SPLINTR:

Spatial Place Recognition in a Topologically Mapping Robot

by

Bradley Joe Wimpey

(Under the direction of Walter D. Potter)

Abstract

For visual approaches to simultaneous localization and mapping, the use of visual keypoint features is a popular choice. Certain areas of the environment, however, may contain ambiguous architecture or areas of repetition, which may cause keypoints to be matched well at multiple locations; it would be useful to consider other sources of evidence in addition to keypoints to make decisions. We propose a system that samples from varying layers of a location's signature, from global or simple properties such as color and straight lines, to texture related characteristics in keypoints, to symbolic, human characteristics and semantic information such as text recognition and object detection and recognition. We propose how these visual feature "ridges" can be associated together and utilized to form our version of a visual "fingerprint" of a place, how these fingerprints can be compared for localization, and how they can be linked together to form a topological map of the environment. This new approach is called SPLINTR: **S**patial **Pl**ace Recognition **in** a **T**opologically Mapping **R**obot.

Index words: robotics, mapping, localization, machine vision, SLAM

SPLINTR:

Spatial Place Recognition in a Topologically Mapping Robot

by

Bradley Joe Wimpey

B.S., Computer Science, University of Georgia, 2003

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2012

SPLINTR:

Spatial Place Recognition in a Topologically Mapping Robot

by

Bradley Joe Wimpey

Approved:

Major Professor: Walter D. Potter

Committee: Prashant Doshi
Suchendra Bhandarkar

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2012

SPLINTR:

Spatial Place Recognition in a Topologically Mapping Robot

Bradley Joe Wimpey

Acknowledgments

I would like to thank my major professor, Don Potter, for all of his time, dedication, guidance and suggestions in this dissertation work. Also, my committee members, Suchi Bhandarkar and Prashant Doshi, for all of their time and help in this research work. Furthermore, my family, for always being there for me and providing so much support. A thank you goes out to all of my friends for suggestions, comments, encouragements, prayers, meals, time, proof-reading, and laughs. Furthermore, I would like to thank the Computer Science Department and the wonderful group at Veterinary Medicine for the assistantship opportunities over the years. And my Heavenly Father for His many blessings and help in my life.

Contents

List of Figures

Chapter 1

Introduction and Motivation

Autonomous localization and mapping for a mobile robot is a difficult task; to accomplish this goal, the robot must have a way of storing a representation of its environment while it explores, as well as using information from its current scene to compare with locations in its map to form a hypothesis on its whereabouts within the map. Certain approaches in the appearance-based domain of simultaneous localization and mapping (SLAM) involve the use of keypoints, that is, interesting texture points or patches within an image. While this may be useful in textured areas of the environment, what about areas with less texture, such as smooth colored walls of a hallway or office wall? True, keypoint extraction is still possible, but what are these keypoints telling us? They may be easily matched to other, similar areas of the environment, like similar areas of brick walls or cement blocks. What about when areas of an environment are similar but differing in color, which may not be detected as different when using a single feature extractor? These situations call for a more robust approach.

What are the benefits of having a robot discover where it is and make a map as it explores? Being able to extract features and intelligent representations of an environment is useful in military or police exercises, such as deploying a robot into a hostile building.

Given such a situation, the robot could perform tasks based on its location or extract useful visual information as it explores, all without the danger of losing a human life. Further benefits exist within service robotics. For robots to be aware of their location and to form connections between locations, they could autonomously roam a person's home and perform tasks, search for items in specific locations, or fetch or deliver items to locations. Such actions could also be useful to provide assistance to the elderly and the disabled. Furthermore, when information such as semantic text and objects is extracted from the environment, it opens the door for higher level reasoning, including intelligent searching, place classification and planning.

The domain we are interested in is an office environment. Such an environment, however, can be repetitive – there may only be subtle differences between one location in a hallway versus another hallway location. When using keypoint extraction, we may run into issues of "common" keypoints, those features extracted from repetitious architectural features such as corners of nameplates on a door, crevices between blocks in the wall, or areas between a door and a keycard scanner (which may indeed form a keypoint feature, but may appear next to many different doors). What about being able to distinguish locations that are similar visually except different in other regards, such as the text on a sign or a differing logo on a poster or sticker?

The robot may encounter areas where relying on one set of features is not sufficient (and ill-advised). Corridors may have similar texture but be distinguishable in other regards, such as color, text, objects or other characteristics. These issues can be addressed when we include other sources for decision making on localization, if another source offers differing (or corroborating) evidence. We chose other features to provide distinguishing dimensions, including symbolic or semantic information that humans use in the environment. Our method of semantic mapping uses these features not only to populate and annotate the map, but to localize as well. Furthermore, if areas are lacking in texture, features may still be extracted

even though they may be generic in nature. Therefore, our system also extracts generic image features like color distribution information and statistics from detected lines.

We report here what we have done toward the realization of our goal of simultaneous localization and topological mapping.

1.1 Applications to Semantic SLAM

For this research and experimental application, we mainly focus on an indoor environment, particularly areas with semantic context. A location can be composed of a variety of visual features, be it objects, texture, structure and lines, color, and alphabetical and numeric text. To be able to sample over this gamut of visual clues what features are present at a certain location would greatly increase an autonomous robot's ability to recognize where it is and store where it has been, along with being able to connect differing locations – the essentials of topological localization and mapping.

There may be locations within the environment where the robot needs more than key-points alone to make a decision. Consider the environments constructed for everyday use, such as our homes, our places of work, our commercial businesses, places of worship, and museums to name a few. Various locations within these buildings could contain building specific objects, logos or symbols, text features, straight lines resulting from the architecture of the environment and surroundings, as well as texture and color schemes, and each may vary location to location within the environment. We would like an intelligent robot to use more than one source of visual evidence and to utilize a more intelligent representation of space as well, calling upon the extraction and use of various features as visual evidence of location. With other layers to provide support, we produce a more robust system, build a more intelligent representation of space, and allow the robot to utilize multiple sources of visual evidence to recognize and map locations. Furthermore, we as humans usually do

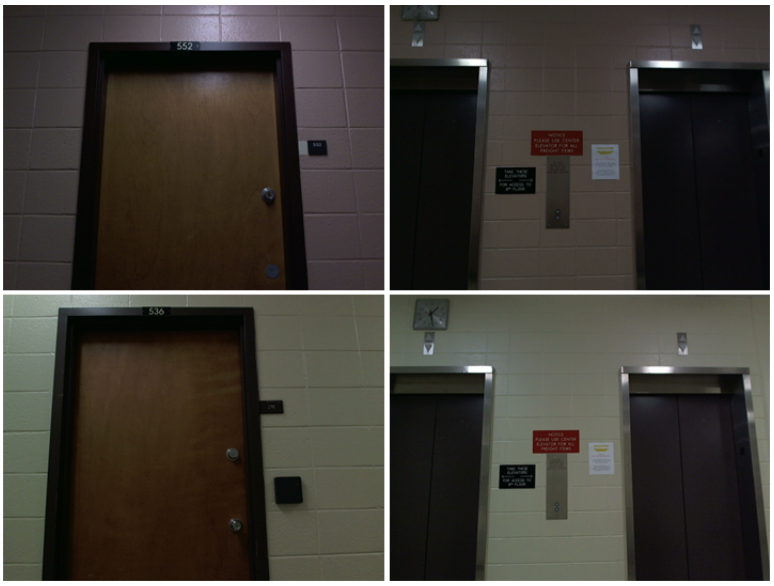

Figure 1.1: Locations in the Boyd Graduate Student Research Center building, illustrating areas of repetition but differing in certain regards to the visual signature.

not depend on a single source of information to make decisions on our location or to form relationships between places. We draw from multiple visual cues like text and signs, structural architecture and the straight lines that result, color schemes of the surroundings, and the objects recognized in an environment. Therefore, we formulated a visual mapping and localization method similar to how humans recognize places, relying upon multiple machine vision features to sample from different layers of the visual representation/structure of a place – a location's signature.

We chose different feature extractors in order to sample from various segments of this notion of a location signature. To capture interesting details existing in textured regions of an image, we use a keypoint feature extractor. Keypoint features can also be used to recognize objects, so object recognition was added to the system, providing a labeling to be applied to certain sets of keypoints. We also added object detection, which employs the use of boosted classifiers [76, 38] to detect instances of objects within a scene. We

added text recognition to the system, as text can be found rather ubiquitously in man-made environments, both indoors and out. To cope with areas with fewer interesting features, the system creates a color histogram to capture the color distribution. Straight line statistics are used as the final technique, collecting information on line lengths, orientation and color. Similar to how the patterns of friction ridges on the tips of fingers form human fingerprints [63], these visual feature "ridges" form our version of location "fingerprints." When these features are encapsulated into location fingerprints, they can be associated together to form a topological map and compared with one another for localization. We call our system SPLINTR, **S**patial **Pl**ace Recognition **in** a **T**opologically Mapping **R**obot.

If the robot is to use these visual signatures in an intelligent way to build a map, and if the robot is to be able to assess if it is in unexplored territory or if it is traveling within the confines of its map, it needs a way of estimating the similarity between visual fingerprints. Based on the evidence presented in the images of the robot's surroundings, how can it measure how close the evidence comes to the data within its map? Briefly introducing this subject, to compare different locations' fingerprints, we compare their individual ridges and then combine the individual ridge feature comparison results together into one overarching similarity measure. Details on how we compare each ridge, and how they are combined to form one single matching result are discussed later in the Localization chapter, specifically section 4.1.

1.2 Contributions of this Work

- We present a unique approach to place representation for an autonomous mobile robot, combining a novel mix of visual feature extraction, forming a new way to allow robots to recognize and record locations.

- This project attempts a style of visual feature fusion. We propose an approach to represent locations in a manner not attempted before, combining features from different visual complexity levels (from lines and color distribution to texture keypoints, objects and text).
- We present our method of comparing various genres of visual features, separately comparing each ridge and combining the results into a single numerical similarity score between fingerprints.
- We apply a multi-resolution focus of attention process in the text detection/recognition pipeline, combining the results of text detection pixel classification to provide better input to the text recognition engine.
- We adapt a text detection algorithm and an open-source Optical Character Recognition (OCR) engine for use in scene text extraction, using heuristics and dictionary validation to reduce erroneous text results.
- We introduce our approach to measuring similarity using keypoints, and to temper this based on the average number of keypoints extracted.
- We also contribute to the ideas of maintenance of location hypotheses and topological map building methodology, using off the shelf data structures such as vectors and graphs and the evidence presented by visual fingerprints.
- We use existing tools in a specialized way, to extract features and semantic information from specific layers of the visual representation of a place, which we refer to as the signature of a location.

1.3 Dissertation Outline

The rest of the dissertation is organized as follows. In Chapter 2, we discuss research related to topological and semantic localization and mapping, as well as analyze related topics such

as visual features, data representation and comparison measures. Chapter 3 details our visual feature fingerprints, describing the extraction methodology of each ridge. In Chapter 4, we cover how to compare the individual fingerprint ridges and examine SPLINTR's use in robotic localization. In Chapter 5, we discuss using the SPLINTR fingerprints for topological SLAM. And in Chapter 6, we conclude the dissertation with a summary of contributions and discussion of future work.

Chapter 2

Related Work

There are important aspects to keep in mind when designing an appearance-based topological fingerprint mapping system. One issue is the choice of visual features; what machine vision techniques could be used for localization and mapping? How will the features represent locations; in what manner are the topological nodes and individual layers of those nodes to be represented and stored on the robot? Once the features are stored, how are we to measure the similarity between the features of the current scene and those in the map? How should we combine the individual feature set similarity scores to get a score between entire nodes?

In this chapter, we discuss various machine vision techniques that have been employed by others to address the problems of recognition, localization and mapping, as well as methods used to compare features and represent locations.

2.1 Topological and Semantic Mapping

FAB-MAP [20] is, as the authors describe it, an "appearance-only SLAM" technique, where an input scene is classified as either a place visited before (from the map) or a new location. Appearance-based SLAM appears to be a topological localization and mapping approach,

since the map is composed of discrete, individual locations. These locations are represented in their probabilistic bag-of-words model, with the bag-of-words vocabulary consisting of keypoint features. The representation of locations is probabilistic in that they use a probability distribution over the vocabulary of features. In FAB-MAP 2.0 [21], Cummins and Newman extend the FAB-MAP method by approximating their previous model in order to implement an "inverted index" representation of visual vocabulary words at locations in order to support large scale mapping trajectories (approximately 1,000 kilometers in their experimentation). When localizing, the FAB-MAP system does not return a similarity measure, but instead returns the probability that the scene and location(s) in the map came from the same place. Their probabilistic function incorporates a method to suppress non-distinctive locations, those that look similar and common, such as scenes containing shrub bushes or brick walls. In their calculation of the probabilistic location given observations, the normalizing term is divided into probabilities that the observation came from the map and the probability the observation came from an unmapped portion of the environment. This second probability is estimated with a sampling method that creates random location models; if the place is distinctive, then it is less likely that they can randomly sample a place with equal or higher probability, but if it is a less distinctive location, then it is more likely that they can build up a similar random location model. This would create a larger denominator in their normalizing factor, and therefore a lower probability of the scene coming from a mapped location, thus handling perceptual aliasing.

Vasudevan et al. [74] created combination metric/topological maps of the environment, with metrical measures encoding object locations, recognized based on SIFT features. In this work, doors represent the separation between locations, and recognized objects are used to classify the place the robot is in. Similarity between a map location and the robot's current location is measured by matching the recognized objects and relationships (distance and angle measurements) among those objects.

Case et al. [16] manually drove a robot around to create a map using the GMapping SLAM tool of the Robot Operating System (ROS). Once the map was created, images were collected from a set of locations to visit. The authors attempted to extract only signposts of room numbers and names, simplified by using the ADA guidelines for signpost placement within an office building. Text detection was done by extracting a set of image features, including local pixel variance, local edge strength, and vertical and horizontal edge strength. They trained a logistic regression classifier using the maximum, minimum and average values within a 10 x 10 pixel window of these extracted image features (thus they classify one 10 x 10 box of pixels at a time, then move the window 3 pixels and classify again). The window is applied over the image to output the probability of containing text, and then the image was thresholded and binarized. After heuristically thinning the detection results, each text detected region (after connected component labeling) was binarized using nine different thresholds, with each applied through the Tesseract engine, and the result with the highest confidence score (a value returned by Tesseract) was kept as the text resulting from the region. For text extracted within the same 3D location (within a certain distance), the authors attempted to merge these texts using the Needleman-Wunsch sequence alignment algorithm [50]; if two texts have an alignment score exceeding a certain threshold, then the two text pieces were replaced with the optimally aligned text. The authors' goals here were slightly different than our own. Instead of using the text to localize and form a map, the text was used to annotate an already built map. Furthermore, they did not use a dictionary or rules to validate words. However, they did include a user query interface where a user can type in a name or room number, and using the Smith-Waterman alignment score [68], the robot navigated to the best match in the map.

Posner et al. [55] developed a system for semantic image retrieval. They applied text detection and recognition to street view images acquired from a mobile robot. The authors trained a cascade of boosted classifiers for text detection, combined with Tesseract OCR for

recognition. They applied a probabilistic model to spelling corrections for recognized text; the incorporation of the prior probability of a word occurring in an image is accomplished by using the word frequencies provided in the British National Corpus [18]. The authors mentioned that use of this approach, especially on their dataset which included French names, may lead to false corrections of spelling. The authors also provided a probabilistic model to return subject-related images when provided a subject query, and conversely to provide an image's subject given its set of text detections.

Rogers et al. [59] used textual semantic information in their SLAM approach. They used door sign text to represent the desired semantic objects to extract from the environment. In order to create their door sign detector, they first applied a saliency operator [28] to extract image regions. These regions were hand labeled, and a rescaled version was used to train a Support Vector Machine (SVM) on the result of extracting the Histogram of Oriented Gradients (HOG) feature from these regions. The map created is in the form of a graph map, which stored measurement locations of wall and text data extracted from the environment as they were situated relative to the robot's pose. By including text as well as the location of the text within the map, they found that they could correct errors within their map, such as when a robot is lost within its map due to pose errors. By matching text extracted from a door sign in the environment with that stored in its map, the robot can reduce its pose error and correct its map. They used GoogleGoggles rather than an onboard text recognition engine, thus this approach would require network or cellular connectivity. They also limited text extraction strictly to door signs, rather than scene text in general, and assumed a certain number-text pattern on the door signs.

2.2 Visual Features In Location Recognition and Mapping

In this section, we review visual features used for localization and mapping, including rather simple machine vision results, interesting keypoints, color histograms, and recognition of text.

2.2.1 Use of Simple Visual Features

There has been work in the area of location recognition using simple image features. In his work, Braunegg [14] extracted straight-line segments from stereo images as features to build occupancy grid maps (locations of the features in the scene) of rooms of the robot's environment. In [34], Lamon et al. used a robot to localize itself using location "fingerprints" represented as strings of characters. Each character in the string fingerprint represented a detected color patch or a vertical straight line edge. For each feature detected in the image, its representative character was recorded in the string fingerprint in the order it was detected at that location. Locations within the map were also represented as strings. The robot would then try to localize by string-matching the resulting character-based fingerprint of the current location to one in the map. In [70], Tapus and Siegwart extended the string fingerprint to also include corners detected from laser scans. This time the string fingerprints were used for topological mapping (as opposed to the localization method of [34]), with new nodes in the topological map created based on a "dissimilarity" threshold. Also, each node was created from an average of similar, nearby fingerprints.

2.2.2 Use of Histograms

Work has also been done to apply histograms to localizing a mobile robot. Blaer and Allen [10] used histograms and an omnidirectional camera for coarse localization in their AVENUE project, where the histogram localization is used as a precursor to reduce the set of regions the robot could be in and thin the search space for a finer visual localization module. Rañó et al. [57] estimated color histograms using self-organizing maps (SOM), with regions of a limited sequence of previous images used to train the SOM on the fly. The distance measures between the obtained histogram of the current location and the histograms of stored locations in the map decided the location of the robot.

2.2.3 Use of Text Recognition

We have introduced previously some related works involving text recognition and robotics. Furthermore, Tomono and Yuta [72] equipped a robot with the ability to navigate to a specific room within a hallway corridor. The robot could detect doors through vertical edges and color matching; once a door is detected, the robot could maneuver itself to recognize room number plates and use character matching to recognize the room number. Liu et al. [40] also enabled a mobile robot to follow directions from a starting point to a goal location within the environment, as well as to extract text landmarks of Japanese characters from the environment as part of their localization and mapping technique.

