Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Robust Estimation of Geometric Models for Retrieval with Neural Networks

Master's Thesis

Bc. Daniel Hubáček

Master program: Open Informatics
Specialisation: Computer Vision and Image Processing
Supervisor: prof. Mgr. Ondřej Chum, Ph.D.

Prague, May 2024

Thesis Supervisor:
prof. Mgr. Ondřej Chum, Ph.D.
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13
121 35 Prague 2
Czech Republic

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 24 May 2024

.....
Bc. Daniel Hubáček

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hubáček Daniel** Personal ID number: **483726**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Computer Vision and Image Processing**

II. Master’s thesis details

Master’s thesis title in English:

Robust Estimation of Geometric Models for Retrieval with Neural Networks

Master’s thesis title in Czech:

Robustní odhad geometrických modelů pro vyhledávání pomocí neuronových sítí

Guidelines:

Implement a robust estimator of models of various complexity and evaluate their benefit in image retrieval. Suitable models include affine transformation, planar homography, and multiple occurrences of these models in a scene. Consider using other image elements than point correspondences. Apply a robust estimator to improve search results, both when reranking the results and when learning the image description extractor.

Bibliography / sources:

[1] Perdoch et al., Efficient Representation of Local Geometry for Large Scale Object Retrieval, CVPR 2009
[2] Hartley & Zisserman: Multiple View Geometry, Cambridge University Press, March 2004
[3] Tolieas et al., Learning and aggregating deep local descriptors for instance-level recognition, ECCV 2020

Name and workplace of master’s thesis supervisor:

prof. Mgr. Ondřej Chum, Ph.D. Visual Recognition Group FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **22.09.2023** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **16.02.2025**

prof. Mgr. Ondřej Chum, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student’s signature

Abstract

Instance level image retrieval makes use of a spatial verification, which is based on a robust estimation of geometrical transformations between a pair of images. The spatial verification generates various transformations from tentative feature correspondences. This thesis aims to implement this algorithm and extend it by utilising line segments detected in the neighbourhood of the feature pair. Two matching strategies are used to determine segment correspondences in order to generate soft constraints on the estimated transformation. Furthermore, vanishing points obtained from underlying lines are utilised to create additional hard constraints. All approaches are evaluated, and it is shown that lines can compensate for inaccurate or completely missing feature data.

Keywords: Image retrieval, feature matching, robust estimation, transformation estimation

Anotace

Vyhledávání obrázků na úrovni instancí využívá prostorové ověřování, které je založeno na robustním odhadu geometrických transformací mezi dvojicí obrázků. Prostorové ověřování generuje různé transformace z provizorně určených odpovídajících si rysů. Diplomová práce si klade za cíl implementovat tento algoritmus a rozšířit jej o využití úseček detekovaných v okolí těchto rysů. Dvě strategie párování jsou použity k určení korespondencí jednotlivých úseček, aby se vytvořila dodatečná omezení na odhadovanou transformaci. Dále jsou využity úběžníky získané z nadetekovaných přímek k vytvoření dalších omezení. Všechny přístupy jsou vyhodnoceny a je ukázáno, že úsečky mohou kompenzovat nepřesná nebo plně chybějící data rysů v obrázcích.

Klíčová slova: Vyhledávání obrázků, párování oblastí v obrázku, robustní odhadování, odhadování transformací

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor, prof. Mgr. Ondřej Chum, Ph.D., for his guidance, remarks, and kind attitude. It was an honour to be led by such an experienced character. I also greatly appreciate Ing. Tomáš Jeníček for his help and technical support, as well as the rest of the Visual Recognition Group members for making me feel welcome in their team.

I would also like to thank all of my friends and family for the care and support they have provided throughout my academic journey. Their cheer, love, and optimism were essential to my success.

List of Tables

List of Figures

List of Acronyms

$\mathcal{R}$Oxford Revisited Oxford. 17–19, 38, 40, 43, 44, 46, 48, 49, 51, 52, 54–56, 59, 60, 75, 77, 80, 83, 84, 92

AP Average Precision. 20, 21

ASMK Aggregated Selective Match Kernel. 16, 17, 38, 40, 58, 59, 77

BBox Bounding Box. xiii, xv, 18, 40, 50, 53–55, 57

BoW Bag of Words. 14, 16, 38, 40, 58, 60

BRIEF Binary Robust Independent Elementary Features. 13, 14

CBIR Content-Based Image Retrieval. 2, 3

CNN Convolutional Neural Network. 37, 38

DELF Deep Local Features. 14

DLT Direct Linear Transformation. 25, 50

DoF Degrees of Freedom. 32, 36, 37, 49, 72

FAST Features from Accelerated Segment Test. 10, 14

FN False Negative. 20

FP False Positive. 20

HE Hamming Embedding. 16

HOG Histograms of Oriented Gradients. 13

HT Hough Transform. 62, 63, 67

IDF Inverse Document Frequency. 15

LO Local Optimisation. 28, 45, 46, 54, 59

LO-RANSAC Locally Optimised Random Sample Consensus. 28, 29, 44

LoG Laplacian of Gaussian. 11

LSD Line Segment Detector. 64–66, 68, 69

mAP Mean Average Precision. 20, 40, 41, 43–46, 48, 49, 52, 54–56, 58–60, 73–77, 80, 83, 84, 92

MLE Maximum Likelihood Estimation. 23, 29

NLP Natural Language Processing. 14

OLS Ordinary Least Squares. 23, 29

ORB Oriented FAST and Rotated BRIEF. 14

PPHT Progressive Probabilistic Hough Transform. 63, 64, 66, 68

PROSAC Progressive Sample Consensus. 30, 35

QE Query Expansion. xiii, 57, 59

RANSAC Random Sample Consensus. 24, 27–30, 33–35, 37, 39, 40, 50, 84, 87, 88

SfM Structure from Motion. 56

SIFT Scale Invariant Feature Transform. 11–13, 16, 30, 38, 40, 43, 45, 46, 48, 49, 52, 54–56, 58–60, 74, 83, 84, 95

SURF Speeded-Up Robust Features. 13, 16

SV Spatial Verification. 23, 40, 41, 46, 48, 49, 52, 55, 56, 58–60, 74–76, 80, 83, 84, 89, 91, 92

SVD Singular Value Decomposition. 90

TBIR Text-Based Image Retrieval. 2

TF Term Frequency. 15

TF-IDF Term Frequency-Inverse Document Frequency. 15, 58

TN True Negative. 20

TP True Positive. 20, 41

VLAD Vector of Locally Aggregated Descriptors. 16

VP Vanishing Point. 84, 89–92

Contents

Chapter 1

Introduction

Even though no exact numbers are known, there are millions and billions of images taken and uploaded to the internet every day. There are good reasons to believe that not only the number of images, but also the growth rate, will increase in the future. People's private albums, public social media profiles, data backups, websites, blogs, and even images of places and street views in maps contribute to this trend. Pictures are, and will be, almost everywhere in the digital sector.

Images are often considered unstructured data. Unstructured data refers to information that does not have a pre-defined data model or is not organised in a pre-defined manner. Unlike structured data, which fits neatly into databases and spreadsheets, unstructured data lacks a consistent format, making it more challenging to analyse using traditional data processing techniques.

An image typically consists of pixels arranged in a grid, with each pixel containing colour information. While there are patterns and structures within images (such as edges, shapes, and textures), the data itself is not organised in a standardised way that lends itself to easy analysis. Additionally, images can vary widely in content, resolution, aspect ratio, and other characteristics, further contributing to their unstructured nature.

Despite being unstructured, images contain valuable information that can be extracted and analysed using techniques from fields like computer vision and/or machine learning. These techniques allow computers to interpret and understand the content of images, enabling applications such as image recognition, object detection, 3D reconstruction, video tracking, image stitching, classification, and image retrieval.

Image retrieval is the problem of searching for images in a large, unordered collection. It has numerous applications across various domains, such as, but not limited to, medical imaging, security and surveillance, fingerprint or trademark identification, copyright protection, art and cultural heritage, geospatial retrieval, e-commerce, place recognition, etc. There are already commercial products that can search image collections. For example, Google Image Search (`images.google.com`), TinEye (`www.tineye.com`) or Bing Image Feed (`www.bing.com/images`), which retrieve images from the internet, or even Apple Photos, which can perform searches in user's personal library.

Two major categories of image retrieval are Text-Based Image Retrieval (TBIR) and Content-Based Image Retrieval (CBIR). The former approach uses textual annotations of images, where keywords are used to describe the content, and text-based database management systems and search approaches to perform the retrieval. Text annotations are made by humans and therefore are subject to human perception system and its subjectivity. That is, the same image content may be perceived differently by different people and can even change over time. Furthermore, the difficulty of annotating images increases with the collection size [1].

The latter approach, CBIR, analyses the information derived from the image pixels and structure alone. It pays greater attention to extracting various kinds of visual features based on global and local information, such as colour, shape, and texture [1]. These visual features are then stored in a database and used for retrieval. Images that contain the same visual features are considered *similar*, in a sense.

For CBIR, a query image is on the input, and a list of images from a collection is expected as output. The query image often contains an object of interest. This problem is also referred to as *instance image retrieval*. Having two images, a query and a collection image, they both should contain the same object of interest, at least in the ideal scenario. Then there might exist a geometric transformation between the two images, mapping the object from one image to the other.

This transformation estimation is the subject of this thesis. The transformation provides more information about the nature of the image pair. It can be then used for all kinds of things, such as filtering collection images, sorting the results, or repetitive querying.

The thesis first describes the process of Content-Based Image Retrieval. From feature detection to the retrieval itself. Then, a transformation estimation algorithm for instance image retrieval is introduced. This algorithm uses the features detected in the first part. Later, lines are considered and utilised during the transformation estimation, as they provide more information about the geometry.

Chapter 2

Image retrieval

Image retrieval is a field within computer vision and information retrieval that focuses on finding and retrieving relevant images from large collections, and it has many use cases and applications in the real world. This chapter introduces the task of instance image retrieval and describes methods based on local visual features. It aims to create an intuition of how an instance image retrieval system can really work under the hood, as well as to dive into some details of basic approaches. These details create a comprehensive and complete view of the task itself, such that it can be implemented and transformed into a working search engine. It serves as a foundational framework for subsequent methodologies. At the end, the focus is given to how a retrieval system is evaluated and possibly compared to other methods.

2.1 General image retrieval

Image retrieval is a very general task that can differ already in the type of data given as input. It can be a text description of images, a sketch of an object, an image itself, or even just a part of an image. The latter options are subject to CBIR.

CBIR is a task where an image is given as a query, and the goal is to retrieve all *relevant* images from an unordered collection of pictures. The term *relevancy* is not uniquely defined and can vary depending on the particular task and use case. Relevant images can mean that the images should be (almost) identical, semantically or contextually similar, depicting the same scenery, capturing identical objects, or all objects from one particular category.

Ideally, the retrieval system should retrieve all relevant images. However, a common approach is to rank all the images in the collection according to a *score*. The score is a scalar value representing the likelihood that a pair of images is relevant. At the end, the first N images or images with a score above a certain threshold are retrieved as results. This ability to retrieve relevant images first is referred to as *performance*. Other characteristics of retrieval approaches include *speed* (needed time for a query), *compactness* (memory requirements), and *scalability* (applicability to large image collections).

Instance level image retrieval takes an image of an object as input and aims to retrieve all images from the collection that contain the same object. Unlike image or object classification,

where only a limited set of classes needs to be pre-defined, no such assumptions are imposed in case of instance image retrieval; the queried object can be arbitrary. Hence, standard image classification and object detection techniques are not directly applicable to all retrieval tasks, although there might be some applications where this solution would be sufficient.

Instance image retrieval remains an open problem not only because of the variety of applications but also due to the high variation in the visual appearance of the same object. Common challenges to take into consideration are:

- Viewport and/or scale change; two pictures of the same object can be taken from different locations in space
- Illumination change; there can be different lighting conditions, e.g., when picture is taken inside/outside, during a day/night or due to seasonal changes
- Colour variations; not only the illumination, but also colour balance settings and the properties of the imaging sensor can influence the pixel values
- Occlusions; the object of interest can be partially hidden from view by other objects in the scene
- Deformation; it can occur due to factors such as perspective distortion or object movement
- Visually similar but different objects should not be retrieved together; e.g., when the query contains a painting, the goal is not to find all paintings, but only images of the queried one
- Various types of image modalities; an image can be a photograph, a painting or a drawing, sketch, illustration, mosaic, etc.
- Noise and distortions; artefacts can be introduced during the image acquisition or compression process

Various visual features are used for retrieval, and these features can be either local or global. Global features describe the image as a whole in order to generalise the entire object. They can be understood as a dimensionality reduction technique because they represent an image of width W, height H and with C channels as vector of D dimensions. The goal is to have a small Euclidean distance between vectors corresponding to mutually relevant images so that retrieval can be performed using a nearest neighbour search (i.e., finding images with the vector representation closest to the query image vector representation).

Global features have been successfully employed for image retrieval, but they possess one important weakness that limits their efficiency. It is the inability to differentiate between image parts, i.e., object of interest, other objects contained in the image, or the background. As a result, they do not perform well in tasks with complex and cluttered scenes [1].

2.2 Local-feature-based methods

Local visual features focus on capturing information in a local neighbourhood of a particular point in an image. Each image has multiple local features, which are then compared. The idea and ideal scenario is that relevant pairs of images have the same or similar features in them. Therefore, the features should be designed in such a way that they preserve their essence even under already mentioned circumstances (viewport and/or scale change, illumination change, noise and distortion, etc.).

Typical pipeline consists of three major steps:

1. **Feature detection:** An image I from a space of all images $\mathcal{I}$ is taken and a set of points $\mathbf{x}_i = [x_i, y_i]^\top$ is detected using a function $detect(I) \rightarrow \{\mathbf{x}_i\}$. These points ought to have some potential to carry a valuable piece of information. Up-scaled and down-scaled images are often used for the detection as well to cover various sizes of local neighbourhoods since the detection methods usually do not have such capabilities. This up-scale and down-scale factor is referred to as a *scale*.

2. **Feature description:** A vector representation $\mathbf{d}$ is assigned to each detected feature $\mathbf{x}$ based on its local neighbourhood using a function $describe(I^{[\mathbf{x}]}) \rightarrow \mathbb{R}^D$, where $I^{[\mathbf{x}]}$ represents the local neighbourhood of the point $\mathbf{x}$ in the image I. Thus, an image I has a set of vectors $\mathcal{D}_I = \{\mathbf{d}_i\}$. The vector $\mathbf{d}$ is called *a descriptor* and its value can capture for example the local shape, colour, texture, or even a mixture of such properties describing the neighbourhood. Theoretically, it would be also possible to have multiple descriptors for one detected feature, each descriptor representing different properties. It can be hand-crafted or even learnt by machine learning approaches. The goal is to create descriptors in such a way that similar visual local features (features with visually similar neighbourhood) have assigned descriptors with small Euclidean distance. At the same time, descriptors of different local features should have significantly larger distance.

These first two steps, feature detection and description, are executed on images from three datasets - training dataset, collection dataset, and queries.

First, a large dataset of training images $\mathcal{I}_{train}$ is taken and descriptors for all images are computed, resulting in a large set of all descriptors from the whole training dataset $\mathcal{D}_{train}$,

$$\mathcal{D}_{train} = \{\mathbf{d} \mid \mathbf{d} \in \mathcal{D}_I, I \in \mathcal{I}_{train}\}$$

These descriptors are then *clustered* into N *clusters*. Clustering is a technique wherein a collection of data points is partitioned into groups (clusters), such that the data points within each cluster are more similar to each other than to those in other clusters. Descriptors within each group could be seen as if they represented a single visual feature. A popular clustering algorithm is K-means [2]. In this case, the similarity is measured by the Euclidean distance between descriptors. Each cluster has a *centroid* $\boldsymbol{\mu}$, which is a central (or representative) point, computed

as the mean of entries belonging to the corresponding cluster. These clusters are called *visual words*. Set of all visual words is called *a vocabulary* or *a codebook*.

Having N visual words estimated from the training set, feature detection and description is performed on images from the collection dataset (also referred to as *database*), $\mathcal{I}_{db}$. Each descriptor is then assigned to a cluster j, so each descriptor $\mathbf{d}_i$ has a corresponding visual word $w_i = j$. In case of previously described K-means, the nearest cluster j is taken. The nearest cluster is considered to be the cluster whose centroid is the closest, i.e., $w_i = j = \text{argmin}_j \, dist(\boldsymbol{d}_i, \boldsymbol{\mu}_j)$.

Afterwards, the following data is stored in a database for each image from the collection dataset: its features locations $\mathbf{x}_i$, scales s_i on which it was detected and corresponding visual words w_i. Other properties of the features can be stored in addition to the mentioned ones, e.g., original descriptors, residuals from centroids, detection and description confidence, etc. These will be mentioned and used later.

Lastly, very similar workflow is used also for queries. Features in a query image are detected, described and assigned to the pre-trained visual words. Then, the last, third, major step takes place:

3. **Ranking and retrieval:** Mutual relevance of two images is estimated, given a set of visual words $\mathcal{W}_I$ for each of them. Exact evaluation strategy depends on the retrieval and ranking algorithm used, but the fundamental idea is to perform a *feature matching*. Two features are considered to match if they are assigned to the same visual word. The more matching visual words the pair of images has, the more relevant the images appear. This process can be represented by a function $matching(\mathcal{W}_{I1}, \mathcal{W}_{I2}) \rightarrow \mathbb{R}$ returning a scalar value, which can be interpreted as a score used for ranking the images.

The query image is compared to all images in the database $\mathcal{I}_{db}$, which are then sorted according to the score. At the end, a portion of images is retrieved. As it was already mentioned, it can be first N images with the highest score or images with a thresholded score.

Each of these steps will be now described in detail.

2.3 Feature detection

Local features, as it was already described, are distinct regions within an image (i.e., in principle arbitrary sets of pixels) that exhibit certain properties, notably having a *discriminative neighbourhood* and high *repeatability*. They can be understood as a form of attention.

Discriminative neighbourhood means that the surrounding context of each *keypoint* (i.e., a unique pixel within the region) should contain unique information that facilitates accurate localisation and matching. This property enhances the distinctive nature of local features, enabling them to effectively capture and represent the salient characteristics of the underlying image content.

Repeatability, in the context of image processing and computer vision, refers to the ability of a feature detector (or/and descriptor) to reliably detect (or/and describe) the same visual pattern

or structure across different instances of an object or scene. In simpler terms, it measures how consistently a feature is detected under various changes in the appearance of an object or scene, such as changes in viewpoint, scale, rotation, illumination, and occlusion. This is especially important from the geometric point of view, for transformation estimation.

These regions, often characterised by distinctive patterns or structures, serve as key points of interest for further analysis and processing. Each region is represented by a keypoint pixel, usually located in the centre of the region. Keypoints also have other properties associated with them (e.g., scale or orientation) based on the image or the neighbourhood of the pixel.

There are usually two steps in the detection process - detecting and sorting of keypoints. A detector marks pixels with a potential to be keypoints. But a very permissive detector could potentially mark (almost) all image pixels. Therefore each keypoint should have a rank based on which they could be sorted and only top N taken. The ranking should be consistent under a nuisance factor.

As a tangible example, one method will be described in detail: Harris detector, classical analytical method seeking for local extremes.

2.3.1 Harris detector

The Harris Corner Detector is a popular method in computer vision used to detect corners in images. It was developed by Chris Harris and Mike Stephens in 1988 [3].

In order to be a distinguished region, a region must be at least distinguishable from all its neighbours. Therefore the detector analyses small windows of an image and computes the variation in intensity when the window is moved in any direction. Ideally, shifting a window in any direction should give a large change in intensity. Specifically, it calculates the *corner score* for each pixel in the image, which indicates how likely that pixel is to be part of a corner. The corner score is based on the amount of change in intensity that occurs when the window is shifted in a certain direction. Pixels with high corner scores are considered to be part of corners.

Let $I(x, y) \to \mathbb{R}$ be an intensity function of an image, $I(x, y)$ be the intensity value of the pixel at coordinates $[x, y]$. Then *energy function* E is defined as a weighted sum of squared differences of intensities in a window W centred at $[x_0, y_0]$ and corresponding intensities in the same window moved in $[u, v]$ direction:

$$E(x_0, y_0; u, v) = \sum_{(x,y)\in W(x_0,y_0)} w(x, y)\big(I(x, y) - I(x + u, y + v)\big)^2 \tag{2.1}$$

Window W is typically a 3×3 square of neighbouring pixels with its centre in $[x_0, y_0]$. Weight function w can be a constant or (better) a Gaussian. The energy E is computed for all pixels $[x_0, y_0]$ in the image independently.

Potential keypoints should have a large change in intensity (in other words, large energy) in all directions $[u, v]$. Therefore minimum value of the energy function E over $[u, v]$ shifts is taken, $\min_{u,v} E(x_0, y_0; u, v)$. The higher minimum energy, the higher corner score for a particular pixel. Local maxima are sought.

Equation (2.1) gets computationally expensive and therefore other approach is used. The intensity function in the shifted position is approximated by the first-order Taylor expansion.

A first-order Taylor expansion of a function f near the point a is defined as:

$$f(x) \approx f(a) + f'(a)(x - a),$$

in case of the shifted intensity function:

$$I(x+u, y+v) \approx I(x,y) + \begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix},$$

where I_x, I_y are partial derivatives of $I(x,y)$ in the x, y directions, respectively. Partial derivatives of an image can be efficiently approximated using the Sobel operator [4] (or a similar approach). Sobel operator computes the derivative as a difference of neighbouring pixels along corresponding axis. Weights and larger neighbourhood can be also used.

The approximation of the shifted intensity function can be used in the Equation (2.1):

$$\begin{aligned}
E(x_0, y_0; u, v) &= \sum_{(x,y) \in W(x_0,y_0)} w(x,y) \big(I(x,y) - I(x+u, y+v)\big)^2 \\
&\approx \sum_{(x,y) \in W(x_0,y_0)} w(x,y) \Big(I(x,y) - \big(I(x,y) + \begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}\big)\Big)^2 && (2.2) \\
&= \sum_{(x,y) \in W(x_0,y_0)} w(x,y) \Big(- \begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}\Big)^2 && (2.3) \\
&= \sum_{(x,y) \in W(x_0,y_0)} w(x,y) \Big(\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x(x,y) \\ I_y(x,y) \end{bmatrix} \begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}\Big) && (2.4) \\
&= \begin{bmatrix} u & v \end{bmatrix} \underbrace{\sum_{(x,y) \in W(x_0,y_0)} w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y) I_y(x,y) \\ I_x(x,y) I_y(x,y) & I_y(x,y)^2 \end{bmatrix}}_{\mathbf{M}(x_0,y_0)} \begin{bmatrix} u \\ v \end{bmatrix} && (2.5) \\
&= \begin{bmatrix} u & v \end{bmatrix} \mathbf{M}(x_0, y_0) \begin{bmatrix} u \\ v \end{bmatrix} && (2.6)
\end{aligned}$$

First, the approximation is substituted (2.2), then the parentheses are removed resulting in $I(x,y)$ term cancellation (2.3). Next, in (2.4), the second power is expanded in order to sort the terms (2.5) which leads to the introduction of $\mathbf{M}(x_0, y_0)$ (2.6). The matrix $\mathbf{M}(x_0, y_0)$ is a structure tensor, also referred to as second-moment matrix, and is derived from the gradients of the intensity function I for each pixel independently.

The energy E, as well as the structure tensor $\mathbf{M}$, is computed independently for each pixel in an image. Therefore, denote $E(x_0, y_0; u, v)$ as $E(u, v)$ and $\mathbf{M}(x_0, y_0)$ as $\mathbf{M}$ from now on.

This second-moment matrix $\mathbf{M}$ actually describes how do the intensity values change in the close neighbourhood of $[x_0, y_0]$. By an analysis of this matrix, it is possible to ascertain in which directions does the energy E increase or decrease and also how significantly does the

energy change. Eigenvectors and corresponding eigenvalues of $\mathbf{M}$ describe exactly this behaviour. Moreover, the matrix $\mathbf{M}$ is positive semi-definite, which means that it is symmetric and has non-negative eigenvalues.

Let λ_1, λ_2 be the eigenvalues of $\mathbf{M}$. There are three cases which can happen:

1. Both λ_1, λ_2 are small, which means that the energy function E is almost constant in all directions, i.e., no significant changes in the intensities.

2. $\lambda_1 \ll \lambda_2$ or $\lambda_1 \gg \lambda_2$, one of the eigenvalues is significantly larger that the other. This represents the case when the energy E is increasing only in one direction. Typically, these are edges.

3. Both λ_1, λ_2 are large and approximately the same. In this case, there are significant changes in the energy in all directions. This case is typical for corners, which are good keypoint candidates.

In general, having two non-negative numbers x and y such that $x \ll y$, then the following holds:

$$\frac{xy}{x+y} = x\frac{y}{x+y} = x\frac{1}{\frac{x+y}{y}} = x\frac{1}{\frac{x}{y}+1} \approx x\frac{1}{0+1} = x$$

In case of the second-moment matrix, this idea can be used for an estimation of the smaller eigenvalue:

$$\lambda_{min} \approx \frac{\lambda_1\lambda_2}{(\lambda_1+\lambda_2)} = \frac{det(\mathbf{M})}{trace(\mathbf{M})}$$

Such computation can be expensive and/or numerically unstable for matrices with small eigenvalues or those close to being singular. Not only for these reasons, another approach is used to evaluate the score of a pixel to be a corner. For each point $[x_0, y_0]$ in an image, define a Harris response R as follows:

$$R = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2 = det(\mathbf{M}) - k \cdot trace(\mathbf{M})^2,$$

where k is an empirically determined constant, $k \in [0.04, 0.06]$. Using this parameter, it is possible to adjust the sensitivity to corners of the detector. It can be tuned based on the specific characteristics of the images being analysed.

The choice of the Harris response R formulae has its own justification. Similarly to the eigenvalues analysis, also here are three cases which can happen:

1. The absolute value of the response $|R|$ is small, which corresponds to the case when both λ_1 and λ_2 are small, i.e., the region is flat.

2. Response $R < 0$ is negative, which corresponds to the case when one eigenvalue is significantly larger than the other, i.e., there is an edge.

3. Response $R > 0$ is positive, which corresponds to the case when both eigenvalues are large, i.e., there is a corner.

Input: intensities image I, sensitivity parameter k, Harris response threshold T, window size W
Output: Set of keypoints $[x_i, y_i]$

$keypoints \leftarrow \{\}$
$I \leftarrow smooth(I)$ ▷ Smooth out any noise
$I_x \leftarrow derivative(I, axis = x)$ ▷ Compute derivatives
$I_y \leftarrow derivative(I, axis = y)$
$A \leftarrow \sum_W I_x^2$ ▷ Compute elements of **M**
$B \leftarrow \sum_W I_x I_y$
$C \leftarrow \sum_W I_y^2$
$R \leftarrow AC - B^2 - k(A + C)$ ▷ Harris response
for each pixel $[x, y]$ in the image **do**
    if $R(x, y) \geqslant T$ **and** $R(x, y)$ is maximum in the neighbourhood **then**
        $keypoints.add([x, y])$
return *keypoints*

Figure 2.1: Harris detector algorithm.

A response value is computed for each pixel in an image. Then, the values are thresholded by a hyper-parameter T and *non-maximum suppression* is applied (on either 4-neighbourhood or 8-neighbourhood). Non-maximum suppression is a procedure which suppresses values which are not maximum in their neighbourhood. Together with the thresholding, it filters only locally significant keypoints. The response value can be also used as the previously mentioned sorting criterion for a case that there are too many detected keypoints. This is especially important for intensity multiplication invariance:

When there is a shift in the image intensity, $I' = I + b$, the response R is unchanged because only derivatives are used. Nonetheless, in case of a multiplicative change, $I' = a \cdot I$, the response R is increased as well. It may cause more keypoints to pass through the threshold filter, but the order of keypoints is preserved.

The final algorithm is described in the Figure 2.1. Note that usually a smoothing of the image or of the intensity function is applied in order to smooth out any noise. Also, border points must be taken care of. Usual strategies are to add a padding (extend the image using artificial values, e.g., zeros, mirrored values, extrapolated values, etc.) or, contrarily, crop the image (leave the border pixels out). And, as already mentioned, the detection is evaluated on up-scaled and down-scaled images as well in order to have features detected at different scales (and therefore acquire scale invariance).

2.3.2 Other detection methods

Another detection method, similar to the Harris detector, is Shi-Tomasi [5]. The response is calculated as the minimum eigenvalue of the structure tensor, representing the minimum amount of corner-like structure in the local neighbourhood. This generally provides more stable results compared to the Harris detector.

Features from Accelerated Segment Test (FAST) [6] is another representative. FAST is a corner detection algorithm that identifies corners based on the intensity difference around a pixel

by comparing it with a threshold. It is known for its computational efficiency and is widely used in real-time applications such as feature tracking and object recognition.

Besides corner and edge detectors, there are also blob detectors. One of the first and also very common blob detectors are based on the Laplacian of Gaussian (LoG). It involves convolving the image with a Gaussian kernel to smooth it and then applying the Laplacian operator (second-order differential operator). It detects intensity variations and highlights regions of abrupt intensity change. These often correspond to blobs. LoG can be further generalised to elliptical blob structures detection [7].

There are also neural network based methods for detecting keypoints in images. One representative is LF-Net [8]. LF-Net is a neural network architecture that learns to extract local features directly from images without relying on handcrafted detectors or descriptors. It employs a fully convolutional network to predict keypoints and descriptors, which can be used for image retrieval and other tasks requiring feature matching.

2.4 Feature description

As already mentioned, each feature is described by a vector $\mathbf{d} \in \mathbb{R}^D$. These descriptors are then used for further analysis, e.g., clustering or matching features between images. In theory, the descriptor can be based on any information from the local neighbourhood, like colours, structures, or textures. The key is for the descriptor not to change significantly when the visual appearance of the underlying object changes, because the set of descriptors is supposed to characterise the object.

There are two approaches of how to achieve such descriptors. First, the descriptor itself is designed in a way its value does not depend on the orientation and viewport. This approach was used in the past and lately has been abandoned. Alternatively, an invariant procedure can be designed to estimate a geometric normalisation parameters. The feature is then normalised before being described.

The particular mapping of a feature into the descriptor space $\mathbb{R}^D$ can be pre-defined or learnt by some machine learning models. One popular approach will now be described in detail in order to demonstrate how such methods can work.

2.4.1 SIFT descriptors

Scale Invariant Feature Transform (SIFT) is an algorithm to detect, describe and match local features in an image, which was invented by a Canadian computer scientist David G. Lowe in 1999 [9]. The descriptors are computed by analysing the gradient magnitudes and orientations within localised regions of an image. They provide distinctive representations of keypoints that can be matched across different images. They are robust to changes in scale, rotation, and illumination, making them highly effective in various real-world scenarios. The algorithm itself was originally protected by a patent, but it expired in 2020.

Before the descriptor is computed, the feature must be geometrically normalised. If a non-

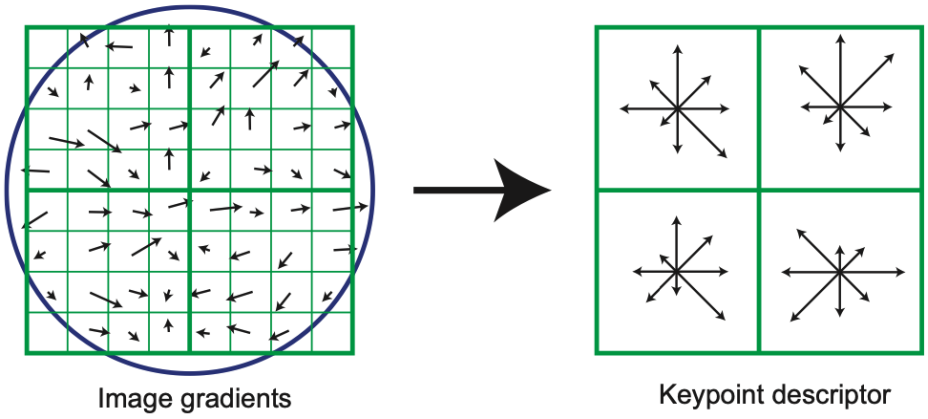


Figure 2.2: Gradients used to compute a SIFT keypoint descriptor [9].

scaled image is used while the feature was detected on a scaled image, the feature region must be transformed accordingly. Furthermore, orientation of the corresponding keypoint must be determined as well in order to gain rotation invariance.

