UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGENEERING AND COMPUTING

MASTER THESIS nr. 360

Localization of a mobile robot within the visual memory

Petra Bosilj

Rennes, June 2012

I would like to thank my supervisor Siniša Šegvic for all the patience and support he offered during my studies, and especially while I was writing this thesis. He was always happy to help and share his knowledge with his students, and I am very glad I was able to learn from him. Seeing his unwavering inspiration has inspired me to pursue a research career.

I would also like to thank my co-supervisor, François Chaumette for inviting us to work with him and his team, and for his interest in my works.

My thanks to the whole Lagadic team in Inria, for providing such a supportive and enjoyable work environment.

Special thanks to Josip Krapac, for every time he listened and helped with my work on this thesis.

My thanks to all my friends for their support and their input. There are too many of them to mention all their names, but none of their help is forgotten.

Additionally, I would like to thank the whole `stackoverflow.com` *community for sharing their ideas when I needed a second opinion. They have made the process of searching for information faster and much more enjoyable.*

Last, but not the least, I wish to thank my family who always encouraged me to do the best I possibly can.

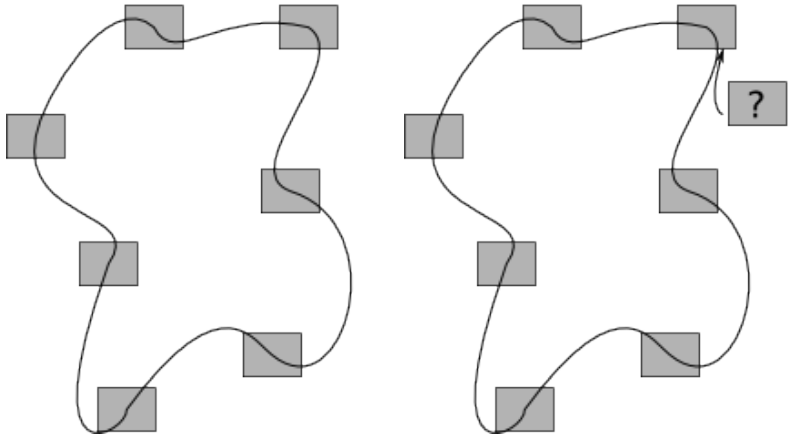
CONTENTS

1. Introduction and related works

The goal of this thesis was to research and implement methods and models that would allow efficient content-based queries on a given image database. Developed procedures would be used to find the topologically closest image contained in a robot's visual memory. A robust visual feature matching procedure is also researched, to be applied to pairs consisting of the current location image and image retrieved from the visual memory as the closest (most similar) match.

Wider context for developing such components is the construction of an appearance-based navigation framework for large outdoor environments, divided, similarly to [15], in to the mapping, task preparation and navigation phase. In *appearance-based* approach to navigation ([3], [15]) the control algorithm relies on (visual) sensor readings correlated with the topological information stored in robot's memory. The topological information is created during the *mapping phase* where the robot is navigated by a human controller. During this phase, the *visual memory* is formed by selecting key images from the video of the entire motion. The properties of the key images selected need to allow successful visual navigation on the path between the images, and are addressed in a complementary Master thesis [14]. The *navigation* phase (also the focus of [14]) uses the information stored in the visual memory and the data extracted during task preparation to achieve autonomous robot navigation along the recorded path.

The main problems addressed in this Master thesis are the processes performed during the *task preparation* phase [15]. Firstly, explicit topological localization of the robot is performed, by determining the image from the visual memory most similar to the image of the initial robot position (cf. Figure 1.1). Secondly, a fine-level localization with respect to the closest and most similar visual memory image (with its topological information available) has to be performed. In order to achieve this, correspondences between interest regions of the initial position image and the selected visual memory image have to be determined. These interest regions correspond to the positions and sizes of visual features extracted from the images. Further fine-level localization is achieved by the recovery of multiple view geometry based on the de-



(a) (b)

Figure 1.1: Figure (a) is a schematic representation of the visual memory. The visual memory contains the key images depicted on the robot path. Figure (b) summarizes the topological localization task: for the image of an unknown location, the goal is to find the topologically closest image from the visual memory.

termined correspondences during the navigation phase, and exceeds the scope of this thesis.

1.1. Topological localization

Searching for the image contained in an image collection most similar to a query image is a problem addressed in the field of *content-based image retrieval*. Approach explored in this thesis is heavily based on procedures previously utilized in the field of language processing. The concepts of *term frequency–inverse document frequency (TF-IDF)* (c.f. [12, 11]) and *inverted files* (cf. [12]), conventionally used for content based search of text documents, are exploited in [13, 9] for CBIR. In [13], viewpoint invariant regions with complementary properties are extracted and treated as *visual words* that serve a role similar to search terms in text analytics. Shape Adapted regions that are centered on corner-like features are extracted, as well as Maximally Stable regions corresponding to blobs of high contrast with respect to their surroundings. These regions are then represented by SIFT (Scale-invariant feature transform) descriptors, first introduced in [5], which are suitable for the purposes of this thesis because they are invariant to small shifts in region positions. Since the best match within the visual memory is expected to be an image taken with robot position and rotation offset from the query image, some localization error in the region extraction process is to be ex-

pected [13]. SIFT descriptors ensure a large subset of common visual words between subsequent key images, a property useful for CBIR application as well as for the latter processing step of visual features matching.

The basis for the CBIR part of this thesis is [9], where the visual words are extracted from images in the visual memory in a similar manner as in [13]. An improvement to scalability with respect to the methods presented in [13] is achieved in [10] by constructing a hierarchical tree structure from the collection of visual features, called a *vocabulary tree*. By utilizing a hierarchical search structure, authors of [9] allow for much larger database sizes compared to a non-hierarchical approach from [13] without compromises in speed or accuracy of the image retrieval process.

1.2. Wide baseline matching of visual features

The process of visual feature matching is implemented in two steps: firstly, a tentative correspondances set is established where a large portion of the features in the query image are matched with their closest matches in the visual memory image, which gives a significant portion of false matches. The performance of simple brute force matching where each feature is matched to the feature with smallest Euclidean distance between descriptor vectors is compared to the performance of the voting scheme proposed in [7], and the brute force approach to matching then further used for finding the basic correspondance set.

After the correspondance basis is established, the second step is applying several filtering methods aimed to reject false matches to the basis in varying orders and combinations. While relatively simple, the filtering methods rely on some expected image properties, such as small positional and rotational viewpoint offset (e.g. a match between a feature in the upper-left and a feature in the lower-right corner can most probably be rejected) and object compactness [13]. Filtering based on visual word frequency, also used in [13], is not explored in this thesis. The best combination of filters is selected empirically to be used in the final implementation of the matching procedure. The choice of feature detection algorithms and feature descriptors is shared between the CBIR and matching processes, eliminating the need for repeating the process of feature extraction for the query images. Best results were achieved with Difference of Gaussians (DoG) features [5] and Maximally Stable Extremal Regions (MSER) [7], while the developed implementation allows for easy choice between those and other types of features (e.g. Harris corners) as well as using a combination of features. Arguments that using a combination of features enhances the accuracy of the matching

process, as well as the support for the choice of features used in this thesis, can be found in [13, 15, 2]. Also, in both [15, 13] it is argued that a combination of features shows promise while used in tracking process, which, although not encompassed by this thesis, is the phase following the matching process in the appearance-based navigation framework according to [15].

The rest of the thesis is structured as follows: the next Chapter gives a detailed overview of the methods used in the thesis, the implementation is described in Chapter 3, followed by the analysis of results in Chapter 4. CBIR and visual feature matching are addressed separately through all the Chapters. The thesis is concluded in Chapter 5.

