

Real-Time Localization of Planar Targets on Power-Constrained Devices

**A Journey into Stable Keypoint Detection, Fast Binary Descriptors and
Virtual Viewpoint Generation**

by

Sharat Saurabh Akhoury

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

© Sharat Saurabh Akhoury, Ottawa, Canada, 2013

Abstract

In this thesis we present a method for detecting planar targets in real-time on power-constrained, or low-powered, hand-held devices such as mobile phones. We adopt the feature recognition (also referred to as feature matching) approach and employ fast-to-compute local feature descriptors to establish point correspondences. To obtain a satisfactory localization accuracy, most local feature descriptors seek a transformation of the input intensity patch that is invariant to various geometric and photometric deformations. Generally, such transformations are computationally intensive, hence are not ideal for real-time applications on limited hardware platforms. On the other hand, descriptors which are fast to compute are typically limited in their ability to provide invariance to a vast range of deformations.

To address these shortcomings, we have developed a learning-based approach which can be applied to any local feature descriptor to increase the system's robustness to both affine and perspective deformations. The motivation behind applying a learning-based approach is to transfer as much of the computational burden (as possible) onto an offline training phase, allowing a reduction in cost during online matching. The approach comprises of identifying keypoints which remain stable under artificially induced perspective transformations, extracting the corresponding feature vectors, and finally aggregating the feature vectors of coincident keypoints to obtain the final descriptors. We strictly focus on objects which are planar, thus allowing us to synthesize images of the object in order to capture the appearance of keypoint patches under several perspectives.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Professor Robert Laganière, for giving me the opportunity to further my education at the University of Ottawa. I am grateful for his continuous support of my postgraduate studies and research, for his patience, motivation, enthusiasm, and immense knowledge. I could not have imagined having a better advisor and mentor for my M.A.Sc. study.

I would like to thank my colleagues at the VIVA Lab for providing a stimulating work environment. Thanks for your friendship, your enthusiasm, and for always having a ready ear. Valuable bonds of friendships were made and strengthened.

Last but not the least, I would like to sincerely thank God, my parents, Dr. Mukesh Akhoury and Mrs. Nutan Akhoury, and my sister, Tulika Tanu Akhoury, for their unconditional love, constant support and encouragement. I dedicate this work to them.

Contents

List of Symbols	xii
List of Variables	xiv
1 Introduction	1
1.1 Motivation	2
1.2 Problem Description	3
1.3 Overall Target Detection System Architecture	3
1.3.1 Model Generation Framework	3
1.3.2 Detecting Targets At Runtime	6
1.4 Contributions	7
1.5 Thesis Outline	8
2 Literature Review	10
2.1 Descriptor-based Methods	11
2.2 Classification-based Methods	16
2.3 Conclusion	20
3 Viewset Generation	21
3.1 Algorithm Overview	23
3.2 Deformation Process	24
3.3 Perspective Transformation	27
3.3.1 Perspective Projection	28
3.3.2 Viewport Transformation	29
3.4 Conclusion	32

4	Feature Detection	33
4.1	Keypoint Detectors	34
4.2	The FAST Keypoint Detector	37
4.3	Experimental Evaluation of Keypoint Detectors	40
4.3.1	Dataset	40
4.3.2	Evaluation Criterion	42
4.3.3	Optimal Threshold value for the FAST detector	43
4.3.4	Comparison to Other Detectors	45
4.4	Conclusion	48
5	Orientation Normalization	49
5.1	Orientation Estimation	50
5.1.1	Problem Definition	50
5.1.2	Orientation Estimators	50
5.1.3	Experimental Evaluation of Orientation Estimators	53
5.1.4	Results	56
5.2	Orientation Clustering	58
5.2.1	Graph-based Hierarchical Clustering	58
5.2.2	Probabilistic Clustering	62
5.3	Conclusion	67
6	Stable Keypoint Identification & Feature Extraction	68
6.1	Identifying Stable Keypoints	69
6.1.1	Spatial Clustering	69
6.1.2	Stability: Local vs Global	69
6.1.3	Stratified Sampling	73
6.2	Feature Extraction & Aggregation	76
6.3	Feature Stability Verification	78
6.4	Binary Feature Descriptors	80
6.4.1	HIP	80
6.4.2	BRIEF	82
6.5	Conclusion	85
7	Detecting Targets at Runtime	86
7.1	Approximate Nearest-Neighbor Matching	87
7.1.1	Motivation	87

7.1.2	Problem Statement	88
7.1.3	Data Structures	88
7.1.4	Locality Sensitive Hashing	89
7.1.5	Implementation	92
7.2	Homography Estimation	94
7.2.1	Notation	95
7.2.2	Problem Statement	95
7.2.3	Characteristics of the Homography Matrix	95
7.2.4	Direct Linear Transform	96
7.2.5	Least-Squares Solution	97
7.2.6	Random Sample Consensus	99
7.3	Handling Motion-Blur	101
7.4	Conclusion	101
8	Experimental Evaluation	102
8.1	Experimental Setup	102
8.1.1	Dataset	102
8.1.2	Evaluation Metrics	103
8.2	Parameter Tuning	104
8.2.1	Parameter Tuning: Without Orientation Normalization	107
8.2.2	Parameter Tuning: With Orientation Normalization	117
8.3	Overall Model Generation Comparison	123
8.4	Comparison to Other Descriptors	126
8.5	Evaluation on Outdoor Scenes	127
8.6	Conclusion	132
9	Conclusion	134
9.1	Future Work	136

List of Tables

8.1	Model generation framework parameters.	105
8.2	Matching thresholds experimented.	108
8.3	Resulting configuration values for the various binary descriptors for which the accuracy $\geq 70\%$ and precision $\geq 30\%$	110
8.4	Parameter configuration values for the parameter tuning experiment (with orientation normalization).	121
8.5	The optimal ϵ parameter values of the orientation clustering algorithms for each descriptor.	122
8.6	Summary of the optimal configuration for the model generation framework parameters.	133

List of Figures

1.1	Typical setup for the target detection system.	4
1.2	Flow chart of the target detection system.	5
3.1	Typical viewing conditions of a target for which localization is to be performed.	22
3.2	Clustering synthetic images to form viewsets.	23
3.3	Perspective transformation of the reference image $\hat{I}$	25
3.4	Square region fully containing an arbitrarily rotated $w_I \times h_I$ reference image $\hat{I}$.	26
3.5	Side view frustum of the planar object.	27
3.6	Perspectively transformed images.	31
4.1	Fast Hessian box filters.	36
4.2	Keypoint detection with the FAST detector using a Bresenham circle of $r = 3$. .	38
4.3	Oxford datasets.	41
4.4	Various thresholds for the FAST keypoint detector.	44
4.5	Average timing results for the various keypoint detectors.	45
4.6	Keypoint detector repeatability evaluation results.	47
5.1	Orientation sampling grid for FREAK.	53
5.2	Test images used in the orientation evaluation experiment.	53
5.3	Orientation estimation evaluation results.	57
5.4	Convergence rates of the proposed graph-based hierarchical clustering algorithm.	62
5.5	Estimating the PDF by considering the periodicity of angles.	63
5.6	Comparison of various kernels for kernel density estimation.	66
6.1	Hypothetical scenario illustrating the effect of δ_K	70
6.2	Stratified sampling illustrated with redistribution strategy.	75
6.3	Stratified sampling with orientation clustering.	77
6.4	Feature extraction: local vs global stability.	79

6.5	HIP Sampling Pattern.	81
6.6	HIP Quantizer.	82
6.7	Illustration of the HIP sampling and quantization process.	82
6.8	BRIEF test pattern for a 31×31 patch.	83
7.1	Locality Sensitive Hashing preprocessing stage: $\mathcal{H} = \{\mathbf{h}_1, \dots, \mathbf{h}_{10}\}$	92
7.2	13-bit Locality-sensitive hashing function.	94
8.1	Dataset used for the evaluation of the model generation framework.	103
8.2	Performance evaluation results of the various parameter configurations investigated.	109
8.3	Dataset analysis results.	111
8.4	Stability Threshold Experiment.	113
8.5	Local vs Global Stability Experiment.	114
8.6	Matching Threshold Experiment.	115
8.7	Pyramid Experiment.	117
8.8	Successful target detection using cBRIEF with 2-pyramid levels.	118
8.9	Time per frame (milliseconds) for the different components during matching at various pyramid levels.	119
8.10	The mean deviation from the average orientation per stable keypoint group.	120
8.11	Performance evaluation results of the orientation clustering algorithms.	122
8.12	Time per frame (milliseconds) for the different components during matching for the Graph-based Hierarchical Clustering Algorithm.	123
8.13	Evaluation results of the three variants of the model generation framework.	125
8.14	Sample detection results on the five test datasets for the s+BRIEF descriptor.	127
8.15	Comparison of our BRIEF descriptors to other approaches.	128
8.16	Evaluation results for the precision-recall curves for the graffiti dataset.	129
8.17	Matching results for cBRIEF on the graffiti dataset.	130
8.18	Matching results for cBRIEF on the boat dataset.	131

List of Symbols

$\hat{F}$	Model database	3
$\hat{I}$	Reference Image	23
I_{mn}	m^{th} image of the n^{th} viewpoint bin	24
B_x	Number of viewpoint bins along the x-axis	24
$[\Phi_{min}^x, \Phi_{max}^x]$	Viewpoint angle specification around the x-axis	24
ϕ^x	Deformation angle specification around the x-axis	24
$\Delta\Phi^x$	Step size specifying the range for one bin	24
$[S_{min}, S_{max}]$	Viewpoint scale specification	24
γ	Scaling factor	24
$\mathcal{G}$	Gaussian kernel	23
$\mathcal{W}$	Viewpoint perspective transformation process	23
$\mathcal{N}(\mu, \sigma)$	Additive Gaussian noise with mean μ and standard deviation σ .	23
$\mathcal{P}$	Perspective projection matrix	28
$\mathcal{D}$	Deformation matrix	25
$\mathcal{R}$	Rotation matrix	25
$\mathcal{T}$	Translation vector	25
$p = (x, y)$	2D point in an image	34
$\mathbb{K}$	Keypoint response function	34
$\mathcal{I}_x$	Derivative of image I w.r.t. x axis	34
$\delta_{\mathcal{E}}$	$\delta_{\mathcal{E}}$ -repeatability criterion	42
$\mathcal{H}$	Homography matrix	42
$\mathcal{R}$	Image intensity patch	50
$\mathcal{C}$	Clusters	59
Θ	Set of orientation values	59
$f(\cdot)$	Probability distribution function	62
$K(\cdot)$	Kernel function	62
$\hat{p}_r$	Point on reference image	69

$\mathbb{G}$	Sets of clustered coincident keypoints	69
Ω	Stability vector	78
$\mathcal{Z}$	Feature matching threshold	86
$\mathcal{H}$	LSH function family	90
$\mathbb{H}^d$	d -dimensional Hamming space	90
$D_{\mathbb{H}}$	Hamming distance	90

List of Variables

M	Number of virtual views per viewpoint bin	24
N	Number of viewpoint bins	24
$\hat{w}_I$	Reference image width	26
$\hat{h}_I$	Reference image height	26
ψ	Aspect ratio	28
α	Vertical field of view	28
w_I	Virtual image width	29
h_I	Virtual image height	29
δ_t	FAST keypoint threshold	36
ϵ	Maximum deviation from the medoid of a cluster to all its nodes	59
ρ	Minimum probability threshold	64
N_c	Desired number of clusters	64
G_r	Group of keypoints mapping back to the same reference point $\hat{p}_r$	69
$\mathcal{K}_{mn}$	Keypoints detected from virtual image m of viewpoint bin n . . .	69
$\delta_{\mathcal{K}}$	Minimum spatial distance to neighboring keypoint	69
Q	Number of stable points per viewpoint bin	71
W	Stratified sampling rate	74

Chapter 1

Introduction

Establishing correspondences between similar real world points appearing in variously deformed images is an essential task and the first step for many computer vision applications. Typical applications include, but are not limited to, object detection, image matching, image retrieval, image registration, panorama synthesis (image stitching), visual odometry and marker-less Augmented Reality (AR). In recent years mobile computing devices, smartphones in particular, have experienced vast growth in miniaturization as well as performance. Such devices are typically inexpensive, and are therefore attractive mediums for vision based applications especially for Augmented Reality. However, they are still confined in both the computational aspects (low throughput, little/none floating-point support) and memory bandwidth (limited storage, slow main memory access, small caches).

The point correspondence problem has been well researched in the computer vision community, with the *feature recognition* approach arising as the most commonly accepted solution. The feature recognition method comprises of two fundamental stages. The first stage involves detecting features in an image containing the object of interest. There are three types of features that are usually found, *point-based*, *edge-based*, and *region-based*. The second stage comprises of feature extraction, whereby a feature vector is computed to describe the image patch encompassing the feature. The process of computing the feature vector is formally known as feature extraction whereas the actual algorithm for computing the feature is known as the descriptor. Feature descriptors can be categorized into two broad classes [39]: *descriptor-based* and *classification-based*.

Descriptor-based approaches address the correspondence problem by computing affine-invariant descriptors of local image patches which are then compared across images. The descriptors usually involve fine scale selection, rotation correction, and intensity normalization.

An advantage of the descriptor-based approach is that a training stage is not required. However, in its current form, it remains computationally intensive and is not well-suited for power-constrained devices.

Classification-based approaches, on the contrary, require an offline training stage to extract feature sets from artificially synthesized viewpoints of the target object, which are then used to train a classifier to recognize the features. The viewpoints are synthesized to model the deformation space for which invariance is to be achieved against. The feature descriptors themselves are reasonably simple and fast to compute, thus allowing for efficient runtime performance. The premise behind the classification-based approach for point correspondences is based on transferring as much of the computational burden as possible onto the training phase in order to reduce the cost of online matching.

By modeling the limitations of mobile computing devices as design constraints, this thesis addresses the problem of point correspondence for detecting planar targets on such potential devices. Therefore, in order to satisfy the real-time requirement, we adopt the classification-based approach.

1.1 Motivation

With the rapidly growing popularity of smartphones in recent times, object detection is gaining more attention and is increasingly becoming more demanding than ever before. However, as mentioned previously, the limited processing capabilities in such devices still pose a set of algorithmic challenges.

The last few decades has given rise to a considerable amount of research in both stages of the feature recognition approach, with modern day algorithms demonstrating improved robustness and reduced processing times. High-end feature detectors [35, 1, 25] have been shown to not only repeatably localize features in real-time, but perform exceptionally well on mobile devices [43, 40, 39]. However, the challenge still remains for the feature extraction process. While fast feature descriptors are generally not robust enough in matching, state-of-the-art descriptor-based approaches [23, 6] are simply not computationally efficient. On the other hand, although classification-based approaches [18, 30, 40, 39] are well suited for achieving fast and robust real-time matching, they still face drawbacks in terms of high training times and/or memory consumption. Hence, there exists a trade-off between robustness, efficiency, memory consumption, and training time.

In this thesis, we focus on the feature extraction stage and present a classification-based approach to improve the robustness and efficiency of fast-to-compute feature descriptors which

are relatively weak, i.e., sensitive to scale and rotation.

1.2 Problem Description

We would like to detect planar targets in real-time on a hand-held device which is characterized by a limited budget hardware platform equipped with a camera. The hardware is limited in the sense that it has significantly lower memory and processing capabilities as compared to a standard desktop machine. We make the target localization problem more interesting (and challenging) by allowing the hand-held device to move around freely in space without any constraints on the extent of the motion.

A typical setup illustrating the problem is presented in Fig. 1.1. A user captures an image of the desired target and sends it to a server (desktop machine). The server then generates a set of features which uniquely describes the target and stores them in a container referred to as the *model database*. Thereafter, the user downloads the model database onto his/her device and proceeds to detect the target from a live camera feed.

Additional requirements of the target localization system are as follows.

1. The time to learn a target should be kept to a minimum, preferably less than a few minutes.
2. The resulting model databases should be reasonably small so that it is memory-efficient.
3. A computationally inexpensive method is required to perform real-time matching on the limited resource hand-held device.

1.3 Overall Target Detection System Architecture

Our classification-based target detection system comprises of two distinct components, namely, *model generation framework* and *runtime target detection*. A flowchart summarizing the overall target detection system architecture is presented in Fig. 1.2. Without getting too technical, we provide a high-level description of each component below.

1.3.1 Model Generation Framework

Constructing a model database, $\hat{F}$, comprises of three primary steps: *viewset generation*, *stable keypoint identification*, and *feature extraction*. In addition to these steps, we propose a novel

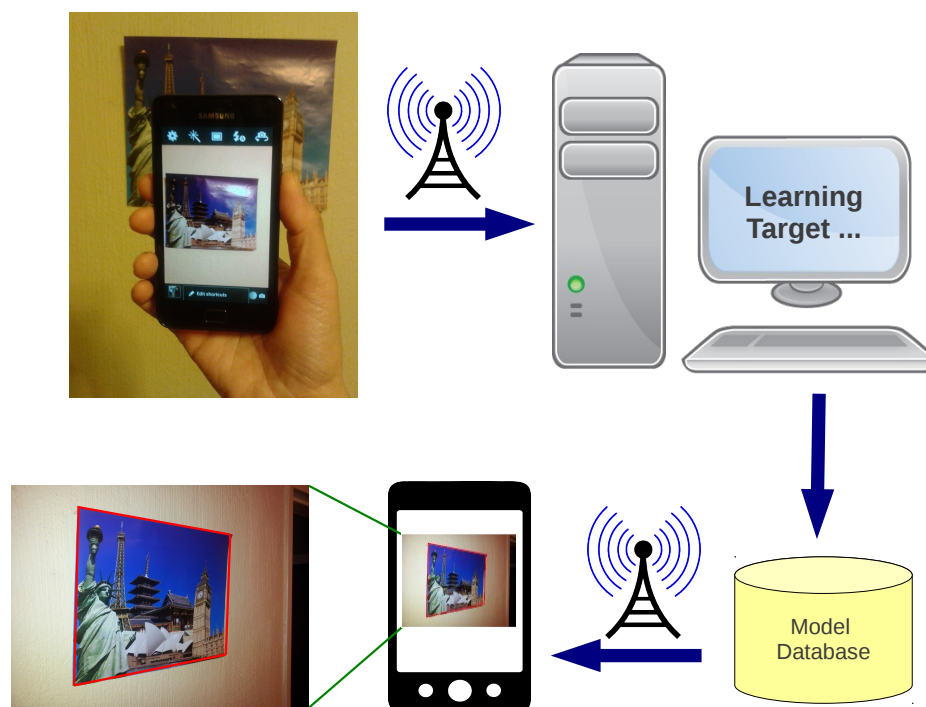


Figure 1.1: Typical setup for the target detection system.

concept for verifying the stability of the extracted features in order to generate strictly reliable database models. These steps are performed in sequential order.

Viewset Generation

In order to achieve invariance to different types of deformations, classification-based descriptors extract sets of features from geometrically transformed images (of the target image) which model the desired deformation space. The process of synthesizing these artificial viewpoint images is known as viewset generation. The viewsets are characterized by groups of virtual images which are synthesized from subspaces of the deformation space. We commonly use the words viewpoint bin and viewset interchangeably throughout this thesis. Instead of applying the conventional affine transformation (which is unable to model the inherent perspective distortions found in real-world images) to generate the viewsets, we apply a perspective projective transformation. This allows us to sample the deformation space in a more representative, controlled way requiring less images per viewset to model the desired deformation space. Hence, we are able to reduce the training time by a significant amount.

Chapter 3 describes the viewset generation process.

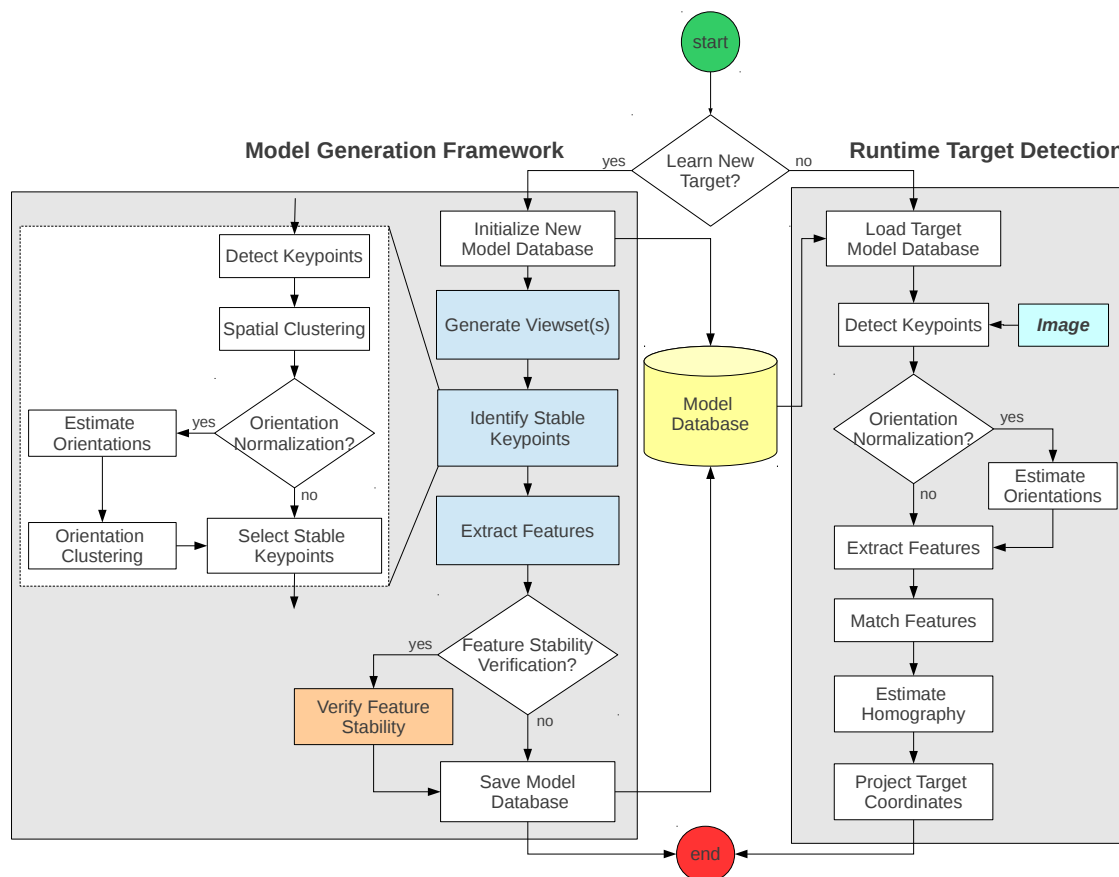


Figure 1.2: Flow chart of the target detection system.

Stable Keypoint Identification

For a given viewset, keypoints are detected from each viewpoint image using a reliable keypoint detector. We review various kinds of keypoint detectors in Chapter 4.

Before extracting the descriptors for the keypoints, we first determine the most stable keypoints by measuring the repeatability of the detected keypoints. This is accomplished by performing spatial clustering. If orientation normalization is to be performed, we then proceed to estimate the local orientation of each keypoint. Orientation normalization is the process of normalizing a feature with regard to its estimated local orientation. By clustering the orientation values, we are able to further refine the spatial groups to ensure that each group contains keypoints with similar spatial and orientation values. The orientation normalization procedure is described in Chapter 5.

The objective of the orientation clustering process is to limit the appearance variation of the region encompassing the groups of keypoints, thus allowing for better descriptors to be ex-

tracted. This stage can be omitted if a highly accurate orientation estimator is applied since, for such an estimator, the orientations should be relatively consistent irrespective of how deformed the images are. However, an accurate orientation estimator is typically computationally intensive and is not well suited for real-time matching on low powered devices. Thus, we transfer the burden of accurate orientation estimation onto the training stage by permitting the use of cheap orientation estimators.

Finally, the stable keypoints are then ranked according to their corresponding cluster size. We adopt a stratified sampling technique to choose the best stable points across the entire image.

Section 6.1 details the stable keypoint identification process.

Feature Extraction

After the stable keypoints have been identified, we then extract features for all the keypoints associated with each stable keypoint group. We refer to the individual features as partial features. The final feature which describes the stable point is obtained by aggregating all the corresponding partial features. For most binary descriptors, aggregation can be performed by a simple majority vote on each bit of the descriptor set. For other binary descriptors which make use of histograms, instead of a majority vote, a cut-off threshold can be applied to identify rarely occupied bins. Floating-point descriptors can be aggregated by computing the average of each feature dimension.

The feature extraction process is explained in Section 6.2.

Feature Stability Verification

Being able to identify keypoints which are detectable across various deformations does not necessarily imply that the associated features are reliable. Since we aggregate multiple partial features to obtain the final feature, a situation can arise where the viewsets contain too much variation in appearances, resulting in an unreliable, noisy feature. Therefore, we propose to verify the stability of the aggregated features by measuring their entropy.

Section 6.3 describes the feature stability verification technique.

1.3.2 Detecting Targets At Runtime

Runtime target detection works as follows. We first load the model database of the target we wish to detect into memory. Thereafter, the keypoints are detected from a given query image

and if orientation normalization is to be performed, the local orientation for each keypoint is estimated. The runtime features are then extracted from the keypoints and are then matched to those in the database model. From the set of point correspondences found, we then estimate the homography which delineates the projective relation between the query and target reference image. To finally locate the target in the query image, we then apply the estimated homography to project the corner locations of the target from the reference image onto the query image.

We now briefly describe the matching strategy adopted. After generating the database model, a classifier can be trained to learn the feature space of the target. This then allows us to use the classifier to establish correspondences directly from the runtime query features. However, this comes at the added cost of having a two stage training process. Thus, to avoid a secondary training stage and to achieve real-time matching, we adopt an approximate nearest-neighbor matching strategy, namely *Locality Sensitive Hashing* [12].

The runtime target detection scheme is fully detailed in Chapter 7.

1.4 Contributions

We have developed an offline training algorithm (i.e. model generation framework) which can be applied to any weak feature descriptor to boost the system’s localization accuracy. To complement the model generation framework, we have also developed an efficient runtime matching system which boasts of matching rates over 60Hz (on a 2.67 GHz processor). We have demonstrated the effectiveness of our system by implementing three extensions to the BRIEF descriptor [8], and have shown them to yield comparable robustness to state-of-the-art descriptors like SIFT [23] and SURF [6].

Our training framework is able to effectively generate models for targets in less than a minute. This is perhaps one of the most significant contributions of our research, as current methods require at least 20 minutes [39] to achieve respectable performances. Other contributory components arising from our research include:

- two novel orientation clustering algorithms for orientation normalization;
- a novel concept for verifying the stability of features in order to generate reliable model databases; and
- a comprehensive empirical investigation into the optimal configuration of the model generation framework parameters.

Lastly, our research work has been selected for an oral presentation in the International Conference on Image Analysis and Processing (ICIAP) [2].

1.5 Thesis Outline

In Chapter 2 we review the literature directly related to our work. We begin by reviewing the descriptor-based and classification-based feature recognition approaches.

In Chapter 3 we explain the the viewset generation process. More specifically, we derive the perspective transformation process which is used to synthesize the viewpoint images.

In Chapter 4 we discuss various keypoint detection algorithms. We then describe the FAST keypoint detector [35] in some depth. Later on, we then investigate various parameter values for the FAST keypoint detector. We then conclude the chapter by comparing various keypoint detectors using a publicly available dataset.

Chapter 5 details the orientation normalization process. We first review well known orientation assignment techniques and later compare them in order to determine the most accurate and efficient approach. We then describe two orientation clustering algorithms, *Graph-Based Hierarchical Clustering Algorithm* and *Probabilistic Clustering Algorithm*, which are used for identifying stable keypoints if orientation normalization is to be applied.

In Chapter 6 we describe the process for identifying stable keypoints. We begin by detailing our spatial clustering algorithm which is used to obtain initial stability scores. We then describe two methods for measuring stability with regard to the scope of the keypoints. Thereafter, we discuss our stratified sampling strategy to assist in selecting a representative set of keypoints throughout the entire target. We then explain the feature extraction and aggregation process, and later introduce our novel feature stability verification component. Also included in this chapter is a section dedicated towards binary feature descriptors, in particular HIP [40] and BRIEF [8]. It is in this section that we propose three variants of the BRIEF descriptor using our model generation framework.

Chapter 7 presents the runtime target detection approach. We first motivate the use of an approximate nearest-neighbor matching strategy. We then describe the locality sensitive hashing (LSH) technique [12] for matching features and then derive the homography estimation process. We also include a short description on the Random Sample Consensus (RANSAC) algorithm [10], which is used to filter outliers from tentative correspondences. We then conclude the chapter by presenting a simple solution to handling motion blurred runtime images.

The experimental evaluation results are documented in Chapter 8. We first describe the experimental setup comprising of the datasets and evaluation metrics used to evaluate the tar-

get detection system. The results of an extensive parameter tuning experiment aimed at determining the optimal parameter configuration for the model generation framework are then presented. We then provide an overall evaluation of three possible variants of our model generation framework. This is then followed by the comparison results of our BRIEF descriptors to other well known descriptor-based approaches including SIFT [23] and SURF [6]. The chapter concludes with an assessment of our BRIEF descriptors on two external datasets.