The orientation is estimated using a histogram of gradients. Gradient orientations of pixels within a region around the keypoint are taken. From them, a histogram of 36 bins covering the 360 degree range of orientations is computed. Furthermore, each sample added to the histogram is weighted by a combination of two factors: (i) by the magnitude of the gradient and (ii) by a Gaussian weighted circular window with the centre in the keypoint. Original algorithm also suggests to use σ equal to 1.5 times the scale of the keypoint [9].

Having the histogram, its peak corresponds to the dominant direction of local gradients. The peak still captures a range of 10 degrees so exact value is determined by an interpolation. Three histogram values which are closest to the peak are used for fitting a parabola. This parabola then determines the exact value of the peak for better accuracy.

If there are multiple significant peaks in the histogram, i.e., there are local peaks which are within 80% of the highest peak, then these local peaks are used to create new keypoints with these orientations. Each of them is later described. It is an example of how a single feature can raise multiple descriptors.

Note that if the orientation of the object is known, it may replace this construction.

Having an orientation of a feature, together with its location and scale, a normalised patch (part of the image) is extracted for description. The description is based on surrounding gradients again.

Gradient magnitudes and orientations within the patch are computed and weighted by a Gaussian window function. Then, the patch is divided into 4×4 sub-regions and within each sub-region, a histogram with eight bins is computed. The value of each bin corresponds to the sum of gradient magnitudes falling into the bin. All the histograms values are then flattened into a single vector. Having 4×4 histograms with eight bins, the vector ends up having $4 \times 4 \times 8 = 128$ elements.

An example of this process is depicted in the Figure 2.2 [9]. It shows a 2×2 descriptor array computed from an 8×8 set of samples. The overlaid circle indicates the Gaussian weighting.

There are boundary effects when the descriptor changes upon a slight shift of a sample between orientation bins or even between histograms. To avoid this, the original paper proposes to do a trilinear interpolation to distribute a value of each gradient sample into adjacent histogram bins.

Finally, the feature vector is normalised in three steps:

1. The vector is normalised to unit length. This normalisation cancels changes in contrast of the patches.

2. In the next step, the unit feature vector values are clipped on 0.2, i.e., all values larger than 0.2 are decreased to this value. This reduces the impact of non-linear illumination changes (e.g., illumination changes of 3D surfaces with different orientations). These effects can change magnitudes of the gradients, but are less likely to change their orientations. The value 0.2 was determined experimentally [9].

3. Finally, the vector is normalised to unit length again.

SIFT descriptors have several useful properties. They are based on gradient orientations, which makes them robust to various illumination changes. Spatial binning of histograms brings a tolerance to small shifts in location. Explicit orientation normalisation is being made which causes the descriptor to be invariant to rotation and uniform scale as well as being tolerant to small affine changes and local deformations. Its popularity brought up a follow-up literature on possible improvements, such as Daisy [10], Histograms of Oriented Gradients (HOG) [11] or RootSIFT [12] descriptors.

2.4.2 Other description methods

Similar to SIFT features are Speeded-Up Robust Features (SURF) [13]. As the name suggests, it is a speeded-up version of SIFT. The time complexity improvement is mainly in the detection algorithm, which is different than the one proposed for SIFT. Descriptors use Wavelet responses in both horizontal and vertical directions in the neighbourhood of the keypoint. Similarly to SIFT, the neighbourhood is divided into sub-regions, whose values then form the final description vector.

Another example of a description method comes from Binary Robust Independent Elementary Features (BRIEF) [14], which is only a feature descriptor independent from a detection algorithm (although one is recommended in the original paper). BRIEF descriptors are binary feature descriptors which (unlike traditional gradient-based descriptors) operate by comparing pixel intensity differences between pairs of predetermined points within a local patch. These comparisons result in a binary string representing the presence or absence of certain intensity differences, providing a compact and efficient representation of local image features. BRIEF descriptors are robust to noise, but poor results are observed when there is a large in-plane rotation. Nevertheless, their computational efficiency makes them suitable for real-time applications on resource-constrained devices.

BRIEF was later used in Oriented FAST and Rotated BRIEF (ORB) [15]. It is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. ORB introduces an orientation assignment step to make keypoints rotation-invariant, unlike BRIEF, further improving its performance in scenarios with rotated or skewed images. Overall, ORB leverages the simplicity and computational efficiency of BRIEF while enhancing it with orientation handling and keypoint detection capabilities from FAST.

Last description methods representatives are methods based on (convolutional) neural networks. Already mentioned LF-Net [8] has a small neural network with three convolutional and two fully-connected layers which take image patches on the input and provide 256-dimensional descriptors on the output. Similarly, Deep Local Features (DELF) [16] is also based on convolutional neural networks. In addition, it uses an attention mechanism for keypoint selection.

2.5 Retrieval and ranking methods

When sets of descriptors $\mathcal{D}_I$ are assigned to each image I, a relevance score can be computed between a pair of images. In case of a retrieval, scores are computed between the query image and all database images. The database images are then sorted according to the score and retrieved.

In principle, score assignment is based on matching features, keypoints, and/or descriptors between images. One example of simple and naive approaches is the nearest neighbour search. For each descriptor from image A, the nearest descriptor in image B is found. If the distance is close enough, they are considered to match. The more matching features, the higher score.

While embracing simplicity, this approach suffers from inefficiency. Comparing all pairs of descriptors is a costly operation. Also, the necessity to store all descriptors in its full size might be challenging for large collections of images. Therefore, other approaches, such as Bag of Words (BoW), were introduced.

2.5.1 Bag of Words

This approach was inspired by a Natural Language Processing (NLP) technique where a text document is represented by the set of words it contains [17].

BoW in images [18] does not use original descriptors of features, but visual words instead. As it was described earlier, a vocabulary (set of visual words) is created using descriptors $\mathcal{D}_{train}$ from a training dataset. Then, features from the collection dataset and from query images can be assigned to these visual words. Descriptors with the same visual words are considered to be matching, as if they had a small distance in the nearest neighbour approach. The task then becomes finding features assigned to the same visual words.

This replacement of descriptors (D-dimensional vectors) for a simple visual word (represented by a cluster identifier) not only makes the matching task simpler and computationally less expensive, but also reduces the required memory load D times. Furthermore, the matching can be implemented efficiently.

Let the vocabulary have size N, i.e., there are N visual words. Let $\mathbf{X}$, $\mathbf{Y}$ be vectors of

size N, where an element at position i refers to the number of visual words w_i included in a corresponding image. Then, the score of such a pair of images can be computed using cosine similarity as follows:

$$score(I_X, I_Y) = \cos\varphi = \frac{\mathbf{X}^\top \mathbf{Y}}{\|\mathbf{X}\|\|\mathbf{Y}\|}$$

If the vectors $\mathbf{X}$, $\mathbf{Y}$ are stored normalised in the database (i.e., $\|\mathbf{X}\| = \|\mathbf{Y}\| = 1$), then the cosine similarity becomes a simple dot product. Because all elements of the vectors must be positive, the range of values is limited to the interval $[0; 1]$. Dot product is computed very efficiently. If an image is compared with itself, the score is maximum possible, $score(I_X, I_X) = \mathbf{X}^\top \mathbf{X} = \|\mathbf{X}\| = 1$. Conversely, images with strictly distinct visual words have the score equal to zero.

Not all visual words may have the same weight. Some of them might be more important or uncommon that the other. Most prevailing methods employ the Term Frequency-Inverse Document Frequency (TF-IDF) weighting scheme to gauge the significance of visual words. Term Frequency (TF) serves as a straightforward metric to assign weight to each visual word within an image. In TF, the importance of each visual word is presumed to be directly proportional to its frequency of occurrence in an image. While TF relates to the frequency of a term within a single image, Inverse Document Frequency (IDF) [19] relates to the occurrence of terms across a collection of images. It assigns a weight to each visual word based on its frequency of occurrence across images. Conceptually, IDF assigns lower weight to commonly occurring visual words, diminishing their influence, and higher weight to infrequent visual words, which are deemed more distinctive.

TF of a term (visual word) w in a document (image) I and IDF of a term (visual word) w in a collection of documents (images) $\mathcal{I}$ of size N ($|\mathcal{I}| = N$) are defined as:

$$tf(w, I) = \frac{\text{count of } w}{\text{count of all words}} = \frac{f_{w,I}}{\sum_{w' \in I} f_{w',I}}$$

$$idf(w, \mathcal{I}) = \log \frac{\#\text{images}}{\#\text{images with word } w} = \log \frac{N}{|\{w \mid w \in \mathcal{W}_I, I \in \mathcal{I}\}|}$$

where $f_{w,I}$ represents the raw count of term w in a document I and $\mathcal{W}_I$ is a set of all terms in I. In TF, if a visual word occurs repetitively in an image, it gets a higher weight, and vice versa. In IDF, if a visual word occurs in every single image, then it has zero weight and basically is dismissed. Similarly, if a visual word appears only in a single image, it has a high weight.

Nevertheless, this approach still requires to iterate over all collection images, even though most of the images have zero visual words in common. This loop can be very inefficient for large collections. This issue is addressed by *inverted files* and the idea is as follows: for each visual word, store a list of images in which this particular visual word occurs. Then, on query time, combine lists of those visual words which appear in the query image. The rest of collection images has no visual words in common and therefore the score is zero.

2.5.2 Other retrieval and ranking methods

A matching model, which extends previously explained and described BoW, is Hamming Embedding (HE) [20]. HE represents each local descriptor $\mathbf{d}_i$ with its quantised value $\boldsymbol{\mu}_i$ and a binary code $\mathbf{b}_i$ of B bits. All pairs of descriptors (or rather the binary codes) assigned to the same visual words are then used to compute the score between an image pair as follows:

$$score(I_X, I_Y) = \sum_{x \in I_X} \sum_{y \in I_Y} w(h(\mathbf{b}_x, \mathbf{b}_y)),$$

where I_X and I_Y are the images (represented as a set of descriptors $\mathbf{d}_i$ and more importantly binary codes $\mathbf{b}_i$), w is i weighting function, h is the Hamming distance and $\mathbf{b}_x$, $\mathbf{b}_y$ are the binary codes corresponding to the descriptors. The Hamming distance is the number of positions at which the corresponding symbols are different. Originally, the weighting function w was proposed to be a shifted Heaviside step function, i.e., $w(h) = 1$ if $h \leq \tau$ for a threshold τ, and 0 otherwise [20]. Nevertheless, a smoother weighting scheme (e.g., thresholded Gaussian function) is a better choice [21].

Another popular method is Vector of Locally Aggregated Descriptors (VLAD) [22]. It is based on aggregating local feature descriptors into a single vector representation for each image. First, local descriptors such as SIFT or SURF are extracted from keypoints detected in the image. These descriptors are then quantised into a visual vocabulary using techniques like k-means clustering. VLAD computes the difference between each local descriptor and the nearest visual word centroid (called *a residual*), accumulating these differences into a single vector for each cluster. Finally, these aggregated vectors are normalised and concatenated to form the VLAD representation, providing a compact yet informative description of the image content.

In [23], a framework is introduced that serves as a bridge between matching-based methodologies (like HE) and aggregated representations (VLAD in particular). This approach exploits the strengths of both paradigms to generate a robust image representation. By integrating an aggregation scheme with a selective kernel, a balanced synthesis of the two approaches is achieved. Furthermore, the resultant vector representation can be efficiently compressed to significantly reduce memory demands, while enhancing search efficiency. The compression is done by binarisation of the descriptor elements. This method will be referred to as Aggregated Selective Match Kernel (ASMK).

2.6 HOW descriptors

For the purpose of development, HOW features were used most often. These features were introduced by Tolias *et al.*, 2020 [24]. In this work, they propose a local feature detector and descriptor based on a deep network.

This neural network is fully convolutional. In the original work, ResNet18 and ResNet50 [25] are used as the backbone network. It generates a mesh of local descriptors, an attention map, and (as a combination of these two results) a global descriptor.

In the training phase, a global descriptor is generated for each image with image-level labels (i.e., the position of an object in the image is unknown). These global descriptors and *contrastive loss* are used to optimise the weights (parameters) of the neural network. Contrastive loss is a technique used in neural networks to learn embeddings by contrasting pairs of samples (images in this case). It operates by minimising the distance between similar (relevant) samples while maximising the distance between dissimilar (irrelevant) ones. This is achieved through a margin-based approach, where embeddings of similar samples are pushed closer together than those of dissimilar ones.

In the testing phase, the image is represented by retaining the most prominent local descriptors as indicated by the attention map. These descriptors are then used for image search using ASMK. They are also assigned to visual words (which had been pre-trained earlier). Local descriptors with low attention are ignored as well as the final global descriptor.

Because the neural network is fully convolutional, local descriptors really correspond to a local region in the original input image. The signal of each local descriptor can be backtracked through the network. It results in a particular group of pixels (usually a square or rectangle area), which has an influence on the descriptor value. Using this approach, each response can be approximately localised to a small region. In that case, the local descriptors can be though of as local features.

From the high-level view, the HOW network outputs a set of features for an input image. Each feature is characterised by: its approximate position $\mathbf{x}$; the scale s on which it was detected; the descriptor vector $\mathbf{d}$; corresponding visual word w; residual vector $\mathbf{r} = \mathbf{d} - \boldsymbol{\mu}_w$; and an attention (or importance, strength, weight) coefficient a, where $a \in [0, 1]$. The elements in the residual vectors are quantised in order to reduce the memory demands. Quantisation brings a trade-off between the memory load and efficiency. A visual word represents a region where the feature descriptor is located, the quantised residual then determines a smaller sub-region. Binary quantisation method is often used.

2.7 Commonly used datasets and evaluation

In the image retrieval community, there are several datasets which are often used for a comparison of different methods and approaches. These datasets contain a database of images, several query images and most importantly, the ground-truth information about which collection images are relevant for each query. The comparison of different approaches is then based on an evaluation protocol.

For the purposes of development of this thesis, two datasets have been used and will now be described. First and foremost used dataset was Revisited Oxford ($\mathcal{R}$Oxford) [26]. Another one was 24/7 Tokyo dataset [27].

		Image label			
		Easy	**Hard**	**Unclear**	**Negative**
Evaluation protocol	**Easy (E)**	POSITIVE	*ignored*	*ignored*	negative
	Medium (M)	POSITIVE	POSITIVE	*ignored*	negative
	Hard (H)	*ignored*	POSITIVE	*ignored*	negative

Table 2.1: Evaluation protocols used in $\mathcal{R}$Oxford.

2.7.1 Revisited Oxford Dataset

Original Oxford Buildings Dataset [28] consists of 5 062 high resolution images (1024×768). They were collected from Flickr (`www.flickr.com`) by searching for particular landmarks located in Oxford, e.g., All Souls, Balliol, Christ Church, etc. This collection was then manually annotated, raising 55 queries over which an object retrieval system can be evaluated.

Later on, $\mathcal{R}$Oxford [26] dataset was introduced. It contributes namely by fixing some annotation errors which are present in the original dataset and also by defining 15 new queries, which are more challenging compared to the original ones. Images from which all the queries had been cropped were excluded from the evaluation dataset and are provided separately. This way, off-line pre-processing techniques are unable to use query images.

Besides the two sets of images (query and collection), bounding boxes and the ground-truth are distributed along. Each query image has a Bounding Box (BBox), selecting the landmark in the image. It simulates a user attempting to remove background clutter as well as cases of large occlusion. Only features located inside of the regions cropped according to the BBox are used for the search. Each collection image has a ground-truth label in regards to a query image:

- **Easy**: Query landmark is clearly depicted in the image.
- **Hard**: Query landmark is depicted in the image, but with difficult viewing conditions.
- **Unclear**: The image may show the query landmark under consideration, but the information provided is insufficient to confidently determine its overlap with the queried region.
- **Negative**: None of the above applies.

These labels determine whether a retrieved image should be ignored, treated as positive or treated as negative. There are three evaluation protocols - Easy (E), Medium (M), and Hard (H). Each evaluation protocol defines how a label should be interpreted. All of them are delineated in the Table 2.1 and will be used in result reports and tables all across this thesis.

Examples of images in the $\mathcal{R}$Oxford dataset are shown in the Figure 2.3.

2.7.2 24/7 Tokyo Dataset

24/7 Tokyo Dataset [27] is another city-based collection of images, this time in Tokyo. There are 125 distinct locations, at each location there are 3 different viewing directions, and the pictures

Figure 2.3: Examples of images in the $\mathcal{R}$Oxford dataset.

Figure 2.4: Examples of images in the 24/7 Tokyo dataset.

are taken at 3 different times of day. This sums up to a collection of size $125 \times 3 \times 3 = 1125$ images.

The idea of this dataset is that every place is captured approximately in the same way and from the same place, but with different lightning conditions - during the daytime, during the sunset, and at night. Each image can serve as a query and the task is to find the other two. In terms of the evaluation protocol, for each query, the other two images photographed at the same location are considered to be positive. Other 6 images taken at the same place but with a different viewing direction (2 other viewing directions, 3 times of day, $2 \times 3 = 6$) are ignored for the purpose of evaluation. The rest of the images (124 locations, $124 \times 3 \times 3 = 1116$) is considered to be negative.

Some examples are shown in the Figure 2.4. It is a single place photographed (from left) during the daytime, during the sunset, and at night.

2.7.3 Evaluation metrics

A dataset, such as 24/7 Tokyo or $\mathcal{R}$Oxford, can be used to compare various methods and approaches. But first, a metric system must be established such that the performance of a retrieval system can be even measured.

Assume a standard classification task with two classes - relevant and irrelevant. Then there are 4 possible results of the classification describing the relationship between the classification label and the ground truth label. The options are depicted in the Table 2.2.

There are also other two terms which are often used to describe properties, or performance,

		Classification	
		Relevant	**Irrelevant**
Ground truth	**Relevant**	True Positive (TP)	False Negative (FN)
	Irrelevant	False Positive (FP)	True Negative (TN)

Table 2.2: Binary classification results.

of a classification system. Let items labelled as relevant be called *selected items*. Then:

- **Precision** means: How many selected items are relevant? (precision = $\frac{\text{TP}}{\text{TP+FP}}$)
- **Recall** means: How many relevant items are selected? (recall = $\frac{\text{TP}}{\text{TP+FN}}$)

Image retrieval is not a standard classification task, but after all, the goal is to separate or sort relevant and irrelevant images. A list of images is sorted according to some relevancy score and the desired order is to have all the relevant images first, and irrelevant images after. Therefore terms like *precision@k* and *recall@k* are used. The idea is simple - take top k images (with the highest score) and compute the precision/recall based only on these k samples. Images in the top k list are looked upon as if they were labelled (classified) as positive so the Table 2.2 is applicable for the computation in this case too.

Example: Assume having 10 collection images and sort them as `RRIRRIIIIR` (where `R` stands for ground truth relevant and `I` for irrelevant). Let $k = 4$ and compute precision@4 and recall@4:

$$\text{precision@4} = \frac{3}{3+1} = 75\%$$

$$\text{recall@4} = \frac{3}{3+2} = 60\%$$

This approach has one significant drawback and that is the independence of the total number of relevant images according to the ground truth. Imagine two queries, one with 5 relevant answers and the other with 500 relevant answers. It is unclear which k should be chosen.

There is another evaluation metric used to evaluate a single query which is more suitable for the retrieval task. It is Average Precision (AP). This AP metric is based on the previously described precision and recall. In fact, AP is the area under a curve, where recall is on the X axis and precision is on the Y axis. The previously mentioned example with 10 collection images (`RRIRRIIIIR`) has AP equal to 79.11% and is showed in the Figure 2.5 as well as the ideal result (`RRRRRIIIII`) with AP equal to 100%. Green element depict relevant images while red elements mean irrelevant images (according to the ground truth). Every k-th element from left shows the recall@k on the X axis and precision@k on the Y axis: see again the example mentioned above with precision@4 equal to 75% and recall@4 equal to 60%, which is referred in the Figure 2.5 (a).

The AP metric is used to evaluate a single query. For an evaluation of multiple queries (for example all queries in a dataset), Mean Average Precision (mAP) is used. The mAP metric is computed as an average over all APs for all queries.

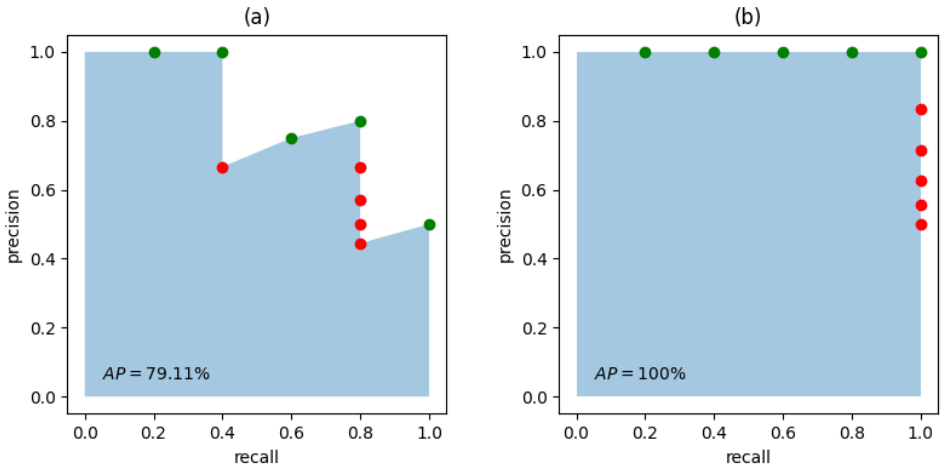


Figure 2.5: Examples of the AP for (a) `RRIRRIIIIR` with $AP = 79.11\%$ and (b) `RRRRRIIIII` with $AP = 100\%$.

Chapter 3

Spatial verification

Having a query and corresponding list of collection images sorted by a retrieval system, the first images of the list are expected to have a lot of visual features in common with the query. Ideally, they should contain the same object as the query. If that is the case, there might exist a geometric transformation which maps the object from one image into the other.

The process of estimating such transformations is called *spatial verification*, sometimes abbreviated as SV. It can be then used to re-sort the images according to various criteria or to find a specific subset of images (e.g., zoomed images or pictures taken from a different and predefined location), just to name a few use cases. However, the utilisation of the transformation estimation does not necessarily apply only for image retrieval, but also elsewhere (e.g., image stitching).

The spatial verification is described in this chapter. First, a commonly used algorithm is introduced together with its possible enhancements. Then, its usage for transformation estimation is shown. Possible improvements and different approaches to this transformation estimation are discussed in the following part. Lastly, some specific use cases are described at the end of the chapter.

3.1 RANSAC

Assume having two sets of measurements, also called *data points*. First set of measurements is generated from a distribution with unknown parameters which one may want to estimate. This distribution will be referred to as *the underlying ground-truth distribution*. Second set of data points is generated from a different distribution. Furthermore, both sets of measurements may include some Gaussian noise. If all the measurements originate only from the first distribution (i.e., the second set is empty), its model parameters can be estimated using methods such as Maximum Likelihood Estimation (MLE) or Ordinary Least Squares (OLS). The same holds also for a case when the second set is not empty, but the measurements are distinguishable. Only data points belonging to the first distribution would be taken into account for the parameters estimation.

If the data points are indistinguishable, another methods should be chosen. These methods

should be *robust*. It means that measurements which are inconsistent with a model should have low (or even better - no) impact on its estimated parameters. Examples of such methods are M-estimators or trimming [29], or Random Sample Consensus (RANSAC), which will be described in further details in this section. Some applications may only need to separate the data points into the two sets (e.g., anomaly detection, spam classification, etc.), while others might utilise the concrete model parameters as well (e.g., homography estimation, stock market prediction, etc.).

RANSAC is a paradigm for model fitting originally introduced by M.A. Fischler and R.C. Bolles in 1981 [30]. It is an iterative algorithm for robust model estimation and has become one of the most popular estimation methods in computer vision community [31].

The input of the RANSAC algorithm is a set $\mathcal{U}$ of data points from a measurements space $\mathbb{X}$. The algorithm itself consists of two steps executed repeatedly: a parameters generation step and the verification step.

In the generation step, model parameters θ from a parameter space Θ are estimated using a subset $\mathcal{S}$ of size m. This subset $\mathcal{S}$ is selected randomly from the input data set $\mathcal{U}$. Particular model parameters θ are called *a hypothesis*. Each hypothesis is then verified.

In the verification step, a hypothesis is evaluated using an error function $\rho(\theta, \mathbf{x}) \to \mathbb{R}$ which assigns an error value to each data point. This error is then compared with a hyper-parameter threshold Δ and if the error is lower or equal than Δ, the data point is considered to be consistent with the proposed hypothesis. Consistent data points are called *inliers*. It is also commonly said that inliers *support* a hypothesis or that a hypothesis has a *support* of n inliers. On the contrary, if the error is higher than Δ, then the point is considered to be inconsistent with the hypothesis and is called *an outlier*.

The output of the algorithm are model parameters θ^* such that θ^* optimise a cost function $J(\theta, \mathcal{U}, \Delta) \to \mathbb{R}$ which typically returns the support for a particular hypothesis. Therefore the hypothesis with maximum support over all the iterations is chosen:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\theta, \mathcal{U}, \Delta) = \underset{\theta}{\operatorname{argmax}} \left|\{\mathbf{x} \in \mathcal{U} \mid \rho(\theta, \mathbf{x}) \leq \Delta\}\right| = \underset{\theta}{\operatorname{argmax}} \sum_{T \in \mathcal{U}} [\![\rho(\theta, \mathbf{x}) \leq \Delta]\!],$$

where $[\![\cdot]\!]$ is the Iverson bracket which returns 1 if the statement is true and 0 otherwise.

An essential question to answer is how exactly does the hypothesis generation work. Before a hypothesis is created, it is needed to determine what is the minimum number m of data points to estimate model parameters uniquely. For example, a line is defined by two distinct points and a homography is defined by four distinct pairs of corresponding points. After m is determined, a subset $\mathcal{S}$ of size m should be randomly selected from the input data points $\mathcal{U}$. This random subset is called *a sample*. A sample can be *contaminated*, if it contains at least one outlier of the underlying ground-truth model, or *uncontaminated*, if it contains all inliers. Only the latter ones are of interest since the parameters of the underlying model can be estimated out of these samples while parameters computed from data points including outliers can be arbitrary. The lower m, the lower probability of generating a contaminated sample. Hence, the sample $\mathcal{S}$ is of

the minimum size m.

Having a random sample $\mathcal{S}$, a hypothesis θ fitting the data can be uniquely and exactly established by a function $f(\mathcal{S}) \to \Theta$, which computes the model parameters out of the random data points. For example, a line would be computed from the two distinct points as $f_{line}(\mathbf{x}, \mathbf{y}) = \underline{\mathbf{x}} \times \underline{\mathbf{y}}$ (where $\underline{\mathbf{x}}$ is point $\mathbf{x} = [x, y]^\top$ in homogeneous coordinates, $\underline{\mathbf{x}} = [x, y, 1]^\top$, and a line is represented in a standard form, $ax + by + c = 0$). A homography would be computed from the four distinct pairs of corresponding points using the Direct Linear Transformation (DLT) algorithm [31].

Another question would be how many times should be the generation and verification steps repeated. Although the number of iterations (generated hypotheses to be evaluated) can be set statically to a fixed number by a hyper-parameter, it can be also computed dynamically based on the probability that a better hypothesis will be generated.

Assume having N points, I of which are inliers of the underlying ground-truth model. Then the probability of generating an uncontaminated (all-inlier) sample is:

$$P(\text{inlier sample}) = \frac{\binom{I}{m}}{\binom{N}{m}} = \prod_{j=0}^{m-1} \frac{I-j}{N-j} \approx \left(\frac{I}{N}\right)^m = \varepsilon^m$$

where ε is the inlier ratio I/N.

Let $\eta \in (0, 1)$ be a confidence that no better sample will be generated. Then in k-th iteration it holds that:

$$P(\text{bad model } k \text{ times}) = (1 - P(\text{inlier sample}))^k = (1 - \varepsilon^m)^k < 1 - \eta$$

therefore at least k samples are required to find a solution with confidence η. From this equation, the number of iterations k can be computed as follows:

$$\begin{aligned} (1 - \varepsilon^m)^k &< 1 - \eta \\ \log(1 - \varepsilon^m)^k &< \log(1 - \eta) \\ k \log(1 - \varepsilon^m) &< \log(1 - \eta) \\ k &\geq \frac{\log(1 - \eta)}{\log(1 - \varepsilon^m)} \end{aligned} \tag{3.1}$$

The larger size of the minimum sample m is, the more iterations are needed for a confident estimate. Similarly, the lower the inlier ratio ε is, the more iterations are needed as well. Demonstration of exact values is provided in the Table 3.1.

In practise, the inlier ratio ε is not known apriori, therefore the current highest support is used instead. Each time a hypothesis with a higher support is discovered, the probabilities are re-computed as well as the maximum number of iterations. The algorithm is terminated once the iteration number exceeds the maximum number of iterations.

Nevertheless, this computation of maximum number of iterations assumes that an uncontaminated sample generates a model which is consistent with all inliers of the underlying ground-

m \ ε	15%	20%	30%	40%	50%	70%
2	$\eta = 0.950$; $k = 130$	73	32	17	10	4
	$\eta = 0.990$; $k = 200$	110	49	26	16	6
	$\eta = 0.999$; $k = 300$	170	73	40	24	10
3	890	370	110	45	22	7
	1400	570	170	70	34	11
	2000	860	250	100	52	16
4	5900	1900	370	120	46	11
	9100	2900	570	180	71	17
	$1.4 \cdot 10^4$	4300	850	270	110	25
8	$1.2 \cdot 10^7$	$1.2 \cdot 10^6$	$4.6 \cdot 10^4$	4600	770	50
	$1.8 \cdot 10^7$	$1.8 \cdot 10^6$	$7.0 \cdot 10^4$	7000	1200	78
	$2.7 \cdot 10^7$	$2.7 \cdot 10^6$	$1.1 \cdot 10^5$	$1.1 \cdot 10^4$	1800	120
12	$2.3 \cdot 10^{10}$	$7.3 \cdot 10^8$	$5.6 \cdot 10^6$	$1.8 \cdot 10^5$	$1.2 \cdot 10^4$	210
	$3.5 \cdot 10^{10}$	$1.1 \cdot 10^9$	$8.7 \cdot 10^6$	$2.7 \cdot 10^5$	$1.9 \cdot 10^4$	330
	$5.3 \cdot 10^{10}$	$2.7 \cdot 10^9$	$1.3 \cdot 10^7$	$4.1 \cdot 10^5$	$2.8 \cdot 10^4$	500
18	$2.1 \cdot 10^{15}$	$1.1 \cdot 10^{13}$	$7.7 \cdot 10^9$	$4.4 \cdot 10^7$	$7.9 \cdot 10^5$	1800
	$3.2 \cdot 10^{15}$	$1.8 \cdot 10^{13}$	$1.2 \cdot 10^{10}$	$6.7 \cdot 10^7$	$1.2 \cdot 10^6$	2800
	$4.8 \cdot 10^{15}$	$2.6 \cdot 10^{13}$	$1.8 \cdot 10^{10}$	$1.0 \cdot 10^8$	$1.8 \cdot 10^6$	4200
30	∞	∞	$1.3 \cdot 10^{16}$	$2.6 \cdot 10^{12}$	$3.2 \cdot 10^9$	$1.3 \cdot 10^5$
	∞	∞	$2.1 \cdot 10^{16}$	$3.1 \cdot 10^{12}$	$4.9 \cdot 10^9$	$2.0 \cdot 10^5$
	∞	∞	$3.1 \cdot 10^{16}$	$5.1 \cdot 10^{12}$	$7.4 \cdot 10^9$	$3.1 \cdot 10^5$
50	∞	∞	∞	∞	$3.4 \cdot 10^{15}$	$1.7 \cdot 10^8$
	∞	∞	∞	∞	$5.2 \cdot 10^{15}$	$2.6 \cdot 10^8$
	∞	∞	∞	∞	$7.8 \cdot 10^{15}$	$3.8 \cdot 10^8$

Table 3.1: Number of iterations needed for a particular minimal sample size m (rows), inlier ratio ε (columns) and confidence η (values 0.950, 0.990 and 0.999 for three rows in each cell, respectively) [32]. Values are given only as an order of magnitude.