2. Overview of the developed framework

The developed procedures can be logically divided to activities conducted offline, before the navigation process, and the activities performed during the navigation process itself. The part to be used in the offline (learning) phase of the navigation process starts with visual memory already filled with selected key images. In the learning phase, a portion of the visual memory images is used to construct the structure supporting efficient CBIR queries, and image descriptors for all the key images are calculated.

After the offline phase, the initial activities needed for the navigation process begin. Upon the start-up of the robot, an image of the current location is acquired and the most similar database image retrieved by using the search structure constructed in the offline phase. The retrieved image should represent the closest topological location and the first location to be traversed during the navigation process.

For the robot establish the correlation with the mapped features, the multiple-view geometry must be recovered for the image pair consisting of the selected base image and the image of the current location acting as a query image. For that purpose, a robust matching procedure was developed taking into account specific assumptions about the image pairs arising from the intended domain (cf. Section 2.3). While the initial matching accepts a significant amount of false feature matches, a satisfactory accuracy is acquired after applying the appropriate filtering methods to the matches.

2.1. Feature extraction and feature descriptors

In order to achieve good performance of matching and content-based image retrieval methods, interest regions – local features with complementary properties have to be extracted from the images [2, 13, 15].

The first type of local features this thesis focuses on are *Maximally Stable Extremal Regions (MSER)*, first introduced in [7]. The other type of used features are based on

the extrema detection on the *Difference of Gaussians* images, introduced as a part of *Scale Invariant Feature Transform (SIFT)* pipeline in [5]. In addition to exellent performance of both types of features in wide-baseline matching [2], the SIFT features have high contrast and good localization along image edges and can be considered complementary to blob-like MSER features, which benefits the CBIR process.

After detecting interest regions, they need to be represented in a standardized manned to ensure efficient comparison between features. Each image is then represented by a set of local feature descriptors it contains. For this purpose, SIFT descriptors are used (the term SIFT is presently most commonly used in reference to only the feature description part of the pipeline presented in [5] and refined in [6]). The choice of MSER and DoG feature extraction methods ensures good feature localization while using SIFT descriptors provides invariance to small positional shifts of the regions that can occur when extracting features on images with a difference in scale and camera position.

Maximally Stable Extremal Regions are a subset or *extremal regions* proposed in [7]. In this article, extremal regions are defined by their two main desirable properties: this set of regions is closed under continuous transformation of image coordinates (that include perspective transformations) and under monotonic transformation of image intensities. The MSER regions are an affinely-invariant stable subset of extremal regions for which a detection algorithm of near linear complexity was developed [7]. Since an efficiend algorithm for extracting all of the extremal regions has not yet been developed, only MSER regions are of practical for the matching process and CBIR. The authors offer an intuitive explanation of the introduced MSER interest regions: all possible thresholds are applied to the input image, and the resulting black and white images are ordered in a sequence by increasing threshold value. From the set of all connected components from the entire sequence, the ones that are stable over the large range of thresholds are selected. This is done by only extracting the connected components at intensity levels that are local minima of the rate of change of the function describing the black area in the thresholded image.

Extraction of the features based on **Difference of Gaussians** extrema is conducted in several steps, according to [5, 6]. To determine the locations of the features invariant to scale changes of the image, the potential stable features are determined across all possible scales using a continuous function of scale known as scale space [6, 17]. A scale space is defined as a convolution of a variable-scale Gaussian, $G(x, y, \sigma)$ with an input image, $I(x, y)$ according to:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.1}$$

where $*$ is the convolution operator in $x - y$ space of the image, and the Gaussian is defined as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{2.2}$$

The initial image is repeatedly convolved with Gaussians and a set of scale space images, called an *octave*, is produced. Scale space images are produced through multiple octaves, where the initial image for the next octave is obtained by down-sampling the Gaussian image from the current octave that has twice the initial value of σ. The process of blurring the images with the Gaussian kernels is then repeated for the next octave [6].

Stable keypoint locations are not detected directly in the convoluted images, but rather in the difference-of-Gaussian function convolved with the image [6], $D(x, y, \sigma)$. The expression for calculating $D(x, y, \sigma)$ indicates that it can be easily computed by image subtraction from smoothed images $L(x, y, k\sigma)$ and $L(x, y, \sigma)$ which are separated by a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned} \tag{2.3}$$

This ensures near scale invariance of the features, as the difference-of-Gaussian function is a close approximation of the scale-normalized Laplacian of Gaussian required for true scale invariance (cf. [8]). Local extrema detected on the difference-of-Gaussian images are then considered keypoint candidates. During keypoint localization process, introduced in [6], keypoints that have low contrast or are poorly localized along an edge are rejected. For the remaining features, location, scale and octave (and from that, feature radius) are determined based on the interpolated location of the maximum.

The final step is assigning a consistent orientation to all keypoints, so that keypoint descriptors could be extracted relative to this orientation and achieve invariance to image rotation. For each keypoint, an orientation histogram is formed based on gradient magnitude and orientation values on the corresponding smoothed image corresponding to the determined keypoint scale. The highest peak in the histogram determines

the keypoint orientation, while the keypoints with two distinguished peaks in the histogram are split to produce two different keypoints with a different orientation.

To represent the extracted features, **SIFT descriptors** (introduced together with DoG features in [5]) which ensure viewpoint and illumination invariance [16] are used. To calculate the descriptor, scale and octave information about the keypoint is used. Gradient magnitudes and orientations are sampled around the keypoint location, with the gradient orientations rotated relative to the keypoint orientation. A Gaussian weighting function is applied to the calculated gradient magnitudes, with the σ equal to half of the width of the descriptor region. This region is then further divided into 4×4 sample regions and their orientation histograms with 8 directions are calculated. The magnitudes from the orientation histograms of all the sample regions form a keypoint descriptor with $4 \times 4 \times 8 = 128$ components, which is normalized before use.

2.2. Methods used for content-based image retrieval

The search structure used to perform content-based image retrieval on the images from the visual memory is based on ideas presented in [9], with TF-IDF scoring that was introduced to the field of content-based image retrieval in [13]. DoG and MSER features are used, with the choice explained in Section 2.1.

Usage of the developed CBIR system can be divided in two distinct phases:

- **Offline** or **learning phase**, in which a part of the visual memory is used to construct a *vocabulary tree* and *image descriptors* are calculated for all the images in the visual memory,
- **Online** or **retrieval phase**, in which the vocabulary tree is queried by an unknown image to try and find the topologically closest image (and consequently, the most similarimage) from the visual memory.

While the focus of [9] was to develop a technique that would primarily be *fast* and work on *large image databases* containing the images of *different objects*, this thesis attempts to utilize the same technique for a slightly different purpose. As the query is performed only once, on the beginning of the autonomous navigation (and before any actual movement of the robot), achieving highest possible speed was not among the priorities. Secondly, the visual memory is expected to contain on the order of 500 images (c.f. [9] works with database as large as 1 million images). Thirdly, the most distinguished difference in the application domain is that while the original article tries to retrieve an image of an object highly different from all the others in the database

(e.g. find an image of a shoe amongst images or guitars, flowers and rubber ducks), the consecutive images in the visual memory are similar, containing a high number of common features. Finally, the procedure is used here on environment images while the original article was mostly dealing with objects on a neutral background.

2.2.1. The learning phase

The main goal of the learning phase is to construct a search structure allowing for an efficient content-based search of the visual memory. This is achieved by forming a *vocabulary tree* using the process presented in [9]. The input to the learning phase of the algorithm are all images (more specifically, image paths) contained in the visual memory, and the learning is unsupervised. To limit the memory consumption and time requirements according to the characteristics of current computer ahchitectures, not all the images from the visual memory have to participate in the vocabulary tree construction. Loading of all the images and holding all the features from all the images in the memory is avoided.