Finally, in Chapter 9 we summarize the results of our target detection system and indicate possible future work.

Chapter 2

Literature Review

As mentioned previously, the keypoint recognition problem consists of two main components. The first component involves detecting salient features (such as points or regions) which characterize the object. The second component then attempts to extract a unique, distinguishable signature (aka feature descriptor) for each detected feature. It is this signature that will be used to match corresponding features between a query and reference image of the object of interest. The simplest descriptor comprises of a vector of raw image intensities around the local neighborhood of the feature. Such a descriptor requires computing the cross-correlation between two descriptors to obtain a similarity measurement of the two features. However, cross-correlation on the image intensities is highly sensitive to changes such as affine or 3D viewpoint change or non-rigid deformations. Furthermore, the high dimensionality of such a description results in a high computational complexity for recognition.

Many notable algorithms have been proposed to detect features in real-time [35, 1, 25]. However, the same cannot be said about the latter component which still poses many challenges as a compromise must be made on the robustness of the descriptor in order to achieve real-time matching performances on power-constrained devices. As shall be seen, prior works that have been done on extracting local descriptors still face limitations in terms of computation time and memory usage. These limitations impede local descriptor-based applications from running on devices equipped with low-end processors such as mobile phones. Hence, our focus in this thesis is on the feature description component of the keypoint recognition problem.

Many different techniques for describing local image regions have been developed. In general, feature descriptors can be categorized into two broad classes, namely, *descriptor-based* and *classification-based* [39]. Descriptor-based approaches rely on designing the feature to be as robust as possible against specific types of deformations. They typically require fine scale

selection, rotation correction, and intensity normalization. Classification-based approaches, by contrast, require an offline training stage to learn feature sets from artificially warped images (of the source object) allowing us to achieve insensitivity to specific kinds of deformations. These approaches are designed to transfer much of the computational burden onto the training phase in order to reduce the cost of online matching while increasing its robustness.

We review the most significant contributions from both categories to gain an insight into their working as well as to highlight their limitations. A review of feature detectors is presented in Chapter 4.

2.1 Descriptor-based Methods

Most descriptor-based approaches focus on scale and in-plane rotation invariant detectors and descriptors. These seem to offer a good compromise between feature complexity and robustness to commonly occurring deformations. Skew, anisotropic scaling, and perspective effects are assumed to be second-order effects, that are covered to some degree by the overall robustness of the descriptor.

In general, to be classified as a descriptor-based method, the feature descriptor should possess the following characteristics:

- Must be highly distinctive, i.e. low probability of a mismatch.
- Should be easy to extract.
- A good local feature should be invariant (tolerant) to: image noise, changes in illumination, uniform scaling, rotation, and (minor) changes in viewing direction.

Non-parametric Local Transforms

One of the early descriptor-based approaches which led to the culmination of distribution-based descriptors was developed by Zabih and Woodfill [44]. They apply a non-parametric local transform to the query and reference image, and match features by cross correlation. The descriptor is based on histograms of relative ordering of local intensity values, and not on the intensity values themselves. The binary relations between intensities of several neighboring pixels are mapped by binary strings and an empirical distribution of all possible combinations is represented by histograms. For their approach to succeed, the transform must result in significant local variation within a given image. Furthermore, it must also yield similar results near corresponding points between the two images.

SIFT

The Scale Invariant Feature Transform (SIFT) descriptor [22, 23] is perhaps one of the most well-known, well-studied feature descriptor in the computer vision community. SIFT has become the de facto standard to which most researchers benchmark their algorithms against. It comprises of a 3D histogram of gradient locations and orientations from a local image region centered on the keypoint. The spatial locations are quantized into a 4×4 grid using the Cartesian coordinates, whereas the gradient angles are quantized into one of 8 orientation bins, thus resulting in a 128-dimensional descriptor. The final descriptor is obtained by concatenating the orientation histograms over all locations. To achieve invariance to illumination, the descriptor is normalized to unit length and thresholded to remove the influence of large gradient magnitudes. The descriptor is again renormalized to unit length.

PCA-SIFT

The PCA-SIFT [17] descriptor, as its name suggests, applies principal component analysis (PCA) to the SIFT descriptor to obtain a more compact representation. The resultant feature descriptor is significantly smaller than the standard SIFT feature vector. The PCA-SIFT approach requires a once-off offline processing stage to pre-compute an eigenspace of gradient images of local patches. At runtime, a local gradient image is computed from an orientation normalized patch, centered at the keypoint. The resultant gradient image vector is then projected onto the eigenspace to obtain a compact feature vector. Ke and Sukthankar [17] empirically established that the first 20 principal components are sufficient to obtain comparable detection rates to SIFT at a much reduced matching cost. The standard SIFT representation employs 128-element vectors; whereas PCA-SIFT results in significant space benefits.

GLOH

The Gradient Location-Oriented Histogram (GLOH) descriptor [26] is another extension to SIFT designed to increase its robustness and distinctiveness. Instead of quantizing the spatial locations in the conventional Cartesian plane, GLOH makes use of a log-polar location grid. There are 3 bins that model the radial direction and 8 for the angular direction. However the central radial bin is not partitioned angularly, resulting in a total of 17 location bins. The gradient orientations are quantized into 16 bins (as compared to 8 for SIFT), thus yielding a 272 dimensional descriptor. To reduce the size of the descriptor, principal component analysis is applied. To this end, a once-off offline training stage is required to estimate the covariance matrix for PCA. This is done using a large sample of image patches collected from various

images. Mikolajczyk and Schmid [26] retain the 128 largest eigenvectors to build the final descriptor. The GLOH descriptor has been shown to be more robust than SIFT, however, it is also more computationally involved.

MOPS

The Multi-scale Oriented Patches (MOPS) descriptor [7] is based on the Haar wavelet transform which is applied to sparsely sampled patches. The descriptor is computed as follows. A sparse 8×8 patch of pixels is first sampled around the sub-pixel location of the keypoint, using a spacing of 5 pixels between samples. Note that prior to sampling, the local patch is rotated to the dominant local orientation to provide some invariance against in-plane rotations. Thereafter, the sampled pixels are normalized so that the mean is 0 and the standard deviation is 1. This makes the features robust to affine changes in intensity (bias and gain). Finally, the Haar wavelet transform is applied to the 64 normalized samples resulting in a 64-dimensional descriptor vector containing the wavelet coefficients. Due to the orthogonality property of Haar wavelets, Euclidean distances between features are preserved under this transformation.

SURF

In contrast to SIFT, which computes the distribution of gradient vectors, the Speeded-Up Robust Features (SURF) descriptor [6] describes the distribution of the intensity content within the keypoint neighborhood using first order Haar wavelet responses. The SURF descriptor is computed as follows. The first step comprises of extracting a square patch centered around the keypoint and rotated along the keypoint's local orientation. This patch is then partitioned into smaller 4×4 square sub-regions in order to preserve important spatial information. From each sub-region, the Haar wavelet responses are computed at 5×5 regularly spaced sample points and weighted according to a Gaussian function centered at the keypoint. From the wavelet responses, the following feature vector is then computed:

$$\left[\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y| \right], \quad (2.1)$$

where d_x and d_y are the filter responses to the Haar wavelets along the x and y directions respectively. The SURF descriptor is finally obtained by concatenating the 16 vectors computed for every sub-region and by normalizing the corresponding 64-dimensional vector so that its Euclidean norm is unitary, thus making the descriptor invariant to affine contrast changes.

CS-LBP

The Center-Symmetric Local Binary Pattern (CS-LBP) descriptor [15] builds on ideas from SIFT (histogram of weighted gradients) and LBP (local binary patterns) [29]. Prior to computing the feature descriptor, the local image regions are first pre-processed. An adaptive Wiener filter is applied to remove noise artifacts while preserving high frequency content. Thereafter, the contrast is enhanced so that 1% of the data is saturated at the low and high intensities of the region. The feature vector is constructed similarly to SIFT, however, the descriptor algorithm is different. The gradient features used by SIFT are replaced by simple features extracted by a center symmetric local binary pattern operator similar to the LBP operator. Furthermore, a uniform weighting scheme is applied instead of the Gaussian weighting technique adopted in SIFT. Not only is the CS-LBP descriptor well suited for planar targets, it also boasts of high-levels of invariance against changes in illumination. However, more importantly, the primary contribution of the CS-LBP descriptor comes from the computational simplicity of the algorithm itself.

ASIFT

The Affine-SIFT (ASIFT) [28] was designed to extend the SIFT descriptor to be completely affine invariant. The robust SIFT descriptor is primarily invariant to translation, rotation, and scale deformations. However, in its current form, it cannot handle other affine deformations. Affine-SIFT operates by simulating a set of sample views of the target image by varying the two camera axis orientation parameters, namely the latitude and the longitude angles. Then it applies the SIFT descriptor to all the rendered images. Thus, ASIFT covers effectively all parameters of the affine transform. Although it would be preferable to compute the descriptors from the sample views offline, it can be done online as well depending on the desired level of robustness required. Morel and Yu [28] also provide a mathematical proof showing ASIFT to be fully affine invariant. It is needless to say that the ASIFT descriptor is by far one of the most robust descriptor algorithms in the literature.

BRIEF

BRIEF (Binary Robust Independent Elementary Features) [8] is a fast-to-compute feature descriptor which makes use of simple binary tests based on intensity differences between pixels in a smoothed image patch. A significant advantage of binary descriptors is that they can be efficiently compared using the Hamming distance (bitwise XOR followed by a bit

count), in contrast to floating point descriptors (which are compared using the more computationally intensive Euclidean distance). The BRIEF descriptor comprises of an n -bit string ($n = 128, 256, 512$) description of an image patch constructed from a set of binary intensity tests. Calonder *et al.* [8] have demonstrated BRIEF to be highly discriminative and invariant to changes in illumination, blur as well as partial perspective distortions. However, it remains sensitive to both scale and rotation deformations.

ORB

ORB (Oriented FAST and Rotated BRIEF) [36] overcomes the lack of rotation invariance of BRIEF by estimating the local orientation between the feature location and the centroid of its neighborhood. Furthermore, the sampling pattern employed in ORB uses 256 pairwise intensity comparisons, but in contrast to BRIEF, is constructed via machine learning, maximizing the descriptors variance and minimizing the correlation under various orientation changes.

BRISK

The Binary Robust Invariant Scalable Keypoints (BRISK) descriptor [20] was recently developed to provide a binary descriptor that is both scale and rotation invariant. For feature description, instead of using random or learned patterns (as in BRIEF and ORB), BRISK applies a symmetric pattern with the emphasis on maximizing descriptiveness. The pattern comprises of sample points which are positioned in concentric circles surrounding the keypoint, with each sample point representing a Gaussian blurring of its surrounding neighbors. The standard deviation of this blurring is increased with the distance from the keypoint. After estimating the scale and dominant orientation of the keypoint, the sampling pattern is then scaled and rotated accordingly. Thereafter, the descriptor is computed using 512 short-distance sample point comparisons representing the local gradients within the keypoint neighborhood. Overall, BRISK requires significantly more computation and slightly more storage space than either BRIEF or ORB.

CARD

Ambai and Yoshida [4] have recently developed a novel, fast descriptor which they refer to as Compact And Real-time Descriptor (CARD). The descriptor is computed using an efficient algorithm based on lookup tables for extracting histograms of oriented gradients, which results in approximately significantly faster computation time per descriptor than that of SIFT. The algorithm can handle arbitrary layouts of bins, such as Cartesian grid binning (SIFT [23]) or

log-polar binning (GLOH [26]), thus yielding sufficient discrimination power. Furthermore, the authors also apply a learning-based sparse hashing technique to convert the extracted descriptors to short (compact) binary vectors. This allows for rapid matching with the Hamming distance instead of the computationally expensive Euclidean distance metric. The conversion is achieved by multiplying a highly sparse integer weight matrix by the descriptors and aggregating signs of their multiplications. The weight matrix is determined using a machine learning technique in order to preserve the dissimilarity of two descriptors even in Hamming space.

FREAK

The Fast Retina Keypoint (FREAK) descriptor [3] is inspired by the human visual system, more precisely the retina. In many aspects, FREAK is similar to BRISK, with notable differences being in the way the local orientations are computed as well as the sampling pattern used for the brightness comparison tests. FREAK applies a retinal sampling grid which is also circular however, it has a higher density of points near the center. As with BRIEF, ORB and BRISK, each sample point needs to be smoothed to decrease the sensitivity to noise. BRIEF and ORB use the same kernel for all points in the neighborhood. On the other hand, BRISK uses different kernels size for every sample point. FREAK follows suit of BRISK, however, the difference with BRISK is the exponential change in size and the overlapping receptive fields. Alahi *et al.* [3] argue that the overlapping fields add redundancy which increases the discriminative power. Given the large number of receptive fields, there are potentially thousands of sampling pairs which can be used to construct the descriptor. To select the best possible pairs, FREAK applies a machine learning algorithm identical to ORB to select the best 512 sampling pairs.

2.2 Classification-based Methods

In contrast to descriptor-based methods which are characterized by building affine-invariant feature descriptors (primarily dealing with rotation, scale and translation), most classification-based approaches train classifiers to identify keypoints from a target object using fast-to-compute, relatively weak descriptors. These descriptors are weak in the sense that they are invariant to a limited range of deformations only. In order to increase the invariance to different types of deformations, classification-based methods train the classifiers on sets of features extracted from geometrically transformed images (of the target) which model the desired deformation space. The major drawback of this approach is the explicit requirement of a training

step. However, using classifiers has a few advantages over descriptor-based approaches [11]:

- For recognition, a keypoint is simply “fed” to the classifier, which is faster than computing a complex descriptor (such as SIFT or SURF).
- The classifiers can be trained to be robust against exactly the conditions that are expected to occur.
- During the training stage, the algorithm can identify the keypoints that are the most stable under the expected viewing conditions.

Recognizing Shapes with Randomized Trees

Amit and Geman [5] developed a new approach to identifying shapes based on training a random forest classifier on a very large set of binary features of the image. Their proposed approach later served as a basis for other classification-based methods such as [19, 18, 30]. The binary features correspond to spatial arrangements of several local topographic codes which are in themselves too primitive and non-distinctive to be informative about shape. However, by selecting informative features, Amit and Geman [5] trained multiple random decision tree classifiers on them using inductive learning to model the feature space of the shape. In essence, each tree provides an approximation to the full posterior where the selected features depend on the branch which is traversed. The leaf nodes are assigned a label according to estimates of the corresponding posterior distribution over the shape classes. At runtime, a query image is classified by sending it down every tree and aggregating the resulting distributions. This method was applied to classifying handwritten digits and synthetic linear and nonlinear deformations of three hundred Latex symbols. State-of-the-art error rates were reported on the NIST database of digits [5].

Keypoint Matching using PCA

Being inspired from [5], Lepetit *et al.* [19] revolutionized the keypoint matching problem for object recognition by casting it as a classification problem. Given an image of a target object, a feature set $\hat{F}$ is generated by combining the statistics of the set of warped patches, referred to as a viewset, around each keypoint. Each detected keypoint denotes a distinct class which in turn corresponds to the set of all possible views of such a point. Invariance to various deformations is achieved by synthesizing a sufficient amount of viewsets. Principal component analysis was applied to extract a compact description for all the viewsets associated with each keypoint. At runtime, a nearest-neighbour (NN) classifier is employed to identify keypoints which are in $\hat{F}$.

Bayesian One-Shot Learning of Object Categories

Fei-Fei *et al.* [9] developed a novel approach to learn object categories using a single image, or a handful of images, of an object (hence the term one-shot) and by representing them as probabilistic models. They detect the most salient regions of the object by clustering over location and scale to give a reasonable number of features per image, each with an associated scale. Once the regions are identified, they are extracted from the image and scaled to an 11×11 pixel patch. Each patch exists in a 121-dimensional space. The dimensionality is then reduced by applying principle component analysis. A fixed PCA basis, pre-calculated from a separate dataset, is used for this task. The first 10 principal components from each patch are then sampled to form the feature vector. A Bayesian framework is then applied to build a probabilistic model of the extracted features for a given object. As more images of the object are successfully recognized, the authors suggest adapting the probabilistic models accordingly to increase the system's robustness. Fei-Fei *et al.* [9] compared the Maximum Likelihood (ML) and Maximum A Posteriori (MAP) methods to their Bayesian approach and found that on a database of more than 100 categories, the Bayesian approach produces informative models when the number of training examples is too small for other methods to operate successfully.

Keypoint Matching using Randomized Trees

Lepetit and Fua [18] further extended their initial work in [19] by replacing the feature vector representation and substituting the nearest-neighbor classifier for a random forest classifier similar to [5]. The set of all possible appearances of an image patch i is once again modeled as a class C_i . While creating a “forest” of randomized decision trees, each node n gets assigned two randomly chosen pixel displacements q_{n_1}, q_{n_2} . For a candidate keypoint p_k , the node checks if the pixel at $p_k + q_{n_1}$ is darker than the pixel at $p_k + q_{n_2}$. If it is the case, the keypoint is referred to the left child, otherwise to the right child, until it finally reaches a leaf. During training, a set of artificial views is synthesized for each keypoint (“class”), representing the changes that the classifier will be invariant to. At runtime, each query keypoint is “dropped down” each of the trees, eventually reaching one of the leaves. Afterwards, a posterior probability for each class C_i and leaf l_j can be extracted, denoting the probability that a sample of C_i reaches leaf l_j , and, with Bayes' theorem, the probability that a sample that reaches leaf l_j belongs to class C_i [18].

Keypoint Matching using Naive Bayesian Classifier

The random forest classifier in [18] models the full joint probabilities for each possible combination of node test results, thus branching out to 2^n leaves for n node tests. To obtain a compact representation, a naive Bayesian approach can be adopted and the node tests can be assumed to be conditionally independent of each other. Under this assumption, the “tree” structure then collapses into a single “string” of nodes. However, to avoid making such a strong assumption, Özuysal *et al.* [30] propose applying a semi-naive Bayesian approach and model only some of the dependencies by clustering the node tests into groups. Within each group, the joint probabilities are fully modeled, whereas each group is assumed to be conditionally independent from the other groups. Structure-wise, the decision tree classifier now resembles a “fern”, hence this approach is also famously known as the “Ferns” method. Özuysal *et al.* [30] use the same training and classification procedure as [18]. A drawback to this approach is that the classifier needs to represent complicated joint distributions for each feature, hence requiring a large amount of memory.

Histogrammed Intensity Patches

Inspired by [18, 30], Taylor *et al.* [40] present another training-based keypoint recognition approach, which generates a rich feature set by computing coarse histograms of the intensities of selected pixels around a keypoint for each viewset. They refer to their method as Histogrammed Intensity Patches (HIP). Although the training process is quite similar to that used in [18, 30], a slight modification is made pertaining to the viewset generation process. For each keypoint, instead of clustering all the viewsets into a single class, Taylor *et al.* [40] proposed organizing them into separate groups (referred to as viewpoint bins) such that each group is characterized by a similar set of deformations. This is done to limit the appearance variation in the viewsets in order to extract more representative features. As with [18], orientation normalization was later added in [39] to reduce the number of features initially required in [40], allowing recognition of a planar object across a wider range of viewpoints. Since a cheap orientation estimator was used in [39], the authors suggest clustering keypoints using both the spatial and orientation information. This further restricts the appearance variation allowing for better features to be extracted. Although fairly good matching robustness can be achieved, the training method in [40, 39] is computationally expensive as the authors acknowledge that it takes at least 20 minutes to generate a model.

Boosting Binary Keypoint Descriptors

Trzcinski *et al.* [41] have recently developed a novel supervised learning framework that finds a low-dimensional, yet highly discriminative binary descriptor. They refer to their approach as BinBoost. Similar to the AdaBoost classifier, a hash function (comprising of non-linear weak learners) is learned for each dimension of the descriptor. Gradient-based image features are used as the weak learners. This allows one to automatically learn the image gradient pooling configuration of the final descriptor. Trzcinski *et al.* [41] apply a sophisticated objective function, which makes the optimization far more challenging. However, the resulting binary descriptor significantly outperforms its binary competitors. Furthermore, the authors argue that their approach is more general and discriminative than those used in conventional binary descriptors, which often depend on simple thresholded linear projections [42].

2.3 Conclusion

The goal of a feature descriptor algorithm is to compute a compact description of the local neighborhood of the feature that is highly distinctive (i.e., patches from different keypoints map to different representations) and yet robust to changes in illumination and camera view-point (i.e., the same keypoint in different images maps to similar representations). While fast-to-compute feature descriptors are generally not robust enough in matching, state-of-the-art descriptor-based approaches [23, 6] are simply not computationally efficient. On the other hand, although classification-based approaches [18, 30, 40, 39] are well suited for achieving fast and robust real-time matching, they still face drawbacks in terms of high training times and/or memory consumption. Our objective in this thesis is to reduce both the training time as well as the memory footprint while maintaining a high level of robustness against a large range of deformations.

From the following chapter onwards, we commence with our discussion on the proposed target localization system. Chapter 3 to Chapter 6 describe our classification-based learning approach (i.e., the model generation framework) of Fig. 1.2. The runtime target localization approach is detailed in Chapter 7.

Chapter 3

Viewset Generation

Our aim in this thesis is to develop a keypoint recognition technique which can successfully localize a target in real-time under various viewing conditions such as those depicted in Fig. 3.1. Keypoint recognition techniques can be organized under two broad categories: descriptor-based and classification-based. From Chapter 2, we have seen that most descriptor-based approaches require estimating the scale and rotation of a feature in order to desensitize the patch (from which the descriptor is computed) to those deformations. For these methods to obtain satisfactory detection rates, the orientation and scale parameters must be computed with a high degree of accuracy. However, the processes involved in estimating these parameters are generally quite computationally intensive. Nonetheless, say for instance that even if we are able to optimize these processes to run on low-end devices, the descriptors are still limited in their ability to handle perspective deformations which arise when the viewpoint conditions change significantly.

Contrary to descriptor-based approaches, classification-based techniques train a classifier on simulated images of the deformations to which localization must be achieved. The motivation behind applying this approach is twofold. Firstly, we can completely specify the environment under which we wish to localize the targets. Secondly, we can transfer some of the computational burden from matching to training, which is performed beforehand. Taking the above into consideration, we adopt the classification-based paradigm in this thesis. Furthermore, since our emphasis is on localizing targets which are primarily defined by planar surfaces, it is possible to render simulated images which are quite close to their real-world counterparts.

The process of rendering the synthetically deformed images is known as *viewset generation*. For a specified deformation space, early classification-based approaches [19, 18, 30]

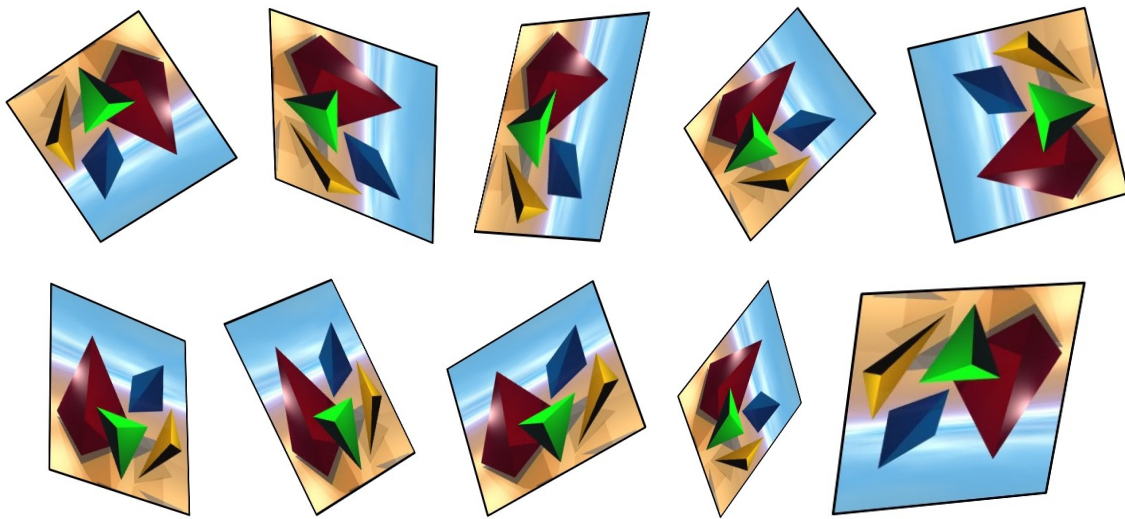


Figure 3.1: Typical viewing conditions of a target for which localization is to be performed.

used to train the classifier on the entire population of synthesized images. More recent approaches [40, 39] suggest clustering the synthesized images according to the amount of deformation overlap. This allows the classifier to be trained on features that are extracted from patches which are coherent with regard to their appearance. The images which are clustered together are referred to as *viewsets* or *viewpoint bins*. Throughout this thesis, we use both terms interchangeably. Figure 3.2 illustrates the concept of clustering the synthetic images into viewpoint bins. Also shown in Fig. 3.2 are the subsequent stages (after viewset generation) of the model generation framework comprising of stable keypoint identification and feature extraction. These topics are addressed in Chapter 6.

To avoid the use of a priori information of camera intrinsics, most approaches [19, 18, 30, 40] apply an affine transformation to generate these synthetic images. However, the affine transformation does not model the perspective deformations frequently encountered in real-world images. Hence, such techniques can only produce approximate results. Furthermore, to fully achieve a reasonable invariance to the perspective deformations, they typically require generating an excessively large number of images to span the desired deformation space.

We step away from this convention and apply a perspective projective transformation instead. Since our target application is for cameras on mobile computing devices such as smartphones, it is possible to approximate the camera calibration process to simulate real world scenarios. This allows us to sample the deformation space in a more representative, controlled way requiring less images per viewset to model the desired deformation space. We therefore

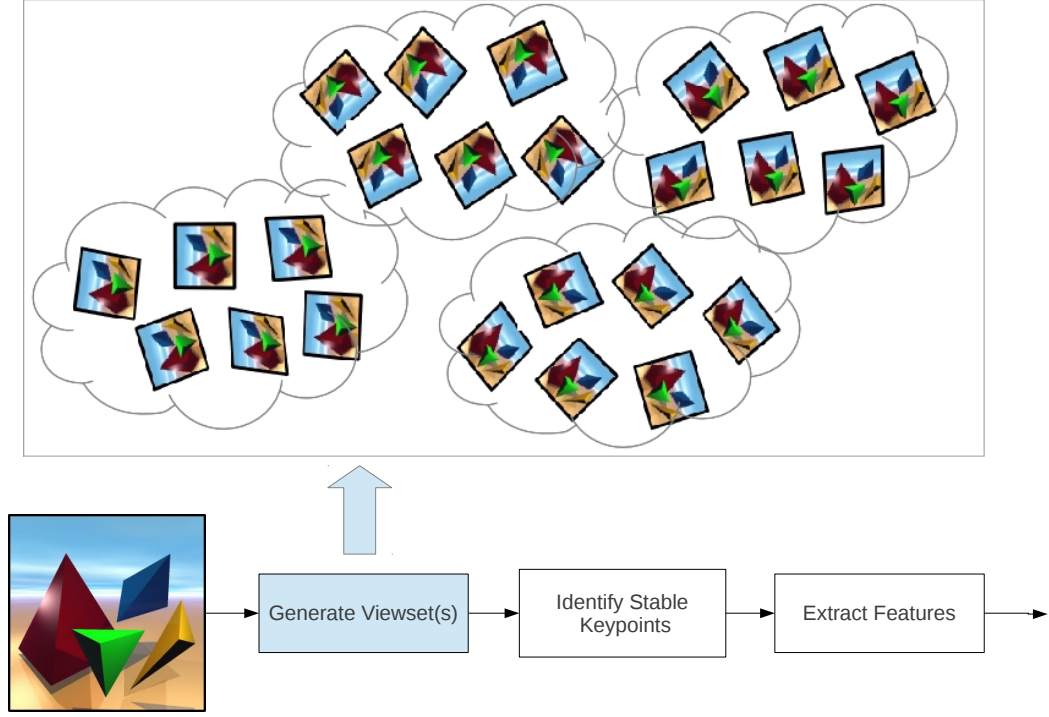


Figure 3.2: Clustering synthetically deformed images into corresponding groups (viewsets or viewpoint bins) according to their deformation overlap.

fix the vertical field of view of the camera at 30° and set our virtual plane at the distance such that the view frustum has a width equal to the plane diagonal. This ensures that the plane will remain entirely visible under an arbitrary rotation around any of the three axes.

3.1 Algorithm Overview

During training, multiple artificial views of the reference (aka target) image $\hat{I}$ are synthesized and grouped into numerous viewpoint bins (viewsets) so that each bin shares a similar range of deformations. A virtual image is created by perspectively warping the reference image $\hat{I}$. In an attempt to simulate real world images, we also add Gaussian noise after convolving each synthesized image I with a Gaussian kernel modeling the optical blur:

$$I = \mathcal{G} * \mathcal{W}(\hat{I}) + \mathcal{N}(0, \sigma), \quad (3.1)$$

where $\mathcal{W}()$ denotes the warping process, $\mathcal{G}$ is the Gaussian convolution kernel, and $\mathcal{N}(0, \sigma)$ denotes the additive Gaussian noise with mean 0 and standard deviation σ .

