BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

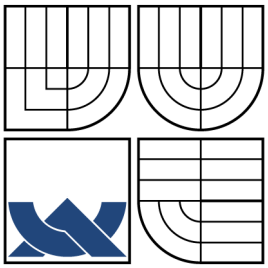
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FAST FEATURE MATCHING FOR SIMULTANEOUS LOCALIZATION AND MAPPING

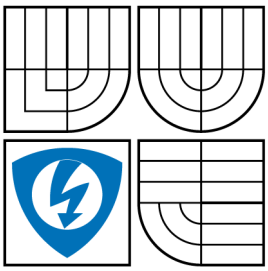
BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR ONDŘEJ MIKŠÍK
AUTOR PRÁCE

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

FAST FEATURE MATCHING FOR SIMULTANEOUS LOCALIZATION AND MAPPING

RYCHLÉ VYHLEDÁVÁNÍ OBRAZOVÝCH VLASTNOSTÍ PRO SOUČASNOU LOKALIZACI A MAPOVÁNÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ MIKŠÍK
AUTHOR

VEDOUCÍ PRÁCE — KRYSTIAN MIKOLAJCZYK, PhD., MSc.
SUPERVISOR

BRNO 2010

ZDE VLOŽIT LIST ZADÁNÍ

Z důvodu správného číslování stránek

ABSTRACT

The thesis deals with the fast feature matching for simultaneous localization and mapping. A brief description of local features invariant to scale, rotation, translation and affine transformations, their detectors and descriptors are included. In general, real–time response for matching is crucial for various computer vision applications (SLAM, object retrieval, wide–robust baseline stereo, tracking, . . .). We solve the problem of sub–linear search complexity by multiple randomised KD–trees. In addition, we propose a novel way of splitting dataset into the multiple trees. Moreover, a new evaluation package for general use (KD–trees, BBD–trees, k–means trees) was developed.

KEYWORDS

Local Invariant Features, Feature Detectors and Descriptors, KD–trees, BBD–trees, K–means trees, Simultaneous localization and mapping (SLAM), Performance Evaluation

ABSTRAKT

Bakalářská práce se zabývá rychlým vyhledáváním lokálních obrazových vlastností v rozsáhlých databázích pro simultánní lokalizaci a mapování prostředí. Součástí práce je krátký přehled detektorů a deskriptorů invariantních vůči rotaci, translaci, změně měřítka a affinitě. Pro řadu aplikací z oblasti počítačového vidění (SLAM, object retrieval, wide–robust baseline stereo, tracking, . . .) je odezva reálném čase naprosto nezbytná. Jako řešení sublineární časové náročnosti vyhledávání v databázích bylo navrženo použití vícenásobných náhodně generovaných KD–stromů. Dále je předkládán nový způsob dělení dat do vícenásobných KD–stromů. Navíc byl navržen nový, obecně použitelný vyhodnocovací software (podporovány jsou KD–stromy, BBD-stromy a k-means stromy.)

KLÍČOVÁ SLOVA

Lokální invariantní vlastnosti, Detektory, Deskriptory, KD stromy, BBD stromy, K–means stromy, Simultánní Lokalizace A Mapování (SLAM), Vyhodnocení výkonosti

MIKŠÍK, Ondřej *Fast Feature Matching for Simultaneous Localization and Mapping*: bachelor's thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of control and instrumentation, 2010. 91 p. Supervised by Krystian Mikolajczyk, PhD., MSc.

DECLARATION

I declare that I have elaborated my bachelor's thesis on the theme of "Fast Feature Matching for Simultaneous Localization and Mapping" independently, under the supervision of the bachelor's thesis supervisor and with the use of technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the bachelor's thesis I furthermore declare that, concerning the creation of this bachelor's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal copyright and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Vol., including the possible consequences of criminal law resulted from Regulation § 152 of Criminal Act No 140/1961 Vol.

Brno ....

(author's signature)

ACKNOWLEDGMENTS

I am very grateful to my supervisor Dr. Krystian Mikolajczyk for his advices, feedback and leadership. I would also like to thank my consultant Dr. Luděk Žalud as well as to all the other people who helped me with my internship in the Centre for Vision, Speech and Signal Processing at the University of Surrey in Guildford.

At last but not least, I am very grateful to all my family and my girlfriend Markéta for their support.

Brno ....

(author's signature)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The mobile robotics community made an enormous effort to build robots with elements of autonomous behavior during the past two decades. The field of possible objectives is various from maze solving robots, letter–carrier robots to fully autonomous vehicles which can operate in an unknown environment.

Many successful projects has proven in the past that the idea of fully autonomous vehicle is not utopia. The most famous are probably projects Argo [13] or No Hands Across America [52]. The essential boom started with the three challenges organized by Defense Advanced Research Projects Agency (DARPA). In the first DARPA Grand Challenge in 2004 (Mojave Desert), the best robot which was built by Carnegie Mellon University (CMU) made only an 11.78km length run, nevertheless it was a great achievement, because they proved that it is possible to do it.

(a) Stanley, Stanford Racing, courtesy of [62]

(b) Boss, Tartan Racing, courtesy of [64]

Fig. 1.1: DARPA challenges winning robots

One year later the route was even more difficult, but all robots exceeded the largest run from the previous competition. The winner was Stanley created by Stanford Racing Team. In 2007, DARPA Urban Challenge took place in Victorville. The competition was more challenging because vehicles had to follow all Californian traffic regulations. The winner Boss was made by Tartan Racing (CMU).

One might seem, that the DARPA competitions were disposable, however technologies which were used can be applied in various fields like army autonomous vehicles preventing human casualties or space exploration robots. Afterwards, these technologies will enforce in our everyday life like safety systems in cars, mapping of abandoned mines, etc.

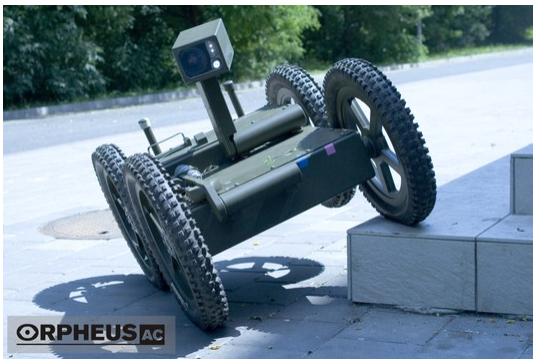
1.1 Problem Formulation

The general problem of mobile robot navigation can be summarized by three essential questions [12]:

- Where am I?
- Where am I going?
- How should I get there?

Answers for these questions are necessary for both the teleoperated as well as fully autonomous robots, however fully autonomous robots are able to solve these problems themselves. The most interesting case is when robots are able to operate in an unknown environment. Then, arises one of the most fundamental problem of robotics – the Simultaneous Localization And Mapping (SLAM). The SLAM[1] is not an unique algorithm, rather it is a set of different approaches solving the chicken–and–egg problem of incremental acquiring of a consistent map of unknown environment while its relative location to this map is simultaneously determined [18, 6, 65].

(a) Mars rover, JPL Caltech, courtesy of [36]



(b) Orpheus–AC, LTR BUT, courtesy of [50]

Fig. 1.2: Various robotic platforms

The SLAM exists and is implemented in a number of different forms from indoor robots to outdoor, underwater and airborne systems. Let us mention historically the earliest probabilistic approach which influenced many recent algorithms – the Extended Kalman Filter (EKF) SLAM or one of the most actual progressive approach based on particle filter – the FastSLAM (2.0) [45].

[1] Other widely used term is Concurrent Mapping and Localization (CML)

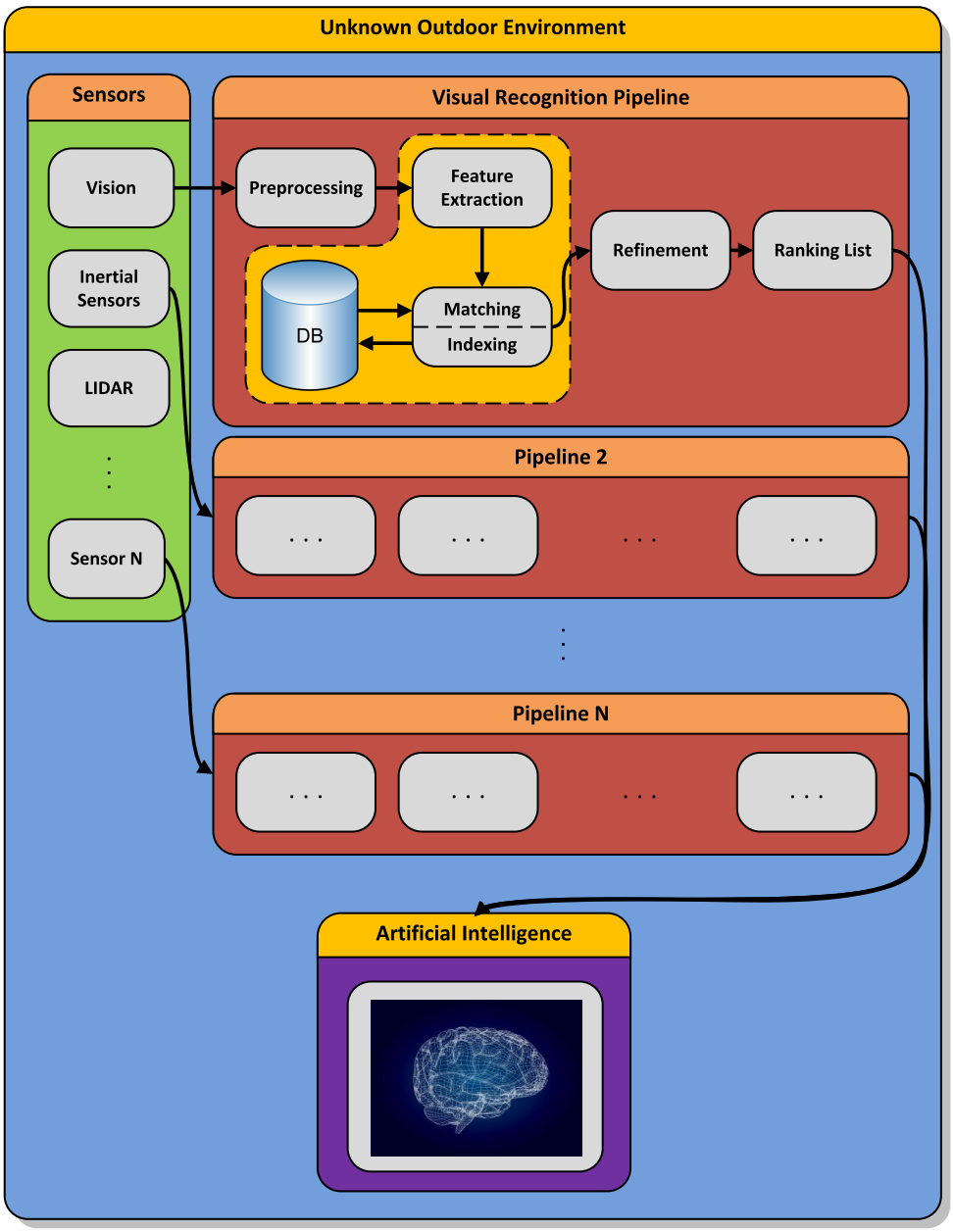


Fig. 1.3: Overview of SLAM–like systems

The figure 1.3 describes one possible block scheme of SLAM. We aims at image processing pipeline which may contain preprocessing block (equalization, denoising, subsampling, ...), local invariant feature extraction and description, indexing features to large database and matching of features, refinement of found matches, and creation of ranking list which is used as an output. The most time consuming steps are feature detection and matching. Several efficient implementations of feature detectors and descriptors exist (SIFT, SURF, FAST, ...), so we can assume that this step is partially solved, however feature matching in a large database is a serious bottleneck problem when the real–time response is required.

1.2 Challenges

SLAM algorithms use different types of sensors. Many of them are designed to get data from ultrasound sensors, radar or recently laser range finders. Another challenging goal may be to develop a SLAM which will use information obtained by camera. SLAM based on camera should be useful, because it is a cheap sensor and besides, robots usually use cameras for different tasks. Another benefits may be ability not just to detect any objects and landmarks, but rather recognize and distinguish them and track moving objects (not to consider them just as a noise). It is possible to classify those landmarks and objects, therefore smarter path planning could be developed. Even humans receive most information by vision.

However, computer vision is not easy, due to the several reasons. Even if we passover all troubles connected with image acquisition process, there still exists many reasons why it is quite difficult to retrieve objects. Let us summarize some of them:

- **Viewpoint changes** – objects are seen under different viewpoint. It is absolutely necessary to deal with rotation, translation, scale and affine invariant.
- **Partial occlusion by other objects** – object are usually occluded by different objects like people, cars, trees, etc.
- **Different illumination conditions** – objects or their fragments are seen under different illumination throughout the day, even at the same time, due to shadows or oversaturated parts of a scene.
- **Data amount** – it is necessary to process a huge amount of data

Many of these problems are solvable by a progressive idea of local invariant features. Various feature detectors and descriptors have been proposed. This approach may be roughly described in a few steps. After image preprocessing, the local invariant features are detected, described and stored in a database. Then, extracted features from the query image are matched to the database and different classifiers are used to recognize the objects.

All of these steps may be time consuming. During the past decade, several efficient implementations of the local image detectors and descriptors have been proposed. However, sufficient descriptors are highly dimensional and the databases usually contain hundreds of samples. Consequently, a brute–force (linear) search in the database may take much time, up to few hours or days. In particular, we are interested

(a) Viewpoint change – Duke of Kent Building, University of Surrey

(b) Occlusion by other objects – York Minster

(c) Different illuminations – Big Ben

Fig. 1.4: Challenges in computer vision, courtesy of [51]

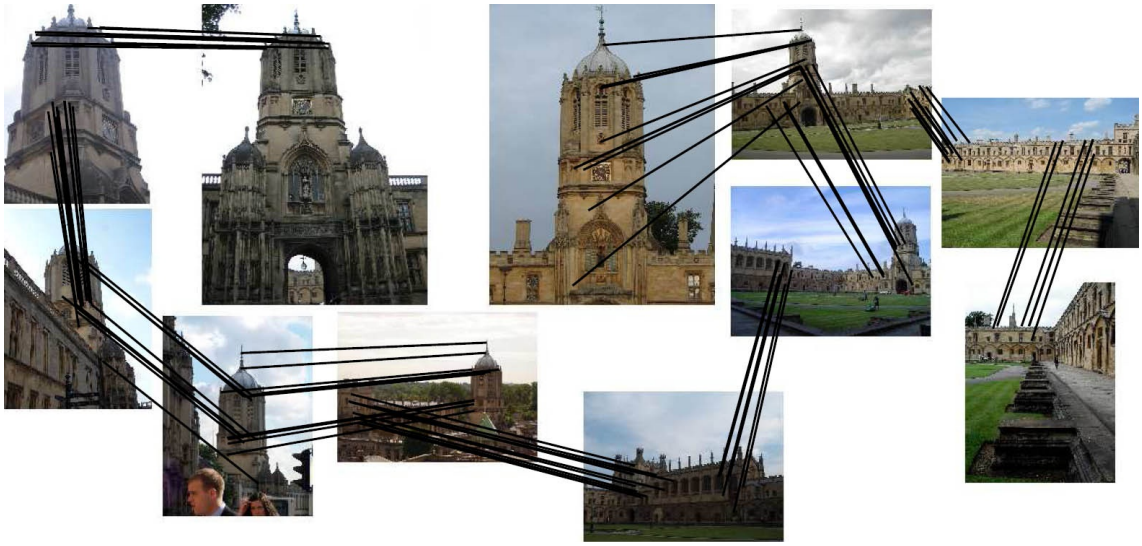
in on–line SLAM, therefore the fast feature matching very close or equal to the real–time is crucial.

1.3 Applications

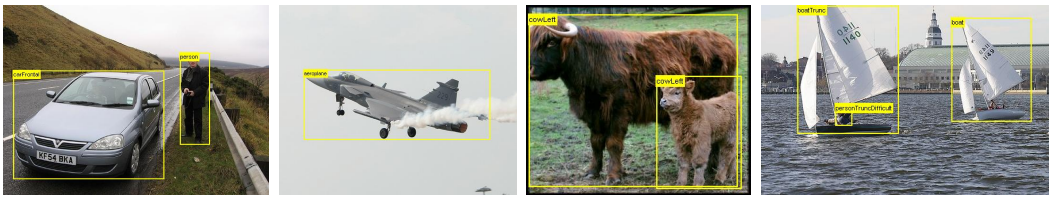
Our motivation for fast feature matching is clear as we have described above. In addition, fast feature matching allows the following applications (and even more):

- **Image based recognition** – it is not necessary to assign images or videos by text tags, it is possible to search similar to the query one.
- **Panorama stitching** – extracted features are used to find correspondences to create fully automated panorama.
- **3D scene modeling** – accurate feature localization is used to build a sparse 3D model of the viewed scene.
- **Detection of unsafe images** – it is possible to detect unsafe images or videos for kids (murders, . . .) and prevent watching.

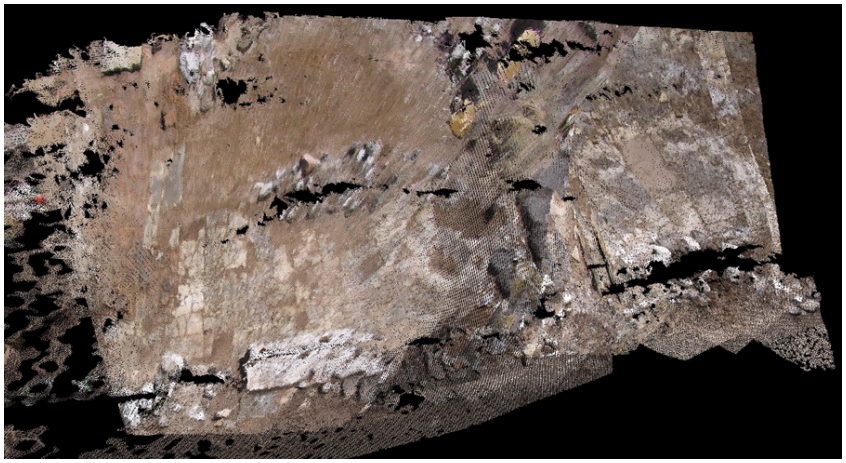
(a) Automatic panorama stitching (80 input images), courtesy of [14]



(b) Visualization of a part of spatially related images, courtesy of [15]



(c) PASCAL Visual Object Classes Challenge 2007, courtesy of [19]

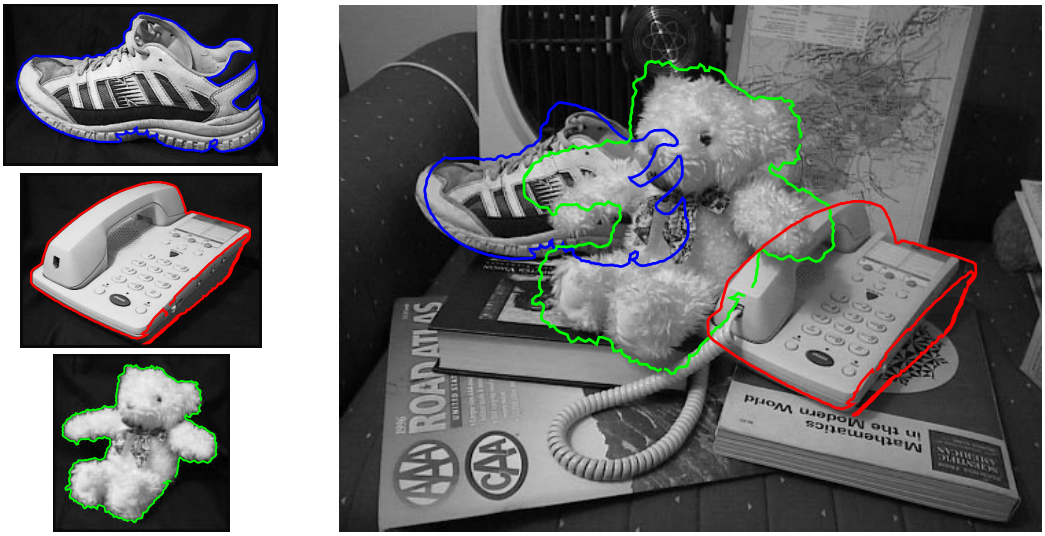


(d) 3D reconstruction of scene (35 views), courtesy of [7]

Fig. 1.5: Applications of local invariant features

1.4 Thesis Structure

The rest of the thesis is organized as follows: in chapter 2, state of the art on local invariant feature detectors, descriptors and efficient data structures is briefly discussed. Then, we continue with description of used data sets, standard evaluation criterion which are used for the experiments and with an overview of our performance evaluation framework in chapter 3. Chapter 4 deals with results obtained from various experiments. Finally, all conclusions are discussed and summarized in chapter 5. Moreover, appendix A describes usage of each part of our performance evaluation framework, appendix B describes local feature detectors more in detail and appendix C summarize contents of enclosed DVD.



(a) (b)

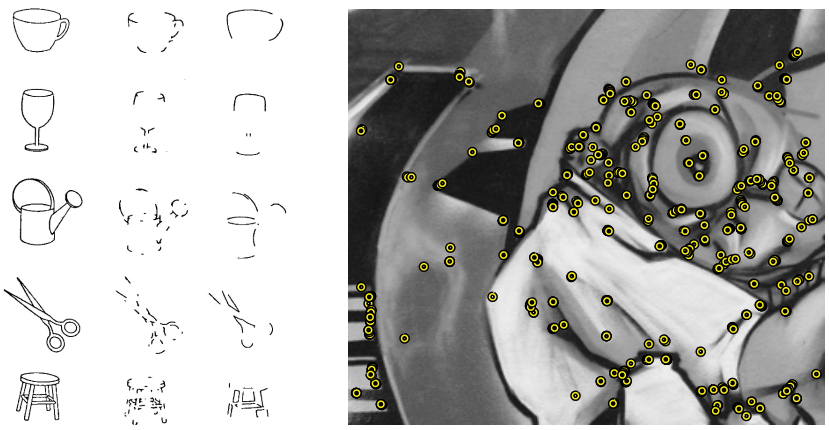
Fig. 1.6: 3D object recognition with occlusion from local scale–invariant features, courtesy of [32]

2 STATE OF THE ART

In this chapter we discuss previous work. We begin with description of properties of local invariant features, their detectors and descriptors. In section 2.2 we present the state of the art on a local feature detectors. Then, in section 2.3 main modern local feature descriptors are discussed. Finally in section 2.4 we describe different data structures for sublinear search time. More information about feature detectors may be found in appendix B.

2.1 Overview of Local Invariant Features

A local feature[1] can be point, edgel (edge pixel) or image patch which differs from its neighborhood in their properties such as intensity, color, or texture, but features are not necessarily localized exactly on this change [42, 66]. It is not just a method to select interesting points or to speed up the analysis but rather it is a new representation of the image because local features are distinctive, robust to occlusion and clutter and do not need the segmentation. Recently, many approaches are focused on making these features invariant. Usually, local features are detected through some measurements taken from sliding windows (regions) and converted into the new image representation with descriptors. Next, the features are usually stored in a database and used in a wide range of applications such as wide baseline matching [37], building panoramas [14], object or texture recognition [33, 29], image retrieval [55], robot localization [56], tracking [25], etc.



(a) (b)

Fig. 2.1: Importance of corners and T–junctions for perception of objects (a), courtesy of [11] and local features extracted by Harris detector (b), courtesy of [66]

[1] Other widely used terms are interest point, region of interest, ...

Local features usually have some spatial extend over their support regions, this can be any subset of image. Some uninteresting (mainly homogeneous) areas of an image may be uncovered, other regions do not have to correspond to object boundaries, there might be an overlap of multiple support regions.

Local invariant[2] features may have different properties depending on application. We will summarize some general properties which should be common for ideal features [66]:

- **Repeatability** – high percentage of the features should be detected and matched on the scene part visible in two different images.
- **Distinctiveness** – each feature should be matched to a large database of objects.
- **Locality** – the detected features should be local, so they should be robust to occlusion and clutter, different viewpoint and/or illumination, blur, discretization effect, etc.
- **Accuracy** – the features should be accurately localized in the image and should respect scale and possibly shape.
- **Quantity** – Even for small objects should be detected many features. Number of the features should be adaptable by a simple threshold, because optimal number of features depends on application.
- **Efficiency** – detection of features should be close to the real–time.

It has been such a long time since it has been understood that intersections of straight lines and straight corners are strong indications of man made structures [5]. Many papers, journals and books were written about local feature detection over more than the last fifty years. There exists many directions of research, for example corner detectors, analyze of image intensities, other methods are based on derivatives or color information. Another brach is inspired by the human brain. Recently, many approaches deal with geometric transformations like scale and/or affine invariant detection. We focus on the main modern approaches as Harris and Hessian affine regions [41], Scale Invariant Feature Transformation (SIFT) [33], Speeded Up Robust Features (SURF) [8] or Maximally Stable Extremal Regions (MSER) [37] feature detectors and SIFT [33], GLOH [42], PCA–SIFT [26] and Moment Invariants descriptors [22].