2.2.4 Use of Feature Keypoints

In [23], Filliat combined SIFT points with hue and grayscale histograms of the image in an interactive localization and mapping technique. To compute the histograms, Filliat divided the image into smaller windows and computed the hue and grayscale histograms for each window. Murillo et al. [49] used Speeded Up Robust Features (SURF) [8] in their method

of topological and metric localization. In their localization method, Botterill et al. [12] utilized SURF combined with small hue histograms around each feature point for added depth to the feature descriptors. Also, Neubert et al. [51] implemented FastSLAM [47] using SURF features. In their localization system, Wang et al. [77] detected features using the Harris-Laplace feature detector [45], and they represented those feature points using the SIFT feature descriptor format. Cummins and Newman [20] used SIFT features to represent locations in their original FAB-MAP research, introduced previously. The authors discussed, however, that FAB-MAP is formulated to not be keypoint specific, as FAB-MAP 2.0 [21] used SURF features. They also discussed that it can be used with other sensor input as well, not just with images.

2.3 Methods of Representing Data and Measuring Similarity

In this section, we analyze various data representation techniques used for localization and mapping, or related problems such as image retrieval and landmark recognition. We furthermore discuss various methods used in comparing features to calculate a similarity or distance measure.

2.3.1 Maintaining Information with Ontologies

One possible means of storing location information is to incorporate the use of an ontology or some semantic storage facility. Maillot and Thonnat [42] used an ontology to hold various visual concepts and hierarchies of those concepts, including texture, color, and spatial information such as geometrical shape and size. This hierarchical descriptive framework was used to separate concepts and subsequently train classifiers to be used to classify objects.

Durand et al. [22] used an ontology to structure and relate the objects they needed to recognize from satellite imagery and to hold value ranges of attributes of the objects. After segmentation, features of those regions were extracted, and the regions extracted from the images were matched against the concepts and values contained in the ontology. Similarity scores were calculated between the features of the regions and the attribute value ranges within the ontology.

2.3.2 Locations Represented Using Histograms

As introduced earlier, a simple structure to represent images or locations is the use of histograms. Ulrich and Nourbakhsh [73] used a robot for place recognition, with omnidirectional camera images converted into six one-dimensional color histograms, one for each channel in HSL (hue, saturation, luminance) and RGB (red, green, blue) color spaces. For the environmental map, they hand labeled images from the environment as specific locations, with each location getting n reference images. To compare the environmental map's color histograms with the input images, they tested several metrics in comparing histograms, including χ^2, and found that the Jeffrey divergence gave them the best results. They give the Jeffrey divergence as

$$d(H,K) = \sum_i \left(h_i \log \frac{2h_i}{h_i + k_i} + k_i \log \frac{2k_i}{h_i + k_i} \right)$$

when comparing two histograms H and K, with entries h_i and k_i. They only consider the hypothesized location and its neighbors as matching candidates with the input image. Each color channel votes for the best matching location (via the smallest matching distance from the Jeffrey divergence measure). Each color channel (or color band) also calculates a confidence measure c_b to accompany the vote, $c_b = 1$ - minimum Jefferey divergence measure distance/ second minimum Jefferey divergence measure distance. Thus the distance between the closest and second closest histogram matches determines the confidence (a similar method

is employed by Lowe [41] for matching individual SIFT features, as matches must be a certain distance from the next nearest match). Each channel's confidence must be above a threshold to cast a vote for a location, and all confident votes must be unanimous to update the robot's belief of its location. Otherwise, the system does not classify the input image, and the robot's location belief remains at the last confident classified location.

In [11], Blaer and Allen extended the system employing only histograms for localization [10] to include signal strengths of wireless networks as well. Their histograms were also changed to multiresolution histograms of RGB color channels (blurred and downsampled to create the histogram resolutions). The histograms were normalized so that they affect the difference measure equally. They used a histogram difference measure, where bins between the two histograms are compared, and the absolute value difference between each bin is summed, and the difference of each bin of each channel is summed. Their histogram difference appears to be the L_1 distance measure, but modified to sum up the bin differences of each channel and each resolution. This value represented the distance measure between the histograms. The histogram difference was normalized to a range 0 to 1. Access point strengths were normalized from a range of about -80 dBm to -20 dBm to a range 1 to 50. The access point strength sets for the current location and the database entry were compared, and the absolute differences between the strengths were summed to represent the difference measure. The result was again normalized to be within a 0 to 1 range, and the overall difference was a weighted sum of the two difference measures (color and wireless network strength). The authors mentioned that the overall difference measure value could be used as a confidence measure of the location estimate of the robot. This method is used to find which region a robot is in (which they divided part of the Columbia University campus into 13 regions in their experimentation of outdoor localization).

2.3.3 Locations as SIFT Feature Sets

Kosecka and Yang [29] represented the map of the environment as a database of "views," or one to four representative images of individual locations, with each view represented by SIFT features. Localization is done by voting of feature matches between the query image and all views of the map. They also attempted to capture relationships between adjacent locations using a Hidden Markov Model, an n x n adjacency matrix to represent the probability of locations being next to each other for the n locations, and an observation likelihood calculated by the number of matches between a location and the query image divided by the sum of the matches amongst all locations.

Regarding representation, the global map of [65] was composed of local, smaller maps represented by sets of SIFT features that the authors call "SIFT fingerprints." As for similarity matching, SIFT fingerprints of the robot's input images were only matched with fingerprints within an uncertainty area of the robot. Within that area, SIFT features were compared based on Euclidean distance, nearest neighbors were selected as matches, and outliers were eliminated with RANSAC. Their matching score for fingerprints was a weighted sum of the number of matches with the ratio of inliers to outliers. When a loop is detected using SIFT fingerprint matching, Epipolar geometry was used to update the map of features.

2.3.4 Feature Vocabularies and Data Structures

Another means of location representation is to represent places or images using weights or frequencies of a list of visual features. The choice of representation in [12] was a bag-of-words approach of SURF descriptors combined with color histograms on the SURF patches (a small 11x11 Gaussian weighted patch around the SURF keypoint). In this bag-of-words approach, features were mapped to their nearest representative visual word in a dictionary, so each image can be represented based on visual word frequencies within the image. Similarity of

images can then be analyzed by comparing the vector of frequencies, which [12] weighted by the log of the ratio of frequency over total words, and compared using L_1 (Manhattan) distance. For the authors, pairing the hue histogram with the SURF keypoints improved recognition. To compare the histograms, they found best results using χ^2 and L_2 metrics, but they were comparing relatively sparse histograms. They searched the database on each input image query as opposed to creating an inverse file. As briefly introduced earlier, in [77], their database of locations was represented as a "visual vocabulary," with weights of that vocabulary at each location. They used the Harris-Laplace feature detector, with each feature described by the SIFT descriptor format. Weights were based on a term frequency multiplied by the inverse document frequency, $w = tf * idf$. Term frequency is the number of occurrences of term i in location j divided by the total terms at location j, while the IDF, or inverse document frequency, is the log of (the number of locations in their model / number of locations containing term i). To evaluate the similarity between the vectors of term frequencies (extended to also include orientation of the features), they used the cosine of the angle between the query vector and the target vector from their model. This "coarse" localization result was used to rank the matches, and the best five were considered for "fine" localization. In their fine localization method, SIFT point sets are compared between the current scene and possible locations, and SIFT point matches cast a vote. The location with the most votes was considered the winning location, which was further verified if votes between locations were too close by using RANSAC and discarding the outliers.

2.3.5 Mixture of Features

Next, we outline representations that use multiple features, sometimes consisting of rather simple machine vision results, to represent locations and images. The following reviews are organized according to the end use of the mixture of features – for localization and landmark recognition and for image retrieval, a related problem to localization.

Localization and Landmark Recognition

Zhou et al. [80] combined various image features to represent locations. They combined color histograms of RGB and HSV (hue, saturation, and value) color channels with the zero crossing and texture measures used by Argramon-Engelson [6]. Further features were added, including a measure of gradient magnitude and pixel rank, where pixel rank is a count of pixel intensities less than a pixel within a local neighborhood. These features were combined into a multi-dimensional histogram to represent locations. To compare histograms between the robot's current location and the database, they used the Jeffrey divergence. The authors cut down on searching the entire database by taking advantage of topological map node adjacency and searching nearby nodes to a previously matched location.

In a kind of combination between localization and image retrieval, Hays and Efros [25] provided a localization technique based on how similar a query image is to their database of GPS-tagged images. Their IM2GPS system [25] involved matching scenes of images with a large GPS-tagged image database (6 million images), and using nearest neighbor results, they created a probability density over the Earth of the likelihood of the image coming from locations on the Earth. They tested several image features, and the most useful in their image GPS endeavor were gist [52], color histograms, and texton histograms [43]. Gist images, formed from responses to a set of filters applied to the image, were useful for scene matching, and thus could be considered in the future as an additional layer to SPLINTR. They also created small, 5x5 color (CIE $L^*a^*b^*$) images as a feature. Furthermore, color histograms are already used in our project, but specifically the color space they used for their histograms was CIE $L^*a^*b^*$ color space, separating colors into lightness (L^*), a magenta/green range (a^*, negative is green, positive is magenta) and yellow/blue range (b^*, blue is negative, yellow is positive) [19], with 4x14x14 bins in L^*, a^*, and b^* respectively. For the distance measure between the color histograms, they chose the χ^2 distance. They also formed a texton dictionary [43], or small texture structures, by clustering responses of the database

of images to a set of filters. They also used χ^2 distance to compare texton histograms. Furthermore, they extracted line features and formed a histogram of angles and lengths of the lines, compared using L_1 distance. The authors indicated that line features were useful for separating natural and man-made scenes as well as for determining similar vanishing points between images. They scaled feature distances to bring their standard deviations to be approximately the same so that each feature has equal effect on ranking the scene matches. For a query image, the combined feature distances were used to find the nearest neighbors from the image database, of which the GPS tags were used to find geological location on the Earth. Other features were tested but discarded for experimentation due to their lack of discriminatory ability, including "geometric context" (classifying image regions into categories like sky, ground and "vertical," or parts of the image coming up from the ground) [27] and small 16x16 images.

Image Retrieval

The work of Rahman et al. [58] focused on image retrieval by comparing the contents of a query image to a database and retrieving similar photographs. The three features on which they focused included color, texture and edge features. Color of the image was represented in a 108 dimensional color histogram in the HSV color space (12x3x3 for HSV respectively). Texture features were extracted from co-occurrence matrices at four different orientations, resulting in a 20 dimension vector of texture values. Edges were produced from the Canny edge detector, and edge direction values were stored as a 72-bin histogram, 5 degrees to each bin and normalized by the number of edge points. They reduced the high dimensional feature vector using principal component analysis. The authors compared different similarity measures for comparing the query features to a database of images, including Bhattacharyya (with which they had the best results), Mahalanobis, and Euclidian (with which they had the worst results) distances.

The Multimedia Analysis and Retrieval System (MARS) [53] allows a user to specify, for example, the characteristics of the desired image or set of images to retrieve from a database. Using MARS, one can query the system to return images that have both the color characteristics similar to one image as well as the texture characteristics similar to another. They used hue-saturation color histograms with 8x4 bins. To measure similarity between two two-dimensional histograms, they used the intersection similarity, which they defined as $\text{similarity}_{color} = \sum_{i=1}^{N} \sum_{j=1}^{M} \min(H_1(i,j), H_2(i,j))$ with H_1 and H_2 being the two histograms, and N and M being the number of bins representing hue and saturation respectively. They used normalized histograms, where each bin holds the percentage of pixel values within that HS bin's range of values.

To represent texture, they used standard deviations of the coefficients of wavelet filters that they applied to the image, resulting in a 10 dimensional texture feature. To compare their texture features, they used the Euclidean distance between the vectors, which was also normalized and converted into a similarity value.

A key aspect to take away is their normalization of the feature similarity and distance values, which converts the comparison values to have a zero to one range. Much like in our project, the feature representations in [53] vary, as the authors have features stored as vectors, histograms, and a "modified Fourier descriptor" representation for shape boundaries.

2.4 Summary

There were various techniques employed to represent location and maintain data. Some extracted sets of keypoints to represent location, such as Kosecka and Yang's use of SIFT sets called "views" [29] or Schleicher et al.'s "SIFT fingerprints" [65], while others relied on global image features such as histograms [73]. Others made use of ontologies to organize data or value ranges [42, 22]. Others represented locations as a bag-of-words [12] or a

visual vocabulary [77]. Another approach was the mixture of various features [80]; in [25], they used various features to identify possible locations on the Earth from which query images could have been taken. A related approach to localization is image retrieval [53], as the methods used to compare features to retrieve similar images from a database is highly akin to comparing image features from a robot's location to the image features stored in the robot's map. It seems that different researchers used different statistical similarity or distance measures depending on their own experiments, and they used the ones that worked best for their individual projects. Some of the statistical measures used and tested in the works for image retrieval and localization included Jeffrey divergence, χ^2, intersection, Manhattan distance and Euclidean distance. We furthermore discussed approaches used for appearance-based SLAM [21] and topological mapping [70]. A related and important subject is semantic mapping. Certain features used for semantic mapping included objects [74] and text information [16, 59].

Chapter 3

Visual Fingerprints

Considering one of the goals of this research is to sample from the various visual layers of a location, SPLINTR uses multiple machine vision methods with which to extract features. This chapter describes our approach to multi-layered visual fingerprints, including the feature choices made and their implementation methodology within SPLINTR.

3.1 Introduction

We introduced earlier this idea of multiple types of visual features possible to specify a location – the signature of a location – and the benefits of enabling a mobile robot to sample over this set of visual clues. Such a technique produces a more intelligent representation of space and provides a robust set of features to use for topological localization and mapping. The layers of the signature we wish to sample from include the scene's texture, rudimentary image properties such as color and straight lines, and symbolic and human features, such as text and objects present in the environment.

The fingerprint metaphor has been used previously in literature, including Lamon et al. [34], Tapus et al. [71], Tapus and Siegwart [70], and the term "SIFT fingerprints" for

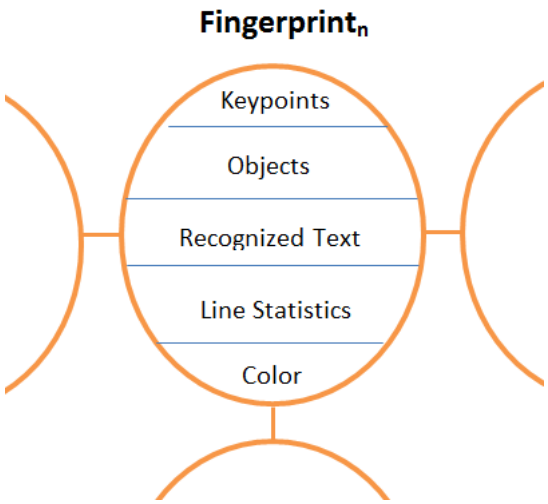

Figure 3.1: The layered ridges representing a fingerprint in SPLINTR.

sets of SIFT features used by Schleicher et al. [65]. As SPLINTR was initially a change and improvement to the string fingerprints of Tapus and Siegwart [70], we also adopted the fingerprint metaphor. The fingerprint term applies a label to the nature of our location descriptor, as it uses multiple features combined together to make up a location's topological node; calling the visual feature layers "ridges" serves to complete the metaphor between the pattern of friction ridges forming the human fingerprint and the various visual features encompassing our location fingerprint.

The following sections describe each of the chosen ridges for our approach to visual fingerprinting and how they are implemented for use in SPLINTR. We describe each ridge in detail, what features are extracted, and how each ridge is represented in SPLINTR.

3.2 Keypoint Ridge

The Keypoint Ridge of our visual feature fingerprint contains the extracted keypoint features from the image (or images) representing the robot's current scene and location. What we gain from keypoint extraction is essentially tens or hundreds of patches from the scene image; we can think of these features as landmark patches of the scene, though there will be "common" keypoints found in the environment. Such common keypoints may exist as features from repetitious architecture in the environment, like the corners of doors; a corner of a door frame is indeed an interesting feature due to its contrasting corner nature of a frame against a wall, however it may appear at multiple locations in the environment due to the usual presence of multiple doors in a hallway. Since we do not want to adorn one particular keypoint extraction method as the solution here (and a check of literature will produce multiple choices of keypoint extraction methods), the idea we wish to portray is to produce a result of interesting points resulting from texture of a location, since keypoint features are indeed a powerful feature to use for localization and mapping. Furthermore, keypoints will be stored with other features extracted from a location, thus forming a more intelligent representation and utilizing more information available from the environment.

3.2.1 Discussion of Keypoint Extraction Methods

This subsection discusses various methods of extracting and representing keypoints, interesting locations within an image, be it characteristic patches or corners. In the discussion, we will use the terms "detector" and "descriptor." The term "detector" refers to the method or algorithm of extracting or finding keypoint features within an image. The term "descriptor" refers to the method of representing the feature as data, so that it can be stored or compared with other keypoints; often this quantization takes the form of a vector of values, with the values, range and vector size depending on the particular descriptor method used.

These terms should become clearer as the section progresses. Particularly, some keypoint extraction methods may represent both a detector algorithm and a descriptor, such as the Scale Invariant Feature Transform (SIFT) [41]. In [41], Lowe discusses the algorithm for detecting SIFT features, along with the means of representing the features as data in its descriptor, and what those descriptor values represent. As is discussed later in the comparison measure of the keypoint ridge, we make use of the orientation of a keypoint feature, but not all keypoint detectors produce an orientation as part of the extraction method.

FAST - Features from Accelerated Segment Test

The FAST feature detector [60] is used for locating corner features within an image. To classify a pixel p as a corner, the FAST feature detector examines the pixel intensities on a circle surrounding p. If n consecutive pixels lying on the circle are all brighter than the candidate center pixel p plus a threshold t, or alternatively are all darker than p minus threshold t, then p is classified as a corner. The feature vector is composed of the pixel values comprising the circle, and the features can be classified by whether the feature is made up of pixels brighter than p ("positive") or darker than p ("negative"); thus in matching, positive features are compared with positives and negative features to negatives. Features are sorted by average intensity, and the feature with the closest average value is compared with the query feature using the Sum of Squared Differences (SSD). Certain heuristics are used to terminate SSD computation early for mismatches. In [61], this feature detector is sped up using a machine learning technique; particularly they employed a decision tree, thus training a more efficient feature detector which reduced the number of average checks to classify a pixel as a corner. The FAST feature detector is advantageous in regards to extraction speed, but it does not produce an orientation associated with the feature and thus is not rotation invariant. FAST also does not produce as rich a feature vector as say SIFT, with

the proposed FAST feature vector limited to the circumference of the pixel values lying on a Bresenham circle (of radius size 3) surrounding the feature.

SURF - Speeded Up Robust Features

Bay et al. [8] proposed both a feature detection algorithm as well as a representative descriptor method for extracted features. They constructed a feature detector based on approximating the Hessian matrix, a matrix that holds the values of the second order derivatives over a location in the image. To approximate the Hessian matrix values, they used a set of weighted box filters and approximated the determinant of the Hessian matrix, and to calculate the results of the weighted filters quickly, they used integral images. Briefly, an integral image, introduced by Viola and Jones [76], is a precomputed image where the pixel value at (x,y) represents the original pixel value at (x,y) plus all the pixel values to the left and above it. The integral image thus becomes a lookup table and used in the quick calculations of pixel values over regions of the image, including the computation of Haar wavelet filters and the weighted box filters. For detecting features at differing scales, rather than computing an image pyramid of downsized and blurred images (as SIFT does in [41]), they increase the size of their box filters, which does not increase computation time due to the use of integral images. Maximum determinants within a localized window in location and scale (a 3x3x3 neighborhood according to the authors) are determined, including their location in the image and approximate position based on scale.