The first step consists of retrieving all feature descriptors from all the images used as the basis of the vocabulary tree construction. The number of images used for the construction is a parameter of the construction process, with key images selected randomly. The goal is for the vocabulary tree to define a good quantization of the feature descriptors, where each quantization cell defines a single *visual word*. All feature descriptors that fall into the same quantization cell are represented by a same visual word, and an image can be thought of as a (multi)set of visual words it contains.

The quantization step is performed hierarchically, as proposed in [9]. For the construction of the first level of the vocabulary tree, a simple k-means clustering is performed on the training feature descriptors. This partitions the training data in k groups, where each group is represented by its cluster center in the first level of the vocabulary tree. The k-means clustering is then recursively applied on the feature descriptors belonging to each separate group, as illustrated in Figure 2.1. The tree is built level by level, up to maximum of L levels. An example of the tree built for $k = 3$, $L = 5$ is shown in Figure 2.2. The cluster centers in the last level of the tree define the final feature descriptor quantization and are treated as visual words.

After the vocabulary tree is constructed, only the quantisation of features belonging to images used in construction is determined. The set of visual words for the images from the visual memory not used as the basis for vocabulary tree is obtained as follows: each feature is propagated down the vocabulary tree to determine the quantization cell

it belongs to. Since all the feature descriptors and cluster centers are represented as 128-component vectors, determining the closest cluster center at each level is done by comparing the k dot products of a feature descriptor with each of the cluster centers. Thus, determining the visual word that represents a feature descriptor requires a total of kL dot products to be performed.

In order to be able to determine the relevance of the images in the visual memory to the query image, image descriptors need to be calculated. The similarity measure between images is based on the similarities between the paths down the vocabulary tree for the feature descriptors of the images from the visual memory and the query image. As some features are more common through all the images than the others, not all of them carry the same discriminatory properties. The weights assignment to the visual words in the last tree level as well as to the inner nodes of the vocabulary tree is inspired by TD-IDF, a technique adapted from the area of text analytics [11, 12]. In text analytics, weight of a search term is higher for more discriminative terms (i.e. the ones occuring in a small portion of the documents), while the words with no discriminative properties (e.g. "the", "a") are assigned weights close to zero. A similar weighting scheme, where feature descriptors with high frequency through all images are assigned lower weights is used in the context of image retrieval in [13, 9]. The entropy weighting used here assigns weights to all tree nodes according to the following equation:

$$w_i = \ln \frac{N}{N_i}, \tag{2.4}$$

where w_i is the weight assigned to the i^{th} node of the tree, N is the total number of images in the visual memory, and N_i is the number of visual memory images with

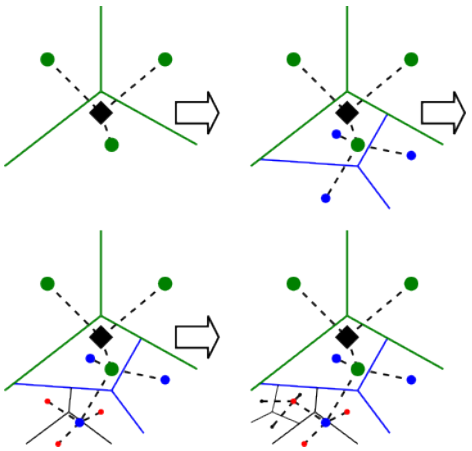


Figure 2.1: Process of recursive hierarchical clustering, with $k = 3$ for simplification. The initial clustering is shown in the upper-left corner image. The other images show recursive application of the clustering process to one of the initial clusters.

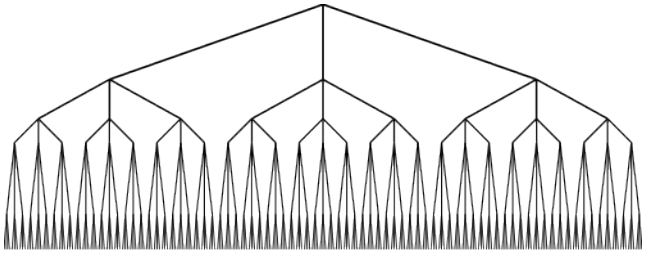


Figure 2.2: An example of the vocabulary tree for $k = 3$, $L = 5$. The depicted tree is a complete tree, while some of the branches can be missing in an actual vocabulary tree generated during the learning phase.

feature descriptors with a path through node i. An image descriptor contains one entry per each node in the tree, calculated as follows:

$$d_i = m_i w_i, \tag{2.5}$$

where m_i is the number of feature descriptors with a path through node i that come from the image for which the descriptor needs to be determined . It is important to notice that because of the weight assigning technique, a large portion of the weights equal to zero. Also, the maximum number of nodes through which feature descriptors from the same image can pass is at most L times the number of feature descriptors (if every descriptor is on a path through a different node on each level of the tree). That means that the total number of non-zero entries in the image descriptor is commensurate with the number of feature descriptors of an image (in the order of 300 in case of using only MSER, or 1000 in case of using both DoG and MSER features). This allows for an efficient storage of image descriptors as sparse vectors. The stored image descriptors are normalized to avoid scoring bias in favor of images with many feature descriptors.

When the vocabulary tree is queried with a new image, a relevance score between two images is calculated as follows:

$$s(\mathbf{q}, \mathbf{d}) = \Big\| \frac{\mathbf{q}}{\|\mathbf{q}\|} - \frac{\mathbf{d}}{\|\mathbf{d}\|} \Big\|, \tag{2.6}$$

where q and d are query image descriptor and the descriptor of an image from the visual memory, respectively. According to [9], the most suitable norm to use is the L_1 norm. To efficiently implement scoring, another technique from text analytics is used. The idea of *inverted files* was introduced in the process of designing an efficient engine to search through large textual documents databases [1]. In the inverted file, each leaf of the vocabulary tree stores the information about the images containing feature descriptors represented by the visual word of that leaf as well as the number of

such feature descriptors, m_i, for each of those images. The inverted files of the inner nodes are not stored explicitly, but are calculated as a concatenation of the inverted files of the leaf nodes as the approach suggested in [9] suggests.

2.2.2. The retrieval phase

When the vocabulary tree is queried with a new image, the first steps are to extract features and retrieve feature descriptors for that image and then calculate the query image descriptor in the same manner that image descriptors for the images from the visual memory were calculated. When the query image descriptor vector, q_i, is calculated and normalized, a score between the query image and all database images has to be calculated efficiently. Since the L_1 norm is used, the score between two normalized image descriptors can be expressed differently by the formula:

$$\begin{aligned} \|\mathbf{q}-\mathbf{d}\| &= \sum_i |q_i - d_i| \\ &= \sum_{i|d_i=0} |q_i| + \sum_{i|q_i=0} |d_i| + \sum_{i|q_i\neq 0, d_i\neq 0} |q_i - d_i| \\ &= \|\mathbf{q}\|_1 + \|\mathbf{d}\|_1 + \sum_{i|q_i\neq 0, d_i\neq 0} (|q_i - d_i| - |q_i| - |d_i|) \\ &= 2 + \sum_{i|q_i\neq 0, d_i\neq 0} (|q_i - d_i| - |q_i| - |d_i|) \\ &= 2 - 2 \sum_{i|q_i\neq 0, d_i\neq 0} b_i, \end{aligned} \tag{2.7}$$

where the fact that all the image descriptor components are non-negative is considered when calculating b_i:

$$b_i = \begin{cases} d_i & \text{if } q_i > d_i \\ q_i & \text{if } q_i \leq d_i \end{cases} \tag{2.8}$$

To simplify the calculations, the constant addend and a constant factor can be eliminated, obtaining a simpler relevance score in which higher values represent a better score for the image from the visual memory:

$$s'(\mathbf{q}, \mathbf{d}) = \sum_{i|q_i\neq 0, d_i\neq 0} b_i \tag{2.9}$$

This representation suggests an efficient way to calculate relevance scores in respect to the query image for all images in the visual memory. For each non-zero entry

of the query descriptor $q_i \neq 0$, all image descriptors with a non-zero entry $d_i \neq 0$ can be accessed easily using the information stored in the inverted file. All the relevance scores are calculated simultaneously, adding to each sum as the corresponding non-zero entry is encountered.