Each viewpoint bin comprises of M images synthesized using Eq. 3.1 with random angle and scale distortions selected from the viewpoint bin range. We specify the viewpoint bin range in terms of the rotation angles around all three axes, $[\Phi_{min}^x, \Phi_{max}^x], [\Phi_{min}^y, \Phi_{max}^y], [\Phi_{min}^z, \Phi_{max}^z]$ and the scale, $[S_{min}, S_{max}]$. Each bin range is determined by a step $\Delta\Phi$ computed as:

$$\Delta\Phi^x = \frac{\Phi_{max}^x - \Phi_{min}^x}{B_x}, \quad (3.2)$$

where B_x is the number of bins along the x axis. The angle range for the b^{th} bin is specified by $b\Delta\Phi^x \leq \phi_x < (b+1)\Delta\Phi^x$. Similar equations hold for the other two axes. Binning of the scale follows a multiplicative scheme:

$$\gamma = \sqrt[B_s]{\frac{S_{min}}{S_{max}}} \quad (3.3)$$

with b^{th} bin including scales in the range $\gamma^{b+1}S_{max} \leq s \leq \gamma^b S_{max}$. When combined together, the total number of viewpoint bins is $N = B_x B_y B_z B_s$. Thus, the overall number of virtual views to be synthesized is $M \times N$. We denote the m^{th} virtual image from the n^{th} viewpoint bin as I_{mn} .

The warping process, $\mathcal{W}()$, employs a perspective transformation to render a 2D image after arbitrarily rotating $\hat{I}$ around the x , y and z axis. A block diagram illustrating the warping process is shown in Fig. 3.3. Each point in $\hat{I}$ is converted to a 4D homogeneous point in the local coordinate system of the object, i.e. the object coordinate system. The object coordinate system defines the initial position and orientation of the object before any transformation is applied. Since $\hat{I}$ is a planar object (a 2D image), the z component is 0 for all points in $\hat{I}$. Thereafter, the object undergoes a deformation process comprising of an arbitrary 3D-rotation and translation. The resulting 3D object is then perspectively transformed to obtain a 2D representation in the normalized device coordinate (NDC) plane. The NDC coordinates are then mapped to window coordinates by applying a viewport transformation. Finally, the virtual image is rendered by passing the window coordinates to an interpolation process to obtain the corresponding pixel intensity values. The deformation, perspective projection, and viewport transformation processes are described in the following sections.

3.2 Deformation Process

The deformation process comprises of an arbitrary rotation (about the x , y and z axes) and a translation (along the z axis). Both these steps can be performed in a single step by combining

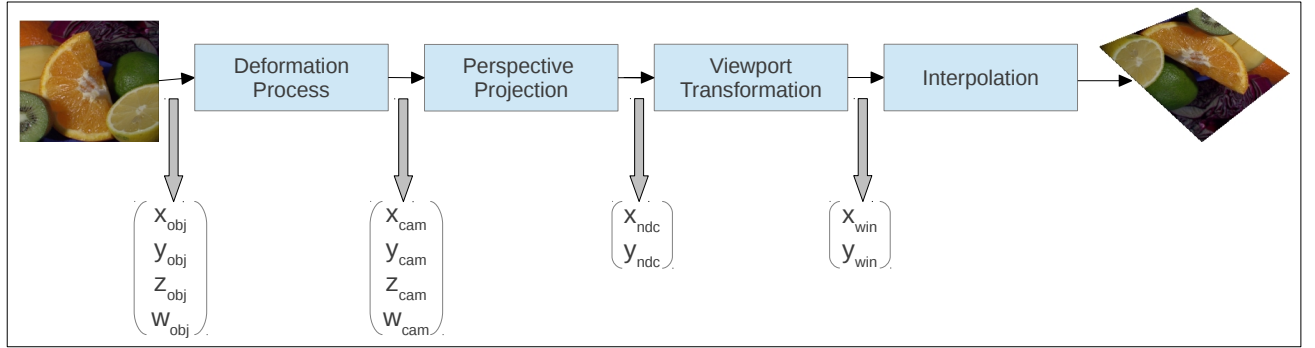


Figure 3.3: Perspective transformation of the reference image $\hat{I}$.

a 3×3 rotation matrix, $\mathcal{R}$, and a 1×3 translation vector, $\mathcal{T}$, into a single 4×4 matrix $\mathcal{D}$. We refer to $\mathcal{D}$ as the deformation matrix. The object coordinates are therefore transformed as follows:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ w_{cam} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathcal{R} & \mathcal{T} \\ \mathbf{0}^T & 1 \end{bmatrix}}_{\mathcal{D}} \begin{bmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{bmatrix}. \quad (3.4)$$

The rotation matrix $\mathcal{R}$ is the product of the individual rotation matrices around the x , y , and z axes:

$$\mathcal{R} = R_{\phi_x} R_{\phi_y} R_{\phi_z}, \quad (3.5)$$

where ϕ_x , ϕ_y , and ϕ_z represent the respective rotation angles around each corresponding axis. These individual matrices are simply:

$$R_{\phi_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi_x & \sin \phi_x \\ 0 & -\sin \phi_x & \cos \phi_x \end{bmatrix}, \quad (3.6)$$

and

$$R_{\phi_y} = \begin{bmatrix} \cos \phi_y & 0 & -\sin \phi_y \\ 0 & 1 & 0 \\ \sin \phi_y & 0 & \cos \phi_y \end{bmatrix}, \quad (3.7)$$

and

$$R_{\phi_z} = \begin{bmatrix} \cos \phi_z & \sin \phi_z & 0 \\ -\sin \phi_z & \cos \phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.8)$$

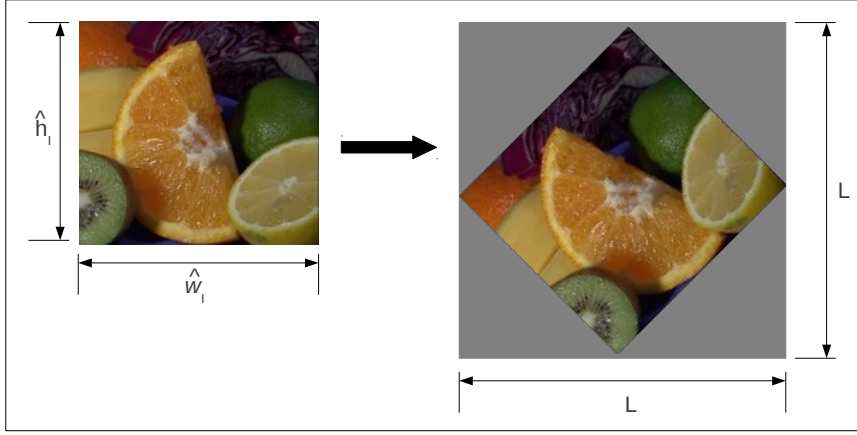


Figure 3.4: Square region fully containing an arbitrarily rotated $w_I \times h_I$ reference image $\hat{I}$.

After the object is rotated, it needs to be positioned in the camera coordinate space. The camera coordinate space models a 3D space as seen through a virtual camera. This is done by simply translating the rotated object along the simulated camera axis (z axis). The amount of translation is specified by the variable t . To compute t , we must first determine the side length, L , of a square that would wholly contain any rotation of the object around the camera axis. Figure 3.4 illustrates this setup. For a planar object of size $\hat{w}_I \times \hat{h}_I$, this is simply the diagonal connecting the top left corner to the bottom right corner of $\hat{I}$:

$$L = \sqrt{\hat{w}_I^2 + \hat{h}_I^2}. \quad (3.9)$$

We can now proceed to determine t using the setup presented in Fig. 3.5. Note that the camera coordinates are defined by the right-handed coordinate system. The virtual camera, situated at the origin, faces down the negative z -axis, with the positive y -axis pointing up, and the positive x -axis pointing to the right.

To ensure that all parts of the deformed object remain visible in the warped image, we need to translate the object by an amount such that all the points of the object fall within the view frustum boundaries. Therefore, in order to determine t for an arbitrary $\mathcal{R}$, we must consider the specific case in which the object is rotated about the x axis by ϕ_x such that the object touches the upper plane of the view frustum (as shown in Fig. 3.5). By computing the hypotenuse of the resulting right-angled triangle formed by the object and the view frustum, we can obtain t as follows:

$$\sin\left(\frac{\alpha}{2}\right) = \frac{L/2}{t} \implies t = \frac{L}{2 \sin\left(\frac{\alpha}{2}\right)}. \quad (3.10)$$

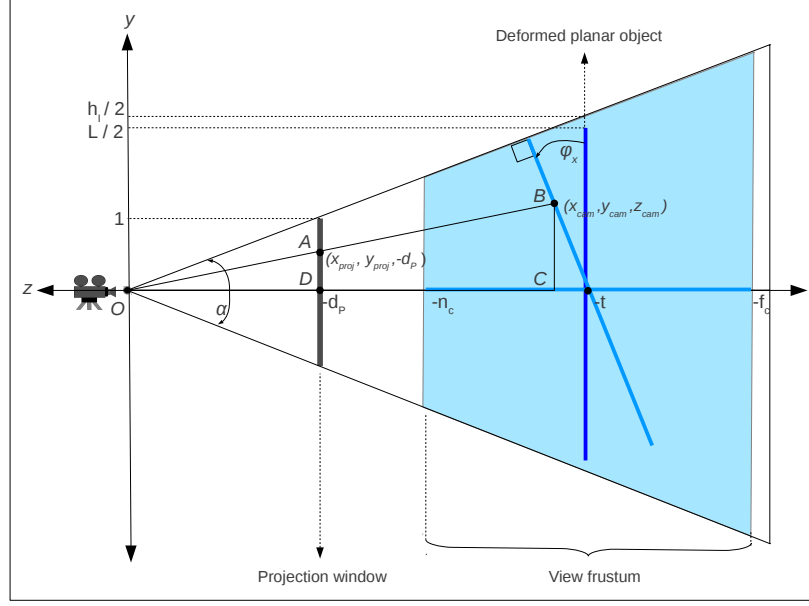


Figure 3.5: Side view frustum of the planar object centered at $(-t, 0)$: α is the vertical field of view, t represents the the distance of the object's center from the camera, L denotes the side length of a square that would wholly contain any rotation of the object, n_c and f_c represent the near and far clipping planes respectively, and ϕ_x is the rotation angle around the x axis.

Since we embed $\hat{I}$ in a $L \times L$ square image, t remains the same as Eq. 3.10 for the pathological case for the rotation about the y axis. Hence, for arbitrary out-of-plane rotations, the translation vector $\mathcal{T}$ is specified as

$$\mathcal{T} = \begin{bmatrix} 0 & 0 & -t \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & -\frac{L}{2 \sin(\frac{\alpha}{2})} \end{bmatrix}^T. \quad (3.11)$$

3.3 Perspective Transformation

Rotating $\hat{I}$ and positioning it in the camera coordinate space results in a 3D virtual scene. The task at hand is to now project the object points contained within the view frustum of the 3D camera space onto a 2D image. In order to achieve realism in the rendered images, we adopt a perspective transformation to maintain the depth relationship between points. The perspective transformation comprises of two main components: *perspective projection*, and *viewport transformation*. Points from the view frustum are first projected onto the NDC space using a perspective projection process. Thereafter, a viewport transformation is applied to

map the points from the NDC coordinates to the window coordinates. Both these stages are described below.

3.3.1 Perspective Projection

The perspective projection process requires 2 parameters to be specified: *aspect ratio* ψ , and *vertical field of view* (FOV) α . The aspect ratio defines the relationship between the width and height of the rendered image. Ideally, ψ should be set such that the aspect ratio of $\hat{I}$ is maintained. The FOV parameter allows us to determine the distance of the virtual image plane from the camera, i.e. the focal length.

We start by determining the distance of the virtual image plane from the camera. The image plane is an infinite plane parallel to the x - y plane. However, we can only perceive points which are projected from the view frustum onto a finite rectangular region of the image plane. This finite region is known as the projection window. In order to avoid distortion, the aspect ratio of the projection window should maintain the aspect ratio of $\hat{I}$. Thus, since we embed $\hat{I}$ in a $L \times L$ square image (Fig. 3.4), the aspect ratio is always 1:

$$\psi = \frac{\text{width}}{\text{height}} = \frac{L}{L} = 1. \quad (3.12)$$

Although for our application the aspect ratio is 1, we continue to use the variable ψ to derive the projection process for an arbitrary aspect ratio for the general case.

Our objective is to determine a projection matrix $\mathcal{P}$ which can map points from the view frustum onto the normalized NDC space. The NDC space is defined in the interval: $x_{ndc} \in [-1, 1]$, $y_{ndc} \in [-1, 1]$. We can conveniently assign the height of the projection window as 2 with $y_{proj} \in [-1, 1]$. This implies that the width of the projection window is exactly twice the aspect ratio (from Eq. 3.12). Hence, the projection window is defined as: $x_{proj} \in [-\psi, \psi]$ and $y_{proj} \in [-1, 1]$. Anything outside this range is not visible. We can already see that the coordinates inside the projection window will have their y -axis component in the required normalized range $[-1, 1]$. The x -axis components of the projection window can be normalized by division with the aspect ratio ψ .

We now determine the distance, d_P , of the projection window from the camera. With the knowledge of the height of the projection window, and for a given value of the vertical FOV α , we can compute d_P as follows:

$$\tan\left(\frac{\alpha}{2}\right) = \frac{1}{d_P} \implies d_P = \frac{1}{\tan\left(\frac{\alpha}{2}\right)}. \quad (3.13)$$

Having obtained an expression for d_P , we can now compute the projection of an arbitrary point $(x_{cam}, y_{cam}, z_{cam})$ onto the image plane. From Fig. 3.5, we can see two similar triangles: $\triangle_{OAD}$ and $\triangle_{OBC}$. By exploiting the rule of similar triangles, we can compute the normalized (x_{proj}, y_{proj}) as follows:

$$\frac{x_{proj}}{-d_P} = \frac{1}{\psi} \frac{x_{cam}}{z_{cam}} \implies x_{proj} = -d_P \frac{x_{cam}}{\psi z_{cam}} = -\frac{x_{cam}}{\psi z_{cam} \tan\left(\frac{\alpha}{2}\right)} \quad (3.14)$$

$$\frac{y_{proj}}{-d_P} = \frac{y_{cam}}{z_{cam}} \implies y_{proj} = -d_P \frac{y_{cam}}{z_{cam}} = -\frac{y_{cam}}{z_{cam} \tan\left(\frac{\alpha}{2}\right)} \quad (3.15)$$

The z_{proj} value is simply $-d_P$ (Eq. 3.13).

We can represent the division by z_{cam} (in Eq. 3.14 and Eq. 3.15) using homogeneous coordinates by setting the w -component of the projected homogeneous coordinates as $-z_{cam}$. Thus, we can model the perspective projection in matrix form:

$$\begin{bmatrix} x_{proj} \\ y_{proj} \\ z_{proj} \\ w_{proj} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{1}{\psi \tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & -\frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{\mathcal{P}} \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ w_{cam} \end{bmatrix}. \quad (3.16)$$

After the camera coordinates are transformed by multiplying with the projection matrix $\mathcal{P}$, the projected coordinates are still in homogeneous coordinates. By dividing the x and y coordinates by w_{proj} , the projected points are converted from homogeneous to normalized device coordinates:

$$\begin{bmatrix} x_{ndc} \\ y_{ndc} \end{bmatrix} = \frac{1}{w_{proj}} \begin{bmatrix} x_{proj} \\ y_{proj} \end{bmatrix}. \quad (3.17)$$

This process of dividing by the w -homogeneous component is known as *perspective division*.

3.3.2 Viewport Transformation

The perspective projection process maps the 3D volume enclosed by the view frustum onto the 2D NDC space. A viewport transformation is then applied to project the points from the NDC space onto window coordinates in order to determine the correct pixel associated with each point. The viewport transformation requires two parameters to be specified, namely, the width w_I and height h_I of the final rendered image. The window coordinates are determined

by scaling and translating the NDC coordinates as follows:

$$\begin{bmatrix} x_{win} \\ y_{win} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} x_{ndc} + 1 & 0 \\ 0 & y_{ndc} + 1 \end{bmatrix} \begin{bmatrix} w_I \\ h_I \end{bmatrix}. \quad (3.18)$$

We now address the issue of specifying the size of the rendered image. Since we maintain an aspect ratio of 1 (Eq. 3.12), w_I and h_I should be equivalent: $w_I = h_I$. Under normal circumstances, the scale of the rendered image is governed by the choice of α . In order to maintain the original scale of the reference image $\hat{I}$, we need to disassociate the influence of α on the scale of the rendered image. This can be accomplished if the viewport transform maps to a square image with a side length which is expressed in terms of α . From Fig. 3.5, we can compute the distance from the center point of the deformed object to the point directly above it on the upper plane of the view frustum as:

$$\frac{h_I}{2} = t \tan\left(\frac{\alpha}{2}\right) \implies h_I = 2t \tan\left(\frac{\alpha}{2}\right) = L \sec\left(\frac{\alpha}{2}\right). \quad (3.19)$$

Note that we are immune to the selection of α , since we translate the object accordingly (Eq. 3.10) and maintain its original size (Eq. 3.19). However, Eq. 3.10 places a constraint on the feasible range of α . Since $\sin 90^\circ = 0$, α must be in the range $(0^\circ, 180^\circ)$. Unless otherwise stated, we use the value $\alpha = 30^\circ$ for synthesizing all our training images.

Before interpolating the pixel intensity values, the rendered image can be scaled by an external parameter γ by modifying Eq. 3.19:

$$h_I = \gamma L \sec\left(\frac{\alpha}{2}\right). \quad (3.20)$$

Figure 3.6 presents a set of warped images using the aforementioned deformation and perspective transformation processes. The images were generated with the following ranges: $[\Phi_{min}^x, \Phi_{max}^x] = [-45^\circ, 45^\circ]$, $[\Phi_{min}^y, \Phi_{max}^y] = [-45^\circ, 45^\circ]$, and $[\Phi_{min}^z, \Phi_{max}^z] = [-180^\circ, 180^\circ]$. The scale parameter was kept constant at 1.

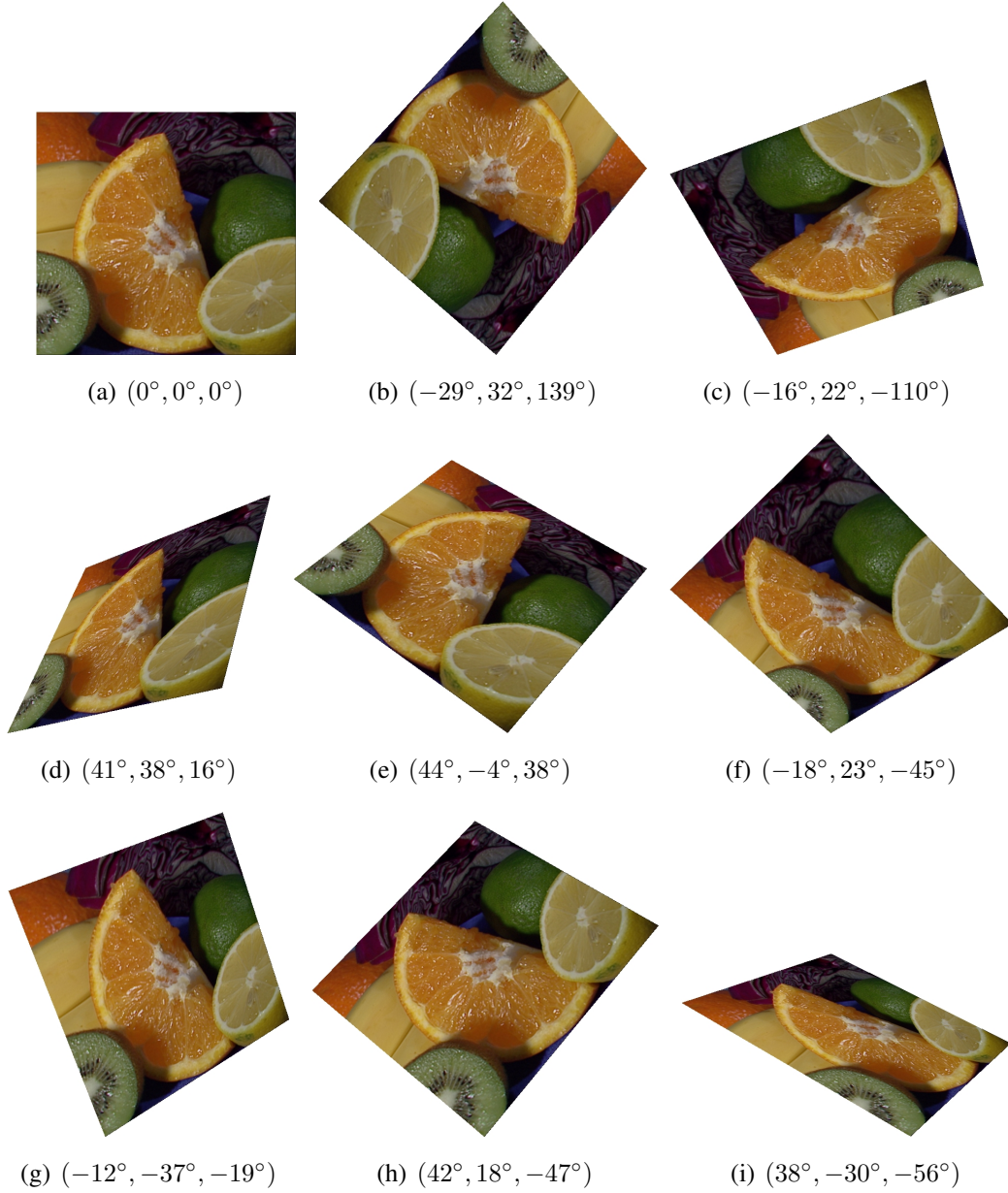


Figure 3.6: Perspectively transformed images. The rotations are specified as follows: (ϕ_x, ϕ_y, ϕ_z) . No external scaling was used, i.e. $\gamma = 1$. However, for visual representation, some of the synthesized images were scaled.

3.4 Conclusion

In this chapter, we have described the viewset generation process. The viewset images are synthesized by applying a perspective transformation (instead of the conventional affine transformation). The synthesized images are clustered into viewpoint bins according to their similarity with regard to their deformation content. The viewpoint clustering process restricts the patch appearance variation which allows a classifier to be trained on features that are extracted from patches which are visually coherent. The application of the perspective transform enables us to sample the deformation space in a more controlled way requiring less images per viewpoint bin to cover a specified range of deformations.

The next stage in the model generation framework comprises of detecting features from each image for the viewpoint bin under consideration. We discuss the feature detection component in the following chapter.

Chapter 4

Feature Detection

The first step in any image matching application, which requires establishing correspondences between similar real world points appearing in variously deformed images, is based on feature detection. There are four types of features that can be detected, namely, *edges*, *salient points*, *blobs* and *ridges*. An important property of any feature detector is that it should be able to consistently localize the same feature in an arbitrarily deformed image.

Edge features are a set of contiguous points, characterized by strong gradient magnitudes, forming a contour between two image regions. Salient point (or keypoint) features are point-like features in an image. Initially, the concept of salient points was developed for locating corners by analyzing the edges to find fast changing regions across the vertical and horizontal directions. These algorithms were later refined to detect high levels of curvature in image gradients, thus mitigating the edge detection stage. On the contrary to edges and keypoints, blob features encapsulate regions of an image for which some property, for instance saturation or hue, are constant or vary within a prescribed interval. Ridge features are inspired from the actual geographical concept of ridges. In an image, a ridge occurs when there is a connected sequence of pixels having intensity values which are higher, or lower, in the sequence than those neighboring the sequence. Thus, a ridge can be considered as an approximate medial axis of an elongated object such as a road in an aerial image, or a blood vessel in a retinal scan.

Keypoint-based feature matching systems are robust to large deformations, making them relatively easy to localize [34]. Furthermore, they are generally the fastest to compute, with the least complexity, and are therefore more suitable for low-powered devices (such as smart phones). For these reasons, we investigate various kinds of keypoint detectors in this chapter and describe one of the most efficient of them all, the FAST detector [34, 35].

4.1 Keypoint Detectors

Most keypoint detectors operate by computing a keypoint response function ($\mathbb{K}$) across an image. Pixels which exceed a threshold keypoint quality score, and are locally maximal, are then retained. We provide a review of the most well known and widely used keypoint detectors in the field.

Harris detector

The Harris detector [13] is an extension of the initial work of Moravec [27]. The detector works as follows. For a given image I , an approximation to the local auto-correlation function of I is computed for each pixel:

$$\mathfrak{A} = \begin{bmatrix} \sum_{u,v} w_{u,v} \cdot [\mathcal{I}_x(x_u, y_v)]^2 & \sum_{u,v} w_{u,v} \cdot \mathcal{I}_x(x_u, y_v) \mathcal{I}_y(x_u, y_v) \\ \sum_{u,v} w_{u,v} \cdot \mathcal{I}_x(x_u, y_v) \mathcal{I}_y(x_u, y_v) & \sum_{u,v} w_{u,v} \cdot [\mathcal{I}_y(x_u, y_v)]^2 \end{bmatrix}, \quad (4.1)$$

where $\mathcal{I}_x$ and $\mathcal{I}_y$ denote the derivatives of image I , $(x_u, y_v) := (x + u, y + v)$, and $w(u, v)$ represents both a window and weighting function. From the eigenvalues, λ_1 and λ_2 , of $\mathfrak{A}$, a region can be classified as either uniform (λ_1 and λ_2 small), edge region (one small, one large) or keypoint region (both large). To avoid having to explicitly compute the eigenvalues, Harris and Stephens [13] propose the following keypoint score (which they derived from the eigenvalues):

$$\begin{aligned} \mathbb{K} &= \lambda_1 \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2 \\ &= \det(\mathfrak{A}) - k(\text{trace}(\mathfrak{A}))^2. \end{aligned} \quad (4.2)$$

To filter out keypoints which are spatially adjacent to one another, an 8-neighborhood non-maximum suppression is applied: candidates with a score of less than a predefined percentage δ of the maximum response encountered, $\mathbb{K}(p) < \delta \cdot \max_p \{\mathbb{K}(p)\}$, where $p = (x, y)$ is the candidate keypoint, are removed.

Difference of Gaussians

Lowe [23] introduced a robust method for detecting keypoints by identifying local extrema of an image filtered with differences of Gaussians (DoG). Lowe demonstrates that the DoG is an approximation to the Laplacian of Gaussian (LoG) which is used in [21]. To achieve

scale-invariance, a pyramid of images is constructed by convolving the image I with DoGs at various scales σ :

$$\begin{aligned} \text{DoG}_{k,\sigma}(x, y) &= \mathcal{G}(x, y, k\sigma) - \mathcal{G}(x, y, \sigma) \\ &= \frac{1}{2\pi(k\sigma)^2} e^{-(x^2+y^2)/2(k\sigma)^2} - \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}, \end{aligned} \quad (4.3)$$

where $\mathcal{G}$ is a Gaussian filter. This is accomplished by first convolving I with $\mathcal{G}(\sigma)$ and then computing differences of the resulting images:

$$\begin{aligned} \nabla I_0 &= \text{DoG}_{k,\sigma_0} * I = \mathcal{G}(k\sigma_0) * I - \mathcal{G}(\sigma_0) * I \\ \nabla I_1 &= \text{DoG}_{k,k\sigma_0} * I = \mathcal{G}(k^2\sigma_0) * I - \mathcal{G}(k\sigma_0) * I \\ &\vdots \end{aligned} \quad (4.4)$$

Keypoints are then detected by selecting local extrema, which are found by comparing each sample to its eight neighbors in the current image ∇I_n and the 18 neighbors “above” (in ∇I_{n-1}) and “below” (in ∇I_{n+1}). The final keypoint locations are then obtained by fitting a parabola to the sample point and its immediate neighbors (subpixel refinement) and applying non-maximum suppression.