[2]In fact, local features are covariant to some class of transformations, invariant are local photometric descriptors due to the normalization, however the term invariant is widely used.

2.2 Local Feature Detectors

In this section, we summarize various local invariant feature detectors. We focus on so–called corner detectors in section 2.2.1. Next, we briefly discuss blob detectors in section 2.2.2 and region detectors in section 2.2.3. Finally, section 2.2.4 deals with widely used efficient implementations of local feature descriptors.

2.2.1 Corner Detectors

As we have already noted, it has been such a long time since the first publication of results of the observation about importance of structures like corners and T–junctions in computer vision and pattern recognition [5]. The detected points do not necessarily correspond to the corners of real objects. In fact, corner detectors are sensitive even to various types of junctions, highly textured surfaces, etc, however it does not matter – the goal is to have a set of stable and repeatable features [66]. Moravec proposed one of the earliest corner–like local feature detector which was used for navigation of the Stanford Cart [47]. Moravec detector defines the corners as points with a large intensity variations in eight principal directions. Nowadays, the most widely used is Harris detector (other well known approaches are SUSAN [60], Shi & Tomasi [57], ...). Next, we discuss the Harris detector and its scale and affine invariant extensions.

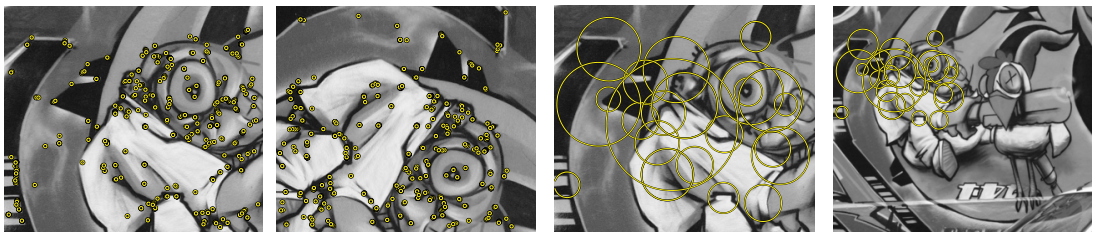
Harris/Plessey Detector

The Moravec detector was improved by Harris and Stephens in 1988 [23]. By comparison with the Moravec detector, the corner score is measured directly instead of using shifted windows and has isotropic response. It is based on eigenvalues of the second moment matrix (auto–correlation matrix), which represents the most principal signal changes in two orthogonal directions around the point – Harris points are detected if both eigenvalues are large [66].

Harris corner detector is invariant under rotation and translation only, many features are localized on edges instead of real corners and sensitive to noise.

Harris–Laplace

Harris–Laplace detector was proposed by Mikolajczyk and Schmid [39] as an scale invariant extension to the Harris corner detector. It is based on multi–scale Harris corner detector and the characteristic scale is selected as was proposed by Lindenberg in 1998 [31]. Firstly, scale–space representation is built with the Harris



(a) (b) (c) (d)

Fig. 2.2: Features extracted by Harris (a), (b) and Harris–Laplace (c), (d) corner detectors, courtesy of [66]

function. Then for each initial point it is checked whether the LoG attains a maximum at the scale of the point.

Harris–Laplace detector provides efficient method extracting rotation, translation and scale invariant local features [41, 66].

Harris–Affine

Harris–Laplace detector fails in the case of significant affine transformations when the scale change is not necessarily the same in every direction. Harris–Affine [40] is an affine invariant extension to Harris–Laplace detector. The main idea is to use a second moment matrix in affine Gaussian scale-space where an elliptical affine regions are used instead of circular region.

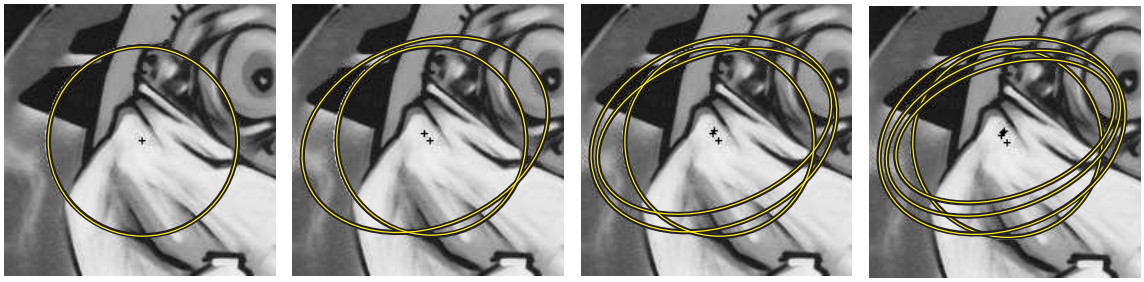


Fig. 2.3: Iterative detection of an affine invariant feature by Harris–Affine detector, courtesy of [66]

Harris–Affine detector is invariant to rotation, translation, scale and affine transformation. The number of detected local invariant features depends on the type of scene and the threshold [41, 66].

2.2.2 Blob Detectors

In this section, we briefly describe fundamentals of blob detectors. We begin with a derivative–based method called Hessian detector 2.2.2. Next, we continue with scale and affine 2.2.2 invariant extensions to this detector.

Hessian Detector

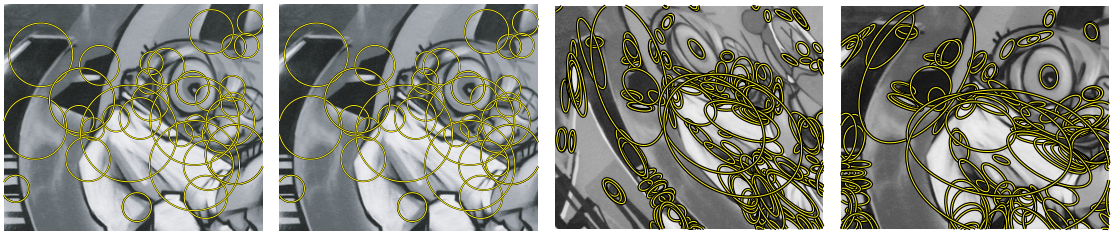
The class of Hessian detectors are based on the Hessian matrix, which is obtained as the second matrix from the Taylor expansion of the image intensity function.

Location and scale for which the trace and the determinant of the Hessian matrix attains the local extremum simultaneously are used to detect more stable maxima, because the trace maximize the curvature and the determinant penalizes small second derivations in only a single direction [66].

Hessian–Laplace/Affine

The concept behind the Hessian–Laplace detector is similar to the Harris-Laplace, however by comparison with Harris–Laplace, the Hessian–Laplace detector is initialized from the determinant of the Hessian matrix, instead of Harris detector. Hessian–Laplace/Affine detectors were also proposed by Mikolajczyk and Schmid [41].

Hessian based detectors are complementary to the Harris detectors, because they are sensitive to the different parts of the image. Scale and affine invariant properties of the detectors are provided in a similar manner – Hessian detector localize blobs in space, Laplacian in scale and iterative approach is used for affine invariance. Hessian based detectors are translation, rotation, scale and affine invariant [42, 66].



(a) (b) (c) (d)

Fig. 2.4: Features extracted by scale invariant Hessian–Laplace blob detector (a, b), and affine invariant features extracted by Hessian Affine detector (c, d), courtesy of [66]

2.2.3 Region Detectors

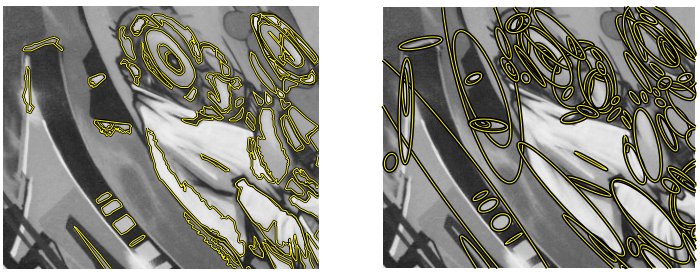
In this section, we summarize detectors which extract image regions. We begin with Intensity–based Regions. Then, we focus on Maximally Stable Extremal Regions. Let us note that the importance of segmentation based methods (superpixels) is growing, but they are not discussed in this thesis.

Intensity–based Regions

Intensity–based regions were proposed by Tuytelaars and Van Gool in 2000 [67]. The main idea of this approach is following: the neighborhood of a point with extrema intensity (detected at multiscale) is searched along each ray of the region and the points for which is reached the maximum are invariant under both affine linear photometric and geometric transformations.

Maximally Stable Extremal Regions

Maximally Stable Extremal Regions (MSER) were proposed by Matas et al. in 2002 [37]. The MSER are obtained by thresholding the image. The set of all connected components, for which the binarization is stable over a large range of thresholds (all intensities within the area are lower or higher than pixels on boundaries), is the set of all MSERs (set of all maximally stable extremal regions).



(a) (b)

Fig. 2.5: Features (regions) extracted by Maximally Stable Extremal Regions detector (a) and fitted ellipses based on the first and second shape moments (b), courtesy of [66]

MSER features are accurately localized, because it is sensitive to region boundaries. The algorithm is very efficient for structured scenes with regions separated by strong intensity changes. The main drawback of MSER is the sensitivity to image blur.

(a) (b)

Fig. 2.6: Estimated epipolar geometry and points associated to the matched regions for robust wide baseline stereo from MSER, courtesy of [37]

2.2.4 Efficient Implementations

Many of previously discussed local invariant feature detectors are computationally expensive, due to the computation of derivatives or second moment matrices. Fortunately, there exists several efficient implementations of local invariant feature detectors, which can be used for real–time applications. In this section, we discuss the most important approaches as Scale Invariant Feature Transformation, Speeded Up Robust Features and Features from Accelerated Segment Test.

Scale Invariant Feature Transformation

Scale Invariant Feature Transformation (SIFT) was proposed by Lowe in 1999 [33, 32, 66]. The detector use well–known approximation of the Laplacian of Gaussian by the Difference of Gaussian. It is based on searching the maxima in scale–space representation of DoG filter response. Firstly, the scale–space representation $L(x, y, \sigma)$ of the image $I(x, y)$ is computed by convolution of the image with a variable–scale Gaussian $G(x, y, \sigma)$

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

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

where $*$ denotes the convolution. Next, the stable keypoint locations in scale space of Difference–of–Gaussians are computed from the difference of two nearby scales separated by a constant multiplicative factor k

$$D(x, y, \sigma) = \Big(G(x, y, k\sigma) - G(x, y, \sigma)\Big) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.3}$$

Local maxima in these steps are located both in space and in scale with non-maxima suppression. After a few steps, the process continues with subsampling of the image

and computing next octave. Additional filtering step may be used for filtering strong responses of the Laplacian on edges.

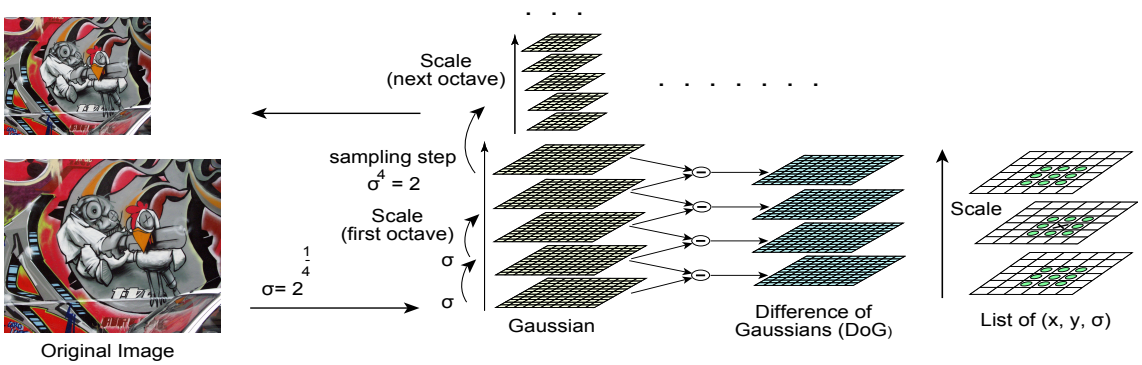


Fig. 2.7: SIFT detector overview, courtesy of [33]

Speeded Up Robust Features

Speeded Up Robust Features (SURF) are based on the idea of integral images, which allow fast computation of Haar wavelets or any other box–type filters:

$$I_{\Sigma}(x,y) = \sum_{i=0}^{i<x} \sum_{j=0}^{j<y} I(i,j) \tag{2.4}$$

SURF detector is based on the Hessian–Laplace detector, but both the location and scale are measured by the determinant of Hessian matrix. The approximated box–type filters are used instead of discretized Gaussian filters which allows fast computation of the second order derivatives due to used integral images [8, 66].

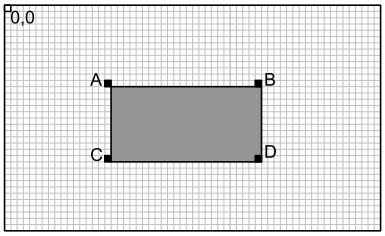


Fig. 2.8: Concept of the integral images allows to count in four additions sum of intensities over any rectangular area independently on its size ($\sum = D - B - C + A$), courtesy of [63]

The SURF detector with box–type filters has comparable results with the discretized Gaussians, however is five times faster than difference–of–gaussians.

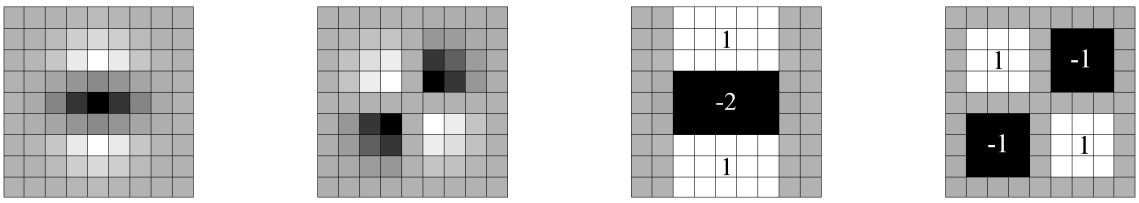


Fig. 2.9: Comparison of the discretized Gaussian second–order partial derivatives and their approximations by SURF (gray pixels denote zero), courtesy of [66]

Features from Accelerated Segment Test

Features from Accelerated Segment Test (FAST) detector is based on the extension of the SUSAN detector [53, 66]. FAST compares intensities of the pixels only on the circle of fixed radius around the point instead of all pixels in the neighborhood. The algorithm begins with the comparing of pixels labeled 1 and 2 with the threshold, 3 and 4 respectively. Then, all other pixels are compared in the same way and the pixels are classified into three subsets. The ID3 algorithm and entropy measurements are used to determine whether the pixel represents corner or not. This process runs iteratively until the entropy of a subset is zero.

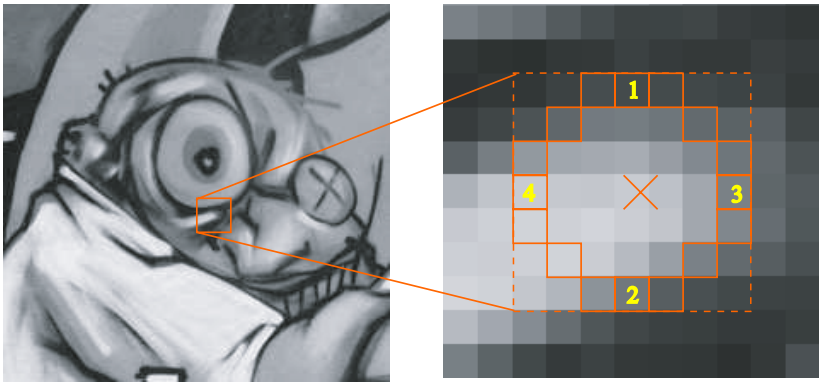


Fig. 2.10: Concept of FAST detector, courtesy of [66]

The FAST detector is very efficient, it can run 30× faster than DoG detector, however it is no invariant to scale. This can be overcome by a multi–scale detector.

2.3 Local Image Descriptors

In this section we briefly summarize various local invariant descriptors. We begin with Distribution Based Descriptors and their representatives in section 2.3.1, then we describe basics of Spatial–Frequency Techniques in section 2.3.2 and Differential Descriptors in section 2.3.3. We conclude with Moment Invariants in section 2.3.4.

2.3.1 Distribution Based Descriptors

Distribution based descriptors are usually based on histograms representing their characteristics. In this section we discuss approach proposed by Lowe which use Scale Invariant Feature Transform (SIFT detector is discussed in section 2.2.4) and its extensions like Gradient Location–Orientation Histogram and PCA–SIFT. Next, we discuss similar descriptor called Shape Context.

SIFT Descriptor

Local invariant SIFT descriptor was introduced as same as the detector by Lowe [33]. Firstly, for all pixels of each image $L(x, y)$ gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are precomputed

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}, \tag{2.5}$$

$$\theta(x,y) = \tan^{-1} \frac{L(x+1,y) - L(x-1,y)}{L(x,y+1) - L(x,y-1)} \tag{2.6}$$

Next, the 36 bins orientation histogram which covers 360 degree range of orientations is created from the gradient orientations in the region around the keypoint point. Each contribution added to the histogram is weighted by a circular Gaussian window with a σ 1.5 times the scale of the keypoint. All peaks which magnitude is higher than 80% of the highest peak in the orientation histogram are selected.

Then, the histograms are grouped into (usually 4×4) areas and 8 bins histogram (quantized) descriptors are created. Thus, a vector of the SIFT descriptor usually has $4 \times 4 \times 8 = 128$ dimensions. Finally, the vector is normalized to length unit due to the contrast invariance. Brightness invariance is provided because the image gradients are computed from pixel differences. Consequently, the SIFT descriptor is invariant to affine changes in illumination and geometry as same as to rotation, translation and scale. The influence of large gradient magnitudes is reduced due to limiting of non–linear illumination changes.

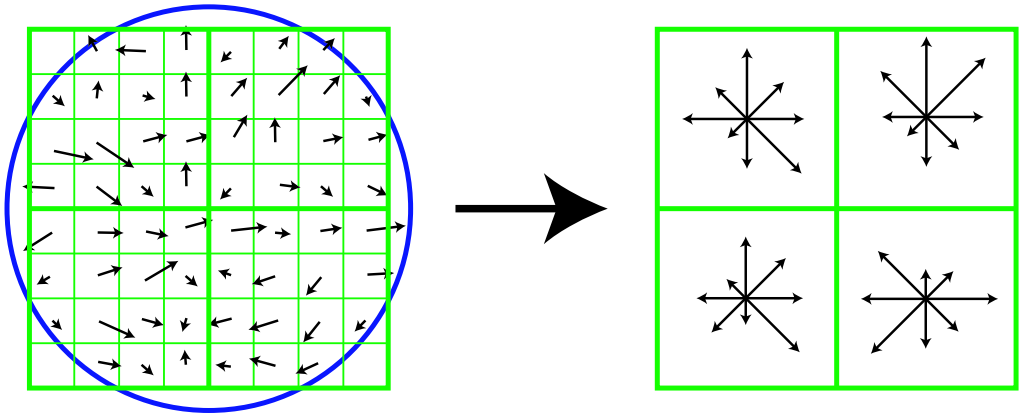


Fig. 2.11: SIFT Descriptor – gradient magnitudes and orientations around the keypoints are weighted by a circular Gaussian (blue circle). Next, orientation histograms are computed (the figure shows 2×2 descriptor computed from 8×8 array), courtesy of [33]

Gradient Location–Orientation Histogram

Gradient Location–Orientation Histogram (GLOH) descriptor was introduced by Mikolajczyk and Schmid in 2004 [42]. It is an extension of SIFT and was proposed to increase its robustness and distinctiveness. By comparison with SIFT, GLOH use log–polar location grid with (usually) three bins in radial and eight in angular direction. The central bin is not divided into angular directions (17 bins together). The gradient orientations are quantized into 16 bins, therefore the GLOH histogram has $17 \times 16 = 272$ dimensions. Finally, the dimension of the vector space is reduced by Principal Component Analyses (PCA).

PCA–SIFT

PCA–SIFT descriptor is based on gradient orientations in x and y directions computed within the support region (usually 41×41) centered at the keypoint. Thus the input vector has $2 \times 39 \times 39 = 3042$ dimensions. The vector space is normalized to unit length to reduce illumination changes. Finally, the PCA–SIFT descriptor is created by reduction of the vector space by PCA [26].

Also, it is good to noted that there exist other modifications of SIFT as Mahalanobis SIFT [38], Colored SIFT [1], ...

Shape Context

Shape Context is similar to the SIFT class of descriptors, however it is based on edgels instead of keypoints. Firstly, edges are estimate by Canny edge detector. Next, next locations and orientations of edge points are quantized to the 3D log–polar histogram of locations and orientations. Usually, 9 radial bins and four angular (horizontal, vertical and two diagonals directions) are used. Therefore, the Shape Context descriptor has $9 \times 4 = 36$ dimensions.

Let us note that it is possible to add weighting of each histogram sample by its magnitude. It was reported that this approach has better results than the original Shape Context algorithm [42].

2.3.2 Spatial–Frequency Techniques

The Fourier transform decompose the image into the set of basis function. However, these function do not describe the spatial relations between the points and the basis functions are infinite. There exists a ways, how to adopt the Fourier transformation to describe the characteristics of a local patches, for example Gabor transform or generally Short Time Fourier Transformation in general or wavelets. The main drawback of these approaches is that it is necessary to have large number of these filters to describe small changes in frequency and orientation [61].

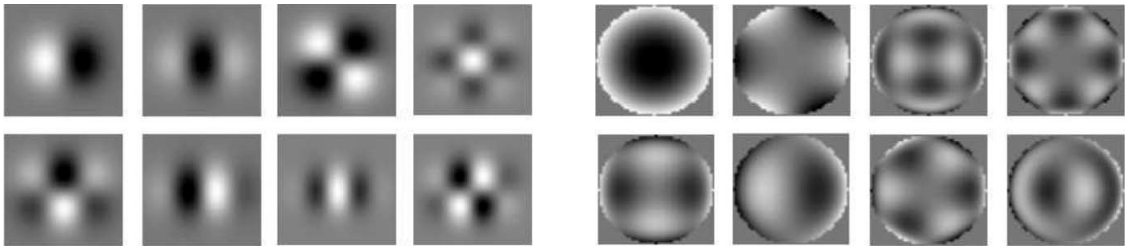
2.3.3 Differential Descriptors and Complex Filters

Differential Descriptors are based on a set of image derivatives which approximates a point neighborhood. Steerable filters steer derivatives in a particular direction given the components of the local jet. Steering derivatives in the direction of the gradient makes them invariant to rotation.

Complex filters are derived from the family [54, 42]

$$K(x, y, \theta) = f(x, y)e^{i\theta} \tag{2.7}$$

where θ is the orientation. Filters are usually computed up to the fourth or sixth order.



(a) (b)

Fig. 2.12: Gaussian derivatives (a) up to fourth order. Complex filters (b) up to sixth order, courtesy of [66]

2.3.4 Others Techniques

Moment Invariants

Another approach could be Moment Invariants based descriptors. These descriptors are better to use for color images due to the moments of higher order and degree are sensitive to small geometric and photometric distortions. Hence, with color images, it is possible to compute moment invariants for each color channel and between them.

Mindru et al. proposed Generalized Color Moments [44]. They characterize the shape, the intensity and the color distribution of the region pattern in a uniform manner

$$M_{pq}^{abc} = \iint_{\Omega} = x^p y^q I(x,y,R)^a I(x,y,G)^b I(x,y,B)^c \,\mathrm{d}x\mathrm{d}y \tag{2.8}$$

with order $p+q$, degree $a+b+c$ and R, G, B denotes color channels of image I. They are independent and can be computed for any order and any degree, however usually are used up to the second order and first degree.

Visual Vocabularies

A method adapted from the text search (inspired by Google web search) was published by Sivic and Zissermann [59]. It is based on representation of a feature descriptor by a vector of word frequencies. The method term frequency—inverse document frequency (tf–idf) is computed as follows: given a vocabulary of k words, then each image is represented by a tf–idf weighted vector

$$\mathbf{v} = (t_1, \ldots t_i, \ldots t_k)^{\mathrm{T}} \tag{2.9}$$

where

$$t_i = \frac{n_{wi}}{n_i} \log \frac{N}{n_w} \tag{2.10}$$

where n_{wi} is the number of occurrences of word w in the image i, n_i is the total number of words in the image i, n_w is the number of occurrences of the word w in the whole database and N is the number of images in the database. The idea behind the weighting of vectors is that the words which occurs often in a particular image helps to describe the image. On the other hand, words which appear often in the database are downweighted, because they do not help to distinguish the different images.

2.4 Efficient Data Structures

The most time consuming part of many computer vision applications is a search of the nearest neighbours matches in high dimensional spaces. Due to our two claims, fast response which will be close to the real–time and a large database of features, it is clear that it is necessary to use some algorithm with sublinear searching time.