Orientation of the feature is estimated based on results of applying Haar wavelets in the x and y directions and gauging their responses. Maximum response from sliding a wedge-shaped orientation window around the feature determined the angle. From a square grid around the feature point, the descriptor is composed of directional Haar wavelet responses (Gaussian weighted, with its center on the feature point location) summed over the regions in the grid. The summation of the responses in the x and y direction at sample points over

the regions, combined with the summation of the absolute values of the responses, form four values for each region. Since they used 4x4 regions, the full descriptor vector contains 64 values.

The authors used the sign of the Laplacian for quicker matching; this allowed for comparing light features on a dark background with features of similar contrasting qualities, and dark features on light background with similar characteristics. The authors indicated that due to the mechanism of extracting their descriptor and the underlying method the descriptor values represent, SURF is less prone to noise than SIFT. In comparing SURF [8] with SIFT [41] and the Harris-Laplace and Hessian-Laplace [46] keypoint extractors, they found SURF to be comparable or better than other keypoint feature extraction methods while also being quicker to extract.

CenSurE - Center Surround Extremas

In [4], Agrawal, Konolige and Blas described the CenSurE feature extractor. CenSureE (Center Surround Extrema) applies an approximation to the Laplacian using "center surround" filter responses; according to Lowe [41] and reiterated in [4], approximating the Laplacian is easier than the Hessian, and the Laplacian is better at feature extraction across scales [45]. To approximate the Laplacian, they explored using Difference of Boxes and Difference of Octagons, where the filter is represented by concentric shapes, that is, a shape (they tested use of a square and octagon) inside of a larger shape (hence, center surround). Within their difference filters, the smaller shape is weighted opposite of the larger shape, particularly they used the weights 1 and -1; these responses can be calculated quickly using an integral image (though modified to use the integral image for the octagon-shaped filter). They applied non-maximal suppression, filtered out weaker responses, and used the Harris measure [45] to filter out features located along edges, which according to the authors works better than using the Hessian, as SIFT used to eliminate edge features.

CenSurE also differs from SIFT in that the scale feature detection is not accomplished by incremental blurring and shrinking of the image, rather the feature detector is applied over every pixel at every scale. The authors apply this method for greater accuracy in feature detection at larger scales.

The CenSurE descriptor is a modified version of the SURF descriptor; the grid is altered to allow for regions to overlap and subregions to have Gaussian weighting (centered on the subregion), weighting the Haar wavelet response values before being summed into the four values representing a subregion, as in SURF. These modifications of SURF were inspired from SIFT; they were used to suppress descriptor values fluctuating as information may shift from being represented in one sampling area to a neighboring area. Thus, the overlapping regions and weighting are used to account for these boundary issues within the descriptor. Their implementation of the modified SURF descriptor ran faster in their experimental results as well (which they point out that they do not have a clear idea as to why, though they do mention certain C++ techniques such as the use of shorts and quick descriptor matching using "compiler vectorization"). Since the features are signed, dark features and light features can be distinguished, thus trimming matching time, similar to SURF.

The authors showed two experimental image sequence results, one in which CenSurE outperformed the other detectors tested, including SIFT and SURF, and one sequence in which it did not perform as well as its competitors. In another experiment concerning visual odometry, however, CenSurE (octagon version) performed better than competitors including SIFT, SURF, FAST and Harris corners [24].

Recent Contributions – BRIEF and ORB

Some of the recent contributions to the keypoint descriptor and detector research include BRIEF [15] and ORB [62]. First, BRIEF (Binary Robust Independent Elementary Features) is a descriptor, and it therefore requires a mechanism with which to detect keypoints. Once

a detector is applied to an image resulting in keypoint locations within that image, each keypoint must be quantified by a descriptor. Consider an NxN sized window around the keypoint location, which is first smoothed by a Gaussian. Within the window, BRIEF first requires two sets of sampling points; when considered together, the two sets form the pairs of pixel comparison locations within the patch used to form the descriptor's binary string. When a pair of pixels within the patch is compared, the test's corresponding bit receives a 1 if the first pixel has a value less than the second pixel, 0 otherwise. This process of choosing a pair of pixels and comparing them continues in order to fill the BRIEF descriptor of a chosen size. From their experiments, they received the best results by choosing pairs of test pixels by choosing sampling locations from a Gaussian distribution.

Since the descriptor is composed of bits representing pixel comparisons from a distribution within the patch, they employ the Hamming distance when comparing two keypoints. Thus, BRIEF offers descriptor size advantages, depending on the number of bit comparisons used, as well as speed in comparing keypoints. Their method, however, does not account for orientation differences, and thus this is not orientation invariant (though they point out that it can withstand a small amount of orientation difference).

ORB (Oriented FAST and Rotated BRIEF) [62], combines improvements made upon the FAST corner detector and the BRIEF feature descriptor. ORB builds on FAST by adding an orientation; the orientation is calculated based on the centroid of the corner. The orientation angle is formed from the line connecting the center of the corner feature window to the centroid. Rather than using BRIEF's method of selecting a set of test locations based on a Gaussian distribution, the authors learned a set of test locations for the purpose of decreased correlation and increased variance among the tests. The benefits of these two characteristics: uncorrelated tests allow contribution of each test to the end result, high variance allows for more discriminative binary features [62]. This learned set of test locations for BRIEF descriptors (and a lookup table to determine test locations based on orientation),

combined with orientation calculated from FAST corners contribute to the formation of the ORB feature detector. ORB is applied to a pyramid of scaled images, thus allowing for detection of features at various scales.

The authors compared their ORB feature detector to SIFT and SURF, for which it performed similarly or better, and it particularly did better on a set of images that gave CenSurE difficulty. Since CenSurE is an improvement upon SURF, it would have been nice to see how ORB faired against CenSurE. The authors also discussed a hashing technique for fast descriptor matching. ORB does improve upon its predecessors of FAST (by adding orientation) and BRIEF (better distribution of test locations and rotating the descriptor values based on orientation), while also keeping the quickness offered by the FAST detector over other more computationally intensive detectors of SIFT and SURF. However, one must decide if the features extracted using corner detection suffices for its intended end use.

Concluding Remarks

We use keypoints to sample the texture of an image, with the idea that a certain slice of the signature of the image (in this case texture) could be estimated from the application of a feature detector. Lowe's SIFT keypoint extractor [41] is a gold standard to detect interesting locations/patches within an image with both scale and rotation invariance. Theoretically, the Keypoint Ridge of SPLINTR could utilize a keypoint detection and descriptor method best suited for its task, and, as we have described above, certain methods offer strengths in the areas of descriptor size, extraction speed, comparison methods and accuracy; use of SIFT could be replaced in future work, though since we depend upon orientation, and SIFT comes equipped with both scale and orientation invariance, there is a reason new detectors compare themselves against it and a reason it has been used and utilized for over a decade.

Other keypoint detectors may be geared toward specific applications or end use devices. For instance, since ORB offers both a quick feature detector and a speedy descriptor for

matching, in addition to testing on an image dataset and for object recognition, the authors demonstrated its use in tracking features on a mobile device. CenSurE improved upon the SURF detector and descriptor, though they showed in experimentation that it is not perfect. They demonstrated results on a dataset (involving testing rotation and zoom) in which it did not perform better than competitors SIFT or SURF. However, the authors did show CenSurE's better performance over competitors in the application of visual odometry. FAST, and by extension its oriented brother ORB, use corner detection, and we would like to be able to detect more texture features than corners alone. Being able to dynamically and intelligently choose which detector/descriptor to use would be useful future work. Somewhat related to this future work is ReIn (Recognition Infrastructure) [48], an architecture with a defined interface which allows the use of multiple object recognition schemes within the same system, intended for use in robotics. ReIn is implemented for use with ROS, the Robot Operating System. When object detectors are wrapped to use the ReIn interface, it allows the benefit of data sharing when appropriate, and the use of multiple algorithms to be switched between, run in parallel, or pipelined. In [48], Muja et al. used ReIn to combine two of their object detectors, a gradient based object detector called BiGGPy and a 3D point cloud based object detector called VFH. One detector was used to over-detect for objects in a scene, and the second was used to verify the detections and reduce the over-detected false positives. They showed the combination of the two working in sequence was better than using the BiGGPy detector alone, which was allowed by using the ReIn framework.

3.2.2 Our Use of SIFT

To reiterate our plan of using keypoint extraction, we want to sample from the texture of the robot's environment, particularly its current scene. To extract keypoints as well as form a descriptor of those keypoints, we use the Scale Invariant Feature Transform, or SIFT [41]. The benefits of using the SIFT keypoint extractor and descriptor include its

invariance to scale changes and orientation changes. Rotation invariance can be used to exclude some obvious "common" features from matching when they happen to be extracted at differing locations. Furthermore, according to Lowe [41], SIFT is also partially invariant to light changes and three-dimensional viewpoint/affine changes, which is important for a mobile robot which may detect keypoints at varying viewpoint locations while exploring the environment.

In an introduction to the SIFT feature extraction method, SIFT features are extracted by filtering the image with the Difference of Gaussian (DoG) function; this function serves as an approximation to the Laplacian of Gaussian normalized for scale invariance [39]. An image is convolved with the Gaussian operator in a manner that produces images at increasing "scales" of smoothed images (that is, the σ differs between scales by a constant factor), and the DoG images are produced from the difference of these blurred images from neighboring scales. Images are downsampled and blurred more to produce multiple "octaves" of blurred images, and the process is repeated. Preliminary SIFT keypoints are the pixels that are extrema of their neighbors within the same scale and the scale above and below. Further methods are described in [41] to eliminate poor keypoints, such as those with low contrast, or keypoints unfortunately located near edges. More specifically, using values calculated from an approximation to the Hessian matrix, they can estimate the principal curvatures and eliminate keypoints along edges; edge keypoints are removed because of their instability, for instance to noise. This process, as pointed out earlier, is different from the use of the Harris measure to eliminate edge keypoints in CenSurE [4]. An orientation histogram is formed from gradient information around the keypoint (with input to the bins weighted by the gradient magnitude and a Gaussian weighted window). Orientation of the keypoint is then assigned according to the orientation histogram bin with the maximum value; if other bin values are within 80% of the maximum value, then the keypoint is duplicated and assigned that additional orientation, repeated for all bins exceeding this 80% threshold. That is, key-

points are duplicated and assigned different orientations if there are several strong gradient orientations concerning a particular keypoint. The descriptor of the keypoint is formed from the orientation histogram values around the keypoint, relative to the keypoint's assigned orientation, thereby making it orientation invariant. Values inputted to the histogram are weighted by gradient magnitude and by a Gaussian circular window scale-weighted particular to that keypoint's scale, and furthermore calculated and distributed to adjacent bins to aid in boundary issues (handling of boundary issues was brought up earlier in the discussion of CenSurE in Section 3.2.1). Further methods are applied to the descriptor values, including normalization to unit length, followed by a thresholding (any values greater than threshold t is replaced by t, which [41] uses a t of 0.2) and normalizing again to unit length.

3.2.3 Keypoint Ridge Representation

In the implementation of our system, we use an open source library of SIFT available for free from Rob Hess [26]. It is a C implementation of the algorithm described by Lowe, and it provides the extraction of the SIFT descriptor keypoints and storage within a k-d tree data structure. Furthermore, Hess's SIFT library implements the Best-Bin-First keypoint matching algorithm by Beis and Lowe [9], which returns an input keypoint's match from a k-d tree of SIFT descriptors; this algorithm is approximate, though it does provide a quicker alternative to brute force.

Thus since our keypoint ridge uses the SIFT keypoint extractor, we implement the keypoint ridge using a subridge, a SIFT ridge. The SIFT ridge maintains the SIFT keypoint features, which are extracted from the scene image of the robot's location and maintained within a k-d tree to be used to find keypoint matches. Similarity between keypoint ridges when comparing two fingerprints is described later in Section 4.1.1.

3.3 Object Ridge

Within the scene image, the robot may find objects via object detection or object recognition. Within the context of this research, we consider the difference between the two as the difference between a generic object (detecting a face) and a specific object (recognizing that face is Abraham Lincoln's face). In experimentation, SPLINTR is equipped with a library of objects it should detect and a library of objects it should recognize. The mechanisms for how each is implemented in our system are described later in this chapter.

The importance of the object ridge is to ascribe a higher order label to specific surroundings of the robot, since we as humans are able to do this very thing. Landmark objects help us remember locations and places (the door next to the fire extinguisher) and thus are strong candidates to be included in a localization and mapping project.

3.3.1 Object Recognition/Detection

In order to recognize and detect objects in the environment, SPLINTR needs to take as input prior knowledge of objects for which it is to find in the environment while exploring. For recognition, upon startup we can examine our database of objects and extract keypoints to represent each object within our system (described in more detail in the next section). For object detection, since we use a set of boosted classifiers to detect objects within the scene image, these identifiers must be trained in advance.

It is not hard to imagine scenarios in which prior knowledge of objects to recognize and detect could be provided to the robot. Such cases may include an environment with a known intended use, such as a specific museum, of which a list of art pieces or artifacts known to exist within the museum could be provided to the system in advance. Other times, acquired intelligence involving a military exercise could provide objects particular to an environment the robot is to explore. Still other times an object list could be acquired

from domain knowledge of the environment, such as exploring the Institute of Artificial Intelligence on UGA's campus; in this case, logos of research projects could be acquired from the University's webpage, or AI related images in general from the Internet (for example, popular or respected research journals such as AAAI) to be found in the environment. Still even further information of the environment could be acquired from social engineering or manipulation, but such tactics are outside the field of this research.

Object Recognition

First, we shall describe the use of SIFT keypoints for object recognition. When the system starts, we use an array of file names to signify the object images for which the system will be searching and attempting to recognize in the environment. We refer to the set of images representing objects as our "object database." These object images are inputted to the system using OpenCV, and a set of SIFT features is extracted from each object image to represent that object in the system and to facilitate recognition while the robot explores. After a set of SIFT keypoints is extracted from each object image to recognize, it is represented as a k-d tree and stored in an array to be accessed later for recognition. Therefore, given a list of image names to represent objects to recognize, the object database is represented in an array of SIFT keypoint k-d trees, one k-d tree per object to recognize. A separate string array houses the names of the objects. Now that we have a representation of our objects to recognize, the process to decide whether or not an object is present in the scene image is described next (for each object).

For each SIFT feature in the input scene image, we attempt to find its match within the set of object keypoints. If there are enough good matches, we can consider that object as being present in the scene image presented to the robot.

How do we determine good matches? We adopt some of the Hough voting procedures used by Lowe [41]. Matches between the object database keypoints and the SIFT keypoints

from the camera image are used to cast votes on the orientation and location of the object within the scene image. We can then use these votes to form a consensus on orientation and location of the object in the scene image, and therefore eliminate bad matches not agreeing to the consensus.

Why the use of this consensus voting? If an object is indeed within the scene image, the keypoints can be considered "grouped" together, so that when the object moves and rotates, the SIFT keypoints likewise move and rotate with the object respectively; if the object is rotated in the image, we can expect the difference in orientation of the matched keypoints to differ by similar orientation change, since SIFT keypoints have an associated orientation. We thus create a histogram of orientation differences between scene keypoints and their matches within the object model keypoints. The bin of the orientation differences containing the most votes represents the orientation of the model object in our scene. Adapting the voting mechanism from [41] for our own purposes, the orientation difference histogram is made up of 24 bins, and votes are cast into the two nearest bins. The keypoint match is used to cast a vote for the object's position in the image; the image area is divided into 80 different cells, eight cells tall (six on the image, and one above and below the image boundary) and ten cells wide (again, eight on the image with one bin on either side to catch votes off of the image range). If the orientation difference and region projection do not agree to the consensus, the match is discarded.

If there are at least four good matches, we use OpenCV and the matches to form a homography (RANSAC is used in calculating the homography). We then check the homography in order to eliminate those that are poor in quality. One method of identifying poor homographies is to find those homographies with determinants close to zero [32, 75]. If the absolute value of the determinant of the homography is too small (below a threshold), the object is declared not found. Through experimentation, we discovered poor homographies indicated by a determinant value that is too large, so we also declare the object not found if

the determinant is above a certain threshold. Another sign of a poor homography is a large condition number [54], indicative of an ill-conditioned matrix [17]. Therefore, we check the condition number of the singular value decomposition (SVD) of the homography, and if the condition number is above a certain threshold, the object is declared not found. This may invalidate actual objects from being found in the environment, but provides an extra layer of protection against including poor object recognition results.

Object Detection

We use OpenCV to detect objects within the scene image. More specifically, we use OpenCV's functionality to train and use a set of boosted classifiers, introduced by Viola and Jones [76] and extended by Lienhart and Maydt [38]. From the description of the work of Viola and Jones, a cascade of boosted classifiers uses a set of classifiers applied in a "cascading" sequential manner to classify areas or "sub-windows" of an image as positive or negative instances of an object. In the training process, simple Haar-like features are selected for use based on their classifying rate on training data. Sub-windows of the image that pass the initial classifier of this cascade highlights areas of the image possibly containing that object; that is, all sub-windows that pass the initial classifier move on through the chain of classifiers to be further processed. This narrows the focus of attention on certain areas of the image more likely to contain the object. These initial candidate locations are further verified by applying the next classifier in the cascade (and as Viola and Jones describe it, each successive classifier is slightly more complex, thus conserving processing time for the more promising areas). If a sub-window fails a classifier in the cascade, it is removed from consideration. If it passes all classifiers in the cascade, then that sub-window is considered a positive instance of the object in the image. Such a technique is repeated for each object to detect, and at different sizes to detect objects at different scales.

To train the cascade of boosted classifiers to be used for object detection, we first use OpenCV to create training samples from an image of an object to detect; if we used more than one image of an object for training, we use MergeVec [66] to combine the multiple sample files into a single file for use by OpenCV. We use OpenCV's training utility to create cascade classifiers for object detection. We also use the default boosting algorithm of Gentle Adaboost; Lienhart et al. [37] showed that Gentle Adaboost outperformed other boosting algorithms, specifically Discrete and Real Adaboost, in their experimentation.

A set of cascade classifiers is trained offline for each object to be detected in the environment. To apply object detection to the scene image, we call OpenCV's multiscale object detection function. The function uses the trained cascade of boosted classifiers for a particular object and returns locations of that object within the image. This function is called for each object classifier we have trained and that we wish to detect in the environment. This process results in a set of locations for each object detected in the image, as well as the count of each object detected. This information can then be used to sort the objects in their detected left-to-right order, described next.

3.3.2 Object Ridge Representation

Once we have applied object detection, we combine the results (the names of the objects) with our object recognition results into a single array of *char**s. These names are subsequently sorted in the left-to-right x-axis ordering that they were recognized/detected within the scene image.