Fast Hessian

The fast Hessian keypoint detector was developed in conjunction with the SURF (Speeded Up Robust Features) descriptor [6]. It is derived from the determinant of the Hessian matrix, which at scale σ is defined as follows:

$$\mathfrak{J}(x, y, \sigma) = \begin{bmatrix} \frac{\partial^2}{\partial x^2} \mathcal{G}(\sigma) * I(x, y) & \frac{\partial}{\partial x} \frac{\partial}{\partial y} \mathcal{G}(\sigma) * I(x, y) \\ \frac{\partial}{\partial x} \frac{\partial}{\partial y} \mathcal{G}(\sigma) * I(x, y) & \frac{\partial^2}{\partial y^2} \mathcal{G}(\sigma) * I(x, y) \end{bmatrix}. \quad (4.5)$$

Convolution with Gaussian second order derivatives is computationally intensive, especially at higher scales. For this reason, Bay *et al.* [6] approximate them with simple box filters (shown in Fig. 4.1) allowing for efficient computation using an integral image. Based on the box filter approximation, each potential keypoint p is scored as follows:

$$\begin{aligned} \mathbb{K}(p, \sigma) &= \tilde{\mathcal{I}}_{xx}(\sigma) \cdot \tilde{\mathcal{I}}_{yy}(\sigma) - \left(0.9\tilde{\mathcal{I}}_{xy}(\sigma)\right)^2 \\ &\approx \det(\mathfrak{J}(p, \sigma)), \end{aligned} \quad (4.6)$$

where $\tilde{\mathcal{I}}_{xx}$, $\tilde{\mathcal{I}}_{xy}$, and $\tilde{\mathcal{I}}_{yy}$ are the results of convolving the image with the filters depicted in Fig. 4.1, and the factor 0.9 assists in approximating $\det(\mathfrak{J}(p, \sigma))$ more accurately. As with the DoG detector, non-maximum suppression and subpixel refinement are then applied.

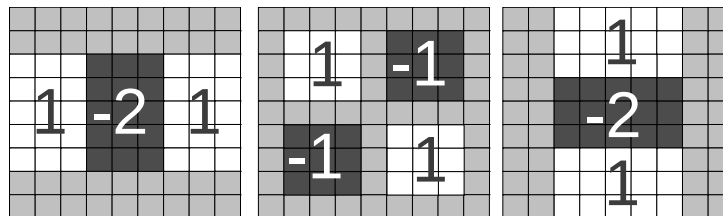


Figure 4.1: The box filters used by the Fast Hessian detector to approximate second order derivatives of Gaussians. The weights of the grey regions are zero, whereas the weights of the white and black regions are as shown.

Features from Accelerated Segment Test (FAST)

Rosten and Drummond [34, 35] developed a high-speed corner detector which they coined FAST, for Features from Accelerated Segment Test. The algorithm operates on a discretized Bresenham circle around a candidate point p . p is regarded as a keypoint if there exists a contiguous arc of at least n pixels that are all brighter or all darker than p by a threshold δ_t . This criterion for classifying a pixel as a keypoint is known as the *segment test*. By only testing for a subset of the pixels, a quick check can be made to reject false candidate keypoints prior to conducting the full segment test. This criterion is known as the *Accelerated Segment Test* [34]. In [35], Rosten and Drummond employed a machine learning approach to further accelerate the classification of pixels into keypoints. An ID3 decision tree [31] was trained to optimize the selection of the ordering of questions, i.e., which neighboring pixel in the Bresenham circle should be tested next to quickly determine whether p is a valid keypoint. As with all keypoint detectors, non-maximum suppression is also applied by computing a supplementary keypoint score (since the segment test criterion does not inherently quantify the quality of the keypoint). We describe the FAST detector in more depth in Sect. 4.2.

Adaptive and Generic Accelerated Segment Test (AGAST)

Based on the principles of the FAST detector, an approach called AGAST (Adaptive and Generic Accelerated Segment Test), is proposed by Mair *et al.* [25]. In AGAST, the ternary decision tree used in FAST is replaced by a binary decision tree. In order to reduce the false responses and automatically adapt to homogeneous and textured regions, two trees are constructed: one for homogeneous and one for structured regions. By combining these trees, the keypoint detector automatically adapts to the environments and provides the most efficient decision tree for the image region. The authors claim that the AGAST detector does not have to be re-trained for every new environment (a limitation FAST suffers). The same non-maximum

suppression used in FAST is adopted in AGAST as well.

BRISK detector

Leutenegger *et al.* [20] developed a novel binary feature matching approach which they termed as Binary Robust Invariant Scalable Keypoints (BRISK). Along with the BRISK descriptor, the authors also provide a keypoint detection algorithm which exploits the FAST detector. A drawback of both FAST and AGAST is that they are susceptible to changes in scale. The BRISK detector aims at addressing this limitation by going a step further by searching for extrema not only in the image plane, but also in scale-space using the FAST score as a measure for saliency. The FAST score is defined as the maximum threshold still classifying an image point as a keypoint using the FAST detector with $n = 9$.

4.2 The FAST Keypoint Detector

The FAST detector boasts of low complexity levels (as will be seen) and is therefore an ideal candidate for keypoint detection on low-power devices. It was initially developed by Rosten and Drummond in [34] with the intention to achieve real-time 3D model-based tracking. Keypoints are determined by applying a relaxed version of the SUSAN (Smallest Univalue Segment Assimilating Nucleus) corner criterion [38]. Instead of evaluating the circular disc, only the pixels on a Bresenham circle of radius r , centered on the candidate point p , are considered. If n contiguous pixels are all brighter than $I(p) + \delta_t$ or all darker than $I(p) - \delta_t$, where δ_t is the FAST threshold parameter, then p is classified as a keypoint. We investigate various values for δ_t in Sect. 4.3.3. In [34], a Bresenham circle of radius 3 was used with $n = 12$ as shown in Fig. 4.2.

For $r = 3$, Rosten and Drummond [34] applied a heuristic optimization process to speed up keypoint detection in an image by rejecting false candidate pixels prior to applying the full segment test. They achieve this by examining pixels 1, 9, 5 and 13, i.e. the four compass directions, of Fig. 4.2. A keypoint can only exist if at least three of these test points are all brighter than $I(p) + \delta_t$ or darker than $I(p) - \delta_t$. The full segment test criterion can then be applied to the remaining candidates by examining all pixels on the circle. With the added optimization, Rosten and Drummond claim that the algorithm examines, on average, 2.8 pixels to decide whether there is a keypoint at a given location [34].

Although this detector in itself demonstrates high performance, there are several weaknesses as noted by the authors [35]:

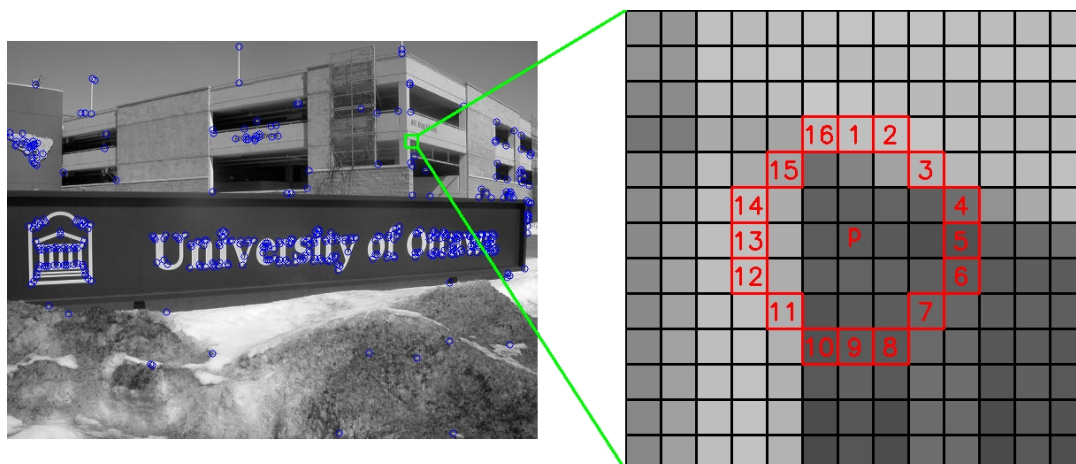


Figure 4.2: Keypoint detection with the FAST detector using a Bresenham circle of $r = 3$. The red squares are the pixels used in the keypoint detection. The pixel at p is the center of a candidate keypoint.

- Multiple features are detected adjacent to one another.
- The heuristic optimization tests do not generalize well for $n < 12$.
- The selection and ordering of the optimization test pixels impose implicit assumptions about the distribution of keypoint appearance.
- Knowledge from the first 4 tests is discarded.

The first issue is easily addressed by applying non-maximum suppression to filter out excessive candidate keypoints. In contrast to all aforementioned detectors, detection with the FAST algorithm does not directly provide a measure of the “quality” of a keypoint. In order to apply non-maximum suppression, the following score is computed for each candidate keypoint [35]:

$$\mathbb{K}(p) = \max \left\{ \sum_{p_i \in \mathcal{S}_+} |I(p_i) - I(p)| - \delta_t, \sum_{p_i \in \mathcal{S}_-} |I(p_i) - I(p)| - \delta_t \right\} \quad (4.7)$$

where $\mathcal{S}_+$ is the subset of pixels on the Bresenham circle that are brighter than $I(p) + \delta_t$ and $\mathcal{S}_-$ the subset of pixels that are darker than $I(p) - \delta_t$.

To address the remaining issues, Rosten and Drummond [35] train an ID3 decision tree to learn the results of the segment tests. Therefore, the problem of detecting a keypoint is reformulated as one of classification: given an image region centered at p , it is required to

classify p as a keypoint or not. We present a summary of the machine learning approach as described in [35].

Machine Learning Approach: FAST

1. Select a set of training images.
2. For every pixel p in each training image:
 - (a) Extract the pixels p_i on the Bresenham circle centered at p . By concatenating these pixel values, a feature vector ζ is formed. This will be used to train a decision tree later. Note that each value in the vector can be in one of three states. Darker than $I(p) - \delta_t$, brighter than $I(p) + \delta_t$, or similar to $I(p) \pm \delta_t$.
 - (b) Classify each value p_i in the feature vector into one of the following states:

$$\mathcal{S}(p_i) = \begin{cases} \mathcal{S}_-, & I(p_i) \leq I(p) - \delta_t \quad (\text{darker}) \\ \mathcal{S}_0, & I(p) - \delta_t < I(p_i) < I(p) + \delta_t \quad (\text{similar}) \\ \mathcal{S}_+, & I(p) + \delta_t \leq I(p_i) \quad (\text{brighter}) \end{cases} \quad (4.8)$$

- (c) Based on the result of the FAST segment test criterion, label p as either “keypoint”, or “not-keypoint”.

At the end of this step, we have available a training set of feature vectors with both positive (“keypoint”) and negative (“not-keypoint”) instances. This training set is denoted as $\hat{\zeta} = \{\zeta_1, \zeta_2, \dots\}$.

3. Partition $\hat{\zeta}$ into three subsets according to the state of each feature vector element: $\hat{\zeta}_-$, $\hat{\zeta}_0$, and $\hat{\zeta}_+$.
4. Define a random variable X which is true if p is a keypoint and false if it is not.
5. Apply the ID3 algorithm [31] to query each subset using the random variable X for the knowledge about the true class. The ID3 algorithm is based on the principle of entropy minimization and is terminated when the entropy of a subset is zero.

The objective of applying a machine learning process is to construct a classifier in order to test the least number of pixels for classifying a candidate pixel as keypoint or non-keypoint. To this end, Rosten and Drummond [35] claim that on average, the decision tree asks 2.26 (for

$n = 9$) and 2.39 (for $n = 12$) questions per pixel to determine whether or not it is a keypoint. Another incentive behind using a decision tree classifier is that it can easily be converted into C-code, by creating a long sequence of nested if-then-else statements which can further be optimized to exploit low-level instruction sets. It is important to note that since the training data contains an incomplete representation of all possible keypoints, the learned detector is not precisely the same as the segment test detector.

4.3 Experimental Evaluation of Keypoint Detectors

We now empirically evaluate various keypoint detectors against the FAST keypoint detector in an attempt to investigate their repeatability with regard to their processing time.

4.3.1 Dataset

Figure 4.3 shows examples from the Oxford datasets¹ used to evaluate the detectors. Five different changes in imaging conditions are evaluated (the resolution of the dataset images are shown in parenthesis):

- varying viewpoint changes: Fig. 4.3(a) (800×640) & 4.3(b) (1000×700);
- varying zoom and in-plane rotation: Fig. 4.3(c) (800×640) & 4.3(d) (765×512);
- varying image blur: Fig. 4.3(e) (1000×700) & 4.3(f) (1000×700);
- varying JPEG compression: Fig. 4.3(g) (800×640); and
- varying light: Fig. 4.3(h) (921×614).

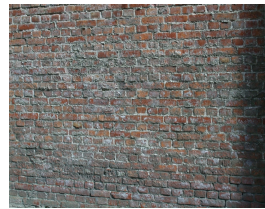
Note: Each of the datasets contains 6 images with an increasing geometric or photometric transformation. The viewpoint change datasets has out-of-plane rotations of 10° increments up until 60° . The zoom and blur change datasets are characterized by varied camera zoom and focus respectively, with the scale changed by an approximate factor of four. The light changes are introduced by varying the camera aperture. The JPEG sequence is generated using a standard *xv* image browser with the image quality parameter varying from 40% to 2%.

In the cases of viewpoint, zoom and blur change datasets, the same change in imaging conditions is applied to two different scene types. One scene type contains homogeneous regions with distinctive edge boundaries, whereas the other contains repeated textures of various

¹<http://www.robots.ox.ac.uk/~vgg/research/affine/>



(a) graffiti



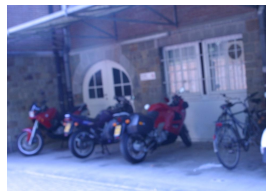
(b) wall



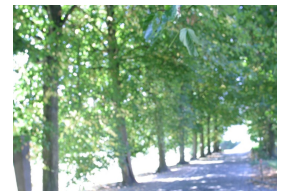
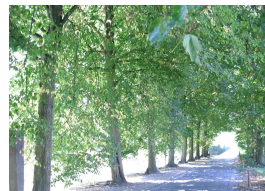
(c) boat



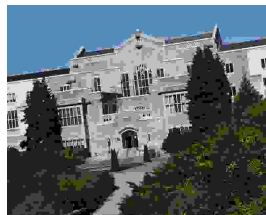
(d) bark



(e) bikes



(f) trees



(g) ubc



(h) leuven

Figure 4.3: Oxford datasets.

motifs. The ground truth homographies relating the reference (leftmost) image to the other 5 images are provided for each dataset.

4.3.2 Evaluation Criterion

Repeatability is one of the most important evaluation metrics used for assessing the stability of feature detectors. Simply stated, points detected in one image should also be detected at approximately corresponding positions in subsequent ones, irrespective of imaging conditions. Given a point $\hat{p}$ and two projection matrices P_1 and P_i , the projections of $\hat{p}$ onto images I_1 and I_i are $p_1 = P_1\hat{p}$ and $p_i = P_i\hat{p}$. A point p_1 detected in image I_1 is repeated in image I_i if the corresponding point p_i is also detected in image I_i . The number of such repeatable keypoints can be estimated between two images if the mapping between the images is known. For the case of planar scenes, this mapping is defined by a homography:

$$p_i = \mathcal{H}_{1i}p_1, \quad (4.9)$$

where we use the notation $\mathcal{H}_{1i} = P_iP_1^{-1}$. P_1^{-1} to represent the back-projection of image I_1 . The repeatability rate is defined as the fraction of repeatable points between two images with respect to the total number of points detected. Since the images used for computing repeatability are taken under varying imaging conditions, the observed scene parts differ. This implies that points detected in one image will have no corresponding points in the other image thus corrupting the repeatability measure. Therefore, while computing the repeatability measure, only points that lie in the common region between two images are considered. This common region is obtained by the homography relating the two images. Points $\tilde{p}_1$ and $\tilde{p}_i$ which lie in the common part of images I_1 and I_i are defined by:

$$\{\tilde{p}_1\} = \{p_1 | \mathcal{H}_{1i}p_1 \in I_i\} \text{ and } \{\tilde{p}_i\} = \{p_i | \mathcal{H}_{i1}p_i \in I_1\} \quad (4.10)$$

where $\{\tilde{p}_1\}$ and $\{\tilde{p}_i\}$ are the points detected in images I_1 and I_i respectively, and $\mathcal{H}_{ij}$ is the homography between images I_i and I_j .

Furthermore, since the images are transformed by various deformations, the corresponding point in the second image will not necessarily be where the homography predicts, but will lie in some neighborhood close to this prediction. The size of this neighborhood is denoted by $\delta_{\mathcal{E}}$ and repeatability within this neighborhood is called $\delta_{\mathcal{E}}$ -repeatability [37]. The repeatability rate for image I_i is thus defined by:

$$\text{Repeatability rate}(\delta_{\mathcal{E}}) = \frac{|\{(\tilde{p}_1, \tilde{p}_i) \mid \text{dist}(\mathcal{H}_{1i}\tilde{p}_1, \tilde{p}_i) \leq \delta_{\mathcal{E}}\}|}{\min(n_1, n_i)} \quad (4.11)$$

where the numerator denotes the number of points $(\tilde{p}_1, \tilde{p}_i)$ which correspond within an $\delta_{\mathcal{E}}$ -neighborhood, and $n_1 = |\{\tilde{p}_1\}|$ and $n_i = |\{\tilde{p}_i\}|$ are the number of points detected in the common regions of images I_1 and I_i respectively.

4.3.3 Optimal Threshold value for the FAST detector

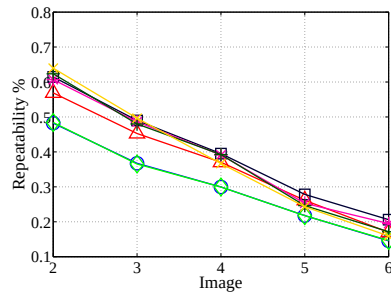
Before comparing the FAST detector to other approaches, we first present the results of an empirical investigation pertaining to the selection of the FAST threshold parameter (δ_t). We experimented with the following set of values: $\delta_t = \{10, 20, 30, 40, 50, 60, 70\}$. A maximum of 2000 keypoints was detected per image, and the repeatability was measured with $\delta_{\mathcal{E}} = 1.5$. Therefore, the likelihood that two points are accidentally within the error distance is very low. Unless otherwise stated, we use a Bresenham circle of radius 3 with $n = 9$ for the FAST detector in the all experiments in this thesis. The repeatability results are presented in Fig. 4.4.

Viewpoint Change. The results of viewpoint changes for the structured graffiti scene (Fig. 4.3(a)) and textured wall scene (Fig. 4.3(b)) are shown in Fig. 4.4(a) and Fig. 4.4(b) respectively. For the structured scene, $\delta_t \geq 40$ provides the best results, however, for the textured scene, larger threshold values ($\delta_t \geq 60$) yield slightly better repeatability scores for viewpoint changes up to 40° .

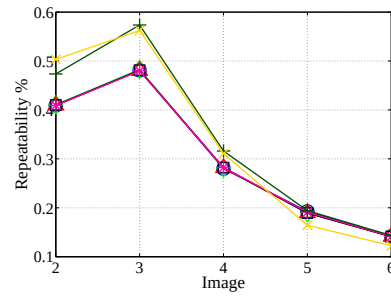
Rotation & Scale Change. Figure 4.4(c) presents the results for the structured scene (Fig. 4.3(c)), while Fig. 4.4(d) shows the results for the textured scene (Fig. 4.3(d)). In both sequences, the images are deformed by varying the scale and in-plane rotation. It is evident that for a larger range of scale and in-plane rotations, smaller threshold values are preferable for both textured and structured scenes, i.e., $\delta_t \leq 40$.

Blur. Figures 4.4(e) and 4.4(f) show the results for the structured (Fig. 4.3(e)) and textured scenes (Fig. 4.3(f)), both undergoing increasing amounts of image blur. For insignificant amount of blurring, higher threshold values yield better results. Since the FAST detector is scale-variant, it is unable to detect keypoints with large amounts of blur. This can be seen by the rapid decline in repeatability scores. Furthermore, as with the viewpoint and scale change datasets, the FAST repeatability scores are generally much lower for the textured scenes.

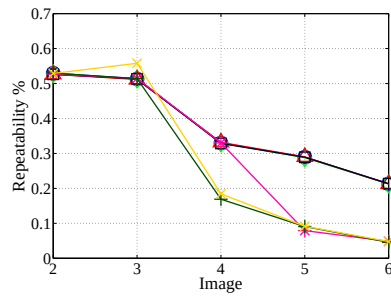
JPEG Artifacts. Figure 4.4(g) presents the results for the JPEG compression dataset from Fig. 4.3(g). We can see that for compression artifacts, greater repeatability scores are provided by higher thresholds, $\delta_t \geq 60$. This is due to the fact that, at higher thresholds, potentially less keypoints are detected, therefore increasing the repeatability score.



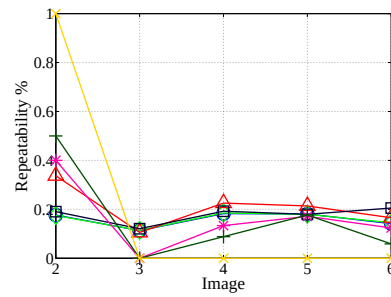
(a) graffiti



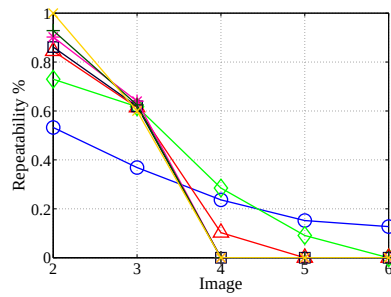
(b) wall



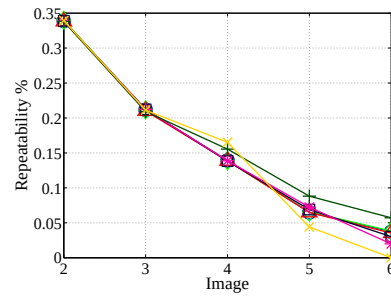
(c) boat



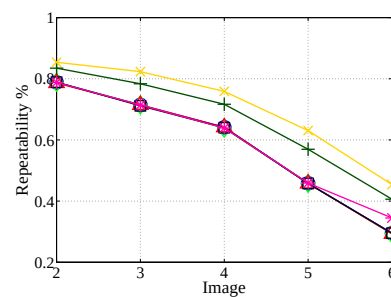
(d) bark



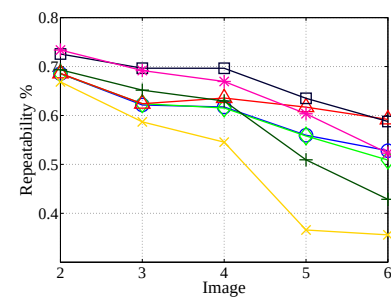
(e) bikes



(f) trees



(g) ubc



(h) leuven

Figure 4.4: Various thresholds for the FAST keypoint detector.

Light Change. The results for light changes for the images on Fig. 4.3(h) is shown in Figure 4.4(h). A threshold between $30 \leq \delta_t \leq 50$ presents the most invariant results to illumination changes, with $\delta_t = 40$ being the best performer for the given image.

Overall, without looking too deeply into the nature of the deformation, we can see that thresholds between $30 \leq \delta_t \leq 50$ yield the most repeatable keypoints. Unless otherwise stated, we use a threshold of $\delta_t = 40$ for all experiments requiring the FAST detector.

4.3.4 Comparison to Other Detectors

We compared the FAST detector to the following detectors: Harris, fast Hessian, Difference of Gaussians (DoG), BRISK. We used the OpenCV (version 2.4.3) implementations for these detectors with the recommended default settings. As with the previous experiment, a maximum of 2000 keypoints per image was detected with $\delta_\varepsilon = 1.5$ to measure the repeatability.

Timing Results

The average timing results for the detectors for each of the datasets is shown in Fig. 4.5. Clearly the fastest detector is the FAST detector requiring, on average, less than 2.5ms for fairly large sized images. Hence, for low-power devices, the FAST detector is a natural candidate for detecting keypoints owing to its simplicity and speed.

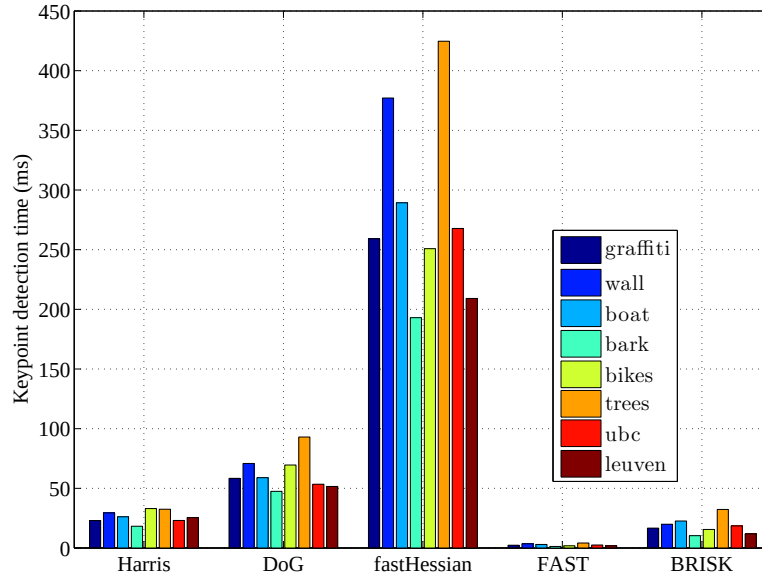


Figure 4.5: Average timing results for the various keypoint detectors.

Repeatability Results

The detector repeatability results are shown in Fig. 4.6.

Viewpoint Change. The results of viewpoint changes for the structured graffiti scene (Fig. 4.3(a)) and textured wall scene (Fig. 4.3(b)) are displayed in Fig. 4.6(a) and Fig. 4.6(b) respectively. The best results are obtained with the Harris detector followed by the FAST detector for both scene types. This is primarily due to the high detection accuracy especially on the homogeneous regions with distinctive boundaries. The repeatability score for a viewpoint change of 20 degrees varies between 26% and 64% and decreases for large viewpoint angles to 6% - 25% for typical structured scenes. It is interesting to note that the DoG and fast Hessian detectors perform better on textured scenes as compared to structured scenes.

Rotation & Scale Change. Figure 4.6(c) presents the results for the structured scene (Fig. 4.3(c)), while Fig. 4.6(d) shows the results for the textured scene (Fig. 4.3(d)). In both sequences, the images are deformed by varying the scale and in-plane rotation. The results are generally much better for the structured scenes, with the best performing detector being the Harris detector. For the textured scene, the unstable repeatability scores is due to the small number of detected points in this type of images. The sensitivity of the detectors to the scene type is clearly evident from these plots.

Blur. Figures 4.6(e) and 4.6(f) show the results for the structured (Fig. 4.3(e)) and textured scenes (Fig. 4.3(f)), both undergoing increasing amounts of image blur. The detectors which apply automatic scale selection, such as DoG, fast Hessian and BRISK, demonstrate better results than those that do not. Furthermore, for these detectors, the results are better than for viewpoint and scale deformations, especially for the structured scene. The DoG and fast Hessian detectors demonstrate a nearly horizontal repeatability curve, illustrating a high level of invariance to image blur. Clearly, the Harris and FAST detectors are more sensitive to this type of deformation. Since the image regions are smoother, it becomes more difficult to detect the keypoints at the same scale.

JPEG Artifacts. Figure 4.6(g) presents the results for the JPEG compression dataset from Fig. 4.3(g). For this type of structured scene, with large homogeneous regions and distinctive corners, the fast Hessian and Harris detectors are clearly best suited. The degradation under increasing compression artifacts is similar for all detectors.

Light Change. The results for light changes for the images on Fig. 4.3(h) is shown in Figure 4.6(h). All the curves are nearly horizontal, demonstrating good robustness to illumination changes, with the Harris and FAST detectors obtaining the highest repeatability scores for this type of scene.