We can define our objective as follows: given a data set of n features $\mathcal{F}_D$ in d–dimensional Euclidean space $\mathbb{R}^d$, these points must be preprocessed in such a way that finding the nearest neighbor (or k–nearest neighbors) of a new query $\mathcal{F}_q$ in $\mathcal{F}_D$ whose distance to some point from $\mathcal{F}_D$ is minimum and this search is executed efficiently. This distance can be defined in many ways, e.g. Euclidean distance, Manhattan distance, ... In object recognition applications, $\mathcal{F}_D$ represents the set of image descriptors located in the database and $\mathcal{F}_q$ is a set of query image descriptors [3, 38].

2.4.1 Tree Structures

KD–Tree

Concept of KD–tree was introduced by Bentley [9] in 1975 as a straightforward generalisation of the binary search tree in higher dimensions. KD–tree belongs to a category of a geometric data structures and is based on iterative partitioning of input dataset. This splitting of $\mathbb{R}^d$ space into subspaces can be performed as follows: let us have a hyperrectangle, which is the d–fold product of closed intervals on the coordinate axes $(x_1, x_2, \ldots, x_d)$. Each internal node of the KD–tree is associated with a hyperrectangle and a hyperplane orthogonal to one of the coordinate axis which divide the hyperrectangle in half at each level of the tree. If this splitting is done at the median of the dimension of the largest variance, it will create a balanced binary tree with depth $\log_2 N$. The resulting subcells are then associated with the two child nodes in the tree and this process iteratively continuous until the number of data point in the hyperrectangle falls below some given threshold or until convergence. Subdivision of space into hyperrectangles is defined by the hyperrectangles associated with the leaf nodes. These hyperrectangles are called buckets. Data points are stored only in the leaf nodes of the tree, not in the internal nodes [2].

Let us note that the aspect ratio (the ratio of the length of the longest side to the shortest side) of these buckets are not bounded, so these buckets may be long in one dimension and short in others, therefore these elongated cells may be intersect during the search by a query sphere. However, there exist many variations on this splitting

rule, but there is no assurance that the tree will be balanced. For example Silpa-Anan and Hartley [58] introduced an improved version of the KD–tree algorithm in which multiple randomized KD–trees are created to speed–up approximate nearest neighbor search. The main difference is that the split dimension is chosen randomly from the first D dimensions on which the data has the greatest variance. The nearest neighbor search should not take more than $\mathcal{O}(\log n)$ time.

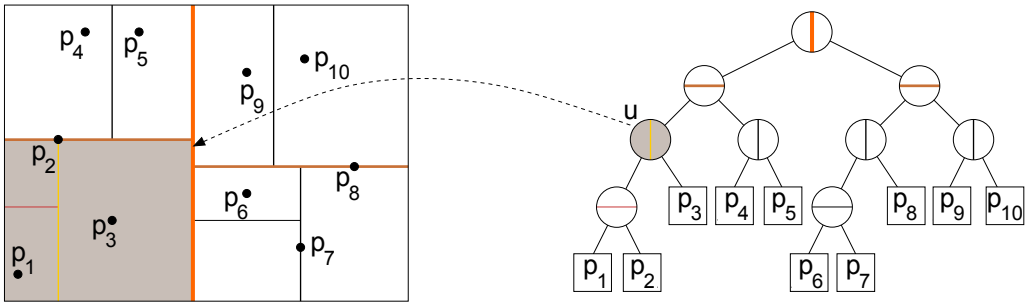


Fig. 2.13: A KD–tree of bucket–size one and the corresponding decomposition, [48]

It is reported in many papers that the biggest speed–up is up to 10 dimensions [68, 38]. Unfortunately, the most robust descriptors used in computer vision have many more dimensions (e.g. SIFT has 128 dimensions), however it could be overcome by Approximate Nearest Neighbor (ANN) search (will be discussed in section 2.4.3).

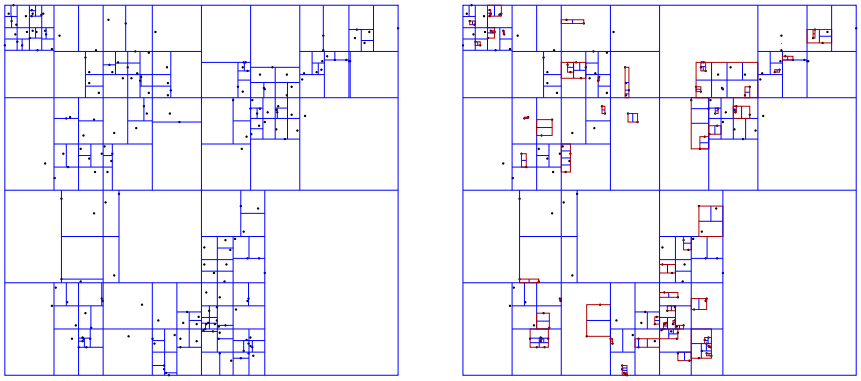
Balanced Box–Decomposition Tree

As we have discussed above, the aspect ratio of KD–trees is not bounded. Thus, it may be inefficient to use KD–tree on highly clustered data. Balanced Box–Decomposition (BBD)–tree should provide greater robustness on highly clustered data.

BBD–trees are closely connected with the KD–trees. The fundamental difference between BBD–tree and KD–tree is that each node of the BBD–tree is not associated simply with hyperrectancle, but each node of the BBD–tree is associated with a subspace called a cell. The cell is defined by a box or the set theoretic difference of two boxes, outer box and an optional inner box. All data points lying within the cell are associated with this cell. Points which lie on the boundary between two cells may be assigned to either cell, because cells are thought to be closed.

The construction of BBD–trees is again based on iterative partitioning of input dataset, but by comparison with KD–trees, BBD–trees are constructed through regular split and a more general decomposition called shrinking. These operations represent two different ways of partitioning a cell into two smaller subcells. Regular

split partitions a cell by an axis–orthogonal hyperplane into two subcells that are called low child and high child. A shrinking rule use rather a shrinking box than a hyperplane to split a cell into disjoint subcells (inner child and outer child).



(a) KD–tree (b) BBD–tree

Fig. 2.14: Comparison of KD–tree and BBD–tree. Blue denotes hyperplane splits and box partitions (shrinking) are red, courtesy of [38]

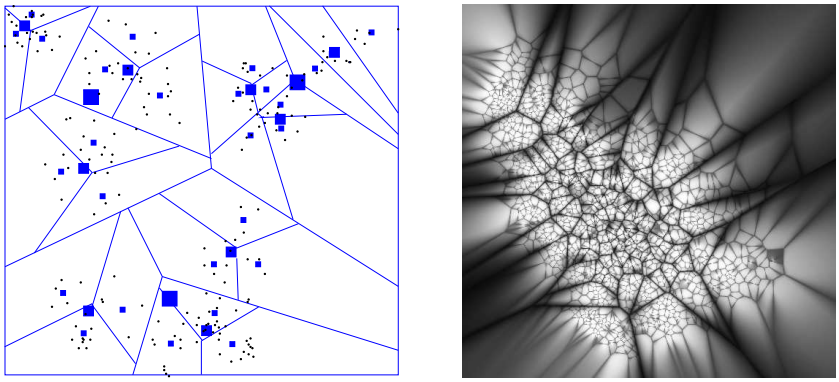
The question is, whether to use splitting or shrinking. More approaches are available and usual it is determined at each step of the algorithm. One of the simplest strategy is that these rules are applied alternately: splitting rule provides that the geometric size will decrease exponentially as we descend a tree and shrinking rule provides the same for the number of points associated with each cell node. Division of the space is repeated until the number of associated points is less than some threshold (bucket size). It implies that an evenly partitioned balanced tree with bounded aspect ratio of cell is constructed. BBD–tree for a set of n data points in E^d can be constructed in $\mathcal{O}(dn \log n)$ time and has $\mathcal{O}(n)$ nodes [4, 38].

Hierarchical K–Means Tree

Another efficient data mining method are hierarchical k–means trees. We discuss two main approaches: a top–down approach called divisive and bottom–up approach called agglomerative.

Divisive hierarchical k–means trees are derived from original k–means algorithm. The k–means algorithm could be described as partitioning method based on unsupervised learning. The input dataset is divided into k clusters, where k is fixed a priori. The objective is to find a partition in which points within each cluster are as close to the cluster center (the points have lowest variance) as possible and as far

from centers in other clusters as possible. It is often initialized randomly by k centers which may lead to different results from run to run. The first stage of iteration is to assign every point from input dataset to the closest center. The next stage is recalculating of the clusters centers – it is calculated as a mean of clusters. This algorithm should converge to a local optimum after a few iterations (the iterations stop when the k centers do not change much). Benefit of k–means is the efficiently of the partitioning method. On the other hand, the several drawbacks are sensitivity to outliers, centroids far from real clusters resulting in large cells or difficulties with setting appropriate k (centers may lie between more real clusters). Clustering of N data points of d dimensions with k centers and ℓ iterations is computational cheap because could be done in $\mathcal{O}(Nk\ell d)$ time, but complexity grows up when its k is comparable with N [28, 30, 38, 49].



(a) (b)

Fig. 2.15: Projections of hierarchical k–means trees, courtesy of [38, 49]

Agglomerative clustering represents bottom–up approach to hierarchical k–means trees. It builds a hierarchical merging tree from the leaves to the root. The algorithm is initialized by assigning every point in leaf nodes to its own cluster and then iteratively selecting and merging pairs of closest clusters in parent node until the top node is reached. The crucial parameter is the criterion used for selecting of merging clusters. For example average–link criterion is based on the measurement of the similarity of two candidate clusters as the average pairwise similarity between their members. The main advantage of agglomerative clustering is that it allows to specify the size or compactness of the resulting clusters. On the other hand, time $\mathcal{O}(N^2 \log N)$ and space $\mathcal{O}(N^2)$ complexity is the main drawback. Fortunately, post-processing can reduce number of nodes by merging some parents with children, however it is still inconvenient for large datasets due to space requirements [17, 30, 38].

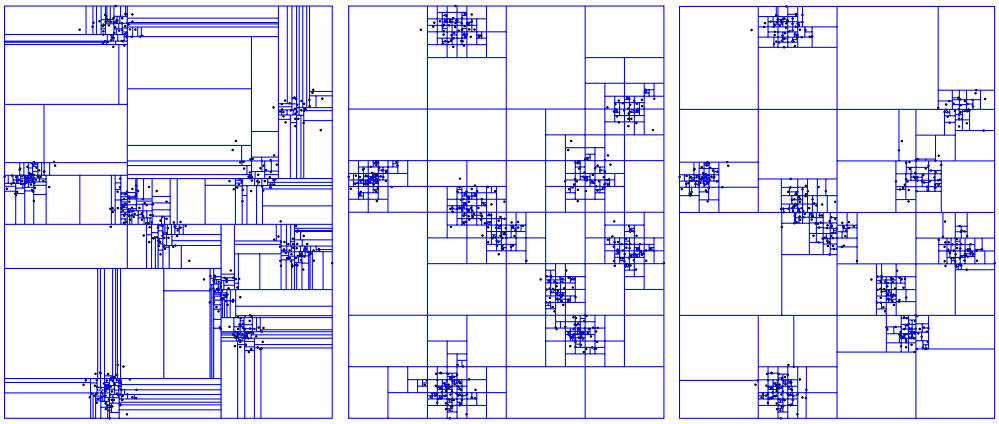
Bottom–up trees provide efficient nearest neighbour search, but have more complexity. Top–down trees may have some problems caused by their structures. Middle–out technique represents trade-off between efficiency and compactness of the tree [38, 46].

2.4.2 Splitting and Shrinking Rules

Many parameters of tree structures can be influenced by a set of different splitting and shrinking rules. In this section, we briefly discuss some of them.

Standard Splitting Rule

The splitting dimension is the dimension of the maximum spread of the current subset of input data points. The splitting point is the median of the coordinates of the subset along this dimension. A median partition of the points set is then performed. This rule guarantees that the final tree has height $\log_2 n$, and size $\mathcal{O}(n)$, but the resulting cells may have arbitrarily high aspect ratio (cited [48]).



(a) Standard split (b) Midpoint split (c) Sliding midpoint split

Fig. 2.16: KD–trees produced by different splitting rules, courtesy of [35]

Midpoint Splitting Rule

The midpoint splitting rule guarantees that the cells have bounded aspect ratio. It cuts the current cell into two identical subcells by a hyperplane passing through its midpoint and orthogonal to the longest side. The main disadvantage of this rule is that it can produce trivial splits (all of the points of current subset lie to one side of the splitting plane), so the resulting tree can be very large. This drawback can be overcome by simple modification called sliding–midpoint rule: if a trivial split is a

result of splitting, it tries to avoid this by sliding of the splitting plane toward the points until it finds the first data point. Maximum depth of the tree is n and size $\mathcal{O}(n)$ [4, 48].

Fair–Split Rule

Fair split rule is an efficient combination of the standard splitting rule and the midpoint splitting rule. The objective is to get balanced and partitioned tree without any elongated cells. The algorithm first selects the sides which could be split without violating a constant which defines maximum aspect ratio. Next, one side with the points largest spread is selected. Then, the cell is split under some assumptions: to preserve the aspect ratio, the longest side (other than this side) is selected and is determined how narrowly could be this side cut. This approach should be more robust than simple standard splitting rule or midpoint rule, however for highly clustered data the size of the tree may exceed $\mathcal{O}(n)$. Modification of this approach, which is based on theory that there are two types of good cells for splitting is called sliding fair–split rule. Firstly, the algorithm determines the longest side, selects the side, which could be split without violation of the maximum aspect ratio and among these, selects the side with the largest spread as same as fair–split does. Finally, the algorithm tries to do the most extreme cut which is allowed by the maximum aspect ratio [35, 48].

Simple Shrinking Rule

Simple shrinking rule depends on two fixed constants c and g and computes distances between each side of the rectangle for current subset of input data points and the corresponding side of the current cell's bounding rectangle. If at least c of these distances are larger than the length of the longest side of the cell's rectangle times g, then it shrinks all sides whose gaps are at box of the shrink and the outer box contains no point. If none of the gaps is large enough, then no shrinking is performed, and splitting is performed instead (cited [48]).

Centroid Shrinking Rule

Centroid shrinking rule depends again on two fixed constants f and m. It iteratively applies the current splitting rule. At each stage, the algorithm checks which subcell contains more points and repeat splitting on this part. It runs recursively until the number of points falls below a fraction of f of the size of the current subset of data point. If it iterates more than number of dimensions times m, it shrinks to the final inner box and all other points of the current subset of input data points are stored in the outer box. Otherwise splitting is performed instead of shrinking [48].

2.4.3 Search Techniques

In this section, we discuss different search techniques. Let us note that the brute force search is well known and used as a reference search technique. Thus, the brute force (linear) search which simply computes the distances from the query to each point in database in $\mathcal{O}(dn)$ time is not discussed here in detail.

Range Search

Range search is useful technique if it is necessary to find points with different distance in each dimension from the query point. The objective is to find all data points within some distance from query or to find all data points whose coordinates lie in a specified query range. The search can be defined as follows: given a query region R and v denotes current node in the tree. Let us assume that current v is a leaf. Then search verifies whether the points stored in v lie in R and reports positives. If the v is internal node, then search visits all subtrees as follows: if its region lies entirely in R, search reports all data point stored within the current subtree. If the current subtree partially overlaps R, search proceeds iteratively. All other nodes are ignored [10].

Standard k Nearest Neighbor Search

Standard search was introduced by Friedman et al. in 1977 [20]. The search starts by descending the tree to find appropriate bucket within query point lies. The algorithm then iteratively visits surrounding cells to determine whether the radius of the B ball centered at the query point is intersected and whether the radius is equal to distance between the query point and the closest data point visited so far. The nodes are checked until k leafs are discovered. The distance to the k–th neighbour determines which points are then reported as nearest neighbours [2].

Approximate k Nearest Neighbor Search

In many literature it is reported that k nearest neighbor search is inefficient for high dimensional trees (more than 10 dimensions), because query sphere can intersect many (especially elongated) cells and all of these cells have to be visited. This drawback could be overcome by reducing the number of the leaves to visit. One way could lead through maximum visited leaves criterion. Other approach, called ϵ–approximate k–nearest neighbor search was introduced by Arya et al. [3]. The main thought of this approach is that the search terminates if the distance from the

closest cell to the query point exceeds

$$\delta = \frac{d(p,q)}{1+\epsilon} \tag{2.11}$$

where p is the nearest neighbor found so far, q is the query point and ϵ is the positive termination parameter. In other words, p is within relative error ϵ of the true nearest neighbor. Approximate k–nearest neighbor search is efficient search with sufficiently precision, because many parts of computer vision algorithms are inaccurate [27, 38].

Priority Search

Priority search was described by Arya and Mount in 1993 [2]. Standard k–NN search usually explores the nearest neighbor before it visits all cells. Priority search presents more efficient way how to get nearest neighbors because it can terminates earlier. On the other hand, the search does not guarantee that the nearest neighbor was found. This search visits the cells in increasing order of distance from the query point. This can be efficiently performed by priority subtrees as follows: if the current node is internal, the distances from the query point to the cells of children are computed and the closer child is traversed iteratively. The farther child is added on the priority queue and sorted by distance. If the current node is leaf, distances to the points stored within this cell are computed and the algorithm proceeds by the next item from the top of the queue. The search terminates either when the priority queue is empty or when the distance to the nearest cell on the priority queue is farther than distance to the closest data point [2, 3, 48, 58].

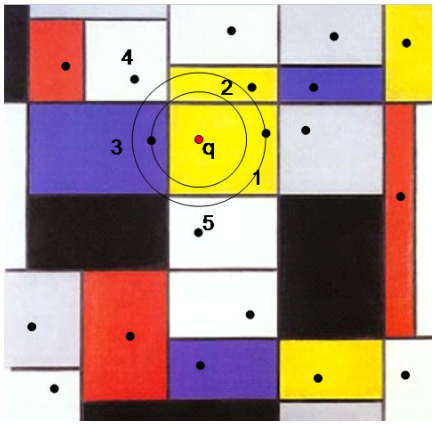


Fig. 2.17: Illustration of priority search – query tree is represented by a red dot, the nearest neighbour lies in cell 3. The algorithm works exactly as we describe above, courtesy of [58]

2.4.4 Locality Sensitive Hashing

Another possibility for sub–linear search by reduction of high–dimensional space is Locality Sensitive Hashing (LSH) introduced by Indyk and Motwani [24]. The main idea is to use a family of locality-sensitive hash functions to hash nearby objects in the high–dimensional space into the same bucket, because the objects that are close to each other have a higher probability to fit than object their distances to each other are farther [34, 21]. There exists many extensions which allow to use different distances like Hamming distance, or are based on p–stable distributions which have $\mathcal{O}(\log n)$ time complexity. The hash function is defined as:

$$h_{a,b}(v) = \frac{ab + v}{W} \tag{2.12}$$

where a is a d–dimensional random vector with entries chosen independently from a p–stable distribution and b is a real number chosen uniformly from $b \in < 0, W >$. The p–stable distributions are for example Gaussian distribution or Cauchy distribution [16].

LSH techniques have high accuracy, however need to use multiple hash tables to produce a good candidate set.

3 PERFORMANCE EVALUATION

In this chapter, we describe used datasets for performance evaluation in section 3.1. Then, we discuss standard evaluation criterion in section 3.2, overview of performance evaluation framework in section 3.3 and finally implementation details in section 3.3.6.

3.1 Data Sets

We are using two different data sets[1] to create the database – the Grafitty data set for reference and query images and the PASCAL data set images as padding.

The Graffiti Data Set

The Graffiti data set is provided by the University of Oxford. It consists of eight image sequences with different geometric and photometric transformations. Each image sequence is composed by six images in `png` or `ppm` format. The ground truth homography (plane projective transformation) between the first image and the rest of images in the sequence is provided. The data set sequences show six different transformation: rotation, scale change, viewpoint change, image blur, `jpeg` compression and change in illumination (acquiring process and transformations are discussed in detail in [42]). Also, two different scene types are provided: structured scenes (homogeneous regions separated by distinctive edges) and the other contains of repeated textures of different forms.

All images in each sequence are related by the ground truth homography estimated between the first image and the rest of images in the sequence. The homography matrices are computed in two steps. The first step consists of estimation of the homography using the manually selected correspondences. Then, the images are aligned with the reference image. The second step is to compute an accurate residual homography between the reference image and the transformed image, with automatically detected and matched points of interest. Finally, the homography estimation is based on combination both, the approximate and residual homography results in an accurate homography between the images.

The PASCAL VOC 2007 Data Set

The other data set, which is used to extend the database is provided by the PASCAL Visual Object Classes Challenge 2007 [19]. This database consists of `jpg` images in

[1] Both data sets are available at http://www.featurespace.org

(a) Bark, Boat: zoom + rotation

(b) Graffiti, Wall: viewpoint change

(c) Bikes, Trees: image blur

(d) Ubc: JPEG compresion, Light: light change

Fig. 3.1: Graffiti data set

medium resolution. The database comprehends 7818 images with different classes of objects like aeroplanes, trains, birds, boats, ... Due to the time complexity of all tests, we use only first 1000 images of PASCAL data set.

3.2 Evaluation Criterion

We use standard and widely accepted evaluation criterion [42] which is based on the number of correct and false matches. As a quality criterion, ground truth S_A, precision S_B and recall S_{BA} are used. For time measurements, speed up S_T is used rather than pure time in seconds.

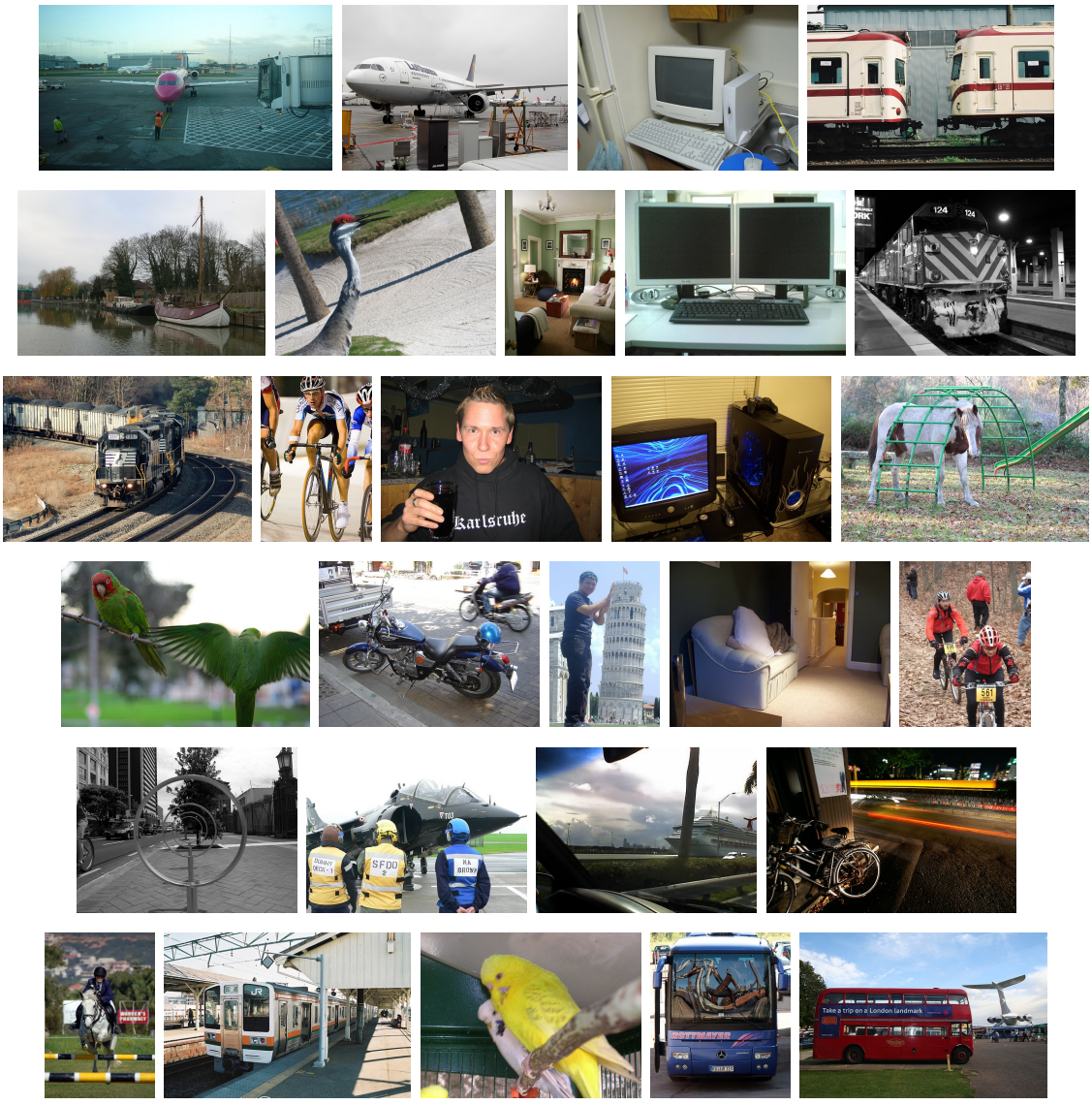


Fig. 3.2: Examples of PASCAL VOC 2007 data set

3.2.1 Ground Truth

Firstly, the A parameter is determined by a number of detected features in the query image (file M_q)

$$A = \sum_{i=1}^{i=n} F_i, \tag{3.1}$$

where F_i denotes i–th extracted local invariant feature in the query image and n denotes the maximum of detected features.