As all relevance scores are calculated simultaneously, the retrieval time is the same for accessing any number of highest-scored result images.

2.3. Approaches to wide baseline matching

Although the matching procedure has been developed with DoG and MSER features described by SIFT descriptors in mind, the presented approach is implemented modularly to allow the usage of any kind of vectorized descriptors calculated for arbitrarily selected visual features (e.g. Harris corners). The matching procedure can be divided in multiple modular steps, where the concrete method for each step can be chosen from all available methods requiring input and producing output data corresponding to the step:

- **Base step:** generates an initial set of potential matches, allowing for a significant amount of false matches. The potential matches are ordered according to the concrete method's similarity measure.
- One or more **filtering steps:** each of the filters rejects some of the proposed feature matches based on it's rejection condition. A varying number of filter steps may be applied to the matches in arbitrary order.

While widely-used methods for wide-baseline feature point matching have been tried out for the base steps, the filters were designed with specific feature point matching application in mind. Because of that, some filters show significantly better performance on the majority of the test image pairs (the typical settings) than on pairs of less common consecutive images from the visual memory (e.g. turns and curves of the mobile robot). Although the performance on the special cases may not be satisfactory with current parameters of the procedures, the potential shown through more common cases indicates that their performance may improve when using data collected in the other parts of the navigation process.

No information about the geometry between the pair of images is used. This is because the output of the matching is used in [14] as an input to the multiple-view geometry calculations, where the geometry is estimated and bad matches further rejected with *RANdom SAmple Consensus* (RANSAC) algorithm.

2.3.1. Base matching step

First algorithm considered as a base step in visual feature matching was a simple brute force matcher *based on Euclidean distance* between vectorized feature descriptors. The distance of the descriptor vector of the interest region from the query image and every descriptor vector from the base image is measured, and the query feature is matched to the base feature with the *smallest distance*. An additional criterion to improve the percentage of accepted false matches was introduced in [6], where the distance to the *second closest neighbor* is considered. If the distance ratio $closest/nextClosest$ is greater than a coefficient $k \in [0, 1]$, the match is rejected. This models the assumption that correct matches have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching [6]. Another additional criterion is proposed in [10]: in addition to calculating the closest neighbors from the base image of every feature descriptor in the query image, the closest neighbors are calculated after reversing the roles of the images. This produces a symmetrical algorithm, where only the feature pairs that were matched in both cases are contained in the final set of accepted matches.

The pseudocode follows for this simple matching method, allowing for a choice of the coefficient k (where $k = 1$ is equivalent to ignoring this criterion) and a possibility of the "marriage" criterion:

```
BFM(k, marriage, base_image, query_image)
    matches = {}
    for every feature_vector VQ from the query_image:
        (fst, snd) =  calc_closest(VQ, base_image, 2)
        if (fst distance < k*snd distance)
            add fst match to matches
    if doing marriage
        matches_reverse
            = BFM(k, false, query_image, base_image)
        for every match in matches
            if match not in matches_reverse
                remove match from matches
    return matches
```

As the basic brute force matching, as well as the functionality to find n nearest neighbors to the match are already implemented in the OpenCV library, the available imple-

mentations were used with additional conditions to implement two proposed criteria ([6, 10]).

The second algorithm considered for this step is the *voting scheme based matching* proposed in [7]. The core of the considered wide-baseline matching algorithm is the robust matching voting scheme focused on the component-wise similarities between description vectors as opposed to considering the descriptor vectors as a whole. In this method, when considering a query feature, every component of the descriptor vector is compared to the base feature descriptor components separately. For a component $\mathbf{q}_c$ in the query feature vector $\mathbf{q}$, the k of the base feature vectors $\mathbf{b^i}$ with the most similar component $\mathbf{b^i}_c$ cast a vote for a match between $\mathbf{q}$ and $\mathbf{b^i}$. A query feature gets matched to a base feature with most votes. The k parameter has been set to 1% of the features present in the images, as suggested by [7]. The pseudocode for this procedure can be written as follows:

```
VSM(k, base_image, query_image)
    matches = {}
    for every feature_vector VQ from query image:
        for every component c in feature_vector:
            find k feature_vectors from base image
                with min|VQ[c] - VB[c]|
            for every feature_vector VB out of k minimal:
                cast a vote for match(VQ, VB)
        VBH = match(VQ, VB) with most votes
        add match(VQ, VBH) to matches
    return matches
```

The implemented methods both return the matches sorted by quality, the match with the smallest Euclidean distance being returned first by the brute force approach and the match with most votes by the voting approach. Although it is not obligatory for a matching basis to work, some of the implemented filtering steps expect the potential matches set to be sorted by some priority criterion.

As the execution times of the voting scheme based matching were more than 3 times longer than the times for the brute force matching, and the testing of the brute force matcher produced satisfactory results (c.f. Chapter 4), only the brute force matcher is used in the final implementation.

2.3.2. Filtering steps

As the matching basis produces a set with a significant percentage of false matches, filtering must be carried out on those potential matches in order to achieve better accuracy. Each implemented filter exploits a certain assumption about the target pair of images.

The simplest implemented filter is the **Coordinate filter**. It considers every match, and keeps only the matches where base feature coordinates are within $d \times d$ pixel square around the query feature coordinates, as shown on Figure 2.3.

The reason for exploration of this approach is in the fact that image pairs used as inputs for the visual feature matching process are expected to be very similar, e.g. the offset between robot position and orientation used to capture the images can usually be assumed to be small (a significant amount of features used in tracking in [15], [14] is expected to be present in the image pair). This is true for robot movement along mostly straight paths and slight curves, where the filter shows good characteristics and improves the accuracy of base match set. Experiments with the filter show that its usefulness depends heavily upon window size and that the window size exhibiting good characteristics on the typical pictures has problems when applied to image pairs taken on strong curves. Still, usage of this filter could still be possible if an information about average or median feature displacement between consecutive key images is calculated during visual memory preparation in the learning phase to be used as a parameter for the filtering process.

The **Spatial filter** is based on the presumption that features of the same object stay localized through keyframes. This method corresponds to the spatial filtering method presented in [13], with slight differences: while the method in [13] was intended to

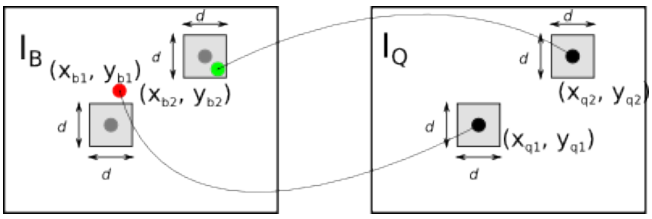


Figure 2.3: Schematic representation of the Coordinate filter. An example of a rejected match are the features with coordinates (x_{q1}, y_{q1}) from the query image I_Q and (x_{b1}, y_{q1}) from the base image I_B, shown in red. It can be seen that the base feature falls out of the $d \times d$ rectangle, shown around the query feature in I_Q and it's position (marked in gray) in I_B. An example of an accepted match are the query and base features with coordinates (x_{q2}, y_{q2}) and (x_{b2}, y_{b2}) respectively. The base feature from the accepted match is shown in red.

be used primarily when queries were only image clips representing one object, here the modifications to the method were needed to apply it to query images. For each matched pair of features, the following is considered:

- k nearest neighbors of the base feature from the matched pair are considered (good results with $k = 1\%$ of all base features, [13])
- median m of the distance between the neighbors and the base feature is calculated
- the *query area* is defined in the query image as a circle of radius $130\% \cdot m$ centered in the query feature from the matched pair,
- to cast a vote for the match, a neighbor of the base feature should be matched to a feature inside the *query area*

The term of *query area* had to be redefined with respect to the method in [13]. Since in [13] a query is a section of the image containing the queried object, the query area in the original method is simply defined as the entire queried section. Here, defin-

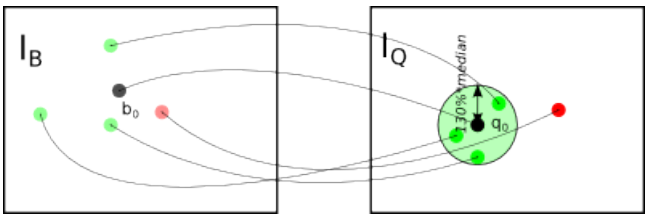


Figure 2.4: Schematic representation of the Spatial filter. A possible match between the feature q_0 from the query image I_q and feature b_0 from the base image I_b is considered. The 4 nearest neighbors of b_0 have been emphasized as well as their matches from the query image. The area highlighted in green in the query image is the *query area*. The neighbor match pairs that fall in to the query area and cast a vote for the initial match are marked in green, while the match pair that falls outside of this area and does not cast a vote is marked in red.

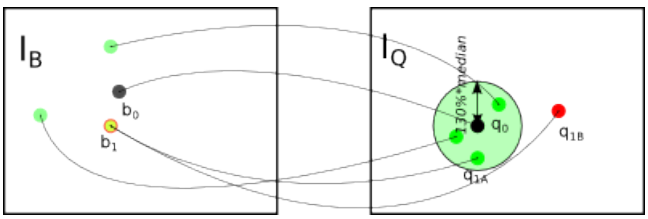


Figure 2.5: Explanation of the vote pondering for the Spatial filter. One of the neighbors of the base feature b_0, b_1 is matched to two query features, q_{1A} and q_{1B}. If the votes are not pondered, the match between b_0 and q_0 will receive 3 votes out of 4, giving it a score of 0.75. If votes are pondered, the match receives $\frac{1+1+0.5}{3}$ votes, receiving a slightly better score of 0.83.

ing the query area through the median of the base neighbors distances proved to be a good choice. Different percentages of the median distance were assessed, and the size of 130% of the median distance has been chosen for the query area. A vote threshold of 50% of all possible votes has been empirically chosen as the one with best performance. All the votes also have to be pondered since more than one query feature can be matched to a particular base feature. The graphical explanation of the base concept behind this filtering method can be seen in Figure 2.4, while the Figure 2.5 gives a graphical explanation of the vote pondering.

Output matches from the spatial filter are also ordered by the number of supporting votes cast in favor of each match, much like the base methods order their outputs by their respective priority criteria. It should also be noted that the results of this method of spatial filtering depend on the percentage of false matches in the initial matching set. This happens because with the low accuracy initial matching, the presumption that most features of the same object will say localized through keyframes no longer holds: since a lot of the matches are false, a significant amount of the features belonging to the same object are falsely matched and these false matches are not localized.

The last filter implemented is the **Multiple position filter**. Upon careful examination of the extracted features, it can be noticed that many of the visual features are extracted at the same (or very close) position within the image. Although those features do contribute to the accuracy of CBIR process and increase the chances of successfully matching a large portion of the extracted features, multiple correct matches in the same position are of no use in the multiple view geometry calculations for which the output of the matching process is used. Also, not all of the query image visual features from the same grid cell are matched correctly to their pairs in the base image.

This filter aims to remove query features that are spatially equivalent from the matching results. Ideally, it would leave only the best match from a collection of

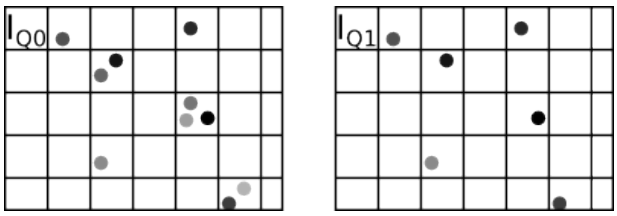


Figure 2.6: Schematic representation of the Multiple position filter. On the left, original query image I_{q0} is shown, split into grid cells. The darkness of the salient query features corresponds with the sorted order of the features, with the darkest features being the one with the highest priority. On the right, in the image I_{q1}, only one feature is left per grid cell.

matches located in in the same query image coordinates. For this reason, the input matches for this method are expected to be sorted by some priority criterion. It can be applied to all of the base matchers and filters introduced in this section, since both base matchers and the Spatial filter sort their outputs, and the Coordinate filter does not change the order of features that have not been rejected during the filtering process. If the input matches to for the Multiple position filter are not sorted before the filtering process, a match that appears first in the list of matches will be accepted, while all the other matches with the query feature in the same position will be rejected.

For the purpose of this filtering, the image area is split in to $x \times x$ pixels wide grid cells, as shown in Figure 2.6. When the query feature from the matched pair is encountered, the corresponding grid cell is marked. Any new matches with the query image falling in the same grid cell are rejected.

3. Implementation

Methods and procedures researched for this thesis are implemented in *C++* programming language, using *OpenCV*[1] open source library and following the style of the *ViSP* library[2], standing for Visual Servoing Platform. ViSP is an open-source cross-platform solution providing functional implementations of procedures used in visual tracking and servoing. The only other outer library used while implementing the methods of interest to this thesis is OpenCV computer vision library. Since the ViSP library is already using OpenCV, no new outside dependencies are introduced by using OpenCV functions. The implementation is focused on decoupling all the major functionalities, and providing access to them through interfaces or superclasses to ensure modularity of the components. Detailed explanations of the design patters used can be found in [4].

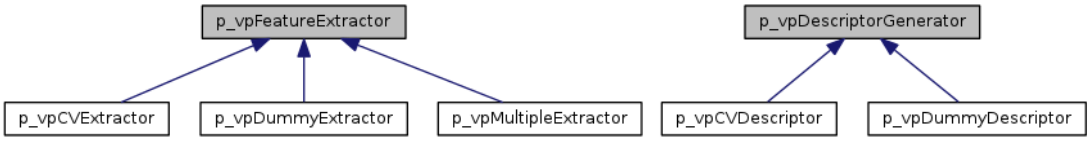
3.1. Modules used for feature detection

To store all the information concerning an interest region of an image – a visual feature, the class `p_vpExtractedFeature` is implemented. The purpose of this class and the data stored in every instance is comparable to class `cv::KeyPoint` from OpenCV: it stores the position of the interest region center within the image, the scale on which the feature has been detected, and the radius of the interest region.

Feature detection is separated into two processes: feature extraction and calculating descriptors on the set of already extracted features. Each of this parts provides its own interface, namely `p_vpFeatureExtraction` and `p_vpDescriptorGenerator`. The inheritance diagrams for these interfaces can be seen in the Figure 3.1, where it can be noted that wrappers for the corresponding OpenCV functionalities have been implemented, as well as dummy concrete classes for each of the interfaces. The dummy classes help with testing the interaction of the components, while, addition-

[1] `http://opencv.willowgarage.com/wiki/`

[2] `http://www.irisa.fr/lagadic/visp/visp.html`



(a) (b)

Figure 3.1: Inheritance diagrams for the interfaces used in feature detection procedures. The interface used for feature extraction is shown in (a), while (b) is showing an interface used for specifying the way of forming the descriptor vectors.