To represent the set of objects recognized and detected within the object ridge, we actually use a single string representation, an object list "acronym." The first letter of each object's name is stored in the string in the left-to-right sorted order that it was detected/recognized. This might best be illustrated with an example. Suppose in the scene image the robot detects a "water fountain" and a "fire alarm" in that order; then the object

ridge representation would be reduced to the string "wf." This style of representation is similar to the string fingerprints of [70] and the string map representation of environment architectural objects in our previous work [79]. The benefit of this representation style is that, similar to Lamon et al. [34] and Tapus and Siegwart [70], we can now use string matching techniques to compare the object ridges between two fingerprints, discussed later in section 4.1.2.

3.4 Recognized Text Ridge

We wish to help represent a place by another mechanism that humans use to assess location, that is, by the text we can read. The Recognized Text Ridge houses scene text extracted from a location by SPLINTR. Our pipeline process of text recognition includes the following: text detection → multi-resolution Focus Of Attention → thresholding → text recognition → dictionary and rules checking validation. We shall go through each of these segments below.

3.4.1 Text Detection

We use OpenCV in our text recognition pipeline. Given a scene image, in experimentation, we found that some camera image qualities or environments may call for preprocessing before executing text detection, since our chosen text detection algorithm is edge-based, and since it does not call for any steps to ensure uniformity in edge detection with varying environment lighting conditions. If our experiments make use of preprocessing, we first applied OpenCV's histogram equalization, which adjusts the image's contrast, and then applied Gaussian smoothing. This process of adjusting contrast makes for a more uniform distribution of pixels, but due the increased contrast, may cause too strong of a response to edge detection. Therefore, after histogram equalization of the image, we dampen its effects

with Gaussian smoothing. This contrast sharpening followed by smoothing is intended to provide a more uniform input to, and a better response from, the text detection mechanism.

Our goal is to capture scene text, that is, text appearing within the environment. Samarabandu and Liu [64] developed an algorithm to detect scene text. Their algorithm is edge-based, with emphasis and importance placed upon 90 degree edges. The authors claim that regions of an image containing text have stronger edge strength (resulting values of directional edge filter), density (strength of edges within a neighborhood window) and greater variety of orientation. Our implementation of the text detection algorithm follows closely to the algorithm laid out by Samarabandu and Liu [64].[1]

Once the text detection algorithm is applied to an image, we are left with regions of interest, areas of the image possibly containing text pixels. We have found certain textures to be tricked by the text detection algorithm, particularly the bump texture of cinder block walls at close range. Therefore, if a text detected blob's area is greater than one-third of the 640x480 image (resolution of input image used for detection – may be concatenated with other images from different rotation angles of the robot to form multi-image detection – explained more in the Localization experimentation section), it is declared wall texture, a false text detection area, and the area is filtered out of further text recognition processing steps.

3.4.2 Multi-resolution Focus of Attention

Next, we describe our method of multi-resolution images for acquiring better text pixel information. To understand what we mean by "multi-resolution," as the robot explores the

[1]We deviate in the details of a few steps. We do not perform an edge thinning that is called for in one step of the algorithm, which should reduce the edge responses to a width of one pixel; we assume the authors call for applying a Canny edge detector, since Canny reduces lines to a single pixel width, though we decided to keep the thicker edge candidates and not perform a thinning operation. Furthermore, in the "feature clustering" step, after the 7x7 dilation of the operator, we perform a 3x3 closing operation to merge nearby text detected areas.

Figure 3.2: Text detection algorithm (Samarabandu and Liu, 2007, [64]) applied to scene image (top) which returns possible areas containing text, highlighted (bottom).

environment, images are captured at a high resolution (1600x1200), but downsampled before being passed to ridge feature extraction. Thus, we can use the lower resolution images to perform text detection in a faster manner, since fewer pixels are being processed and filtered. These higher resolution images, however, are passed to the Recognized Text Ridge to be used to gain better pixels for use in text recognition.

The text detection algorithm [64] classifies pixels within the image as being a text pixel or background pixel. A connected component labeling algorithm using an OpenCV blob library, cvBlob [36], is applied which groups the pixels into connected components. These connected components of text detected pixels are located in the high resolution image by simple scaling mathematics to find their x-y position and region, forming a pseudo-zoom function we can use.

We do not pass the high resolution sub-images directly to the recognition engine. We go through a process of masking, copying, and thresholding first. The text detection highlighted pixels are resized from the low resolution to a high resolution size; the areas of the high resolution image corresponding to character pixel classifications in the high resolution mask image (character pixels are indicated as white in the mask) are copied over into a new sub-image, leaving areas already classified as background out of the new image. Next, a threshold value is calculated using just the text detected pixels (those pixels already classified as background are not used to calculate the threshold value). This produces a more accurate threshold value to threshold text pixels from background pixels. The process is illustrated with an example in Figure 3.3. The idea here is that the masked pixels, the pixels copied over into a new sub-image, may not be free of background pixels, thus we should threshold to separate text pixels from background. Once we have a binary image, the connected components are analyzed, and small blobs (less than 15 pixels in area) are considered noise and removed. We also eliminate blobs that are considered "overhang," such as part of a wall captured in the sub-image containing text from a door number. If the wall is white,

and the text is also white, and the plate is a contrasting color such as black, then pixels of the wall may be classified as text pixels due to their similar color. We count the number of blobs whose centroid y is above the average maximum y of all blobs. If this count is one or two, those one or two blobs are considered noise and eliminated before the image is sent for processing by the recognition engine. We can eliminate these blobs because we have rules in place to eliminate small words as noise in the validation check, which is applied next in the pipeline. Likewise, one or two blobs whose centroid y is below the average minimum y of all blobs are also eliminated as "underhang."

A final step to increasing the likelihood of recognizing scene text in the environment is to orient the sub-image based on the orientation of the text detected blob. Such an approach of orientation correction based on connected component orientation angles has been used in the work of [7]. We use cvBlob to calculate the angle of orientation of the text detection blob. If this angle is within a reasonable range (between -70 and 70 degrees), the sub-image is rotated to correct for this angle. Our approach is a two dimensional correction to a three dimensional problem, and future work could use the range data from the Kinect to correct for 3D pose of scene text in the environment. The sub-image corrected for orientation as well as the non-corrected sub-image are both sent to Tesseract for recognition. Both returned sentences are validated. If the oriented image returns more valid text, then that text is kept. Otherwise, the valid text from the image not corrected for orientation is kept.

3.4.3 Text Recognition and Validation

The sub-image pixels (following thresholding and noise removal explained previously) are then passed to the recognition engine. Specifically, we use the Tesseract OCR Engine version 3.01 [56], with the option enabling use of the Tesseract/Cube combination. According to Tesseract's Release Notes [3], use of Cube in combination with Tesseract should produce

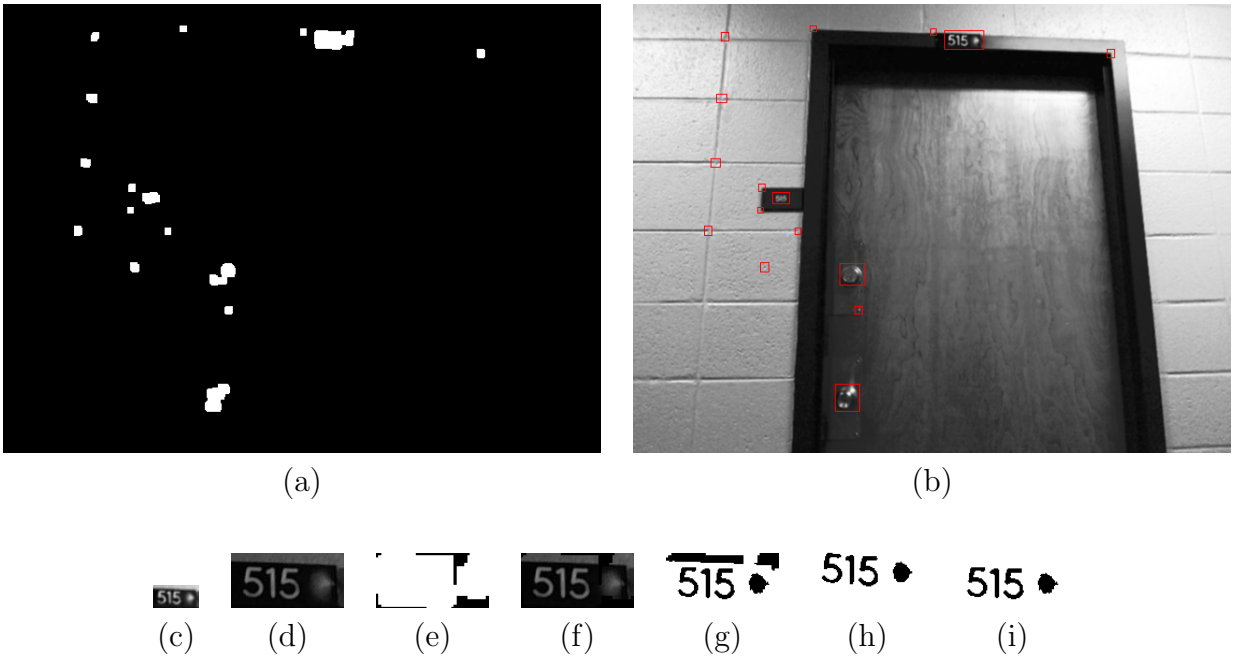

Figure 3.3: Example of text recognition using text detection pixels as a mask for better OCR images. (a) Text detection pixels from scene image highlighted. (b) Text detection results boxed in scene image. (c) Low resolution sub-image (focusing on "515" text box above the door). (d) High resolution sub-image (focusing on the same "515" text box from the high resolution image). (e) High resolution sub-image of same area in text detection results (resized from low resolution text detection results) – used as a mask to copy text detection pixels into new sub-image. (f) Result of using mask to copy only text detected pixels to high resolution sub-image (black pixels from masked area ignored for calculating threshold). (g) Binarized result. (h) Overhang/underhang removed, and sub-image corrected for orientation. (i) Overhang/underhang removed, but no orientation correction. Both (h) and (i) are sent to OCR and validated.

more accurate text recognition results at the cost of more processing time. In the engine settings, we also use a whitelist of characters to extract, particularly:

$whitelist = 0123456789abcdefghijklmnopqrstuvwxyz$
$ABCDEFGHIJKLMNOPQRSTUVWXYZ :; ., \backslash/"!\#$

After passing the pixel data for the Tesseract engine to process, the engine returns the text that it recognized from the sub-image.

If one is working in a perfect detection/recognition environment, with perfect detection and recognition capabilities, one would not need to verify the recognition results. Our detection and recognition methods are not perfect. We use an implemented version of [64], though, for detection. We also use an off-the-shelf text recognition engine which may work best with contrasted characters scanned in from a document or book. We, however, have adapted it for use in scene text recognition. Our detection process may return false positives, and our recognition engine may still return "garbage" recognized on a false text detection area. We need a mechanism for thinning out poor recognition results.

A full response from a region of the image sent through text recognition we describe as a "sentence." If there are multiple text detected areas within the image, we then have multiple sentences returned from that image. First, a sentence is passed through an initial garbage detector. We have noticed a set of characters and symbols that occur in blocks of garbage text returned from false positive text detected areas of images. We accumulate their occurrences into a sum we call an "indicator strength," and if that indicator strength is greater than one-third the length of the sentence, then we declare the sentence as garbage. It is then discarded, and the next sentence is checked.

Once a sentence passes the initial garbage detector, we further verify its contents by checking the words for validity. Tesseract allows one to check words against a dictionary, however if using a large dictionary, we found it must be in the format that Tesseract internally uses rather than a long list of words in a text file. Thus we derived a dictionary from the CMU

Pronouncing Dictionary v0.7a [2]. To form the dictionary, this CMU Pronouncing Dictionary (all words in uppercase) was edited down to only the word list, with word duplicates removed. We then duplicated the list to include sentence case and lower case versions of the words, and finally a set of numbers from 0 to 9999 was added to the list before being converted to the format used by Tesseract. Furthermore, we added "UGA" and "uga" to a user-defined word list which Tesseract checks.

After the garbage check, the sentence is validated using this dictionary and specific rules. First, the sentence is delimited, and each word is checked for validity in the dictionary. If a word is returned as invalid, it is still kept as valid if it meets certain criteria. If an invalid word is in all caps, it is kept as valid. For a word to be in specialized case like this is useful information. This may indicate that the text recognition may have been slightly off in recognizing a word (perhaps off by a character), or the word did not appear in the dictionary, such as a proper name or acronym. Furthermore, if an invalid word is composed of four characters, the first three of which are numbers and the last a letter, then it is kept as valid. This is to keep special room numbers with a trailing capital letter. Other such environment-specific rules may need to be added to the system for a given environment.

After checking individual words for validity within a sentence, we then check to make sure sentences as a whole pass certain criteria. Sometimes, areas of an image containing a hallway wall get falsely detected as containing text, and short text sentences may be returned by the recognition engine analyzing the sub-image. Therefore, we eliminate sentences that are less than or equal to three characters in length, unless all three are numbers. If all three characters are numbers, then the sentence is kept; this is to ensure we save door numbers, which may exist separately and by themselves in the image, but may return the correct three digits of a room number. Within each sentence, validated words (either by rules or dictionary check) must also meet one of the following criteria:

1. be composed of 3 or more numbers

2. be composed of 3 or more letters in all caps
3. be longer than 3 characters

The general idea behind this last set of rules is to keep longer words and eliminate shorter ones, with the idea being that it would be more difficult for the text recognition engine to recognize longer valid words from false positive text detection sub-images. These rules attempt to reduce noisy words while keeping valid data within sentences, even if the data may itself be imperfect. Smaller word results may end up being noise, but if they follow a certain pattern, then we keep it, assuming this pattern (indicated by the rules above) is an indication of validity and importance. If valid words do not meet one of those rules, they are invalidated.

3.4.4 Representation in the Fingerprint

After each sentence's words are checked, only validated words (those that pass the dictionary and/or the rules) remain. The validated sentence is then saved into an array. The Recognized Text Ridge is represented as this array of validated sentences, one validated sentence per array slot.

3.5 Color Histogram Ridge

If we wish to capture information on the distribution of color in a location, we can utilize a color histogram. The theory behind the inclusion of a color histogram is that color distribution is a generic image feature, and can be useful to distinguish general areas of an environment. That is, use of a color histogram may distinguish one hallway from another or locations that may have similar texture but dissimilar color. Information from a color histogram may be useful when keypoint features show a good match between the current

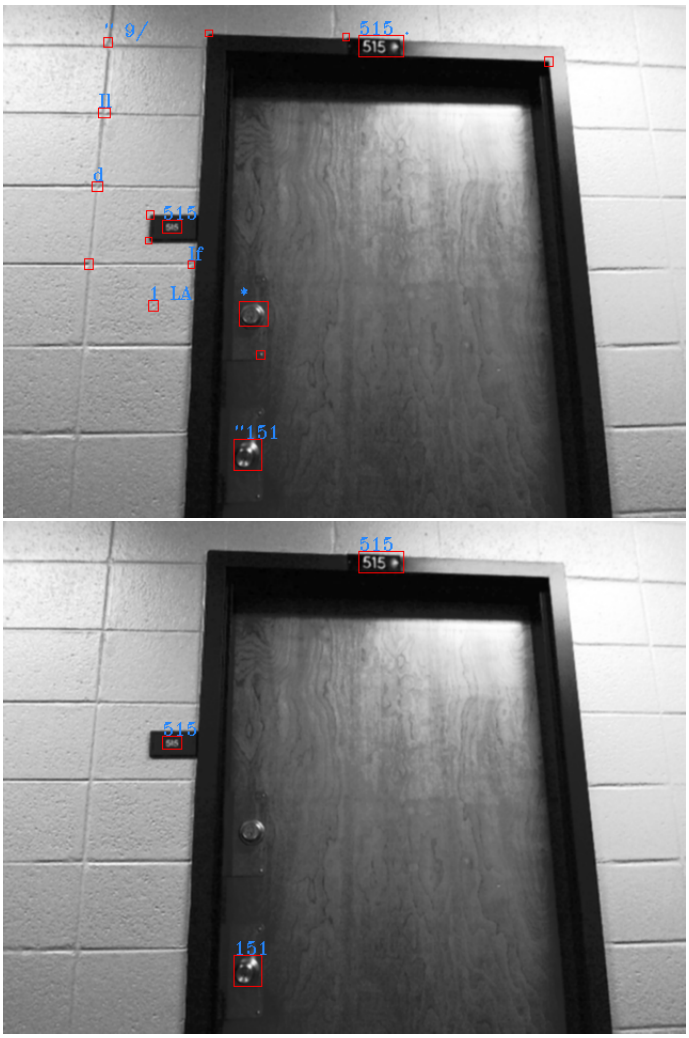

Figure 3.4: Tesseract OCR engine v 3.01 results (top). Upon validation check, we get the results on the bottom. We see a false positive text result passed through the rules.

location and a location in the map, however the color distribution shows that these locations are clearly not the same location.

Furthermore, if no other information can be ascertained, that is, no objects, lines, keypoints or text can be extracted, this is the fundamental unit of the fingerprint that can be used. As long as an image is provided, a color histogram can be formed on the distribution of color.

The color histogram in SPLINTR is a two-dimensional histogram of the hue and saturation channels of the HSV color space. We use OpenCV to convert the input image from the RGB color space to HSV color space. This particular mapping can be done with a mathematical formula on the RGB pixel data. The HSV color model separates value into its own channel, separate from hue and saturation. As noted in the MARS image retrieval system by Ortega et al. [53], the value channel can be affected by light variations, so they use a 2D color histogram from the hue and saturation channels. Since the value channel may be affected by the changes of light in the environment, we too build the histogram from the hue and saturation channels.

The circular representation of hue in the HSV color model has values in degrees, from 0 to 360, though to fit the values in an eight bit format, they are divided by two by OpenCV and returned in the range 0 to 180. For the 2D hue-saturation histogram used by Ortega et al. in MARS [53], the authors use eight bins for hue and four bins for saturation. In our 2D color histogram, the hue dimension is divided into 30 bins and the saturation dimension is divided into 32 bins; this higher dimensionality follows the number of bins in the histogram online tutorial of N. Kuntz [33], which is based on the Learning OpenCV book by Bradski and Kaehler [13].

The representation of the Color Histogram Ridge in SPLINTR is this two-dimensional hue-saturation histogram.

3.6 Straight Line Ridge

The theory behind the use of straight lines as a feature in SPLINTR is that it is another generic environmental feature. Straight lines can be formed from the architecture in the environment, from doorways, from the lines in the wall, and thus collecting statistics on these extracted lines can be useful to describe location for the robot as it explores.