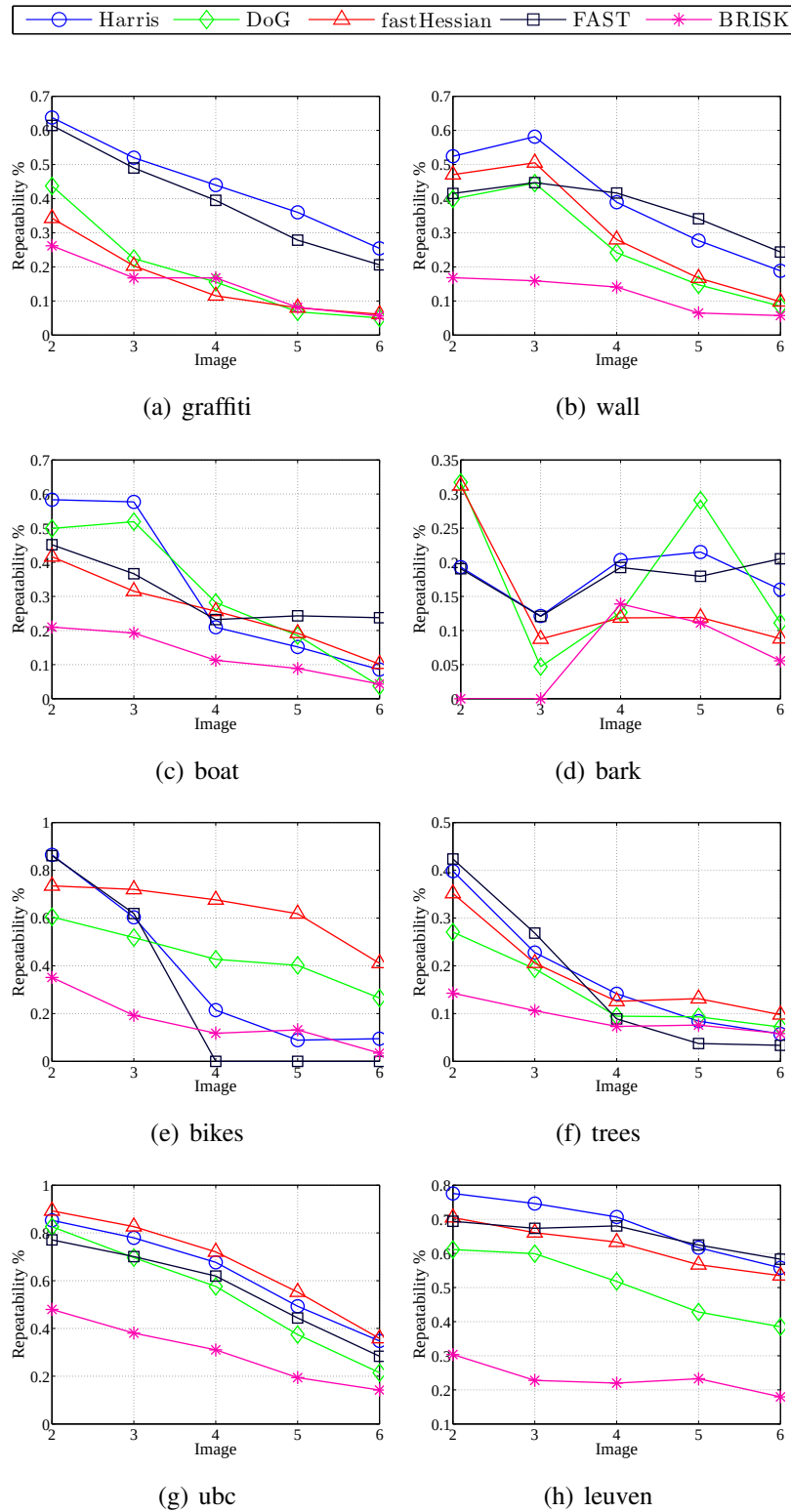


Figure 4.6: Keypoint detector repeatability evaluation results.

4.4 Conclusion

In this chapter we have reviewed the literature on feature detectors, in particular, keypoint detectors. We have empirically compared several keypoint detectors in order to select the most feasible approach in terms of computational complexity (detection time) and repeatability. For most of the deformations and artifacts, the Harris detector, followed by the fast Hessian and FAST detector was found to produce the best results. However, the results are largely dependent on the nature of the scene as well as the target application. In general, it was observed that textured scenes presented the greatest challenges. Since our focus is on real-time target detection, we recommend the FAST detector due to its minimal computing requirements and compatible repeatability results. Furthermore, we have found the following range of values to provide the best repeatability results for the FAST threshold parameter: $30 \leq \delta_t \leq 50$.

In the following chapter, we will detail the orientation normalization component of the model generation framework which involves estimating the orientation of each detected keypoint.

Chapter 5

Orientation Normalization

Orientation normalization is the process of normalizing a feature with regard to its estimated local orientation. The objective of this stage is to desensitize the features to in-plane rotations. Orientation normalization is achieved as follows. After a keypoint is detected and its orientation computed, the patch (centered at the keypoint) is rotated such that it is aligned with the estimated orientation. The pixel intensities corresponding to the newly rotated locations are obtained through interpolation. A descriptor can then be computed from the orientation normalized patch resulting in rotation invariant features.

The degree of invariance is determined by the accuracy and robustness of the orientation estimator. Obviously an accurate estimator is desirable, however, such an estimator would typically increase the computational burden, thus preventing real-time matching on power-constrained devices. On the other hand, cheap orientation estimators are generally sensitive to noise and might therefore potentially negate any advantage offered by orientation normalized features.

To retain the benefits of orientation normalization by using a cheap orientation estimator, Taylor and Drummond [39] suggest clustering keypoints which have similar spatial and orientation values from within the same viewpoint bin. In doing so, we can identify sets of keypoints with corresponding patches that share large appearance similarity. Therefore, when partial descriptors are aggregated, we are able to obtain more stable and reliable features. Clustering keypoints spatially is rather arbitrary and is therefore discussed in Chapter 6 when we describe methods for selecting stable keypoints. However, clustering the orientation values is non-trivial and requires special attention. In this chapter, we review existing orientation estimators and develop two novel orientation clustering techniques.

5.1 Orientation Estimation

5.1.1 Problem Definition

The orientation estimation problem can be defined as follows. Given a keypoint $p = (x, y)$ in an image, it is required to calculate an arbitrary angle $\theta_p \in [-180^\circ, 180^\circ]$ based on the point's local neighborhood. θ_p is arbitrary in that there is no right or wrong value associated with any individual image. However, if in two different images, the same real world point is encountered, then the orientations obtained should be consistent. Furthermore, θ_p must, to some extent, be invariant to image noise, blurring and scale - all of which influence the keypoint location as well as the image intensities.

5.1.2 Orientation Estimators

In the following section, we briefly review some well-known techniques used in the literature to estimate orientations of keypoints.

Intensity centroid method

The intensity centroid method was developed by Rosin [33]. First order geometric moments are employed to determine the keypoint orientations by computing the center of mass (aka centroid) of a region $\mathcal{R}$ centered on the keypoint. The geometric moments are computed as follows:

$$m_{ij} = \sum_{x,y} x^i y^j I(x, y) . \quad (5.1)$$

Note that the coordinate frame is set such that the image region containing the keypoint is centered at the origin $(0, 0)$. The keypoint orientation is obtained as the angle of the vector from the origin to the centroid $\left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right)$, with a 180° correction to compensate for keypoints which are slightly darker with respect to their neighboring intensities:

$$\theta = \begin{cases} \arctan\left(\frac{m_{01}}{m_{10}}\right) & \text{if color = bright} \\ \arctan\left(\frac{m_{01}}{m_{10}}\right) + 180^\circ & \text{if color = dark} \end{cases} . \quad (5.2)$$

Rublee *et al.* [36] adopt this method for estimating orientations for the ORB descriptors. They improve the rotation invariance of this measure by computing the moments with x and y remaining within a circular region of radius $\mathcal{R}$.

Gradient centroid method

A limitation of the intensity centroid method is that it requires an analysis of the intensity levels of the region $\mathcal{R}$ to correctly provide an estimation of the local orientation. To avoid this dependency, Rosin [33] suggests using the moments of the intensity of the gradient magnitude $\nabla I(x, y)$ instead of the moments of the image intensities:

$$\theta = \arctan \left(\frac{\sum_{x,y \in \mathcal{R}} y \nabla I(x, y)}{\sum_{x,y \in \mathcal{R}} x \nabla I(x, y)} \right). \quad (5.3)$$

A drawback to this method is that it requires an additional processing step to determine the gradient magnitude image $\nabla I(x, y)$.

Symmetry method

The symmetry orientation estimation method was designed to exploit the symmetry property of simple keypoints [33]. This method relies on the premise that the orientation which maximizes the symmetry of a patch, should coincide with the keypoint orientation. Keypoint symmetry is measured by requiring that intensities on either side of the keypoint bisector are approximately equal. This is achieved by rotating the image patch $\mathcal{R}$ by the hypothesized orientation θ and interpolating the intensities for $I_\theta(x, y)$. The local orientation can then be found as the rotation angle that minimizes the summed absolute differences in corresponding intensity values:

$$\theta = \min_{\theta} \sum_{y=0}^{\mathcal{R}} \sum_{x=-\mathcal{R}}^{\mathcal{R}} |I_\theta(x, y) - I_\theta(x, -y)|. \quad (5.4)$$

SIFT's orientation assignment method

In his seminal paper, Lowe [23] proposes a scale-invariant orientation estimator which is used to make the SIFT features rotation invariant. The gradient orientation $\theta_p = \arctan(\nabla I(p))$ at each pixel p within a region $\mathcal{R}$ centered at the keypoint is first computed. Thereafter, an empirical distribution of all such θ_p is estimated using a histogram. Each sample used to estimate the distribution is weighted according to the magnitude of the local gradient ∇I and a Gaussian centered at the keypoint location. The highest peak in this distribution, along with all the other peaks that are within 80% of the highest peak, are refined by fitting a parabola to the 3 histogram values closest to each peak to interpolate the peak position for improved accuracy.

MOPS's orientation assignment method

For the MOPS descriptor, Brown *et al.* [7] estimate the orientation of a keypoint from the local gradient of the image convoluted with a Gaussian kernel $\mathcal{G}(\sigma)$:

$$\theta = \arctan \left(\nabla \left(\mathcal{G}(\sigma) * I \right) \Big|_p \right). \quad (5.5)$$

An increasing σ results in a smoother representation of the gradient across the image. Thus ensures a certain robustness to errors in the keypoint location $p = (x, y)$. However, the computational costs also increases quadratically with σ .

SURF's orientation assignment method

The orientation assignment used in SURF [6] operates as follows. First-order Haar wavelet responses in the x and y directions are computed and weighted with a Gaussian centered at the keypoint. The wavelet responses at certain sampling points around the keypoint are then represented as a vector in a 2D space. Thereafter, the dominant orientation is estimated by calculating the sum of all the responses within a sliding orientation window of size 60° . This is done by summing all the horizontal and vertical responses within the window, resulting in a local orientation vector. The final orientation of the keypoint is obtained from the longest such vector over all windows.

Taylor's orientation assignment method

Taylor and Drummond [39] propose estimating the keypoint orientation from the sum of gradient vectors sampled across a Bresenham circle of radius 3. Since the x and y coordinates of the ring are fixed (relative to the keypoint location), it is possible to quickly estimate the orientation vector from the weighted sums of the 8 pixel differences. Although exceptionally fast to compute, a drawback to this method is that it requires prominent intensity differences in a relatively small neighborhood.

FREAK's orientation assignment method

FREAK [3] employs a sophisticated retinal pattern to compute local gradient vectors over pairs of selected sampling points. The pairs are chosen to have symmetric receptive fields, with larger fields in the perifoveal region, with respect to the center as shown in Fig. 5.1. More information is obtained by adding redundancy through the selection of overlapping receptive

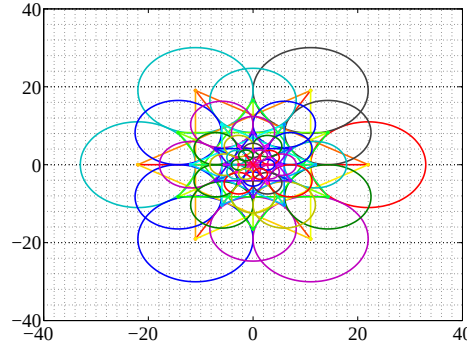


Figure 5.1: Orientation sampling grid for FREAK.

fields. For a set of sampling point pairs $\mathcal{L}$, the resultant gradient vector $\bar{g}$ is computed as follows:

$$\bar{g} = \frac{1}{|\mathcal{L}|} \sum_{(p_i, p_j) \in \mathcal{L}} (I(p_j) - I(p_i)) \frac{p_j - p_i}{\|p_j - p_i\|}. \quad (5.6)$$

The local orientation angle for the keypoint under consideration is then computed as the arctangent of the x and y components of $\bar{g}$.

5.1.3 Experimental Evaluation of Orientation Estimators

A significant design specification of our target detection system is speed. Therefore, we investigate the following three orientation estimation methods: Intensity centroid, Taylor's and FREAK's orientation assignment methods. These methods generally require the least amount of processing and are thus ideal for low-computing devices. We evaluate these methods against five test cases comprising of the following distortions: in-plane rotation, Gaussian noise, blur, and scaling. For evaluation purposes, we use the two images shown in Fig. 5.2 and synthetically transform them according to the test case.



Figure 5.2: Two test images used in the orientation evaluation experiment: (left) box, (right) puzzle.

Evaluation Criterion

The following strategy is adopted in order to measure the performance of each orientation estimation method. Keypoints are first detected on the original test image using the FAST keypoint detector. Thereafter, the initial orientation (θ'_p) for each keypoint p is computed for all three methods. This initial orientation measurement serves as the reference value to which comparisons are made against. We then transform the image as per the specifications for the test case under evaluation, and project the keypoints detected in the original image to the newly transformed image and compute the new orientations. Since we know the amount of in-plane rotation applied to each transformed image, we can compute the RMSE (root mean square error) of the estimated orientation from the transformed image to the original image as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{p=1}^N (\theta'_p + \phi - \theta_p)^2}, \quad (5.7)$$

where N is the number of keypoints, ϕ is the amount of in-plane rotation used to rotate the image, and θ'_p and θ_p is the initial and computed orientations for the original and transformed image for keypoint p respectively.

This evaluation approach enables us to strictly observe the consistency in orientation assignment, unlike the technique employed in [23], which attempts to measure the performance of the orientation estimator through a keypoint matching process. The drawback of such an approach is that the true performance of the orientation estimator is concealed behind the robustness of the feature descriptor used to evaluate the system.

Test cases

Test case 1

The first test case investigates the consistency in orientations assigned by the aforementioned algorithms in the presence of in-plane rotation only. In this case no noise or blurring is added, and no scaling is applied to the images. We simply rotate the images with 5° steps from 0° to 360° . Thus we generate a total of 72 images. This test case allows us to measure the best performance achievable by each method in ideal conditions.

Test case 2

The second test case investigates the effect of rotation in the presence of noise and blur. The 72 images of test case 1 are first blurred by convoluting them with a 5×5 Gaussian kernel.

Thereafter, Gaussian noise is added to the each image. This test case allows us to measure the consistency in orientation assignment under the presence of noise and blur.

Test case 3

The third test case investigates the scale-invariance of each orientation estimator in conjunction with in-plane rotation. The test images are created by scaling the images of test case 1 at the following scales: $[0.4, 1.6]$ with a step size of 0.1. Thus, a total of 13×72 images are created and evaluated. A scale of 1.0 represents the original size of the image, i.e. no scaling is applied.

Test case 4

The fourth test case investigates the orientation estimators under all four distortions: in-plane rotation, scaling, blur, and Gaussian noise. For this test case, we first rotate the test images, and then scale them accordingly. Thereafter, the resulting images are blurred and corrupted by adding Gaussian noise. The same distortion specifications from test cases 1 to 3 are applied to synthesize the images. This test case allows us to measure the robustness and repeatability of each method under worst case conditions.

Test case 5

As with test case 4, test case 5 also investigates the effect on the orientation estimators with respect to all four distortions. Therefore, the synthesized images of test case 4 are reused here as well.

Note that the learned model $\hat{F}$ contains features extracted at various scales. These features are computed from stable keypoints which are found by clustering keypoints which have similar spatial and orientation values for a given viewpoint bin (for a specified scale range). For this reason, we are more concerned with estimators that can approximately provide consistent orientation measurements for a similar scale range. Hence, in order to effectively evaluate the orientation estimators in this regard, we need to recompute the reference measurements θ'_p from the scaled version of the source test image. The scaling factor is the same as that used in synthesizing the test case image.

For descriptor-based approaches, such as [23, 6, 3, 20, 36, 7], which rely heavily on orientation assignment methods that are required to consistently produce the same measurements irrespective of the scale, the results of test case 3 and 4 are more significant.

5.1.4 Results

The results for the five test cases are presented in Fig. 5.3. A 31×31 region was used to compute the center of mass for the intensity centroid method (Eq. 5.2).

Test case 1. Figure 5.3(a) presents the RMSE results of the orientation estimators for the case of in-plane rotation only. The most consistent orientation assignment method, i.e. the method which yields the lowest RMSE, is the intensity centroid method which never has more than 10° in error. However, not too far behind is the FREAK orientation estimator. All three methods perform consistently relative to their mean error. Furthermore, at multiples of 90° rotation, the RMSE is zero for all three methods.

Test case 2. For the second test case, in which Gaussian noise and blurring are applied to the rotated images, the intensity centroid method once again yields the lowest RMSE error as evident in Fig. 5.3(b). It is satisfying to note that, in the presence of noise and blur, the mean RMSE error for all three methods only increases by 5° . Furthermore, as can be seen from all the results, the estimated orientations for the puzzle image (Fig. 5.2(a)) are always better than that of the box image (Fig. 5.2(b)). This can be due to the fact that the puzzle image has less symmetrical patches and is not as “busy” as the box image.

Test cases 3 & 4. The average RMSE errors for all in-plane rotations at particular scales for test cases 3 and 4 are presented in Fig. 5.3(c) and Fig. 5.3(d) respectively. It is obvious that the most consistent performer with regard to change in rotation and scale (with/without the presence of noise and blur) is the FREAK orientation estimator. Given the simplicity of Taylor’s method, it is overwhelming to see it outperform the intensity centroid method at scales less than 0.7 for the puzzle image. This is because the size of the neighborhood of the gradient vectors computed from the Bresenham circle is relatively small. Therefore, the resultant gradient vector remains consistent across the smaller scales. Given that the intensity centroid method yields the lowest RMSE errors for test cases 1 and 2, it is rather disappointing to see it perform so poorly with change in scale: a reduction in scale by 20% results in an increased RMSE of 20° . Of special interest, barring the intensity centroid method, the lowest RMSE errors for both test images are provided at a scale of $1.1 \times$ the original size of the image.

Test case 5. Figure 5.3(e) presents the results of test case 5. These results are of greater interest to us than that of test cases 3 and 4. We can see that the orientations computed at each scale remain more or less consistent with the reference orientations obtained from the corresponding scaled versions of the source test images for all three methods. From this experiment we can deduce that, amidst the three orientation estimators, the intensity centroid method yields the lowest errors.

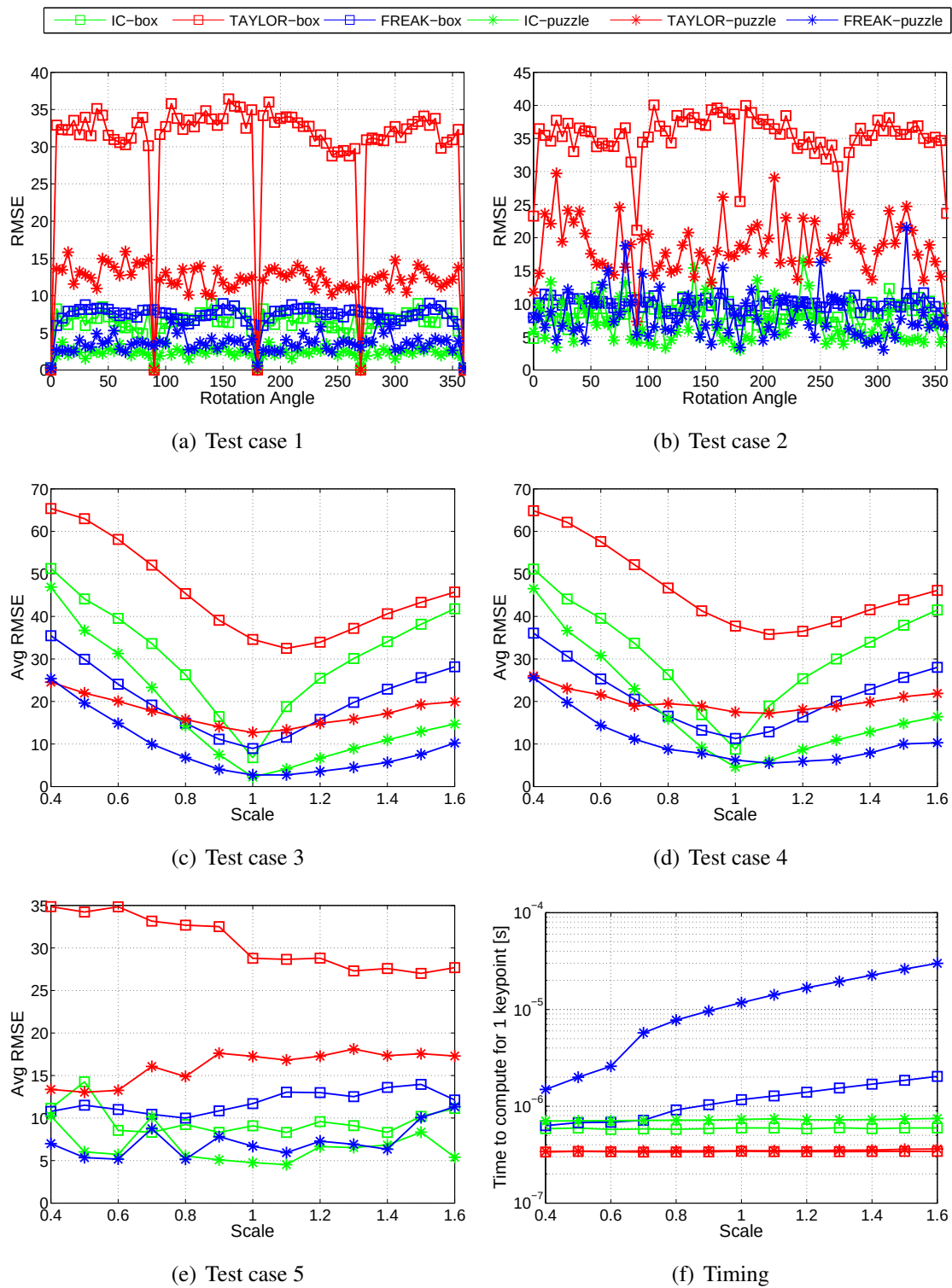


Figure 5.3: Orientation estimation evaluation results.

Timing Results

The average time required to compute the orientation for a keypoint is shown in Fig. 5.3(f). These results were obtained from a 2.67 GHz processor running Ubuntu 12.04 (32-bit). Furthermore, the timing results shown are the average times obtained from executing test cases 4 and 5. As expected, Taylor’s method results in the fastest computation time consuming less than $0.3\mu s$ irrespective of the scale of the image. At an average of $0.6\mu s$, the intensity centroid method is the second fastest of the lot. FREAK’s orientation estimator is the slowest, requiring between $1\mu s - 10\mu s$ at full scale, with time increasing almost exponentially proportional to the scale of the image.

In order to speed up keypoint matching during runtime, one may construct a lookup table of pre-computed rotated sampling grids or test patterns corresponding to discretized angle increments [39, 36, 20, 3].

5.2 Orientation Clustering

The objective of orientation clustering is to group similar orientation values for keypoints which are in spatial proximity to one another. These clusters will then be used in the subsequent stages of the model generation framework for identifying the most stable keypoints (Chapter 6) and for extracting and aggregating partial descriptors. To this end, we propose two characteristically diverse methods to perform orientation clustering. The first method is based on an agglomerative hierarchical clustering approach in which a graph is used to facilitate the convergence rate. The second method adopts a probabilistic approach which requires estimating the probability density function (pdf) of the orientations.

5.2.1 Graph-based Hierarchical Clustering

Most clustering algorithms are efficient and conceptually simple to implement, however, they carry with them a number of drawbacks. One such limiting factor is the requirement of a priori knowledge pertaining to the number of clusters that are to be found. Hierarchical clustering algorithms (HCA), although computationally intensive, do not require us to pre-specify the number of clusters.

We propose an agglomerative approach where initially each data point is considered as a potential cluster. Clustering is then performed by merging pairs of clusters at each iteration until there is a single large cluster (or until the desired number of clusters have been found).

Algorithm 5.1 Agglomerative Hierarchical Clustering Algorithm.

Input: $\Theta = \{\theta_i\}_{i=1,\dots,n}$: data points

Output: $\mathcal{C}$: clusters

```

1  $\mathcal{C}_i = \{\theta_i\}_{i=1,\dots,n}$ 
2  $\text{cnt} \leftarrow n$ 
3 while  $\text{cnt} \neq 1$  do
4   Find nearest clusters  $\mathcal{C}_i$  and  $\mathcal{C}_j$ 
5   Merge clusters  $\mathcal{C}_i$  and  $\mathcal{C}_j$ 
6    $\text{cnt} \leftarrow \text{cnt} - 1$ 
7 return  $\mathcal{C}$ 

```

The general agglomerative nearest-neighbor hierarchical clustering algorithm is presented in Algorithm 5.1. A by-product of the HCA algorithm is a hierarchical tree like structure, referred to as a dendrogram, which can be used to extract any desired number of clusters. The minimum number of iterations required to generate the complete dendrogram for n data points is $n - 1$. This is quite expensive seeing that the clusters at each subsequent hierarchical level need to be compared to each and every other cluster in order to perform the merge.

To increase the convergence rate and overcome the inefficiencies posed by the HCA, we make use of a graph. The primary motivation for using a graph for the clustering process is based on the fact that graphs implicitly encode the distances to each node in their structure. There are three types of graphs that can be created. The *k-nearest neighbor graph* connects a vertex v_i with vertex v_j if v_j is among the k -nearest neighbors of v_i . The *fully connected graph* simply connects all vertices to each other, whereas the *ϵ -neighborhood graph* connects all points whose pairwise distances are smaller than ϵ .

Since we do not know a priori how many neighbors (k) to associate with each vertex, and, the fully connected graph is computationally demanding, we use the ϵ -neighborhood graph. It is important to take note of the fact that ϵ allows us to explicitly control the maximum deviation from the medoid of a cluster to all its nodes.

Graph Notation

The following graph notation is used in this thesis. Let $G_\epsilon = (V, E)$ be an undirected, weighted graph with vertex set $V = \{v_1, \dots, v_n\}$. The edge between two vertices v_i and v_j is weighted by w_{ij} , with $w_{ij} < 0$ implying that the vertices v_i and v_j are disconnected. The weighted adjacency matrix of the graph is represented by the matrix $W = [w_{ij}]_{i,j=1,\dots,n}$. Since G_ϵ is

undirected, we require W to be symmetric, i.e. $w_{ij} = w_{ji}$. For convenience we introduce the shorthand notation $i \in A$ for the set of indices in $\{i | v_i \in A\}$.

Graph-based Hierarchical clustering method

For a given set of orientation values, $\Theta = \{\theta_i\}_{i=1,\dots,n}$, we construct an ϵ -neighborhood graph G_ϵ with n vertices at the corresponding θ_i . The adjacency matrix W is then computed by representing the weights of the edges connecting two arbitrary vertices, v_i and v_j , as the distance between them as follows:

$$w_{ij} = \begin{cases} D(v_i, v_j) & \text{if } D(v_i, v_j) \leq \epsilon \\ -1 & \text{otherwise} \end{cases}, \quad (5.8)$$

where

$$D(v_i, v_j) = 180^\circ - |180^\circ - |v_i - v_j||. \quad (5.9)$$

We then find a vertex v_A which has the largest number of connected vertices. The group of vertices connected to v_A forms a subset $A \subseteq V$. If the size of subset $|A| \leq 1$, the algorithm is terminated. This can happen if either $V = \emptyset$, or $w_{ij} = -1 \forall i, j \in V : i \neq j$. On the other hand, it is possible that multiple vertices can be found to have the same number of connections. In these cases, contention is resolved by selecting the vertex with the smallest degree. The degree of a vertex v_i is computed as:

$$\text{degree}(v_i) = \sum_{j \in A} w_{ij}, \quad (5.10)$$

where A represents the subset of vertices connected to v_i . Intuitively, such a node is preferable since it represents a closely connected subset of neighbors. After finding the best vertex v_A , we then merge all vertices which are in A and disassociate other vertices which are connected to them. Subset A is then removed from G_ϵ and $V = V \setminus A$. The process is repeated until no further vertices can be grouped ($|A| \leq 1$). Algorithm 5.2 summarizes the proposed approach.

Convergence Rate

At each iteration, we always select the vertex which encompasses the largest subset, therefore, at each subsequent iteration, the graph is drastically reduced in size. This allows a much faster convergence rate. In general, the expected number of iterations required by the algorithm is $360^\circ/\epsilon$. Note that the distances to each node is only calculated once. Since W is symmetric, for

Algorithm 5.2 Graph-based Hierarchical Clustering Algorithm.

Input: $\Theta = \{\theta_i\}_{i=1,\dots,n}$: orientation values, ϵ : maximum distance to neighboring vertex**Output:** $\mathcal{C}$: clusters

```

1 Construct  $\epsilon$ -neighborhood graph  $G_\epsilon = (V, E)$  with  $V_i = \{\theta_i\}_{i=1,\dots,n}$ 
2 Compute adjacency matrix  $W$  using Eq. 5.8
3  $\mathcal{C}_i = \{\theta_i\}_{i=1,\dots,n}$ 
4 while (true) do
5   Find a vertex  $v_A$  which has the most number of connected vertices
6    $A \leftarrow$  all vertices connected to  $v_A$ 
7   if ( $|v_A| == 0$ ) or ( $|A| \leq 1$ ) then
8     break
9   else if ( $|v_A| > 1$ ) then
10    Select the contending vertex with the smallest degree (Eq. 5.10)
11    Merge clusters  $\mathcal{C}_i \in A \subseteq V$ 
12    Remove vertices  $v_i \in A$  from  $V$  ( $V \leftarrow V \setminus A$ ) and update  $W$  accordingly
13 return  $\mathcal{C}$ 

```

n orientation values, we only need to compute $n(n-1)/2$ distances at the first iteration. Furthermore, each time a subset A is removed, W is adjusted by removing the rows and columns corresponding to the vertices in A .