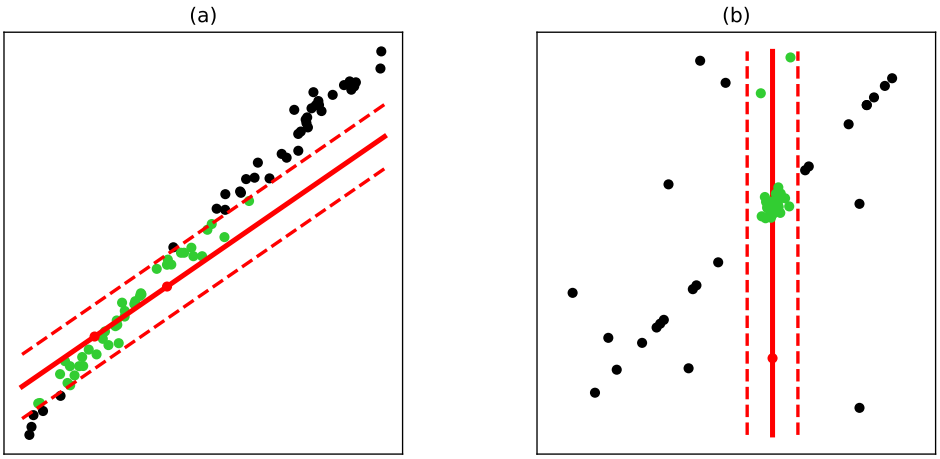


Figure 3.1: Examples of RANSAC assumptions violation [33]. Image (a) shows a case when hypothesis is generated from an uncontaminated sample, but not all data points are consistent with it. Image (b) shows a case of a contaminated sample with a large support.

truth model. However, this assumption is often violated due to a noise in the measured data points. Violation of the assumption causes that the quality of the resulting model is affected and/or the number of iterations needed is higher since the estimated inlier ratio is lower than the ground-truth ratio, hence more iterations are needed.

Another assumption for the algorithm to work with the demonstrated probability guarantee is that a hypothesis generated by a contaminated sample has a small number of inliers. This also is not guaranteed since there can be cases when this assumption is violated. A wrong estimate of parameters can be returned because of its high support.

Cases of the assumptions violation are depicted in the Figure 3.1. A few measurements are used to estimate a line parameters. In the left picture, uncontaminated sample generates a hypothesis which does not consider all other points as inliers, which can lead to imprecise estimate as well as longer runtime. In the right picture, a contaminated sample generates a hypothesis with a large support which leads to a wrong estimate [33]. Estimated line is red, dashed lines represent the inlier threshold Δ, and red points form the sample used to generate the line hypothesis. Green points are inliers consistent with the line while black points are outliers. The underlying ground-truth line in both examples is $y = x$.

Full algorithm consists of looking for a hypothesis with the highest possible support over multiple iterations until the probability of finding a better model is low enough. In each iteration, a random sample of input data points is generated. Using this random sample, model parameters are estimated and evaluated. The evaluation resides in computation of errors for each data point given the generated model. Data points with error lower than a threshold are considered to be inliers and their count forms the objective function which is being maximised. At the end, the model with the highest support is returned. The whole algorithm is described also in the Figure 3.2.

Input: data points $\mathcal{U}$, minimum sample size m, model function $f(\mathcal{S})$, inlier threshold Δ, confidence η
Output: Model parameters θ^*, hypothesis support s^*

$k \leftarrow 0$ ▷ Iteration number
$k_{max} \leftarrow \infty$ ▷ Maximum iterations
$\theta^* \leftarrow$ undefined ▷ Best hypothesis
$s^* \leftarrow -\infty$ ▷ Best support
repeat
    Select random sample $\mathcal{S}$ of size m
    Estimate parameters $\theta = f(\mathcal{S})$
    Count the support as $s = J(\theta, \mathcal{U}, \Delta) = \left|\{\mathbf{x} \in \mathcal{U} \mid \rho(\theta, \mathbf{x}) \leq \Delta\}\right|$
    if $s > s^*$ **then** ▷ So far the best
        $\theta^* \leftarrow \theta$
        $s^* \leftarrow s$
        $k_{max} = \log(1-\eta)/\log(1-(s/|\mathcal{U}|)^m)$
    $k \leftarrow k + 1$
until $k \geq k_{max}$
return θ^*, s^*

Figure 3.2: RANSAC algorithm.

3.2 RANSAC variants

There are several potential enhancements to the basic RANSAC algorithm. These improvements can help to estimate the parameters of the final model more precisely or speed up the computation. Sometimes, prior knowledge or heuristics can be used to achieve such improvements.

A few enhancement examples will be described in the following subsections. Each of these RANSAC variants offers advantages in different scenarios, providing researchers and practitioners with flexibility in choosing the most suitable algorithm based on the characteristics of their data and computational requirements. Together, they show the flexibility, extendability, and effectiveness of the original algorithm.

3.2.1 LO-RANSAC

First enhancement is Locally Optimised Random Sample Consensus (LO-RANSAC) [34]. This adaptation aims to estimate model parameters more precisely, resulting in an increase in the number of inliers and consequently, a reduction in the number of iterations according to the Equation (3.1).

As it was already described, the number of iterations predicted from the mathematical model is significantly lower than necessary due to assumptions which only rarely hold in practice. One of the assumptions is that *an uncontaminated sample generates a model which is consistent with all inliers of the underlying ground-truth model.* As depicted in the Figure 3.1 (a), this assumption is easily violated by a noise in the data. And LO-RANSAC validates the aforementioned assumption by employing local optimisation (often abbreviated as LO) on the solution estimated from the random sample.

In the RANSAC algorithm, when a new hypothesis (generated from a minimal sample) is

evaluated and happens to be so far the best one, the model parameters are simply saved, as in the Figure 3.2 on the line 10. The change in LO-RANSAC is that instead of simply saving the parameters, the local optimisation step is carried out using current inliers and the locally optimal parameters are saved instead.

The optimisation strategy can vary. The original paper [34] proposes four methods - simple (linear optimisation, e.g., OLS), iterative (iteratively perform linear optimisation and progressively reduce the threshold), inner RANSAC (new RANSAC with samples being drawn from current inliers and evaluated on all data points) and inner RANSAC with iteration (combination of the *inner RANSAC* and *iterative* strategies). Nevertheless, any other optimisation strategy can be used too, such as MLE or Levenberg-Marquardt algorithm [35] for non-linear least squares problems.

This locally optimal parameters estimation deals with the noise present in the data, which makes the original assumption valid and therefore the number of inliers is increased and the number of iterations decreased. Concurrently, an experiment [34] verified that the number of parameters updates (and hence number of optimisation steps) is lower than logarithm of the number of samples drawn. Thus, local optimisation does not slow the whole procedure down. On the contrary, the benefit in iterations decrease speeds up the whole procedure.

3.2.2 R-RANSAC

Matas and Chum pointed out that the speed of the RANSAC algorithm is influenced by two key factors [36]. Firstly, the level of contamination, which determines the number of samples that have to be randomly drawn in order to have a confidence guarantee in the optimality of the solution. Next, the time spent by evaluating the quality of each hypothesis scales with the size of the dataset. Typically, a large number of assessed model parameters are derived from contaminated samples and are consistent with only a small fraction of the data. This insight can be leveraged to extensively enhance the speed of the RANSAC algorithm.

They propose to implement a two-step randomised procedure for the hypothesis evaluation (verification) step. First, a pre-check is performed. A hypothesis is verified only over a small number d of measurements randomly drawn out of all N data points, $d \ll N$. If the number of measurements consistent with the hypothesis is low, it is concluded with a high confidence that the hypothesis is erroneous. On the contrary, if the number of inliers is high (i.e., the pre-check passed), full evaluation on all N data points is performed.

This two-step hypothesis verification significantly decreases the number of overall evaluations, hence increases the speed. As it was shown in the original work, the randomisation of the model verification does not change the nature of the solution since the basic RANSAC is already a randomised algorithm. The result is still correct only with a certain probability.

This variant of the algorithm nicely demonstrates the flexibility and universality of the RANSAC. The idea that only a fraction of measurements is evaluated in order to speed up the computation can be further extended even to the space of hypotheses. For example, in the spatial verification, the fundamental question is whether two images contain the same object.

When a hypothesis has a large enough score, the image pair can be considered to be verified and the rest of the hypotheses does not need to be evaluated anymore.

3.2.3 PROSAC

Another RANSAC adaptation is Progressive Sample Consensus (PROSAC) [37], which aims to speed up the standard algorithm. It addresses limitations of the original algorithm by prioritising the selection of more reliable samples early in the process.

Given a (heuristics) method to sort the input data measurements according to a *quality* or *reliability*, the PROSAC algorithm can employ this information in the process. Assuming that the order of sorted samples is better than random, PROSAC uses the first portion of data to generate the hypotheses. In other words, the samples which are more likely to be uncontaminated are drawn earlier. The main idea is that hypotheses generated from this high quality data subset should be more likely to be close to the underlying ground-truth distribution. They give a large support and low probability of generating a better hypothesis.

The quality measure is specified depending on a particular problem and prior knowledge. For example, SIFT descriptors are being matched in the original paper [37]. The quality measure was defined as the ratio of the distances in the vector space of the best and second to best match.

The size of the ordered subset of data used for hypothesis generation is gradually increased, up to the whole set. This implies that all possible sample combinations are drawn with a non-zero probability. After all, the same samples as in standard RANSAC are drawn, only in a different order, prioritising more promising samples first.

The sorted subset is used only for hypothesis generation, each hypothesis is verified against the whole set of input measurements.

The algorithm has two stopping criteria, one of which must be satisfied. First, *maximality* criterion, which follows the original RANSAC stopping condition - the probability of there being an undiscovered solution with a higher number of inliers is sufficiently low. Second, *non-randomness* criterion, stating that the probability of current inliers belonging by chance to an incorrect model is low enough.

Sampling from progressively larger subsets can leads to substantial computational efficiency gain. In contrast to RANSAC, PROSAC demonstrated an increase in speed by a factor of more than one hundred when tackling non-trivial problems, as shown in [37].

3.3 RANSAC for spatial verification

At this point, local features for an image are detected and described. Each local feature is identified by a location $[x, y]$ and scale s, where it was detected. The scale s is determined by the image up-scale or down-scale coefficient, which was used for detection. The location coordinates are normalised into the original, non-scaled, image. It means that if a keypoint was detected at coordinates $[100, 200]$ in a two times up-scaled image, the location of the keypoint in the original non-scaled image is $[50, 100]$. So the local feature carries its information $x = 50$,

$y = 100$, and $s = 2.0$, along with the descriptor $\mathbf{d}$, visual word w, and possibly other properties.

Each image is be represented by a set of local features with the above mentioned properties. Since the properties contain some geometric information, they can be used to establish a transformation between two images. If one image is the query and the other image is a collection image, it is said that the collection image is *spatially verified* when the transformation is established.

In the first place, a type (group) of used transformations must be decided. There are several of them, from a simple translation to a fully flexible homography. Then, correspondences between points in the pair of images are established. These correspondences significantly reduce the number of combinations - the search space of transformations to be examined. Lastly, the transformation itself is established based on properties of the local features, or namely of the local geometry (geometry present in the feature keypoint neighbourhood).

3.3.1 Types of transformations

First of all, points in an image (which is a two dimensional plane) are represented by a pair of coordinates (x, y). Thus, the image is commonly identified with a vector space $\mathbb{R}^2$, where each image point (x, y) is identified by a vector $\mathbf{x} = [x, y]^\top \in \mathbb{R}^2$. Furthermore, their homogeneous coordinates are often used instead during computations, $\underline{\mathbf{x}} = [x, y, 1]^\top$. The homogeneous vector is always normalised in such a way the last coordinate equals to one, i.e., vectors $[kx, ky, k]$ for $k \neq 0$ are equivalent and refer to the same point (x, y) in an image. Adding also vectors with the last coordinate equal to zero results in a *projective space* $\mathbb{P}^2$ [31], which will be described later in the Section 4.3.1.

For now, consider only homogeneous vectors with non-zero third coordinate. This representation introduces two major benefits and consequences: (1) projective transformations can be used, as will be described in this section, and (2) simple translations can be expressed in a basic matrix notation by multiplication, without the necessity of introducing additions to the formula:

$$\begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad \text{corresponds to} \quad \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Therefore, all 2D image transformations can be represented by a 3×3 matrix.

There are several basic types (groups) of 2D geometric image transformations: translation, rigid, similarity, affine, and projective [38]. All of them are reproduced in the Table 3.2 and in the Figure 3.3. Also, a visualisation is provided in the Figure 3.4. Each type has a specific format, properties, and interpretation. First three of them (translation, rigid and similarity transformations) are clear - they add support for translation, rotation, and a isotropic (single) scale, respectively. Affine transformations support also anisotropic (dual) scale and shear. They hold significance due to their close relation to projective transformations.

A projective transformation is the most general 2D image transformation, which maps four points onto any arbitrary four points. This transformation is also referred to as a *homography*

Name	#DOF	Preserves	Supports
translation	2	orientation + ...	translation
rigid (Euclidean)	3	lengths + ...	... + rotation
similarity	4	angles + ...	... + isotropic scale
affine	6	parallelism + ...	... + anisotropic scale, shear
projective	8	straight lines	... + perspective transform

Table 3.2: Hierarchy and properties of basic 2D geometric image transformation types [38].

$$\text{(translation)}\quad \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(rigid)}\quad \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(similarity)}\quad \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{(affine)}\quad \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(projective)}\quad \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Figure 3.3: Matrices for transformation types.

matrix. Because of the homogeneous vectors, the transformation is defined up to scale (hence only eight Degrees of Freedom (DoF), not nine, see the Table 3.2). Defining the full matrix, including the scale, would have no influence since the point coordinates would be normalised anyway. It is common to normalise the projective matrix in such a way that the very last element, h_{33}, equals to 1.

All these transformation groups are closed under composition and inverse. It means that a composition of, for example, two affine transformations is still an affine transformation and even its inverse is an affine transformation.

In addition, there are three special transformation sub-types used later in this thesis. They are restricted versions of the basic types defined earlier, meaning that they have an additional constraint limiting their flexibility as well as the number of parameters (and DoF).

1. A combination of a translation and an isotropic scale. It is a special case of a similarity transformation with no rotation, i.e., $\theta = 0$. Hence, it has three DoF. See the Figure 3.5 (1).

2. A combination of a translation and an anisotropic scale (different scale coefficients for each axis). It is not a similarity anymore, but a special case of an affine transformation with no shear. Hence, it has four DoF. See the Figure 3.5 (2).

3. A combination of a translation and a vertical shear. It is a special case of an affine transformation with five DoF. See the Figure 3.5 (3).

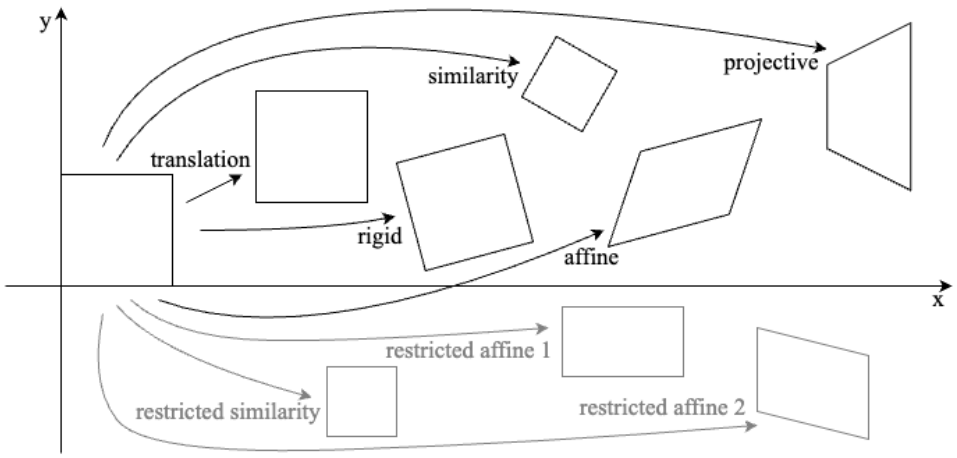


Figure 3.4: Basic types of 2D geometric image transformations [38]. Additionally, restricted sub-types are added at the bottom.

(1) (restricted similarity)	(2) (restricted affine 1)	(3) (restricted affine 2)
$\begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} a & 0 & t_x \\ b & c & t_y \\ 0 & 0 & 1 \end{bmatrix}$

Figure 3.5: Matrices for restricted transformation sub-types.

The matrices for these restricted transformations are reproduced in the Figure 3.5. Some of these types and sub-types will be used as models for transformation estimation between two images. All of them are also visualised in the Figure 3.4.

3.3.2 Point correspondences

In order to estimate parameters of a line, two points in space must be provided. The points are *primitives* (or basic entities) used for estimation of the model parameters, in this case the line. Given n measurements, there are $\binom{n}{2} \approx n^2$ combinations of point couples, resulting in n^2 line parameters. In terms of the RANSAC algorithm, there are n^2 possible hypotheses.

For 2D geometric image transformations, the situation gets more complex. The transformation defines a mapping of points in one image onto points in another image, so the primitives defining a particular transformation are tentative point correspondences. The most general transformation type is the projective transform, as described in the Section 3.3.1, which is defined by four point correspondences.

Let there be two images, with n detected points each, i.e., each image has n points for the transformation estimation. A single model is defined by four point correspondences. Therefore, there are $\binom{n}{4}$ combinations of how to select the foursome of points in a single image. The same amount applies for the second image. At the end, there are 4! ways how to assign the tentative

correspondence relations. Putting that all together, there are

$$\binom{n}{4} \cdot \binom{n}{4} \cdot 4! \approx n^4 \cdot n^4 = n^8$$

possible hypotheses. It is unbearable even for a small n and also hostile to the RANSAC algorithm in terms of the inlier ratio ε, as defined in the Section 3.1. One point in the first image can correspond to every single point in the second image while at most one tentative correspondence is correct. This leaves an upper bound on the inlier ratio $\varepsilon \leq \frac{1}{n}$. Considering an example with $n = 7$, there must be more than 5 900 iterations according to the Table 3.1 ($m = 4$, $\varepsilon \approx 15\%$).

In order to reduce the number of iterations, either the size of the minimum sample has to be decreased or the inlier ratio has to be increased. Ideally both. Minimum sample size is discussed in the subsequent Section 3.3.3, now the focus is given to the inlier ratio.

Considering all possible combinations of points as tentative correspondences is superfluous. Instead, the idea is to restrict the consideration to only those correspondences that have a potential to be correct. The simplest restriction is to pair only those points which belong to the same visual word.

Points in an image are detected and after the detection, they are described, assigning each feature a descriptor $\mathbf{d}$ and corresponding visual word w. The descriptor is dependent on the neighbourhood of the detected point and points with visually similar neighbourhoods should have similar descriptors, hence they should have the same visual words assigned. Pairing only features belonging to the same visual word means that only visually similar regions are considered for tentative correspondences.

The basic algorithm for the tentative correspondences generation is as follows: assume a pair of images is represented by a pair of feature sets, each feature $\mathbf{F}$ has an assigned visual word w. For each visual word w_i, take a subset of features from the first image $I1$ belonging to the visual word w_i ($\mathcal{F}_{I1}^{(w_i)}$) and in the same way take also a subset of features from the second image $I2$ belonging to the very same visual word w_i ($\mathcal{F}_{I2}^{(w_i)}$). Their Cartesian product defines tentative correspondences $\mathcal{C}^{(w_i)} = \mathcal{F}_{I1}^{(w_i)} \times \mathcal{F}_{I2}^{(w_i)}$. This $\mathcal{C}^{(w_i)}$ is just a subset of all tentative correspondences generated from a single visual word w_i. The set of all tentative correspondences $\mathcal{C}$ is a union over all visual words:

$$\mathcal{C} = \bigcup_{w_i \in \mathcal{W}} \mathcal{C}^{(w_i)}$$

where $\mathcal{W}$ is a set of all visual words.

A subset $\mathcal{C}^{(w)}$ can be further restricted or completely disregarded. Having a lot of features belonging to the same visual word w, the number of tentative correspondences gets too large and the inlier ratio decreases. This resembles the original issue. A common technique is to completely discard the subset $\mathcal{C}^{(w)}$ if its size is larger than a threshold. If the tentative correspondences can be sorted according to their quality, instead of discarding all of them, only a top part is kept. The quality can be measured based on the similarity of the corresponding descriptors or residuals and it can be also returned together with the correspondences in order to be used later

Input: input features $\mathcal{F}_{I1}$ and $\mathcal{F}_{I2}$ sorted by visual words, corresponding visual words $w(\mathbf{F})$
Output: List of all tentative correspondences $\mathcal{C}$

$\mathcal{C} \leftarrow \{\}$
$i, j \leftarrow 0, 0$
while $i < \text{length}(\mathcal{F}_{I1})$ & $j < \text{length}(\mathcal{F}_{I2})$ **do**
  if $w(\mathcal{F}_{I1}[i]) = w(\mathcal{F}_{I2}[j])$ **then** ▷ The same visual word
    $j_{start} \leftarrow j$
    while $j < \text{length}(\mathcal{F}_{I2})$ & $w(\mathcal{F}_{I1}[i]) = w(\mathcal{F}_{I2}[j])$ **do** ▷ Fix i and iterate over j
      $\mathcal{C}.add([i, j])$
      $j \leftarrow j + 1$
    $j \leftarrow j_{start}$
    $i \leftarrow i + 1$
  else if $w(\mathcal{F}_{I1}[i]) < w(\mathcal{F}_{I2}[j])$ **then** ▷ Different visual word, increment i or j
    $i \leftarrow i + 1$
  else
    $j \leftarrow j + 1$
return $\mathcal{C}$

Figure 3.6: Basic unrestricted algorithm for tentative correspondences generation.

on, e.g., for a weighted evaluation (as will be described in the Section 3.4.1). However, retaining these descriptors or residuals places high demands on memory which makes it an unsuitable choice for large collections.

Although the tentative correspondences generation algorithm might seem to be intricate, there is an efficient method. All correspondences can be generated with a linear time complexity for sorted feature lists ($\mathcal{O}(n \log n)$ for unsorted), with a slight overhead for the version with the limited discarding, because Cartesian products for all visual words need to be generated. In the worst case, all features in a pair of images would belong to the same visual word, resulting in $\mathcal{O}(n^2)$ time complexity for the product. Nevertheless, this scenario has a low probability and it is intended to avoid this — it would mean the vocabulary is not very descriptive.

Collection images are processed offline, making it possible to have the sorted feature lists prepared. The main idea is to have the features sorted according to their visual word and then iterate through the two lists. If current pair of features has the same visual word, then tentative correspondences should be generated. Otherwise, move the belated list one step forward. The maximum number of tentative correspondences can be again restricted by a threshold, a hyper-parameter. A basic version of the algorithm (without the restrictions for maximum correspondences) is described in the Figure 3.6. Note that the output list of tentative correspondences is again sorted according to visual words. This known order can be used later to speed up an advanced hypothesis verification.

This algorithm significantly reduces the number of tentative correspondences from *any possible* ($\sim n^8$) to just a few hundreds. It leaves only a subset of promising correspondences, which is then used in the RANSAC algorithm. It can be seen as a form or adaptation of PROSAC described in the Section 3.2.3. All possible tentative correspondences are sorted using visual words, obviously unpromising ones are disregarded right away, leaving only primitives with a higher potential for a success.

3.3.3 Local geometry for object retrieval

Second way how to increase the inlier ratio is to decrease the minimum sample size for hypothesis generation. Previously mentioned projective transformation with eight DoF needs four correspondences to estimate the transformation parameters, while affine transformation with six DoF needs only three tentative correspondences. The resulting transformation may not be as precise, but is sufficient in many cases. Furthermore, it can serve only as an initial estimate for further re-estimation, as described in the Section 3.4.6.

The transformation type can be eased up further more. Each feature has at least its position and scale detected, so in the worst case scenario these pieces of information can be used to establish a restricted similarity with three DoF, as described in the Section 3.3.1 and showed in the Figure 3.5 (1). To estimate parameters of this transformation, only a single tentative correspondence (only one primitive) is needed. While the number of hypotheses exponentially decreases, the precision of such models is poor since it is unable to capture the underlying ground-truth transformation. Nevertheless, this approach will be used very often throughout the thesis.

There is yet another option. Instead of using less flexible transformation types, some assumptions can be introduced and each feature can be equipped with additional pieces of information. These tools could be used for better parameter estimation - high number of DoF is kept while less primitives are used for hypothesis generation.

Typically, photos are taken from a restricted range of canonical views. One of the axes in the image is often parallel with the ground. Modern cameras can also detect landscape mode (where the image is rotated by ninety degrees) and transform the image accordingly. This results in the vast majority of pictures having the x axis parallel with the ground and most importantly, with similar (upright) view orientation. This prior information can be used as an assumption [28].

Next, shape information in the feature image region can be exploited, even in a memory efficient way, as suggested in [39]. Local geometry around the feature can be represented by an ellipse. It is convenient to represent the ellipse by a lower triangular matrix $\mathbf{A}$, which is mapping points on a unit circle to points on the ellipse:

$$\mathbf{A} = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix}$$

This matrix $\mathbf{A}$ can be defined up to an unknown rotation. The position, ellipse shape, and the (unknown) rotation form an *affine frame*. As proposed in the original work [39], it is assumed the orientation can be ignored. This assumption, coupled with the selection of $\mathbf{A}$ that preserves the orientation of the vertical axis, can be understood as implying the existence of a *gravity vector* – a vector within an image that points downward in the direction of gravity and is preserved. This gravity vector can be leveraged in the initial feature description. For each feature, a local affine frame is computed and utilised to normalise a neighbourhood region surrounding the keypoint. Subsequently, a descriptor is generated based on this normalised neighbourhood.

Furthermore, it has been shown that the local affine frames can be discretised into a com-

pact representation [39]. The memory requirements are reduced without a significant loss in performance.

In other words, from a high level perspective, each feature can be represented by an affine frame. This affine frame specifies feature location $\mathbf{x} = [x, y]^\top$ and some other parameters, a, b, and c, describing the image structure in a local neighbourhood. These five parameters can be then used to estimate a restricted affine transformation with five DoF (see the Section 3.3.1 and the Figure 3.5 (3)) using just a single tentative point correspondence.

Even though this approach brings a useful insight into the local geometry of all features, it cannot be used in all cases. For example, HOW features (described in the Section 2.6) are detected using a Convolutional Neural Network (CNN). It is not possible (nor desired) to normalise the local feature patches according to a gravity vector and even the exact location is unknown. Therefore simple transformation types, as described in the beginning of this chapter, must be used.

3.3.4 Transformation estimation

First, a short review of symbols used in the RANSAC algorithm: there is a set $\mathcal{U}$ of data points $\mathbf{x}$ from a measurements space $\mathbb{X}$; a random subset $\mathcal{S}$ of size m for hypothesis generation; model parameters θ from a parameter space Θ; function $f(\mathcal{S}) \to \Theta$ to estimate a hypothesis parameters from a minimal sample; error function $\rho(\theta, \mathbf{x}) \to \mathbb{R}$; threshold Δ for inlier recognition; cost function $J(\theta, \mathcal{U}, \Delta)$.

The algorithm can be applied for a robust estimation of a transformation between two images. The measurements space $\mathbb{X}$ is made up of all possible pairs of feature correspondences in the two images. A tentative correspondence (an item from the measurements space $\mathbb{X}$) will be denoted as T such that $\mathbf{x}$ can be used for a feature location. Each feature consists of a position $\mathbf{x} = [x, y]^\top$, scale s, visual word w and possibly some other pieces of information. The amount of all possible correspondences is way too large, so just a subset $\mathcal{U}$ is picked based on the visual words, as described in the Section 3.3.2. A random subset of m tentative correspondences is selected for the subset $\mathcal{S}$. In most cases, the estimation task is defined in such a way $m = 1$ (either by providing additional pieces of information for each feature, by introducing assumptions or by using restricted transformation types, as described earlier in the Section 3.3.3). Furthermore, the random subset $\mathcal{S}$ is not random per se, but all tentative correspondences from the pre-defined set $\mathcal{U}$ are used sequentially. The parameter space Θ consists of all 9-tuples defining 3×3 matrices representing image transformations. Just a subset of all matrices is often considered given the transformation type being estimated (e.g., for a simple translation, only a 2-tuple is needed). The function $f(\mathcal{S}) \to \Theta$ then assigns a particular n-tuple θ given a tentative correspondence of two features. Error function $\rho(\theta, T) \to \mathbb{R}$ takes a transformation matrix given from θ and a tentative correspondence T. Since the correspondence T is a pair of features, each feature having a position $\mathbf{x} = [x, y]^\top$, the error function tries to transform the feature from the first image into the second image using the estimated matrix. Then, the distance of the transformed feature from the first image and the original feature in the second image is returned as the error value.

This error value is then compared to a threshold Δ to determine whether the correspondence should be considered as an inlier or not. In this context, Δ is measured in pixels. All tentative correspondences from the set $\mathcal{U}$ are evaluated. Finally, all inliers are simply counted and the count is used as the value of the cost function $J(\theta, \mathcal{U}, \Delta) \to \mathbb{R}$.

Throughout the thesis, two types of features are used, both are evaluated mainly over the $\mathcal{R}$Oxford dataset. First are HOW features [24] (described in the Section 2.6). ResNet50 [25] with the last block skipped was used as the backbone fully-convolutional network. It was initialised by pre-training on ImageNet [40]. During the training phase, vocabulary of size $64 * 1024 = 65\,536$ was estimated using the K-means algorithm. As a shortlist, ASMK retrieval system was used, as in the original work [24]. Only top 100 images are used for the spatial verification. Each feature $\mathbf{F}$ has an approximate location $\mathbf{x} = [x, y]^\top$, a scale s, a descriptor $\mathbf{d}$, a visual word w, and a quantised residual vector $\mathbf{r}$. The HOW features are the main features for the thesis. Since they are estimated using a CNN, the exact position is unknown. The location $\mathbf{x} = [x, y]^\top$ is only an approximation acquired by backtracking the network, knowing its structure. This may lead to inaccurate transformations being estimated.