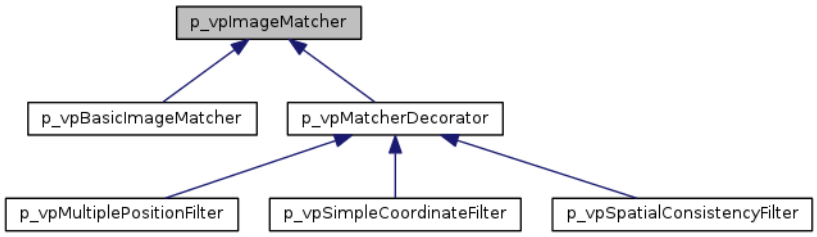
ally, `p_vpDummyExtractor` can also be used to combine the features extracted externally with the rest of the implemented system. An additional concrete class for feature extraction, `p_vpMultipleExtractor`, implements the *composite design pattern* to allow unified work with features extracted by different procedures. An interface `p_vpFeatureDetector` provides methods that perform the complete feature detection process. The class `p_vpCompositeDetector` provides implementations of these functionalities in case stand alone feature detection. This class implements the *strategy design pattern*, allowing for a choice of feature extraction method and feature descriptors. Accessing the features of an image used as a query in the content-based image retrieval process should be done through an instance of `p_vpTreeDetector` class, to avoid repeating the feature extraction process and calculating the feature descriptors. This class does not provide a public constructor, and an instance can only be obtained from the content-based image retrieval module. It is also possible to access an instance of `p_vpTreeDetector` for all the images stored in the visual memory, but the feature descriptors are coarse representations of the real descriptors as they are the descriptors of the corresponding visual words.

3.2. Modules used in content-based image retrieval

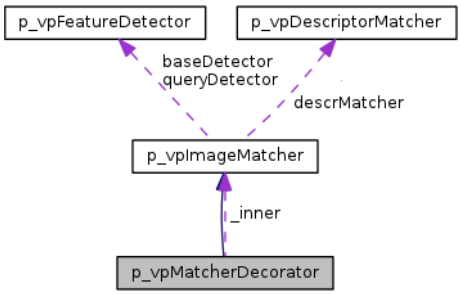
As explained in Section 2.2, the main structure used for content-based image retrieval is the vocabulary tree. To construct and manipulate the vocabulary tree structure, two classes are implemented: `p_vpImageSearchNode` and `p_vpContentImageSearcher`. The class `p_vpImageSearchNode` provides low-level functionalities to manipulate the tree, such as calculating tree weights or accessing the visual words from their integer id-numbers. The `p_vpContentImageSearcher` class works with those methods to provide complete functionalities for content-based image retrieval: filling the visual memory, vocabulary tree construction and querying the visual memory.

3.3. Modules for visual feature matching

The module for visual feature matching is also implemented as a strategy design pattern through the interface `p_vpImageMatcher`. It allows for a choice of different feature extraction methods for the each component the image pair it operates on, as well as a choice between different base matching methods. Base matching methods all implement a common interface `p_vpDescriptorMatcher` and need to be initialized with already extracted features paired with their descriptor vectors for both images. The interface for image matching also implements the *decorator design pattern*, where the matching basis can be obtained by instantiating a `p_vpBasicImageMatcher` with appropriate feature detectors and a base matching method. The concrete matching filters are interfaced by `p_vpMatchingDecorator`, allowing them to work with matched feature pairs obtained from the base methods. Complete inheritance and collaboration diagrams for the matching module are shown in the Figure 3.2.



(a)



(b)

Figure 3.2: Inheritance and collaboration diagrams for the matching module. The inheritance diagram is shown in (a) and the collaboration diagram in (b). The use of the *decorator design pattern* is visible from the diagrams.

4. Testing and results

The testing has been done on video sequences acquired from an electric car-like robot CyCab. Various sequences of different locations were used, as well as multiple sequences taken along the same path at different times and lighting conditions. For parts of the testing, only key images extracted from sequences are used, extracted with the methods implemented in [15] and [14]. Detailed methods for testing and their tabular and graphical representations are described in the Sections that follow, separately for the localization within the visual memory and for visual features matching.

4.1. Localization of the robot within the visual memory

The localization within the robots visual memory was tested on two pairs of different video sequences take along the same path (c.f. Figure 4.2(a)). For each testing experiment, query images were randomly selected from one of the sequences while the visual memory contained only key images from the other sequence (key images were selected with the process described in [14]). The branching factor and the depth of the vocabulary tree are set to $k = 6, L = 10$ for both experiments, as those were the parameters that produced the best results in [9].

For the first experiment, the outdoor video sequences were taken under similar lighting conditions (similar weather and illumination) and in a small time gap (no major changes to the environment). Only the two highest ranking answers are considered. One of the two topologically closest key images is ranked highest in the 113 out of 120 query attempts, while the key image with an offset of 4 or less was not found only in 2 of those query attempts. Tabular results presented in 4.1 show the distance of the highest ranked and best of two highest ranked answers to the actual closest key image. The distance of 1 means that one of the two best matches is found, while the answer distance of $5+$ is considered an unsuccessful query. Medians and distances of the successful queries as well as the percentage of the successful queries are also included.

Table 4.1: Distance information for the first experiment (learning and testing on sequences with similar lighting conditions), taking in the account the highest ranking and the best of two highest ranking responses. The closest possible distance (correct answer) equals 1. Successful queries are considered those where the answer is found among the two highest ranking answers, with a distance equal to or less then 2. Parameters $k = 6, L = 10$, and the vocabulary tree was built using all of the 42 images from the visual memory. The testing was done on 120 randomly chosen images.

answers considered	median	average	successful (%)
1	1	1.07	98.33
best of 2	1	1.05	100.00

The second experiment was done on video sequences with different illumination and with environment changes. The illumination on the sequence from which the visual memory images were selected was such that images were of good contrast, while the strong illumination in the testing sequence produced images with very bright and dark areas with almost no interest regions in which the features could be selected (c.f. Figure 4.1). The map of the visual memory for this experiment is shown on Figure 4.2(a), while the video sequence used for the testing was taken only in the area marked blue on Figure 4.2(b). The area marked red on the Figure 4.2(b) is the area where the images were taken from the shadow, while strongly illuminated parts are visible. It is visible on the image that the red area covers a part of the recorded path with a high density of key images, while the testing results indicate that it corresponds to most of the unsuccessful attempts in localization in this experiment. The tabular representation in Table 4.2 offers the same data as the Table 4.1 for the first experiment, except that the data is offered for up to 5 of the highest ranking answers considered.

These experiments lead to conclusion that the implemented methods work well on sequences taken under the same illumination conditions as well as when the illumination is different but still allows for clear detection of the features. Most of unsuccessful queries came from parts of the sequences where dark and bright areas in the image were almost homogeneous because of the poor illumination and the number of extracted features in those areas are low. Most of the unsuccessful queries in the first experiment and a smaller portion of them in the second experiment came from the parts of the path where homogeneous surfaces, such as walls and asphalt, with a low number of extracted features, make a significant portion of the image area.

In Table 4.3, the execution times for building the vocabulary tree and for querying

Table 4.2: Distance information for the second experiment, taking in the account up to the 5 highest ranking responses. Parameters $k = 6, L = 10$, and the visual memory contains 102 key images. The testing was done on 72 randomly chosen images

answers considered	median	average	successful (%)
Images used to construct the vocabulary tree:			
	10		
1	1	1.72	50.00
best of 2	1	1.57	61.11
best of 3	1	1.50	63.89
best of 4	1	1.50	69.44
best of 5	1	1.44	86.11
Images used to construct the vocabulary tree:			
	25		
1	2	2.24	56.94
best of 1	1	1.72	65.28
best of 3	1	1.68	65.28
best of 4	1	1.36	65.28
best of 5	1	1.37	70.83
Images used to construct the vocabulary tree:			
	50		
1	2	2.28	40.28
best of 1	2	2.07	48.61
best of 3	2	1.93	55.56
best of 4	1	1.74	58.33
best of 5	1	1.73	62.50

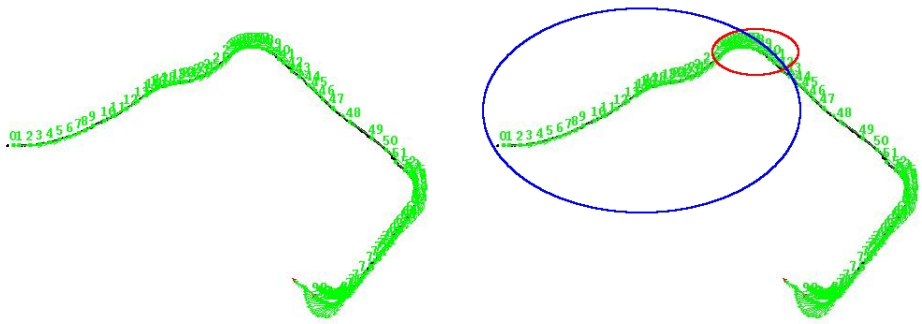
Table 4.3: Average execution times for building the vocabulary tree and querying the database.

