Image based localization in urban environments. In *Proceedings of the International Symposium on 3D Data Processing, Visualization, and Transmission*, 2006.

[170] Z. Zhang. Determining the epipolar geometry and its uncertainty a review. *International Journal of Computer Vision*, 27(2):161–195, 1998.

[171] S. Zickler and M. Veloso. RSS-based relative localization and tethering for moving robots in unknown environments. In *IEEE International Conference on Robotics and Automation*, pages 5466–5471, 2010.

[172] J.B. Ziegler and N.B. Nichols. Optimum settings for automatic controllers. *ASME Transactions*, 64:759–768, 1942.

[173] Z. Zivkovic, B. Bakker, and B. Krose. Hierarchical map building using visual landmarks and geometric constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2480–2485, 2005.

[174] Z. Zivkovic, B. Bakker, and B. Kröse. Hierarchical map building using visual landmarks and geometric constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2480–2485, 2005.

[175] Z. Zivkovic, B. Bakker, and B. Krose. Hierarchical map building and planning based on graph partitioning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 803–809, 2006.

[176] Z. Zivkovic, O. Booij, and B. Krose. From images to rooms. *Robotics and Autonomous Systems*, 55(5):411–418, 2007.

[177] Z. Zivkovic, O. Booij, B. Kröse, E.A. Topp, and H.I. Christensen. From sensors to human spatial concepts: An annotated data set. *IEEE Transaction on Robotics*, 24(2):501–505, 2008.