Next, spatial coordinates of extracted features are projected by the 2D plane

projective transformation (homography) from query image to the reference image

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.2}$$

where $h_{31} = h_{32} = 0$ and $h_{33} = 1$ in the case of an affine transformation [61]. It can be rewritten in a compact form

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{3.3}$$

Next, the projection of a spatial coordinates of extracted features can be done by conversion from homogeneous coordinates ($x_3 = 1$, because we assume that the point is not a point at infinity[2])

$$x = \frac{x'_1}{x'_3} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}}{h_{31}x_1 + h_{32}x_2 + h_{33}} \tag{3.4}$$

$$y = \frac{x'_2}{x'_3} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}}{h_{31}x_1 + h_{32}x_2 + h_{33}} \tag{3.5}$$

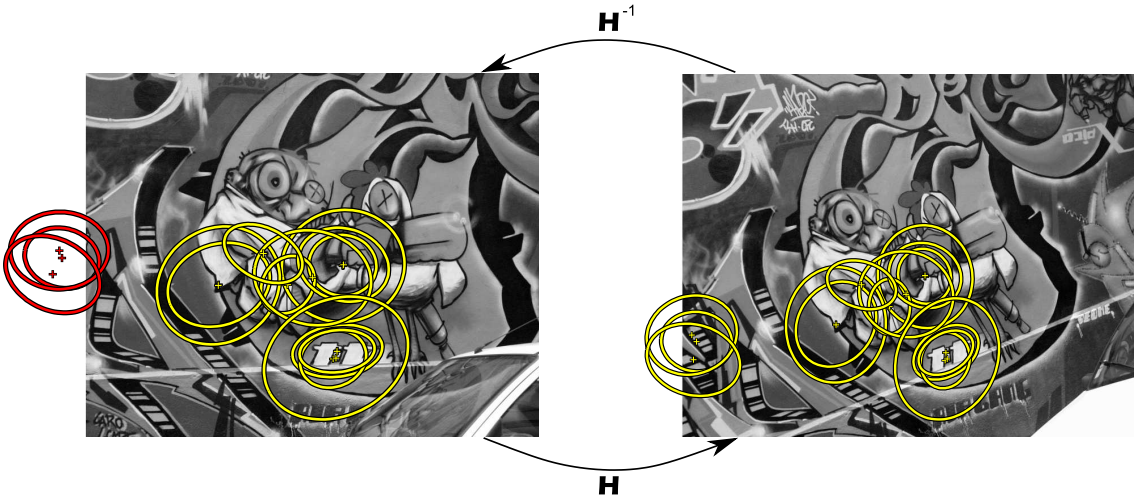


Fig. 3.3: Figure shows detected features in a query image (right) and projected features to the reference image (left). Yellow are accepted features by A^*, red are rejected features (only in left image, yellow ellipses in right image are A). It is possible to project features in both directions, because 2D plane projective transformation is invertible. Only first 15 features are shown for figure clarity.

The A^* parameter is the number of all projected features by plane projective transformation (homography) from query image (M_q) to the reference image (M_r),

[2]Points $[x_1, x_2]^{\mathrm{T}}$ can be expressed in homogeneous coordinates in 3D vector space as $[\lambda x_1, \lambda x_2, \lambda]^{\mathrm{T}}$, where $\lambda \neq 0$. However, usually are used expressions like $[x_1, x_2, 1]^{\mathrm{T}}$

but the features which projected spatial coordinates are outside of the boundaries of reference image are rejected.

$$\text{if} \quad F_i(x,y) \begin{cases} 0 & < x < w_r \\ 0 & < y < h_r \end{cases} \Rightarrow A^* = \sum F + F_i \qquad F_i \in A \tag{3.6}$$

where F_i is the i–th feature detected in the query image, w_r is the with and h_r is the height of the reference image.

Next, by thresholding the Euclidean distance between query points included in A^* and (all) reference points by factor t_a, we get parameter A^+ (Eq. 3.8). The threshold t_a is relatively set and depends on scale

$$t_a = M\sqrt{s_1 s_2}, \tag{3.7}$$

where s_1 and s_2 are scales of features and M is a multiplication constant.

$$\text{if} \quad d(F_{iq}, F_r) < t_a \Rightarrow A^+ = \sum F + F_{iq}, \qquad F_{iq} \in A^*, F_r \in M_r \tag{3.8}$$

where $d(F_{iq}, F_r)$ denotes the Euclidean distance between the i–th query point F_{iq} contained in A^* and reference point F_r from M_r.

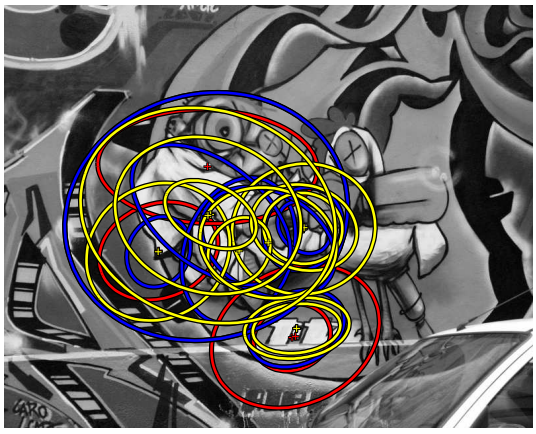


Fig. 3.4: Figure shows rejected features by A^+ (red), accepted features (yellow) and correspondences in the reference image (blue). All query features are projected to the reference image. Only first 15 features from A^* are shown for figure clarity.

Finally, S_A is a ratio between number of correspondences A^+ and number of projected features A^* to reference image

$$S_A = \frac{A^+}{A^*} \tag{3.9}$$

Regions Overlap

Let us note, that an overlap error is usually used instead of t_a and t_b (discussed in section 3.2.2). Overlap error describes, how well the regions (depending on the feature scale) correspond under a homography transformation. It can be defined by the ratio of the intersection and union of the regions

$$\epsilon_S = 1 - \frac{\mathbf{A} \cap \mathbf{H}^{\mathrm{T}}\mathbf{B}\mathbf{H}}{\mathbf{A} \cup \mathbf{H}^{\mathrm{T}}\mathbf{B}\mathbf{H}} \tag{3.10}$$

where $\mathbf{A}$ and $\mathbf{B}$ are regions and $\mathbf{H}$ is homography matrix. Due to the computational complexity of the numerical enumeration of this approach and a huge amount of data and experiments, we do not use it.

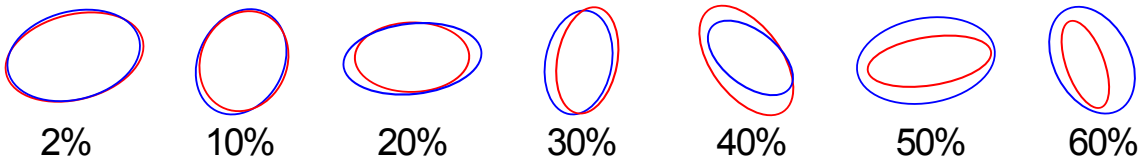


Fig. 3.5: Overlap error, courtesy of [43]

3.2.2 Precision

The second criterion is a precision S_B. The number of correct matches B^* and potential features B found in a database are needed.

We get the number of potential features B by thresholding the distance between the query feature (file M_f) and the found feature in the database (efficient data structure) by factor t_{kd}. A range of values of t_{kd} depends on properties of used descriptor and efficient data structure. It is possible to use either, the square root of Euclidean distance or just square of distance [3, 4].

$$\text{if} \quad d(F_{iq}, F_{nn}) < t_{kd} \Rightarrow B = \sum F + F_{iq}, \qquad F_{iq} \in M_f, F_{nn} \in M_f \tag{3.11}$$

where $d(F_{iq}, F_r)$ is the distance between a query point and its nearest neighbour found in a database.

[3] In fact, it is possible to use even any other Minkowski distance metric like L_1 (Manhatten) or L_∞ (Max) metric, however we use L_2 Euclidean metric for all experiments.

[4] It is necessary to use the same metric for both, fast feature matching in database and for performance evaluation.

The second parameter, number of correct matches B^*, is a little bit more complicated. At first, it is compared if the ID_{SR} of the image which feature is found in the database and the ID_{QS} of the image which feature is a query are equal. Next, homography and another threshold t_b is used to check if the query feature is quite close to the feature in the reference image [5]. The features which are not rejected is a set of correct matches and the size (quantity) of this set is B^*.

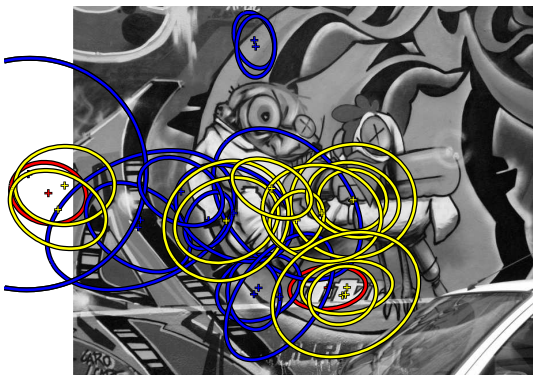


Fig. 3.6: Figure shows rejected features by B (red) with threshold set to 122500 (square of Euclidean distance), accepted features (yellow) and potential features (blue). All query features are projected to the reference image. Only first 15 features are shown for figure clarity.

Precision is a ratio between the number of correct matches B^* and potential features B found in a database

$$S_B = \frac{B^*}{B} \tag{3.12}$$

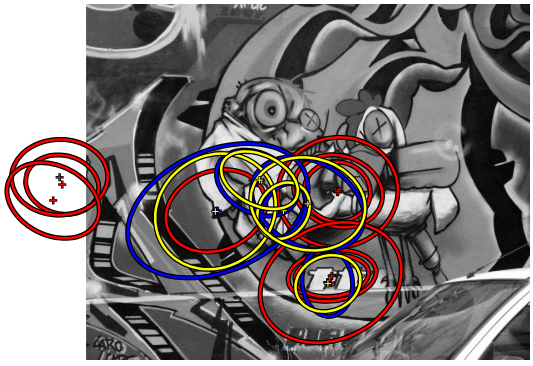


Fig. 3.7: Figure shows rejected features by B^* (red), correct matches (yellow) and appropriate nearest neighbours (blue). All query features are projected to the reference image. Only first 15 features are shown for figure clarity.

[5] The threshold t_b is determined by S_A. We set $t_b = 0.1467$ for all experiments.

3.2.3 Recall

Finally, the last quality criterion is recall S_{BA}. It denotes a ratio between the correct matches B^* and correspondences A^+

$$S_{BA} = \frac{B^*}{A^+} \tag{3.13}$$

The expected curve shapes and their interpretation is discussed in the next chapter. Also, let us note, that recall and precision are independent.

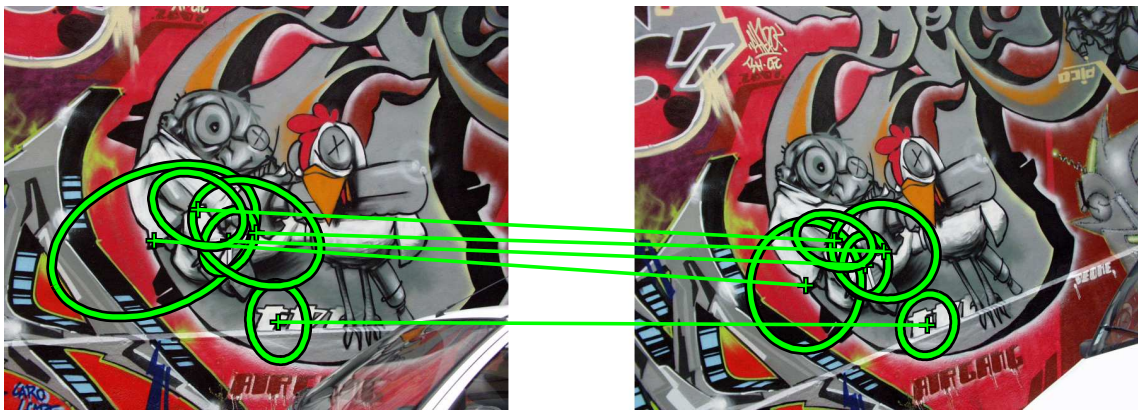


Fig. 3.8: Figure shows correct matches for first 15 features.

3.2.4 Speed–up

As we have mentioned in the introduction, the time complexity of feature matching is crucial for SLAM applications. Therefore, another criterion is necessary to measure the speed efficiency. We would rather use speed up S_T than time in seconds because raw time measurement is more sensitive to the used computing system. Moreover, we get how many times are the efficient data structures faster than sequential search

$$S_T = \frac{T_{brute\_force}}{\frac{T_{efficient\_data\_structure}}{N_{trees}}} = \frac{T_{brute\_force}}{T_{efficient\_data\_structure}} N_{trees} \tag{3.14}$$

where T_{brute_force} is time measured for sequential search, $T_{efficient\_data\_structure}$ is the time measured for efficient data structure search and N_{trees} is the number of multiple randomized trees.

3.3 Overview of Experimental Framework

In this section, we discuss the overview of the performance evaluation framework (Fig. 3.11). Our evaluation system consists of three parts. The first step is to compute local invariant descriptors and its parameters for detected features in both, the Grafitty data set and the PASCAL data set. We use `compute_descriptors` application provided by Mikolajczyk at Feature Space project[6]. It is necessary to compute the descriptors with the same settings for all images.

Next, computed features are divided into database and query images. Only the descriptors and their parameters for the first image from each sequence from the Graffiti data set are stored in the database. The database is then extended by all descriptors and their parameters from the PASCAL data set. The `ID` to each image is assigned. The rest of Graffiti data set are then used as query images. After these steps, it is possible to start with fast feature matching with our application `feature_eval`. The KD–trees, BBD–trees, k–means trees structures and brute force (sequential) search are provided (section 3.3.6). The trees are created only from descriptors (last d numbers on each database row). Also, the queries use only the descriptors. When the appropriate descriptors are found in the database, the parameters of the feature are obtained by the ID of the nearest neighbor (ID_F). Finally, the distance between the query and its nearest neighbor, the ID_{SR} of the relevant image, retrieved parameters, ID_{QS} of the query sequence and the query parameters (for the query feature) are produced as a result.

The third part of our system is used to get appropriate scores for the matches. Only this part of our system is implemented in Matlab, others are written in C++. It loads in a loop results provided by `feature_eval`, the reference image descriptors and its parameters, the same for query image, both images and homography matrix. It computes the entire score set and produces appropriate figures.

3.3.1 Database

The database consists of images from both datasets[7]. Only first images from each sequence from Graffiti data set are used as reference images (Fig. 3.9). Then, the database is extended by other images from PASCAL data set. The number of images used from PASCAL (or other) dataset is dependent on the required size of the

[6] http://www.featurespace.org

[7] It is possible to add more images from various datasets.

database. Figure 3.2 shows examples of PASCAL dataset which are used to extend the database [8].

(a) Bark, Boat, Graffiti, Wall

(c) Bikes, Trees, Ubc, Light

Fig. 3.9: Reference images

All images are processed by `compute_descriptors` with the same parameters, so we have files (M_r) with extracted features, their parameters and descriptors (Listing A.2). The list of these files is stored in another file (Listing A.1) which is used to determine the files (images) which creates the database.

Next, the file which defines the database (`df` file) is created by `feature_eval`. In this step, each feature is assigned with ID_{SR} of its appropriate file. Then, the database file is used to create efficient data structure. More information about usage of `feature_eval` could be found in appendix A.2.1.

3.3.2 Efficient Data Structures

The file which contains all features which should be stored in efficient data structures (`df` file) is used by `feature_eval` to load all features. Each line which is corresponding with different feature is separated. Only last d numbers on each row are used to create efficient data structure(s)[9]. All features parameters are stored in a list, which is used when a nearest neighbour is found. Detailed description of the database file is in Listing A.2.

Various efficient data structures are provided. We use KD–trees in all experiments, however BBD-trees and k–means trees are also provided by `feature_eval`.

[8] 1000 PASCAL images were used for our experiments.
[9] d – number of dimensions of used descriptor

Different search strategies (ϵ–Approximated Nearest Neighbour, priority search, ...) are available as well as different options of dividing the whole data set into multiple randomised trees.

All steps described above could be run in off–line, so the time which is consumed by them is not crucial.

3.3.3 Query Images

The rest of all sequences provided by the Graffiti dataset (it means all images from Graffiti dataset which are not used to create the database) are used as query images. All query images are shown below (Figure 3.10). Again, `compute_descriptors` is used in the same way as in the case of reference files (M_r) and query files (M_q) are produced.

3.3.4 Fast Feature Matching

Fast feature matching is the most important step for various computer vision applications which are able to work with the real–time response. As an input of fast feature matching (`feature_eval`) are used query files (M_q), which are loaded one by one, just like video frames in real application. The list of files which should be used for fast feature matching as queries is stored in a file f_q. More information about this file could be found in Listing A.3. Query files are again divided into two parts – feature descriptors are used for matching and parameters are stored in memory.

Then, fast feature matching works as follows: descriptors are used to find the nearest neighbour of the feature in efficient data structure (single or multiple trees). In the case of single tree, it is straightforward: the nearest neighbour is that one, which was found in the tree. In the case of multiple randomised trees, the shortest distance between the query and found nearest neighbor is chosen from all found neighbours in multiple trees. Next, index of the found nearest neighbor (ID_F) is used, to select the appropriate row of parameters from the list of parameters.

Finally, file with results (M_f) is created. The file has the same name as the query file, however it differs in extension which is used (depending on parameters of efficient data structure and search). It consists of the distance between the query feature and found nearest neighbour, ID_{SR} which determines the image which feature was found. These numbers are followed by parameters of feature which is determined by the nearest neighbour found in efficient data structure(s). The next number is ID_{QS}

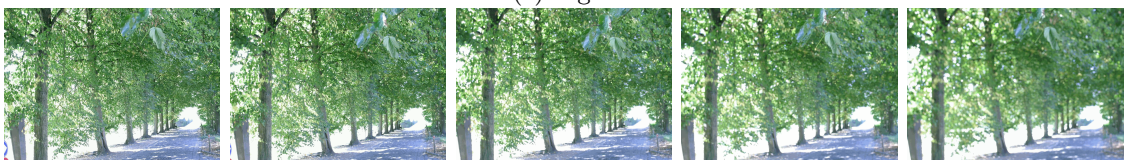
(a) Bark

(b) Bikes

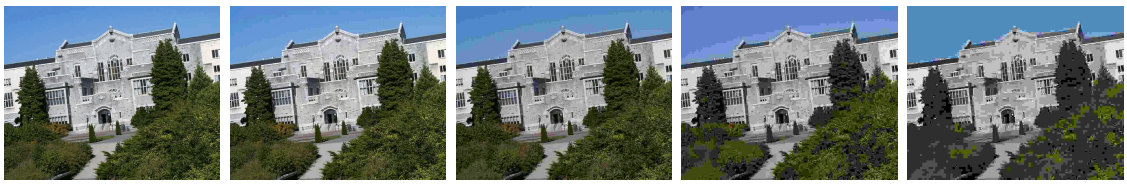
(c) Boat

(d) Graff

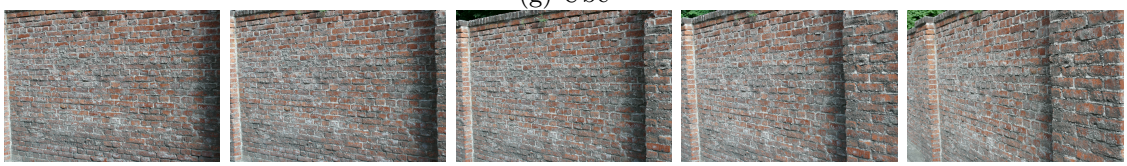
(e) Light



(f) Trees



(g) Ubc



(h) Wall

Fig. 3.10: Query images

which identifies the sequence of a query image which was used for matching and this number is followed by parameters of the feature which was used for matching.

Let us note, that it is important to store the images in the same order in which query images are used for matching. This rule is required, because it is necessary to distinguish during the performance evaluation step whether the found feature is produced by appropriate reference image or not.

Moreover, `feature_eval` saves the time needed for fast feature matching in a file with the same name as f_q, but with extension `.tms`.

3.3.5 Performance Evaluation

Matlab scripts are provided for performance evaluation of detectors, descriptors and fast feature matching. It allows to create a file, which contains paths to all required files (M_r, M_q, M_f), images (reference and query) and homography matrix. These files may be used by function `repeat` for batch processing of results.

Next, all evaluation criterion are counted (Section 3.2) and all results are stored in Matlab matrices. It is possible to load these matrices and use the results, to produce figures with various curves, for criterion S_A, S_B, S_{BA} and speed up S_T by files `plotEvaluation` and `timesPlot`.

3.3.6 Implementation

Here, we provide a few information about the implementation of our system. It is discussed more in detail in the appendix A. The `feature_eval` is written in C++, it could be run either on Windows or Linux operating system. It is based on the two libraries, for the fast approximate nearest neighbor search in KD–trees and BBD–trees it uses ANN library written by Arya and Mount [2, 3, 4, 48]. The other library, which is used for k–means trees is FLANN [49]. The whole framework could be run easily from command line, thus some PowerShell (for Windows, we expect that every Unix–like user can use their own) scripts are provided for batch evaluation. More information about usage of the framework can be found in the appendix A.

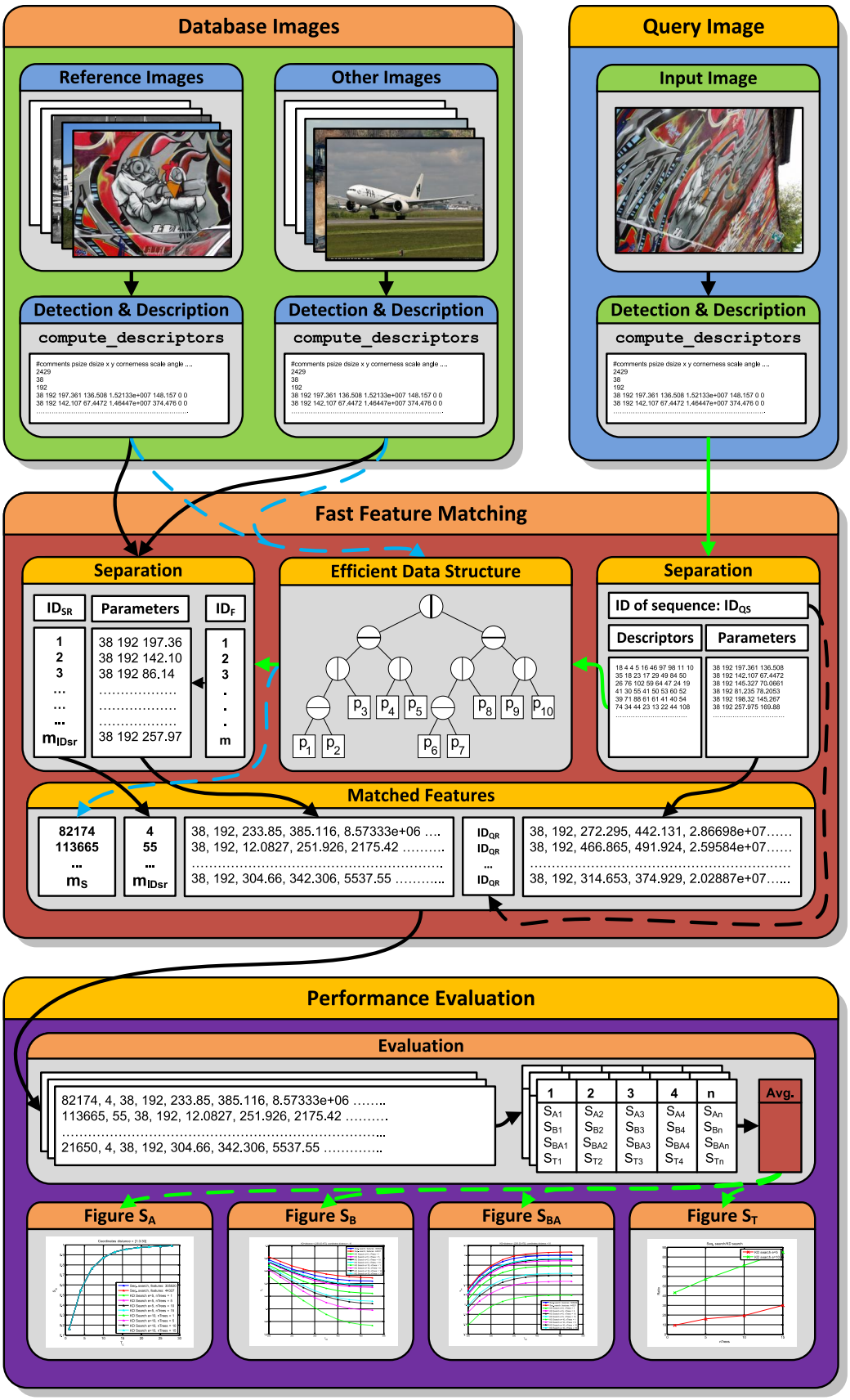


Fig. 3.11: Overview of the performance evaluation framework

4 EXPERIMENTAL RESULTS

In this chapter, we produce performance evaluation of all experiments. We begin with definition of various matching strategies in section 4.1. In section 4.2, we discuss interpretation of figures and possible curve shapes. Section 4.3 describes properties of both data sets and section 4.4 provides ground truth, which is used for other experiments.