# of images used in vocabulary tree construction	construction time (min)	query time (s)
10	7	1.45
25	28	1.45
50	101	1.62

such tree are presented for a varying number of images used in the vocabulary tree construction. It is visible that the time required to get the response to a query does

(a) (b)

Figure 4.1: A pair of images used for training and testing in the second experiment. The image (a) is an example of a visual memory image, while the image (b) is an image of the same locations from the training sequence.



(a) (b)

Figure 4.2: Figure (a) represents topological map of the visual memory used in the second experiment. Figure (b) indicates the shadowed areas producing low contrast input images (marked red) and the part of the path used for testing (marked blue).

not depend much on the number of images used for vocabulary tree construction. The measured times are commensurable with the times presented in [9]. Query times measured for the implemented procedures are about 0.5 seconds larger than the ones in [9] because both MSER and DoG features are extracted from the images, in contrast to only MSER extraction in [9]. It should be noted that the building of the vocabulary tree includes feature detection for all the images contained in the visual memory and query features are extracted before performing a query on the vocabulary tree.

Table 4.4: Average accuracy of matching with different filter combinations for $k = 0.8$

	DoG (%)	MSER (%)
base method	88.96	79.33
multiple position filter	87.90	77.86
multiple position + spatial filter	97.05	89.81
spatial + multiple position filter	96.93	90.71

Table 4.5: Average size of the accepted matches set after applying filtering methods to potential matches after brute force matching with $k = 0.8$

	DoG	MSER
reference set size: *average number of matches (brute force $k = 0.6$ + multiple position filter)* (#)	212.4	36.0
features remaining (%)		
multiple position + spatial filter	118.1	137.4
spatial + multiple position filter	119.7	136.9
multiple position + coordinate (100px) + spatial filter	122.6	154.5

4.2. Performance of the matching process

To test the accuracy of matching process on the visual features of an image pair, the matches for the desired image pair must be annotated by hand. Because the lengthiness of the testing procedure, the image pairs that were used to obtain accuracy results have been selected by hand from available key image databases containing the potential visual memory of the robot. Experiments were done with DoG and MSER feature detection algorithms on consecutive key images. Accuracy testing for the DoG features was done on 5 different consecutive key images from 3 different sequences while for MSER features testing an additional image pair was included in the results for the total of 2 image pairs per sequence. The key images were selected from the sequences by the process implemented in [15]. Choice of images aimed to include the most representative image pairs to obtain the results most relevant for the broad range of possible cases.

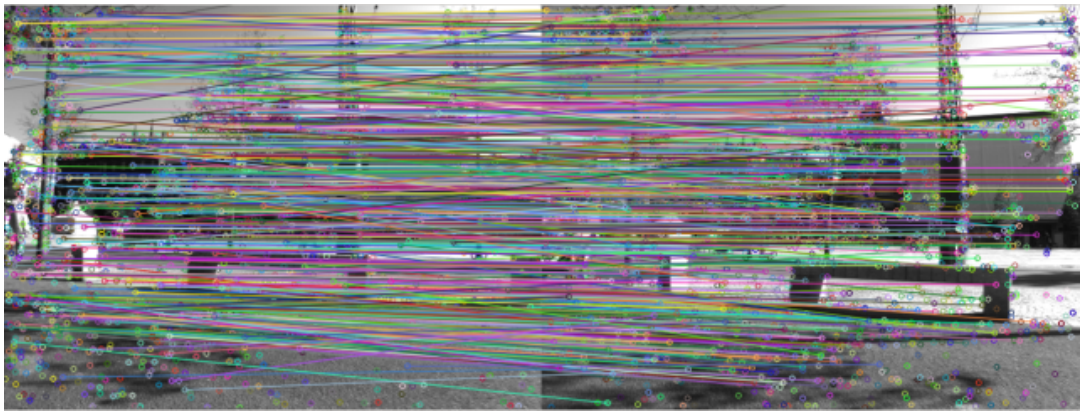
The Table 4.4 shows the percentage of correct matches out of all accepted matches for different combinations of filters. Although the Coordinate filter with 100px win-

dow size shows results commensurable with the best results presented in Table 4.4, it was not included in the results listings because of its instability. Conclusions about the instability of the Coordinate filter can be drawn from examining the percentage of the correct matches in the accepted matches set, as well as its size for 50px setting. While the filter performs very good for image pairs with low positional offsets (typically, on a straight part of the path), the number of accepted matches is reduced to as little as $\frac{1}{50}$ of the number of accepted matches before applying all the filtering methods. This happens because the predetermined window size for the Coordinate filter is not the most suitable one for all of the image pairs. An example of such image pair with a significant positional and rotational offset, and the behavior of the Coordinate filters with 100px and 50px window size are shown in Figures 4.3 and 4.4. Still, the testing results obtained when the Coordinate filtering is applied with approximately appropriate window size are promising. During the visual memory preparation done in the offline phase, described in [14], information about the average distances in the matched feature coordinates could be obtained. If this, or some similar measure is available, it could be used to precalculate a more appropriate window size for the Coordinate filter, thus increasing the accuracy of the matching obtained while keeping the desirable property of higher match acceptance rate.

The "marriage" criterion was not used as well, since the size of potential matches set was initially to small, and the accepted match sets did not have a satisfactory size despite a high percentage of correct matches. The acceptance rate for the Spatial filter is set to 50% of all pondered votes. The size of the Multiple position filter window is set to 11 pixels, since that is the size of the tracking window that is to be used on the features during the navigation process described in [14].

It should be noted that Multiple position filtering should always be applied, as multiple matches at the same image coordinates need to be eliminated. Experimentally, it is shown that the Multiple position filter should be applied to the matches set after applying the Spatial filtering. Another relevant factor for determining the quality of results is the number of matches in the end of the process. The distance ratio k for the brute force matching was chosen with a compromise between the percentage of the accurate matches and the size of the accepted matches set in mind. The Figure 4.5 displays the correlation between the accepted set size and the percentage of the correct matches. Based on this data, the distance ratio was set to $k = 0.8$, which coincides with the distance ratio suggested in [6]. Table 4.5 summarizes the changes in size of the accepted matches set for the filter combinations used in the final implementation. The accepted matches set size taken as a reference was the size of the matches set

(a) Coordinate filter with window size 50×50 pixels



(b) Coordinate filter with window size 100×100 pixels

Figure 4.3: Pair of images on which where Coordinate filter with a smaller window performs *worse* than if a larger window is used. Images filtered with Multiple position filter and Coordinate filter are shown before applying the Spatial filtering. It can be observed that the number of matches on the image pair (a) is greatly reduced with respect to the total number of features, while the accuracy is low. The number of matches and their accuracy on the second image pair shown in (b), however, is visibly higher. These results were obtained with $k = 1.0$, but the Coordinate filter performs in the same manner for all image pairs with a large positional and rotational offset with all parameter combinations for the matching basis.

acquired with brute force matching without the "marriage" criterion and distance ratio limit k set to 0.6, with only Multiple position filter applied. The average number of accepted matches for the final choice of parameters is around 50 per image pair when using MSER features, and around 250 when using DoG features.