First, to extract straight lines, we use OpenCV to transform the input image to grayscale. Then we preprocess the image with histogram equalization followed by a smoothing operator. We then apply the Canny operator followed by a version of a probabilistic Hough Transform. We then represent straight line information in three histograms. One histogram represents orientation information and one captures length information. How to utilize extracted straight lines was an early question, but these two representations (length and orientation) were based on the work of Hayes and Efros [25]. We utilize a third statistical measure of the lines, that is, information on the color of the lines. For each line, we calculate its orientation angle and length. Length values are capped at 255 pixels; if a value is greater than this, it is replaced with 255. The lengths of all lines are then used to form a histogram of lengths. We use a length histogram of 115 bins. The orientation histogram has 30 bins for the value range of 0 to a maximum of 180 degrees. To form the line color histogram, we sample each line in the hue and saturation channels of the image, then accumulate this data into a cumulative hue-saturation 2D histogram, thereby collecting line color information from each line, again with 30 hue bins and 32 saturation bins.

In experimentation, we found that line length histograms were not too discriminative from site to site, so we eliminated them from the experimentation results found herein. Each line histogram is normalized and stored within the Straight Line Ridge, along with a count of the line pixels. The line pixel count is used to detect if any lines were extracted from

the scene image. We discuss the use of such information in the comparison section in the Localization chapter.

3.7 Combining the Ridges to Form a Location Fingerprint

We have explained the representation of each ridge of our fingerprint model. Each ridge (and therefore its features extracted and represented in that ridge) is stored as an individual ridge object within a single Fingerprint object, much like our own human fingerprint ridges are separate yet when thought of as a whole they form the pattern of our fingerprint.

Within each associated ridge, we maintain certain statistical counts:

1. number of keypoints extracted
2. number of lines
3. number of valid text sentences
4. number of objects recognized/detected

When comparing fingerprints, using these counts, if one ridge or the other has zero features of that type, we can automatically assign a 0 to that ridge comparison result. Alternatively, if both ridges contain zero features, we can eliminate that ridge from the overall fingerprint similarity measure (and distribute its weight to the other ridges). Further explanations on comparing the fingerprint ridges are explained in the Localization chapter, Chapter 4.

Chapter 4

Robotic Localization

Localization is the process of finding oneself on a map. If you are handed a map of a city and asked to find where you are within the given map, you may start by looking at your surroundings trying to pick out, say street signs or buildings. This process of being given a map and the task of finding where you are within that map is a general description of localization. In robotic localization, the robot is given a map of its environment and must be able to pick out its location within the map.

4.1 Comparing SPLINTR Fingerprints

So far, we have discussed our implementation of fingerprint ridges. We have discussed the theory that these ridges sample from various layers of the signature given by a particular place. Since we now have a way of representing locations, we have a way to 1) represent our map and 2) assess the features of a particular location visited by a robot to find out where it is in its map.

To apply our visual feature fingerprints to robotic localization, we need a way of assessing the similarity between two fingerprints, the fingerprint of our current scene and a fingerprint

from our map. If we can assess similarity between two fingerprints, we can repeat this process comparing the input fingerprint with each fingerprint in our map. SPLINTR therefore has a process of finding the similarity between 1) each ridge of the two fingerprints and combining each ridge similarity into 2) a similarity between the fingerprints as a whole.

4.1.1 Keypoint Ridge Similarity Measure

Here we describe our formula to compare the keypoint ridges between a fingerprint of the input, *fp_Input*, and the fingerprint of a mapped location, *fp_Map*. Since we use SIFT keypoints, for each SIFT feature in the keypoint ridge of *fp_Input*, we extract its nearest neighbor from *fp_Map* using the SIFT library [26]; now according to the implementation in [26], for the nearest neighbor to be considered a "match," its Euclidean distance from the input SIFT feature must be closer than a certain threshold requirement when compared to the Euclidean distance between the input SIFT feature and its second closest neighbor.

Since SIFT keypoints contain information regarding their orientation, we can use this information in our system to rule out false matches. Therefore, within SPLINTR, to be a "good" match, the input keypoint and its preliminary match returned from the SIFT library must agree within a certain error in the orientation of their descriptors. A gross difference between orientations is an indicator that this is not the same SIFT keypoint extracted from an earlier visit; certain keypoints are indeed more distinctive than others, as described by Lowe [41]. Lowe uses an orientation checker for object recognition [41], and we make use of orientation difference here in our keypoint ridge similarity measure.

Once all input SIFT features are compared between the keypoint ridges of *fp_Input* and *fp_Map*, and after all matches are found and classified as good or bad matches as just described, if the total number of matches is greater than zero, then we calculate the keypoint

similarity measure as a ratio of good matches to total matches between the two fingerprints:

$$keypointsSimMsr = \frac{numGoodMatches}{numTotalMatches} \tag{4.1}$$

Now one may note that this similarity measure is advantageous to locations that have few keypoint features and match well to fingerprints in the map with similar keypoint features that are also few in number. An example might be a location that has only two matches to a fingerprint in the map, and both matches are "good" matches based on our orientation error checker. Therefore, when this similarity result is combined with the other ridge similarity measures, it is dynamically weighted based on the average number of keypoints the robot has seen up until the present location. If the current location is richer with keypoints and extracts more keypoint features than the average, then this ridge's similarity measure will be weighted more; if it extracts fewer keypoint features from the location, the similarity measure will be weighted less, calculated as the ratio of the keypoint extraction count from the input scene over the average count seen up until now. The dynamic weight pulls equally from the weights of the other ridges when weighted more, and distributes equally over the other weights when the keypoint ridge similarity measure is downplayed. This dynamic weight is capped in experimentation to avoid entirely eliminating the other ridge similarity measures when calculating the full similarity.

Therefore, we use an interesting approach to measuring similarity of location based on keypoints that does not rely on the ratio of matches to total number of keypoints extracted, makes use of orientation to eliminate some matches of commonly occurring keypoints, and does not solely use match count alone. This ratio of good matches to total matches allows us to use the calculation as a comparison measure which returns a value within the range 0 to 1. This is done so that we can weight the individual ridges later and return a total similarity measure with a value 0 to 1.

4.1.2 Comparison Technique for the Object Ridge

Recall that our object ridge contains objects recognized and detected in a string "acronym" of recognized/detected objects. This compact representation allows us to use a string matching algorithm when comparing the object ridges of two fingerprints. We shall first display our measure for object ridge comparison and then explain its constituent parts. Our object ridge similarity measure is:

$$objSimMsr = \frac{numMatches + (0.5 * maxSeqLength)}{1.5 * maxLength} \tag{4.2}$$

This function allows us to ascribe a value to objects matching between two strings as well as matched objects in the right order. This dual value is useful in case the robot missed detecting/recognizing an object, an object was occluded, or the environment was modified between visits.

To calculate the $objSimMsr$, first we count the number of matching objects between the two strings, $numMatches$. We compare the minimum length object list to the maximum length string, and we replace the matching object character with a space in the maximum length string so that we do not count the same object twice; that is, we do not match multiple object characters in string 1 with the same object in string 2 (we of course work with copies of the object string so as not to edit the actual contents of the Object Ridges). The $maxSeqLength$ is the longest shared sequence between the two strings, that is, the number of objects in the right order (though they may be separated by other objects not in the right order). This is implemented with a dynamic programming solution to the longest increasing subsequence algorithm [5, 67]. Finally the $maxLength$ is the longest string between the two Object Ridges and represents the most objects the two strings can have in common.

The reason we include an order is to capture a further sense of what the robot got right and what the robot missed (say, due to an object not being detected or recognized out of a

set of objects). In future work, the order should account for the heading of the robot, and the images could instead be from an omnidirectional camera with images aligned to account for heading.

4.1.3 Similarity Measure for Recognized Text Ridge

First, each Recognized Text Ridge is composed of an array of validated "sentences," where each sentence is a specific sub-image result of the text recognition pipeline. Therefore we needed to assess similarity between two arrays of string sentences. Our first step is to compare the ridge with the greater number of characters to the ridge with the fewer characters; this provides a better assessment of similarity. For each sentence in the larger ridge, we find its best match in the smaller ridge; the best match is determined by the Levenshtein Distance [35], which is an edit distance, or how many changes it would take to transform one text sentence into the other. The best match is therefore the string with the minimum Levenshtein Distance. We sum up the distance results for each string's best match ($minDistanceSum$). We also sum the maximum distances between each string and its match – the maximum distance between a string and its match is the length of the longest string between the two. This value forms the $maxDistanceSum$. The Recognized Text Ridge similarity measure is then calculated as:

$$textSimMsr = 1.0 - \frac{minDistanceSum}{maxDistanceSum} \tag{4.3}$$

4.1.4 Comparison Measure for Color Histogram Ridge

To compare the Color Histogram Ridges between two fingerprints, we used OpenCV's correlation measure, which when given two histograms H_i and H_m, returns the correlation as:

$$correlation(H_i, H_m) = \frac{\sum_i (H_i[i] - \overline{H_i})(H_m[i] - \overline{H_m})}{\sqrt{\sum_i (H_i[i] - \overline{H_i})^2 (H_m[i] - \overline{H_m})^2}} \tag{4.4}$$

where $\overline{H}$ is the average of H's histogram values. We confine the correlation results of normalized histograms to be capped at 1.0 in the maximum and 0.0 at the minimum, if OpenCV happens to return values outside of that range.[1]

4.1.5 Ridge Comparison Measure for Straight Line Ridge

To compare the Straight Line Ridges between two fingerprints, since this ridge is composed of three separate, normalized histograms, we first apply the correlation measure to each histogram, capping results at 1.0 as the maximum and 0.0 as the minimum. We then average the results of the three individual histogram correlation measures and return that value as our similarity measure for the Straight Line Ridge. In experimentation, if using only the orientation and color from lines, then the average of only those two values is returned.

4.1.6 Comparing Two SPLINTR Fingerprints

When comparing two fingerprints, a fingerprint report object ("fingReport") is created that stores the similarity measures for each ridge comparison result. Each ridge is compared separately following the comparison measures outlined in this chapter, and the results are stored individually in the fingReport. To get an overall similarity measure between two fingerprints, we apply a weighted summation of the individual ridge similarity results of the five ridges:

$$\begin{aligned} totalSim = {} & wtKey * keypointsSimMsr + \ wtObj * objSimMsr \\ & + wtLines * linesSimMsr + \ wtText * textSimMsr \\ & + wtColorHist * colorHistSimMsr \end{aligned} \tag{4.5}$$

[1]For example, in testing this measure in OpenCV, a strictly dissimilar pair of histograms returned a value slightly less than 0.0.

The individual weights are adjusted to follow the dynamic weight of the keypoints. Also, weights may be increased if ridges are excluded from the measure, as described next.

4.1.7 Fallback Procedure

What does the system do when no features can be extracted from specific ridges? We outline SPLINTR's fallback procedure next. Since we have multiple visual feature ridges, it is possible that, say, no objects are recognized or detected in a particular location. If there are likewise no objects recognized or detected in the map fingerprint with which we are comparing, then that ridge similarity measure is excluded from the overall similarity measure and its weight is evenly distributed to the other ridges. Similarly, if no keypoints were extracted in the keypoint ridge of our current location, and no keypoints are stored in the map fingerprint, that ridge similarity is also excluded and its weight distributed to the other ridge similarities. This same fallback procedure also applies to the Straight Line Ridge and Recognized Text Ridge. At the minimum, the system defaults to just a single ridge, the Color Histogram Ridge, as we can always measure the color distribution of an image, even if the image is pure black.

4.2 Evaluating the Need for Symbolic and Generic Features

Next we illustrate validation for branching beyond the use of texture based keypoints and including multiple visual feature ridges. Systems reliant upon keypoint extraction may ignore other available information to make an intelligent decision. Even environments that may look self-similar often have small differences, such as the doors have different room numbers or name plates, an emergency light or fire extinguisher is present at specific places, or a water

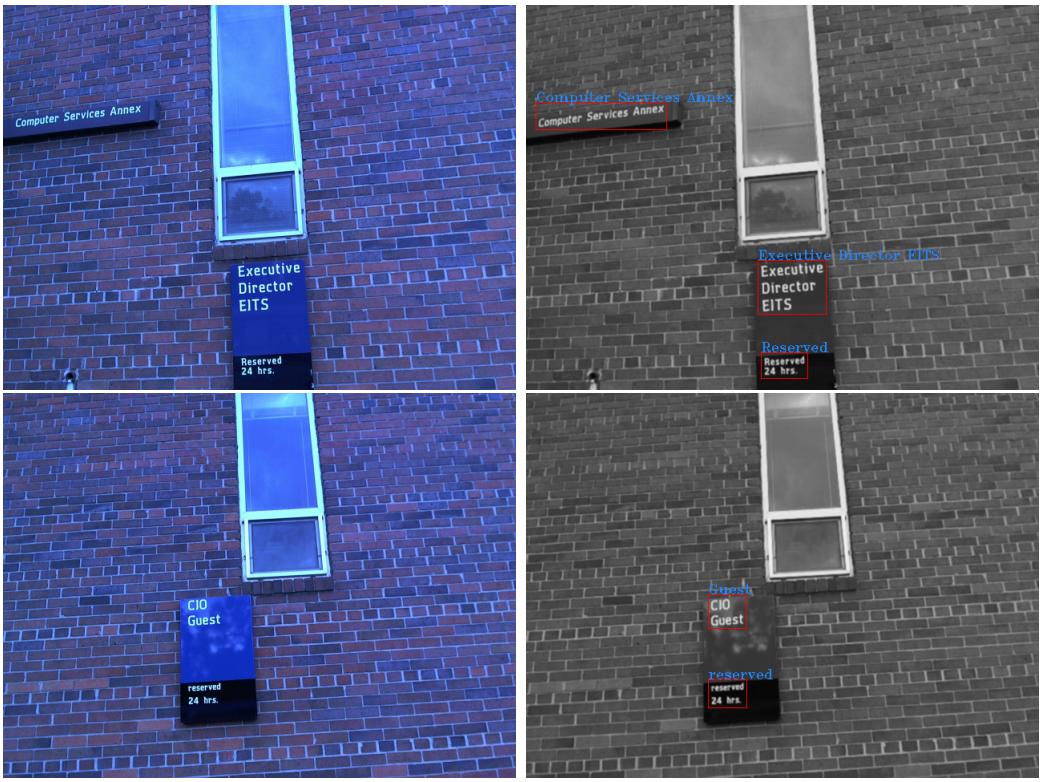

Figure 4.1: For these outside images, these results were obtained after we turned off the histogram equalization which adjusts the contrast of the input images. Notice in the top image set, the rules have eliminated "hrs" (punctuation is delimited during checking) as well as "24" since it is too short of a word, and in the bottom image set, "CIO" was incorrectly recognized at "C10" and was eliminated. These two parking spots are located along the same side of the building, though form two separate parking spots.

fountain is located in a hallway but not in another. We see that as we store and compare multiple sources of visual evidence, we start to develop a more intelligent representation of the environment. Assume in the examples below that our threshold for considering two locations a "match" is set at a total similarity measure of 0.7.

In Figure 4.1, we show images from two parking spot locations outside of the Computer Services Annex on UGA's campus. Clearly the two images are different, as can be seen by the results of the Recognized Text Ridge. According to the SPLINTR comparison measures

between the ridges of the fingerprints resulting from these two locations,

****

sim_key: [0.791667]

sim_obj: [excl]

sim_lines: [0.984120]

sim_<lines:ori>: [0.977138]

sim_<lines:color>: [0.991102]

sim_colorhist: [0.981295]

sim_text: [0.277778]

sim_TOTAL: [0.758715]

****

we see that the ridges agree and are all above our threshold of 0.7 except for the Recognized Text Ridge. Some rules may be formed to eliminate some wrong matches, or warn if the robot is about to make a wrong decision. We could implement rules checking for this type of case, where

key && lines && color && !text → display warning.

We implemented this, and we receive this text warning for this case:

WARNING: Consider disregarding match. Failed "similar texture, dissimilar text" check.

This does require more confidence in the scene text extraction method.

Another case may exist where two locations agree on many ridge comparisons, except in the ridges/sub-ridges measuring color distribution. For example, in Figure 4.2, we show two images taken from outside the elevators on the fifth and sixth floors of Boyd. The elevator area is painted different colors, a pink/orange color on the fifth floor, and a smoky color on the sixth floor. It may be difficult to distinguish in black and white, and is also hard to distinguish in the color photographs, but it is rather obvious in person and shows up differently in the SPLINTR ridges measuring color distribution. Even though they are of

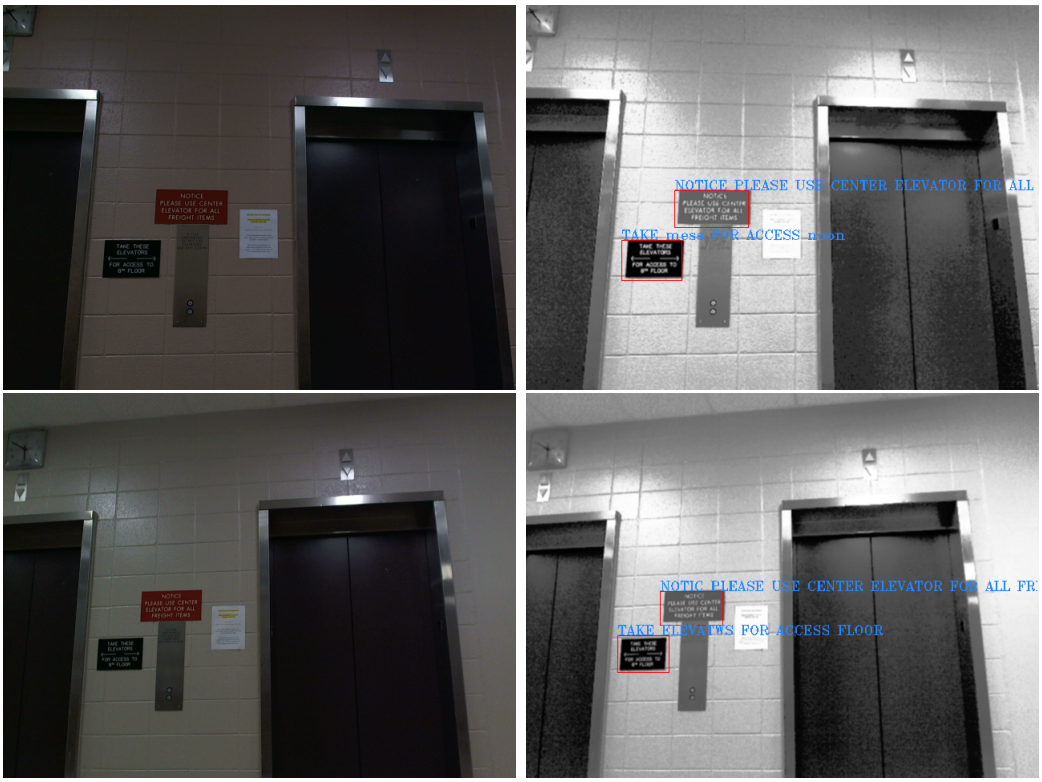

Figure 4.2: Images from outside the elevators on the fifth and sixth floors of Boyd Graduate Studies Research Center.

similar locations functionally, the "fifth floor elevator" and "sixth floor elevator" do occupy two separate locations and meanings.