Section 4.5 provides results for ϵ approximate nearest neighbour search. Performance of multiple randomised KD–trees are summarized in section 4.6. Section 4.7 compares ANN search and priority search. Our novelty data partitioning approach is discussed in section 4.8. Importance of dimensionality of used descriptors is discussed in section 4.9. Other important parameter for real–time response, number of maximum visited leaves is discussed in section 4.10. The last but one experiment deals with influence of database size in section 4.11. Finally, section 4.12 summarize construction times of efficient data structures.

4.1 Matching Strategies

The definition of the matching depends on the appropriate matching strategy. Several strategies could be used, we briefly discuss three of them. If we use simple thresholding, then we say, that two regions **A**, **B** are matched, if the distance between their descriptors are bellow some threshold. Another approach is nearest neighbour matching. In this case, the regions are matched, if the descriptor $\mathbf{D}_A$ is the nearest neighbor of descriptor $\mathbf{D}_B$. The third approach, which is called nearest neighbour distance ratio (NNDR) matching, is similar to the previous, but the threshold is applied to the ratio between the first and the second nearest match

$$\frac{\|\mathbf{D}_A - \mathbf{D}_B\|}{\|\mathbf{D}_A - \mathbf{D}_C\|} < threshold \tag{4.1}$$

In the first case, the descriptor may have several matches and several or even all of them may be correct/false. In the case of nearest neighbour, the only one match is possible. It was reported, that nearest neighbour based matching has better results than threshold based. Also, it was reported, that the results of nearest neighbor matching and nearest neighbour distance ratio matching are comparable, but NNDR penalizes the descriptors which have many similar matches. Therefore, the results are even better than in case of simple nearest matching, however it is difficult to implement this rule in large databases [42].

4.2 Interpretation of Figures

In this section, we discuss interpretation of figures and expected curve shapes of evaluation criterion defined in section 3.2. The figures which will be produced for score S_A represent ratio between the number of correspondences depending on appropriate threshold and projected features. The shape of curve for S_A criterion is increasing. An perfect feature detector and descriptor should have S_A score as high as possible (equal to one) for all possible thresholds (overlap errors, ...).

The second criterion called precision (S_B) is a ratio between the correct matches and a potential features found in database. The figures are produced for various distance thresholds from the query to the nearest neighbour and the comparatively set threshold for spatial distance between the reference and projected features coordinates (and ID of sequence of the feature found in database must respond to the query sequence). The third criterion called recall describes the ratio between the number of correct matches and correspondences.

An ideal feature detector and descriptor should have recall equal to one for any precision. However, in practice the precision curve decreases with the growing distance between the query and its nearest neighbour. On the other hand, recall increases for an increasing distance threshold as noise which is introduced by image transformations and region detection increases the distance between similar descriptors. At some point, recall attains its maximum (increase is very slow), due to the remaining corresponding regions are very different from each other and, therefore, the descriptors are different [42].

The last criterion is speed up (S_T). It simply denotes time complexity of search in efficient data structure for appropriate parameters. The curve should be increasing with the number of the multiple randomized trees, because the search may run in parallel. At some point the speed up factor is saturated and the curve increases slowly. It could be explained by the consideration of following: when the search in each multiple tree is done, it is necessary to sequentially select from the found neighbors that one, which has the shortest distance to the query point. At some point, the selection of the appropriate neighbor, takes much time (the number of multiple trees is too high) and the search time in a data structure starts to be very similar, however it is necessary to do the search in more trees. That is the reason, why speed up is produced only by the growing number of trees and not by the more efficient search after that point. The objective is to find the parameters which will ensure the trade–off between the sufficient recall, precision and the time complexity.

4.3 Properties of Data Sets

Before we start with performance evaluation of efficient data structures, we briefly discuss the properties of the database images and test sequences. All images (both the database as same as the query) are processed by `compute_descriptors` with the same parameters. We use combination of Harris and Hessian scale invariant detectors, and Colored SIFT descriptor. By comparison with standard SIFT which has 128 dimensions, it has additional 64 dimensions for descriptors computed in opponent color space. Therefore the full descriptor has 192 dimensions. All time measurements was produced by the same computer[1]. Square of Euclidean distance was used for all experiments. We set the thresholds for all experiments to these values [2]: t_a =`[0.02:0.03454:0.4]`, t_{kd} =`[230:20:450].^2` and $t_b = 0.1467$. The nearest neighbor matching strategy is used for all experiments.

Table 4.1 summarize the number of a detected features in the Graffiti data set. Only features detected in the first image of each sequence (44647 features) are used to create a database. It is shown, that in some cases (`Bikes`, `Light`) the number of features detected in the images decreases with the difficulty of transformation (images 2 – 6). Table 4.2 recapitulate how many features are stored in database from each sequence and from each dataset. The Graffiti dataset participates only by 2.947% on the database. All other features are produced by the PASCAL VOC 2007 dataset.

For all experiments, we use average over the all test sequences of S_A, S_B, S_{BA} and speed up S_T. Before we start with evaluation of various parameters of efficient data structures, we used database created only from reference images to evaluate each test sequence independently by sequential search. It is useful to show the influence of test sequences to the overall score. The average score is used to distinguish two different errors. The first one rises when the database is extended and is introduced by the descriptors quality, the second one is introduced by efficient data structures.

It was observed, that all sequences have similar values of a ratio between number of correspondences and number of projected features (S_A). Others criterion are more interesting – it can be stated, that the Graffiti data set contains three difficult sequences (`bark`, `graff`, `boat`). On the other hand, three sequences are quite easy (`bikes`, `light`, `ubc`) and the rest has similar values close to the average.

[1] Dell Power Edge 6850, SuSE Linux 9.3, 4× 3.3GHz/8MB cache P4 Xeon EM64T with HT, 32GB RAM, however we have run the tests sequentially.

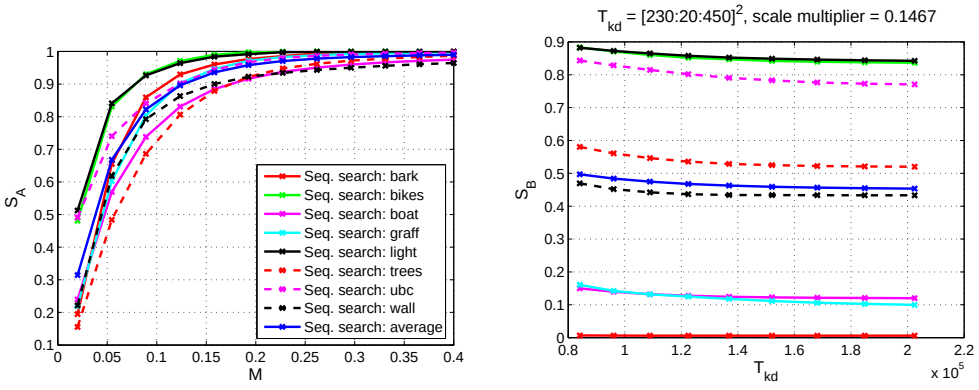
[2] Written in a standard Matlab syntax: `[a:b:c]` `a` = start number, `b` = the value of the increment, `c` = last number and `.^` denotes array power.

Tab. 4.1: Number of detected features in the Graffiti data set

	Image						
Sequence	1	2	3	4	5	6	**Sum**
Bark	3 915	3 915	3 915	3 915	3 915	3 915	23 490
Bikes	5 815	4 318	4 150	3 039	2 509	2 379	22 210
Boat	5 779	5 779	5 779	5 779	5 605	5 415	34 136
Graffiti	5 119	5 119	5 119	5 119	5 119	5 119	30 714
Light	4 902	3 860	3 380	3 124	2 956	2 891	21 113
Trees	6 999	6 999	6 999	6 999	6 999	6 999	41 994
UBC	5 119	5 119	5 119	5 119	5 119	5 119	30 714
Wall	6 999	5 983	5 983	5 983	5 983	5 983	36 914
Sum	**44 647**	41 092	40 444	39 077	38 205	37 820	**241 285**

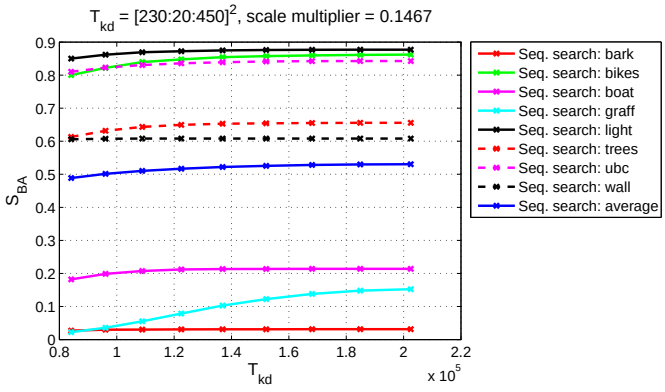
Tab. 4.2: Number of features used in database. It is shown that only 2.947% of features stored in the database is produced by Graffiti dataset images.

Sequence	Bark	Bikes	Boat	Graffiti	Light	Trees	UBC	Wall	PASCAL	**Sum**
# features	3 915	5 815	5 779	5 119	4 902	6 999	5 119	6 999	1 470 323	**1 514 970**
Percentage	0.258	0.384	0.381	0.338	0.324	0.462	0.338	0.462	97.053	**100.000**
G.D.S %	**2.947**									



(a) S_A

(b) S_B

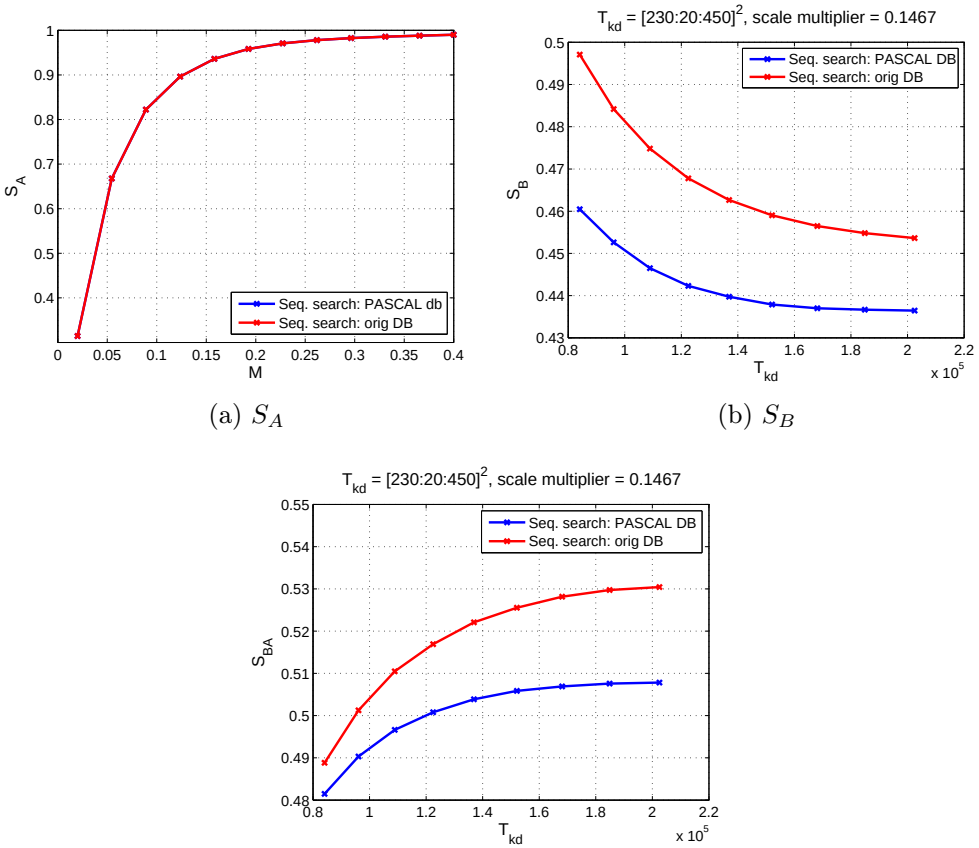


(c) S_{BA}

Fig. 4.1: Properties of test sequences.

4.4 Ground truth

Next, we evaluate the extended database by sequential search which shows the error produced by an extended database. Even more important is, that this evaluation will be used as a ground truth (reference results) for all evaluations of efficient data structures. As same as the time complexity, which will be used to measure the speed up. Our objective is to find data structure and its parameters, which quality criterion will be as close as possible to the sequential search, however the time complexity will be as close as possible to the real–time.



(a) S_A (b) S_B

(c) S_{BA}

Fig. 4.2: Comparison of original and extended database.

It is clear, that the difference between the original, which contains only the reference images (44 647 features) and extended database (1 514 970) is an error produced by used local invariant feature detector and descriptor. For all next experiments, the results obtained for sequential search in an extended database are used. As well, the pure time which was consumed by sequential search (613 346 s. per 40 query images $\Rightarrow$ cca 15 334 s. per image) is used to measure the speed–up factor.

4.5 ANN Search

In this section, we arrange the whole data set into the efficient data structure and evaluate an influence of ϵ parameter of the approximate nearest neighbor search[3] (we use implementation described in 2.4.3). We use KD–tree structure (for all further evaluations), however it is possible to use any other efficient data structures which are (or not) provided by `feature_eval`. Only the KD–tree(s) are used for the following experiments, due to the time complexity of the whole experimental process.

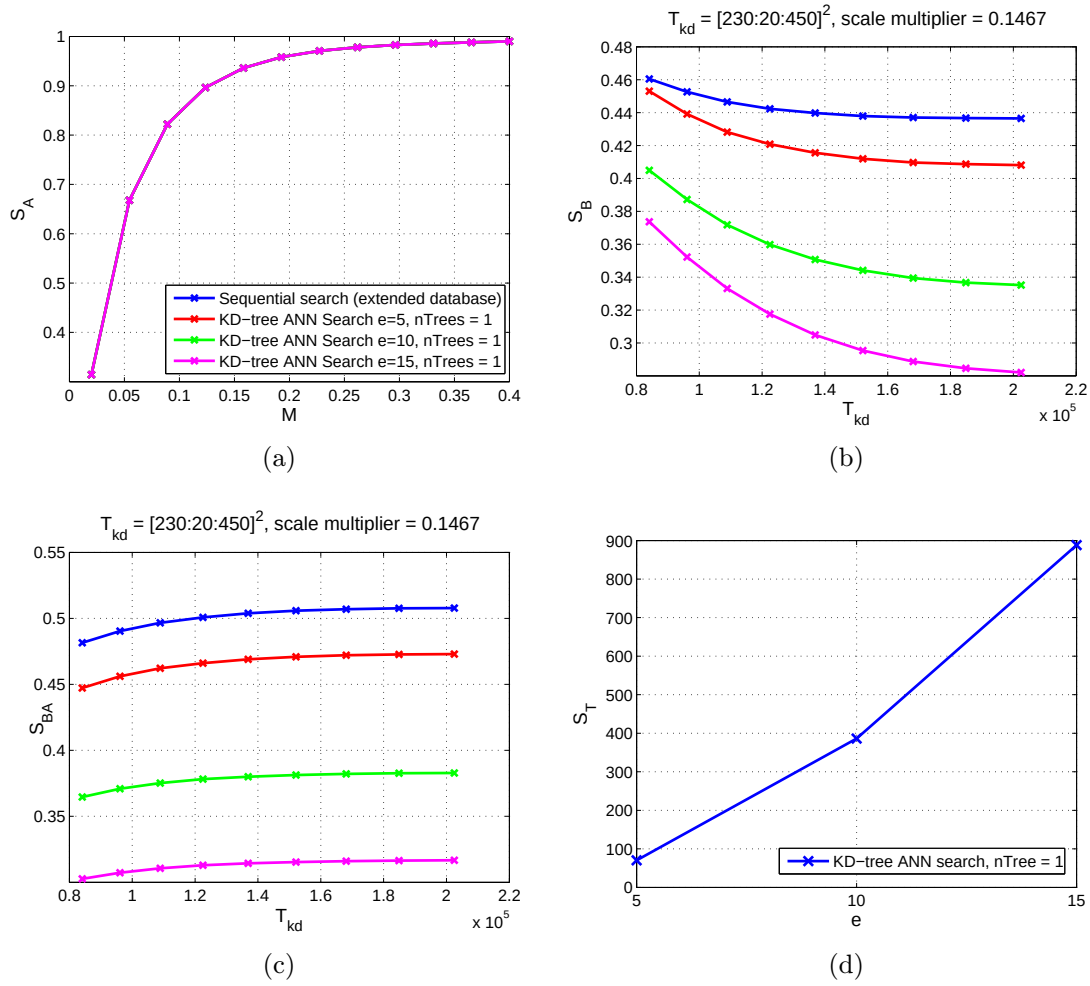


Fig. 4.3: Performance evaluation of a single KD-tree and different ϵ. The figure shows parameters S_A (a), S_B (b), S_{BA} (c) and S_T (d).

[3]The abbreviation ANN is used for ϵ approximate nearest neighbour search in the remaining parts of the thesis.

Tab. 4.3: S_A parameters for different values of ϵ parameter for a ANN search in a single KD–tree.

ϵ	M											
	0.020	0.054	0.089	0.123	0.158	0.192	0.227	0.261	0.296	0.330	0.365	0.400
Sequential	0.314	0.667	0.822	0.896	0.936	0.958	0.970	0.978	0.982	0.985	0.988	0.990
5	0.314	0.667	0.822	0.896	0.936	0.958	0.970	0.978	0.982	0.985	0.988	0.990
10	0.314	0.667	0.822	0.896	0.936	0.958	0.970	0.978	0.982	0.985	0.988	0.990
15	0.314	0.667	0.822	0.896	0.936	0.958	0.970	0.978	0.982	0.985	0.988	0.990

Tab. 4.4: S_B criterion for different values of ϵ parameter for a ANN search in a single KD–tree.

ϵ	t_d								
	84100	96100	108900	122500	136900	152100	168100	184900	202500
Sequential	0.4605	0.4526	0.4465	0.4423	0.4397	0.4379	0.4370	0.4367	0.4365
5	0.4531	0.4391	0.4281	0.4208	0.4156	0.4119	0.4096	0.4087	0.4081
10	0.4049	0.3872	0.3718	0.3598	0.3507	0.3441	0.3394	0.3366	0.3352
15	0.3736	0.3522	0.3331	0.3175	0.3048	0.2954	0.2887	0.2846	0.2820

Tab. 4.5: S_{BA} criterion for different values of ϵ parameter for a ANN search in a single KD–tree. (t_a is set to 0.1467).

ϵ	t_d								
	84100	96100	108900	122500	136900	152100	168100	184900	202500
Sequential	0.4815	0.4903	0.4966	0.5008	0.5039	0.5058	0.5069	0.5076	0.5078
5	0.4473	0.4561	0.4622	0.4661	0.4690	0.4709	0.4720	0.4726	0.4730
10	0.3646	0.3709	0.3752	0.3782	0.3800	0.3813	0.3822	0.3826	0.3829
15	0.3026	0.3072	0.3106	0.3129	0.3143	0.3153	0.3160	0.3165	0.3167

Tab. 4.6: Speed evaluation for different values of ϵ parameter a ANN search in a single KD–tree.

	time (s.)/query set	**avg. time (s.)/image**	**speed–up**
Sequential	613346	15334	———
5	8750	219	70.0925
10	1588	40	386.2040
15	690	17	888.2893

It was observed, that the sufficient accuracy is provided only by $\epsilon = 5$. Also, the speed–up is significant, however the pure time is too far from the real–time. Otherwise ($\epsilon = 10, 15, \ldots$), the speed is better, however is still so far from the real–time. In addition, the accuracy of search is not high.

4.6 Multiple Randomized Tree Structures

As it is shown above, the speedup against the sequential search is significant, however it is still quite far from the real time. In addition, the search with the $\epsilon = 10, 15$ is quite inaccurate. In this section, we try to overcome these limitations of the single KD–tree by a multiple randomized KD–trees.

The idea behind this approach is following: the whole dataset is split into subsets and these subsets are used to create multiple trees. Then, each structure contains less points. Thus, the chance to find faster a nearest neighbour with sufficient distance to query is larger. It is possible to split the points into the subsets in a several ways, in this section, we use only the sparse trees (discussed more in detail in 4.8).

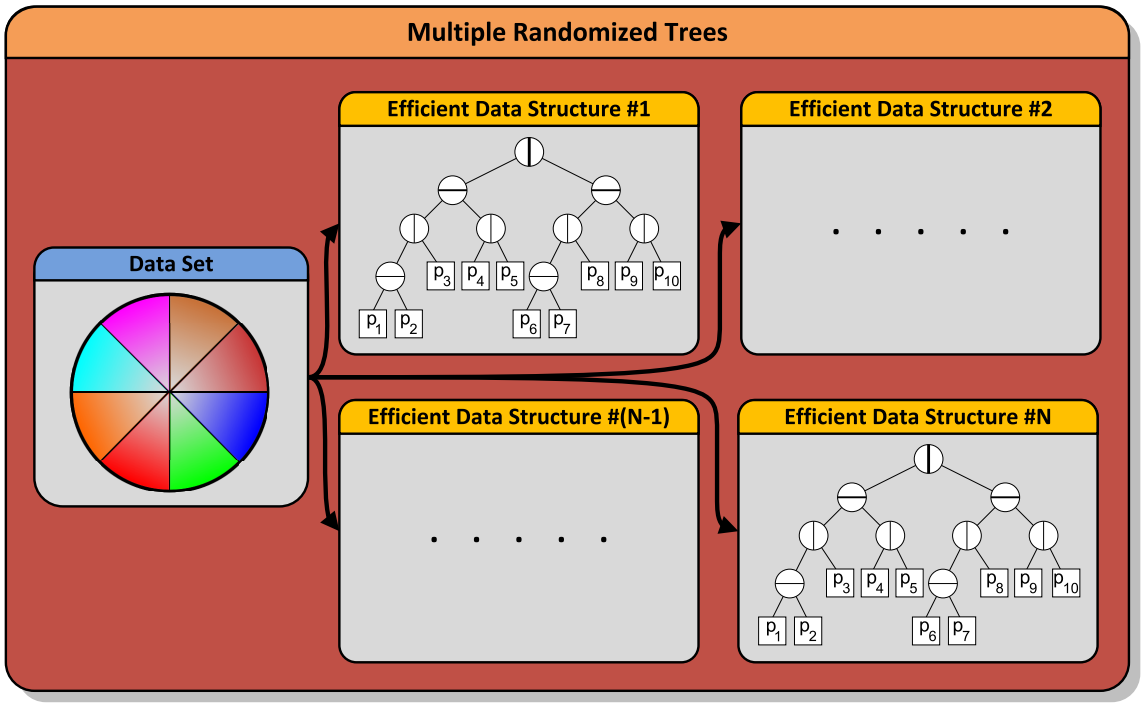
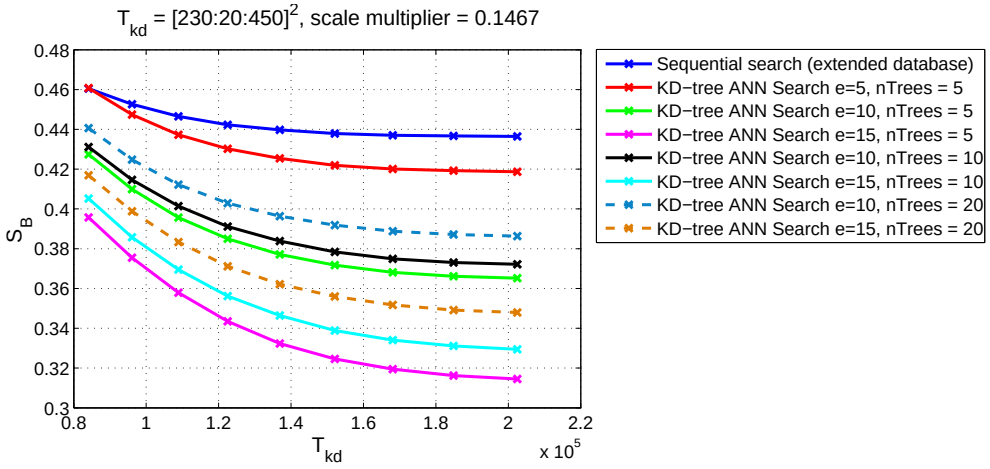
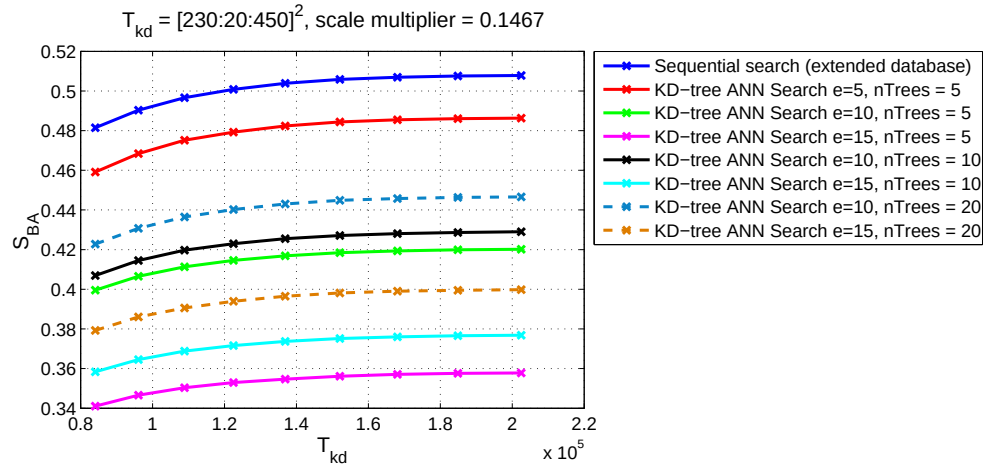


Fig. 4.4: Multiple Randomized Trees – each subset is assigned to another tree structure.