Secondly, SIFT features [9] together with local affine frames [39] are used as well. In this case, the BoW approach (as described in the Section 2.5.1) was used to obtain a shortlist. The vocabulary has size of 1 000 000 and was estimated using the K-means algorithm as well. Again, top 100 images were re-ranked using the spatial verification. Each feature $\mathbf{F}$ has a position $\mathbf{x} = [x, y]^\top$, visual word w, and parameters a, b, and c corresponding to a local affine frame. These descriptors were used to verify the correctness of the base algorithm implementation since it has already been shown to bring improved results while spatial verification with HOW features has not been used yet. In order to move closer to the first case with HOW features, residuals and scales must be somehow acquired. Because neither the original descriptors nor the cluster centroids were available in the data provided for the purpose of development of this thesis, residuals are assumed to be zero vectors. As a consequence, tentative correspondences cannot be sorted according to a similarity of descriptors. Regarding scales, the local affine frame parameters are used to estimate the scale s. The affine frames form an ellipse, so features containing only a scale can be understood as if they represent a circle. As it was already described in the Section 3.3.3, the parameters a, b and c form a lower triangular 2×2 matrix, which can be directly used to estimate an affine image transformation:

$$\mathbf{A} = \begin{bmatrix} a & 0 & t_x \\ b & c & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

This matrix transforms points on a unit circle with the centre in $[0, 0]$ onto an ellipse with the centre in the keypoint $[t_x, t_y]$. The parameters a and c define the scale over the x and y axes, respectively, while b defines the vertical shear. The parameter b is ignored and only a and c are used to estimate the isotropic scale s. It is computed as a geometric mean over the two numbers, i.e., $s = \sqrt{a \cdot c}$. It can be interpreted as the square root of the determinant of the 2×2

sub-matrix (the local affine frame), $s = \sqrt{|\mathbf{A}_{2\times2}|} = \sqrt{a \cdot c - b \cdot 0} = \sqrt{a \cdot c}$.

Having features $\mathbf{F}$ with a location $\mathbf{x}$, a scale s, a visual word w, and residual $\mathbf{r}$ (i.e., $\mathbf{F} = (\mathbf{x}, s, w, \mathbf{r})$), the RANSAC algorithm can be described in detail.

First, the measurement space $\mathbb{X}$ is a space of all tentative correspondences between features. Features, which are detected in the pair of images and assigned to visual words, are used to generate a set of tentative correspondences, $\mathcal{U}$. Maximum number of all tentative correspondences was set to 1 500, while maximum number of tentative correspondences generated from a single visual word was set to 15. In the case of HOW features, residuals are utilised to sort the features and pick only the top N in the event of encountering either of the thresholds. Since a binary quantisation method was used for their compression, Hamming distance is used to estimate the similarity between two residuals. As it was described earlier, Hamming distance is the number of positions at which the corresponding symbols are different. The Hamming distance is then normalised into the interval $[-1, 1]$ in such a way that -1 stays for completely different and 1 for the same (i.e., the Hamming distance equal to 0).

The tentative correspondences are then iterated over, each one of them serving as a subset $\mathcal{S}$ of size $m = 1$ and generating a hypothesis. The estimated transformation type is chosen to be the restricted similarity sub-type, i.e., a combination of a translation and an isotropic scale, as in the Figure 3.5 (1). The parameters space Θ consists of 3-tuples defining the translation and scale between the images. Every hypothesis θ (a set of parameters) is estimated from a single pair of (tentatively) corresponding features $\mathbf{F}_1 = (\mathbf{x}_1, s_1, w_1, \mathbf{r}_1)$, $\mathbf{F}_2 = (\mathbf{x}_2, s_2, w_2, \mathbf{r}_2)$. In particular, only the locations $\mathbf{x}_1 = [x_1, y_1]^\top$, $\mathbf{x}_2 = [x_2, y_2]^\top$ and scales s_1, s_2 are used. The restricted similarity matrix can be decomposed into two translation and two scale matrices for an easier interpretation:

$$
\begin{aligned}
\mathbf{M} &= \mathbf{T}_2\mathbf{S}_2\mathbf{S}_1^{-1}\mathbf{T}_1^{-1} \\
&= \begin{bmatrix} 1 & 0 & x_2 \\ 0 & 1 & y_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_2 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{s_1} & 0 & 0 \\ 0 & \frac{1}{s_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{s_2}{s_1} & 0 & -x_1\frac{s_2}{s_1} + x_2 \\ 0 & \frac{s_2}{s_1} & -y_1\frac{s_2}{s_1} + y_2 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.2}
$$

The translation matrices $\mathbf{T}_1$ and $\mathbf{T}_2$ translate the coordinate origin into the keypoint locations $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. Similarly, matrices $\mathbf{S}_1$, $\mathbf{S}_2$ transform the normalised scale into the scale of a particular image. Given the tentative correspondence features $\mathbf{F}_1$ and $\mathbf{F}_2$, the hypothesis is generated using the function $f(\mathcal{S}) = f(\{(\mathbf{F}_1, \mathbf{F}_2)\}) \rightarrow (\frac{s_2}{s_1}, -x_1\frac{s_2}{s_1} + x_2, -y_1\frac{s_2}{s_1} + y_2)$. The transformation matrix is then generated simply by substituting the values into the matrix $\mathbf{M}$ in the Equation (3.2), which transforms points from the first image into the second image.

Each hypothesis should then proceed to the verification step. Given a particular transformation matrix $\mathbf{M}_\theta$, all tentative correspondences from the set $\mathcal{U}$ are evaluated using an error

		Easy	Medium	Hard
SIFT	BoW	66.26	53.53	31.64
	Basic SV	69.74	55.92	33.49
	Optimal	79.41	61.68	40.40
HOW	ASMK	94.75	78.19	55.51
	Basic SV	85.80	71.83	45.93
	Optimal	97.80	83.97	67.43

Table 3.3: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - BoW and ASMK baselines, basic SV and optimal results.

function. In this case, the error function transforms keypoints of all features in the first image into the second image using the transformation matrix $\mathbf{M}_\theta$. Then, the distance of this transformed keypoint and the keypoint belonging to the corresponding feature in the second image is returned as the error value.

$$\rho(\theta, T) = \rho(\theta, (\mathbf{F}_1, \mathbf{F}_2)) = |\mathbf{M}_\theta \underline{\mathbf{x}}_1 - \underline{\mathbf{x}}_2|_2$$

The error value is then compared to the threshold Δ. During experiments, this hyperparameter was set to $\Delta = 62$. If the error is lower or equal, then the correspondence is considered to be an inlier. The count of all inliers determines the final value of the cost function for a given hypothesis:

$$J(\theta, \mathcal{U}, \Delta) = \big|\{T \in \mathcal{U} \mid \rho(\theta, T) \leq \Delta\}\big|$$

After evaluating all hypotheses (one hypothesis per tentative correspondence), the one with the highest cost function value is returned.

The algorithm described above represents the very basic version, leaving ample room for improvement. Several enhancements, extensions, and adaptations will be introduced and described in the next section. But already this algorithm gives meaningful results if the RANSAC assumptions are met.

This algorithm was evaluated on the SIFT and HOW features. The results are in the Table 3.3. Each column represents one evaluation protocol, as described in the Section 2.7.1. For SIFT features, the BoW approach was used to obtain the shortlist. For HOW features, ASMK approach was used instead. In both cases, top 100 images were proceeded to the spatial verification. The results for the SIFT features are improved while the results for HOW features are significantly worse. The optimal results were computed as well - how would the mAP look like if the first 100 images for each query were sorted perfectly, i.e., the relevant in the beginning and irrelevant only after.

An example of how an estimated transformation can look like is shown in the Figure 3.7. The transformation of the query Bounding Box together with inlier correspondences is depicted in red colour.

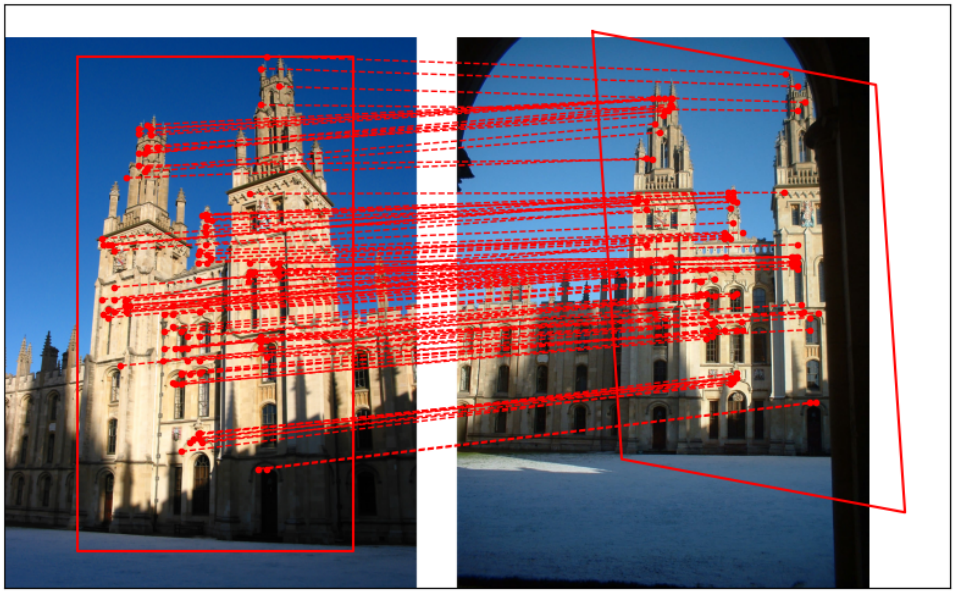


Figure 3.7: An example of an estimated affine transformation.

3.4 Possible adaptations

The basic spatial verification algorithm has ample room for improvement. Weights of tentative correspondences can be utilised, different cost functions may be tried or local optimisation might be added. Not only these possible enhancements will be described in this section. The goal is to improve the basic version of the algorithm in such a way that the results are better both in increased mAP and in more precise transformations (e.g., more inliers for TP images).

Some of the approaches will be described in the following sections. In each case, the base algorithm, to which are applied described changes, is the one just described in the Section 3.3.4. It will be referred to as the *Basic SV*.

3.4.1 Weighted correspondences

The first method utilises weights of the tentative correspondences — or, more precisely, the weighted inlier score in the cost function [39].

In the original formulation, all features belonging to the same visual word are considered to be completely equal. But a visual word can be understood as a part of the descriptor space (it is a Voronoi cell in a Voronoi tessellation in case of the K-means algorithm). Its size is unknown and depends on the training data and method. The original descriptor space is often infinite, meaning that at least some visual words cover an unbounded partition of the space. This is just one option which leads to the possibility of having two very different features with distant descriptors, yet assigned to the same visual word. If a tentative correspondence generated from these features is considered to be an inlier given a hypothesis, it is most likely a mistake. Therefore, it might be beneficial to assign a lower weight to such correspondences. On the contrary, if two features have exactly the same descriptors, the weight should be high.

When the original descriptors or residuals are known, the similarity of corresponding features

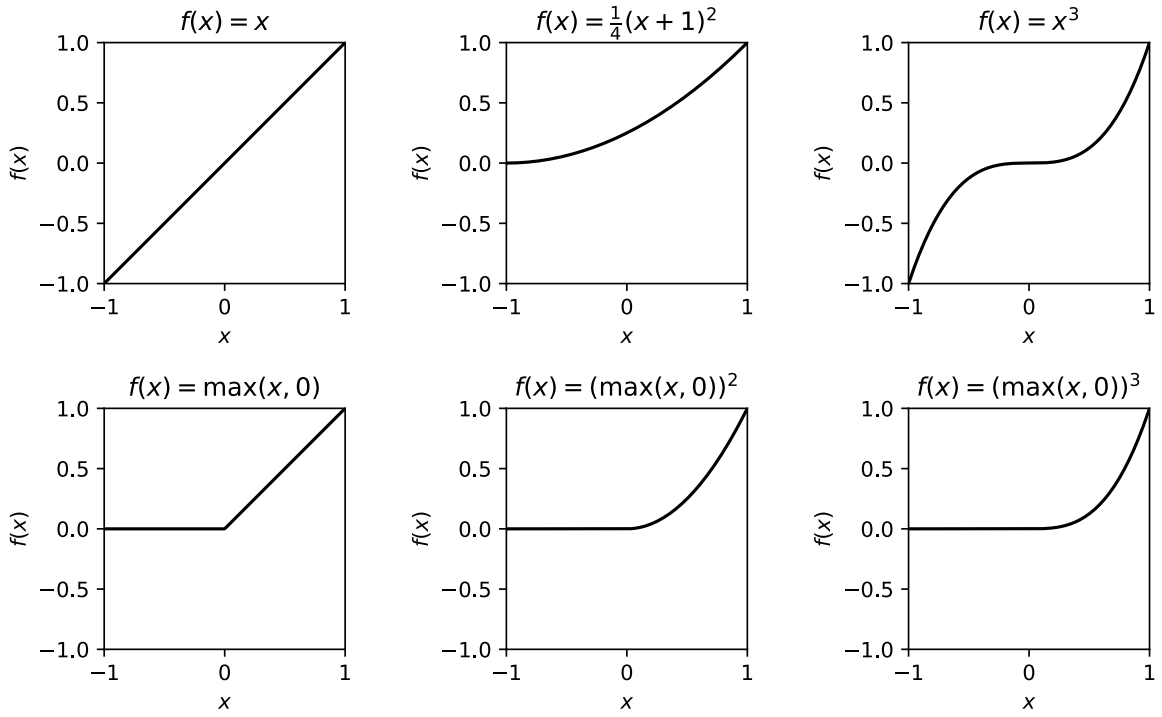


Figure 3.8: Different weight adjustment functions.

can be computed and used for the weights assignment. But this almost never happens for large-scale image databases, because storing entire vectors for each feature is memory demanding. Therefore, as it was already described, quantised residuals are used instead as a trade-off. They are used already in the tentative correspondences generation, as explained in the Section 3.3.2 and the Section 3.3.4. Let it be reminded that the similarity falls within the range from -1 to 1.

These similarities (generated from the corresponding quantised residuals) are returned together with the tentative correspondences. Although the similarity relates to the pair of corresponding features, it is being referred to as a *correspondence similarity*. This similarity coefficient can be then used as a weight in the cost function for a particular element. Due to the range of values, weights are often passed through one more function, adjusting their final influence on the score. The final cost function is as follows:

$$J(\theta, \mathcal{U}, \Delta) = \sum_{T \in \mathcal{U}} w(s(T)) [\![\rho(\theta, T) \leq \Delta]\!],$$

where $s(T)$ is the similarity corresponding to the tentative correspondence T, w is the weight adjustment function and $[\![\cdot]\!]$ is the Iverson bracket (it outputs one for true and zero for false statements).

Different weight adjustment functions were tried — linear, square, and cubic functions. They were also combined with clipping the value at a minimum of zero. All the functions are visualised in the Figure 3.8. In addition, exponential function $f(x) = e^x$ was used too.

Note the shift and the multiplicative constant in the square weight adjustment function

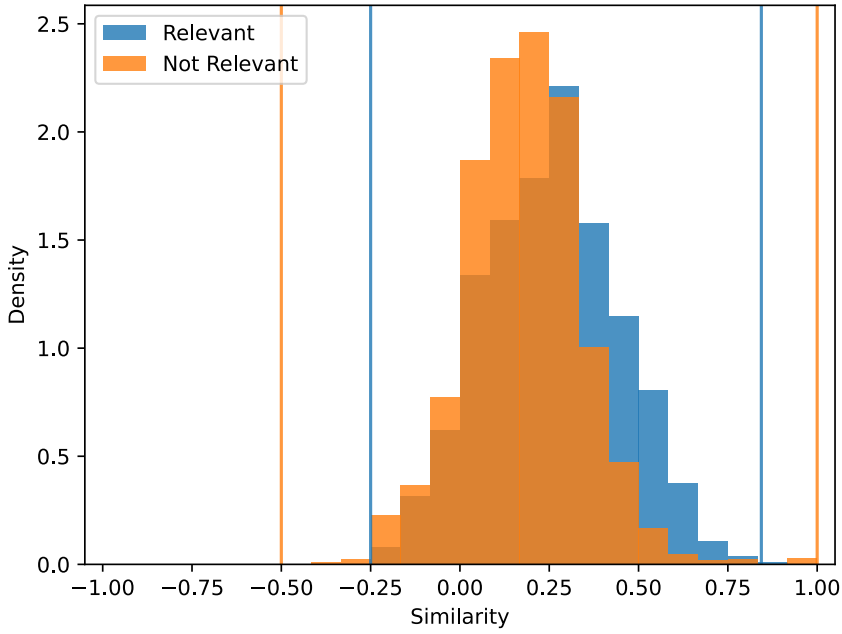


Figure 3.9: Histogram of similarities of correspondences, which generated optimal hypotheses.

($f(x) = \frac{1}{4}(x+1)^2$). The reason for the shift is for the function to be increasing on the interval $[-1, 1]$. The multiplicative constant then keeps the range of values in the original interval. It does not change the order, which is the decisive factor in the cost function. Hence, the results are the same as if there was no (or any other positive) coefficient.

Clipping values at a minimum of zero results in ignoring the tentative correspondences with a negative similarity in the cost function. But these correspondences are still used for hypothesis generation, which is an important factor. The histogram in the Figure 3.9 shows that tentative correspondences with a negative similarity can also end up generating the optimal hypothesis and therefore are useful. Weighted spatial verification with the clipped linear function ($f(x) = max(x, 0)$) was computed over all queries in the $\mathcal{R}$Oxford dataset on a shortlist of size 100 images. For each image pair, an optimal hypothesis was estimated. Then, the original tentative correspondence was taken and its similarity was looked up. The histogram was computed from these similarity values, separated by the relevance of the image pair. It is representing which similarity values do tentative correspondences generating optimal hypotheses have. It can be seen that optimal hypotheses between relevant images tend to be generated from correspondences with a higher similarity and also that 9.75% of them were generated from a correspondence with a negative similarity. In case of irrelevant image pairs, 13.80% of optimal hypotheses were generated from a correspondence with a negative similarity. The vertical lines in the chart represent the minimum and maximum value in the corresponding list of similarities.

Because the dataset with SIFT features does not contain the residuals, this adaptation was evaluated only on HOW features, which have all the necessary information. Weighting in general seems to improve the results. All weight adjustment functions achieved a higher mAP than the basic spatial verification. Best results (in terms of the highest mAP) were achieved by the linear

	Easy	**Medium**	**Hard**
Basic SV	85.80	71.83	45.93
$f(x) = x$	88.35	73.93	49.52
$f(x) = \frac{1}{4}(x+1)^2$	87.74	73.16	48.50
$f(x) = x^3$	87.07	73.44	49.66
$f(x) = \max(x, 0)$	88.21	73.85	49.40
$f(x) = (\max(x, 0))^2$	88.31	73.98	49.77
$f(x) = (\max(x, 0))^3$	87.04	73.45	49.68
$f(x) = e^x$	87.08	72.89	48.28

Table 3.4: mAP percentage for HOW features on $\mathcal{R}$Oxford - weighted score with different weight adjustments functions.

function, simple clip function, and the clip combined with the square function. All the results are shown in the Table 3.4.

3.4.2 Local optimisation

Another method applies the local optimisation, similarly to the LO-RANSAC described in the Section 3.2.1. The only difference is that the original formulation uses random subsets for hypotheses generation and optimises the model each time a better hypothesis is found, while in this case, a pre-defined list of primitives is sequentially evaluated and the local optimisation is performed only once, at the end, for the best model.

Assume a hypothesis with N tentative correspondences marked as inliers. Let $(\mathbf{x}_i, \mathbf{x}'_i)$ for $i = 1...N$ be the corresponding points $\mathbf{x}_i = [x_i, y_i]^\top$ in the first image and $\mathbf{x}'_i = [x'_i, y'_i]^\top$ in the second image. Then for an affine transformation $\mathbf{H}$ the following holds:

$$\mathbf{H}\mathbf{x}_i = \lambda \mathbf{x}'_i$$

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix}$$

Each correspondence produces a set of two linear equations:

$$\underbrace{\begin{bmatrix} x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \end{bmatrix}}_{\mathbf{A}_i} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ t_x \\ t_y \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} x_i' \\ y_i' \end{bmatrix}}_{\mathbf{b}_i}$$

The vector $\mathbf{h}$ represents the parameters of the affine transformation matrix $\mathbf{H}$. Denote $\mathbf{A}$, $\mathbf{b}$ as a set of all equations, i.e., stacked $\mathbf{A}_i$, $\mathbf{b}_i$ for all $i = 1...N$. Since there are six unknowns, at least three correspondences (which are not collinear) are required in order for the set of equations to be determined with a unique solution. In that case, the equations matrix $\mathbf{A}$ has shape 6×6 and an inversion exists. The solution is obtained as $\mathbf{h} = \mathbf{A}^{-1}\mathbf{b}$. If more than three correspondences are available, the solution can be obtained using a pseudo-inverse as:

$$\mathbf{h} = (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top \mathbf{b}$$

Computing the locally optimal affine transformation might change the arrangement of inliers, i.e., outliers may become inliers and vice versa. Therefore this optimal transformation is evaluated. If the value of the cost function increases, it means the inliers arrangement has changed and the local optimisation process can be executed again. This forms a recursive loop which ends as soon as the cost function stops increasing.

Since the number of correspondences is limited, the cost function has an upper bound too. As a consequence, the stopping criterion is always reached. If the inliers arrangement changes were observed instead of the cost function, there could be two hypothesis alternating infinitely, including and excluding two correspondences over and over again. A change in the cost function implies a change in the inliers arrangement, but this implication does not hold in the opposite direction.

The local optimisation was evaluated on both SIFT and HOW features, results are in the Table 3.5. In the case of HOW features, the mAP has increased in comparison to the basic spatial verification algorithm. For SIFT features, the performance has surprisingly slightly dropped. It can happen due to an issue described in the Section 3.4.4 which causes mistaken correspondences to influence the optimal solution.

3.4.3 Local optimisation on top N models

The local optimisation does not have to be used only for the best hypothesis. It can happen that another hypothesis converges to a better solution. That is the idea which leads to a slight enhancement of the previously mentioned LO version.

Simply perform the local optimisation on top N models, not only on the single one. Model with the best cost function value is then chosen.

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	SV with LO	69.47	55.63	32.95
HOW	Basic SV	85.80	71.83	45.93
	SV with LO	86.81	71.55	44.01

Table 3.5: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - basic SV and SV with LO.

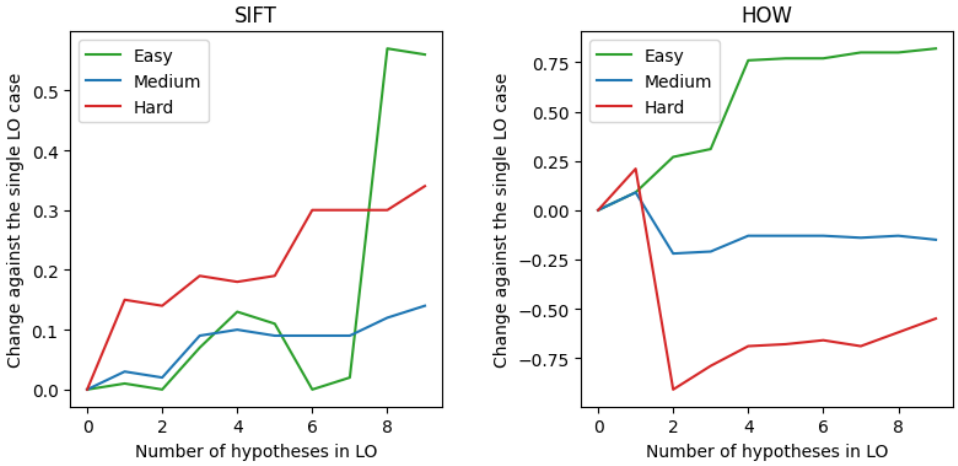


Figure 3.10: Change in mAP when multiple hypotheses are passed into the LO.

Results are presented in the Figure 3.10. The x axis shows the number of hypotheses passed into the local optimisation. The y axis shows how much does the mAP change in comparison to the case when only a single hypothesis was optimised.

3.4.4 Enhanced verification

Currently, the inliers are determined only based on a threshold of a distance. This approach has one large flaw — a single feature in an image can be considered as an inlier multiple times [39].

As described in the Section 3.3.2, tentative correspondences of features belonging to a visual word are generated using a Cartesian product. In other words, multiple features in the first image can correspond to a single feature in the second image, and vice versa. If some of the features in the first image are located nearby each other, all of them can fall under the threshold Δ after being transformed by a hypothesis. This results in a single feature in the second image generating more than one inlier. This situation is depicted in the Figure 3.11. There are four features in the first image, all of them corresponding to a single feature in the second image. All features are depicted in black. The arrow represents a hypothesis which transforms the features from the first image into the second one. The transformed features have grey colour. The threshold Δ around the feature in the second image is shown as a black circle. Since all four transformed features fall under the threshold, all four tentative correspondences are considered to be inliers.

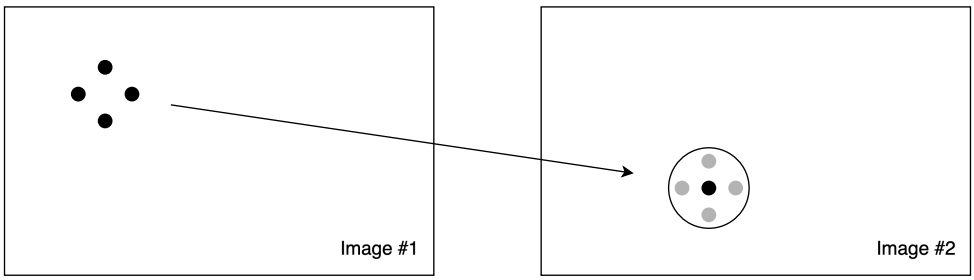


Figure 3.11: Example of multiple features corresponding to a single feature.

This case does not happen only rarely. It often occurs in real pictures when a repetitive pattern is present, for example: brick wall, tiled floor, foliage in a forest, textile designs, urban skylines, ripples in water, roof tiles, etc.

In reality, it does not make any sense for a single feature in an image to correspond to multiple features in another image. Inlier features should be in a One-To-One relationship. This constraint can be enforced in the inlier determination process.

The inlier determination process is formulated as follows: Let Δ be a threshold for an inlier recognition, $\rho(\theta, T) \to \mathbb{R}$ be an error function and $\mathcal{C} = \{(\mathbf{F}_i, \mathbf{F}_j) \mid \mathbf{F}_i \in I_1, \mathbf{F}_j \in I_2\}$ be a set of tentative correspondences T of features $\mathbf{F}_i$ in the first image I_1 and features $\mathbf{F}_j$ in the second image I_2. The task is to select a subset of $\mathcal{C}$, such that $\rho(\theta, T) \leq \Delta$ and each feature $\mathbf{F}$ is present in the subset at most once.

Ideally, the subset should be as large as possible and features should be paired in such a way that the sum of errors is minimum. The formulation above with these additional constraints forms a well known task — optimal matching in a bipartite graph. However, this task cannot be solved in a polynomial time. If the assignment is loosen a little bit by rounding the error such that only integers are used, then it can be solved using Ford Fulkerson algorithm for Max Flow problem [41]. The time complexity of the algorithm is $\mathcal{O}(C^2F)$, where C is the number of correspondences satisfying the thresholded error condition and F is the number of involved features in both images. Nevertheless, it is unbearable for each hypothesis to run a verification function with a time complexity higher than $\mathcal{O}(n)$. Hence, a greedy algorithm is used instead.

Greedy algorithm selects a feature in one image, takes a correspondence with the lowest error and inserts this correspondence into the final subset (if the error is lower than the threshold Δ). Corresponding feature in the second image is considered to be taken and is ignored further on. The pre-defined order of tentative correspondences generated by the algorithm described earlier in the Section 3.3.2 and the Figure 3.6 can be utilised in this task. The list of correspondences is sorted by the features from the first image, meaning that all tentative correspondences with a particular feature are consecutive. This helps to find a usable correspondence in an efficient manner by a sequential iteration. Features in the second image must be marked as taken. It can be done either by using an array of boolean values (flags) or a set. The former approach has look-up and write operations with a constant time complexity, but is memory consuming for large vocabularies (an array of the vocabulary size must be initialised while just a few elements are used). The latter approach is more complex, but sets are usually implemented using hash-tables

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	Enhanced verification	73.18	58.1	35.97
HOW	Basic SV	85.80	71.83	45.93
	Enhanced verification	86.88	72.59	47.89

Table 3.6: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - basic SV and enhanced verification.

with a constant time complexity for look-up and write operations as well.

This greedy algorithm runs in a linear time, but often does not return optimal results. For example, let **A**, **B** be features in one image and **1**, **2** features in the second image. All of them belong to a same visual word, raising a set of four tentative correspondences: **A1**, **A2**, **B1** and **B2**. Let the errors be 1, 2, 2 and 10, respectively. Then, the optimal solution is to have **A2**, **B1** with the errors $2+2=4$, but the greedy algorithm outputs **A1**, **B2** with the errors $1+10=11$.

Even though the greedy algorithm has no guarantee of how far the optimal solution is (in other words, it is not known how bad the greedy solution is), it gives sufficient results in practise. The mAP increases for both SIFT and HOW features, as shown in the Table 3.6.

3.4.5 Multiple assignment

Some relevant image pairs have just a few visual words in common. When descriptors are assigned to visual words, only a single visual word is chosen, based on the nearest cluster centre. The distance to the second nearest cluster, or even third, is not taken into consideration at all. But if these distances are similar to the distance to the nearest centre, then the features may be similar to the features belonging to the neighbouring visual words too. Unfortunately, this information is completely lost in the quantisation (by clustering) and potential correspondences of features are disregarded.

In case of spatial verification, this issue is even more noticeable for large vocabularies. Visual words in an image are very sparse, generating just a small amount of tentative correspondences. As a side effect, it creates a low upper bound on the maximum score.

Multiple assignment aims to partially solve this issue by assigning multiple visual words to each feature [39]. A common technique is to assign a pre-defined number of the nearest clusters to each descriptor. Similarly, the assignment could be based on the distances.

When each feature in a database image is assigned to multiple visual words, the memory complexity linearly increases. This may not be desired for large collections. As a compromise, the multiple assignment procedure might be performed only on query images. Each query is used in the spatial verification multiple times. It makes use of the approach while the memory load is increased only for a single image. Furthermore, in real retrieval systems, the query image is processed (detection, description, clusters assignment, computation) on the fly, without the necessity to store all the metadata on a hard disk.

		Easy	Medium	Hard
HOW	Basic SV	85.80	71.83	45.93
	Multiple assignment	87.28	72.30	46.22

Table 3.7: mAP percentage for HOW features on $\mathcal{R}$Oxford - basic SV and multiple assignment approach.

This approach was evaluated on the HOW features. Original descriptors of the SIFT features were not present in the dataset used for the development of this thesis. Four visual words are assigned to each feature in query images and only one visual word per feature in database images. Multiple assignment was used only for spatial verification, not for the original shortlist retrieval. The results are shown in the Table 3.7.

Note that multiple assignment can have a side effect while being performed on both query and database images, which is not really obvious. If two features with very close descriptors are being assigned to multiple clusters, they can be assigned to the same set of clusters. As a consequence, it behaves as if multiple features were matching while it is just a single one. This is even more notable in the spatial verification. One pair of features generates multiple tentative correspondences. All of them share the same parameters in terms of locations and scales, therefore they generate the very same hypothesis. Because the descriptors are close to each other, the correspondences might have a higher similarity. If the number of overall tentative correspondences is high, then they are sorted according to the similarity (as described in the Section 3.3.2) and only a portion of them is chosen. Since all of the correspondences generated from a single pair of features with close descriptors may have a high similarity, they are more likely to survive, eliminating other unique hypotheses.

Furthermore, either all of them or none of them can be inliers for a hypothesis. This can be interpreted as if a single tentative correspondence of two features had a several times higher weight than the other correspondences.

3.4.6 Homography estimation

Restricted similarity models are used to estimate a naive transformation out of a single tentative correspondence. This transformation is often good enough to find a set of inliers, which can be then used to estimate a better, more precise and more flexible, transformation. This approach was already described in the Section 3.4.2 about local optimisation, where an affine transformation was estimated using inliers of a naive hypothesis. In the same way, another transformation type can be used. The most flexible type is a homography.