Below, we see a difference in the Color Histogram Ridge comparison measure and the line color comparison measure in the Straight Line Ridge (the images were also taken at slightly different angles, so the line orientations also differ). We see that the keypoint similarity measure is extremely high. We use SIFT, and SIFT is actually extracted in a grayscale version of the input images. Since these locations are quite similar, if we were to exclude color information, we may make a wrong decision. It happens that the other ridges are low enough to make the total similarity fall below the threshold, but we can form a rule for

situations such as this, whether Keypoint Ridges agree and color does not, or if most ridges agree including text, but color does not. Such rules could be executed on the robot, or could signal for human assistance in making the decision.

****

sim_key: [0.931035]

sim_obj: [excl]

sim_lines: [0.357245]

sim_<lines:ori>: [0.480306]

sim_<lines:color>: [0.234185]

sim_colorhist: [0.225892]

sim_text: [0.835294]

sim_TOTAL: [0.587366]

****

Thus, we have identified places, environments, or situations where multiple ridges are useful. Sometimes these locations are similar enough to still be considered the same location by our system, though we have identified certain areas where rules can be triggered should symptoms call for it. These are simple cases using a single directional image for feature extraction, rather than multiple images from different angles, but does show the possible mistakes made if relying on a single ridge or the consensus of ridges without considering what ridges disagreeing with the consensus are trying to tell us. Other rules may be formulated for other uses of ridge information to rule out mismatching locations, and these rules could be used to trigger other behaviors, such as ignoring a match, identifying tricky situations, or signaling for assistance in decision making.

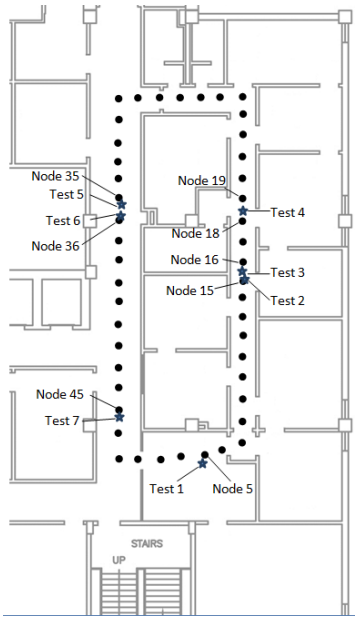

Figure 4.3: In the localization tests, the map consisted of 46 locations from the Boyd Graduate Studies Research Center's first floor, indicated as black circles. Test (query) locations are represented with stars. The test nodes and nearby map nodes are labeled in the figure. Figure from [78].

4.2.1 Experimental Results

To test for localization using the comparison measures of SPLINTR fingerprints, we used a map created from images collected from the first floor of the Boyd Graduate Studies Research Center building on UGA's campus. The map consisted of 46 discrete locations approximately one meter apart, navigating the inside hallway of the Institute of Artificial Intelligence and the outside hallway of the Institute; approximate locations of the map are depicted in Figure 4.3. These images were inputted to SPLINTR, features were extracted, and fingerprints were created to form the robot's "map." This data was collected from the point of view of an ER1 robot. To aid in capturing scene text of door numbers and name plates, the camera was extended above the ER1 approximately three feet and tilted at an upward angle of about 45 degrees. Though the dataset contains images from all four sides of

the robot, in the testing we utilized only the left and right images, which were fused into a single image as depicted in Figure 4.7c. Limiting the number of input images helped conserve memory space, as well as increased the focus of the topological locations to the sides of the robot rather than the oncoming and receding areas in front and behind the robot. A test set of locations was acquired by moving the robot to new locations and collecting images. In Figure 4.3, the map nodes are depicted with black dots and the test locations with blue stars; test locations and nearby map nodes are labeled. The map is formed by reading in the environment images and extracting the visual feature ridges and forming fingerprints on each location. For the localization tests, the system reads in the test location images, the scene features are extracted to form a fingerprint, and the test fingerprint is compared with each fingerprint in the map. Behavior for handling similar localization matches is discussed in the Topological SLAM chapter.

For objects to be recognized, we provided the system with a list of object images. The use of these objects could be considered as reconnaissance or "intelligence" information. Other images were captured and trimmed from cell phone images or webcam images of certain objects, like the smiling frog, the alarm, the fire extinguisher and exit signs, as shown in Figure 4.4. These can be read in at runtime, and the SIFT features extracted for use in matching in the environment. The objects used for detection in the localization experimental results include the IAI logo, the CASPR logo, an emergency light and exit sign (thus we have multiple ways of detecting exit signs in the environment). While not all objects may be detected in the environment (and false positives may get detected), SPLINTR is provided with the means of handling the objects found for use in localizing.

Figure 4.4: Object recognition images provided to the system.

Figure 4.5: Object images representing the trained object detection classifiers. We therefore had multiple methods to extract exit signs from the environment.

4.2.2 Results Discussion

Results of the comparison tests with the map are shown in Figure 4.6. These results use a threshold of 0.7 on similarity. We see that several tests have results with several locations above the threshold, and we discuss behaviors to handle such situations in the next chapter.

In looking closer at Test 3 results, there were opportunities to extract information for localization that did not get extracted based on these images and the trained object classifiers used in these experiments. The location is near a conference poster dealing with text analysis in schizophrenia, listing author names and the name of the research paper. In the test image, the keyword "Schizophrenia" was successfully extracted, but it was incorrectly misspelled (and therefore invalidated) during extraction in the map image. Furthermore, there were opportunities to extract research group logos which did not get successfully extracted in this experimental run relating to Test 3. In analyzing Test 3, we see the opportunity SPLINTR

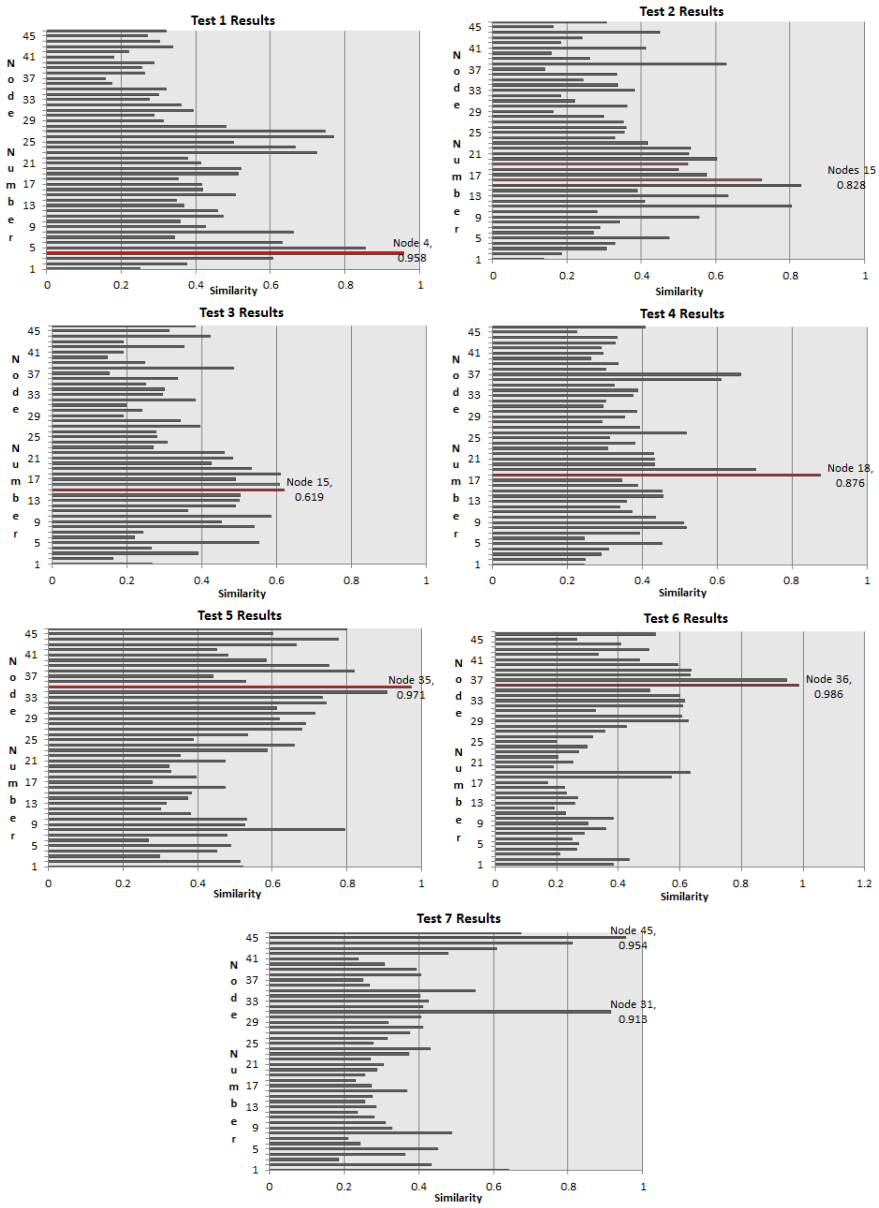

Figure 4.6: Result graphs using the test fingerprints and map fingerprints shown in Fig. 4.3. The closest fingerprint match from the map and the similarity measure is indicated. Test 7 shows the closest match as well as the second closest match. We see the location for Test 2 has several matches above the threshold. Handling of this type of situation when encountered during SLAM is described in the next chapter.

has to capitalize on and use for localization, but due to false recognition of text (keyword "Schizophrenia" extracted in the input but not successfully in map node 15) and missed detection of logos (two CASPR logos are visible, but only one is detected in the Test 3 image and none in map node 16), the similarity does not exceed the 0.7 threshold. We received less than desirable text recognition results in the localization tests. Images were dark. Keypoint extraction was also muted, but still useful, in these tests. Images used in the topological SLAM experiment were taken with a different webcam and were also brighter.

Test node 7 is highlighted in greater detail in Figure 4.7. What is shown is the test image as the top image, and the middle and bottom images are the closest matches according to the fingerprint comparison results with the map. As shown, using a method of matching such as counting keypoint matches may cause an incorrect localization decision, as shown in Figure 4.8.

In discussing some things we learned from the combination of rudimentary features with semantic features from the environment, there may be areas of the system not fully exercised given the set of images representing the environment or the path or locations of the robot. This could be manifested in not providing or having enough objects in the environment recognized or detected. Furthermore, scene text can be rather sensitive and fragile when using Tesseract, depending on the sub-image we provide the engine. Some of the noise returned from Tesseract was curbed through the use of rules, though still better scene text extraction is an area of future work in this system. Still, within SPLINTR, there are other layers of the system to help in making a decision, but since objects and text are both important to localization, false positives can be harmful. Likewise, true positives of an object or text can help clear up uncertainty on position, such as detecting the emergency light in the hallway can reduce uncertainty as to which part of the hallway the robot is located. Test 7 shows a hard problem of localization, but in using multiple visual layers of the SPLINTR system, Test 7 returned higher quality matches.

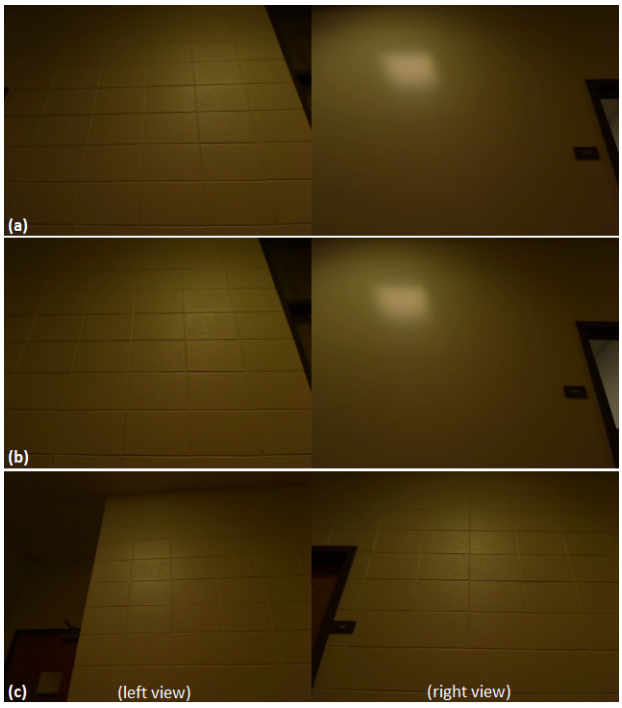

Figure 4.7: (a) The query image, Test 7, (b) The scene image of Test 7's closest fingerprint match from the map (Node 45) and (c) the scene image from its second closest match (Node 31). Left and right views from the perspective of the robot are shown. From [78].

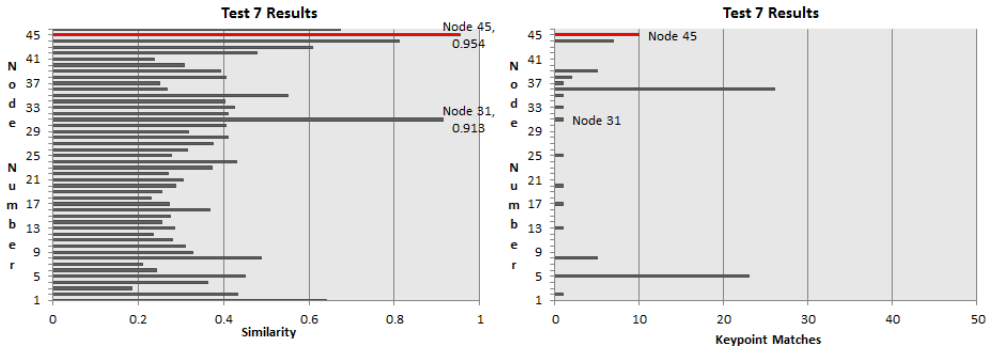

Figure 4.8: Fingerprint similarity of Test 7 with the map. Compare this to the number of keypoint matches. We can see that two locations have high keypoint matches. These two locations each have a common keypoint that matches well to many different keypoints in the input image.

Chapter 5

Topological SLAM

Next, we investigate using the visual fingerprints previously defined in Chapter 3 and compared in Chapter 4 to simultaneously localize and topologically map an environment by an autonomous mobile robot.

5.1 Topological Mapping Defined

A topological map is essentially a graph of visited locations; places are represented using nodes in the graph, and paths between locations are represented by graph edges. As Mataric explains [44] (Ch. 12, page 147), these locations in a topological map can be considered "landmark" locations. The topological map and its relation to human cognitive maps are found in the early work of Benjamin Kuipers; such cognitive maps were first comprehensively studied in Kuipers's dissertation work of the TOUR model [30]. In the later work of Kuipers and Byun [31], the TOUR model was applied to robots, with "distinctive places" representing the nodes in the robot's topological map, which are connected by "travel edges." This is differentiated from metric mapping in that it does not consist of metric placement of walls

with a legend and scale, similar to what one would see of a blueprint or printed layout of a building.

5.2 Using SPLINTR for Topological Localization and Mapping

Topological mapping with SPLINTR involves our already defined visual fingerprints in combination with a graph structure to maintain the connection between locations. To facilitate localization, we use the fingerprint similarity measure we described in Section 4.1.6. Furthermore, we use vectors to maintain hypotheses on our current position, positions to expect next, and to facilitate intelligent loop closing.

The motivation behind this work is that we need an intelligent way of attaching together in the map the places that the robot visits. We cannot iteratively add fingerprint nodes in a linear string, because this would not be a true representation of the structural layout of the environment. The robot may turn around and try to map the same area that it just covered, or travel in a loop and revisit a location already mapped earlier in its exploration. There needs to be a mechanism for handling these situations and maintaining an up-to-date belief of where the robot thinks it is in its topological map. Therefore, we identified specific situations a SPLINTR controlled robot may encounter, and we handle these scenarios accordingly to build a topological map.

We may use the terms "fingerprint" and "node" interchangeably at times when discussing our topological SLAM approach. Essentially in our implementation, when we go to save a fingerprint within our map, we store the fingerprints within a node data structure that can then be used in the topological graph forming our map. Thus, if we need to compare the input fingerprint with fingerprints in our map, we can easily use a pointer to the fingerprint structure within the graph node data structure. The node just wraps the fingerprint so that

we can associate fingerprints together in the map and connect them with edges. If, say, we decide to store the fingerprint of our current location within our map, we create a new node and assign to it the data of our current location's fingerprint. Therefore, usually we will talk about the input of our current location, we actually mean the latest input after we have taken a step from our *currentActiveLocation* at time t, a known position in our map, to our next location in the environment at time $t + 1$. We use the term "*currentActiveLocation*" to reference our last known whereabouts within the map, until the current scene is analyzed and becomes the new *currentActiveLocation*, and the former *currentActiveLocation* becomes *previousActiveLocation*.

5.2.1 Introduction to SPLINTR's SLAM Behaviors

Using SPLINTR, the robot begins building its map by first capturing images at its first location and running ridge feature extraction, extracting keypoints, recognizing and detecting objects, extracting statistics on straight lines and the color of its surroundings, and extracting text from the images, and storing these within a fingerprint that represents its current location. The first case is simple, our map is empty, and thus our first location's fingerprint input goes into a node which is stored as the first node of our topological map. The robot then proceeds forward, captures images, extracts features, and forms the fingerprint *fp_Input*. Now we must decide what to do with this fingerprint.

Generally speaking when exploring its environment using SPLINTR, the robot will be executing under three different situational behaviors. While exploring, localizing and mapping, the robot will either be following a path, that is, exploring an area that it has already mapped and following along with its position in its map (further discussed in Navigating a Topological Path, section 5.2.3); it could have detected multiple loop closure candidates and is attempting to eliminate false matches (Sequence Landmark Localization, section 5.2.4); or it could be in unexplored territory and looking through its map to see if it has been at

this location before (Loop Detection Check, section 5.2.2). We elaborate on each of these situations, as well as define structures we will need to maintain sets of nodes that represent 1) where we think we could be, 2) which locations are loop match candidates, and 3) which nodes to expect upon moving again.

We are going to define a few terms to help us illustrate key locations. Let us call the *currentActiveLocation* the node in our map at time step t. The robot takes another step, and has 1) an idea of where it came from (*currentActiveLocation*) and 2) a new current location with fingerprint features already extracted (*fp_Input*). We now discuss what to do with *fp_Input* given the different situational behaviors.

5.2.2 Loop Detection Check

The first situation we discuss handling is when the *currentActiveLocation* is a new node added to the map at time step t; thus, the robot is not on a path, not confirming a loop, it is simply exploring and mapping new territory according to the last time it checked its map. When it moves again (let us call the robot now at time step $t+1$), it must check to see if *fp_Input* at time step $t+1$ is a place it has previously visited and mapped. Thus, it iterates over its map, comparing *fp_Input* to each fingerprint node in its map, to see if it can find one or more fingerprint node matches in the map. We define a "match" to be a fingerprint in the map whose similarity measure with *fp_Input* exceeds a certain threshold.