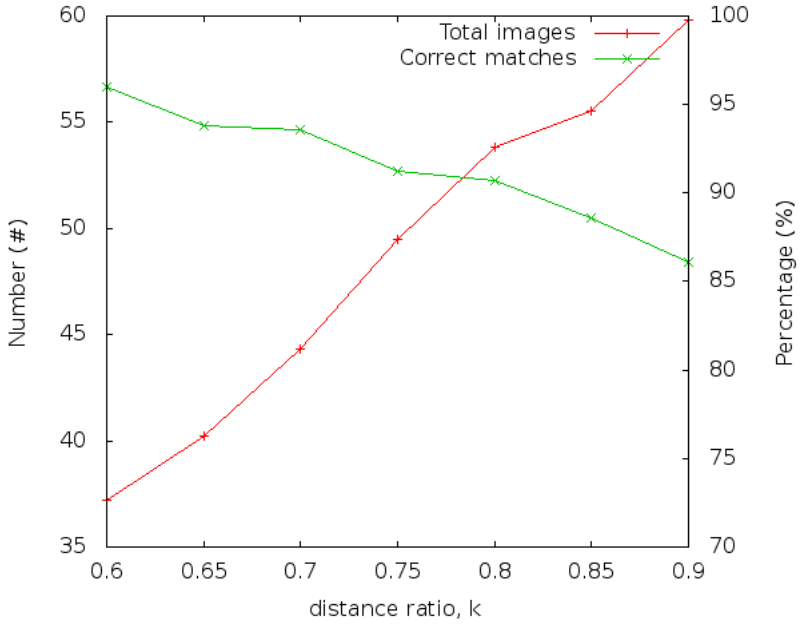
The speed of the optimal combination of filters (Spatial filter followed by the Multiple position filter) has been tested for the DoG and the MSER feature matching done simultaneously with the SIFT feature descriptors. Tests were done on 4 different image pairs, and for each image pair the process was run 50, averaging the execution

(a) Coordinate filter with window size 50×50 pixels

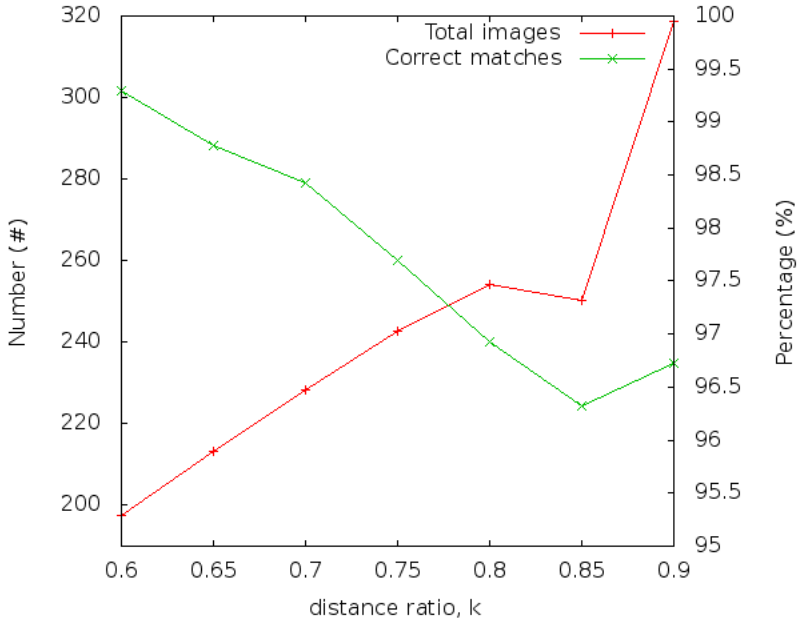
(b) Coordinate filter with window size 100×100 pixels

Figure 4.4: Pair of images on which where Coordinate filter with a smaller window performs *better* than if a larger window is used. Images filtered with Multiple position filter and Coordinate filter are shown before applying the Spatial filtering. While the number of accepted matches after applying both filters (a) and (b) is fairly large, the matched features are visibly more accurate on (a) than on (b). The best size of the Coordinate filter window to achieve the best percentage of correct accepted matches depends on the positional and rotational offset between the images in the pair.

times for final calculations. The average execution time was 2.26 seconds. Most time is spent on the feature detection process, while the matching procedure is much faster.



(a) MSER features used



(b) DoG features used

Figure 4.5: Size of the accepted matches set and the percentage of the correct matches accepted for different distance ratio k for the brute force matching

5. Conclusion

From the presented results, it can be concluded that though the developed modules are not yet good enough to use without human intervention for the autonomous navigation, this first iteration of implementing these functionalities shows potential in exploring the implemented approach. The modules developed for this thesis were integrated with the visual navigation modules developed in the complementary thesis [14] to be used for the autonomous navigation of INRIA CyCab. As both content-based image retrieval and the matching process rely on visual features detection process designed primarily for structured images, the localization and the visual feature matching tend to give unsatisfactory results in the environment containing large homogeneous and untextured surfaces (such as walls).

In spite of this, developed functionalities can be considered a basis for future development of a more robust localization modules. Possible improvements to localization could come from utilizing image description techniques better suited to represent the contents of images containing weak textured objects. The other improvement possibility to the visual matching approach would be to obtain some approximation of maximum/median visual feature distance for an image pair before starting the visual matching, to utilize the implemented Coordinate filter with appropriate parameters. Obtaining a larger initial set of tentative correspondances without increasing the portion of false matches in the set is expected to accept more features in the final set of correspondances.

BIBLIOGRAPHY

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[2] A.L. Dahl, H. Aanæs, and K.S. Pedersen. Finding the Best Feature Detector-Descriptor combination. In *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011*, pages 318–325. IEEE, 2011.

[3] F. Fraundorfer, C. Engels, and D. Nistér. Topological mapping, localization and navigation using image collections. In *International Conference on Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ*, pages 3872–3877. IEEE, 2007.

[4] E. Freeman, E. Freeman, K. Sierra, and B. Bates. *Head First Design Patterns*. O'Reilly Media, 2004.

[5] D.G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999*, volume 2, pages 1150–1157. IEEE, 1999.

[6] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[7] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004.

[8] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *The Proceedings of the Eighth IEEE International Conference on Computer Vision, 2001*, volume 1, pages 525–531. IEEE, 2001.

[9] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In

IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006, volume 2, pages 2161–2168. IEEE, 2006.

[10] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004*, volume 1, pages I–652. IEEE.

[11] S.E. Robertson and K.S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[12] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[13] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *The Proceedings of the Ninth IEEE International Conference on Computer Vision, 2003*, pages 1470–1477. IEEE, 2003.

[14] Ante Trbojević. Robust point tracking for visual navigation. Master thesis, Faculty of Electrical Engeneering and Computing, University of Zagreb, 2012.

[15] S. Šegvić, A. Remazeilles, A. Diosi, and F. Chaumette. A mapping and localization framework for scalable appearance-based navigation. *Computer Vision and Image Understanding*, 113(2):172–187, 2009.

[16] Ozana Živković. Izlučivanje istaknutih točaka slike u ekstremima konvolucijskog odziva razlike Gaussovih funkcija. Master thesis, Faculty of Electrical Engeneering and Computing, University of Zagreb, 2011.

[17] A. Witkin. Scale-space filtering: A new approach to multi-scale description. In *Proceedings of the 1984 IEEE International Conference on Acoustics, Speech, and Signal Processing, 1984*, volume 9, pages 150–153. IEEE, 1984.