A homography is a mapping between two planar projections of an image. In other words, homographies are image transformations that describe the relative motion between two images, when the camera (or the observed object) moves. In general, it maps four points onto any arbitrary four points. A homography matrix is defined up to scale, leaving eight DoF.

At least four corresponding points must be known in order to estimate the matrix parameters.

There are two ways of how to do so: by using a RANSAC algorithm again or using the least squares method.

In case of the RANSAC, inliers are used as primitives (assuming the naive hypothesis had more than four inliers). They are more likely to be correct, which increases the inlier ratio ε and decreases the number of iterations needed. Four correspondences are randomly selected and the homography is estimated out of them. The algorithm follows the standard formulation described in the Section 3.1. Nevertheless, this approach is not really suitable for the HOW features since the locations of the features are not known exactly. Small errors in the feature location (caused by the approximation) can lead to large errors in the transformation estimation.

Another option is to use all inliers to estimate the homography using the least squares method, DLT algorithm [31] in particular. The derivation is similar to the derivation of the affine matrix parameters earlier in the Section 3.4.2, including the iterative step. The homography matrices are recomputed as long as the score increases (hence the inliers arrangement changes).

Alternatively, yet another method can be used. A homography can be estimated from correspondences of local elliptical features [42]. This approach is based on the fact that a first order Taylor expansion $\overline{\mathbf{A}}(\mathbf{H}, \mathbf{x})$ of a homography $\mathbf{H}$ at a point $\mathbf{x}$ is an affine transformation [43]. Furthermore, affine transformations can be generated from local features using affine frames [39], as explained in the Section 3.3.3. The knowledge of only two affine transformations that locally approximate the homography at different points provides sufficient linear constraints to estimate the homography matrix $\mathbf{H}$. Unfortunately, this approach does not use the full potential in the case of this thesis since only scale is utilised — ellipses (affine frames) are disregarded in favour of simple circles.

Real example is depicted in the left part of the Figure 3.12. First, an affine transformation (red transformation) was estimated using local optimisation (Section 3.4.2) and enhanced verification (Section 3.4.4). Inlier correspondences (also depicted in red) were then used for further computations. All three approaches were tested and compared — least squares solution (blue), inner RANSAC for homography estimation (green), and a homography estimated from correspondences of elliptical features (yellow). All three resulting transformations are very inaccurate. The flexibility of the permissive transformation type causes the transformation to over-fit the points.

This appears to be a frequent problem. Naive hypotheses tent to have inliers mostly around the original correspondence (the correspondence which generated the hypothesis). It can be caused by the fact that features which are more distant from the original correspondence have a higher error value due to the inaccurate hypothesis and, as a consequence, they may not fall under the threshold.

There was an attempt to somehow regularise the inaccurate corners of the Bounding Box (BBox). Top left and bottom right corners were transformed and the two correspondences of the original and transformed points were added. The motivation was that it regularises the transformation such that the corners would not be so off, while the influence on the final model would be low due to being just two correspondences out of many. Nonetheless, as it can be seen in

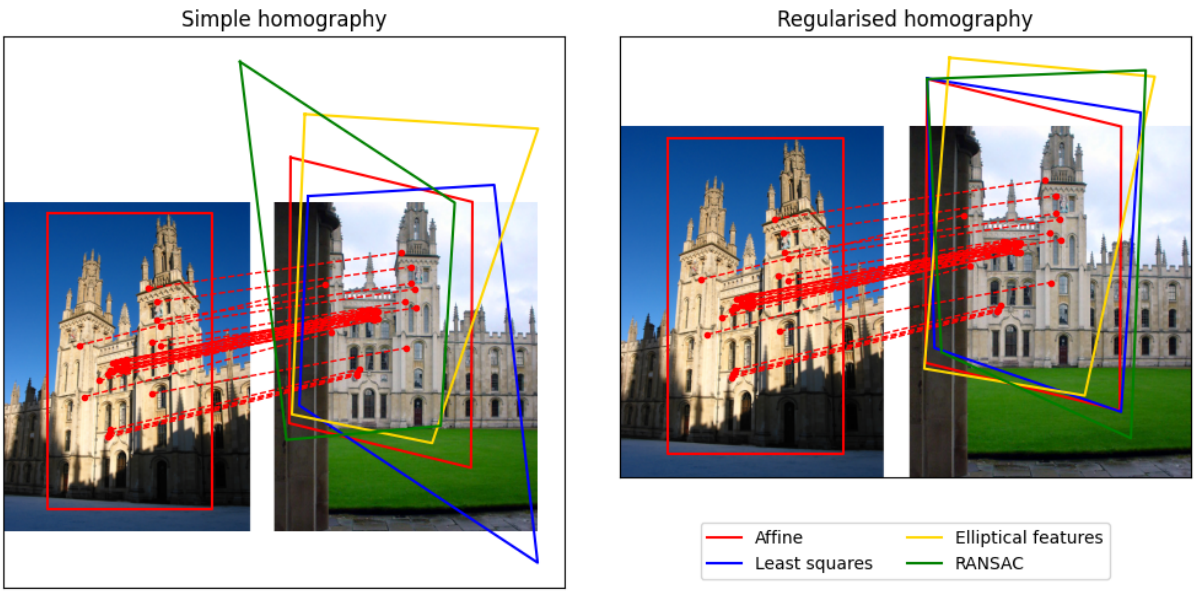


Figure 3.12: Example of different homography estimation approaches.

the right part of the Figure 3.12, the flexibility of the homography is perceptible and the artificial correspondences moved the homography significantly closer to the affine transformation.

Homographies did not seem to have the expected effect and therefore were not evaluated on the whole $\mathcal{R}$Oxford dataset.

3.4.7 Determinant and scale verification

So far, only the feature location $\mathbf{x} = [x, y]^\top$ was used in the verification process of a hypothesis. The scale of the feature was not taken into consideration.

But even the scale can be used for a verification of an affine transformation [39]. In theory, if a feature was detected on a scale $s_1 = 2.0$ and a corresponding feature on a scale $s_2 = 0.5$, then it is expected to find a transformation, which reduces the size of the object (reduces the scale) $2.0/0.5 = 4$ times. On the contrary, if the hypothesis increases the object size (increases the scale), then it is likely that the features are not true correspondences.

In this context, the scale change is actually the change in a two dimensional area — how the area of a unit square changes after being transformed. In case of an affine transformation $\mathbf{A}$, this area change is specified by the determinant of the matrix. The shear and scales (which influences the change of an area) are determined by the 2×2 sub-matrix, last column contains just a translation (which does not change the area in any way) and last row does not contain any values in case of an affine transformation.

$$|\mathbf{A}| = \begin{vmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

In the enhanced verification process, it is checked that the ratio of the scale of a transformed feature and a corresponding feature is between some pre-defined bounds. These bounds are

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	Scale verification	69.33	55.85	33.52
HOW	Basic SV	85.80	71.83	45.93
	Scale verification	86.08	72.14	47.35

Table 3.8: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - basic SV and scale verification.

passed as a hyper-parameter B.

$$\frac{1}{B} \leq \frac{s_1 \cdot |\mathbf{A}|}{s_2} \leq B$$

Given a hypothesis with a matrix $\mathbf{A}$, tentative correspondences, which are not satisfying this inequality, cannot be labelled as inliers.

This hypothesis verification toughening was evaluated with the threshold $B = 5$. The results are in the Table 3.8. The results for SIFT features are worse than the Basic SV approach. The HOW features received a slight improvement in the mAP.

This approach is applicable only for affine transformations, including the sub-types, i.e., it is not applicable for projective transforms, namely for homographies described in the previous section. The reason is that the determinant of the 2×2 sub-matrix does not specify the scale change anymore and neither does the determinant of the whole 3×3 matrix.

However, a homography $\mathbf{H}$ can be approximated at a point $\mathbf{x}$ by a first order Taylor expansion $\overline{\mathbf{A}}(\mathbf{H}, \mathbf{x})$ [43]. This approximation is an affine transformation and therefore can specify the scale change in the neighbourhood of a point $\mathbf{x}$. A possible workaround is to compute an affine approximation of an estimated homography at each inlier correspondence and use its determinant to specify the scale change for that particular correspondence.

3.4.8 Eigenvalues verification

Similarly to checking the scales of the features, even the estimated transformation can be further analysed and examined. Naive transformations generated from tentative correspondences are straightforward, there is nothing to look into. But affine transformations received from the local optimisation may have some deficiencies. In this section, an algorithm with local optimisation on top 10 naive hypotheses (as described in the Section 3.4.3) is used as a baseline.

The local optimisation is an iterative process which progressively adapts the transformation to the current set of inliers. The final output estimation can be very different from the input one. In fact, it happens (mostly for some irrelevant images) that the locally optimal transformation tries to fit itself on a line of inliers. In other words, there are features approximately on a line in one image, features approximately on a line in the second image and the features are corresponding with each other. This can happen for example for rooftops (roofing tiles neighbouring with the sky), lawns (grass field neighbouring with a path), etc. It also happens when an object is zoomed out and the features appear to be in a thin strip, or even when just two small clusters

Figure 3.13: An example of a local optimisation which just squeezes points onto a line.

of points correspond.

In those cases, the local optimisation may tend to find a transformation which maps the first line onto the second line (or similarly the first small cluster onto the second small cluster). Since all the points do not lie exactly on a line, it might even try to push the points closer to the line. At the end, the resulting transformation is distorted and deformed.

An example is shown in the Figure 3.13. All the corresponding features (depicted in red) from the naive hypothesis are on the border of the roof and the sky, therefore it makes sense they are assigned to the same visual words. Furthermore, they form just a small cluster in the right picture and features in the left picture are positioned inside a thin horizontal stripe. The naive hypothesis (depicted also in red) makes the original BBox significantly smaller, making it possible for the correspondences to fall under the threshold (therefore become inliers). After five iterations in the local optimisation, the transformation gets deformed (depicted in yellow).

To prevent from having transformations deformed in this way, they can be analysed and ignored based on a criterion. The determinant of a matrix defines only the overall scale, but does not take into consideration the scale of individual axes. The scale coefficients (numbers on the diagonal) cannot be used because they do not take account of the shear. Therefore eigenvalues of the transformation matrix should be used.

Eigenvectors of a matrix are vectors which do not change their orientation under the transformation. They are just scaled. Corresponding eigenvalues define exactly this scale, i.e., how much are they scaled:

$$\mathbf{A}\mathbf{u} = \lambda\mathbf{u},$$

where $\mathbf{A}$ is a matrix, $\mathbf{u}$ is an eigenvector (often with unit length) and λ is a corresponding eigenvalue. In fact, the product of all eigenvalues equals to the determinant. Furthermore, the affine transformation matrix has one eigenvalue equal to one and two eigenvalues equal to the eigenvalues of the 2×2 sub-matrix. It means that the scale changes can be analysed by the two eigenvalues since they characterise the scale changes in particular directions.

The absolute value of an eigenvalue is not a reliable criterion to decide whether a transformation is reasonable (i.e., is not deformed). One image can be just zoomed in or out. Better

		Easy	Medium	Hard
SIFT	LO with 10 entries	70.03	55.77	33.29
	Eigenvalues check added	70.58	56.08	33.69
HOW	LO with 10 entries	87.63	71.40	43.46
	Eigenvalues check added	88.32	72.31	45.29

Table 3.9: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - LO with 10 entries and its adaptation with eigenvalues check.

approach is to analyse the ratio $\frac{\lambda_1}{\lambda_2}$ of the two eigenvalues λ_1, λ_2, i.e., if the scale in one direction is not significantly larger that in the other direction. Again, this ratio is compared with a threshold R (new hyper-parameter):

$$\frac{1}{R} \leq \frac{\lambda_1}{\lambda_2} \leq R$$

This criterion was evaluated with the hyper-parameter $R = 5$. The results are shown in the Table 3.9. There is a slight improvement, but affected image pairs still remain with quite a large score, because the initial naive hypothesis has usually already many inliers.

3.4.9 Bounding box cover scoring

Last tried modification of the basic spatial verification algorithm is a different cost function for determining the scores.

What happens very often is that an image pair has a large number of inliers, but majority of the features in the query image is located in a cluster, a small local region. It can be partially seen already in the Figure 3.12, where most of the features in the first image are in the centre while the top left part and the bottom part of the image are quite empty. On the contrary, the desired situation (for relevant images) is for the inlier features in the query image to cover majority of the BBox area.

Uniform spatial distribution of features can be regarded as more reliable. This BBox coverage can be then utilised as a weight [44]. In the case of spatial verification, the utilisation resides in the cost function.

All inlier features mark their neighbourhood. The cost function used for scoring is the marked area. The larger area, the higher score. If two inlier features are close to each other, the marked area does not increase very much since they share most of the neighbourhood. If two the features are distant, then they share no neighbourhood and both of them have significant influence on the final cost function value.

An example of this idea is depicted in the Figure 3.14. Correspondences are depicted in red and the green area behind each feature represents the marked area. The coverage of the marked area in the BBox (the score) is 41.76%.

The size of the marked area should be relative to the BBox size. Not all queries have the same dimensions, so absolute values would not treat all the queries in the same way — for example, one query might need only twenty features to cover the whole BBox while another

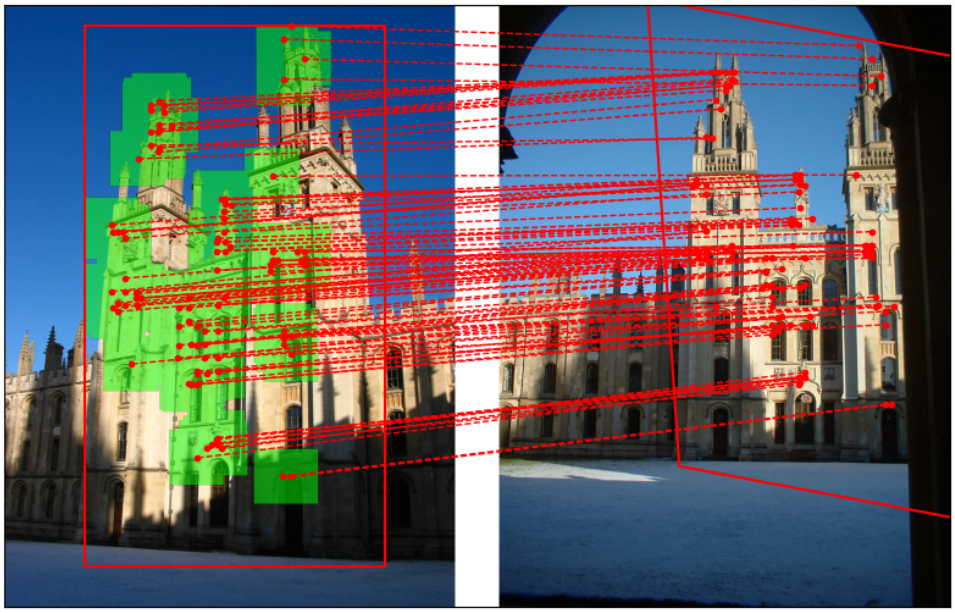


Figure 3.14: BBox coverage by inliers.

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	BBox Coverage Scoring	72.92	57.83	35.25
HOW	Basic SV	85.80	71.83	45.93
	BBox Coverage Scoring	83.68	67.78	40.11

Table 3.10: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - basic SV and BBox Coverage scoring.

query would need sixty.

The following marking strategy was used: given a BBox of width W and height H, each feature marks a square with its side length l. The side length l is determined by the mean side length of the BBox, divided by a hyper-parameter P. This results in the following formulae:

$$l = \left\lfloor \frac{1}{2P} \cdot (H + W) \right\rfloor$$

This approach was evaluated with the hyper-parameter $P = 22$ and the results are shown in the Table 3.10. Dataset images labelled as *hard* often contain query landmarks with difficult viewing conditions, such as an occlusion (see the Section 2.7.1). Therefore, such a significant drop in performance for the HOW features and *Hard* evaluation protocol is rather expected.

3.4.10 Baseline used for further development

At the end, multiple aforementioned enhancements can be applied at once. Nevertheless, one having a positive influence on the results does not mean it works in conjunction with another approach too.

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	Advanced SV	73.91	58.05	36.12
HOW	Basic SV	85.80	71.83	45.93
	Advanced SV	91.74	75.52	51.59

Table 3.11: mAP percentage for SIFT and HOW features on $\mathcal{R}$Oxford - basic SV and advanced SV.

Several combinations of the enhancement methods were tried and the final algorithm configuration used for further development was decided to be as follows: Local optimisation on top 10 models (Section 3.4.3); enhanced verification with the error threshold $\Delta = 62$ (Section 3.4.4); and eigenvalues verification with the ratio threshold $R = 5$ (Section 3.4.8). HOW features additionally utilise weighted correspondences with the weight adjustment function set to $f(x) = \max(x, 0)$ (Section 3.4.1). Spatial verification was performed on a shortlist of length 100 images.

This configuration will be referred to as the *advanced SV*. The results of the advanced SV algorithm in comparison to the basic SV are shown in the Table 3.11.

3.5 Spatial verification use cases

When a transformation between the images is established, it can be utilised. One potential use case is in a training of deep learning models for local-feature-based methods in image retrieval or processing. If the transformation between two images is known, a model can be taught in such a way it recognises corresponding parts. This might be done in an easier and more precise way using approaches like Structure from Motion (SfM) [45], where an object is photographed from many different positions with the relative camera pose known (i.e., the position and direction/rotation of the camera is known). However, in datasets like these, no difficult viewing conditions appear (such as different day time, weather, fog, season of the year, sketches, paintings, etc.). Also, these datasets are harder and more expensive to obtain, while the internet already contains plenty of images of all kinds. This approach of learning the image descriptor extractor was not tried in this thesis because it is a complex task which would require a lot of additional work beyond the time allocation.

Besides this theoretical use case, there are more practical (and already tried) ones. The results can be sorted, but not only using the inlier score (as it was done so far). For example, they can be sorted (or even filtered) based on the zoom. Only zoomed in or zoomed out images can be retrieved. This process can be performed repeatedly, going from an image with a very small object to another image with a detailed part of the same object. Another use case is to analyse the frequency with which is an object photographed in detail [46].

Spatial verification finds a corresponding plane in a pair of images. If the inliers or even whole features belonging to this plane are removed from the image and the spatial verification

algorithm is performed again, it is possible to find yet another, different, plane with a different transformation. This provides more information about the underlying geometry which can be further utilised. In this case, it is beneficial to decrease the inlier threshold and utilise the local optimisation in order to remove solely features from the found plane and not others. However, the primary dataset used in this thesis does not contain objects which would have several dominant planes in the field of interest and therefore this approach was not not further elaborated.

One more practical use case is *query expansion* [47], [48], sometimes abbreviated as QE. The original query can be enriched by new features and the image search can be processed again, using these extended data.

3.5.1 Query expansion

Two distinct images of the same object often contain many visual features, some of which belong to the same visual words and most of which do not. Feature detection and quantisation are noisy processes and can result in variation in the particular visual words that appear in different images of the same object, leading to missed results.

If an image pair (query and database) has a large enough score in the spatial verification, the database image can be considered to be *verified* in the sense that it is assumed it depicts the same object as the query image. The verified image includes additional information. First and foremost, it contains new features, which are not present in the query, including their location. These features can be transformed back to the query image and possibly merged with the original ones. This extended query has a potential to nicely capture the essence of the object, making it possible to retrieve many relevant images with a higher score than before and to revive previously missed results.

Assume a query image Q with a set of features $\mathcal{F}_Q$ inside of a BBox B and a collection of images $\mathcal{C}_{db}$ (database), where each image $C_i \in \mathcal{C}_{db}$ has its own set of features $\mathcal{F}_{C_i}$. Each feature $\mathbf{F}$ is a four-tuple of location $\mathbf{x}$, scale s, visual word w, and residual $\mathbf{r}$, $\mathbf{F} = (\mathbf{x}, s, w, \mathbf{r})$. Assume each image pair $Q \leftrightarrow C_i$ has an estimated transformation $\mathbf{A}_{C_i}$ from the query image Q into the collection image C_i and that the hypothesis corresponding to this transformation has a score φ_i.

Then a subset of collection images $\mathcal{C}_{db}$ with a high enough score (greater than a threshold Φ) can be assumed to be verified and features $\mathbf{F}_i$ from the verified images can be transformed to the query image Q, extending the query feature set $\mathcal{F}_Q$. Only features back-projected inside of the BBox B are considered. The feature properties (in this case only the location $\mathbf{x}$ and scale s) must be changed accordingly.

$$\mathcal{F}_Q^{(\mathrm{QE})} = \mathcal{F}_Q \cup \left\{ \left(\mathbf{A}_{C_i}^{-1}\mathbf{x_i}, \frac{s_i}{|\mathbf{A}_{C_i}|}, w_i, \mathbf{r}_i \right) \;\middle|\; (\mathbf{x}_i, s_i, w_i, \mathbf{r}_i) \in \mathcal{F}_{C_i}, C_i \in \mathcal{C}_{db}, \varphi_i \geq \Phi, \mathbf{A}_{C_i}^{-1}\mathbf{x_i} \in B \right\}$$

This naive approach generates way too many features, which may not be desired. The features are usually filtered. There are two ways of how to so: filter the images which are used for the query expansion or/and filter the features themselves.

In the formulation defined above, all verified collection images are used. Instead, only a subset can be chosen. However, it may not be the best idea to choose the top N verified images as they are expected to be the most similar to the query. Visual features retrieved out of them might be very similar to those already present in the query. Common approach is to take every n-th verified image instead. It is expected to reliably cover the range of object specific visual features — it adds new visual features as well as secures the original ones by heightening their overall weight.

The second way of how to generate less features is to filter the features themselves. Again, every n-th feature can be chosen, but more sophisticated approaches also exist. One of them is based on Term Frequency-Inverse Document Frequency (TF-IDF) (which was described in the Section 2.5.1). All the features are sorted according to the TF-IDF and only a portion is used. Similarly, strengths of features can be used as the sorting criterion, if they are known from the detection and description process.

There are also other, more advanced, approaches, such as transitive closure expansion, average query expansion, recursive average query expansion, or multiple image resolution expansion [47].

For the purpose of this thesis, only a basic query expansion approach was implemented and evaluated on the SIFT features. The verification threshold Φ was set to 15 (inliers). If at least three images were verified, then the images were further filtered and only every n-th was used, where $n = \max(1, \lfloor \frac{v}{10} \rfloor)$ and v is the number of verified images. If less than three images were verified with the threshold $\Phi = 15$, then the verification threshold Φ was decreased to 8 and all collection images satisfying this score constraint were used for the query expansion.

Regarding the features, all of them were included in the query for the second retrieval, where BoW with cosine similarity and TF-IDF weighting was used (see the Section 2.5.1). In case of a second spatial verification, only every n-th feature was utilised, where n was chosen in such a way that approximately 1 500 features were used in total, i.e., $n = \lfloor \frac{1}{1\ 500} \#\text{features} \rfloor$.

The results are shown in the Table 3.12. The query expansion brings significant improvements. Basic SV algorithm (as described in the Section 3.3.4) and advanced SV (as described in the Section 3.4.10) were used as baselines. On the top of them, query expansion was applied, as explained above. Even the values of the hyper-parameter Φ were kept unchanged, even though it might be a good idea to adjust them for each approach individually since the advanced SV is much more restrictive. Also, a second spatial verification was evaluated on the expanded queries. The effect of the second verification is, surprisingly, opposite for the basic and advanced SV.

Note that the optimal mAP from the Table 3.3 was exceeded by the query expansion. It is because this optimal result considers only sorting entries in the top 100 shortlist. Nevertheless, the expanded query performs the image search from scratch using the BoW, which results in a different response (possibly with more relevant database images present in the top 100).

Query expansion was also evaluated on the HOW features with ASMK retrieval engine. However, there was a significant drop in performance.

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	Basic SV + QE	75.36	64.16	41.72
	Basic SV + QE + Basic SV	80.30	67.43	44.24
	Advanced SV	73.91	58.05	36.12
	Advanced SV + QE	83.36	68.69	47.00
	Advanced SV + QE + Advanced SV	73.35	61.51	38.06

Table 3.12: mAP percentage for SIFT features on $\mathcal{R}$Oxford - basic SV, advanced SV and their combinations with QE.

		Easy	Medium	Hard
SIFT	Basic SV, circles	69.74	55.92	33.49
	Basic SV, ellipses	69.85	56.06	33.67
	Advanced SV, circles	73.91	58.05	36.12
	Advanced SV, ellipses	74.21	58.28	36.16

Table 3.13: mAP percentage for SIFT features on $\mathcal{R}$Oxford - circles versus ellipses with $\Delta = 62$.

3.6 Additional evaluations

In order to get to know the data better, additional experiments and evaluations were performed on the two available feature sets for the $\mathcal{R}$Oxford images.

SIFT features with local affine frames define an ellipse for each feature. The original formulation estimates affine transformations out of these ellipse correspondences [39] while in this thesis, parameters of ellipses helped to assess scales (hence, an ellipse becomes a circle) and a restricted similarity type was used instead of the affine. The question is, how much information is actually lost in this approximation.

Since restricted similarity transformations are significantly more rigid than affine transformations and the location of the HOW features is only approximated, the inlier threshold used throughout this thesis was decided to be $\Delta = 62$ (as written in the Section 3.4.10). This is quite a large and benevolent value. If this threshold is kept unchanged, the added value by preserving and using the whole affine frames is not really noticeable since the transformation flexibility and accuracy is lost in the error leniency. The mAP increases approximately by 0.17% for each evaluation protocol, as shown in the Table 3.13.

However, if the inlier threshold is decreased to as low as $\Delta = 25$, the difference becomes more apparent, as it can be seen in the Table 3.14. In case of the advanced SV without local optimisation, the difference in the mAP is as high as 1.24%. When the local optimisation is added, the difference decreases since the LO also outputs an affine transformation which evens up the advantage of ellipses.

Second experiment aims to examine the HOW features. In the ASMK retrieval system, they

		Easy	Medium	Hard
SIFT	Basic SV, circles	69.71	55.86	33.05
	Basic SV, ellipses	70.30	56.35	33.89
	Advanced SV, circles	72.81	58.04	35.90
	Advanced SV, ellipses	73.44	58.59	36.46
	Advanced SV without LO, circles	72.15	57.73	35.50
	Advanced SV without LO, ellipses	73.39	58.54	36.43

Table 3.14: mAP percentage for SIFT features on $\mathcal{R}$Oxford - circles versus ellipses with lower thresholds.

	Easy	Medium	Hard
Basic SV using SIFT features	69.74	55.92	33.49
Basic SV using HOW features	75.01	57.54	32.95
Advanced SV using SIFT features	73.91	58.05	36.12
Advanced SV using HOW features	76.84	59.24	35.57

Table 3.15: mAP percentage for HOW features evaluated on $\mathcal{R}$Oxford using a shortlist from SIFT plus BoW.

achieve state-of-the-art performance, which is ruined after applying the spatial verification (as it was shown before in the Table 3.3). The question is whether the HOW features are unusable for the spatial verification at all or whether the reason resides in the shortlist.

It was tried to take the shortlist given by the SIFT plus BoW approach and then apply the spatial verification using the HOW features. The results are shown in the Table 3.15. It can be seen that the spatial verification *does* work even for the HOW features. The results for the *Easy* evaluation protocol are significantly better (in comparison to the SV with SIFT features), while the performance slightly drops for the *Hard* evaluation protocol. Nevertheless, the HOW features deliver reasonable results and therefore are usable for the spatial verification.

Chapter 4

Using lines in spatial verification

So far, only keypoints in images were used as primitives for a transformation estimation. Single tentative correspondence of two keypoints was used to estimate a naive hypothesis, which was then further verified and possibly optimised.

In this chapter, the spatial verification is further extended. Lines and line segments are used as they also exhibit discriminative neighbourhood and high repeatability properties, just like local features, as explained in the Section 2.3 about feature detection. Lines are strong elements that tend to remain consistent even when there is a change in the visual appearance of the same object. It makes them potentially useful in the spatial verification process.

Their primary utilisation is in compensating measurement errors. As it was described in the Section 2.6, the location of the HOW features is only approximated based on an architecture of a fully convolutional neural network. The lines are expected to compensate for the error and uncertainty.

First, line segment detection methods are explored and described. Then, detected line segments are used to generate new naive hypotheses. They improve the basic hypotheses estimated from keypoint correspondences by creating correspondences of line segments in the neighbourhood of the keypoint pair. These line segment correspondences are then utilised to make the transformations more precise. Lastly, detected lines are used to estimate constrained homographies.

4.1 Line segment detection

Before line segments can be used, they have to be detected in all images. The ability to accurately identify and extract line segments from digital images plays a vital role in understanding the underlying geometric structure and spatial relationships within a scene.

There are several approaches how to do so. These methods can differ based on their underlying principles, computational complexity, hyper-parameters, sensitivity, and robustness to noise, gaps and breaks. Also, some methods are meant to detect entire lines instead of line segments. This might be undesirable in some cases as further processing is required to obtain segments only. As a consequence, each method gives more or less diverse results.

The subsequent sections will focus on specific algorithms for line segment detection which were considered in this thesis.

4.1.1 Progressive Probabilistic Hough Transform

The Hough Transform (HT) is a fundamental technique used in computer vision and image processing for extraction of geometric shapes, particularly in the presence of noise, distortion, or other irregularities. It was first introduced by Paul Hough in 1962 for automatically detecting simple shapes like lines in binary images, and later extended to detect more complex shapes like circles and ellipses. The basic idea behind the HT is to represent a particular shape (e.g., a line) in an image as a point in a parameter space, rather than directly in image space. By doing so, the detection of shapes becomes a problem of identifying peaks in this parameter space. These correspond to the parameters of the desired shapes.

First step is to identify pixels that likely belong to edges or boundaries of objects in the image. Hence, edge detection must be performed first. One of the popular approaches is the Canny edge detector [49].

The Canny edge detector analyses gradients in the image intensity. The higher gradient magnitude at a pixel, the higher potential it belongs to an edge. When gradient magnitude values are computed for each pixel in the image intensity, non-maximum suppression is applied in order to thin the edges and keep only the local maxima in the gradient direction. For each pixel, the gradient magnitude is compared with its neighbouring pixels in the gradient direction. If the pixel has the maximum gradient magnitude among its neighbours, it is retained; otherwise, it is suppressed (set to zero). This step helps in obtaining thin, well-defined edges by removing pixels that do not contribute significantly to the edge structure. Furthermore, the pixels are double thresholded. These two thresholds categorise pixels into three groups — *strong* edge, *weak* edge, *no* edge. Strong edge and no edge groups represent the final classification while weak edge group is further processed by a hysteresis. This very final step aims to connect weak edge pixels to strong edge pixels to form continuous edges. If a weak edge pixel is connected to a strong edge pixel (either directly or through a chain of weak edge pixels), it is classified as part of the edge.

Once edge pixels are detected, they are used to extract shapes, in this case, lines. Each line can be parametrised in multiple ways, one of which is parametrisation by two parameters, $\theta \in [0, \pi]$ and $r \in \mathbb{R}$, such that the following equation holds: $x \cos\theta + y \sin\theta - r = 0$

The parameter space is discretised (or quantised) into bins, each bin stands for a particular instance of a line with a pair of exact parameter values. The discretised parameter space is represented by an *accumulator array*. Each cell in this array corresponds to one bin. The accumulator is then used in a voting scheme.

Every edge pixel is suspected of belonging to a line. Therefore, for each edge pixel in the image, calculate and vote in the accumulator array for possible lines that pass through that pixel. This involves updating the accumulator array based on the possible r and θ values that could represent lines passing through each edge pixel.