To empirically verify the convergence rate of the proposed algorithm, we randomly sampled 1000 angles from $[-180^\circ, 180^\circ]$ and computed the average number of iterations required to cluster the angles. This was done for a 1000 simulations for three different values of $\epsilon = \{10^\circ, 20^\circ, 30^\circ\}$. In practice, we will typically never need to cluster more than 100 orientation values, therefore, we repeated the same experiment for 100 orientation values as well. The result of both simulation runs are presented in Fig. 5.4. Figure 5.4(a) shows that for the worst case scenario of 1000 orientation values, with $\epsilon = 10^\circ$, we are able to cluster the orientations in less than 40 iterations. At $\epsilon = 20^\circ$, the average number of iterations required to perform the clustering is reduced to less than 20 (reduction factor of 2). For the more realistic case of 100 orientation values, no more than 20 iterations are required on average at worst (Fig. 5.4(b))

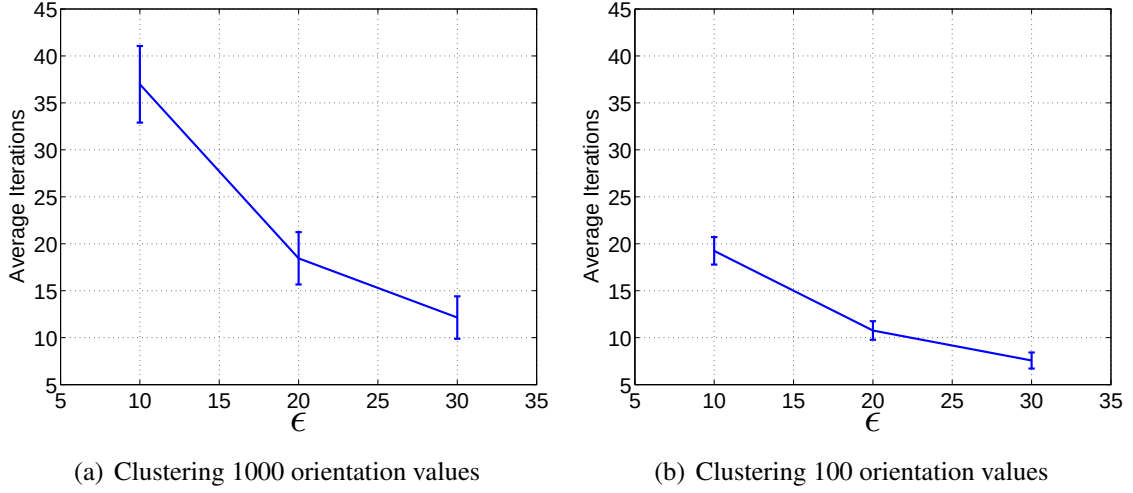


Figure 5.4: Convergence rates of the proposed graph-based hierarchical clustering algorithm.

5.2.2 Probabilistic Clustering

It is desirable to develop a clustering method which takes advantage of the underlying distribution of the orientation estimator's response. By doing so, we are able to qualitatively express the goodness of each cluster. Thus, we are in a position to make an informed decision as whether to accept or reject certain cluster(s). We apply kernel density estimation (KDE) to estimate the probability density function (pdf) of the orientation values. Since the orientation values are univariate, and constrained, the cost of using KDE is rather insignificant.

In kernel density estimation, a kernel function is generated around each observed data point θ_i . An estimate of the density function, $f(\theta)$, is found by adding these kernel functions together and scaling the sum:

$$f(\theta) = \frac{1}{V} \sum_{i=1}^n K(\theta - \theta_i), \quad (5.11)$$

where $K(\cdot)$ is the kernel function around θ_i and V is a normalization factor ensuring that the area under $f(\theta)$ is 1. For n data points, $V = n \times k_b$, where k_b denotes the kernel bandwidth. There are many kernels that can be used, of these the most common include the standard Gaussian, uniform and the optimal Epanechnikov kernels.

Before we introduce the proposed probabilistic clustering algorithm, we need to understand an important aspect pertaining to the orientation values. An orientation value is in essence an angle measurement which is periodic at 360° . Therefore, the true absolute difference between 5° and 355° , is 10° and not 350° . This becomes relevant when we estimate the pdf of the measurements. For instance, if we do not take into account the periodicity of angles, and blindly

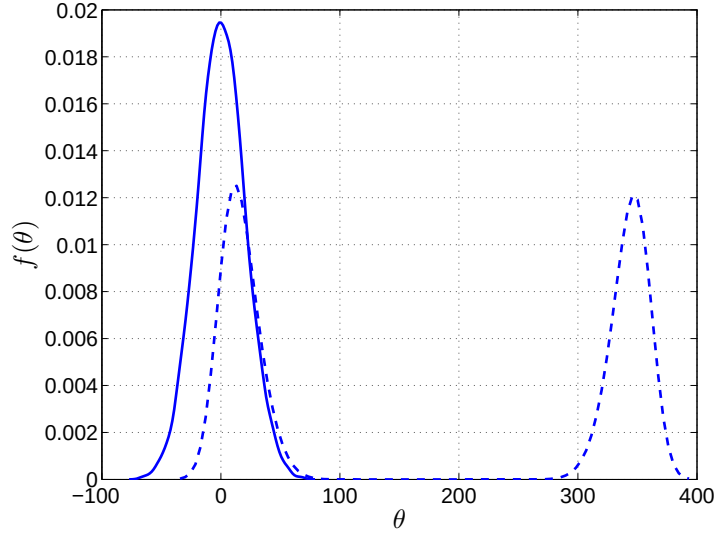


Figure 5.5: Estimating the probability distribution function of angles. The solid and dashed lines denote the estimated pdfs with and without considering the periodicity of angles respectively.

cluster the values. Then for the case when we have orientation values in the range $[-20^\circ, 20^\circ]$, we will falsely estimate a bimodal density function with one mode between $[0^\circ, 20^\circ]$ and another one between $[340^\circ, 360^\circ]$. This is illustrated in Fig. 5.5 by the dashed lines.

This can be overcome by analyzing the mean of the orientation values. If the absolute value of the mean is less than 90° , the majority of the samples are in the first and fourth quadrants, then the orientation angles are expressed in the range $[-180^\circ, 180^\circ]$. On the contrary, if the absolute value of the mean is greater than 90° , the majority of the samples are in the second and third quadrants, then the orientation angles are expressed in the range $[0^\circ, 360^\circ]$. The solid line in Fig. 5.5 illustrates the correctly estimated pdf.

With this in mind, we now present the proposed probabilistic clustering method.

Probabilistic clustering method

We first determine the mean value, μ , for a given set of orientation values $\Theta = \{\theta_i\}_{i=1,\dots,n}$. If $|\mu| \leq 90^\circ$, express Θ in the range $[-180^\circ, 180^\circ]$, otherwise, express it in the range $[0^\circ, 360^\circ]$. Since angles are periodic, the mean value of Θ is computed in the Cartesian plane as follows:

$$\mu = \text{atan2}(\mu_y, \mu_x) , \quad (5.12)$$

where atan2 is the four-quadrant inverse tangent function, and

$$(\mu_x, \mu_y) = \left(\frac{1}{n} \sum_{i=1}^n \cos \theta_i, \frac{1}{n} \sum_{i=1}^n \sin \theta_i \right). \quad (5.13)$$

The probability density function $f(\theta)$ of Θ is then estimated using KDE as described by Eq. 5.11. Thereafter, the mode, θ_m , of $f(\theta)$ is found. Note that if $\Theta \sim N(\mu, \sigma^2)$, then $\theta_m \approx \mu$. We then form a cluster by grouping every θ which has an absolute distance of less than ϵ to θ_m . As with the graph-based hierarchical clustering algorithm, ϵ allows us to explicitly control the maximum deviation from the mode of the cluster to all its nodes. The distance between two orientation values θ_i and θ_j is computed using Eq. 5.9.

The probability of observing orientation values in the range $[\theta_m - \epsilon, \theta_m + \epsilon]$ is then calculated

$$\Pr(\theta_m - \epsilon \leq \theta \leq \theta_m + \epsilon) \approx \sum_{i=\theta_m-\epsilon}^{\theta_m+\epsilon} f(\theta_i). \quad (5.14)$$

If the computed probability is greater than a minimum probability threshold ρ , the cluster is accepted, otherwise it is discarded. We can extend Eq. 5.14 so that we not only search for good clusters around the region enclosing θ_m , but also regions extending to the left and right of it by shifting Eq. 5.14 by a factor of 2ϵ :

$$\Pr(j, \theta) = \Pr(\theta_m - \epsilon \leq \theta - 2j\epsilon \leq \theta_m + \epsilon), \forall j \in \mathbb{Z} : |j| \leq N_c \leq \frac{360^\circ}{\epsilon}, \quad (5.15)$$

where N_c represents the desired number of regions we wish to search. Note that N_c is bounded by the choice of ϵ . The number of clusters found is ultimately dependent on ρ . The lower ρ is, the likelihood of retaining more clusters is high. On the other hand, a high value for ρ ensures less, yet better quality clusters. In the worst case, when Θ is approximately uniformly distributed, the probabilities obtained via Eq. 5.15 should be relatively low. Since a uniformly distributed Θ indicates inconsistent orientation values, by setting ρ slightly higher allows us to reject such weak clusters.

The probabilistic clustering approach is summarized in Algorithm 5.3.

Kernel Density Estimation

The following kernel functions were investigated in order to determine the most efficient kernel for the density estimation process:

- Uniform kernel ($K(\theta) = \frac{1}{2}, \forall |\theta| \leq 1$),

Algorithm 5.3 Probabilistic Clustering Algorithm.

Input: $\Theta = \{\theta_i\}_{i=1,\dots,n}$: orientation values, ϵ : maximum distance from mode, N_c : desired number of clusters, ρ : minimum probability threshold

Output: $\mathcal{C}$: clusters

```

1  $\mathcal{C} \leftarrow \emptyset$ 
2 Compute the mean,  $\mu$ , of  $\Theta$  (Eq. 5.12)
3 if  $|\mu| \leq 90^\circ$  then
4   Express  $\Theta$  in  $[-180^\circ, 180^\circ]$ 
5 Estimate the probability density function,  $f(\theta)$ , of  $\Theta$  (Eq. 5.11)
6 Find the mode,  $\theta_m$ , of  $f(\theta)$ 
7  $i \leftarrow \frac{N_c-1}{2}$ 
8 for  $j = -i$  to  $i$  do
9   Compute  $\Pr(j, \theta)$  (Eq. 5.15)
10  if  $\Pr(j, \theta) \geq \rho$  then
11     $\mathcal{C}' \leftarrow$  Group all  $\theta \in [\theta_m - \epsilon(1 - 2j), \theta_m + \epsilon(1 + 2j)]$ 
12     $\mathcal{C} \leftarrow \mathcal{C} + \mathcal{C}'$ 
13 return  $\mathcal{C}$ 

```

- Gaussian kernel $\left(K(\theta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\theta^2}{2}}\right)$, and
- Epanechnikov kernel $\left(K(\theta) = \frac{3}{4}(1 - \theta^2), \forall |\theta| \leq 1\right)$.

A 1000 orientation values, Θ , were generated using a Gaussian distribution with two different means ($\mu_1 = 150^\circ, \mu_2 = 330^\circ$) and two different standard deviations ($\sigma_1 = 10^\circ, \sigma_2 = 20^\circ$). In total, we generated four sets of Θ for each permutation of μ and σ . Each kernel was then used to estimate the densities of the four sample sets. Figure 5.6 presents the results of the estimated pdfs. It is apparent that the Gaussian and Epanechnikov kernels produce quite similar and accurate density estimates. Given the simplicity of the Uniform kernel, apart from the lack of smoothness, it performs on par with the other two kernels. Furthermore, the lack of smoothness in the Uniform kernel is of no particular concern to the probabilistic clustering algorithm since we only use $f(\theta)$ to compute the probability over a region which is then compared to a threshold ρ . Therefore, to accelerate the model generation process, it was decided to use the Uniform kernel which is relatively accurate, and more importantly, extremely fast to compute.

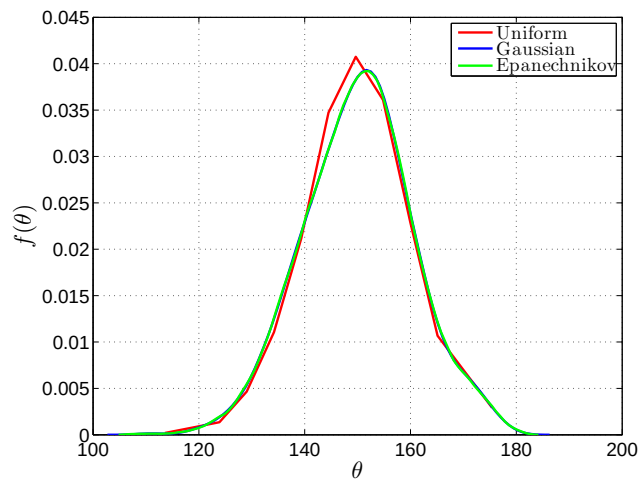
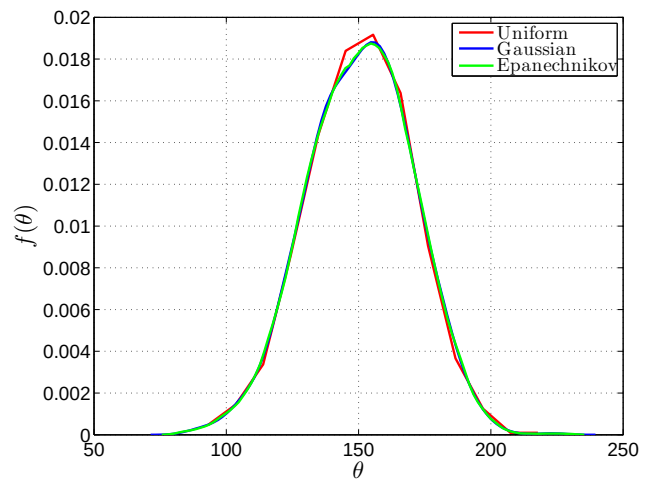
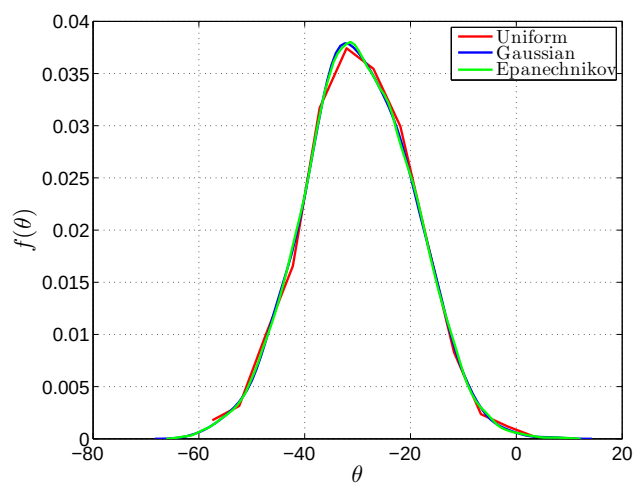
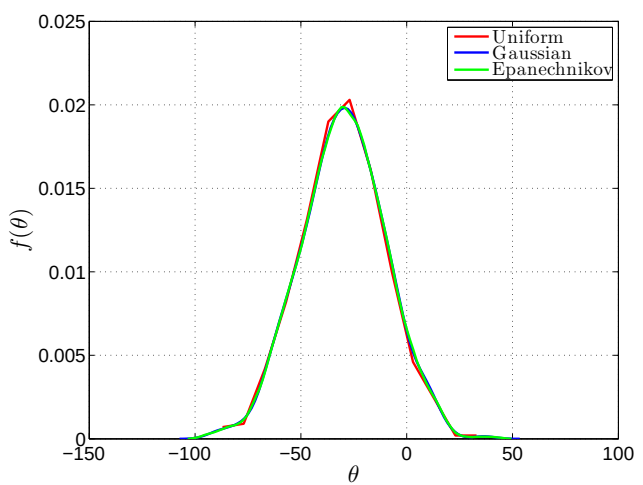
(a) $\Theta \sim N(\mu = 150^\circ, \sigma = 10^\circ)$ (b) $\Theta \sim N(\mu = 150^\circ, \sigma = 20^\circ)$ (c) $\Theta \sim N(\mu = 330^\circ, \sigma = 10^\circ)$ (d) $\Theta \sim N(\mu = 330^\circ, \sigma = 20^\circ)$

Figure 5.6: Comparison of various kernels for kernel density estimation.

5.3 Conclusion

In this chapter we have described orientation normalization, which is the process of normalizing a feature with respect to its orientation. Orientation normalization allows us to obtain invariance to in-plane rotation deformations. The amount of invariance achieved is determined by the consistency of the orientation estimator: although an accurate estimator is desirable, it adds too much computational burden on the low-powered hand-held device, thus preventing real-time matching. On the other hand, cheap orientation estimators are generally sensitive to noise and pose a threat of negating the advantage offered by orientation normalized features.

To solve this problem, we have proposed applying orientation clustering to group similar orientation values for features which are in spatial proximity to one another. The objective of the orientation clustering process is to limit the appearance variation of the region encompassing the groups of features, thus allowing for better descriptors to be extracted. These clusters will then be used in the subsequent stages of the model generation framework for identifying the most stable keypoints and for extracting and aggregating partial descriptors (explained in the next chapter). To this end, we have developed two characteristically diverse methods to perform orientation clustering. The first method is based on an agglomerative hierarchical clustering approach in which a graph is exploited to facilitate the convergence rate. The second method adopts a probabilistic approach which requires estimating the probability density function of the orientations.

In the following chapter we will describe the stable keypoint identification and feature extraction processes of the model generation framework. As will be seen shortly, the orientation clustering component plays a significant role in identifying stable keypoints.

Chapter 6

Stable Keypoint Identification & Feature Extraction

After detecting keypoints from the virtual images for a given viewpoint bin, we are now left with the task of extracting features to describe them. However, before we can extract these features, there are two important issues which have to be taken into consideration. Firstly, from all the virtual images, we need to identify which of the detected keypoints are related, i.e., which of them reference the same real-world point. Secondly, there exist some keypoints which are not repeatable, and should therefore not be included in the model database $\hat{F}$. Non-repeatable keypoints are typically those points which are only detectable under certain viewpoints. In addition, such keypoints can also arise from artifacts such as sensor noise and are thus not interesting.

By combining the first and second issues, we can define the concept of stable keypoints. A stable keypoint is a point which is repeatably detected, with a consistent local orientation, across variously deformed images. We are only interested in extracting features for such detected points.

We measure the stability of keypoints by performing a two stage clustering process: *spatial clustering* and *orientation clustering*. The orientation clustering process was addressed in Chapter 5. We also develop two techniques for selecting stable keypoints. The first method considers keypoints which are locally stable, whereas the second method considers keypoints which are globally stable. Furthermore, to increase the model's robustness to occlusion, stratified sampling is employed.

Once we have identified the stable keypoints, we proceed to extract the descriptors. To ensure that we only add those features to $\hat{F}$ which are meaningful and reliable, we propose a

novel method for verifying the feature’s stability in terms of its entropy. Note that the feature stability verification process is independent from the stable keypoint identification stage.

Finally, we apply the model generation framework to several binary feature descriptor algorithms in order to increase their robustness to scale and perspective viewpoint deformations.

6.1 Identifying Stable Keypoints

6.1.1 Spatial Clustering

A stable keypoint comprises of two fundamental components: a point $\hat{p}_r$ on the reference image, and a group of coincident keypoints, G_r , which map back to $\hat{p}_r$. Given an array of $\mathcal{M}$ sets of detected keypoints from the virtual images, $\bar{\mathcal{K}} = \{\mathcal{K}_1, \dots, \mathcal{K}_{\mathcal{M}}\}$, we need to establish which group of keypoints reference the same point.

Since the virtual images are generated by different transformations we cannot compare the detected keypoints directly. Hence, we first project the keypoint’s spatial coordinate from I_m to $\hat{I}$ by computing the inverse mapping of Eq. 3.1. Thereafter, we find sets of clustered coincident keypoints, $\mathbb{G} = \{G_1, G_2, \dots\}$, which approximately map back to the same reference positions $\hat{\mathbf{p}} = \{\hat{p}_1, \hat{p}_2, \dots\}$. This is done by computing the Euclidean distance between the projected points; if the distance is less than a threshold $\delta_{\mathcal{K}}$, then the points are considered to be close enough to be clustered together in the same group. Note that the number of candidate stable keypoints is therefore determined by $|\mathbb{G}| = |\hat{\mathbf{p}}|$. The spatial clustering process is presented in Algorithm 6.1.

Depending on the value of $\delta_{\mathcal{K}}$, we can have keypoints which are associated to multiple groups. For example, consider the hypothetical scenario in Fig. 6.1, we see that the keypoint p_{mk} :

- is associated to reference point $\hat{p}_2$ only for $\delta_{\mathcal{K}} = \delta_{\mathcal{K}_1}$,
- is associated to reference points $\hat{p}_1, \hat{p}_2$ and $\hat{p}_3$ for $\delta_{\mathcal{K}} = \delta_{\mathcal{K}_2} > \delta_{\mathcal{K}_1}$.

We found $\delta_{\mathcal{K}} = 2$ to produce the best clustering results.

6.1.2 Stability: Local vs Global

There are two possible ways of identifying stable keypoints. The first way is to select stable keypoints local to the viewpoint bin under consideration (local stability). The second way is to pre-determine the global stable keypoints using a separate set of artificially deformed images

Algorithm 6.1 Spatial Clustering Algorithm.

Input: $\bar{\mathcal{K}} = \{\mathcal{K}_1, \dots, \mathcal{K}_{\mathcal{M}}\}$: array of $\mathcal{M}$ detected keypoints, $\delta_{\mathcal{K}}$: minimum distance to neighboring point

Output: $\mathbb{G}$: group of coincident keypoints, $\hat{\mathbf{p}}$: reference points associated with each group

```

1  $\mathbb{G} \leftarrow \emptyset$ 
2  $\hat{\mathbf{p}} \leftarrow \emptyset$ 
3 for  $m = 1$  to  $\mathcal{M}$  do
4   for  $k = 1$  to  $|\mathcal{K}_m|$  do
5      $\hat{p}_{mk} \leftarrow$  Project keypoint  $p_{mk}$  from  $I_m$  to  $\hat{I}$  using inverse mapping of Eq. 3.1
6      $\mathcal{Y} \leftarrow 1$ ;
7     for  $r = 1$  to  $|\hat{\mathbf{p}}|$  do
8       if  $(\|\hat{p}_r - \hat{p}_{mk}\| \leq \delta_{\mathcal{K}})$  then
9          $G_r \leftarrow G_r \cup p_{mk}$ 
10         $\mathcal{Y} \leftarrow 0$ ;
11   if  $(\mathcal{Y} == 1)$  then
12      $\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} \cup \hat{p}_{mk}$ 
13      $r \leftarrow |\hat{\mathbf{p}}|$ 
14      $G_r \leftarrow p_{mk}$ 
15      $\mathbb{G} \leftarrow \mathbb{G} \cup G_r$ 
16 return  $\mathbb{G}, \hat{\mathbf{p}}$ 

```

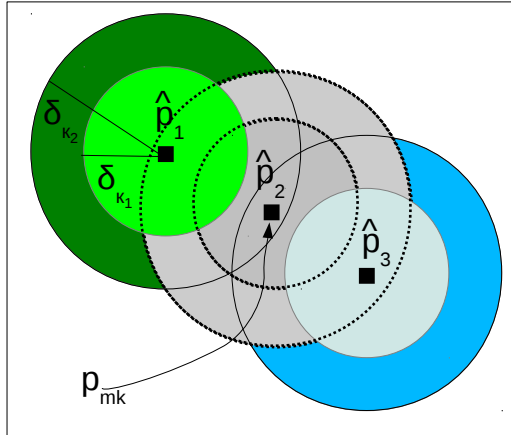


Figure 6.1: Hypothetical scenario illustrating the effect of $\delta_{\mathcal{K}}$.

(global stability). Both of them employ the same procedure of applying spatial clustering, followed by orientation clustering. After clustering the keypoints, the groups, $G_r \in \mathbb{G}$, are then ranked according to their size $|G_r|$. Thereafter the Q most highly ranked groups are selected as the final set of stable keypoints. It is from these Q stable keypoints that descriptors will be extracted and later added to the model database $\hat{F}$.

Local Stability

The local stability method for identifying stable keypoints is based on clustering keypoints which are detected from virtual images for each viewpoint bin. Hence the term “local” which implies that the stable keypoints are local to the viewpoint bin. The process is presented in Algorithm 6.2.

The input to the algorithm comprises of a set of pre-generated viewpoint images $\bar{I}_n = \{I_{1n}, \dots, I_{Mn}\}$ from the n^{th} viewpoint bin. The algorithm begins by detecting the keypoints from each image $I_{mn} \in \bar{I}_n$. Thereafter, spatial clustering is performed on the set of detected keypoints. The result of this stage is an initial clustering of the keypoints (denoted as $\mathbb{G}'_n$). For each spatial group, $G'_{nr} \in \mathbb{G}'_n$, we compute the local orientation of the keypoints and then project them back on to the reference image to obtain $\hat{\Theta}$. This is an important step, since the virtual images are generated by different transformations, the estimated orientations need to be projected onto a common reference axis from which comparisons can be made. The reference axis is chosen to be that of the original un-deformed reference image $\hat{I}$.

Each spatial group is then partitioned (if necessary) by clustering the orientation values ($\hat{\Theta}$) such that the groups of keypoints maintain a consistent set of orientations. This can result in multiple partitions of the same spatial group depending on the deviation of the orientation values. The objective of the orientation clustering stage is to limit the appearance variation of the region encompassing the groups of keypoints, thus allowing for better descriptors to be extracted. This stage can be omitted if a highly accurate orientation estimator is applied since, for such an estimator, the orientations should be relatively consistent irrespective of how deformed the images are. However, an accurate orientation estimator is typically computationally intensive and is not well suited for real-time matching on low powered devices. Thus, we transfer the burden of accurate orientation estimation onto the training stage by permitting the use of cheap orientation estimators.

Algorithm 6.2 Identifying Locally Stable Keypoints.**Input:** $\bar{I}_n = \{I_{1n}, \dots, I_{Mn}\} : M$ virtual images for viewpoint bin n **Output:** $\mathbb{G}_n$: group of locally stable keypoints for viewpoint bin n

```

1  $\bar{\mathcal{K}} \leftarrow \emptyset$ 
2 for  $m = 1$  to  $M$  do
3    $\mathcal{K}_m \leftarrow \text{Detect keypoints from } I_{mn}$ 
4    $\bar{\mathcal{K}} \leftarrow \bar{\mathcal{K}} \cup \mathcal{K}_m$ 
5  $\mathbb{G}'_n \leftarrow \text{Cluster keypoints } \bar{\mathcal{K}} \text{ spatially using Algorithm 6.1}$ 
6  $\mathbb{G}_n \leftarrow \emptyset$ 
7 foreach  $G'_{nr} \in \mathbb{G}'_n$  do
8    $\Theta \leftarrow \text{Estimate local orientations for each keypoint in } G'_{nr}$ 
9    $\hat{\Theta} \leftarrow \text{Project } \Theta \text{ to reference axis using inverse mapping of Eq. 3.1}$ 
10   $\mathcal{C} \leftarrow \text{Cluster orientation values } \hat{\Theta}$ 
11   $G^\dagger \leftarrow \text{Partition } G'_{nr} \text{ based on } \mathcal{C}$ 
12   $\mathbb{G}_n \leftarrow \mathbb{G}_n \cup G^\dagger$ 
13 return  $\mathbb{G}_n$ 

```

Global Stability

The global stability method for identifying stable keypoints is based on clustering keypoints which are detected from an additional global set of virtual images. Hence the term “global” which implies that the stable keypoints are global to all viewpoints. This is in contrast to the local stability method which only identifies stable keypoints from the viewpoint bin under consideration. In a way, the global stability method can be regarded as a special case of the local stability method in which the N viewpoint bins are integrated into a single “super” bin which specifies the complete range of deformations. It is from this “super” viewpoint bin that the global stable keypoints are identified. Algorithm 6.3 summarizes the global stability method.

The algorithm requires two input parameters: the reference image $\hat{I}$ and a parameter $\mathbb{M}$ which specifies the number of images to be generated. We begin by first generating the deformed images of $\hat{I}$ using the viewset generation process described in Chapter 3. The viewset specifications are that of the entire range of angle and scale deformations for which the target is to be localized. As each image is generated, the keypoints are detected using the same keypoint detector that is to be used at runtime. After all $\mathbb{M}$ images have been processed, spatial clustering is then performed using the sets of detected keypoints $\bar{\mathcal{K}}$.