It was observed, that the speed up produced by the multiple randomized trees is not linear. At some point, the response will be even slower, due to the linear filtering of the found nearest neighbors, therefore the real speedup is produced indirectly, because the biggest benefit of the multiple randomized KD–trees is, that the amount of data to be search is reduced by the number of the trees, thus, it is possible to use larger values of ϵ and still have a sufficient accuracy.



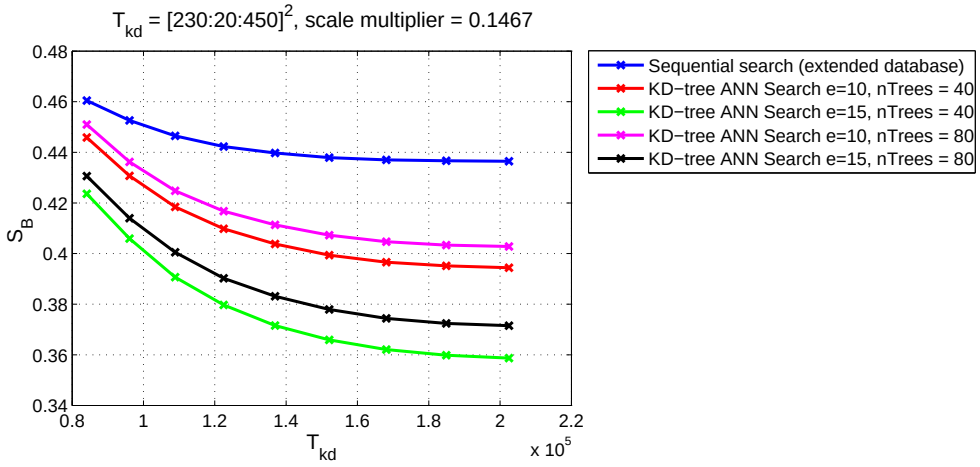
(a) S_B



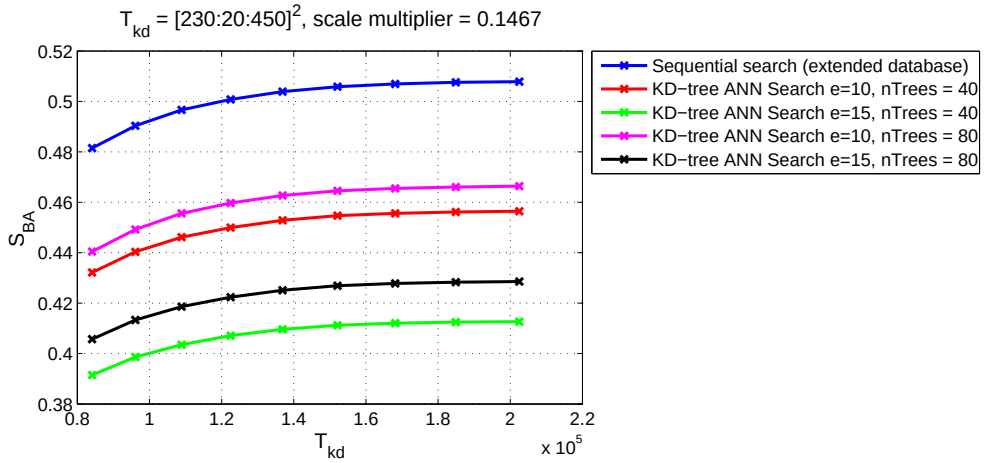
(b) S_{BA}

Fig. 4.5: Performance evaluation of multiple randomized KD–trees and different values of ϵ – S_B (a), S_{BA} (b).

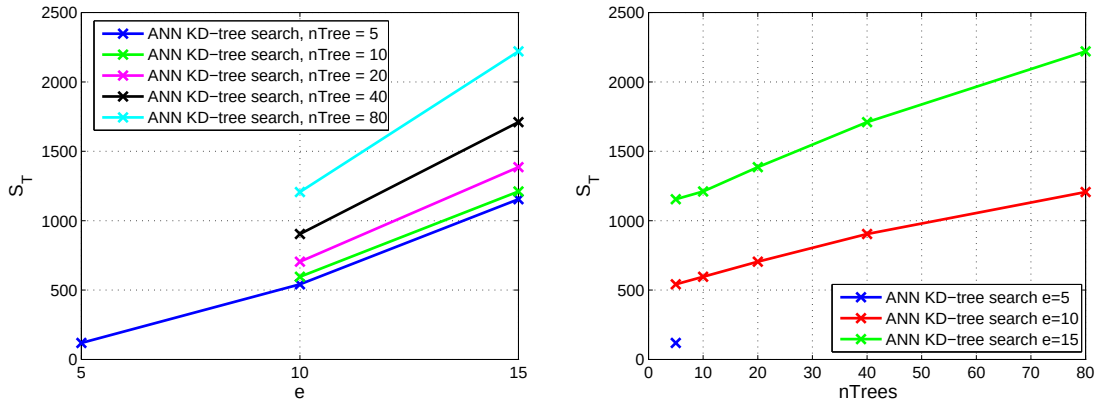
The most significant results are obtained from experiments for 80 multiple randomised trees and $\epsilon = 15$. In this case, both, the precision and recall are fully comparable with 10 trees and $\epsilon = 10$, however, the speed up is $4\times$ larger. By comparison with sequential search, we lose less than 4% in precision and approximately 4% in recall, however the speed up is approximately 2200.



(a)



(b)

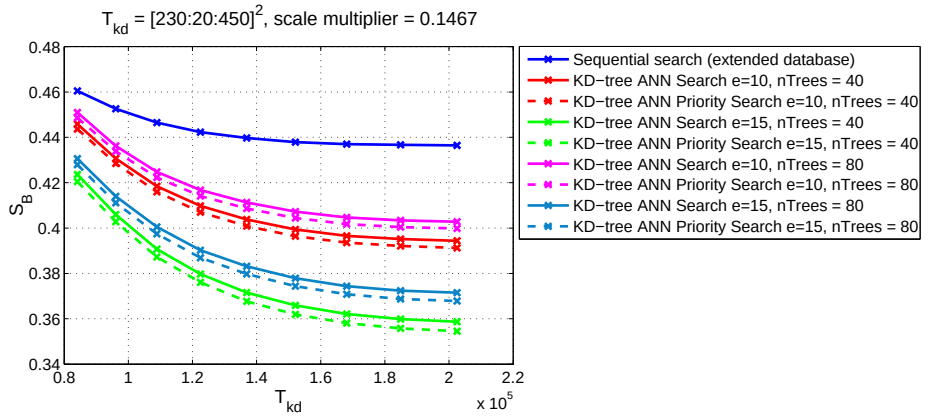


(c) (d)

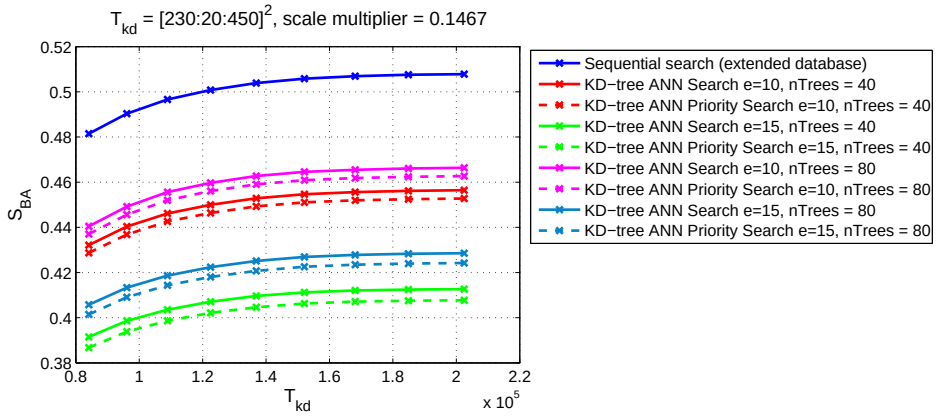
Fig. 4.6: Performance evaluation of multiple randomized KD–trees and different values of ϵ – S_B (a), S_{BA} (b) and speed up S_T with respect to ϵ (c) and number of trees (d).

4.7 Priority Search

The next evaluation is a comparison of the standard ANN search and Priority search 2.4.3. The Priority search should be faster, due to the usage of priority queues, however it was observed from the measurements that the priority search has the both results, the quality criterion and the time measurement worse.

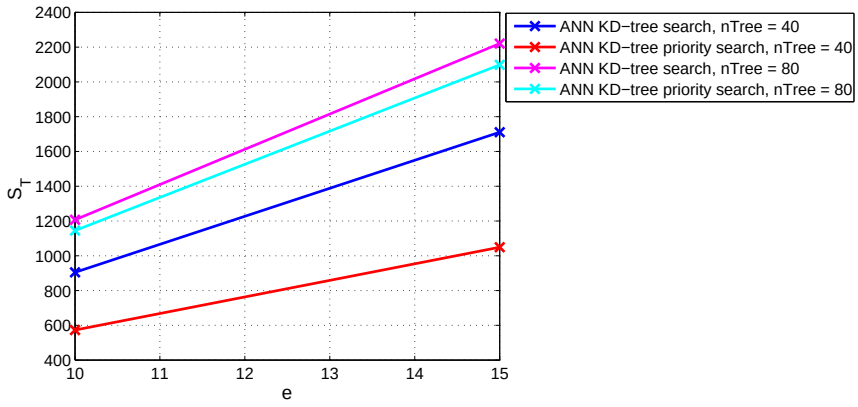


(a)

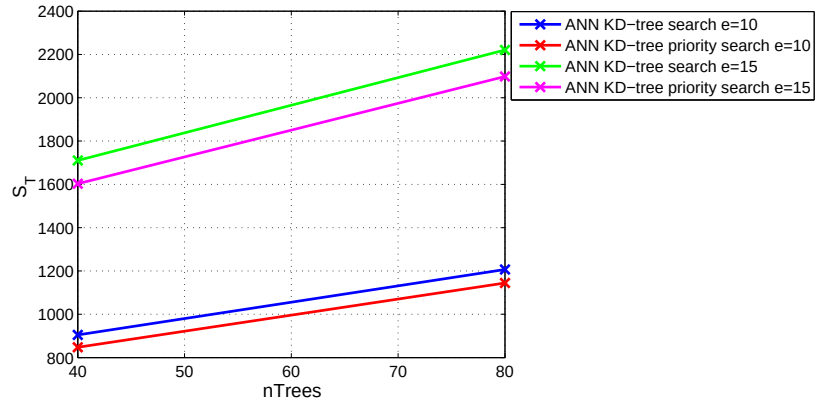


(b)

Fig. 4.7: Performance evaluation of an ANN priority search. The figures show parameters S_B (a), S_{BA} (b).



(a)



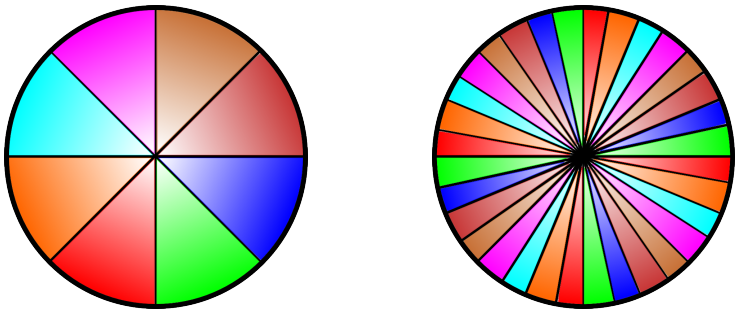
(b)

Fig. 4.8: Performance evaluation of an ANN priority search. The figures show parameter S_T with respect to ϵ (a) and number of trees (b).

It was observed, that for all tested number of trees and values of ϵ, the priority search has worse results than standard search. The differences are not large, however priority search has even smaller speed up for all configurations of tested parameters. Consequently, it is better to use standard ϵ ANN search instead of priority search.

4.8 Data Distributions

The next evaluation is an experiment with the distribution of the data into the multiple trees. As a novelty, we introduce multiple randomised sparse trees which were used for all experiments discussed above. In this section, we compare the results with the results obtained for a compact distribution.

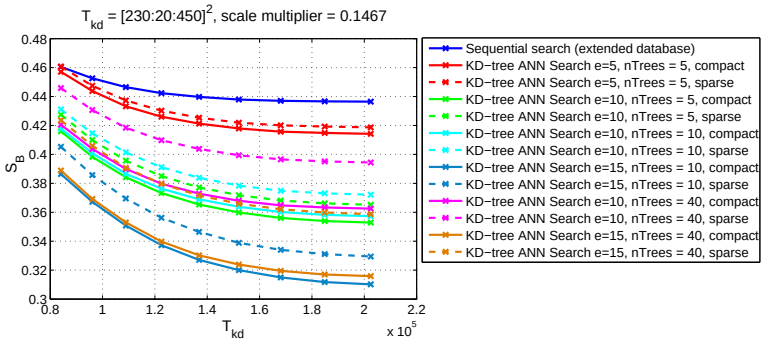


(a) (b)

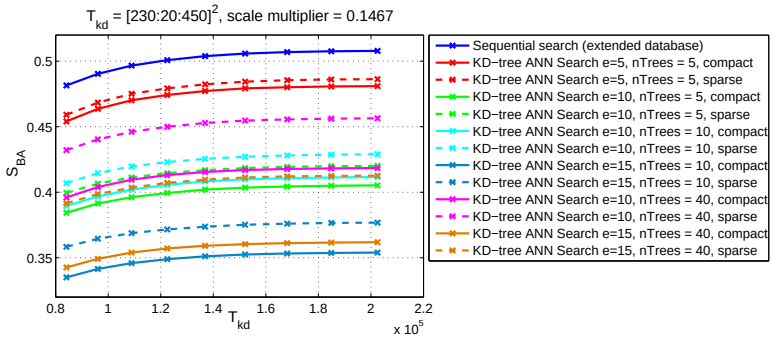
Fig. 4.9: Two different options of dividing input data set into subsets are shown: compact trees (a) and sparse trees (b).

In the case of sparse trees, every n–th feature is assigned to a different subset. By comparison, in the case of compact trees, the dataset is simply divided by a factor n to create the subsets which are used for multiple randomised trees. Theoretically, in the first case, the subsets should have larger variance.

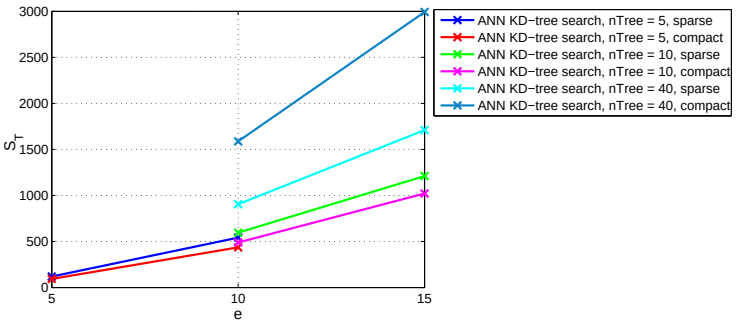
It was observed, that the introduced sparse trees outperform the compact trees in all precision aspects. The time measurements shown, that the compact trees have better results for the high number of trees, however sparse trees showed better results for the less number of trees. It may be justified, because in the case of sparse trees, the features are equally divided in all subsets, so it took more time to get the true nearest neighbor. However, this drawback is overcome, because sparse trees produce better results for higher ϵ (for example 40 sparse trees with $\epsilon = 15$ produce almost the same results as 40 compact trees with $\epsilon = 10$).



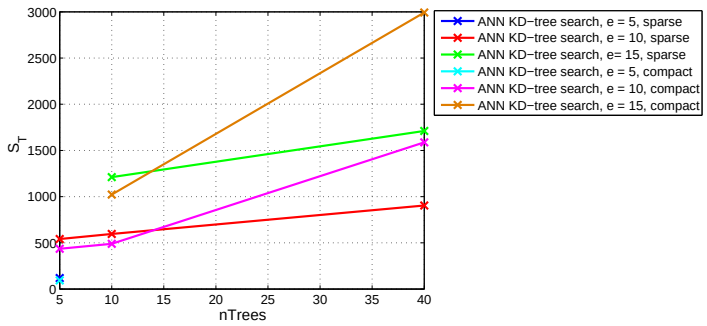
(a)



(b)



(c)



(d)

Fig. 4.10: Comparison of a performance of sparse and compact trees. The figures show parameters S_B (a), S_{BA} (b), S_T with respect to ϵ (c) and number of trees (d).

4.9 Descriptor Dimensionality

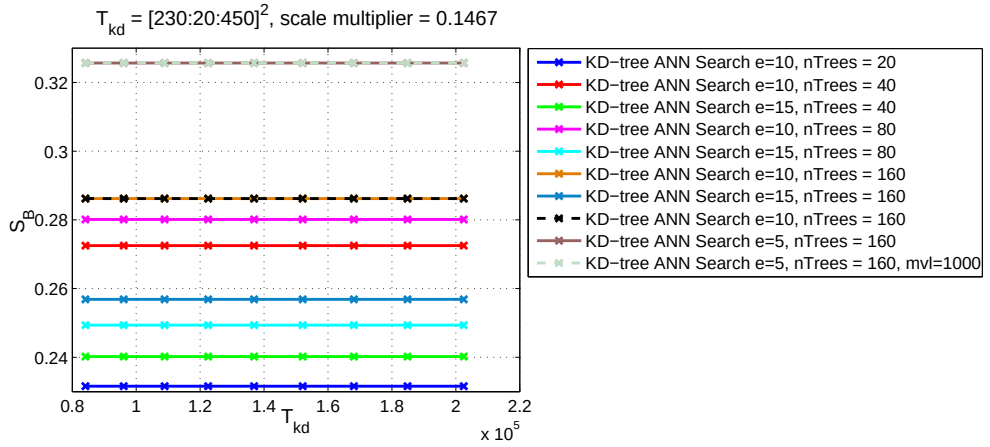
As it was shown above, the speed up produced by a multiple randomised trees is significant, however it is still so far from the real-time. All of the experiments was done with a 192 dimensional C–SIFT descriptor. Search in a highly dimensional data is a time expensive even for efficient data structures. Fortunately, it is possible to use another, less dimensional descriptor (PCA–SIFT, ...).

In this section, we evaluate the improvements produced by a less dimensional descriptor. Only the first 50 dimensions of the C–SIFT descriptors are used in this experiment instead of a completely different descriptor. Consequently, the quality criterion will be a little bit worse, however it is sufficient for this experiment, because the results are compared with the sequential search for 50 dimensions and the most important is a speed–up factor.

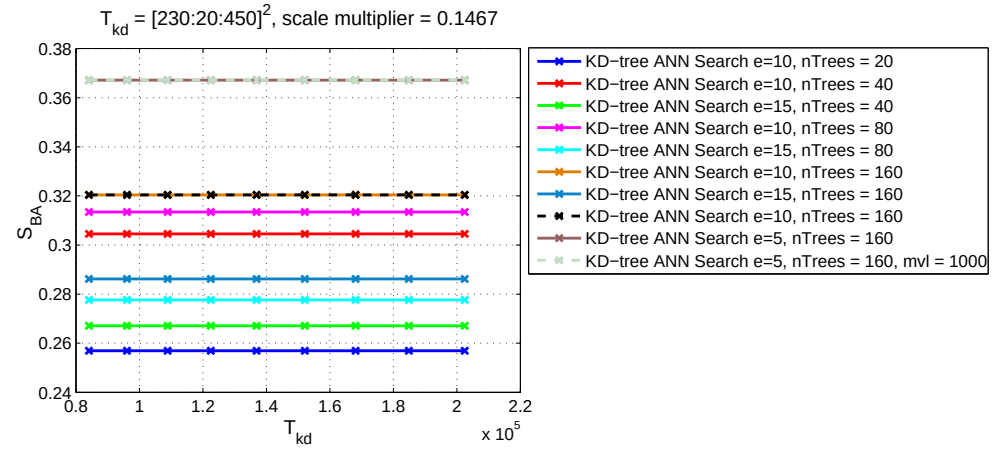
It was observed that it is possible to get the real–time response with sufficient precision of matching for 160 multiple randomised trees with $\epsilon = 5$ and maximum of visited leaves (cf. 4.10) set to 1000 for less dimensional descriptor (50). However, in this experiment was used only first 50 dimensions of 192 dimensional C–SIFT, for appropriate descriptor, the results will be even better.

Tab. 4.7: Speed measurements for a less dimensional descriptor.

nTrees	ϵ	**time (s.)/query set**	**avg. time (s.)/nTrees**	**1 image (s.)/nTrees**
40	**5**	2007.26	50.1815	1.254
40	**10**	662.34	16.5585	0.413
40	**15**	400.94	10.0235	0.250
80	**5**	3198.6	39.9825	0.999
80	**10**	1089.54	13.6192	0.340
80	**15**	678.38	8.47975	0.211
160	**5**	5021.58	31.3849	0.784
160	**10**	1772	11.075	0.276
160	**15**	1132.52	7.07825	0.176
160 – 1000 mvl	**5**	2991.97	18.6998	0.467
160 – 1000 mvl	**10**	1287.68	8.048	0.201
160 – 1000 mvl	**15**	1132.52	7.07825	0.177



(a)



(b)

Fig. 4.11: Performance evaluation of a less dimensional descriptor. The figure shows parameter S_B (a) and S_{BA} (b).

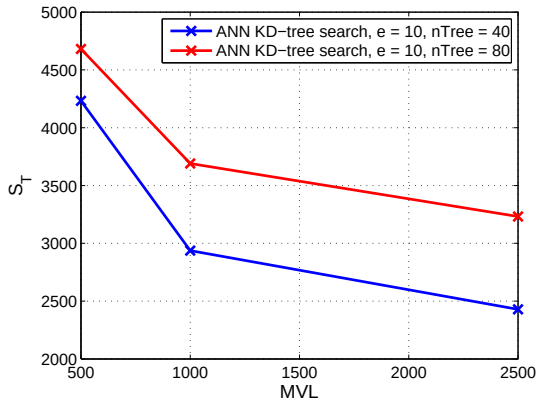
4.10 Maximum Visited Leaves

The both, standard ANN search as well ANN priority search terminates after the sufficient distance is found. However nowhere is defined, how much time the search takes and it may be serious problem for the real–time applications (even for the soft real–time). The maximum time complexity can be determined indirectly by the maximum visited leaves property.

The main idea of this approach is, that the true nearest neighbours should have closer distance to the query points and the search should terminates quite fast. However, much time is wasted by the search of the nearest neighbors of queries which do not have distinctive descriptors and the distance from their nearest neighbors to them are large. Moreover, this search is useless, because the probability

that the false nearest neighbour will be found is high. Instead of this, a modified search with maximum visited leaves property terminates in both cases, when the distance from the nearest neighbor to the query is quite close as well the maximum of visited leaves is reached.

It was observed that with parameter maximum visited leaves set to 1000, we lost only cca 1%, however the speed–up is 4× larger for $\epsilon = 10$.