This iterative checking of *fp_Input* to map nodes leads to three possible outcomes: the robot does not find a match, we find one match, we find multiple matches. We describe handling each of the three outcomes below.

1. *fp_Input* is unique enough to save to our map. (The robot does not find a match in its map)

That is, since each similarity measure between *fp_Input* and each node in its map is below a certain threshold, we now have evidence that this is a unique location, and can create a

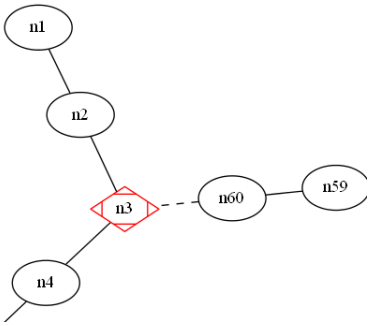

Figure 5.1: Example of closing the loop, where the current input (which would be node 61) matches well with node 3, thus the loop is closed and we do not need to save node 61 into the map. *currentActiveLocation* is indicated by a red diamond node, and the loop closure edge indicated by a dashed line.

new node in our map. We therefore create a new node for the fingerprint *fp_Input*, update *currentActiveLocation* to now be the *previousActiveLocation*, set *currentActiveLocation* to be the new node, and attach an edge between *previousActiveLocation* and *currentActiveLocation*. Once we add the new node to the map, we take another step in the environment and then repeat *Loop Detection Check*.

2. *fp_Input* matched a single fingerprint node in our map.

That is, when using our similarity measures to compare visual fingerprints between the input and each node in the map, we found one single map node similarity that exceeded the threshold. We take this as evidence to close a loop, illustrated in Figure 5.1. To handle a simple loop closure event, an edge is added between the last node before loop detection (which in this case is still set as the *currentActiveLocation* node) and the winning candidate match from our map. Our *currentActiveLocation* node is updated to the winning candidate match. We then transition to the *Navigating a Topological Path* behavior.

3. *fp_Input* matched more than one fingerprint node in our map.

That is, we compared our current input fingerprint to each node in the map, and we found more than one good match in our map; essentially, we should be able to close a

loop on one of these nodes, but we found more than one node that is a good match, so we do not know which is the true node with which to close our loop. To help us solve our problem of multiple loop closing candidate matches, we transition to the *Sequence Landmark Localization* behavior.

Special case: If it just so happens that the best match from among the set of loop closing candidates is the *currentActiveLocation* node, this is treated as a special case. We add a loop edge between *currentActiveLocation* and itself and transition into *Navigating a Topological Path*; this takes precedence over Sequence Landmark Localization, and thus even if more candidate matches exist, we eliminate them immediately in favor of closing the loop on the *currentActiveLocation*. Put simply, the input fingerprint *fp_Input* looked like several locations in our map, but most like the location we just left, and we therefore do not have enough differentiation between it and our current scene to warrant a new fingerprint node in the map.

5.2.3 Navigating a Topological Path

When we are following a topological path, our sense of where we are is based on comparison of our current location's fingerprint, *fp_Input*, and the nodes we expect one horizon out from the *currentActiveLocation* node, which is our last known location in our map. After we take a step in the environment, extract the fingerprint for the current scene as *fp_Input*, we now need to decide at which node we are located within the map.

If the robot is following a previously mapped path, then it expects to take a step and still be on the mapped path. Therefore, from *currentActiveLocation*, we extract the neighbors (the set of nodes one horizon out from *currentActiveLocation*, that is, the predecessors and successors of the node) into a "look ahead" set of nodes. We also add *currentActiveLocation* to the *look ahead* set. There are a couple of reasons for this. The robot may not move very

far when it takes a step, and thus the input scene may be very similar to the *currentActiveLocation*; furthermore, the robot could have avoided an obstacle while taking a step and turned to avoid it, and thus may have changed directions part way through a step. Whatever the case may be, the robot could be at or near the last known location in its map, so we need to be aware of it to match against *fp_Input*. Taking this into consideration, we expect to find a good match to *fp_Input* from among this *look ahead* set. An example is illustrated in Figure 5.2.

When we use the *fp_Input* and compare it with the nodes from the *look ahead* set, we can expect one of three different cases: the robot finds zero matches, it finds one match, it finds more than one match.

1. If we find zero matches for *fp_Input* among our *look ahead* set, then either the robot was kidnapped and placed somewhere else within the environment, or the robot has stepped off of its path. There is another option as to why it did not find a match, which is that the next location is indeed close in spatial proximity to one in the look ahead set, but the fingerprint from the scene still did not match well; the input scene could be different enough visually to not exceed the threshold, or perhaps something did not extract in the current scene like it did in the fingerprint in our map. We still treat this as stepping off of the path, because we encountered a location that was not similar enough to any node from the set we expected from being on a mapped path.

What do we do when we step off of a path? Update the *currentActiveLocation* to become the *previousActiveLocation*, and make a new node out of *fp_Input*, which now becomes our *currentActiveLocation*. Lastly, we add an edge between the *previousActiveLocation* and *currentActiveLocation* in our topological map.

How does SPLINTR handle the kidnapped robot problem? Should the robot not find a good match from among the *look ahead* nodes, it reverts to searching the entire topological

map for a good match. Should it find one, it updates its current location to the match from its map.[1]

2. If we find one single good match for *fp_Input* from our *look ahead* set, we update the *look ahead* set to be the neighbors of the good match (along with the good match itself) and set our current location to the good match. Following this, we remain in the situational behavior for *Navigating a Topological Path*.

3. If we find multiple good matches for *fp_Input* from our *look ahead* set, that is, the current location fingerprint matches more than one of our *look ahead* nodes, then we transition to the behavior of following an ambiguous path. This situation is essentially like following a topological pathway, but our *look ahead* set will be updated with the successors and predecessors from all of the good matches (along with the good matches themselves). Since we found a good match from our *look ahead* set (more than one, actually), we are now aware that our next step could be from the neighbors of more than one node. We save the best match into a variable so we have a reference to the "most likely" location, but we are still on a path and thus expect to find our next step from among this special (multiple neighbor) *look ahead* set. This ambiguity should work itself out; that is, we stay on the mapped path and find a single good match within a future *look ahead* set. If we step off of the path while following the ambiguous path handling procedure, however, we connect the new node to the "most likely" location we just described.

5.2.4 Sequence Landmark Localization

Next, we discuss how SPLINTR handles closing a loop when presented with more than one loop closing candidate node within its map. To solve this problem, we use a technique we call sequence landmark localization.

[1]This feature is implemented, but was turned off during experimentation. Should an exercise call for the use of this behavior, it can be turned on prior to compilation.

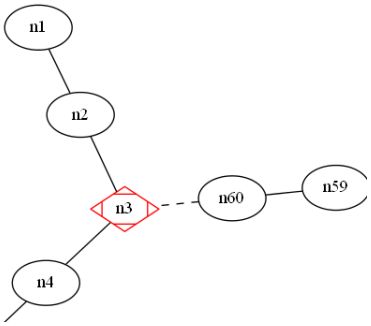

Figure 5.2: In following the topological path behavior, from current location Node #2, we expect the next location to be one of: Node#3, Node#1, Node#50, or Node#2 (self).

Sequence landmark localization (SLL) is a form of loop closing verification; it is our process to eliminate false loop candidate matches when multiple nodes in our map are good matches to close a loop. When the comparison results of the scene and the map nodes are not sufficient to close a loop due to perceptual aliasing, that is, when multiple locations in the map look similar, the general premise of sequence landmark localization is to use a sequence of nodes to validate loop closing candidates. SLL is only used when there are multiple loop closing candidates within the map; the reason is that when the robot closes a loop using only a single matching node, it may simply be crossing the previously constructed path in its map, and thus it would not be able to use a sequence of nodes to verify this loop closing case; therefore it would not be on a previously mapped path (except for that one matching node). Therefore we are justified in using sequence landmark localization only when there are multiple loop closing candidates.[2]

The prerequisite for this situational behavior was discussed previously in section 5.2.2, regarding the third possibility upon issuing and completing a loop detection check, where

[2]Future work of including directional information could be useful, or a verification behavior of single candidate loop closure could be implemented, where further exploration is done in a local neighborhood to verify loop closure.

fp_Input, the fingerprint of the robot's location at time $t+1$, matched more than one fingerprint node in our map.

So what do we do? We put all of the candidate matches into a vector – this allows us to separately hold and verify all matches as we proceed in the Sequence Landmark Localization process. The robot remains in this state of limbo, not knowing exactly where it is in its map, but having multiple good ideas on where it is. We then transition into a topological pseudo-path following behavior, but path following from several possible locations within the map; in addition, we will be using the sets of "look ahead" nodes as proof for or against the validity of candidate loop closing nodes. More specifically, we then fill out what is essentially "look ahead" sets of nodes for each of these candidate matches; the robot takes a step in the environment and looks into each candidate's "look ahead" set with the intention of finding a good matching fingerprint node to its new input, and thus finding a reason to keep a candidate match alive. The theory here is that if we were at a particular candidate match location, and we take another step, we should find the neighbor node that we are now at, and thus proving we were actually at the candidate match when beginning SLL. Only those candidates for which we find proof to keep will continue in consideration to close the loop. The look ahead sets are replaced with the next horizon's look ahead set, the neighbors of the nodes for which we found matches, each still associated with the ancestor line of a particular candidate match. We continue following this process of checking good matches' neighbor nodes each horizon out as we continue to move until we come to certain decision points:

1. We eliminate all but one candidate match node.

We have now used sequences of nodes as proof for or against closing a loop at those specific nodes in the map. The last remaining candidate match becomes the loop closer, and we transition into navigating a topological path based on our current map position from using the look ahead nodes as verification.

2. We eliminate all candidate matches from consideration.

This means we encountered a number of loop false alarms. In this case, sequence landmark localization has eliminated the false alarm loop closing candidate locations, and the locations we have seen between then and now should be added into the map as if we mapped them sequentially. In this case, all the fingerprints from the time that triggered loop detection until now are added on sequentially to the node before the loop detection was triggered. An example of SLL and the handling of this decision point is described in Figure 5.3.

3. We reach a consensus on our present location in our map.

We may reach a consensus on our present location without having eliminated all of the loop candidate matches. In this special case, we update our current location to be the only matching look ahead node from our elimination process. Secondly, the loop is closed at the most likely loop candidate match, the node with the highest similarity from among the map fingerprints that triggered SLL.

Discussion

We must discuss how we handle an ambiguous intersection with our map. An ambiguous intersection occurs when the robot intersects with the map at a node that looks like other nodes but immediately proceeds into unexplored territory. In this situation, we are left with no way to resolve which spot on the map we actually intersected. This case is indistinguishable from a loop detection false alarm. Behavior could be implemented in future work which could cause the robot to change direction and follow a path in order to perform SLL to verify the candidate matches.

5.2.5 Implementation and Visualization

In addition to the topological mapping behaviors, we need mechanisms for handling the structure of the map as well as a visualization process, so that we can visually see the links between nodes and the beliefs of where the robot has evidence that it is.

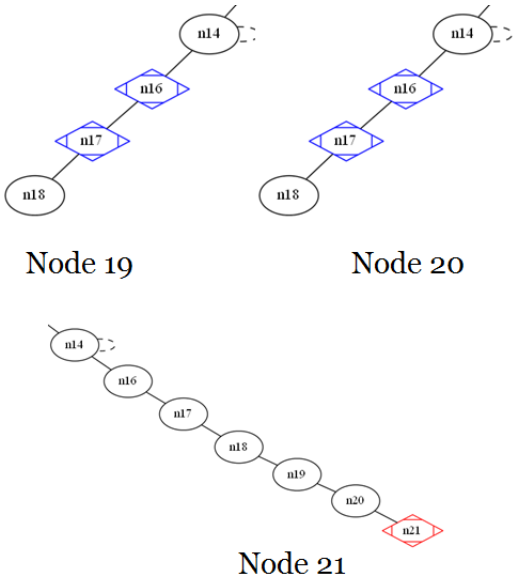

Figure 5.3: As an example, input node 19 looks similar to nodes 16 and 17 of the topological map (the uncertainty is indicated with the blue diamonds), which initiates sequence landmark localization. The next step, Node 20, likewise leaves SPLINTR thinking it could be at either 16 or 17. Node 21 clears up this uncertainty, since it did not match well to any of the adjacent nodes considering where we thought we were, and the input nodes that were accumulated are added sequentially to Node 18.

For housing the visual fingerprints, and for managing connectivity of the topological map, we used the graph library within LEDA v6.3, the Library of Efficient Data types and Algorithms [69]. The graph nodes provided by LEDA essentially wraps around the fingerprint, allowing us to access to the fingerprint with a pointer as needed.

For visualization purposes, the topological map is simultaneously mimicked with Graphviz v2.28 [1]. This allows us to color code and apply different shapes to nodes such as the current active location, ambiguous path following locations, and possible locations we could be at during sequence landmark localization. It also allows us to visually assess node connectivity.

5.2.6 SLAM Experimental Results

For data collection, we use the iRobot Create, with a Logitech c910 webcam attached to a tripod, and a Microsoft Kinect to assist in exploration and obstacle avoidance. In our SLAM experiments involving data collection of the environment, the general behavior (since the robot is equipped with a single forward-facing camera) is to rotate in place at approximately 90 degree angles, taking images from its front, left, back and right sides, before facing forward again and moving to the next spot. This process is repeated at approximately one meter intervals (though the distance is a variable that can be set prior to compilation). This leaves room for future improvements, such as the use of an omnidirectional camera, multiple cameras, or other methods. When the robot is not avoiding an obstacle or rotating in place to collect images of its surroundings, the autonomous exploration behavior is set to move toward depth in the environment, using depth information from the Kinect.

To illustrate the results of the topological SLAM behaviors of SPLINTR, we collected data from the first floor of the Boyd GSRC building, as shown in Figure 5.4. The two lengths of hallways can be noted as being quite textured in the inside hallway (roughly locations 2 through 17 and 60 to 67), and visually ambiguous at locations in the outside hallway (roughly locations 25 to 53). In Figure 5.5, we see the resulting topological map from

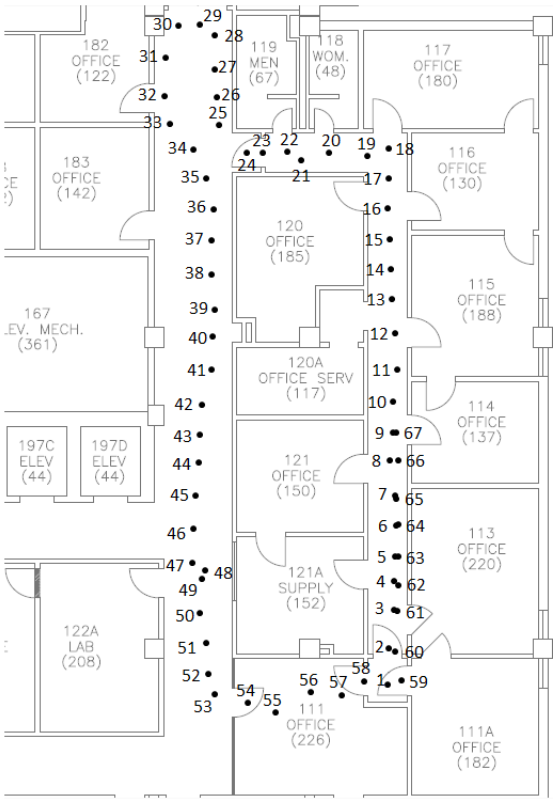

Figure 5.4: Image locations and numbers from a dataset collected from the first floor of the Boyd Graduate Studies Research Center building, UGA. This is a different dataset than the one used in the localization experiments.

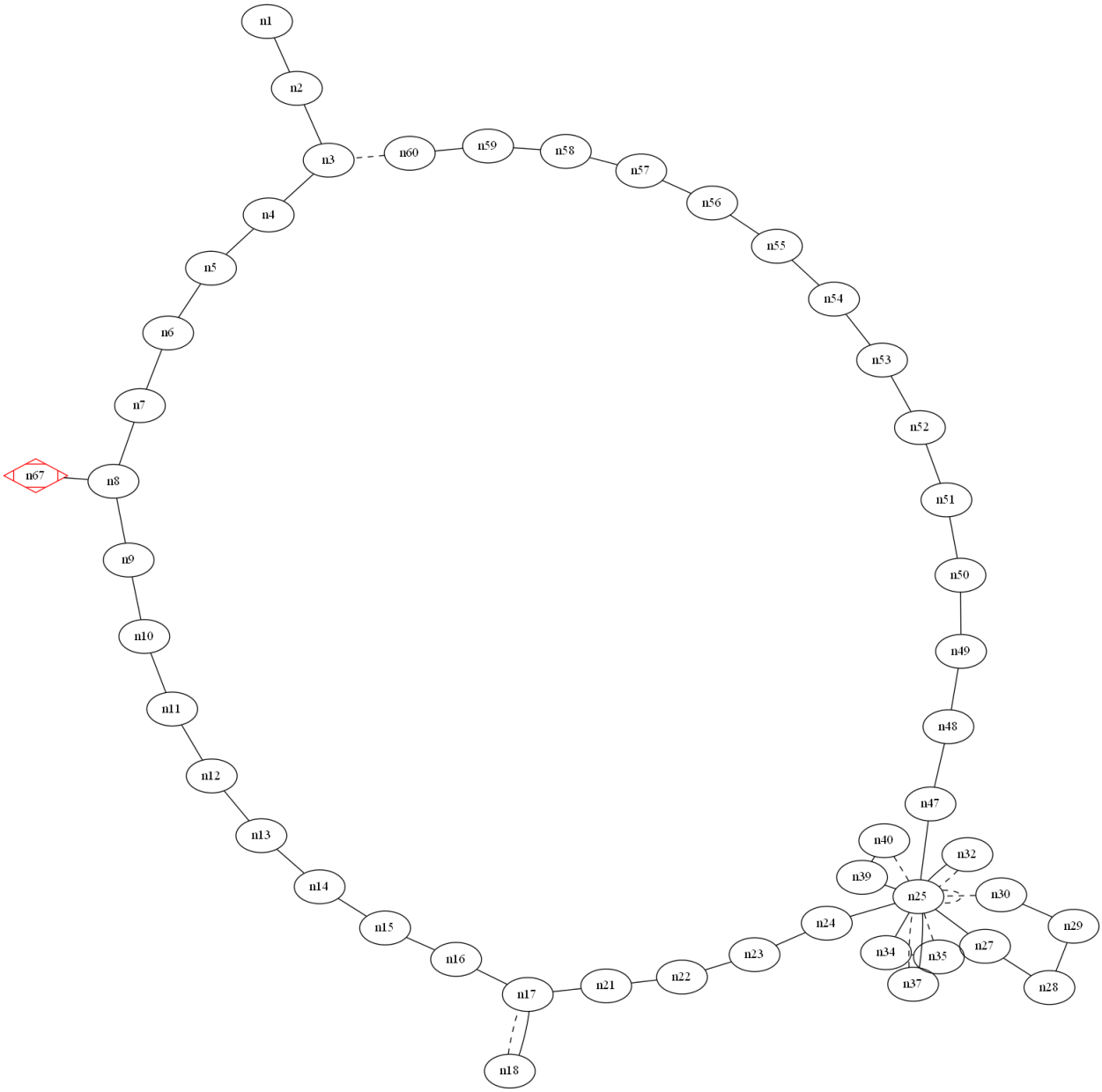

Figure 5.5: The resulting topological map from SPLINTR, when handling the locations in sequence from location 1 to location 67.

inputting the images from locations 1 through 67 using a similarity threshold of 0.7. Similar to the localization experiment, we limited the input images to just the left and right image data collected from the robot. We see the self-similarity of the outside hallway manifest as a coil of nodes on the right side of the topological map. The inside hallway is more separable, and also results in successfully closing the loop, connecting node 60 with node 3. We see the last node, node 67 did not match well to node 9. In looking into this result, since the data were collected over multiple exploration sessions, when the scene resulting in node 9 was taken, the door was only slightly cracked open, but swung farther open when the images for node 67 were collected. Since the inside hallway is narrow in width, a door being open or closed can result in a large change in the features extracted, especially when limited to only the left and right viewpoints of the robot. Since the scenes did not result in similar enough fingerprints, a new node, node 67, was created in the map. The same objects were used for object recognition and detection in the SLAM experiment as were used in the localization tests, though different trained classifiers were used for the IAI and CASPR logos.