Algorithm 6.3 Identifying Globally Stable Keypoints.**Input:** $\hat{I}$: reference image, $\mathbb{M}$: number of images to generate**Output:** $\mathbb{G}$: group of globally stable keypoints

```

1  $\bar{I} \leftarrow \emptyset$ 
2  $\bar{\mathcal{K}} \leftarrow \emptyset$ 
3 for  $m = 1$  to  $\mathbb{M}$  do
4    $I \leftarrow$  Generate synthetic image using Eq. 3.1
5    $\bar{I} \leftarrow \bar{I} \cup I$ 
6    $\mathcal{K}_m \leftarrow$  Detect keypoints from  $I$ 
7    $\bar{\mathcal{K}} \leftarrow \bar{\mathcal{K}} \cup \mathcal{K}_m$ 
8  $\mathbb{G}' \leftarrow$  Cluster keypoints  $\bar{\mathcal{K}}$  spatially using Algorithm 6.1
9  $\mathbb{G} \leftarrow \emptyset$ 
10 foreach  $G_r \in \mathbb{G}'$  do
11    $\Theta \leftarrow$  Estimate local orientations for each keypoint in  $G_r$ 
12    $\hat{\Theta} \leftarrow$  Project  $\Theta$  to reference axis using inverse mapping of Eq. 3.1
13    $\mathcal{C} \leftarrow$  Cluster orientation values  $\hat{\Theta}$ 
14    $G_r^\dagger \leftarrow$  Partition  $G_r$  based on  $\mathcal{C}$ 
15    $\mathbb{G} \leftarrow \mathbb{G} \cup G_r^\dagger$ 
16 return  $\mathbb{G}$ 

```

For each spatial group, $G_r' \in \mathbb{G}'$, we compute estimates of the local orientation of the keypoints and then project them back on to the reference image to obtain $\hat{\Theta}$. Each spatial group is then refined by clustering the orientation values ($\hat{\Theta}$) such that the groups of keypoints maintain a consistent set of orientations. Once again, this can result in multiple partitions of the same spatial group depending on the deviation of the orientation values and the selection of the orientation clustering algorithm along with its parameters.

6.1.3 Stratified Sampling

Up until now, we have ranked the stable keypoint groups according to their size, and have selected the Q most highly ranked groups as the final set of stable keypoints. This strategy works exceptionally well for relatively large values of Q , where we are allowed a large budget to select many stable keypoints. However, for smaller Q , we encounter the risk of only selecting stable keypoints from a dense high-scoring region of the image, increasing the system's

vulnerability to occlusion.

To address this issue, we apply stratified sampling to select the stable keypoints across the entire image thereby improving the system's robustness. Stratified sampling is a sampling method for selecting data points from a population by maintaining some characteristics of the data set within its sampled subsets. This is done by partitioning the dataset into a number of mutually exclusive subsets, each of which is representative of some aspect of the population. In our framework, stratified sampling is performed by tessellating the reference image into $W \times W$ subwindows. We then select the Q/W^2 most highly ranked stable keypoints from each subwindow. This ensures that we select stable keypoints from all segments of the image.

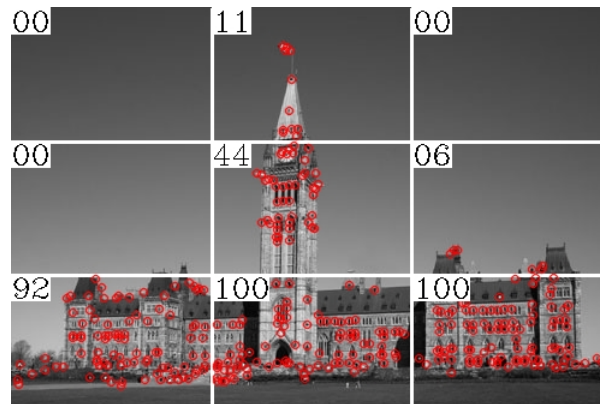
It might happen that when we partition the reference image into $W \times W$ subwindows, some windows may contain little to no keypoints. The following redistribution strategy is proposed to re-allocate those window's stable keypoints to the others in order to obtain the desired Q stable keypoints per viewpoint bin:

1. Calculate the sum of the total number of stable keypoints that are to be redistributed across all the empty subwindows in the image. This sum is denoted as Σ .
2. Compute the fraction of stable keypoints in each subwindow to the total number of stable keypoints, and multiply that with Σ . This quantity is denoted as Υ .
3. Compute the total number of stable keypoints to be selected from each subwindow as:

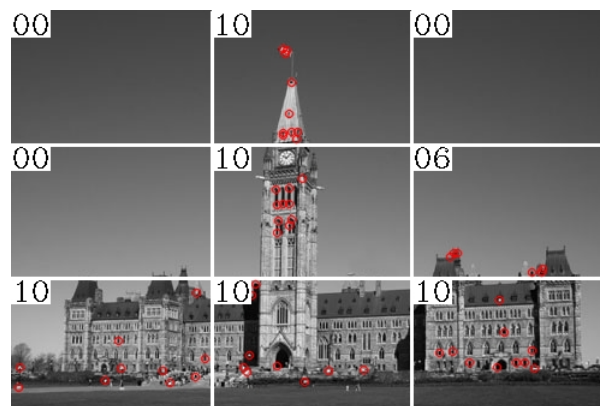
$$Q_w = Q/W^2 + \Upsilon.$$

Figure 6.2 illustrates the redistribution strategy for $W = 3$ and $Q = 90$. In this example, there are a total of 354 stable keypoints. Without using the redistribution strategy, Fig. 6.2(b) presents typical sampling results. Note that we only select 56 stable keypoints where the desired is 90. Figure 6.2(c) shows the stratified sampling results with the redistribution strategy.

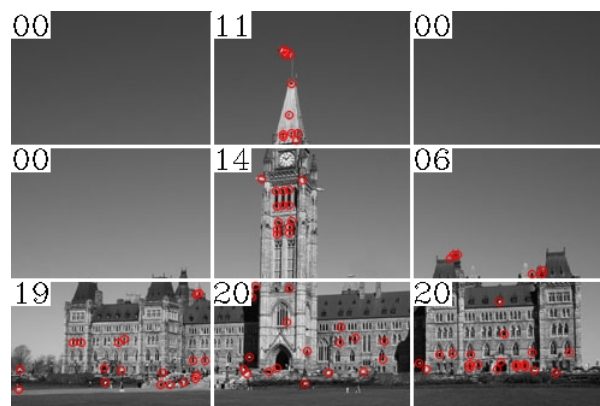
Fig. 6.3 demonstrates the stratified sampling technique in conjunction with the orientation clustering algorithms using the local stability method to determine the stable keypoints. The first ten virtual images of the viewpoint bin are shown in Fig. 6.3(a). Clustering with the Graph-based Hierarchical Clustering Algorithm ($\epsilon = 20^\circ$) and Probabilistic Clustering Algorithm ($\epsilon = 20^\circ$, $N_c = 7$, and $\rho = 0.3$) are shown in Fig. 6.3(b) and Fig. 6.3(c) respectively. In both Fig. 6.3(b) and Fig. 6.3(c), the left image presents the set of keypoints, along with their estimated orientations, which have been mapped back on to the reference image. The middle image illustrates the orientation clustering process, whereby only the central orientation of each subset is shown. This corresponds to vertices v_A in Algorithm 5.2 and to modes θ_m in



(a)



(b)



(c)

Figure 6.2: Stratified sampling, $W = 3$ and $Q = 90$. (a) Total of 354 stable keypoints are detected. (b) Stratified sampling without redistribution strategy. (c) Stratified sampling with redistribution strategy.

Algorithm 5.3. The right image shows the final selected stable keypoints (after stratified sampling). It is pleasing to note that most of the final stable keypoints selected by both algorithms are quite similar. This confirms that both algorithms are able to find keypoints which are not only spatially repeatable, but also maintain consistent orientation values.

6.2 Feature Extraction & Aggregation

After the stable keypoints have been identified, all that is left to do is to extract the features associated with each keypoint for each stable group $p_{mk} \in G_r \in \mathbb{G}$. We refer to the individual features, extracted from the keypoints $p_{mk} \in G_r$, as partial features. The final feature which describes the stable keypoint group is obtained by aggregating all the corresponding partial features.

For most binary feature descriptors, aggregation can be performed by a simple majority vote on each bit of the feature set. For other binary feature descriptors which make use of histograms, instead of a majority vote, a cut-off threshold can be applied to identify rarely occupied bins. Floating-point feature descriptors can be aggregated by computing the expectation over each feature dimension.

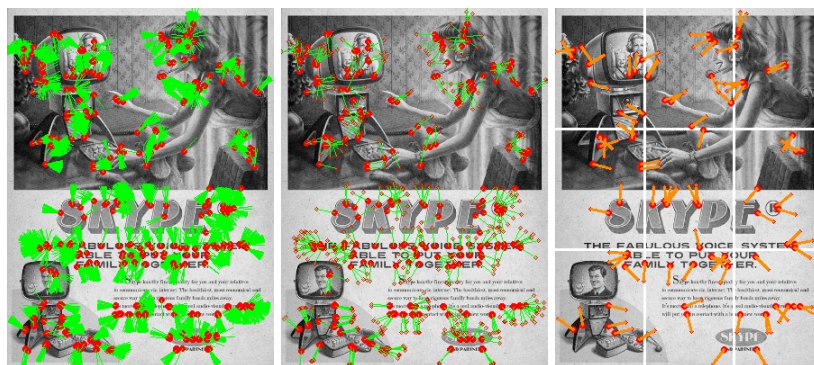
The manner in which the features are extracted for both stability methods are identical, however, there is a subtle difference between the two in terms of the keypoints associated with each stable keypoint group. The local stability method clusters the detected keypoints directly from each virtual image for a given viewpoint bin. Therefore, it only extracts features from those images in which the keypoints were clustered together. On the other hand, the global stability method makes use of a separate set of virtual images to establish the stable keypoints. Since the method claims to identify globally stable keypoints, it assumes that those keypoints found should be detected in all images for the specified deformation space from which the images were generated. Therefore, the global method extracts features from all the virtual images in the viewpoint bin for each stable keypoint. Figure 6.4 demonstrates this difference.

In addition to storing the aggregated feature, we also maintain its corresponding spatial coordinate on the reference image. This is obtained from the reference point $\hat{p}_r$ associated with the stable keypoint group. The spatial coordinate information is used later at runtime for homography estimation. To this end, we make use of two 16-bit integers to store the (x, y) values which allows us to address an image of maximum resolution 65536×65536 .

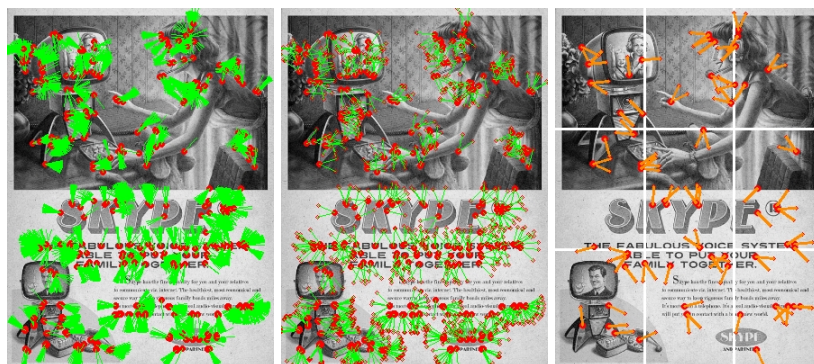
We summarize the feature extraction and aggregation process in Algorithm 6.4. Note that the model database $\hat{F}$ serves as both input and output parameters, since the features from the n^{th} viewpoint bin need to be added to $\hat{F}$, which incorporates features from other viewpoint



(a)



(b)



(c)

Figure 6.3: Stratified sampling with orientation clustering $\delta_K = 2$, $W = 3$, and $Q = 90$. (a) First 10 virtual images of the viewpoint bin. (b) Graph-based hierarchical clustering: $\epsilon = 20^\circ$. (c) Probabilistic clustering: $\epsilon = 20^\circ$, $N_c = 7$, and $\rho = 0.3$.

Algorithm 6.4 Feature Extraction and Aggregation.

Input: $\hat{F}$: Model database, $\bar{I}_n = \{I_{1n}, \dots, I_{Mn}\}$: M virtual images for viewpoint bin n , $\mathbb{G}$: stable keypoint groups, $\hat{p}$: reference points associated with each group

Output: $\hat{F}$: Model database

```

1 for  $r = 1$  to  $|\mathbb{G}|$  do
2    $F^\dagger \leftarrow \emptyset$ 
3   foreach  $p_{mk} \in G_r$  do
4      $F' \leftarrow$  Extract partial feature for keypoint  $p_{mk}$  from  $I_{mn}$ 
5      $F^\dagger \leftarrow F^\dagger \cup F'$ 
6    $F_r(\text{feature}) \leftarrow$  Aggregate the set of partial features  $F^\dagger$ 
7    $F_r(x, y) \leftarrow \hat{p}_r$ 
8    $\hat{F} \leftarrow \hat{F} \cup F_r$ 
9 return  $\hat{F}$ 

```

bins.

6.3 Feature Stability Verification

Being able to identify keypoints which are detectable across various deformations does not necessarily imply that the associated features are reliable. Since we aggregate multiple partial features to obtain the final feature, a situation can arise where the viewsets contain too much variation in appearances, resulting in an unreliable, noisy feature. We propose computing a supplementary feature vector, which we call the stability vector, Ω , in order to identify and reject such random features.

For binary feature descriptors, Ω is determined as follows. For each stable keypoint, we compute the entropy of each bit of the binary representation associated with the group of keypoints from the different virtual images. This entropy measure is based on the fraction of times a given bit has a value 1, which is simply $P_r(1)$, and similarly for $P_r(0)$.

$$\text{Entropy}(G_{nr}) = -P_r(1) \log_2 P_r(1) - P_r(0) \log_2 P_r(0) . \quad (6.1)$$

To be considered stable, the entropy of a bit must be lower than a given threshold. If this is the case the corresponding bit of Ω is set to 1. After all the bits of the feature have been processed, we perform a bitcount on Ω and compare it to a threshold Ω_s (the stability threshold). If the

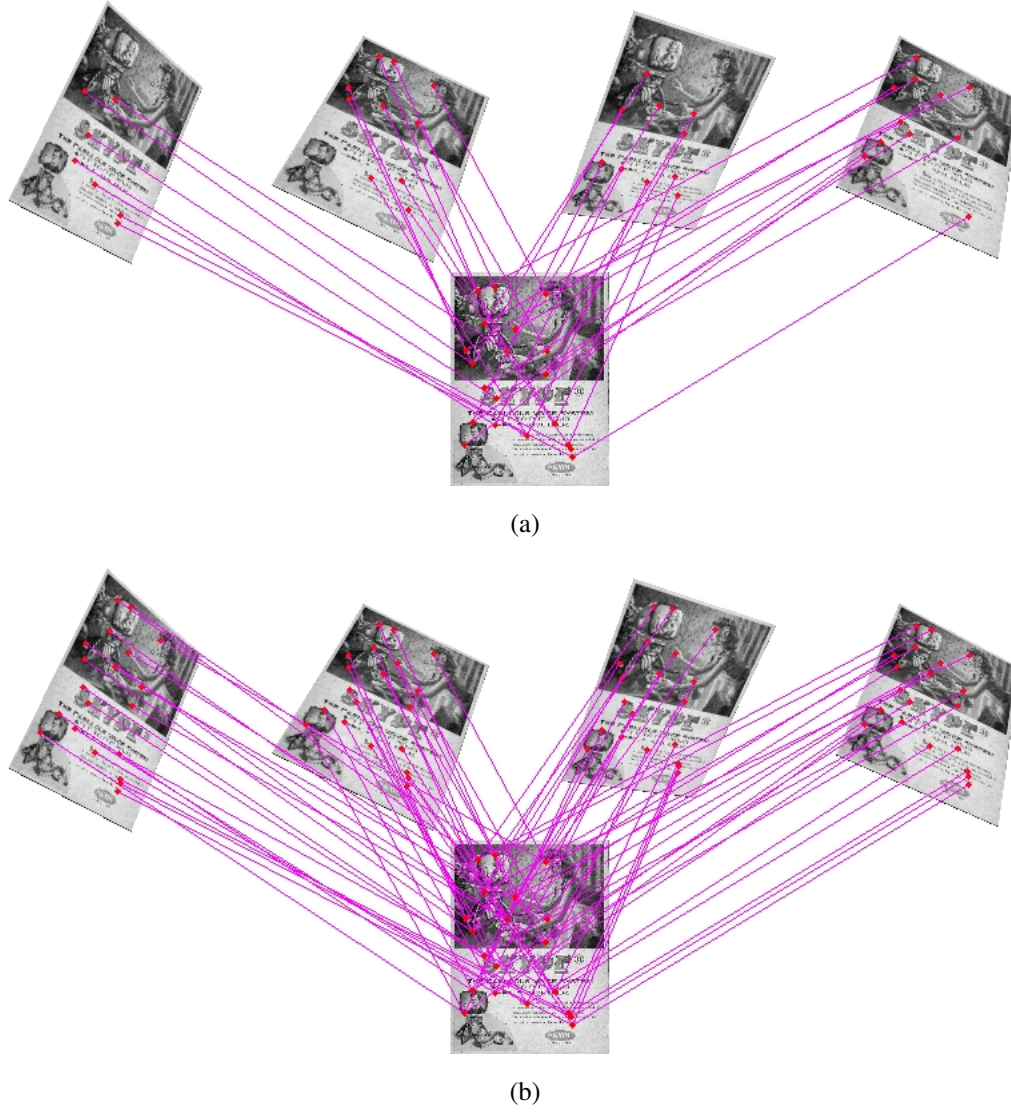


Figure 6.4: Stable keypoints selected for the feature extraction: (a) local stability method (b) global stability method. To illustrate the difference between the two methods, only a subset of the stable keypoints found were selected.

number of stable bits is less than this threshold, then the representation is considered unstable, and is rejected.

Furthermore, by storing the stability vector associated with each feature in $\hat{F}$, we can exploit the knowledge of which elements of the feature are stable for establishing reliable correspondences. Thus, when matching features, we only compare the bits (for binary features) or dimensions (for floating point features) that were deemed stable. However this comes at the

cost of doubling the size of the model database $\hat{F}$.

6.4 Binary Feature Descriptors

Previously, we had described the process for identifying and constructing features for stable keypoints. The process of computing the feature vector is formally known as feature extraction whereas the actual algorithm for computing the feature is known as the descriptor. The objective of all feature descriptors is to compute a robust, compact representation of the input intensity patch that is invariant to various deformations and artifacts including illumination.

Binary descriptors are of particular interest as they require far less storage capacity and offer much faster matching times than conventional floating point descriptors. This is possible since binary features can be compared efficiently using the Hamming distance, which comprises of a bitwise XOR operation followed by a bit count on the result.

In this section, we describe the Histogrammed Intensity Patches (HIP) descriptor [40] as well as the BRIEF descriptor [8]. The HIP descriptor has been shown to produce very good matching results (comparable to state-of-the-art SIFT [23] and Fern [30] descriptors) at a fraction of the processing time. Calonder *et al.* [8] have demonstrated the BRIEF descriptor to be highly discriminative and invariant to changes in illumination, blur as well as partial perspective distortions. However, BRIEF remains sensitive to both scale and rotation deformations. Hence, it is a perfect candidate to be applied to our framework. We therefore extend the original BRIEF descriptor by integrating it with our model generation framework. To this end, we propose three variants of the modified BRIEF descriptor. Although we apply our framework in conjunction these two binary descriptors, it can readily be adapted for any such binary descriptor.

6.4.1 HIP

The Histogrammed Intensity Patches (HIP) feature descriptor, as its name suggests, is based on quantized histograms of intensity patches. A sparse 15×15 grid, shown in Fig. 6.5, is used to sample 64 intensity values from a patch centered on the keypoint. Invariance to illumination is achieved by normalizing the patch such that the samples have zero mean and unity standard deviation. The normalized samples are then quantized to 5 intensity levels.

The boundaries between quantization levels are configured to yield an equal number of samples per level. Assuming that the (normalized) samples follow a Gaussian distribution, Taylor and Drummond [39] compute the upper intensity quantization boundary levels as fol-

lows:

$$Q_i = \aleph^{-1}\left(\frac{i}{5}\right), i = 1, \dots, 5, \quad (6.2)$$

where $\varsigma = \aleph^{-1}(\varpi)$ is the inverse of the cumulative Gaussian distribution:

$$\varpi = \aleph(\varsigma) = \frac{1}{2\pi} \int_{-\infty}^{\varsigma} \exp \frac{-t^2}{2} dt. \quad (6.3)$$

Using Matlab's `norminv` function, we obtain the boundary values shown in Fig. 6.6. The sampling and quantization process is illustrated in Fig. 6.7.

A histogram is then computed using the corresponding quantized samples per keypoint for each stable group. Since each sample can be quantized into one of five possible levels, the histogram is specified by 5 bins. The histogram is computed for all the keypoints in a stable keypoint group. For example, if a stable keypoint group comprises of 50 keypoints, then, all 50 quantized patches will be used to compute the histogram for each of the 64 samples. Once the empirical distribution is approximated, it is then further quantized into a binary representation by thresholding each bin. This enables us to identify bins that seldom occur. Thus, if a bin in the histogram for the i^{th} sample ($i = 1, \dots, 64$) is less than the threshold, the corresponding bit is set to 1. Taylor and Drummond [40] recommend a threshold value of 0.05. The resulting binarized histograms are then concatenated to form the 320-bit database feature.

In contrast to the database features, the runtime HIP features are computed directly from the 64 normalized intensity levels. As with the database features, each pixel is also quantized into one of the 5 possible levels, however, instead of computing a histogram (which is point-less as we only have a single patch), we encode the quantization process directly into a binary representation. For example, if a pixel is quantized into the fourth level (Q_4), its binary representation is 00010₂. All the 64×5 bits are then concatenated to obtain the 320-bit runtime HIP feature.

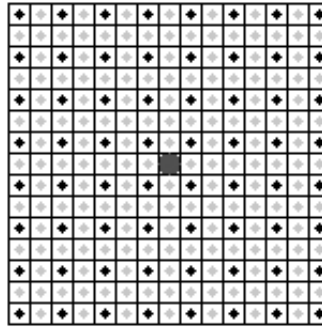


Figure 6.5: The sparse 15×15 sampling grid used by HIP.

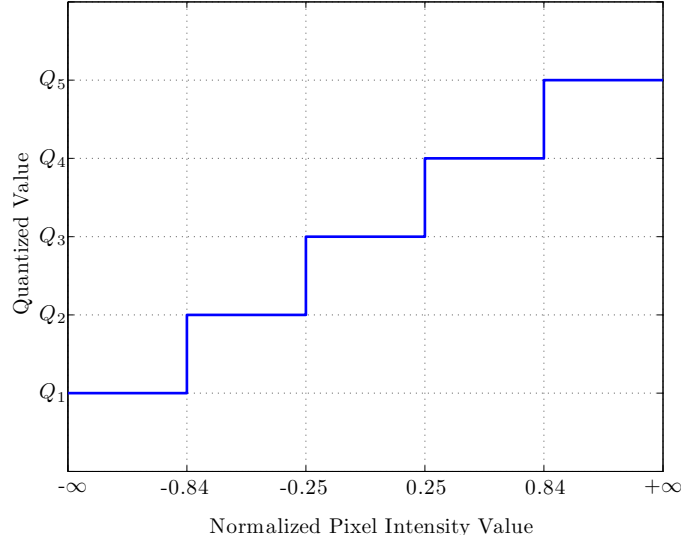


Figure 6.6: The quantizer used by HIP.

A runtime HIP feature is compared against a database feature by computing a dissimilarity score which measures the number of bits where both features are equal to 1. For two features to be considered a match, their dissimilarity score must be less than a user-defined matching threshold.

6.4.2 BRIEF

Prior work [18, 30] has demonstrated that image patches can be recognized by training classifiers on the outcomes of very simple and randomly chosen binary tests. Calonder *et al.* [8]

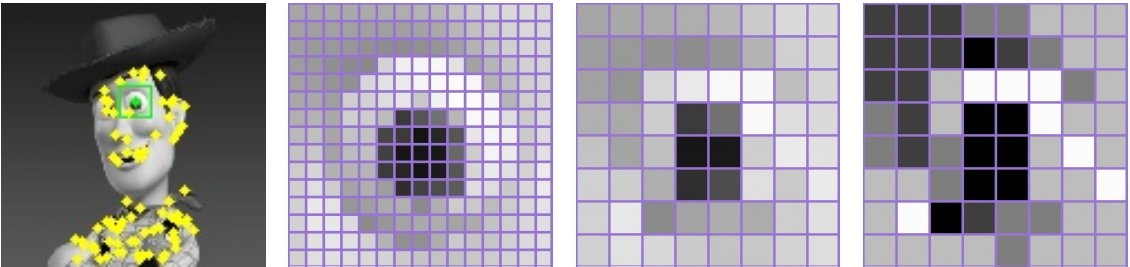


Figure 6.7: Illustration of the HIP sampling and quantization process. (a) Keypoints detected from image. (b) Patch centered at the keypoint of interest is extracted. (c) Sparse sampling grid of Fig. 6.5 is applied on the patch of (b) to obtain the 64 samples. (d) Resultant quantized patch of (c), note that there are 5 distinct quantization levels.

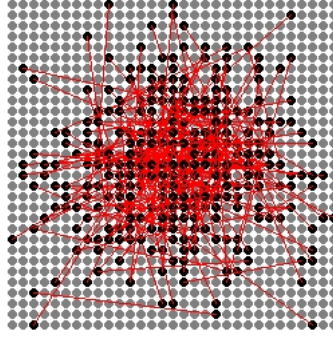


Figure 6.8: BRIEF test pattern for a 31×31 patch.

have developed the BRIEF, Binary Robust Independent Elementary Features, descriptor on the same basis, however, they discard the classifier component, and simply create a binary representation out of the test responses.

The BRIEF feature is computed by concatenating simple binary tests based on the intensity differences between pixels in a smoothed image patch. Consider a smoothed image patch, $\mathcal{R}_s$ with two points, $(p_i, p_j) \in \mathcal{R}_s$. A binary test τ is defined by:

$$\tau(p_i, p_j) := \begin{cases} 1 & \text{if } I(p_i) < I(p_j) \\ 0 & \text{otherwise} \end{cases}, \quad (6.4)$$

where $I(p_i)$ is the intensity of a pixel at point $p_i = (x_i, y_i)$. The BRIEF feature is obtained as the concatenation of d such binary tests, where $d = \{128, 256, 512\}$.

There are many possible ways for selecting the test locations (p_i, p_j) from a patch of size $t \times t$. Calonder *et al.* [8] have experimented with various sampling geometries and have found that tests sampled from an isotropic Gaussian distribution yield the best matching performances. In addition, they experimentally established that the x and y test coordinates sampled from i.i.d. Gaussian $(0, \frac{1}{25}t^2)$ distribution gives the best results in terms of recognition rate. This is motivated by the fact that pixels at the patch center tend to be more stable under perspective distortion than those near the edges. For our application, we set the descriptor length to $d = 256$ bits, and generate the binary test locations according to this Gaussian distribution for a 31×31 patch. The generated BRIEF binary tests are shown in Fig. 6.8.

Calonder *et al.* [8] have demonstrated the BRIEF descriptor to be highly discriminative and invariant to changes in illumination, blur as well as partial perspective distortions. However, BRIEF remains sensitive to both scale and rotation deformations. Hence, it is a perfect candidate to be applied to our model generation framework. We propose here three variants to the BRIEF descriptor by extending the original algorithm using our training framework, namely

cBRIEF, *sBRIEF*, and *s+BRIEF*. They all operate similarly, however, the latter two make use of the feature stability verification process during training (*sBRIEF*) and matching (*s+BRIEF*).

cBRIEF

The first proposed variant of BRIEF is classic BRIEF, or *cBRIEF*. For each group of stable keypoints, the BRIEF binary tests are evaluated on patches extracted around each keypoint within the group. The results are aggregated together by estimating the empirical distribution of the total response to each test. The distributions are then quantized to yield a binary representation. This is achieved by computing the probability of observing a particular test outcome for the keypoints within the group, if the probability is greater than 0.5, the respective bit is configured accordingly.

It should be noted here that the premise upon which the descriptor for *cBRIEF* is extracted remains exactly the same as that in BRIEF (Eq. 6.4). However, the difference between the two is that *cBRIEF* combines the information from multiple test responses to form the final feature, whereas the standard BRIEF descriptor is obtained from only one test response.

sBRIEF

The second proposed variant of BRIEF is stable BRIEF, or *sBRIEF*. When BRIEF descriptors are constructed from multiple test responses, one has access to a great deal of information pertaining to the reliability of the binary tests which cannot otherwise be communicated through a single test. In order to take advantage of the additional information, we construct a supplementary stability binary vector, Ω , using Eq. 6.1 in order to determine whether or not a given test is at least 90% stable (i.e. the test results in the same outcome in 90% of the patches).