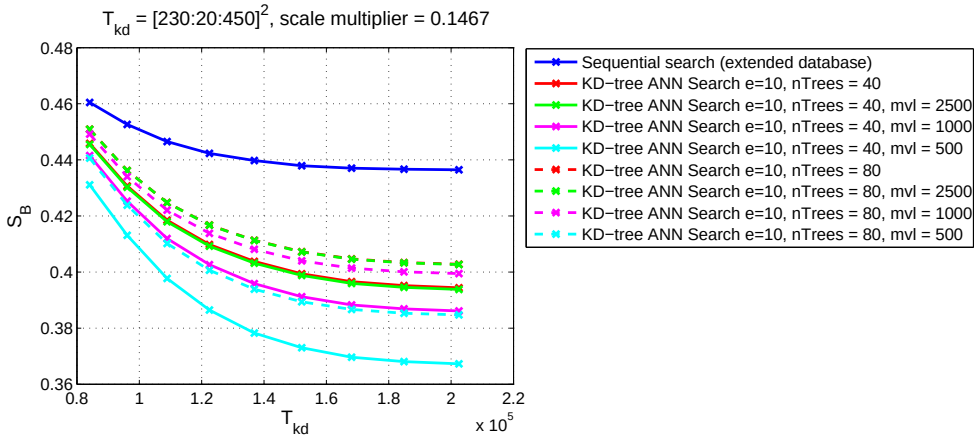


(a)

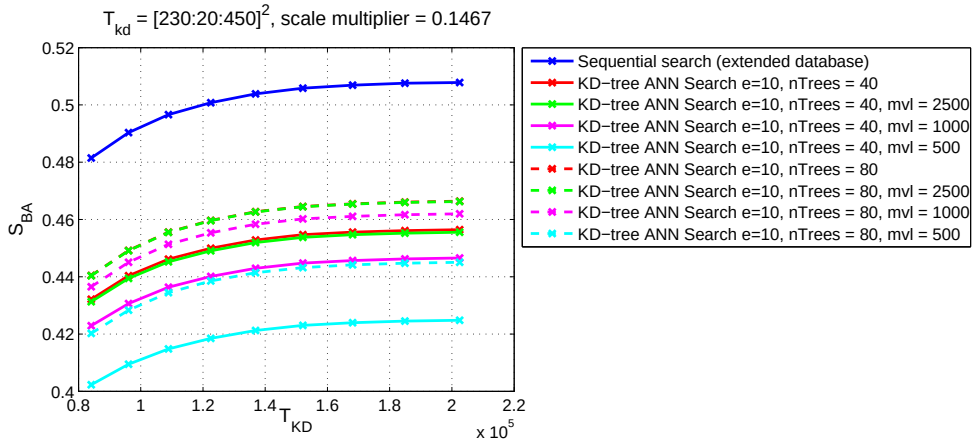
Fig. 4.12: Performance evaluation of a different number of a maximum visited leaves. The figures show parameter S_T with respect to ϵ (a) and number of trees (b).

Tab. 4.8: Time measurements for different maximum visited leaves (all test set)

ϵ	**Number of Trees**	**M.V.L.**	**Time (s.)**	**Time (s.) / nTrees**
10	40	—	27125.90	678.15
10	40	2500	10100.70	252.52
10	40	1000	8354.06	208.85
10	40	500	5796.92	144.92
10	80	—	40654.80	508.19
10	80	2500	15183.40	189.79
10	80	1000	13296.60	166.21
10	80	500	10481.70	131.02



(a)

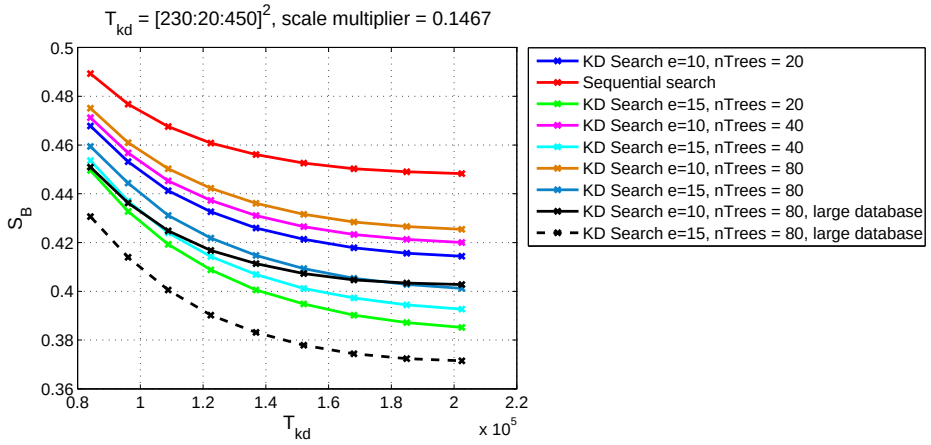


(b)

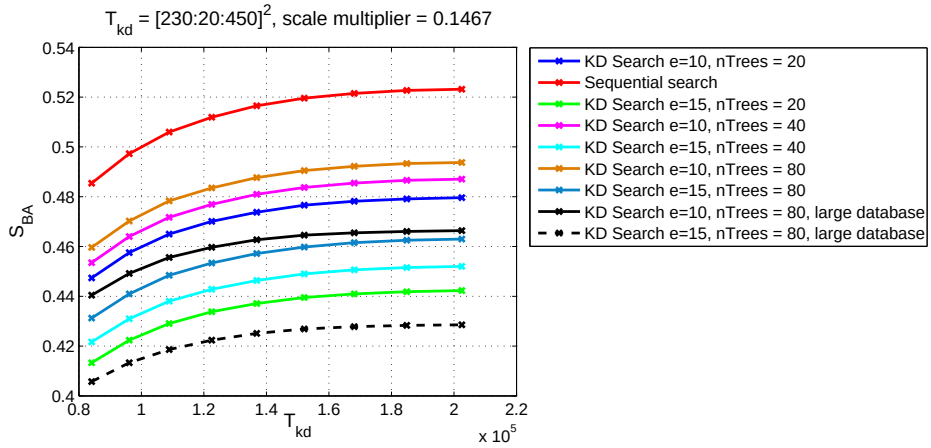
Fig. 4.13: Performance evaluation of a different number of a maximum visited leaves. The figures show parameters S_B (a) and S_{BA} (b).

4.11 Database Size

The next evaluation deals with the influence of the database size. The biggest database used in the thesis contains cca 1 514 970 features extracted from 8 reference and 1 000 other images. The smaller database contains only 190 937 features. It is important to know, how the speed and precision depends on the size, because we need to be able to decide whether it is enough to create one large database or is it better for larger datasets to create more databases and run the search in parallel databases.



(a)



(b)

Fig. 4.14: Performance evaluation of a smaller database. The figures show parameters S_B (a) and S_{BA} (b).

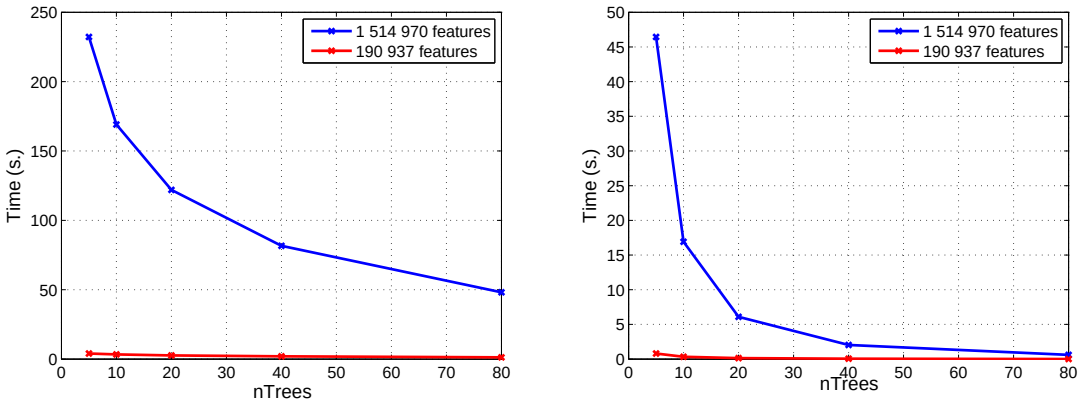
It was observed, that the results depends on database size. It should be a point at discussion for extremely large databases, whether it is or not useful to split them into more subdatabases.

Tab. 4.9: Comparison of large and small databases

Features	**nTrees**	ϵ	**time (s.)**	**time (s.) /nTrees**
190937	20	10	2802.83	140.14
190937	20	15	1653.55	82.68
190937	40	10	3497.67	87.44
190937	40	10	2067.95	51.70
190937	80	10	4659.70	58.25
190937	80	15	2740.10	34.25
1514970	20	10	17404.50	870.23
1514970	20	15	8851.20	442.56
1514970	40	10	27125.90	678.15
1514970	40	10	14342.50	358.56
1514970	80	10	40654.80	508.19
1514970	80	15	22097.30	276.22

4.12 Tree Construction Time

Our last evaluation aims at the time needed for preprocessing of the dataset during the construction of efficient data structures. It was observed, that this criterion is not so important for KD–trees or BBD–trees, which are build quite fast. Moreover, the trees are precomputed off–line, so the speed is not really an important parameter, however it can be useful for some future comparison of a different tree structures like k–means trees which construction process is more time consuming.



(a) (b)

Fig. 4.15: Construction time of efficient data structures for different number of points. The figure (a) shows pure time needed for construction of multiple trees for different databases, figure (b) shows time needed for construction divided by number of trees.

The time needed for construction strongly (indirectly) depends on number of trees which should be stored in a data structure, however it is not an important parameter for KD–trees, because all measured times are quite far. Moreover it is not the most important criterion in general (even for k–means), because the trees may be precomputed in off–line.

5 DISCUSSION AND CONCLUSIONS

The thesis deals with fast feature matching for simultaneous localization and mapping. The goal was to create a system with real–time response which can be used in various applications.

We have developed a new performance evaluation system, which can be used to determine appropriate parameters. The linear time complexity of the brute–force search was replaced by sub–linear which is produced by different efficient data structures like KD–trees, BBD-trees or k–means trees.

Several experiments are described and evaluated. It was observed, that the multiple randomised KD–trees outperform the linear search in time complexity with sufficient precision. However the pure times are still quite far from the real–time. We have shown, that it is possible to get close to the real–time response by setting the maximum visited leaves to 1000. As a novelty, we have shown that the sparse trees completely outperform the compact trees in precision criterion which allows us to use a larger ϵ parameter.

An impressive real–time response can be achieved by ϵ–ANN search in multiple randomised sparse trees constructed by a less dimensional descriptors with setting the maximum visited leaves. In addition, the new performance evaluation framework was developed and is available for general use.

For the future work, it may be interesting to start thinking about the trees with weighted points or adaptive trees with exponential oblivious, which will be created by dynamically added points into the data structure. Moreover, it could be interesting to test different detectors or descriptors, however it is a time expensive job which should be done by the whole community to create the full report.

BIBLIOGRAPHY

[1] Abdel-Hakim, A. E.; Farag, A. A.: CSIFT: A SIFT Descriptor with Color Invariant Characteristics. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA: IEEE Computer Society, 2006, ISBN 0-7695-2597-0, pages 1978–1983.

[2] Arya, S.; Mount, D. M.: Algorithms for Fast Vector Quantization. In *Proc. of DCC'93: Data Compression Conference*, IEEE Press, 1993, pages 381–390.

[3] Arya, S.; Mount, D. M.: Approximate nearest neighbor queries in fixed dimensions. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1993, ISBN 0-89871-313-7, pages 271–280.

[4] Arya, S.; Mount, D. M.; Netanyahu, N. S.; et al.: An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. In *ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS*, 1994, pages 573–582.

[5] Attneave, F.: Some Informational Aspects of Visual Perception. *Psychological Review*, volume 61, nr. 3, 1954: pages 183–193.

[6] Bailey, T.; Durrant-Whyte, H.: Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics & Automation Magazine*, volume 13, nr. 3, September 2006: pages 108–117, ISSN 1070-9932, doi:10.1109/MRA.2006.1678144.

[7] Bastanlar, Y.; Yilmaz, E.; Yardimci, Y.; et al.: 3D Reconstruction for a Cultural Heritage Virtual Tour System. 2008, page B5: 1023 ff.

[8] Bay, H.; Ess, A.; Tuytelaars, T.; et al.: Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, volume 110, nr. 3, 2008: pages 346–359, ISSN 1077-3142.

[9] Bentley, J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM*, volume 18, nr. 9, 1975: pages 509–517, ISSN 0001-0782.

[10] Berg, M. d.; Cheong, O.; Kreveld, M. v.; et al.: *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008, ISBN 3540779736, 9783540779735.

[11] Biederman, I.: Recognition-by-components: A theory of human image understanding. *Psychological Review*, volume 94, 1987: pages 115–147.

[12] Borenstein, J.; Everett, H. R.; Feng, L.: *Where am I? Systems and Methods for Mobile Robot Positioning*. March 1996.

[13] Broggi, A.; Bertozzi, M.; Fascioli, A.; et al.: The argo autonomous vehicle's vision and control systems. *International Journal of Intelligent Control and Systems*, 1999: pages 409–441.

[14] Brown, M.; Lowe, D. G.: Recognising Panoramas. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, Washington, DC, USA: IEEE Computer Society, 2003, ISBN 0-7695-1950-4, page 1218.

[15] Chum, O.; Matas, J.: Web Scale Image Clustering – Large Scale Discovery of Spatially Related Images. 2008.

[16] Datar, M.; Indyk, P.: Locality-sensitive hashing scheme based on p-stable distributions. In *In SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, ACM Press, 2004, pages 253–262.

[17] Duda, R. O.; Hart, P. E.; Stork, D. G.: *Pattern Classification*. Wiley-Interscience, second edition, 2000, ISBN 978-0471056690, 654 pages.

[18] Durrant-Whyte, H.; Bailey, T.: Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, volume 2, 2006: page 2006.

[19] Everingham, M.; Van Gool, L.; Williams, C. K. I.; et al.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[20] Friedman, J. H.; Bentley, J. L.; Finkel, R. A.: An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, volume 3, nr. 3, 1977: pages 209–226, ISSN 0098-3500.

[21] Gionis, A.; Indyk, P.; Motwani, R.: Similarity Search in High Dimensions via Hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, ISBN 1-55860-615-7, pages 518–529.

[22] Gool, L. J. V.; Moons, T.; Ungureanu, D.: Affine/ Photometric Invariants for Planar Intensity Patterns. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, London, UK: Springer-Verlag, 1996, ISBN 3-540-61122-3, pages 642–651.

[23] Harris, C.; Stephens, M.: A combined corner and edge detector. In *Proc. Fourth Alvey Vision Conference*, 1988, pages 147–151.

[24] Indyk, P.; Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, New York, NY, USA: ACM, 1998, ISBN 0-89791-962-9, pages 604–613.

[25] Kalal, Z.; Matas, J.; Mikolajczyk, K.: P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints. *CVPR*, 2010.

[26] Ke, Y.; Sukthankar, R.: PCA-SIFT: A More Distinctive Representation for Local Image Descriptors. 2004, pages 506–513.

[27] Kleinberg, J. M.: Two Algorithms for Nearest-Neighbor Search in High Dimensions. 1997, pages 599–608.

[28] Lamrous, S.; Taileb, M.: Divisive Hierarchical K-Means. In *CIMCA '06: Proceedings of the International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce*, Washington, DC, USA: IEEE Computer Society, 2006, ISBN 0-7695-2731-0, page 18.

[29] Lazebnik, S.; Schmid, C.; Ponce, J.: Sparse Texture Representation Using Affine-Invariant Neighborhoods. In *International Conference on Computer Vision & Pattern Recognition*, volume 2, 2003, pages 319–324.
URL `http://lear.inrialpes.fr/pubs/2003/LSP03`

[30] Leibe, B.; Mikolajczyk, K.; Schiele, B.: Efficient Clustering and Matching for Object Class Recognition. In *British Machine Vision Conference (BMVC'06)*, 2006.

[31] Lindeberg, T.: Feature Detection with Automatic Scale Selection. *International Journal of Computer Vision*, volume 30, 1998: pages 79–116.

[32] Lowe, D. G.: Object Recognition from Local Scale-Invariant Features. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, Washington, DC, USA: IEEE Computer Society, 1999, ISBN 0-7695-0164-8, page 1150.

[33] Lowe, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, volume 60, nr. 2, 2004: pages 91–110, ISSN 0920-5691.

[34] Lv, Q.; Josephson, W.; Wang, Z.; et al.: Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, ISBN 978-1-59593-649-3, pages 950–961.
URL `http://portal.acm.org/citation.cfm?id=1325851.1325958`

[35] Maneewongvatana, S.; Mount, D. M.: It's Okay to Be Skinny, If Your Friends Are Fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*, 1999.

[36] Mars Exploration Rover Mision. [online], Date of visit to site 10.5.2010.
URL `http://marsrovers.nasa.gov/home/`

[37] Matas, J.; Chum, O.; Urban, M.; et al.: Robust Wide Baseline Stereo from Maximally Stable Extremal. In *In British Machine Vision Conference*, 2002, pages 384–393.

[38] Mikolajczyk, K.; Matas, J.: Improving SIFT for Fast Tree Matching by Optimal Linear Projection. In *ICCV 2007: Proceedings of Eleventh IEEE International Conference on Computer Vision*, edited D. Metaxas; B. Vemuri; A. Shashua; H. Shum, IEEE Computer Society, Los Alamitos, USA: IEEE Computer Society Press, October 2007, ISBN 978-1-4244-1631-8, page 8, cDROM.

[39] Mikolajczyk, K.; Schmid, C.: Indexing based on scale invariant interest points. In *Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada*, 2001, pages 525–531.
URL `http://perception.inrialpes.fr/Publications/2001/MS01a`

[40] Mikolajczyk, K.; Schmid, C.: An affine invariant interest point detector. In *Proceedings of the 7th European Conference on Computer Vision, Copenhagen, Denmark*, Springer, 2002, pages 128–142, copenhagen.
URL `http://perception.inrialpes.fr/Publications/2002/MS02`

[41] Mikolajczyk, K.; Schmid, C.: Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, volume 60, nr. 1, 2004: pages 63–86, ISSN 0920-5691.
URL `http://lear.inrialpes.fr/pubs/2004/MS04`

[42] Mikolajczyk, K.; Schmid, C.: A Performance Evaluation of Local Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 27, nr. 10, 2005: pages 1615–1630, ISSN 0162-8828.

[43] Mikolajczyk, K.; Tuytelaars, T.; Schmid, C.; et al.: A Comparison of Affine Region Detectors. *Int. J. Comput. Vision*, volume 65, nr. 1-2, 2005: pages 43–72, ISSN 0920-5691.

[44] Mindru, F.; Tuytelaars, T.; Gool, L. V.; et al.: Moment Invariants for Recognition under Changing Viewpoint and Illumination. *Comput. Vis. Imag Underst*, volume 94, 2004: pages 3–27.

[45] Montemerlo, M.; Thrun, S.; Koller, D.; et al.: FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *In Proc. of the Int. Conf. on Artificial Intelligence (IJCAI*, 2003, pages 1151–1156.

[46] Moore, A. W.: The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, ISBN 1-55860-709-9, pages 397–405.

[47] Moravec, H.: The Stanford Cart and the CMU Rover. *Proceedings of the IEEE*, volume 71, nr. 7, july 1983: pages 872 – 884, ISSN 0018-9219.

[48] Mount, D. M.: ANN Programming Manual. Technical report, 2010.
URL `http://www.cs.umd.edu/~mount/ANN/`

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

[50] Orpheus Robotic System Project. [online], Date of visit to site 10.5.2010.
URL `http://www.orpheus-project.cz/`

[51] Panoramio. [online], Date of visit to site 10.5.2010.
URL `http://www.panoramio.com`

[52] Pomerleau, D.: RALPH: Rapidly Adapting Lateral Position Handler. In *IEEE Symposium on Intelligent Vehicles*, September 1995, pages 506 – 511.

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

[54] Schaffalitzky, F.; Zisserman, A.: Multi-view Matching for Unordered Image Sets, or "How Do I Organize My Holiday Snaps?". In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, London, UK: Springer-Verlag, 2002, ISBN 3-540-43745-2, pages 414–431.

[55] Schmid, C.; Mohr, R.: Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 19, 1997: pages 530–535.

[56] Se, S.; Lowe, D.; Little, J.: Global Localization using Distinctive Visual Features. 2002.

[57] Shi, J.; Tomasi, C.: Good Features to Track. 1994, pages 593–600.

[58] Silpa-Anan, C.; Hartley, R.: Optimised KD-trees for fast image descriptor matching. In *CVPR*, IEEE Computer Society, 2008.

[59] Sivic, J.; Zisserman, A.: Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, October 2003, pages 1470–1477.
URL `http://www.robots.ox.ac.uk/~vgg`

[60] Smith, S. M.; Brady, J. M.: SUSAN - A New Approach to Low Level Image Processing. *International Journal of Computer Vision*, volume 23, 1995: pages 45–78.

[61] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processing, Analysis and Machine Vision*. Thomson, third edition, 2007, ISBN 978-0-495-08252.

[62] Stanford Racing. [online], Date of visit to site 10.5.2010.
URL `http://cs.stanford.edu/group/roadrunner//old/index.html`

[63] Svab, J.; Krajnik, T.; J.Faigl; et al.: FPGA-based Speeded Up Robust Features. In *2009 IEEE International Conference on Technologies for Practical Robot Applications 2009*, November 2009.

[64] Tartan Racing. [online], Date of visit to site 10.5.2010.
URL `http://www.tartanracing.org/`

[65] Thrun, S.; Burgard, W.; Fox, D.: *Probabilistic Robotics*. The MIT Press, 2005, ISBN 978-0-262-20162-9.

[66] Tuytelaars, T.; Mikolajczyk, K.: Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, volume 3, nr. 3, 2008: pages 177–280, ISSN 1572-2740.

[67] Tuytelaars, T.; Van Gool, L.: Matching Widely Separated Views Based on Affine Invariant Regions. *Int. J. Comput. Vision*, volume 59, nr. 1, 2004: pages 61–85, ISSN 0920-5691.

[68] Wang, Q.; You, S.: Fast Similarity Search for High-Dimensional Dataset. In *ISM '06: Proceedings of the Eighth IEEE International Symposium on Multimedia*, Washington, DC, USA: IEEE Computer Society, 2006, ISBN 0-7695-2746-9, pages 799–804.

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

$\mathbf{M}$	Matrix
$\mathcal{F}$	Set
$\mathbf{v}$	n–dimensional vector
$\mathbb{R}^d$	d–dimensional vector space
$\mathcal{O}$	Upper bound of the algorithm's running time
$\det \mathbf{A}$	Determinant of $\mathbf{A}$
$\mathrm{tr}\mathbf{A}$	Trace of $\mathbf{A}$
ANN	Approximate Nearest Neighbor
BBD	Balanced Box–Decomposition
FAST	Features from Accelerated Segment Test
GLOH	Gradient Location–Orientation Histogram
LSH	Locality Sensitive Hashing
MSER	Maximally Stable Extremal Regions
PCA	Principal Component Analyses
SIFT	Scale Invariant Feature Transformation
SLAM	Simultaneous Localization And Mapping
SURF	Speeded Up Robust Features

LIST OF APPENDICES

A TOOLS

In this appendix, we summarise and briefly describe tools, which we have used to obtain results discussed in the thesis. Both, the `compute_descriptors` and `feature_eval` are written in C++ and can be run on any Windows or Linux machine.

A.1 `compute_descriptors`

The application `compute_descriptors` has been implemented by Mikolajczyk[1]. We have used this application to detect and describe local invariant features in images. The application provides several feature detectors (Harris–Laplace/Affine, Hessian–Laplace/Affine, ...) and descriptors (SIFT, GLOH, CSIFT, ...).

A.2 `feature_eval`

The application `feature_eval` is available on enclosed DVD. It provides fast feature matching of local invariant features. The application is based on ANN [48] and FLANN [49] libraries.

WARNING: be sure that your computer has enough free memory! Application may run long time (depends on database size and search options).

Application provides multiple randomized KD–trees, BBD–trees and k–means trees. Standard (approximate nearest neighbour) search is available as same as priority search and brute–force (sequential) search. To see all options, run the application without any modifier.

[1] Available at http://www.featurespace.org

A.2.1 Usage

The most important options are (in addition to that, it is possible to use different splitting and shrinking rules, k–means trees, ...):

`-nt` – number of multiple randomized trees (default = 1)

`-nn` – number of nearest neighbors which are found for each query (default = 1)

`-e` – the error bound for approximate nearest neighbor search (default = 0)

`-mps` – number of points to visit before termination (default = -1, denotes the natural termination condition)

`-sqr` – use 0 for square and 1 for square root of Euclidean norm (default = 1)

`-kd` – KD–tree standard search

`-kdp` – KD–tree priority search

`-sf` – brute–force search

`-dbf` – file with the list of files containing reference features

`-df` – file which contains database features

`-qf` – file which contains list files containing query features

Firstly, it is necessary to build the database. All other runs, which use the same database do not need to build the database again, but can be used directly.