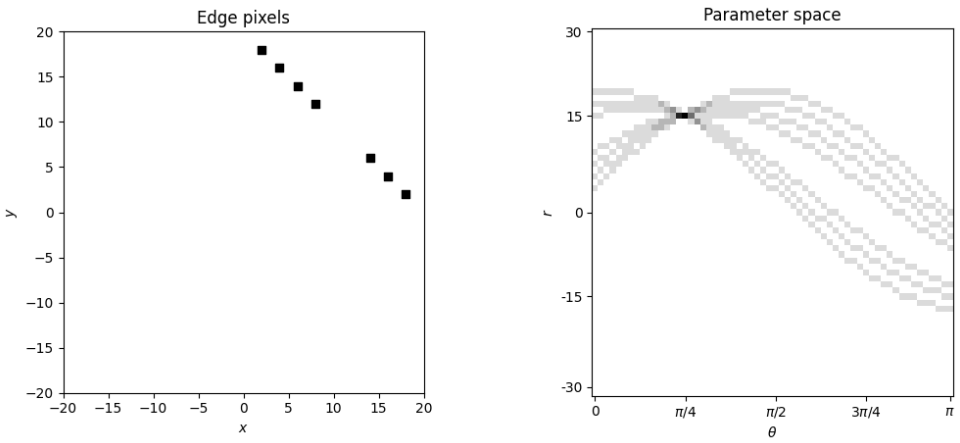


Figure 4.1: Example of a HT to find line parameters.

Finally, when the voting is done, peaks (local maxima) are identified in the accumulator array. Every peak corresponds to the parameters r and θ of the lines that are detected in the image. Nevertheless, if the number of votes is not high enough, it might be a false positive response or featureless line. That is the reason why an additional thresholding is applied as well — to identify most prominent peaks, which correspond to the final lines detected in the image. The threshold helps to filter out noise and spurious detections.

An example is depicted in the Figure 4.1. In the left image, there are seven detected edge pixels belonging to a single line. Each of them votes for all lines passing through it. The accumulator array is visualised in the right image. Notice the seven wave-like curves for seven input points (edge pixels). All seven curves meet in one parameters bin, $\theta = \pi/4$ and $r = 15$, which is the sought for line. There is an error caused by the discretisation, the ground truth value is $r = 10\sqrt{2} \doteq 14.142$.

This approach can be applied for other geometric primitives too, for example a circle is parametrised by a triplet $[x_c, y_c, r] \in \mathbb{R}^3$ such that $(y-y_c)^2+(x-x_c)^2-r^2 = 0$. Nevertheless, the search space (accumulator size) increases with each parameter and gets prohibitively large very easily. The overall time complexity of the algorithm is polynomial in the number of parameters and linear in the number of points — $\mathcal{O}(NB^{(p-1)})$, where N is the number of points, B is the number of bins per parameter (assuming the number of bins is the same for all parameters), and p is the number of parameters of the corresponding geometric primitive.

This is the basic HT. There are many adaptations and enhancements, such as the Progressive Probabilistic Hough Transform (PPHT) [50], which aims to minimise the amount of computations needed.

The PPHT repeatedly draws a random point for voting from the input set of edge pixels. This point is removed from the input set and the accumulator is updated, i.e., the votes are cast. Then, the highest peak in the accumulator that was modified by the new pixel is checked. If the value is not higher than a threshold s, the process starts over (i.e., a new random point is drawn). On the contrary, if the peak value reaches the threshold s, it is assumed a line was found and is about to be extracted. A corridor specified by the peak in the accumulator is

looked along. It is seeking for the longest segment of pixels either continuous or exhibiting a gap not exceeding a given threshold. All such pixels found in the segment are removed from the input set. Next, all the pixels from the line that have previously voted retract their votes from the accumulator. The line segment is added into an output list (it can be conditioned by some additional criteria, such as minimum or maximum length). Finally, the procedure starts over, a random point is drawn. This repeats as long as the input set is not empty. Possibly, a stopping criterion can be introduced and checked as well.

The decision threshold s (whether a bin contains enough votes to assume a line exists) dynamically changes as votes are cast. This check is supposed to test a hypothesis whether the count of votes is due to random noise or not, or more formally: having sampled m out of N points, does any bin count exceed the threshold of s points which would be expected from random noise? [50]

This algorithm exhibits several appealing characteristics. It can be halted prematurely yet still identify some of the lines. Additionally, it operates without the need for a termination criterion. Depending on the data, only a minority of points may contribute votes, while the remaining points are disregarded as supplementary evidence for the detected features. And, importantly, line segments are extracted implicitly, without a need of a post-processing.

An illustration of line segments detected by this algorithm is in the Figure 4.3 (a). Gaussian blur with kernel size of 7 pixels and standard deviation $\sigma = 2$ was applied before the Canny edge detector with the thresholds set to 20 and 80. The PPHT was evaluated with the minimum line length hyper-parameter of 20 pixels and maximum line gap set to 8 pixels. It can be seen that the line segments are quite scattered, especially in noisy areas, and that some areas are almost completely left out. There are just a few line segments detected in the shaded parts of the image.

4.1.2 Line Segment Detector

The Line Segment Detector (LSD) is an advanced gradient-based algorithm developed for detecting straight line segments, introduced by Rafael Grompone von Gioi *et al.* in 2012 [51]. It takes only a grey-level image as input, no hyper-parameters are required. The algorithm actually depends on several numbers that determine its behaviour, but these values were carefully chosen to ensure effective performance across all types of images. The algorithm achieves accurate results while having a linear time complexity, which makes it a very suitable option for real-time applications.

Before the actual algorithm begins, the input image is scaled to 80% of its original size. Scaling is achieved through Gaussian sub-sampling: the image undergoes filtering with a Gaussian kernel to prevent aliasing and quantisation artefacts before being sub-sampled. Then, a gradient for each pixel is obtained using a 2×2 convolution mask. In particular, the gradient magnitude and a *level-line angle* are computed. The level-line can be seen as a line perpendicular to the gradient with approximately the same level of intensity values. The gradient magnitude determines the amount of change in the intensity in the gradient direction. Pixels exhibiting a small

gradient magnitude typically correspond to flat areas or regions with slow gradients. Moreover, such pixels inherently exhibit a higher error in gradient computation due to the quantisation of their values. In the LSD algorithm, pixels with a gradient magnitude smaller than a threshold are consequently discarded and not utilised — it is assumed they do not belong to any line segment.

Next, the pixels are ordered based on the gradient magnitude. Pixels with high gradient magnitude often correspond to the more contrasted edges, hence it makes sense to process them first. Because sorting algorithms usually require $\mathcal{O}(n \log n)$ operations in order to sort n values, pseudo-ordering is used instead as it is possible in linear time. The pixels are simply classified into several (1 024 in particular) bins with equal length according their gradient magnitude. Pixels belonging to the bin with highest magnitudes are processed first.

The algorithm then consists of three major steps: region growing, rectangular approximation and rectangle validation. Unused pixels in the pseudo-ordered list are sequentially processed.

First, a seed pixel is taken from the pseudo-ordered list and used as a starting point. Each region starts just with this one seed and the *region angle* is set to the level-line angle at that pixel. Then, pixels adjacent to the region are tested and those with level-line angle equal to the region angle up to a certain tolerance are added to the region. At each iteration, the region angle is updated to the average orientation of the level-lines of the pixels belonging to the current region. This iterative process continues until no additional pixels can be added. During the region growing, an 8-connected neighbourhood is used and the angle tolerance is set to 22.5 degree (which corresponds to 45 degree range).

When a region is established, it is approximated by a rectangle. The group of pixels is conceptualised as a cohesive entity resembling a solid object, where gradient magnitude of each pixel serves as its *mass*. The centre of mass of this pixel group is then designated as the centre of the rectangle. The primary orientation of the rectangle is aligning with the first inertia axis of the region. The dimensions of the rectangle are adjusted to the minimum values required to fully encompass the whole region.

Last step is to validate the rectangle. It is a complex procedure which will not be described in details, but there are a few key concepts worth mentioning:

- Rectangle validation involves a concept of aligned points. It refers to the pixels within the rectangle whose level-line angle closely matches the primary orientation of the rectangle, within a specified tolerance. The more aligned points, the better.

- Several adjustments are attempted on the initial rectangle approximation to achieve a more accurate representation. These adjustments involve testing variations in both width and lateral position of the rectangle.

- If the density of aligned points is low, it is tried to cut the region either by reducing the angle tolerance or region radius. The idea is to split the region into two (or more) regions which can be better approximated by smaller rectangles.

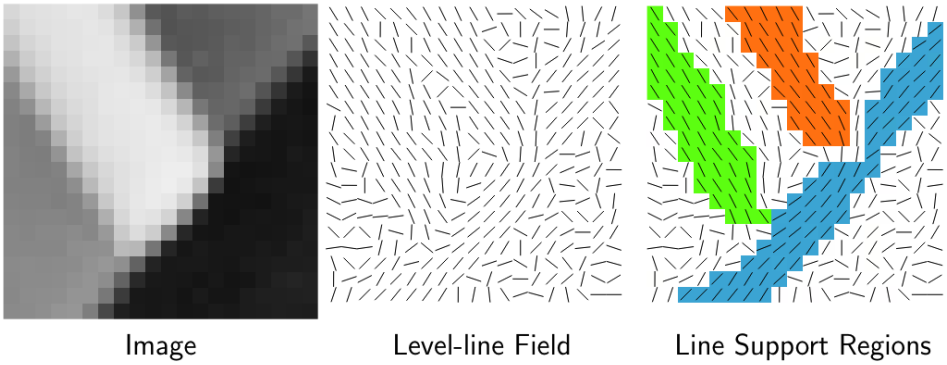


Figure 4.2: LSD and its line support regions [51].

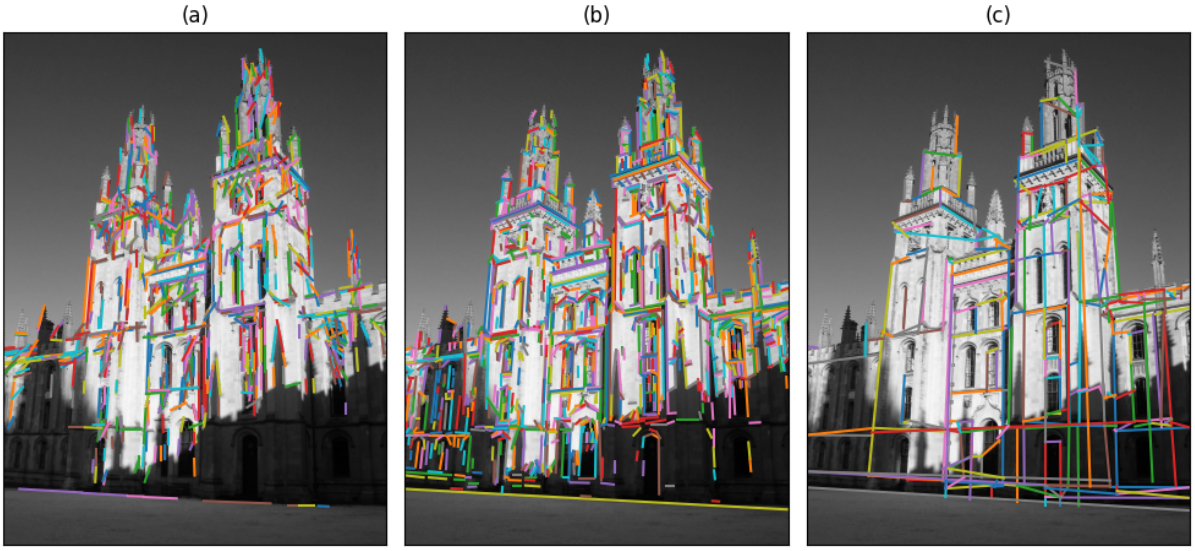


Figure 4.3: Different approaches to line segment detection: (a) PPHT, (b) LSD and (c) Deep learning based detector.

If a rectangle is successfully validated, corresponding pixels are marked as used and are not considered anymore at all, i.e., cannot be used as seed pixels and cannot be assigned to a region. On the contrary, if a rectangle is not validated, the pixels are not used as seed pixels, but they can still be assigned to a region.

The three steps repeat over again as long as there are some seed pixels in the pseudo-ordered list.

For a better conception, the main idea is depicted in the Figure 4.2. A grey-scale image is represented by level-lines. They are then grouped into regions, which are finally approximated by rectangles and verified.

An illustration of line segments detected by the LSD is in the Figure 4.3 (b). It can be seen that many edges are well detected, even in shaded areas of the image. There are some lines which seem to be insignificant or even false positive, but many of them could be easily filtered out by their length.

4.1.3 Deep learning based detector

Another considered approach was to use detectors based on deep learning models. Many neural networks are trained to detect line segments or *wire-frames* [52], [53]. A wire-frame refers to a visual representation of a 3D object or scene using only its underlying geometric structure, typically consisting of lines and vertices. A wire-frame does not include surface details such as texture, colour, or shading; instead, it focuses solely on the shape and layout of the edges and vertices. Hence, it is a subset of lines in an image. Furthermore, the lines should be characteristics for the object or scene since they describe and encapsulate its shape.

There are also neural network architectures which allow End-to-End training to predict wire-frames. One such network is the L-CNN [54]. It contains four modules: a feature extraction backbone, a junction proposal module, a line sampling module, and a line verification module. The backbone layer provides intermediate feature maps for the successive modules. The junction proposal module tries to predict junctions, which are the vertices in the wire-frame. The line sampling module predicts the lines (edges) between the junctions (vertices). These lines are then classified by the last module. The output of the L-CNN are the positions of junctions and the connectivity matrix among those junctions.

This approach was further extended by introducing line priors obtained by the HT [55]. Trainable HT blocks are added into a deep network, the L-CNN in particular. The HT block contains additional trainable convolution layers to perform local operations on the HT accumulator space (discretised line parameters, as described in the Section 4.1.1). Local operations in accumulator space correspond to global operations in the image space. Therefore, local convolutions over the bins are global convolutions over lines in the image. It is demonstrated in the original work that the approach is effective especially for small training datasets.

This model was also evaluated and considered in this thesis. An illustration of the resulting line segments is in the Figure 4.3 (c). It can be seen that the wire-frame is sparse in comparison to the previously explained methods. Also, it covers many of the characteristic and most significant edges. These properties could be beneficial for a line matching, which will be used in the transformation estimation later. But, at the same time, many edges are missed (which leaves almost empty regions) and many segments seem to be inaccurate and false positive.

4.1.4 Post-processing of detected lines

One utilisation of line segments in the transformation estimation process resides in obtaining additional constraints based on line correspondences. However, finding line correspondences in an image with a noisy set of detected line segments is a difficult task. It can be simplified by a thorough post-processing of the detection process.

Ideally, the set of line segments should contain only those which are significant, distinctive, and which capture the nature of the object. The wire-frame definition would nicely fit this description, but results received by the deep learning approach tried in the Section 4.1.3 are insufficient. Some of the significant segments are missing (are not detected; false negative segments) and some of the detected segments do not really represent any underlying edge (false

positive).

The PPHT method described in the Section 4.1.1 is a line detector, not wire-frame, which means that even weak edges (not so conspicuous) are detected. Consequently, there are fewer false negative segments. However, the resulting set of segments contains many false positive samples which also makes hardly applicable.

Out of the three methods, the LSD detector (described in the Section 4.1.2) appears to be the most promising option, leading to the decision to use it. The sensitive gradient approach detects most of the line segments present in an image, leaving just a few false negative edges undetected. At the same time, there are not so many false positive segments. Noisy and cluttered areas generate almost no samples due to the rectangle validation step. Short segments which may seem like a noise at first turn out to copy a weak edge present in the image. Overall, the LSD detector produces a low number of false detections but a high number of segments. They must be filtered in order to pick only the most significant ones.

The filtering process consists of two major steps. First, all segments with a length shorter than a threshold (value of 40 pixels was used for the purpose of this thesis) are removed. Second, the rest of the segments is filtered in such a way no two lines are located near each other. Even though it can happen that there are two significant line segments at one place (e.g., a stripe has two sides), determining correspondences based on the spatial location would be basically impossible. Therefore, only one of them can be kept. The algorithm is as follows.

All pairs of line segments are iterated over and when two segments are located near each other, only the longer one is preserved. The distance between two segments is defined as a combination of the distance from a line and the distance between their projected points: First (longer) segment generates a line. The distances of the border points of the second segment from this line are computed. If both border points are close enough (a threshold of 9 pixels was used), the process continues. The two border points are projected onto the line. If the first line segment and the projected second line segment share an intersection, then the second segment (which is shorter) is removed. If they share no intersection, but they are very close in terms of the minimum distance between their points combinations (a threshold of 5 pixels was used), the shorter one is removed too.

It was also tried not to delete one of the segments, but combine them together. The longer segment was extended by the shorter one — a four-tuple of points created by the first segment border points and the second segment projections is established. Outer points of this four-tuple are used to determine the new (combined) segment border points. This approach had a positive impact on the PPHT detection results because it often fails to detect long segments - they contains breaks and gaps. However, the PPHT was not used after all.

For reasons which will be described in the Section 4.2.1, one additional step was applied. Long line segments (in particular, line segments longer than 110 pixels) were split into several shorter ones.

The post-processing steps are visualised in the Figure 4.4. Output of the LSD detector is in the first image, filtering by line length comes right after, filtering by location follows, and

Figure 4.4: Stages of post-processing of detected line segments: (a) output of the LSD, (b) filtering by length, (c) filtering by location, (d) split of long lines.

the last image depicts the final set of line segments after the splitting of long lines. The figure contains also the number of segments in each stage. For clarity, same lines across images keep the same colour.

The overall time complexity of the algorithm is $\mathcal{O}(n^2)$ as all pairs of detected line segments must be analysed. It could be potentially decreased, e.g., if segments were sorted and a sweeping-like type of algorithm was applied. However, this idea was not thought out thoroughly. Nonetheless, the database images are processed offline which allows a longer processing time.

For query images, the line segments are filtered even more and only those which are located inside the bounding box are kept.

Despite the promising results produced by the LSD detector, there was also consideration given to combining line segments from multiple methods. However, this approach did not yield any benefits. The LSD detector is highly sensitive and generates the majority of segments independently. It results in the loss of most added segments from other detectors during the filtering process.

4.2 Lines utilisation in transformation estimation

Line segments detected in images can be utilised already in the process of generating naive hypotheses. It would be possible to create descriptors for them and apply matching algorithms, just like it was done with keypoints so far. But it would lead to the same problems and solutions as with keypoints. Instead, line segments can serve as additional constraints to the point correspondences, upgrading the family of used transformations and making the model parameters more precise.

Consider a naive hypothesis generated by a keypoint pair. The core idea is to detect segments located near the keypoints, establish a matching between them and generate a set of equations to be solved. These additional constraints allow for the underlying lines and potentially improve the original naive hypothesis.

Two approaches were tested and compared. The first one projects segments from the first

image into the second one using the naive hypothesis and then attempts to find corresponding segments to improve the naive hypothesis. The second approach finds segments in both original images and establishes a matching between them right away.

4.2.1 Projected segment matching

The first approach utilises an existing naive hypothesis before it is tried to improve it by considering line segments detected in the neighbourhood of the corresponding keypoints. The overall process consists of four major steps: find close line segments; transform segments from the first image using the naive hypothesis; match the transformed segments with those present in the second image; and estimate new transformation parameters. The first three steps will be described in this section and the last step right in the subsequent one.

The process of finding close line segments is similar to the algorithm used for filtering segments in the detection post-processing phase. Given a point and a set of line segments, the task is to find a subset such that the point is close to the segments. Assume a single line segment. Initially, distance of the point from the underlying line is computed. If the distance is short enough (threshold of 50 pixels was used in this thesis), the process continues. This simple and computationally efficient operation eliminates most of the entries. Next, the point is projected onto the line. If the projection ends up lying right on the segment, it is accepted. If the projection does not end up right on the segment, but the distance of the original point to one of the segment border points is short enough (using the very same threshold as before), it is accepted as well. Otherwise, the point lies far from the segment and therefore is declined. This is the standard definition of a distance between a line segment and a point, but the line distance check in the beginning makes to process faster as less operations are needed.

Close segments are found in both images for a given keypoint correspondence. As the second step, segments from the first image are transformed using a naive hypothesis. This is straightforward since all the transformations are linear — the border points are projected independently, resulting in new border points of the new segment in the second image.

The third step consists of matching the segments. Projected segments from the first image and original segments in the second image are considered. As it was described in the Section 3.4.4, this is a well-known task of matching in a bipartite graph, whose optimal solution is computationally expensive. Therefore, a greedy approach is used instead. Thanks to the detection post-processing, the presence of line segments in images is quite sparse. This leads to an observation that the greedy approach finds the optimal solution in most correct cases (correct in the sense that the keypoints really do correspond). It works as follows:

All pairs of segments are considered. In order to allow their matching in the first place, several conditions must be satisfied: the underlying lines must not share an angle higher than a threshold (value of 30 degrees was used); and the maximum distance of two border points from the opposite segment must not be too large (maximum of 60 pixels was used). In other words, only segments with similar direction and location are considered, regardless of their length.

Note that the maximum distance condition is the reason for the last step of the detection post-

Figure 4.5: An example of a segments matching. In (a), the detected (red) and projected (yellow) lines are depicted. In (b), the established matching is visualised.

processing procedure, the line splitting (as described in the Section 4.1.4). If a long line segment was about to be matched with a short one, this check would not pass and the correspondence would not have been allowed.

Among all allowed correspondences, final matching is selected in a greedy way. A distance of two line segments is computed as the mean distance of the border points from the opposite segment. The correspondences are cast from the lowest value to the highest, as long as the distance does not exceed a threshold (value of 35 pixels was used). This can happen, for example, for two parallel line segments which are 50 pixels away from each other. It passes the initial check, so the correspondence is allowed, but it does not make it to the final matching.

Two examples of the matching results are shown in the Figures 4.5 and 4.6. Red dots across all images represent the corresponding keypoints. In the left images, the selected and used line segments are visualised in red. In addition, yellow segments represent the transformed segments. In the right images, the final matching is shown. It can be seen in the Figure 4.5 that one of the lines is correctly left out with no correspondence. However, the lines are not detected perfectly in this area and there is a slight misalignment. The Figure 4.6 shows a case with two segments only, but the segments cover an ideal case where the two corresponding lines are perpendicular in reality.

This was one option how to create correspondences between line segments. These correspondences will now be used in the transformation estimation to improve the naive hypotheses.

4.2.2 Transformation estimation constrained by segment correspondences

Once the line segment correspondences are established, regardless of the matching strategy, they can be used for generating additional constraints for the transformation matrix. The segments themselves are not used directly, but the underlying lines are utilised instead. This is due to the fact that the border points are often detected imprecisely and unreliably.

Assume corresponding line segments where the first segment is represented by its border points $(\mathbf{b}_1, \mathbf{b}_2)$ and the second segment utilises its underlying line $\mathbf{l}$. The goal is to estimate a transformation $\mathbf{A}$ from the first image into the second one such that the border points $\mathbf{b}_i$,

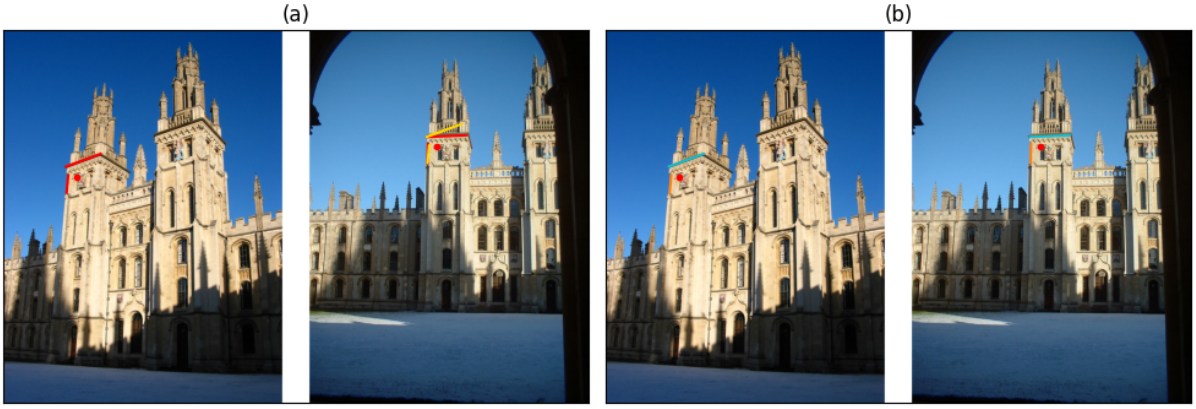


Figure 4.6: An example of a segments matching. In (a), the detected (red) and projected (yellow) lines are depicted. In (b), the established matching is visualised.

$i \in \{1, 2\}$, are projected onto the line $\mathbf{l}$:

$$\mathbf{l}^\top \mathbf{A} \underline{\mathbf{b}}_i = 0 \tag{4.1}$$

Each border point generates one equation, so a single segment correspondence generates two. Furthermore, the original keypoint correspondence also generates two equations. Features contain a piece of information about the scale which can be utilised as well, yielding up to additional two equations. All together, they form a linear system to be solved. A keypoint with a single segment correspondence can generate a set of up to six equations.

The naive transformation type was chosen to be a translation with an isotropic scale. With the additional constraints, the type can be upgraded to something more flexible. It can be decided based on the number of equations. However, it is not so beneficial to choose the most flexible type such that the set of equations is exactly-determined. The measurements are often not very precise, and the exact solution to the linear system is very sensitive to small errors; hence, the resulting transformation lacks accuracy. A less flexible type of transformation is chosen instead, resulting in an over-determined linear system. This way, the additional equations compensate for the measurement errors.

In this thesis, the affine transformation type (see the Figure 3.3) is chosen if at least P non-parallel segment correspondences are established (P stands for *parallel*, this term will be used in evaluation tables later). A set of segments is assumed to be parallel if all pairs of the underlying lines form an angle smaller than a threshold. This condition is checked only for lines in one of the images as it is unlikely to end up oppositely in the other (due to the way the segment correspondences are determined). On the contrary, if there are less than P segment correspondences or if all the segments are assumed to be parallel, a restricted similarity type (translation and an isotropic scale, as in the Figure 3.5) is used instead as it has less DoF. Parallel lines generate dependent equations which lead to under-determined linear system or imprecise results.

An ideal case is to have at least two lines which are parallel in reality (e.g., a corner).

For this reason, the threshold of the maximum angle (between lines to be considered parallel) was set to 45 degrees. Moreover, this rather high threshold for assuming parallelism mitigates consequences of false results as some of the lines are sufficiently different to produce independent equations. By a false result it is meant that the check in the other image would have resulted in an opposite outcome. However, perpendicular lines do not appear in all images. For example, only a triangular gable is queried in images which were used in the Figure 3.13. The benefit of lines then loses its significance.

Two values of the hyper-parameter P were tested and compared. First, $P = 2$ as two non-parallel lines can generate very good constraints, as it will be shown in the Figure 4.7. But the set of equations is not over-determined as it was recommended before for better robustness and stability. Therefore, $P = 3$ was tried as well. The results will be shown in the Table 4.1 at the end of this section, but the difference in mAP is often negligible.

Returning back to the linear system of equations, assume a correspondence of two keypoints $\mathbf{x} = [x, y]^\top$ and $\mathbf{x}' = [x', y']^\top$ with scales s, s', respectively. Assume a set $\mathcal{C}$ of N corresponding line segments, $\{(\mathbf{p}_i, \mathbf{s}_i, \mathbf{l}_i)\}$ for $i = 1...N$, where $\mathbf{p}_i = [q_i, r_i]^\top$, $\mathbf{s}_i = [t_i, u_i]^\top$ are border points of a segment in the first image and $\mathbf{l}_i = [m_i, n_i, o_i]^\top$ is a corresponding underlying line. If there are at least P non-parallel segment correspondences, then the transformation type is chosen to be an affine transform,

$$\mathbf{A} = \begin{bmatrix} a & b & T_x \\ c & d & T_y \\ 0 & 0 & 1 \end{bmatrix},$$

and the linear system of equations derived from (4.1) is as follows:

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \\ m_i q_i & m_i r_i & m_i & n_i q_i & n_i r_i & n_i \\ m_i t_i & m_i u_i & m_i & n_i t_i & n_i u_i & n_i \end{bmatrix} \begin{bmatrix} a \\ b \\ T_x \\ c \\ d \\ T_y \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ -o_i \\ -o_i \end{bmatrix}$$

Contrarily, if there are not P non-parallel segment correspondences, restricted similarity is used instead,

$$\mathbf{A} = \begin{bmatrix} S & 0 & T_x \\ 0 & S & T_y \\ 0 & 0 & 1 \end{bmatrix},$$

and the linear system of equations reduces to:

$$\begin{bmatrix} x & 1 & 0 \\ y & 0 & 1 \\ m_i q_i + n_i r_i & m_i & n_i \\ m_i t_i + n_i u_i & m_i & n_i \end{bmatrix} \begin{bmatrix} S \\ T_x \\ T_y \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ -o_i \\ -o_i \end{bmatrix}$$

Notice that the feature scales s and s' are not used in either of the cases. In general, it was observed that the scales are the worst estimated properties. Therefore, this property is disregarded and the scale is estimated directly from the set of equations.

In either case, the output is an improved version of the naive hypothesis (this hypothesis may not be called naive anymore; it is called a *line hypothesis* instead). Since the transformation matrix changed, the projection of the segments from the first image also changed. It means that different line segments might be matching and, consequently, another and possibly better transformation could be estimated. This leads to an iterative process. It happens that the naive transformation finds only a single matching segment. This segment is then used to adjust the matrix, and the improved hypothesis manages to match even more segments, which finally leads to an affine transformation. The most beneficial is every additional matching segment so this iterative process is stopped as soon as the number of matching segments does not increase.

An example of a line hypothesis is shown in the Figure 4.7. Red dots and red line segments represent corresponding entities used to establish the line hypothesis, which is depicted also in red. Blue dots represent the keypoint correspondence which produced the best naive hypothesis, which is also depicted in blue. To demonstrate the crucial influence of the two line segments, original naive hypothesis which created the red line hypothesis (i.e., the naive hypothesis produced from the red points) is also visualised in the image, in green. The line hypothesis has 105 inliers while the original naive hypothesis has only 22 inliers. The line hypothesis surpasses even the best line hypothesis. In this example, HOW features were used together with enhanced verification and weighted correspondences. The visualised line hypothesis ends up having the highest score among all hypotheses.

The usage of line hypotheses was evaluated on the HOW features and the final results are shown in the Table 4.1. They seem to bring an improvement, especially for the *Easy* evaluation protocol. In case of the Advanced SV, the increase in mAP is not as high. This could be attributed to the fact that there is less room for improvement, and also that the local optimisation can substitute for the role of lines. SIFT features were also used for an evaluation, but there was no betterment in terms of the mAP. The value has decreased by approximately 0.5% in all criteria.