Through the progression of the locations in the outside hallway, the fingerprint of node 25 matched well to multiple locations, and therefore node 25 topologically represented multiple places in the outside hallway. We show node 25 along with one of the locations that matched well with it (location 38) in Figure 5.6. Below, we show in more detail why the fingerprint from location 38 (as numbered in the floorplan in Figure 5.4) was matched with the fingerprint of node 25. Below, we list the number of keypoints detected at location 38, the resulting weights applied to the ridges, and the similarity measure resulting from comparing the fingerprint from location 38 with map node 25.

numKeypointFeaturesDetected: [166]

averageKeypointCount: [886.05]

****

wt_key: [0.062449]

Figure 5.6: (Top) The images from the node 25 location. (Bottom) The images from location 38, one of the locations that matched well to (and was "absorbed" by) node 25.

wt_obj: [excl]

wt_lines: [0.468775]

wt_colorhist: [0.468775]

wt_text: [excl]

****

sim_key: [0.833333]

sim_obj: [excl]

sim_lines: [0.686460]

sim_<lines:ori>: [0.422548]

sim_<lines:color>: [0.950373]

sim_colorhist: [0.952214]

sim_text: [excl]

sim_TOTAL: [0.820212]

****

Since no text or objects were detected at location 38, the ridge weights among the remaining ridges should each be one-third. However, since there were fewer keypoints detected at 38 than the average up until that point, the keypoint ridge weight was lowered and distributed to the remaining ridges. Therefore, the generic ridge features were leaned upon more in localization for the fingerprint at location 38, which produced a high match with the fingerprint of node 25.

Chapter 6

Conclusion and Future Work

In this chapter, we discuss the contributions of SPLINTR toward the goal of simultaneous localization and mapping. We also identify areas of the system needing improvement, and highlight ideas for future additions to the SPLINTR system.

6.1 Conclusion

Humans use multiple sources of evidence to conclude about location, drawing from the color and texture of our surroundings, the lines resulting from the architectural structures, and semantic information like objects and text. Since humans have these intelligent mechanisms, we can conclude that such techniques would benefit a robot in localization and mapping as well. A robot that utilizes this mixture of information could better, more intelligently represent its environment, and have access to multiple layers of visual evidence from which to draw conclusions about location. With a set of rich visual features and knowledge on how to connect locations together, it could form a topological map of its environment. In this dissertation, we have presented our approach to a better, more robust, and more intelligent representation of places. SPLINTR, **S**patial **Pl**ace Recognition **in** a **T**opologically Mapping

Robot, offers a multi-layered visual fingerprint of places to be used for topological mapping and localization. Our system combines complex machine vision features with global/generic image features for various evidence for recording locations and localizing within a topological map. By using visual features extracted to represent locations as fingerprints of keypoints, objects, text, line statistics and color information, SPLINTR provides a mobile robot multiple sources of information with which it could utilize for localization and mapping its environment.

We discussed the SPLINTR fingerprint structure in Chapter 3, describing in detail the multiple "ridges" of extracted visual features and the representation methods we use in each ridge. We furthermore discussed the various rules and validation measures used in our text recognition pipeline, connecting the text detection method of Samarabandu and Liu [64] with the Tesseract OCR engine. In Chapter 4, we discussed how to compare individual ridges between two fingerprints, as well as our method of weighting these individual ridge similarity results to combine them into a single overall fingerprint similarity measure. We showed the results of applying SPLINTR to the task of robotic localization, comparing test locations to a set of fingerprints representing the robot's map. We analyzed certain locations that would call for the use of multiple visual features to make a decision and discussed rules that could be incorporated to keep the robot from making a mistake given similar looking locations. In Chapter 5, we discussed using these fingerprints and comparison measures to form a topological map of the environment. We also discussed how we handle situations during the robot's explorations, how to maintain hypotheses of its location, and how we close loops even when presented with multiple loop closing candidates.

We have offered in SPLINTR an approach to include rudimentary image features such as lines and color, along with features sampling from texture and higher up human elements including text and objects. Not only do these items populate the nodes of our topological map, they are utilized for localization as well. We have shown examples where the inclusion

of more visual information would be useful in certain areas of the environment. This form of rudimentary image features combined with symbolic/semantic information also has ties to semantic mapping. We have shown our method of utilizing such data not just for inclusion in the map, but for use in localization as well. We have shown that combining common sense rudimentary image features with semantic information provides a robot with more information with which to localize and form a richer map.

6.2 Future Work

In this section, we discuss areas of future work regarding the SPLINTR system. Some ideas include useful extensions to the system, but we also identify areas of the system needing further work in its present form.

6.2.1 Fingerprint Sets

Fingerprint Sets is a proposed idea to modify the way SPLINTR collects nodes in its map. Presently, if the current location's fingerprint matches well with one and only one fingerprint node in the map, no new node is created and our position is updated to the found location in the map. The fingerprint of the current location from the new visit, however, may contain additional information not extracted and stored in the mapped fingerprint from the first visit. That is, it could have extracted more/different text, more/different keypoints, or recognized or detected more/different objects during its return visit. Within the idea of fingerprint sets, when SPLINTR finds a fingerprint node match in its map, it starts to add them to the set, keeping each fingerprint separate, but grouping them together in a fingerprint set structure. This differs from the mean fingerprint/clustering idea of Tapus and Siegwart [70]. Thus each fingerprint in a fingerprint set could indeed be representing the same location (or within close proximities), but just more information, or they could be just similar looking

nodes, such as stretches of a plain hallway. Fingerprint sets would give us better coverage of our environment, since a single node will not have to represent bigger stretches of the environment than it is capable of; also the idea is to store more information extracted upon a revisit of a location, which may help to further differentiate areas. This may be important in the absence of certain behaviors to account for close-up images of walls, or objects located higher in the environment than the image can cover.

6.2.2 Extensibility Opportunities of SPLINTR

Opportunities exist to extend the capabilities of the SPLINTR system described thus far. We describe certain extensions below.

Sound Ridge

Branching out beyond visual appearance fingerprint representations would be beneficial. Equipping the robot with a means of estimating the audio signature of a location could further lead to better representation of space and place recognition. The ideas here include being able to audibly characterize locations, such as the noise in hallways near elevators compared to the quiet of hallways away from elevators; the noise from a location near a stairwell versus locations farther away. This would give the robot even further evidence to differentiate locations, especially those that may lack visual differences but vary in sound signature differences.

Fingerprint Disk Storage

Many choices within the Fingerprint structure implementation relay well to saving out information to disk, so that it can be read in later, so that this information can be shared, or so that the map does not have to be kept all in memory.

SPLINTR GUI

SPLINTR's topological map nodes can be extended into a graphical user interface, where hovering or clicking on nodes can display the stored ridge data of that fingerprint.

Text Detection and Recognition

Further extension of the text recognition ridge is needed to make it more robust given the environment. We deactivated the camera's automatic light adjustment feature, as we did not want the camera affecting the color balance of the image of the environment while exploring. This allows the color histogram information to change according to the environment and not the camera. The text detection algorithm would benefit from a learned filter that would adjust the image before detection to get a uniform response from text detection given the environment lighting.

Active Camera

The current system uses a single, forward facing camera to collect image data from the environment. It would be beneficial to the system to include an active camera, one that can move and scan over areas of the environment, panning, tilting and zooming as needed. This would allow the capture of objects from various positions and heights within the environment. Furthermore, equipping SPLINTR with object and text tracking from a continuous image stream could help with missed detections of text and objects.

Bibliography

[1] Graphviz graph visualization software. http://www.graphviz.org/.

[2] The CMU Pronouncing Dictionary v. 0.7a. http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

[3] Tesseract release notes (v3.02), November 2012.

[4] M. Agrawal, K. Konolige, and M. R. Blas. CenSurE: center surround extremas for realtime feature detection and matching. *Lecture Notes in Computer Science*, 5305:102–115, 2008.

[5] Algorithmist.com. *Longest Increasing Subsequence*.

[6] S. Argramon-Engelson. Using image signatures for place recognition. *Pattern Recognition Letters*, 19(10):941–951, 1998.

[7] D.R.Ramesh Babu, Piyush Kumat, and Mahesh Dhannawat. Skew angle estimation and correction of hand written, textual and large areas of non-textual document images: A novel approach. In *International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*, pages 510–515, 2006.

[8] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

[9] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.

[10] P. Blaer and P. Allen. Topological mobile robot localization using fast vision techniques. In *IEEE International Conference on Robotics and Automation*, pages 1031–1036, May 2002.

[11] P. Blaer and P. Allen. A hybrid approach to topological mobile robot localization. Computer science technical report series, Columbia University, 2005.

[12] T. Botterill, S. Mills, and R. Green. Speeded-up bag-of-words algorithm for robot localisation through scene recognition. In *Image and Vision Computing New Zealand (IVCNZ 2008)*, pages 1–6, 2008.

[13] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[14] D. Braunegg. Location recognition using stereo vision. A.I. Memo 1186, Massachusetts Institute of Technology, 1989.

[15] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: binary robust independent elementary features. In *Eurpoean Conference on Computer Vision: Part IV*, pages 778–792, 2010.

[16] C. Case, B. Suresh, A. Coates, and A. Ng. Autonomous sign reading for semantic mapping. In *International Conference on Robotics and Automation*, pages 3297–3303, 2011.

[17] E. Ward Cheney and David R. Kincaid. *Numerical Mathematics and Computing*. Thomson Brooks/Cole, 6th edition, 2007.

[18] J. H. Clear. The british national corpus. In P. Delany and G. Landow, editors, *The Digital Word: text-based computing in the humanities*, pages 163–187. MIT Press, 1993.

[19] Lab color space. *Lab Color Space, Wikipedia.* `http://en.wikipedia.org/wiki/Lab_color_space`.

[20] M. Cummins and P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *International Journal of Robotics Research*, 27(6):647–665, 2008.

[21] Mark Cummins and Paul Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011. originally published online Nov. 2010.

[22] N. Durand, S. Derivaux, G. Forestier, C. Wemmert, P. Gancarski, O. Boussaid, and A. Puissant. Ontology-based object recognition for remote sensing image interpretation. In *19th IEEE International Conference on Tools With Artificial Intelligence*, volume 1, pages 472–479, 2007.

[23] D. Filliat. A visual bag of words method for interactive qualitative localization and mapping. In *IEEE International Conference on Robotics and Automation*, pages 3921–3926, 2007.

[24] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–151, 1988.

[25] J. Hays and A. Efros. IM2GPS: estimating geographic information from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[26] Rob Hess. An open-source sift library. In *ACM-MM*, 2010.

[27] D. Hoiem, A. Efros, and M. Hebert. Geometric context from a single image. In *International Conference of Computer Vision (ICCV)*, volume 1, pages 654–661. IEEE, October 2005.

[28] X. Hou and L. Zhang. Saliency detection: A spectral residual approach. In *CVPR*, 2007.

[29] J. Kosecka and X. Yang. Location recognition and global localization based on scale-invariant keypoints. In *Workshop on Statistical Learning in Computer Vision, European Conference on Computer Vision*, 2004.

[30] B. J. Kuipers. *Representing Knowledge of Large-Scale Space*. PhD thesis, Massachusetts Institute of Technology, 1977.

[31] Benjamin Kuipers and Yung-Tai Byun. A robust, qualitative method for robot spatial learning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-88)*, 1988.

[32] D. Santosh Kumar and C. V. Jawahar. Robust homography-based control for camera positioning in piecewise planar environments. In *5th Indian Conference on Computer Vision, Graphics and Image Processing*, pages 906–918, 2006.

[33] Noah Kuntz. Opencv tutorial 6 - chapter 7. Online Resource, 2009.

[34] P. Lamon, I. Nourbakhsh, B. Jensen, and R. Siegwart. Deriving and matching image fingerprint sequences for mobile robot localization. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1609–1614, 2001.

[35] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. (In Russian. English version - Soviet Physics Doklady 10(8): 707-710).

[36] Cristóbal Carnero Liñán. cvBlob. http://cvblob.googlecode.com.

[37] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM 25th Pattern Recognition Symposium*, pages 297–304, 2003.

[38] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. *IEEE ICIP*, 1:900–903, 2002.

[39] Tony Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2):224–270, 1994.

[40] Y. Liu, T. Yamamura, T. Tanaka, and N. Ohnishi. Character-based mobile robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 610–616, 1999.

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

[42] N. Maillot and M. Thonnat. Ontology based complex object recognition. *Image and Vision Computing*, 26(1):102–113, 2008.

[43] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *International Conference on Computer Vision*, volume 2, pages 416–423, July 2001.

[44] Maja J. Mataric. *The Robotics Primer*. MIT Press, 2007.

[45] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *8th International Conference on Computer Vision*, pages 525–531, 2001.

[46] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 60:63–86, 2004.

[47] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[48] M. Muja, R. Rusu, G. Bradski, and D. Lowe. REIN - A fast, robust, scalable REcognition INfrastructure. In *International Conference on Robotics and Automation*, 2011.

[49] A. Murillo, J. Guerrero, and C. Sagues. Surf features for efficient robot localization with omnidirectional images. In *IEEE International Conference on Robotics and Automation*, pages 3901–3907, 2007.

[50] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[51] P. Neubert, N. Sunderhauf, and P. Protzel. Fastslam using surf features: An efficient implementation and practical experiences. In *6th IFAC Symposium on Intelligent and Autonomous Vehicles (IAV)*, 2007.

[52] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–174, 2001.

[53] M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S. Mehrotra, and T. Huang. Supporting ranked boolean similarity queries in mars. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):905–925, 1998.

[54] O. Pellejero, C. Sagues, and J. Guerrero. Automatic computation of the fundamental matrix from matched lines. *Current Topics in Artificial Intelligence, LNCS-LNAI*, 3040:197–206, 2004.

[55] I. Posner, P. Corke, and P. Newman. Using textspotting to query the world. In *Intelligent Robots and Systems*, pages 3181–3186, 2010.

[56] Google Code Project. Tesseract ocr 3.01. http://tesseract-ocr.googlecode.com.

[57] I. Rañó, E. Lazkano, and B. Sierra. On the application of colour histograms for mobile robot localization. In *European Conference on Mobile Robotics*, pages 189–193, 2005.

[58] M. Rahman, P. Bhattacharya, and B. Desai. Similarity searching in image retrieval with statistical distance measures and supervised learning. In *International Conference On Advances In Pattern Recognition (ICAPR 2005)*, volume 3686 of *Lecture Notes in Computer Science*, pages 315–324, 2005.

[59] J. Rogers, A. Trevor, C. Nieto-Granda, and H. Christensen. Simultaneous localization and mapping with learned object recognition and semantic data association. In *Intelligent Robots and Systems*, pages 1264–1270, 2011.

[60] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, 2005.

[61] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443, 2006.

[62] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, 2011.

[63] Kenneth S. Saladin. *Anatomy and physiology: the unity of form and function*, chapter 6, page 196. McGraw-Hill, 2007.

[64] J. Samarabandu and X. Liu. An edge-based text region extraction algorithm for indoor mobile robot navigation. *International Journal of Signal Processing*, 2:273–279, 2007.

[65] D. Schleicher, L. Bergosa, M. Ocana, R. Barea, and E. Lopez. Real-time stereo visual slam in large-scale environments based on sift fingerprints. In *IEEE International Symposium on Intelligent Signal Processing*, pages 1–6, 2007.

[66] Naotoshi Seo. Mergevec. http://note.sonots.com/SciSoftware.html, 2007.

[67] Steven S. Skiena. *The Algorithm Design Manual*, chapter 8, pages 289–291. Springer, 2008.

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

[69] Algorithmic Solutions Software. Leda C++ library. http://www.algorithmic-solutions.com/leda/.

[70] A. Tapus and R. Siegwart. Incremental robot mapping with fingerprints of places. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2429–2434, Aug. 2005.

[71] Adriana Tapus, Nicola Tomatis, and Roland Siegwart. Topological global localization and mapping with fingerprints and uncertainty. In *International Symposium of Experimental Robotics*, 2004.

[72] M. Tomono and S. Yuta. Mobile robot navigation in indoor environments using object and character recognition. In *IEEE International Conference on Robotics and Automation*, pages 313–320, 2000.

[73] I. Ulrich and I. Nourbakhsh. Appearance-based place recognition for topological localization. In *IEEE International Conference on Robotics and Automation*, pages 1023–1029, April 2000.

[74] S. Vasudevan, S. Gachter, V. Nguyen, and R. Siegwart. Cognitive maps for mobile robots - an object based approach. *Robotics and Autonomous Systems*, 55(5):359–371, 2007.

[75] E. Vincent and R. Laganiere. Detecting planar homographies in an image pair. In *2nd International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2001.

[76] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.

[77] J. Wang, H. Zha, and R. Cipolla. Coarse-to-fine vision-based localization by indexing scale-invariant features. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 36(2):413–422, April 2006.

[78] B. J. Wimpey and W. D. Potter. Using visual fingerprints of places for robotic localization. In *International Conference on Artificial Intelligence*, July 2011.

[79] B.J. Wimpey and W.D. Potter. Generating smooth virtual reality maps using 3D building blocks. In *Proceedings of the 2008 International Conference on Computer Graphics and Virtual Reality*, 2008.

[80] C. Zhou, Y. Wei, and T. Tan. Mobile robot self-localization based on global visual appearance features. In *IEEE Internatinal Conference on Robotics and Automation*, volume 1, pages 1271–1276, Sept. 2003.