Example: the following command loads list of files (`files.ref`) containing reference features and stores it to the file `database.pts` (for next runs, it is not necessary to use `-dbf` option again). The application creates 40 randomised multiple sparse trees. Then, the all files stored in `files.qrf` are loaded and each feature is used to search its approximate nearest neighbor with $\epsilon = 10$. The standard, priority and brute–force search are used. The measured distance is square of Euclidean distance. The time measurement are stored in file `files.qrf.tms`, the names of files containing fast feature matches are distinguish by their name, name of database and parameters:

```
ann_sample.exe -dbf files.ref -df database.pts -qf files.qrf -nt 40 -e
10 -sqr 0 -kd -kdp -sf
```

A.2.2 Input and Output Files

Application uses several files and expects that a few rules are followed.

The file `files.red` contains a pure list of files with features which should be stored in a database. The order of files is important, because it denotes ID_{SR}

```
1  data/bark/out1.desc
2  data/bikes/out1.desc
3  data/boat/out1.desc
4  data/graff/out1.desc
5  data/light/out1.desc
6  data/trees/out1.desc
7  data/ubc/out1.desc
8  data/wall/out1.desc
9  database/desc/db_000001.desc
10 database/desc/db_000002.desc
11 database/desc/db_000003.desc
12 database/desc/db_000004.desc
13 .........
```

Listing A.1: Definition of database files

The database file has the following structure:

```
1 #comments referenceImage psize dsize x y cornerness scale angle ....
2 59715
3 39
4 192
5 1 38 192 728.557 490.751 9.42557e+006 72.576 ...... 17 35 92 79 45 ......
6 1 38 192 729.723 491.81 9.19975e+006 60.48 ...... 13 37 104 87 ......
7 ...........
```

Listing A.2: File with features

Line 1: comments
Line 2: number of features stored in database
Line 3: number of parameters
Line 4: number of descriptors
Line 5, 6, ... 59715: the first number is ID of sequence followed by 39 parameters and 192 descriptors of feature

The last important file is `files.qrf` contains the list of query files. The order of files denotes the ID of sequence. The sequences are separated by the character #.

```
1  #
2  data/bark/out2.desc
3  data/bark/out3.desc
4  data/bark/out4.desc
5  data/bark/out5.desc
6  data/bark/out6.desc
7  #
8  data/bikes/out2.desc
9  data/bikes/out3.desc
10 data/bikes/out4.desc
11 data/bikes/out5.desc
12 data/bikes/out6.desc
13 #
14 data/boat/out2.desc
15 data/boat/out3.desc
16 data/boat/out4.desc
17 ..........
```

Listing A.3: `files.qrf`

The files which contain descriptors use standard format as is produced by the application `compute_descriptors`.

A.3 Performance Evaluation tools

Performance Evaluation tool consists of several matlab functions. The most important function is `repeat` which take two arguments: path to the file with evaluation settings and name of the results file.

The file with evaluation settings has following structure

```
1  \#comments psize dsize x y cornerness scale
2  40
3  7
4  0
5  data/bark/out2.desc.pascal1000.pts.t80.e15.00.kd  data/bark/out2.desc data/bark/
       out1.desc data/bark/H1to2p  data/bark/img1.png  data/bark/img2.png
6  data/bark/out3.desc.pascal1000.pts.t80.e15.00.kd  data/bark/out3.desc data/bark/
       out1.desc data/bark/H1to3p  data/bark/img1.png  data/bark/img3.png
7  ..............
```

Line 1: comments
Line 2: number of files to evaluate
Line 3: number of properties
Line 4: empty
Line 5, 6, ..., 40: path to the file with results, path to the features query file, path to the features reference file, path to the homography matrix, reference image, query image

Other useful files are `plotEvaluation.m` which produce figures for performance criterion and `plotTime.m` which produce the figure for the parameter S_T.

A.4 Other

For Windows users, some PowerShell scripts for batch processing are provided. More information are available in the appropriate directory on the enclosed DVD. It is expected, that Unix–like user are able to write their own scripts.

More information about performance evaluation framework could be found on enclosed DVD.

B LOCAL FEATURE DETECTORS

In this section, we summarize local invariant feature detectors more in detail. The overview discuss the same detectors as in section 2.2, however in this appendix equations and others are discussed.

B.1 Corner Detectors

As we have mentioned, corners and T–like junctions are important even for human vision. They are often sensitive to a different parts of image, not just to corners, however it does not matter. The goal is to have as much as possible stable repeatable local invariant features.

B.1.1 Harris/Plessey Detector

The Moravec detector was improved by Harris and Stephens in 1988 [23]. By comparison with the Moravec detector, the corner score is measured directly instead of using shifted windows and has isotropic response. It is based on eigenvalues of the second moment matrix (auto–correlation matrix), which represents the most principal signal changes in two orthogonal directions around the point – Harris points are detected if both eigenvalues are large [66]. This matrix describes derivation distribution in point neighborhood:

$$\mathbf{M} = \sigma_D^2 g(x, y, \sigma_I) * \begin{bmatrix} I_x^2(x, y, \sigma_D) & I_x(x, y, \sigma_D) I_y(x, y, \sigma_D) \\ I_x(x, y, \sigma_D) I_y(x, y, \sigma_D) & I_y^2(x, y, \sigma_D) \end{bmatrix} \tag{B.1}$$

where

$$I_x(x, y, \sigma_D) = \frac{\partial I(x, y)}{\partial x} * g(x, y, \sigma_D), \tag{B.2}$$

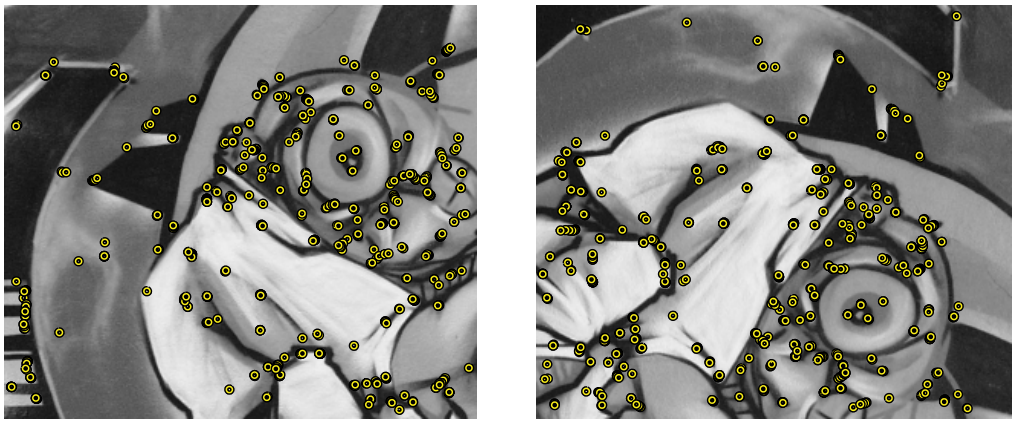
$$I_y(x, y, \sigma_D) = \frac{\partial I(x, y)}{\partial y} * g(x, y, \sigma_D), \tag{B.3}$$

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{B.4}$$

with σ_D denotes differentiation scale and σ_I integration scale and $*$ denotes convolution. The eigenvalues of $\mathbf{M}$ decide whether the point is a corner (both eigenvalues are large), edgel (either eigenvalues is large, the second is close to zero) or no point of interest was found (both eigenvalues are small). Also, the less expensive computational approach was proposed:

$$cornerness = \det(\mathbf{M}) - \kappa \cdot (\mathrm{tr}(\mathbf{M}))^2, \tag{B.5}$$

with κ is a constant, usually $\kappa \in < 0.04, 0.06 >$, det is determinant of a matrix and tr is a trace of a matrix. Non–maximum suppression is used to extract features as local maxima of cornerness function.



(a) (b)

Fig. B.1: Features extracted by Harris corner detector, courtesy of [66]

Harris corner detector is invariant under rotation and translation only, many features are localized on edges instead of real corners and sensitive to noise.

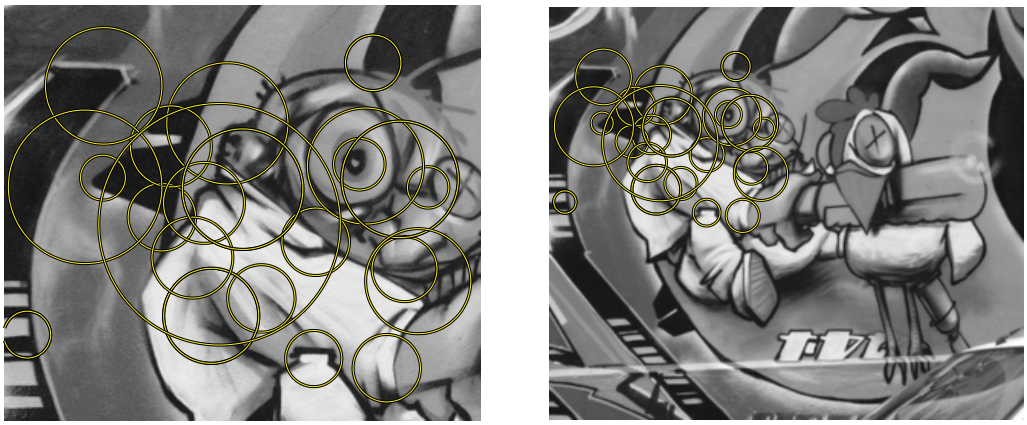
B.1.2 Harris–Laplace

Harris–Laplace detector was proposed by Mikolajczyk and Schmid [39] as an scale invariant extension to the Harris corner detector. It is based on multi–scale Harris corner detector and the characteristic scale is selected as was proposed by Lindenberg in 1998 [31]. Firstly, scale–space representation is built with the Harris function (B.5). Then for each initial point it is checked whether the LoG (B.6) attains a maximum at the scale of the point.

$$|\mathrm{LoG}(x, y, \sigma_n)| = \sigma_n^2 |I_{xx}(x, y, \sigma_n) + I_{yy}(x, y, \sigma_n)| \tag{B.6}$$

The points for which LoG response attains no extreme or is bellow some threshold as same as points for which the scale peak does not correspond to the selected detection scales of an image are rejected due to the lack of a maximum, or inaccuracy.

The LoG is used due to its so-called Mexican hat characteristic shape of the kernel, which serves as a matched filter when its scale is adapted to the scale of a local image structure. Harris–Laplace detector provides efficient method extracting rotation, translation and scale invariant local features [41, 66].



(a) (b)

Fig. B.2: Features extracted by scale invariant Harris–Laplace corner detector, courtesy of [66]

B.1.3 Harris–Affine

Harris–Laplace detector fails in the case of significant affine transformations when the scale change is not necessarily the same in every direction. Harris–Affine [40] is an affine invariant extension to Harris–Laplace detector. The main idea is to use a second moment matrix in affine Gaussian scale-space where an elliptical affine regions are used instead of circular region.

$$\mu(x, y, \Sigma_I, \Sigma_D) = \det(\Sigma_D) g(\Sigma_I) * \Big((\nabla I)(x, y, \Sigma_D)(\nabla I)(x, y, \Sigma_D)^{\mathrm{T}} \Big), \tag{B.7}$$

where Σ_I is the differentiation covariance matrix and Σ_I is the integration covariance matrix which determines the Gaussian kernels. In fact, this equation is identical with the second moment matrix used in Harris-Laplace detector. The algorithm runs iteratively as follows:

1. Initialization with Harris–Laplace detector
2. Estimation of affine shape with the second moment matrix
3. Normalization of an elliptical affine region to the circular one
4. Re–detection of the location and scale in the normalized image
5. Go to step 2, if eigenvalues of the second moment matrix for the new point are not equal

The objective is to determine the transformation that projects the intensity pattern of the point neighborhood to one with the equal eigenvalues. It can be

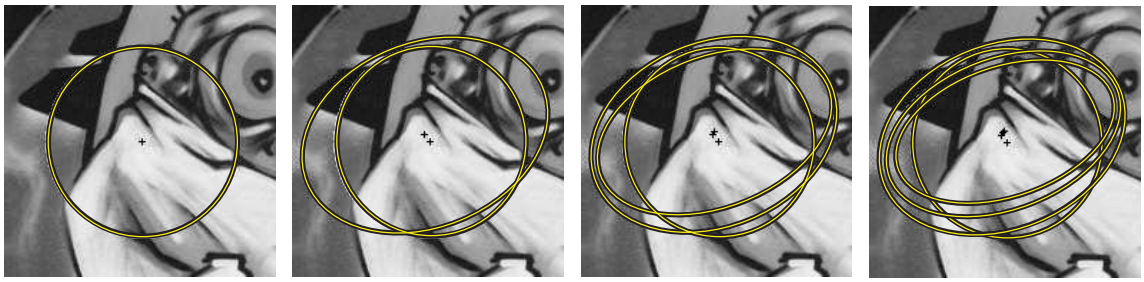


Fig. B.3: Iterative detection of an affine invariant feature by Harris–Affine detector, courtesy of [66]

shown that the affine transformation is given by the square root of the second moment matrix just as it can be shown that if the neighborhoods of the points $\mathbf{p}_R$ and $\mathbf{p}_L$ are related by an affine transformation, then their normalized versions are connected by rotation or mirror matrix

$$\mathbf{A} = \mathbf{M}_R^{-\frac{1}{2}} \mathbf{R} \mathbf{M}_L^{-\frac{1}{2}}, \tag{B.8}$$

$$\mathbf{p}_L = \mathbf{M}_L^{-\frac{1}{2}} \mathbf{x}'_L, \tag{B.9}$$

$$\mathbf{p}_R = \mathbf{M}_R^{-\frac{1}{2}} \mathbf{x}'_R. \tag{B.10}$$

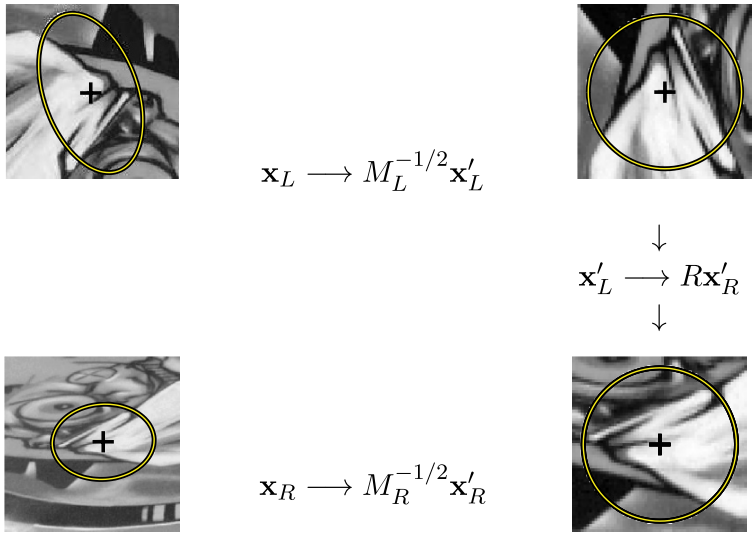


Fig. B.4: Affine normalization using second moment matrices, courtesy of [66]

Harris–Affine detector is invariant to rotation, translation, scale and affine transformation. The number of detected local invariant features depends on the type of scene and the threshold [41, 66].

B.2 Blob Detectors

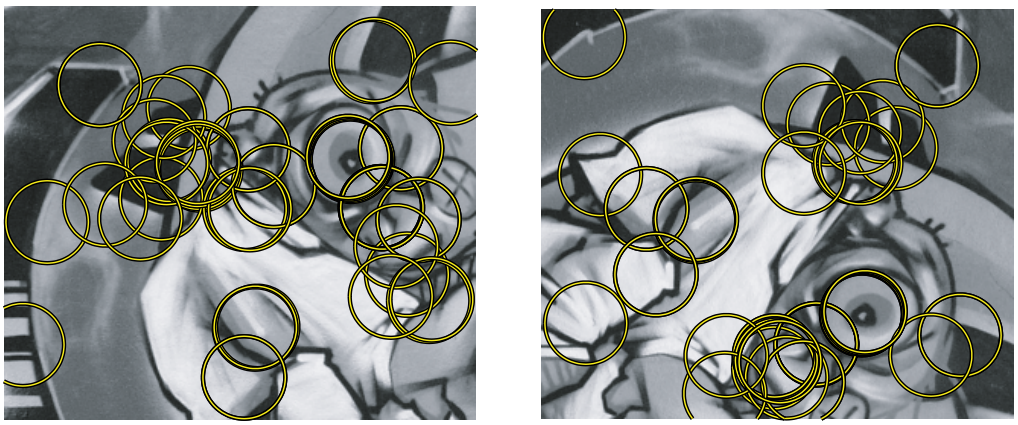
In this section, we briefly describe fundamentals of blob detectors. We begin with a derivative–based method called Hessian detector B.2.1. Next, we continue with scale and affine B.2.2 invariant extensions to this detector.

B.2.1 Hessian Detector

The class of Hessian detectors are based on the Hessian matrix, which is obtained as the second matrix from the Taylor expansion of the image intensity function I(x, y):

$$\mathbf{H} = \begin{bmatrix} I_{xx}(x, y, \sigma_D) & I_{xy}(x, y, \sigma_D) \\ I_{xy}(x, y, \sigma_D) & I_{yy}(x, y, \sigma_D) \end{bmatrix} \tag{B.11}$$

where $I_{ab}(x, y, \sigma_D)$ are second–order Gaussian smoothed partial derivatives. Detector based on the trace of $\mathbf{H}$ is well–known Laplacian (of Gaussian) which can be efficiently approximated by the Difference of Gaussians (see 2.2.4).



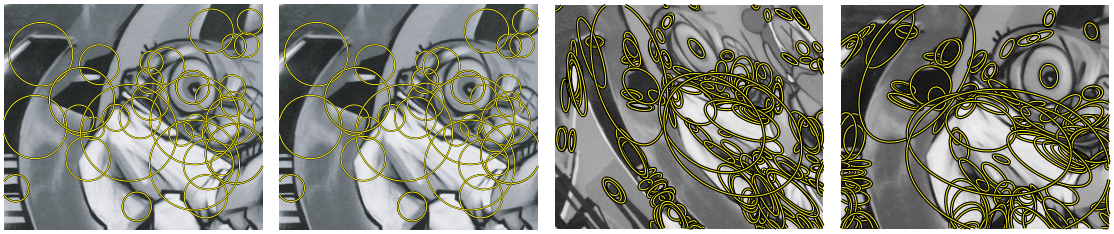
(a) (b)

Fig. B.5: Features extracted by Hessian blob detector, courtesy of [66]

Laplacian detectors often detects local maxima, which are localized close to the edges or contours (signal changed in one direction). These maxima are less stable because detectors are more sensitive to noise or small changes in neighboring texture. Efficient way, how to detect more stable maxima is to detect a location and scale for which the trace and the determinant of $\mathbf{H}$ attains the local extremum simultaneously, because the trace maximize the curvature and the determinant penalizes small second derivations in only a single direction [66].

B.2.2 Hessian–Laplace/Affine

The concept behind the Hessian–Laplace detector is similar to the Harris-Laplace, however by comparison with Harris–Laplace, the Hessian–Laplace detector is initialized from the determinant of the Hessian matrix, instead of Harris detector. Hessian–Laplace/Affine detectors were also proposed by Mikolajczyk and Schmid [41]. Hessian based detectors are complementary to the Harris detectors, because they are sensitive to the different parts of the image. Scale and affine invariant properties of the detectors are provided in a similar manner – Hessian detector localize blobs in space, Laplacian in scale and iterative approach is used for affine invariance.



(a) (b) (c) (d)

Fig. B.6: Features extracted by scale invariant Hessian–Laplace blob detector (a, b), and affine invariant features extracted by Hessian Affine detector (c, d), courtesy of [66]

Number of detected blobs can be controlled by thresholding both determinant of **H** and the Laplacian response. Another convenient property is that a large number of detected features are good covering the image. Hessian based detectors are translation, rotation, scale and affine invariant [42, 66].

B.3 Region Detectors

In this section, we summarize detectors which extract image regions. We begin with Intensity–based Regions. Then, we focus on Maximally Stable Extremal Regions. Let us note that the importance of segmentation based methods (superpixels) is growing, but they are no discussed in this thesis.

B.3.1 Intensity–based Regions

Intensity–based regions were proposed by Tuytelaars and Van Gool in 2000. The main idea of this approach is following: the neighborhood of a point with extrema intensity (detected at multiscale) is searched along each ray of the region:

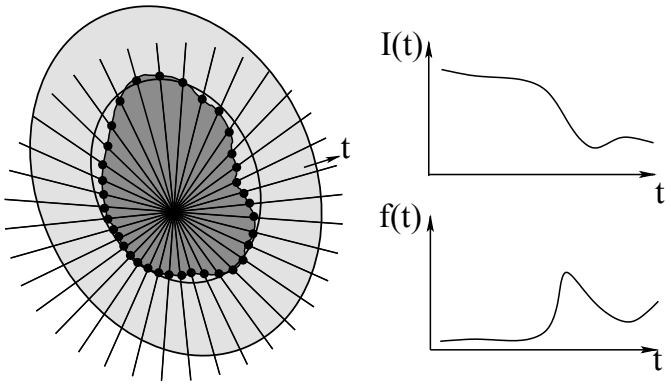


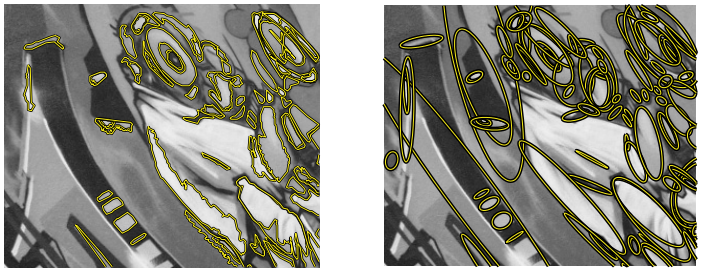
Fig. B.7: Construction of Intensity Based Region, courtesy of [66]

$$f(t) = \frac{|I_t - I_0|}{\max\left(\frac{\int_0^t |I_t - I_0|\,\mathrm{d}t}{t}, d\right)} \tag{B.12}$$

with t the Euclidean arclength along the ray, I_0 the intensity value at extremum, I_t the intensity at position t and d, a small constant to prevent division by zero. This function reached its maximum in positions at intensity suddenly increases/decreases. The point for which the function (B.12) reached the maximum is invariant under both affine linear photometric and geometric transformations. All the points which corresponds the maxima of $f(t)$ along rays originationg from the same local extremum are linked to enclose an affine invariant region. This often irregularly–shaped region is then replaced by an ellipse having the same shape moments up to the second order. Finally, the area of the ellipse is doubled to get the more distinctive feature.

B.3.2 Maximally Stable Extremal Regions

Maximally Stable Extremal Regions (MSER) were proposed by Matas et al. in 2002 [37]. The concept of MSER can be described as follows: sort all pixels of gray level image by its intensity and consider all possible thresholds. Start with thresholding of the image for the lowest thresholds. At the beginning, the whole image will be white. Then, a few pixels will be under the threshold, thus these pixels will be black. When the threshold will be higher, these spots will grow and at some point, two neighboring spots will merge. In the end, the whole image will be black. The set of all connected components, for which the binarization is stable over a large range of thresholds, is the set of all MSER (set of all maximal regions). The same process with inverted tresholds is used to get the set of all minimal regions.



(a) (b)

Fig. B.8: Features extracted by Maximally Stable Extremal Regions detector (a) and fitted ellipse based on the first and second shape moments (b), courtesy of [66]

The word extremal denotes that all pixel in MSER have higher (maxima) or lower (minimal regions) intensity then all pixels on its boundary. MSER is very fast, because the complexity of the algorithm is $\mathcal{O}(n \log \log n)$.

(a) (b)

Fig. B.9: Estimated epipolar geometry and points associated to the matched regions for robust wide baseline stereo from MSER, courtesy of [37]

MSER features are accurately localized, because it is sensitive to region boundaries. The algorithm is very efficient for structured scenes with regions separated by strong intensity changes. The main drawback of MSER is the sensitivity to image blur.

Tab. B.1: Overview of local invariant feature detectors

Feature detector	Invariance						Localization			
	Corner	Blob	Region	Rotation	Scale	Affine	Repeatability	Accuracy	Robustness	Efficiency
Harris	√			√			⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆
Hessian		√		√			⋆⋆	⋆⋆	⋆⋆	⋆
SUSAN	√			√			⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
Harris–Laplace	√	(√)		√	√		⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆
Hessian–Laplace	(√)	√		√	√		⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆
DoG	(√)	√		√	√		⋆⋆	⋆⋆	⋆⋆	⋆⋆
SURF	(√)	√		√	√		⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
Harris–Affine	√	(√)		√	√	√	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆
Hessian–Affine	(√)	√		√	√	√	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆
Salient Regions	√	(√)		√	√	(√)	⋆	⋆	⋆⋆	⋆
Edge–based	√			√	√	√	⋆⋆⋆	⋆⋆⋆	⋆	⋆
MSER			√	√	√	√	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆
Intensity based			√	√	√	√	⋆⋆	⋆⋆	⋆⋆	⋆⋆
Superpixels			√	√	(√)	(√)	⋆	⋆	⋆	⋆

C ENCLOSED DVD

Enclosed DVD contains the following:

- The whole thesis in PDF.
- Source files for Performace Evaluation Framework in C++ and Matlab.
- The Graffiti and PASCAL data sets.