The conflict of the local optimisation and the line hypotheses was investigated further more. For the analysis, HOW features and enhanced verification were used and the SV was evaluated over top 100 entries in a shortlist. The process was as follows: evaluate the SV over all images and save the scores. Then, add line hypotheses and check the difference, i.e., whether the number of inliers increased or decreased. Differentiate between relevant and irrelevant pairs. A

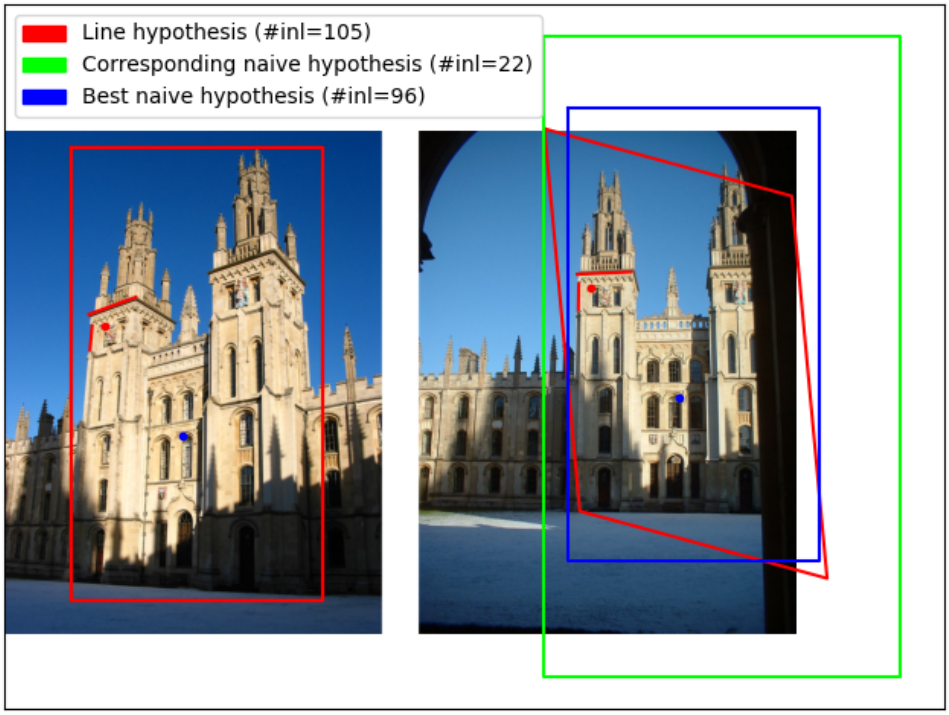


Figure 4.7: An example of the influence of lines on naive hypotheses.

		Easy	**Medium**	**Hard**
HOW	Basic SV	85.80	71.83	45.93
	Basic SV + Lines, $P = 2$	87.82	72.32	45.77
	Basic SV + Lines, $P = 3$	87.86	72.38	45.81
	Advanced SV	91.74	75.52	51.59
	Advanced SV + Lines, $P = 2$	92.17	75.72	51.79
	Advanced SV + Lines, $P = 3$	92.18	75.80	51.81

Table 4.1: mAP percentage for HOW features on $\mathcal{R}$Oxford - basic SV, advanced SV, and their versions with lines (correspondences generated by the projection strategy).

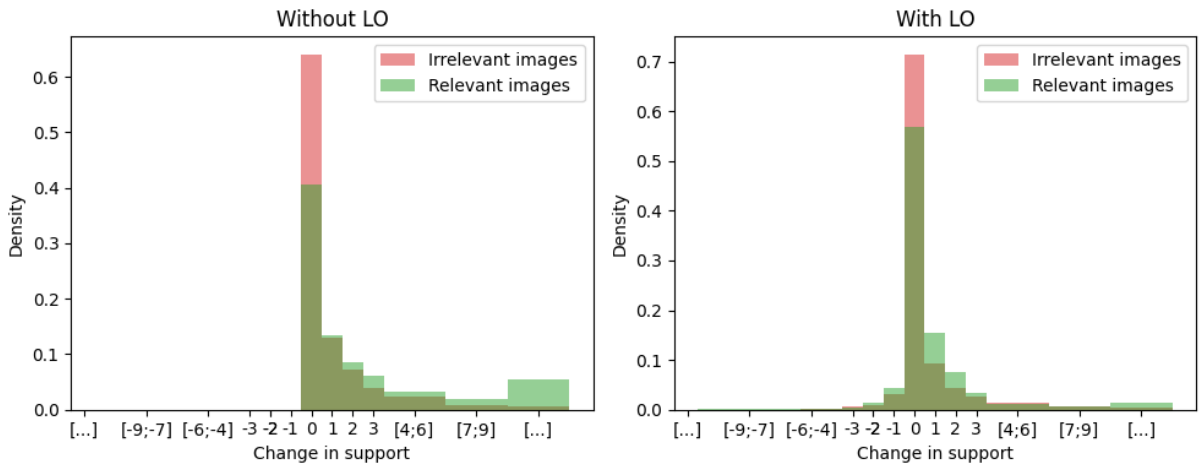


Figure 4.8: Change in support for the basic SV when line hypotheses are (not) used. Positive values are in favour of line hypotheses.

histogram was computed out of the score differences and the results are shown in the Figure 4.8 on the left. Notice especially two things. First, it is not possible for a line hypothesis to decrease the final number of inliers. If the line hypotheses are not good (in the meaning that they do not have many inliers), it does not change the fact that the naive hypotheses are still present with the very same results. Hence, the presence of the line hypotheses can only increase the score (in this case). Second, line hypotheses tend to increase the score for relevant images compared to irrelevant ones. The score for approximately 60% of relevant images has increased in comparison to 35% of irrelevant images. However, the biggest changes happened on images which had a large score already. They did not manage to surpass any irrelevant entries in the shortlist, which would lead to an increase in the mAP. Considering relevant pairs, almost 40% of images whose support increased by more than 20 inliers had already a score higher than 50 inliers.

The very same thing was evaluated also with the local optimisation on top 10 hypotheses added. It can be seen on the right side of the Figure 4.8 that the benefits are not as significant as before. Only approximately ~~60%~~ 30% of relevant images increased their support upon adding line hypotheses and even the amount of added inliers was not as high. Some of the scores were even decreased. This can happen in a case when line hypotheses kick some naive ones out of the list of top 10 hypotheses intended for the local optimisation.

In conclusion, the local optimisation really *does* compensate the effect of added line hypotheses. Typically, many point correspondences are used as an input for local optimisation, whereas only one point and a few line correspondences are used as an input for line hypotheses. This inequality is profitable primarily for the local optimisation. On the contrary, line hypotheses tend to outperform the local optimisation in cases when the image pair has sparse correspondences. There are not many point correspondences present in the local optimisation (or the points form just a small cluster). However, in these cases, there are not so many additional inliers to find and therefore the score is increased only a little by the line hypotheses.

Some additional approaches were also tried, but were relinquished due to their poor performance. First, it was considered to use a combination of a translation and an anisotropic scale

as an upgraded transformation type for line constraints (see the Figure 3.5, restricted affine 1). However, this method did not show any positive impact or results. In the case when only parallel segments are available, one of the scales was often very poorly estimated.

Another considered option was to constraint the transformation matrix such that lines are mapped onto lines, i.e., $\mathbf{l}A^{-\top}\mathbf{l}' \simeq 0$. Nonetheless, this approach seemed to provide exaggeratedly over-fitted results. It tried to match the lines even at infinity to the detriment of points in the image itself.

4.2.3 Spatial verification on Tokyo dataset

The spatial verification algorithm was evaluated also on the Tokyo dataset (see the Section 2.7.2). Objects (or places) in the images are photographed in different lightning conditions, but from nearly the same location and with nearly the same viewing direction. This makes it potentially a good use case for the spatial verification algorithms since the estimated geometric transformation should not change too much between the images.

This dataset does not distinguish between different image labels, like $\mathcal{R}$Oxford. Therefore, just one evaluation protocol is used and only a single mAP value is considered in this case. The dataset has 1 125 images, each image is used as one query and the goal is to find the other (and only) two positive (relevant) images. Due to the high number of queries and the low number of sought images, it was decided that the spatial verification is performed only on top 10 images.

HOW features were used, so the baseline and the shortlist is given by the ASMK retrieval system. Its mAP is 88.61%. There are 90 queries which can be improved by the spatial verification, i.e., there are 90 queries which have both relevant images in the top 10 and at least one irrelevant image in the top 2. The rest of the queries either already has the two relevant images in the top 2 or at least one relevant image is not present in the top 10.

The configuration of the spatial verification was as follows: weighted correspondences with the weight adjustment function set to $f(x) = \max(x, 0)$; enhanced verification with the error threshold $\Delta = 62$; and line hypotheses.

Given this spatial verification configuration, the mAP increases to 91.33% with only 9 queries which can be improved. However, the fundamental benefit comes from the spatial verification itself, not from the line hypotheses. Ignoring the line hypotheses and using only the naive ones results in mAP being equal to 91.27% and 11 improvable queries.

4.2.4 Direct segment matching

The first line segment matching approach described in the Section 4.2.1 is based on improving an existing naive hypothesis by projecting line segments from the neighbourhood. These projected segments are then matched with line segments detected in the second image and these relationships determine constraints for a new, possibly better, transformation.

The approach was shown to work, the hypotheses were often indeed improved. However, they are dependent on the initial naive transformation. If the features are not good, the resulting transformation is completely meaningless. As it was already mentioned, the feature scales are

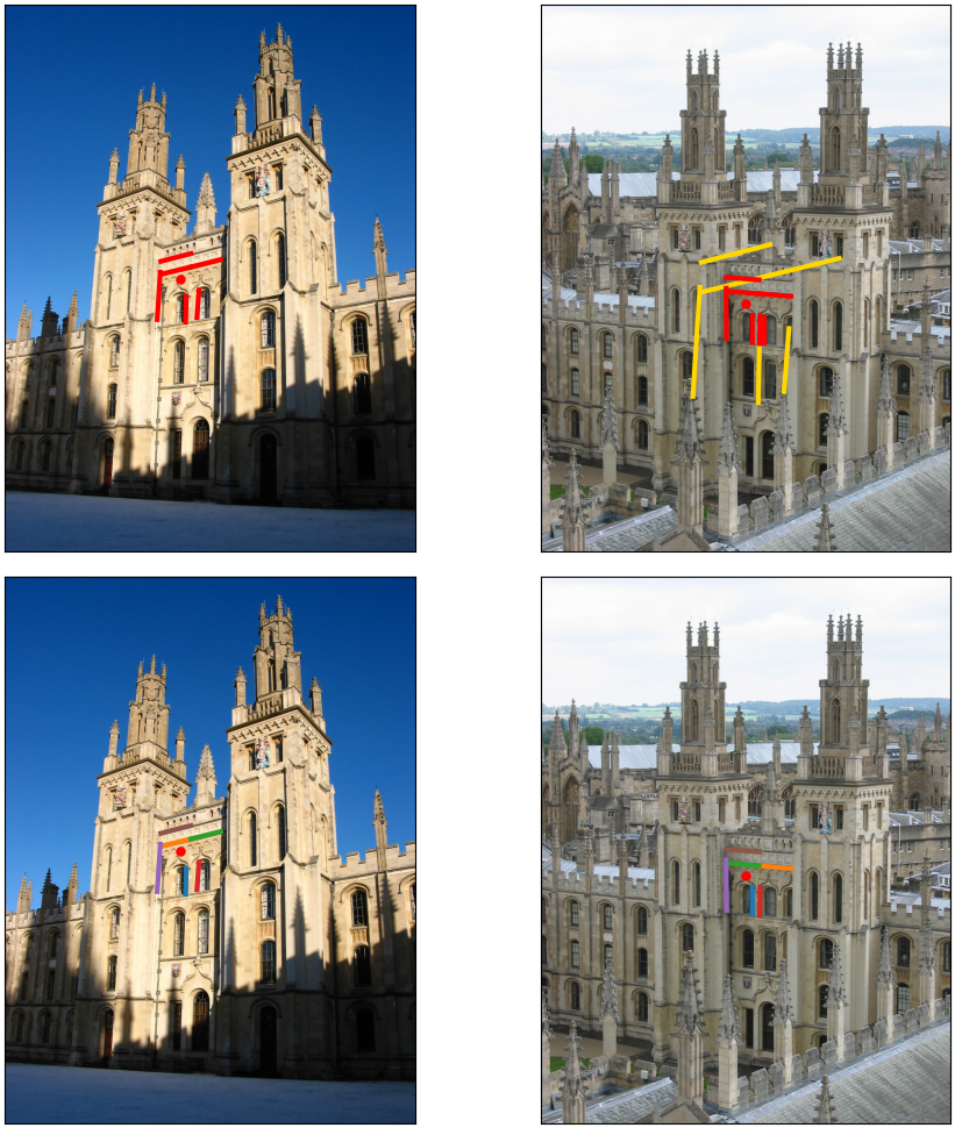


Figure 4.9: A line segments matching example. First row depicts corresponding features and close line segments as well as segment projections. Second row depicts results of the matching algorithm based on a distance.

often very poorly estimated. So even if keypoint locations really correspond, the scale might be inaccurate. The resulting naive transformation projects the lines in such a way the algorithm is unable to match them. An example of this case is depicted in the Figure 4.9 in the first row. The detected line segments are shown in red. The segments from the first image are projected into the second image (yellow) and it is clear that the lines could not be matched.

This issue gives rise to an idea: do not project the lines into the second image, but match the line segments right away. It is obvious that lines which are close to a keypoint should stay close while lines which are distant should stay distant. And this is exactly the main point.

Assume two images with two corresponding features inside each one of them. The goal is to obtain a matching of line segments in such a way it can be used as an input into the optimisation procedure described in the Section 4.2.2.

The beginning of the algorithm is the same. Given an image and a keypoint, line segments

Input: Sorted lists of line segments L_1, L_2
Output: List of line segment correspondences $\mathcal{C}$.

```
1: C ← {}
2: for l_1 in L_1 do
3:     for l_2 in L_2 do
4:         if check_conditions(l_1, l_2) then
5:             C.add([l_1, l_2])
6:             L_2.remove(l_2)
7:             break                          ▷ Breaks out of the L_2 loop only
8: return C
```

Figure 4.10: Line segment matching algorithm.

located in the neighbourhood of the keypoint must be found. Nothing has changed here, except one little detail — the distance of the keypoint from each line (represented by the segment) is saved and returned together with the list of segments. The line segments are then sorted based on the distance from the keypoint.

Given a pair of sorted lists of line segments, the matching itself can start. First segment from the first list and first segment from the second list are taken. If they satisfy a set of conditions, they are considered as correspondences and they are removed from the lists. On the contrary, if they do not satisfy the conditions, the segment from the second list is skipped for now and the same process continues with another one as long as there are some segments left. This procedure is applied also for all other segments from the first list. This algorithm is described also in the Figure 4.10.

The set of conditions is supposed to ensure that the line segment correspondence is meaningful, i.e., the line segments do have some properties in common. There are two main criteria to verify: (i) the corresponding lines have an angle lower than a threshold, and (ii) the two keypoints are located on the same side of the two lines (in a simplified way).

The first condition is very straightforward. It is expected that under-laying lines represented by the corresponding line segments do not do have a large angle. This assumption is reasonable because of the way photos are usually taken — x axis parallel with the ground and similar (upright) view orientation (as described in the Section 3.3.3). A threshold of 30 degrees was used, just like in the previous matching strategy (in the Section 4.2.1).

Second condition checks that *the two keypoints are located on the same side of the two lines*. This formulation does not make any sense yet, because lines in general do not have any orientation, hence they do not have sides. Therefore, an orientation must be first assigned to each line, or line segment, in order to be able to tell whether points are located on the same side or not. From the previous condition it is known that the lines are similar in the sense that both are horizontal, vertical, or in between. Most line segments in the images are only horizontal or vertical. It is an ideal scenario and makes the matching task easier as these are the extreme options. This additional knowledge can be used to create *a sense* of an orientation. For example, assume there is a vertical line (segment) in the first image as well as in the second image. Then it can be assumed that both of the lines are pointing upwards, in the y axis direction, and checked

		Easy	Medium	Hard
HOW	Basic SV	85.80	71.83	45.93
	Basic SV + Lines, $P = 2$	88.06	72.92	46.60
	Basic SV + Lines, $P = 3$	88.50	72.96	46.47
	Advanced SV	91.74	75.52	51.59
	Advanced SV + Lines, $P = 2$	92.25	75.97	52.04
	Advanced SV + Lines, $P = 3$	92.27	75.91	51.90

Table 4.2: mAP percentage for HOW features on $\mathcal{R}$Oxford - basic SV, advanced SV, and their versions with lines (correspondences generated by the distance strategy).

that the corresponding keypoint is located on the left (or right) side of the line.

These two conditions are enough to generate meaningful line hypotheses and even exceed the projection matching strategy in some cases. An example of the final correspondences is shown again in the Figure 4.9, in the second row. While the strategy based on a projection is unable to generate correspondences, this distance based strategy does it perfectly. The green and orange horizontal line segments are swapped, but it does not matter at all since both segments represent the very same line and lines (not line segments) are used in the optimisation step. The evaluation results are shown in the Table 4.2. The mAP has increased in all cases in comparison to the projection matching strategy (Table 4.1).

The current set of conditions has still some flaws. One of them is that two segments which are in reality far away from each other can be matched just because the under-lying lines have a similar relative distance to the keypoint. An example is depicted in the Figure 4.11. The first row of images shows the corresponding keypoints (red dots) and line segments located in their neighbourhood (red lines). The second row shows the current matching strategy with the angle and side checks. It can be seen that there are several incorrectly matched segments:

- The brown and pink line segments correspondences (the left most pair of vertical lines, excluding the one isolated and correctly matched yellow line in bottom left) are swapped. In this case, luckily, the under-lying lines are almost identical. However, it does not have to be the case every time.

- A very similar scenario happens also for the purple and red line segments correspondences (the right most pair of vertical lines). Again, they represent the same ground-truth line so the miss-match does not have any serious consequences. However, in this case, the upper border point of the upper segment in the left image is detected imprecisely and causes that the approximated line is inclined from the ground-truth line a bit. Recall that in the optimisation step, the line segment border points are projected onto a line (Equation (4.1)). Due to the miss-matched segments, the distances of the pair of border points from the line are slightly larger.

- Last incorrectly matched segments correspondence is the light blue pair of horizontal lines.

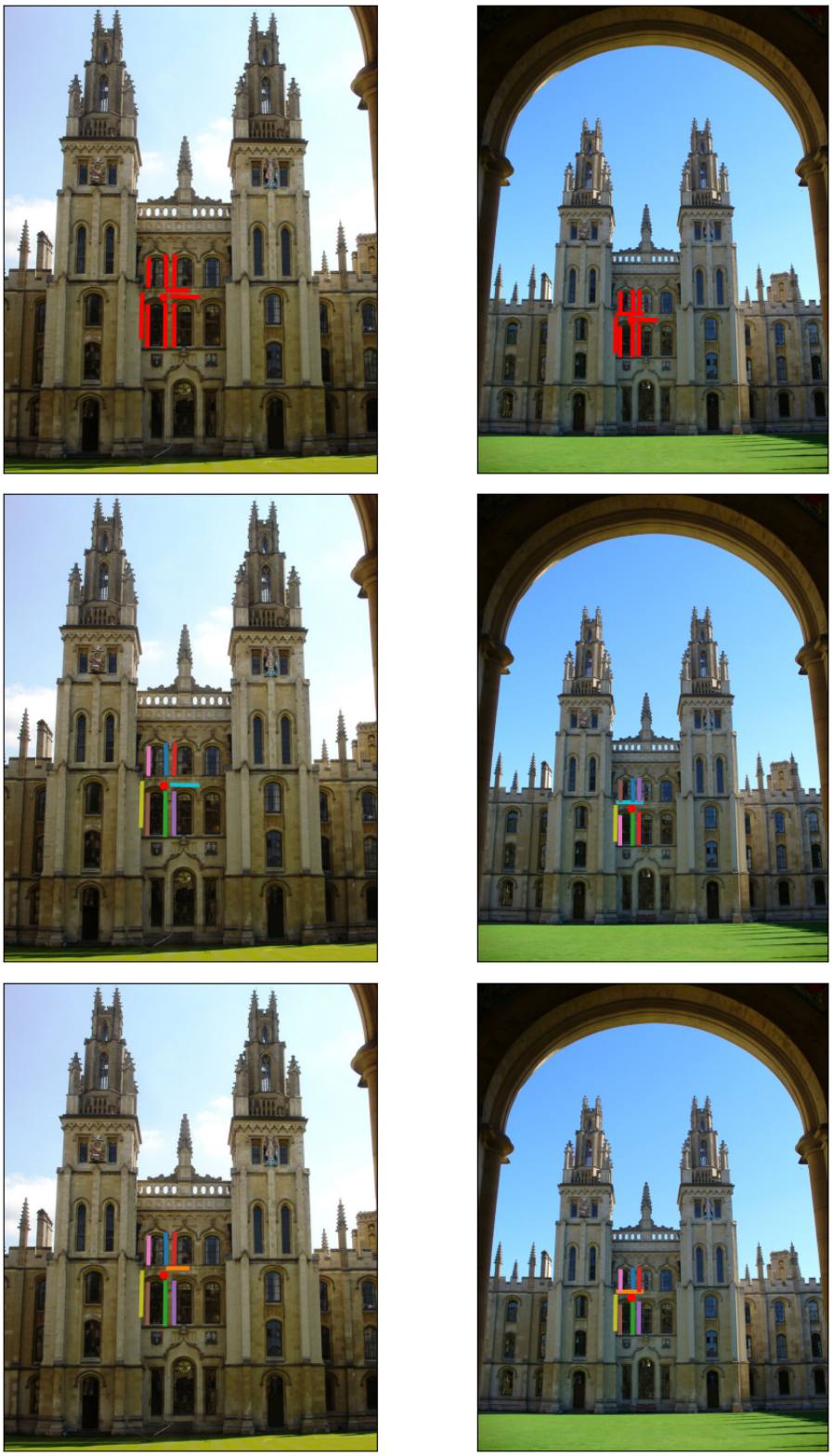


Figure 4.11: Difference in direct line segment matching after adding the quadrant check.

They both are horizontal and they both are close to the keypoint, but they do not really correspond. It is a mistake which should not happen.

To address the aforementioned issues, one more condition was experimented with. It is clear from the example that corresponding line segments should be located in the same relative position to the keypoint. Also, the line side check is not enough since it works only with the lines, not segments. Therefore a new condition was developed and examined. The keypoint can be used to divide the image into four quadrants, where the quadrants are separated by two perpendicular lines passing through the keypoint. Due to the way photos are taken (as described earlier in the first angle condition), it is reasonable to choose lines which are parallel with the x and y axes, respectively. When two segments are about to be matched, they are expected to lie in the same quadrant.

This condition was implemented as four independent checks: is the segment above, below, left, and right. Their combination then forms the whole quadrant check. However, the checks are not complements of their opposites, e.g., above is not a complement of below. In order to cover the whole range of possible cases (a line segment fully above, close to the keypoint, fully below, and all over the keypoint), each check is fully independent with three possible states, e.g., above, non-above, and *undefined*.

To decide whether a line segment $([a_x, a_y]^\top, [b_x, b_y]^\top)$ is above a keypoint $[x, y]^\top$, the y coordinates of the border points and the keypoint are compared. If any of the border points is near the horizontal level of the keypoint, i.e., $\min(|a_y - y|, |b_y - y|) \leq \tau$, where τ is a threshold, then the segment position is marked as *undefined*. It is a safety measure to recompense line segment detection inaccuracies and keypoint location approximation error. Also, it prevents from sudden changes between above and non-above extremes since this ensures an intermediate level. If both points are sufficiently distant, the classification can continue. If both border points (their horizontal levels, to be exact) are higher than the keypoint, i.e., $\min(a_y, b_y) > y$, then the line segment is claimed to be above. Otherwise, the segment is marked as non-above. A pair of line segments can correspond if and only if at least one of them is marked as *undefined* or they are both (non-)above.

Similarly, the line segment can be marked as below, non-below or *undefined*. If none of the border points is located within the τ stripe and both points have the horizontal level lower than the keypoint, i.e., $\max(a_y, b_y) < y$, then the segment is marked as below. Otherwise, it is marked either as non-below or *undefined*. The condition for the corresponding segments is the same as before — at least one of them must be *undefined* or they must be marked equally.

The vertical classification for right and left is made analogously to the above and below, respectively, only with the x coordinates and the vertical levels. A line segment correspondence is allowed if and only if all the quadrant conditions are satisfied.

The third row of the Figure 4.11 shows the final matching when the quadrant check is included. It can be seen that the vertical line segments (pink, brown, purple, red) are now matched correctly as well as the horizontal orange ones. It was also evaluated with the threshold $\tau = 10$ and the results are shown in the Table 4.3. A small improvement is noticeable in

		Easy	Medium	Hard
HOW	Basic SV	85.80	71.83	45.93
	Basic SV + Lines, $P = 2$	88.78	73.05	46.69
	Basic SV + Lines, $P = 3$	88.75	72.99	46.47
	Advanced SV	91.74	75.52	51.59
	Advanced SV + Lines, $P = 2$	92.48	75.99	52.14
	Advanced SV + Lines, $P = 3$	92.48	75.96	52.08

Table 4.3: mAP percentage for HOW features on $\mathcal{R}$Oxford - basic SV, advanced SV, and their versions with lines (correspondences generated by the distance strategy, including the quadrant check).

comparison to the previous version without the quadrant check.

4.2.5 Artificially corrupted data

The original idea why to utilise lines for the spatial verification on HOW features was that the HOW features have inaccurate feature locations. This may lead to imprecise hypotheses and consequently also results. During the development it appeared that the lines can compensate for the errors mostly in two cases: (i) in sparse environments where are not so many correspondences; (ii) or when the input tentative feature correspondence is obviously incorrect but not so different.

Therefore it was tried to artificially corrupt some data and observe whether the lines manage to compensate for it by improving the results. The SIFT features were taken and their scales were inverted: $s' = \frac{1}{s}$. This can have serious consequences mainly for images with a change in a scale.

The spatial verification algorithms were executed over the $\mathcal{R}$Oxford dataset with the corrupted data and the results are in the Table 4.4. It can be seen that for the Basic SV, the lines really *do* compensate for the incorrect scales and increase the score to values comparable to the original ones. In case of the Advanced SV, the lines also improve the results, but the change is not as significant. Most of the relevant image pairs have a similar scale so the inversion does not ruin the naive hypotheses completely, but gives a solid foundation in terms of inliers for the local optimisation. In both cases, the line hypotheses were generated by the direct segment matching strategy with the quadrant check (Section 4.2.4) and $P = 2$ (P refers to the number of non-parallel lines needed for an affine transformation type to be used, see 4.2.2).

Another experiment was to drop the scales completely. All naive hypotheses are disregarded and only line hypotheses are considered. The very same setup as in the previous case was used. Despite the complete loss of data, the results are surprisingly good, as it can be seen in the Table 4.5. They even exceed several approaches which use the scales. It happens that some image pairs do not manage to find a line hypothesis, i.e., no line segment matching is found for any tentative correspondence. In case of the HOW features, this happened for 108 image pairs where 88 of them were irrelevant, 17 relevant, and 3 pairs were unclear (they are ignored during

		Easy	Medium	Hard
SIFT	Basic SV	69.74	55.92	33.49
	Advanced SV	73.91	58.05	36.12
Corrupt. SIFT	Basic SV	66.23	52.77	30.07
	Basic SV + Lines	69.50	55.73	33.46
	Advanced SV	72.32	56.54	34.12
	Advanced SV + Lines	73.26	58.27	36.40

Table 4.4: mAP percentage for the SV with SIFT features in comparison to corrupted SIFT features on $\mathcal{R}$Oxford.

		Easy	Medium	Hard
SIFT	Basic SV	69.60	55.49	32.94
	Advanced SV	73.32	57.97	35.93
HOW	Basic SV	89.10	73.08	46.68
	Advanced SV	92.67	76.03	52.12

Table 4.5: mAP percentage for the SV with SIFT and HOW features on $\mathcal{R}$Oxford when no scale is used.

the evaluation, see the Section 2.7.1). This was even more apparent for the SIFT features. No generated hypothesis had 363 irrelevant image pairs, 38 relevant, and 7 unclear pairs. Image pairs with no generated hypothesis were treated as if they had zero support, zero score. Since the majority of cases regards irrelevant image pairs, the overall results in terms of the mAP are good.

4.3 Vanishing points for homography estimation

Another approach is to use lines to estimate something called a *vanishing point*, sometimes abbreviated as a VP. Its location in the real world is known while it is measurable in images. This makes it possible to adjust transformations in such a way they preserve the vanishing points.

First, the very basics of the projective geometry in images will be described and the vanishing points will be defined. Then, vanishing points in images will be estimated using the RANSAC algorithm and subsequently, they will be used to define hard constraints on transformations between images.

4.3.1 Projective geometry

In order to get a notion about what a vanishing point actually is and why is it any good, projective geometry will be shortly described. It should give enough foundations to understand

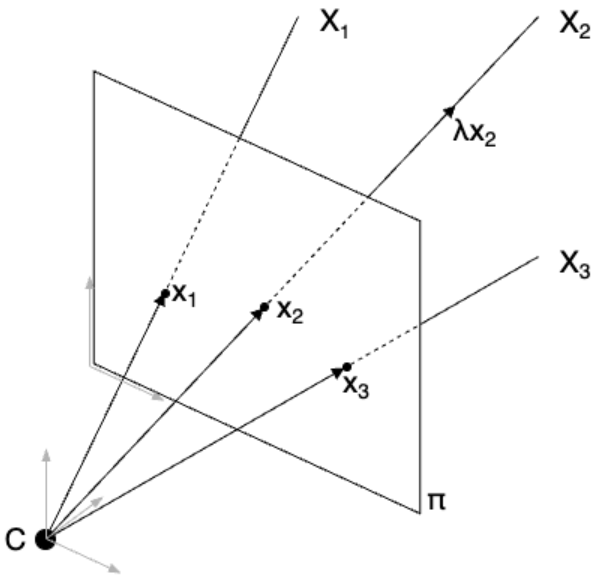


Figure 4.12: Projective plane geometry.

the rest.

Imagine a standard three dimensional world and a camera taking a picture of a scene. When a picture is taken, the 3D scene in front of the camera is projected onto a 2D plane, the image. There is a line between each 3D point and the camera centre. Each such line defines a single point, projection, on the 2D *projective* plane. It is the intersection of the line and the projective plane.

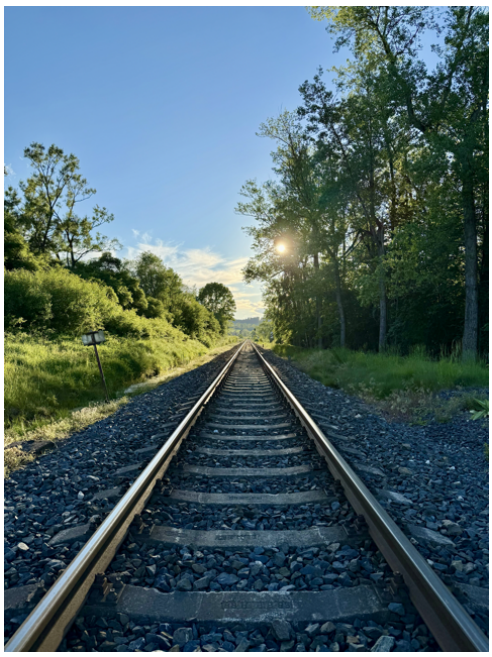
A visualisation is depicted in the Figure 4.12. A camera centre is in the point $\mathbf{C}$. The scene shows three points, $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$, and their projections onto the projective plane π. The projections are intersections of the projective plane π and the lines. These lines are determined by the camera centre C and direction vectors $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$, respectively. Scaling a direction vector of a line does not change the projection, e.g., both lines determined by $\mathbf{x}_2$ and $\lambda\mathbf{x}_2$ refer to the very same point in the plane π (for $\lambda \neq 0$).

It is convenient to choose the coordinate system in such a way that the origin $[0, 0, 0]^\top$ is in the camera centre $\mathbf{C}$, first two axes (x and y) are parallel with the projective plane π and the distance of the camera centre from the projective plane π is one, i.e., $\pi : z = 1$. The basis of the system is depicted by grey vectors in the Figure 4.12. With this system, a point with coordinates $[x, y]^\top$ in an image corresponds to the vector $[x, y, 1]^\top$ as well as vectors $[\lambda x, \lambda y, \lambda]^\top$ for $\lambda \neq 0$.

Given a 3D point $\mathbf{X}$, three different situations may arise [56]:

1. If $\mathbf{X} = \mathbf{C}$, then there is an infinite number of lines passing through this pair of points, or actually a single point since they are equal. An infinite number of lines also intersect π in all of its points, therefore the projection of $\mathbf{X}$ contains the whole plane π.

2. If $\mathbf{X} \neq \mathbf{C}$ lies in a plane π', which is parallel to π and passing through $\mathbf{C}$ ($\pi' : z = 0$), then there is exactly one line passing through $\mathbf{X}$ and $\mathbf{C}$. This line does not intersect the projection plane π in any point, hence the projection of $\mathbf{X}$ is empty. These points have



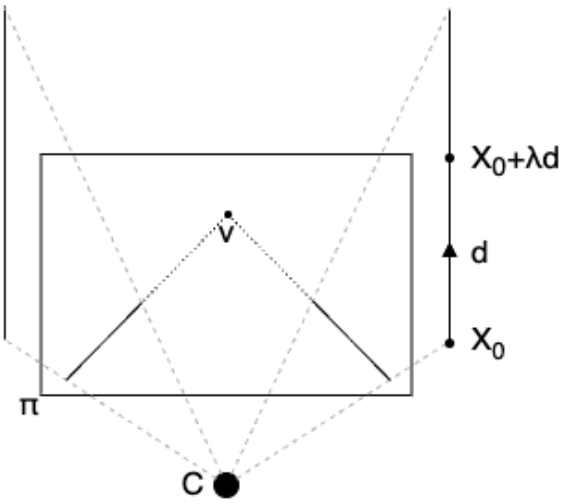


Figure 4.13: An example and a diagram of two parallel lines and a vanishing point.

coordinates $[x, y, 0]^\top$ and are called *points at infinity*.

3. In other cases, there is exactly one line passing through $\mathbf{X}$ and $\mathbf{C}$, and this line intersects the projective plane π in exactly one point. Hence, the projection of $\mathbf{X}$ contains exactly one point. This is the most common situation which is also depicted in the Figure 4.12.

The projective plane and points at infinity form a *projective space*.

Formally: a projective space $\mathbb{P}^2$ is a vector space with three dimensions, with the zero vector excluded (i.e., $\mathbb{R}^3 \backslash [0,0,0]^\top$), and with equivalence relation classes $\underline{\mathbf{x}} \simeq \lambda \underline{\mathbf{x}}$ for $\lambda \neq 0$ (where $\underline{\mathbf{x}} = [x, y, 1]^\top$).

Elements $\underline{\mathbf{x}}$ are often called points, even though they represent a whole line (with one excluded point). The projective space $\mathbb{P}^2$ includes also points at infinity.

Given this projective space and the camera model, the appearance of parallel lines can be examined. See the Figure 4.13, where is a diagram together with a real picture. There are two parallel lines being captured, in case of the real image they are two rails. However, the projections of the parallel lines are not parallel anymore, they intersect somewhere. Consider a line determined by the point $\mathbf{X}_0$ and the direction vector $\mathbf{d}$. Each point $\mathbf{X}_0 + \lambda\mathbf{d}$ of the line is projected onto the projective plane π. As λ goes to infinity, the projection gets closer to the point $\mathbf{v}$. The point $\mathbf{v}$ is a *vanishing point*, representing the projection of a point that moves infinitely in one direction along a space line.

Notice that the other parallel line has the very same vanishing point. As λ goes to infinity, the root point $\mathbf{X}_0$ gets negligible and only the direction vector $\mathbf{d}$ determines the projection. Parallel lines have the same direction, therefore the vanishing point is shared as well.

Lines in an image which are parallel are said to intersect at infinity, which actually *does* correspond to the points at infinity mentioned earlier. Consider two parallel lines in $\mathbb{P}^2$, $l =$

$[a, b, c]^\top$ and $k = [a, b, d]^\top$. Note the identical normal vector, $[a, b]^\top$, and different intercepts c and d. Their intersection is $l \times k = [b, -a, 0]^\top$. The third coordinate is equal to zero, hence it is a point at infinity.

The concept of parallel lines in reality not being parallel in images can be further utilised. The more parallel lines (in reality) in an image, the more significant vanishing point which can be used during the transformation estimation.

4.3.2 Vanishing point estimation

It is reasonable to assume that majority of the ground-truth lines in an image form groups of lines which are mutually parallel and therefore share a vanishing point. Vanishing points can be further utilised, but they must be estimated in the first place. Lines in an image are represented by line segments, which were already detected in the Section 4.1, so they can be re-used now.

Many lines from an image are detected due to some noise or they represent an edge whose vanishing point is insignificant. Usually, only a minority of lines belongs to a concrete and strong vanishing point. For these reasons, a robust estimation algorithm must be used and the RANSAC is a clear candidate.

An existing implementation of the vanishing point estimation by the RANSAC algorithm was used in this thesis [57], [58]. It assumes only horizontal and vertical lines, nothing in between. It estimates a single vertical vanishing point and multiple horizontal ones.

It starts by classifying the line segments into three categories: horizontal, vertical, and *undefined*. A direction of each line segment is taken. It is the normalised difference of the border points, so the direction is a unit vector which is then classified as follows:

- If the absolute value of the x coordinate is greater than a threshold, it is a horizontal line;
- If the absolute value of the y coordinate is greater than a threshold, it is a vertical line;
- If none of the above applies, the line is classified as *undefined*.

The threshold value was set at 0.8, corresponding to a range of approximately ± 37 degrees. This leaves only a 16-degree range for classifying *undefined* segments, which are excluded from the estimation process.

After the line segments are classified, the RANSAC algorithm itself can be performed. The most important parts are the hypothesis generation function and the error function.

In order to generate a hypothesis θ in the form of a vanishing point, minimum sample $\mathcal{S}$ of two line segments is needed. Given a line segment $(\mathbf{a}, \mathbf{b})$, the line can be obtained as a vector product, $l = \underline{\mathbf{a}} \times \underline{\mathbf{b}}$. Then, the vanishing point is an intersection of at least two lines in an image. An intersection can be computed again as a dot product, $\underline{\mathbf{v}} = l_1 \times l_2$, however, the implementation uses a set of equations instead. The intersection $\mathbf{v}$ must lie on each line l: $l^\top \underline{\mathbf{v}} = 0$. This generates one equation, the other line generates second equation, which is the required minimum in order for the set to be exactly determined. A least squares method is used

for this linear system, which has an advantage — it can handle even more than two lines, hence is applicable also in a local optimisation.

When a hypothesis is generated, an error function can be evaluated. It is supposed to output a small values for line segments consistent with the hypothesis (a vanishing point) and large values for segments which are inconsistent. The error function was chosen to be as follows: given a vanishing point $\mathbf{v}$ and a line segment $(\mathbf{a}, \mathbf{b})$ generating a line l, a line c connecting the vanishing point and a mid-point of the segment is created, $c = \underline{\mathbf{v}} \times \left(\frac{\mathbf{a}+\mathbf{b}}{2}\right)$. Then, a point $\mathbf{f}$ on l with a fixed distance from the mid-point is taken and its squared distance from the connecting line c is used as the error value.

By using a point $\mathbf{f}$ on the line segment at a fixed distance from the line mid-point instead of the segment border points, the distance from the connecting line c does not depend on the length of the line segment anymore.

It is apparent that line segments (or lines) consistent with a hypothesis point in that direction and therefore the error is small. On the contrary, inconsistent lines have the distance much greater as the the point f is moving further away from the connecting line c. This error function is equivalent to measuring an angle between the lines, however, this method is more computationally efficient and less sensitive to numerical errors.

In terms of the RANSAC notation: a set of line segments with the same assigned category is used as the set $\mathcal{U}$ of input data points from a measurement space $\mathbb{X}$ of all 2D line segments. The goal is to estimate a vanishing point, which is an intersection of at least two lines, therefore the minimum sample size $m = 2$. One sample $\mathcal{S}$ of two line segments generates a hypothesis θ (a vanishing point) from a parameter space of all points $\Theta \equiv \mathbb{P}^2$ as an intersection of the underlying lines. This hypothesis is then evaluated over all line segments using an error function $\rho(\theta, \mathbf{x}) \rightarrow \mathbb{R}$. This function computes a distance of a point from a line. The point is on the line segment with a fixed distance from its mid-point. The line is connecting the mid-point and the vanishing point nominated by the hypothesis. The error values are compared with a threshold Δ and divided into two groups, inliers and outliers. For the purpose of this thesis, threshold $\Delta = 2$ was used. The cost function $J(\theta, \mathcal{U}, \Delta)$ returns a number of inliers for a given hypothesis, therefore a vanishing point with the highest support is chosen. Additionally, in order to accept the estimated vanishing points, the ratio of inliers in the input set must be higher than 10%.

The algorithm is evaluated on line segments with vertical and horizontal classes separately. It is evaluated only once for the vertical segments and multiple times for horizontal segments. Each time a horizontal vanishing point is estimated, inliers are removed from the input data set ($\mathcal{U}$) and the RANSAC is started again as long as the support of the last vanishing point is large enough (value of 10 was used). Also, more than 20% of the original line segments must be still present in the input data set.

The evaluation was performed over raw line segments detected in an image, before the post-processing described in the Section 4.1.4. Only segments longer than 50 pixels were used. The confidence η for stopping the algorithm was set to 0.999, with additional conditions for the minimum and maximum number N of iterations to be $10\,000 < N < 100\,000$.

To summarise it: vanishing point estimation is performed on each image. Ideally, one vertical, and multiple horizontal VPs are found. Nevertheless, it can happen that none is detected in an image. In fact, vanishing points were computed for 2 133 images in total (the rest of the images was not used in the SV), while 67 of them has no VP at all, 640 is missing a vertical VP, and 102 images have no horizontal VP. 764 images have exactly one horizontal VP and two images have even 8 horizontal VPs.

4.3.3 Vanishing point as a hard constraint

In reality, parallel lines do not intersect (they intersect at infinity, as it was explained before) regardless of the position of an observer. However, in images, projections of these lines *do* intercept and the location of the intersection depends on the camera position and direction. This brings a new measurable element which can be used during a transformation estimation. Its preservation can be used or even enforced in an optimisation.

Consider two images, each with one vertical and one horizontal vanishing point detected. In this thesis, vanishing points were used as hard constraints. It means that their correspondence is enforced during the transformation estimation, i.e., vanishing points in the first image are transformed onto vanishing points in the second image. Since affine transformations preserve parallelism (see the Table 3.2), projective transformations are estimated instead. The first attempt of a homography utilisation in the Section 3.4.6 was not successful. The transformations tent to over-fit the inliers which caused conspicuous incongruities, especially in corners of bounding boxes. The vanishing points constraints could potentially solve this issue as they preserve directions of lines.

Before concrete equations regarding the vanishing points and its constraints are introduced, start with a general homography estimation. Without assuming any knowledge about cameras, the goal is to find a homography matrix out of point correspondences.

Let $\mathbf{H} \in \mathbb{R}^{3\times3}$ be an estimated homography matrix, $\mathbf{x} = [u, v]^\top \in \mathbb{P}^2$ and $\mathbf{x}' = [u', v']^\top \in \mathbb{P}^2$ be two corresponding points. Then the following equation holds [56]:

$$\lambda \underline{\mathbf{x}}' = \mathbf{H}\underline{\mathbf{x}} = \begin{bmatrix} \mathbf{h}_1^\top \\ \mathbf{h}_2^\top \\ \mathbf{h}_3^\top \end{bmatrix} \underline{\mathbf{x}},$$

for $\lambda \neq 0$, where $\mathbf{h}_i$ are rows of the homography matrix $\mathbf{H}$ and underlined variables represent the homogeneous form. The equation can be rewritten row-wise as:

$$\begin{aligned} \lambda u' &= \mathbf{h}_1^\top \mathbf{x} \\ \lambda v' &= \mathbf{h}_2^\top \mathbf{x} \\ \lambda &= \mathbf{h}_3^\top \mathbf{x} \end{aligned}$$

Next step is to eliminate λ using the third equation,

$$(\mathbf{h}_3^\top \mathbf{x})u' = \mathbf{h}_1^\top \mathbf{x}$$
$$(\mathbf{h}_3^\top \mathbf{x})v' = \mathbf{h}_2^\top \mathbf{x}$$

and move everything to one side.

$$\mathbf{h}_1^\top \mathbf{x} - (\mathbf{h}_3^\top \mathbf{x})u' = 0$$
$$\mathbf{h}_2^\top \mathbf{x} - (\mathbf{h}_3^\top \mathbf{x})v' = 0$$

Now, reshape the equations using $\mathbf{x}^\top \mathbf{y} = \mathbf{y}^\top \mathbf{x}$ equality to move the homography matrix row vectors on the right side of each term and change the parenthesis such that the scalar multiplication comes first:

$$\mathbf{x}^\top \mathbf{h}_1 - (u'\mathbf{x}^\top)\mathbf{h}_3 = 0$$
$$\mathbf{x}^\top \mathbf{h}_2 - (v'\mathbf{x}^\top)\mathbf{h}_3 = 0$$

Concatenate row vectors $\mathbf{h}_i$ of the homography matrix $\mathbf{H}$ into a vector $\mathbf{h} \in \mathbb{R}^9$. Then the pair of equations can be rewritten into a homogeneous system of linear equations as:

$$\underbrace{\begin{bmatrix} u & v & 1 & 0 & 0 & 0 & -u'u & -u'v & -u' \\ 0 & 0 & 0 & u & v & 1 & -v'u & -v'v & -v' \end{bmatrix}}_{\mathbf{M}_x} \mathbf{h} = 0 \tag{4.2}$$

and this system can be solved. $\mathbf{M}_x$ refers to a matrix form of the homogeneous set of equations generated by point correspondences. The final homography matrix $\mathbf{H}$ is obtained by re-arranging the elements of the vector $\mathbf{h}$. Since the homography transformation is defined up to scale, at least four correspondences are required to obtain a solution.

Now get back to hard constraints induced by vanishing points correspondences. Assume corresponding horizontal and vertical vanishing points $\mathbf{v}_h$, $\mathbf{v}'_h$, $\mathbf{v}_v$, and $\mathbf{v}'_v$. The vanishing points must satisfy the homography, therefore the corresponding pairs of VPs can be substituted into the Equation (4.2):

$$\mathbf{M}_v \mathbf{h} = 0, \tag{4.3}$$

where $\mathbf{M}_v$ is a matrix form of a homogeneous set of equations generated by vanishing points correspondences. Since each correspondence generates two equations and there are two correspondences, the matrix has four rows ($\mathbf{M}_v \in \mathbb{R}^{4\times 9}$).

The Equation (4.3) means that $\mathbf{h}$ lies in the null space of $\mathbf{M}_v$ and can be expressed as a linear combination of base vectors of its null space. The null space can be found using a Singular

Value Decomposition (SVD),

$$\mathbf{M}_v = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top,$$

and the base $\mathbf{B} \in \mathbb{R}^{9\times5}$ of the null space of the matrix $\mathbf{M}_v$ is obtained by taking five right-most vectors of the matrix $\mathbf{V}$. It is known that the homography vector $\mathbf{h}$ is a linear combination of the base vectors $\mathbf{B}$:

$$\mathbf{h} = \mathbf{B}\alpha,$$

where $\alpha \in \mathbb{R}^5$. Hence, the task is to find the coordinates of the vector α.

The rest of the corresponding image points is then used to obtain a least squared solution of the final homography matrix, constrained by the vanishing points. Assume a set of corresponding points $\mathbf{x}_i$, $\mathbf{x}'_i$ for $i = 1...N$, where N is a number of inliers (of a previously estimated affine transformation). These point correspondences are used to create the matrix $\mathbf{M}_x$ from the Equation (4.2). However, this time, linear combination of the previously obtained base vectors is sought, i.e., coordinates of the vector α:

$$\mathbf{M}_x\mathbf{h} = 0$$
$$\mathbf{M}_x\mathbf{B}\alpha = 0$$

By solving this homogeneous linear system, the vector α is estimated and the homography vector $\mathbf{h}$ (and subsequently the matrix $\mathbf{H}$) is obtained as the linear combination of the base vectors:

$$\mathbf{h} = \mathbf{B}\alpha$$

As it was done several times before, estimating a new transformation is followed by its evaluation. The evaluation may have changed the inlier arrangement and therefore this becomes an iterative process. However, the base vectors remain unchanged and only the second step of the least squares solution is repeated.

Going back to the beginning, each image should have (in an ideal scenario) one vertical VP and *at least* one horizontal VP. However, about $\sim 35\%$ of images has multiple horizontal VPs detected. All their combinations are evaluated, compared, and the best result in terms of the score is taken.

This approach can give nice results which look accurate, for example as in the Figure 4.14, but it is still a projective transform which very easily over-fits. Furthermore, this approach is conditioned by several factors: the line segments must be well detected; the object should be conspicuous and striking such that the VPs are not eclipsed by VPs generated from segments detected in the neighbourhood; the estimated VPs must be fine. If any of the conditions is unsatisfied, no homography can be estimated or it is constrained by false correspondences resulting in a poor transformation.

It was tried to use this approach also in re-ranking. Because not all image pairs have VPs detected, it must be implemented just as an optional superstructure over the basic SV. Used

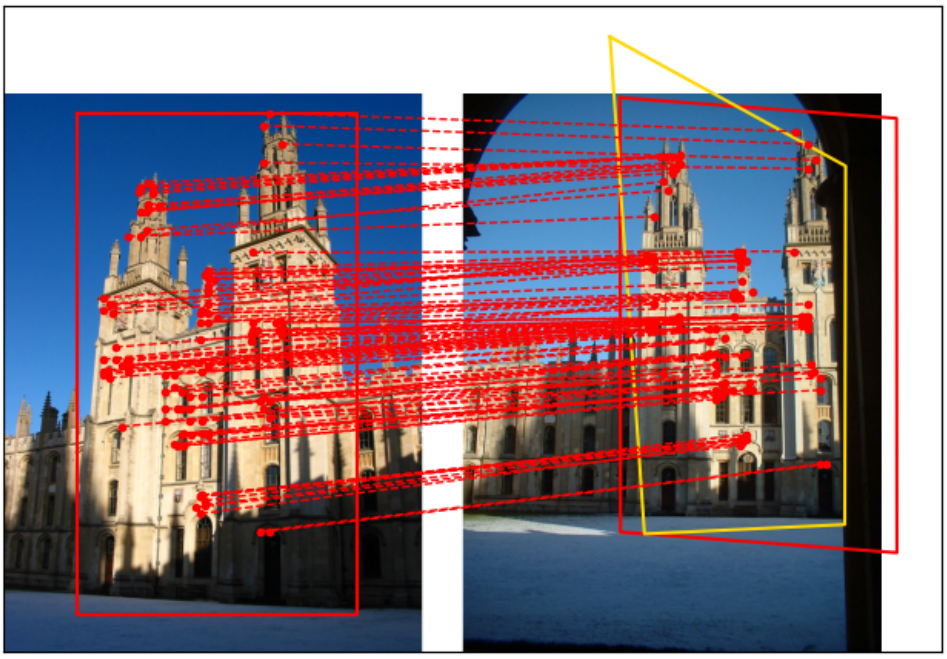


Figure 4.14: An example of a homography (yellow) constrained by vanishing points correspondences. Estimated affine transform (red) is depicted for reference.

		Easy	**Medium**	**Hard**
HOW	Advanced SV with lines	92.48	75.99	52.14
	... + constrained homographies	92.82	75.81	51.99

Table 4.6: mAP percentage for HOW features on $\mathcal{R}$Oxford - advanced SV with line hypotheses as a baseline and its version with constrained homographies.

baseline was the advanced SV with line hypotheses generated by the direct segment matching strategy with $P = 2$. If the final affine transformation has at least five inliers and both images have the compulsory vanishing points, a homography is estimated. After the model is verified and gets a score assigned, additional two points are added. The reason is that the homography satisfies the two vanishing points correspondences, but they are not in the list of tentative point correspondences. Therefore, they are not included in the scoring mechanism and must be rewarded afterwards. It is one point per VPs correspondence. The results are shown in the Table 4.6. The *Easy* evaluation protocol got a slight improvement while the performance for the *Hard* protocol dropped by a few hundredths. These images have difficult viewing conditions which often result in poor (or no) vanishing point estimates.

4.4 Normalisation in least squares solutions

Throughout the thesis, all least squares computations implicitly use normalised input data in order to return precise and stable solutions. The normalisation includes a translation and isotropic scaling such that the points have centroids in their origin and the average distance from the origin is equal to $\sqrt{2}$. This normalisation is performed on all images independently

[31].

Without normalisation, typical image points are of order $[x, y, z]^\top = [100, 100, 1]^\top$ and the linear system of equations used for a transformation estimation then contains terms like xx' of order 10^4, xz' of order 10^2 and zz' equal to 1, just to name a few examples. However, increasing the term zz' by 100 means a huge change, whereas increasing the term xx' by 100 is almost negligible. Therefore, the normalisation of all entries is essential [31].

Assume a set of input correspondences $\mathbf{x}_i$, $\mathbf{x}'_i$ for $i = 1...N$ between two images. It can be either points or even lines. The correspondences generate a system of linear equations which is solved by a least squares method in order to estimate an optimal geometric transformation from the first image into the second one.

First, two normalisation matrices are computed for each image independently as:

$$\mathbf{N} = \begin{bmatrix} s & 0 & -st_x \\ 0 & s & -st_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{N}' = \begin{bmatrix} s' & 0 & -s't_{x'} \\ 0 & s' & -s't_{y'} \\ 0 & 0 & 1 \end{bmatrix},$$

where $[t_x, t_y]^\top = \frac{1}{N}\sum_i \mathbf{x}_i$ determines the translation of the points centroid into the origin and s is a normalisation factor such that the average distance of points from the origin is equal to $\sqrt{2}$. Similarly, the very same parameters s', $t_{x'}$, and $t_{y'}$ are computed also for points $\mathbf{x}'_i$ in the second image.

When the normalisation matrices are computed, they are used to normalise the inputs $\mathbf{N}\underline{\mathbf{x}}_i$, $\mathbf{N}'\underline{\mathbf{x}}'_i$ of the least square solver. When the solver outputs a solution (transformation) $\widetilde{\mathbf{H}}$, it must be de-normalised accordingly. The final (and optimal) transformation $\mathbf{H}$ is obtained as:

$$\mathbf{H} = \mathbf{N}'^{-1}\widetilde{\mathbf{H}}\mathbf{N}$$

As it was shown in [31], such normalised transformations are essential as they are more precise and stable results.

Chapter 5

Conclusions

This thesis addresses the problem of transformation estimation in the spatial verification process. The results are evaluated between image pairs and utilised in image retrieval, primarily for re-ranking. The main contribution lies in employing lines and line segments detected in images. These are used in the transformation estimation process to provide additional information about the local geometry.

After the image retrieval was introduced and the spatial verification was put into the context, a robust estimation algorithm was described along with its advanced variants. These were applied to the problem of transformation estimation between pairs of images and implemented as a toolkit of functions, which included several adaptations of the base version of the algorithm. The correctness of the implementation was verified against a well-known set of SIFT features as well as evaluated with a new set of HOW features.

In the second part of the thesis, line segments were detected in order to be utilised in the transformation estimation process. Three detection methods were implemented and experimentally compared. One of the approaches was used and provided a foundation for further work.

Detected line segments were utilised during a generation of transformation candidates. Two matching strategies were compared, and corresponding line segments were used to provide additional soft constraints for the estimated transformations. This helps to improve the transformation flexibility, accuracy, and to compensate imprecise or missing feature properties. Lines were also used to estimate vanishing points in images, offering hard constraints for projective transformations.

The lines were shown to bring a slight improvement to the overall retrieval performance. The added value was the most noticeable for imprecise or fully missing feature data. It was able to compensate for absent feature scales with performance results comparable to the cases with scales present.

The code will be released at `https://gitlab.fel.cvut.cz/hubacda1/dt-sv`.

Bibliography

[1] A. Halawani, A. Teynor, L. Setia, G. Brunner, and H. Burkhardt, "Fundamentals and applications of image retrieval: An overview.", *Datenbank-Spektrum*, vol. 18, pp. 14–23, Jan. 2006.

[2] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations", in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, University of California Press, 1967, pp. 281–297.

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

[4] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator", *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[5] J. Shi and Tomasi, "Good features to track", in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[6] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection", in *Computer Vision – ECCV 2006*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443, ISBN: 978-3-540-33833-8.

[7] H. Kong, H. C. Akakin, and S. E. Sarma, "A generalized laplacian of gaussian filter for blob detection and its applications", *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1719–1733, 2013. DOI: 10.1109/TSMCB.2012.2228639.

[8] Y. Ono, E. Trulls, P. Fua, and K. M. Yi, *Lf-net: Learning local features from images*, 2018. arXiv: 1805.09662 [cs.CV].

[9] D. G. Lowe, "Object recognition from local scale-invariant features", in *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, pp. 1150–1157.

[10] E. Tola, V. Lepetit, and P. Fua, "A fast local descriptor for dense matching", *Proc. CVPR*, Jun. 2008. DOI: 10.1109/CVPR.2008.4587673.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[12] R. Arandjelović and A. Zisserman, "Three things everyone should know to improve object retrieval", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2911–2918. DOI: 10.1109/CVPR.2012.6248018.

[13] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)", *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008, Similarity Matching in Computer Vision and Multimedia, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2007.09.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314207001555.

[14] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features", in *Computer Vision – ECCV 2010*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792, ISBN: 978-3-642-15561-1.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf", in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[16] H. Noh, A. Araujo, J. Sim, T. Weyand, and B. Han, "Large-scale image retrieval with attentive deep local features", in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 3476–3485. DOI: 10.1109/ICCV.2017.374.

[17] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing", *Commun. ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975, ISSN: 0001-0782. DOI: 10.1145/361219.361220. [Online]. Available: https://doi.org/10.1145/361219.361220.

[18] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos", in *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, 1470–1477 vol.2. DOI: 10.1109/ICCV.2003.1238663.

[19] K. Jones, "A statistical interpretation of term specificity in retrieval", *Journal of Documentation*, vol. 60, pp. 493–502, Jan. 2004. DOI: 10.1108/00220410410560573.

[20] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search", in *Computer Vision – ECCV 2008*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 304–317, ISBN: 978-3-540-88682-2.

[21] H. Jégou, M. Douze, and C. Schmid, "On the burstiness of visual elements", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1169–1176. DOI: 10.1109/CVPR.2009.5206609.

[22] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation", in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3304–3311. DOI: 10.1109/CVPR.2010.5540039.

[23] G. Tolias, Y. Avrithis, and H. Jégou, "To aggregate or not to aggregate: Selective match kernels for image search", in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 1401–1408. DOI: 10.1109/ICCV.2013.177.

[24] G. Tolias, T. Jenicek, and O. Chum, "Learning and aggregating deep local descriptors for instance-level recognition", in *ECCV*, 2020.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[26] F. Radenović, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, "Revisiting oxford and paris: Large-scale image retrieval benchmarking", in *CVPR*, 2018.

[27] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla, "24/7 place recognition by view synthesis", in *CVPR*, 2015.

[28] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching", in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383172.

[29] P. Huber, J. Wiley, and W. InterScience, *Robust statistics*. Wiley New York, 1981.

[30] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography", *Commun. ACM*, vol. 24, no. 6, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. [Online]. Available: https://doi.org/10.1145/358669.358692.

[31] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. New York, NY, USA: Cambridge University Press, 2003, ISBN: 0521540518.

[32] O. Chum, J. Matas, and O. Drbohlav, *Robust model estimation from data contaminated by outliers*, Lecture, 2021. [Online]. Available: `https://cw.fel.cvut.cz/wiki/_media/courses/mpv/2021_ransac.pdf`.

[33] O. Chum, "Two-view geometry estimation by random sample and consensus", Ph.D. dissertation, Czech Technical University in Prague, 2005.

[34] O. Chum, J. Matas, and J. Kittler, "Locally optimized ransac", in *DAGM-Symposium*, 2003. [Online]. Available: `https://api.semanticscholar.org/CorpusID:15181392`.

[35] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory", in *Numerical Analysis*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116, ISBN: 978-3-540-35972-2.

[36] J. Matas and O. Chum, "Randomized ransac with td,d test", *Image and Vision Computing*, vol. 22, pp. 837–842, Sep. 2004. DOI: `10.1016/j.imavis.2004.02.009`.

[37] O. Chum and J. Matas, "Matching with prosac - progressive sample consensus", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 220–226 vol. 1. DOI: `10.1109/CVPR.2005.221`.

[38] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st. Berlin, Heidelberg: Springer-Verlag, 2010, ISBN: 1848829345.

[39] M. Perdoch, O. Chum, and J. Matas, "Efficient representation of local geometry for large scale object retrieval", *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9–16, 2009. [Online]. Available: `https://api.semanticscholar.org/CorpusID:120117297`.

[40] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge", *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015, ISSN: 0920-5691. DOI: `10.1007/s11263-015-0816-y`. [Online]. Available: `https://doi.org/10.1007/s11263-015-0816-y`.

[41] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network", *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956. DOI: `10.4153/CJM-1956-045-5`.

[42] O. Chum and J. Matas, "Homography estimation from correspondences of local elliptical features", in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 2012, pp. 3236–3239.

[43] O. Chum, T. Pajdla, and P. Sturm, "The geometric error for homographies", *Computer Vision and Image Understanding*, vol. 97, no. 1, pp. 86–102, Jan. 2005, ISSN: 1077-3142.

[44] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof, "From structure-from-motion point clouds to fast location recognition", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 2599–2606. DOI: `10.1109/CVPR.2009.5206587`.

[45] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113. DOI: `10.1109/CVPR.2016.445`.

[46] A. Mikulík, F. Radenović, O. Chum, and J. Matas, "Efficient image detail mining", in *Computer Vision – ACCV 2014*, Springer International Publishing, 2015, pp. 118–132, ISBN: 978-3-319-16808-1.

[47] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman, "Total recall: Automatic query expansion with a generative feature model for object retrieval", *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, 2007. [Online]. Available: `https://api.semanticscholar.org/CorpusID:570516`.

[48] O. Chum, A. Mikulík, M. Perdoch, and J. Matas, "Total recall ii: Query expansion revisited", *CVPR 2011*, pp. 889–896, 2011.

[49] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851.

[50] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform", *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119–137, 2000, ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.1999.0831. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314299908317.

[51] R. Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "Lsd: A line segment detector", *Image Processing On Line*, vol. 2, pp. 35–55, Mar. 2012. DOI: 10.5201/ipol.2012.gjmr-lsd.

[52] N. Xue, S. Bai, F. Wang, G.-S. Xia, T. Wu, and L. Zhang, "Learning attraction field representation for robust line segment detection", in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1595–1603. DOI: 10.1109/CVPR.2019.00169.

[53] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, and Y. Ma, "Learning to parse wireframes in images of man-made environments", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 626–635. DOI: 10.1109/CVPR.2018.00072.

[54] Y. Zhou, H. Qi, and Y. Ma, "End-to-end wireframe parsing", in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 962–971. DOI: 10.1109/ICCV.2019.00105.

[55] Y. Lin, S. L. Pintea, and J. C. van Gemert, "Deep hough-transform line priors", in *Computer Vision – ECCV 2020*, Cham: Springer International Publishing, 2020, pp. 323–340, ISBN: 978-3-030-58542-6.

[56] T. Pajdla, *Elements of geometry for computer vision and computer graphics*, Feb. 2021. [Online]. Available: https://cw.fel.cvut.cz/b212/_media/courses/gvg/pajdla-gvg-lecture-2021.pdf.

[57] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo", in *European Conference on Computer Vision (ECCV)*, 2016.

[58] A. Benbihi, C. Pradalier, and O. Chum, "Object-guided day-night visual localization in urban scenes", in *2022 26th International Conference on Pattern Recognition (ICPR)*, IEEE, 2022, pp. 3786–3793.