The feature is computed in the same manner as *cBRIEF*. After extracting the feature and stability vector, an additional decision whether to accept or discard the feature is made. If the number of stable binary tests in Ω exceeds that of a stability threshold Ω_S , the feature is retained, otherwise it is presumed unreliable and is therefore discarded. It is important to note that in *sBRIEF*, we only use the *feature stability verification* component in the training phase to select the most stable features. The runtime matching scheme remains as in the original BRIEF.

s+BRIEF

The third proposed variant of BRIEF is *s+BRIEF*. In this version, we apply the feature stability verification component in both the training phase as well as during runtime matching. The

descriptors are computed identically to that of sBRIEF, however, the stability vectors are stored in addition to the feature. Note that this has the effect of doubling the file size of the model database.

The distance metric used to measure the similarity between a runtime query feature F_q and a candidate database feature $F_p \in \hat{F}$ is modified to incorporate the stability vector, Ω_p , associated with F_p . The new distance is computed by performing a bitcount on the result of ANDing Ω_p with the output of F_q XOR F_p . Since we disregard the unstable bits when measuring the similarity between two descriptors, matching is more likely to result in fewer false positives, i.e., a reduced outlier ratio. This ultimately is advantageous in estimating the homography matrix.

6.5 Conclusion

In this chapter we have described the process of identifying stable keypoints and extracting features for those points. To this end, we have developed two types of stability measures for identifying stable keypoints, namely *local stability* and *global stability*. The local stability measure identifies stable keypoints which are local to the viewpoint bin under consideration. On the other hand, the global stability measure employs a separate set of artificially deformed images to identify stable keypoints which are detectable under a global range of deformations. To decrease the system's vulnerability to occlusion, we adopt a stratified sampling technique to select the final stable keypoints across the entire image. Once we have identified the stable keypoints, we proceed to extract the descriptors. To ensure that we only add those features to the model database which are meaningful and reliable, we have proposed a novel method for verifying the feature's stability based on its entropy. In the last part of the chapter, we have applied our model generation framework to the HIP and BRIEF descriptor. These different descriptors will be evaluated in Chapter 8.

Up until now we have described the processes involved in learning a target. In the next chapter we present our proposed approach to detecting targets at runtime using the model database.

Chapter 7

Detecting Targets at Runtime

We now have a model that describes a given target by a set of features obtained from virtual viewpoints. Furthermore, these features were extracted from the most stable keypoints. The problem at hand is, given a query runtime feature F_q , we now have to determine whether $F_q \in \hat{F}$, where $\hat{F}$ is the model database containing features of the target we wish to detect. The goal is to find a corresponding feature $F_p \in \hat{F}$, such that:

$$\arg \min_{F_p} D(F_q, F_p) \leq \mathcal{Z} , \quad (7.1)$$

where $D(\cdot, \cdot)$ represents the distance metric between two features, and $\mathcal{Z}$ denotes the matching (or error) threshold.

The simplest solution comprises of a linear search, which guarantees to find such a point F_p provided it exists. However, the linear search has complexity $O(n)$ where n is the number of features in $\hat{F}$, and therefore does not provide a real-time solution. In our case it is not necessary to insist on the best match, i.e., the match with the least distance and instead, determining an approximate match, for which the distance between the two features is less than $\mathcal{Z}$, is sufficient. Hence we adopt an approximate nearest-neighbor matching strategy to achieve real-time matching, namely *Locality Sensitive Hashing* [12].

After matching all the query features from a runtime image, we then proceed to estimate the homography relating the query and target images from the tentative correspondences. However, it is most likely that the correspondences contain false matches (outliers) which makes it difficult to compute the homography. To solve this issue, we make use of the robust Random Sample Consensus (RANSAC) algorithm [10].

Since we are dealing with hand-held devices, it is expected that we will encounter a lot of motion blur. Later in this chapter, we propose a simple, yet effective, strategy to address this issue.

7.1 Approximate Nearest-Neighbor Matching

7.1.1 Motivation

One may suggest that an inductive classifier presents a more efficient solution than the approximate nearest-neighbor approach. To this end, there are a number of classifiers at one's disposal, the most prominent including: Support Vector Machine (SVM), Artificial Neural Network (ANN), Bayesian classifier, and Random Forest classifier. However, there are two important criteria we must take into consideration:

1. the dimensionality of the feature, which we shall denote as d , and
2. the total number of features n , which is simply $n = |\hat{F}|$.

These criteria dictate the choice of classifier as well as its effective success rate. Generally, most descriptors are high dimensional, with $d > 100$. Therefore we encounter the “curse of dimensionality” problem, in which time and space requirements grow exponentially with d . Furthermore, we typically have $n \gtrsim 10^3$ even for a small deformation space with a limited number of viewpoint bins.

Classifiers such as SVM, ANN and Random Forest can handle high dimensional feature spaces, however, they require more complex parameter configurations. Consider the SVM classifier, which operates by projecting a potentially non-linearly separable feature space into a higher dimensional space which is linearly separable. To do this, we must specify the projective function (kernel) and initialize its parameters accordingly. Given that each target image is different, although we might be able to use the same kernel, it might be necessary to recompute the optimal parameters to obtain a satisfactory decision boundary.

Let us consider the neural network classifier as well. Given an arbitrary target, it is a non-trivial task to establish the optimal network architecture, i.e., the number of hidden layers as well as the number of neurons per layer. Furthermore, neural networks take an exceptionally long time to train and possess the risk of over-fitting if trained without a separate validation set.

The Random Forest classifier falls under the category of randomized algorithms, hence, is perhaps more well-suited for our application, as it might be possible to re-use the optimal configuration. However, it too requires initialization of a set of unique parameters, such as, the optimal number of trees in the forest and the maximum depth of the trees. A low value will likely under-fit and conversely a high value will likely over-fit. Furthermore, the forest can be quite large and therefore requires independent storage space in addition to $\hat{F}$.

It is also important to note that a majority of these inductive classifiers are not designed for the 1-class problem and typically require a balanced training set comprising of positive and negative instances. Although it is fairly trivial to generate negative instances, the challenge comes from generating characteristic negative instances (i.e. instances which assist the classifier in learning the optimum decision boundaries). An absence of negative instances results in the classifier erroneously associating a query point to some point in $\hat{F}$. This issue has been studied comprehensively under the headings *one-class learning* and *class-imbalanced learning*.

Lastly, we must also take into consideration the feasibility of running the classifier on a low-powered device, as this is where it will be used.

From the above arguments, it is evident that inductive classifiers pose more challenges than offering a solution. Theoretically, it is possible to generate classifier models which are representative of $\hat{F}$, however, as we have shown, it is non-trivial for the general case.

In the subsequent sections, we describe an efficient solution which is based on approximate nearest-neighbor matching. The solution is able to find relatively good correspondences in almost $O(1)$ time, depending on $|\hat{F}|$.

7.1.2 Problem Statement

The approximate nearest-neighbor problem is an optimization problem in which the goal is to find a point which minimizes a certain objective function, in our case, the distance to the query point. Formally, we say that a point F_p is an $\mathcal{Z}$ -near neighbor of a point F_q if the distance between F_p and F_q is at most $\mathcal{Z}$. The formal definition of the approximate nearest neighbor problem is as follows.

Given a set of points, $\hat{F} = \{F_1, F_2, \dots\}$, in a d -dimensional space $\mathbb{R}^d$, and parameters $c > 1$, $\mathcal{Z} > 0$, $\delta > 0$, construct a data structure such that, given any query point F_q , if there exists a $\mathcal{Z}$ -near neighbor of F_q in $\hat{F}$, it reports some $c\mathcal{Z}$ -approximate near neighbor of F_q in $\hat{F}$ with probability $1 - \delta$.

7.1.3 Data Structures

The two most widely used data structures are, *trees* and *hash tables*.

Trees

For a one-dimensional feature set ($d = 1$), it is trivial to efficiently search for values by constructing a tree of the features. Then, for a given query we start from the parent node of the tree, determine whether our query feature is to the left or to the right of the node, and then recursively traverse the tree. For a properly constructed tree, an arbitrary query is processed in $O(\log n)$ time, where n is the number of features. For the one-dimensional case, the resulting data structure is known as a binary tree.

The k-d tree algorithm was designed to extend the binary tree concept for a multi-dimensional feature set. However, as the number of dimensions exceeds 10 to 20, searching in k-d trees and related structures involves the inspection of a large fraction of the feature set, thereby growing the computational complexity to $O(n)$ [12].

Hash Tables

Hashing is a technique which is used to quickly map a symbol (such as a d -bit string) to an integer value. A hash table is a data structure which stores the linked association between the symbol and integer value. In most cases, hashing is achieved by calculating an arbitrary, pseudo-random function of the symbol that maps the symbol into an integer value which then indexes a table entry.

A well-designed hash function distinctly maps two symbols which are reasonably “close” to one another into different indices. Therefore, the computational complexity to search for an item using a well-designed hash function is $O(1)$ time with $O(n_t)$ memory, where n_t is the number of entries in the table. Note that a hash table returns an exact match. This makes a hash table a good means of finding exact matches, but not for finding approximate matches. By contrast, as shall be seen shortly, a locality-sensitive hash is an efficient means of finding near matches.

Provisions must be made to allow more than one symbol per index as it commonly occurs that two (or more) points hash to the same value. This phenomenon is formally known as a *collision*.

7.1.4 Locality Sensitive Hashing

Locality sensitive hashing (LSH), first introduced by Indyk and Motwani in [16], was developed to search high dimensional ($d \gg 10$), large sized databases ($n \gg 10^2$) in sub-linear time. Prior to LSH, most algorithms were only able to marginally outperform the linear search

process for high dimensional feature spaces. The LSH technique forms part of a novel and interesting class of algorithms referred to as randomized algorithms. A randomized algorithm does not ensure an exact answer, but instead provides a high probability guarantee that it will find the correct answer or one in proximity to it.

LSH achieves its success by applying several hash functions to hash points from the database, thereby increasing the likelihood of collision for points which are close to each other. Note that hashing of the database points is performed offline. At runtime, neighbors to a query point are rapidly determined by hashing the point and retrieving elements stored in buckets containing that point.

The hashing functions are generated from a locality-sensitive family $\mathcal{H}$. LSH families have the property that points that are close to each other have a higher probability of colliding than objects that are far apart. Specifically, let $\hat{F}$ be the domain of objects, and $D(\cdot, \cdot)$ be the distance measure between points.

Definition 7.1 A LSH family $\mathcal{H} = \{h : \hat{F} \rightarrow \mathcal{F}\}$ is called $(c, \mathcal{Z}, P_1, P_2)$ -sensitive for $D(\cdot, \cdot)$ if for any $F_q, F_p \in \hat{F}$

1. $D(F_q, F_p) \leq \mathcal{Z}$ then $\Pr_{\mathcal{H}} [h(F_q) = h(F_p)] \geq P_1$
2. $D(F_q, F_p) > c\mathcal{Z}$ then $\Pr_{\mathcal{H}} [h(F_q) = h(F_p)] \leq P_2$.

In order for a locality-sensitive family to be useful, it must satisfy the inequalities $P_1 > P_2$ and $c > 1$. With these constraints, nearby points (those within distance r) have a greater chance (P_1 vs. P_2) of being hashed to the same value than points that are far apart (those at a distance greater than $c\mathcal{Z}$ away).

Of particular interest to us is the case of binary descriptors, for which the resultant features can be represented as points in a d -dimensional Hamming space, i.e., $\hat{F} \in \mathbb{H}^d$. Therefore, the distance measure between any two points $F_p, F_q \in \hat{F}$ is determined by the Hamming metric $D_{\mathbb{H}}$:

$$D_{\mathbb{H}}(F_p, F_q) = \sum_{i=1}^d F_{p_i} \oplus F_{q_i} , \quad (7.2)$$

where F_{p_i} is the value of the i^{th} coordinate (bit) for point (feature) F_p and $\oplus$ is the exclusive-or operator.

Furthermore, for the d -dimensional Hamming space, the LSH family $\mathcal{H}$ is defined as:

$$\mathcal{H}^d = \{h_i : h_i(\mathbf{b}) = b_i | i = 1, \dots, d\} , \quad (7.3)$$

where $\mathbf{b}$ is a point in $\mathbb{H}^d$, and b_i is the i^{th} coordinate of $\mathbf{b}$. Each $\mathfrak{h}_i$ results in a single bit. From this we can compute the following probabilities:

$$\Pr_{\mathcal{H}} [\mathfrak{h}(F_p) \neq \mathfrak{h}(F_q)] = \frac{D_{\mathbb{H}}(F_p, F_q)}{d}, \text{ and} \quad (7.4)$$

$$\Pr_{\mathcal{H}} [\mathfrak{h}(F_p) = \mathfrak{h}(F_q)] = 1 - \frac{D_{\mathbb{H}}(F_p, F_q)}{d}. \quad (7.5)$$

Clearly the Hamming space family is locality-sensitive, with $\mathcal{H}$ being $(c, \mathcal{Z}, 1 - \frac{\mathcal{Z}}{d}, 1 - \frac{c\mathcal{Z}}{d})$ -sensitive.

Using the LSH family $\mathcal{H}$, we can construct a set of k -bit locality-sensitive hash functions, $\hat{g} = \{g_1, \dots, g_l\}$, by concatenating k LSH family members as follows:

$$g_j(\mathbf{b}) = (\mathfrak{h}_{j_1}(\mathbf{b}), \mathfrak{h}_{j_2}(\mathbf{b}), \dots, \mathfrak{h}_{j_k}(\mathbf{b})) , \quad (7.6)$$

where each $\mathfrak{h}_{j_1}(\cdot), \dots, \mathfrak{h}_{j_k}(\cdot)$ is randomly chosen from $\mathcal{H}$ with replacement¹. Each of the l functions $g_j (1 \leq j \leq l)$ is used to construct one hash table, resulting in l hash tables.

By concatenating k LSH family members, the collision probability of far away points becomes very small (P_2^k), however, the collision probability of nearby points is also reduced (P_1^k). The optimal value of k is chosen to maximize the probability that a point F_p “close” to F_q will fall into the same bucket as F_q , and also to minimize the probability that a point F_p “far away” from F_q will fall into the same bucket.

During the training stage, we store each $F_p \in \hat{F}$ in the bucket $g_j(F_p)$, for $j = 1, \dots, l$. To process a query F_q at runtime, we simply search all indices $g_1(F_q), \dots, g_l(F_q)$ and return the point F_p approximately closest to F_q .

Demonstration

We present a toy example to demonstrate the LSH algorithm. We set $n = |\hat{F}| = 5$, $d = 10$, $\mathcal{Z} = 3$, $l = 3$, and $k = 4$. The 4-bit locality-sensitive hashing functions are defined as follows: $g_1(\cdot) = (\mathfrak{h}_1(\cdot), \mathfrak{h}_3(\cdot), \mathfrak{h}_8(\cdot), \mathfrak{h}_{10}(\cdot))$, $g_2(\cdot) = (\mathfrak{h}_5(\cdot), \mathfrak{h}_1(\cdot), \mathfrak{h}_6(\cdot), \mathfrak{h}_2(\cdot))$, and $g_3(\cdot) = (\mathfrak{h}_{10}(\cdot), \mathfrak{h}_7(\cdot), \mathfrak{h}_3(\cdot), \mathfrak{h}_5(\cdot))$. Figure 7.1 presents the LSH training stage for populating the 3 hash tables (shown as one large table) for the model database comprising of 5 binary features.

Now let's say we have a query feature $F_q = 0010001011_2$, using our locality-sensitive hashing functions $\{g_1, g_2, g_3\}$, we compute the corresponding indices as follows: $g_1(F_q) = 5$, $g_2(F_q) = 0$, and $g_3(F_q) = 14$. From Fig. 7.1, we can see that the only corresponding index

¹No benefit is gained from repeated coordinates, on the contrary, no harm is done either. However, not having to account for avoiding repetition makes some of the exposition easier.

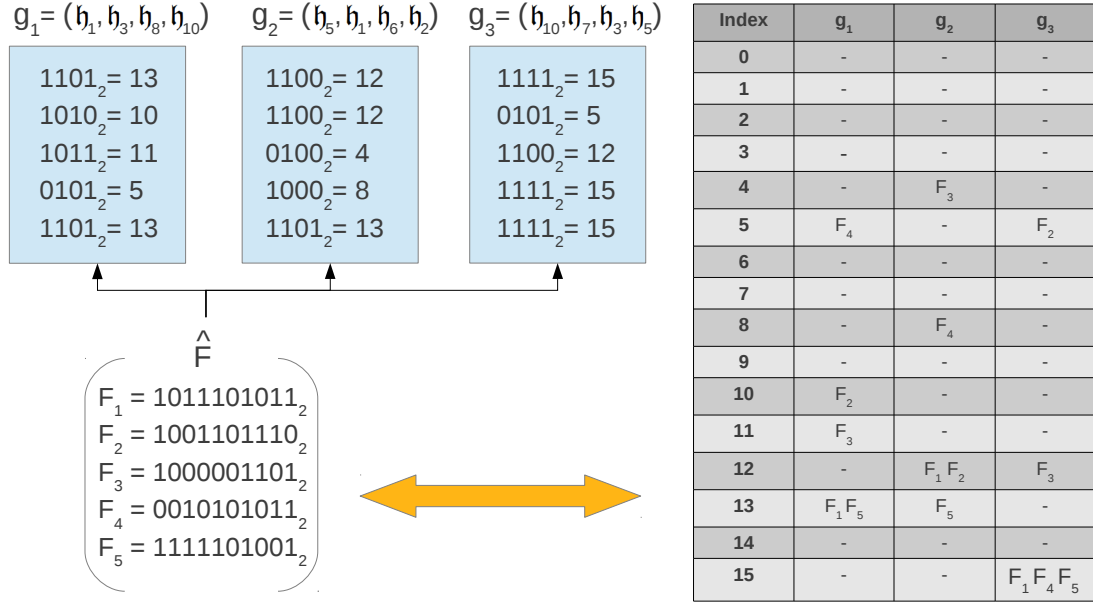


Figure 7.1: Locality Sensitive Hashing preprocessing stage: $\mathcal{H} = \{h_1, \dots, h_{10}\}$.

is from the first hash table ($5 \iff F_4$), the Hamming distance to the associated feature is $D_{\mathbb{H}}(F_q, F_4) = 1$. Since $D_{\mathbb{H}}(F_q, F_4) < 3$ ($\mathcal{Z}$), LSH returns feature F_4 as a match.

To verify that we indeed have a good match, we compute the Hamming distances to each feature in $\hat{F}$ from F_q : $D_{\mathbb{H}}(F_q, F_1) = 3$, $D_{\mathbb{H}}(F_q, F_2) = 6$, $D_{\mathbb{H}}(F_q, F_3) = 4$, $D_{\mathbb{H}}(F_q, F_4) = 1$, and $D_{\mathbb{H}}(F_q, F_5) = 5$. Indeed, the best matching point to F_q is feature F_4 .

Note that the computational complexity for LSH is almost $O(1)$, whereas for the linear search it is $O(n)$.

7.1.5 Implementation

For binary feature descriptors, the LSH family $\mathcal{H}$ is dependent on the dimensionality of the feature vector. Since we need to evaluate our model generation framework against several binary feature descriptors of varying length (d), we might induce a bias by generating the LSH functions from variable dimensional Hamming spaces. Thus, for the sake of a fair evaluation, we develop a common hashing function for all the descriptors.

We achieve this by computing an additional binary feature vector from a small $t \times t$ patch centered on the detected keypoint. For the given patch, we extract all t^2 pixels and compute the mean of the samples. We then quantize the samples by thresholding each pixel to a single bit: 1 if the pixel value is greater than or equal to the mean of the patch and 0 otherwise. Note

that this feature vector is independent of the feature descriptor used, and is extracted only for hashing purposes.

From this new feature space, we can determine the LSH family according to Eq. 7.3 with $d = t^2$. We can now compute our hashing functions, $\hat{g}$, from $\mathcal{H}^{t^2}$. To ensure that all descriptor methods are evaluated under the same environment, we simply apply the same $\hat{g}$ to each descriptor. For the LSH algorithm to be effective, $\hat{g}$ must be designed to have the following properties [24]:

- **Accurate:** A query operation should find point(s) that are very close to those of the linear search approach.
- **Time efficient:** A query operation should take $O(1)$ or $O(\log n)$ time where n is the size of the model database $\hat{F}$.
- **Space efficient:** k should be set as such that the resulting indices occupy a very small amount of space. Note that the hash tables must be loaded into main memory.

In the proposed model generation framework, a feature that is added to the model database is computed by aggregating several partial features. These partial features are extracted from patches of the virtual images from which the keypoint is detected. We now need to address the following question: How do we select the set of index values to represent the aggregated feature? Furthermore, in order to generate $\hat{g}$, we must first specify the parameter $l = |\hat{g}|$ (the number of hashing functions) as well as the length of each function, i.e. k . These parameters greatly influence the question at hand.

There are two ways in which we can approach this issue. In the first method, we compute all the partial indices using $\hat{g}$ from each virtual image from which the keypoint is detected. We then obtain the final set of l -index values by appropriately aggregating the partial indices. In the second method, we determine the set of l -index values from each virtual image which will account for the most possible runtime views for the keypoint under consideration. In this case, every patch from the virtual images that contributes towards the final feature also contributes a vote for the index value computed from that patch. We then select a percentage of the most-commonly encountered indices.

A limitation of the first approach is that in aggregating the partial indices, we might obtain a set of index values which are non-representative of the actual feature. On the other hand, for the second approach, adding the most commonly encountered sets of index values results in significantly large hash tables. The first approach thus affects the accuracy of the system, whereas the second approach proves to be space inefficient for a large l . Nonetheless, of both

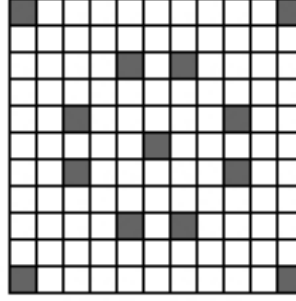


Figure 7.2: 13-bit Locality-sensitive hashing function: h_1 is the first sample on the first row, whereas h_{121} is the last sample on the last row.

these methods, the latter has a greater probability of finding the nearest neighbor to a query point. Since main memory is of importance to us, we cannot have too many hashing functions. Hence, we trade-off the number of hashing functions with the length of each function: l vs. k .

The initial implementation of the HIP binary descriptor [40] employs a 13-bit indexing scheme to achieve fast and reliable matching. We set our hash function to the same indexing scheme with $l = 1$, $k = 13$, and $t = 11$. The 13 samples are selected to be reasonably close to the patch center as they are expected to be more consistent under rotation and scale. Figure 7.2 illustrates these samples. Each sample selected for the hash function is quantized to a binary representation as follows: 0 if the pixel value is less than the mean of the patch and 1 otherwise. The hashing function g is configured as:

$$g = (h_1, h_{11}, h_{27}, h_{29}, h_{47}, h_{53}, h_{61}, h_{69}, h_{75}, h_{93}, h_{95}, h_{111}, h_{121}) , \quad (7.7)$$

where we have abused the notation h_i to denote $h_i(\cdot)$.

7.2 Homography Estimation

The process of detecting a target in a query image comprises of two parts. In the first part, points from the query image are matched to points in the model database. The correspondences are then used in the second part to estimate the homography which describes the projective relation between the query and target reference image. We use a variant of the Direct Linear Transform (DLT) algorithm to estimate the homography. To finally locate the target in the query image, we then apply the estimated homography to project the corner locations of the target from the reference image onto the query image.

7.2.1 Notation

The following notation is used in this section. Points on the query image I are represented in homogeneous coordinates as $p = (x, y, w)$. The corresponding points on the reference image $\hat{I}$ are represented as $\hat{p} = (\hat{x}, \hat{y}, 1)$. We use $\mathcal{H}$ to denote the 3×3 homography matrix.

7.2.2 Problem Statement

Consider two images, I and $\hat{I}$, related by a geometric transformation. Given a set of n_p tentative point correspondences $\mathbf{p}_i \rightarrow \hat{\mathbf{p}}$ for $i = 1, \dots, n_p$, we want to estimate the transformation, T , such that

$$I(x, y) = \hat{I}(T(\hat{x}, \hat{y})) . \quad (7.8)$$

We can model T as a linear coordinate transformation (assuming that the imaging sensor yields images that vary linearly in their geometry, i.e., they do contain additional non-linear distortion artifacts):

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \mathcal{H} \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} , \quad (7.9)$$

where the 3×3 matrix $\mathcal{H}$ represents a planar projective transformation matrix, known as the *homography matrix*.

7.2.3 Characteristics of the Homography Matrix

We can formally define the homography transformation as follows.

Definition 7.2 *A homography is an invertible mapping from a projective plane to itself such that the points lie on the same line if and only if their mapped points are also collinear.*

The form of the homography matrix $\mathcal{H}$ specifies the type of geometric transformation represented. The types of geometric transformations include: *rotation*, *translation*, *scaling*, *skewing*, and *perspective*. These various transformations can be summarized into the following general form of the homography matrix:

$$\mathcal{H} = \begin{bmatrix} \alpha s \cos \theta & -\beta s \sin \theta & t_x \\ \alpha s \sin \theta & \beta s \cos \theta & t_y \\ \varrho_1 & \varrho_2 & 1 \end{bmatrix} , \quad (7.10)$$

where θ is the rotation angle, t_x and t_y is the translation about the x and y axes respectively, s is the scaling factor, α and β denote the skew factor of the x and y axes respectively, and ϱ_1 and ϱ_2 represent the perspective transformation parameters.

From Eq. 7.10, we can see that the homography matrix has eight degrees of freedom (DOF). We denote the general form of the homography matrix for an arbitrary set of geometric transformations as follow:

$$\mathcal{H} = \begin{bmatrix} \mathcal{H}_{11} & \mathcal{H}_{12} & \mathcal{H}_{13} \\ \mathcal{H}_{21} & \mathcal{H}_{22} & \mathcal{H}_{23} \\ \mathcal{H}_{31} & \mathcal{H}_{32} & \mathcal{H}_{33} \end{bmatrix}, \quad (7.11)$$

where $\mathcal{H}_{33} = 1$.

Finally, we observe that since $\mathcal{H}$ operates on homogeneous coordinates, it is homogeneous itself. That is, $\mathcal{H}$ can be changed by multiplying by an arbitrary non-zero constant without altering the projective transformation.

7.2.4 Direct Linear Transform

The Direct Linear Transform (DLT) approach is a simple algorithm which solves the homography matrix given a sufficient set of point correspondences. We present a slightly different derivation of the DLT algorithm as that described in [14].

We wish to estimate the 8 unknown parameters of the homography matrix $\mathcal{H}$, based on known point correspondences. The transformation of a source coordinate, $\hat{p}$, in Euclidean form (i.e., normalized so that $w = 1$) to a target coordinate $p^T = \mathcal{H}\hat{p}^T$ yields:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} \mathcal{H}_{11} & \mathcal{H}_{12} & \mathcal{H}_{13} \\ \mathcal{H}_{21} & \mathcal{H}_{22} & \mathcal{H}_{23} \\ \mathcal{H}_{31} & \mathcal{H}_{32} & \mathcal{H}_{33} \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix}. \quad (7.12)$$

We can rewrite Eq. 7.12 in a way that exposes its inherent non-linear form, where the numerator supplies the parameters required for an affine transformation, and the denominator completes the perspective division process:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{\begin{bmatrix} \mathcal{H}_{11} & \mathcal{H}_{12} & \mathcal{H}_{13} \\ \mathcal{H}_{21} & \mathcal{H}_{22} & \mathcal{H}_{23} \\ \mathcal{H}_{31} & \mathcal{H}_{32} & \mathcal{H}_{33} \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix}}{\begin{bmatrix} \mathcal{H}_{31} & \mathcal{H}_{32} & \mathcal{H}_{33} \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix}}. \quad (7.13)$$

This is equivalent to the more familiar, non-vector form of the perspective transform:

$$x = \frac{\mathcal{H}_{11}\hat{x} + \mathcal{H}_{12}\hat{y} + \mathcal{H}_{13}}{\mathcal{H}_{31}\hat{x} + \mathcal{H}_{32}\hat{y} + \mathcal{H}_{33}} , \quad (7.14)$$

$$y = \frac{\mathcal{H}_{21}\hat{x} + \mathcal{H}_{22}\hat{y} + \mathcal{H}_{23}}{\mathcal{H}_{31}\hat{x} + \mathcal{H}_{32}\hat{y} + \mathcal{H}_{33}} . \quad (7.15)$$

By multiplying each side of Eq. 7.14 and Eq. 7.15 by their respective denominators we get:

$$x (\mathcal{H}_{31}\hat{x} + \mathcal{H}_{32}\hat{y} + \mathcal{H}_{33}) = \mathcal{H}_{11}\hat{x} + \mathcal{H}_{12}\hat{y} + \mathcal{H}_{13} , \quad (7.16)$$

$$y (\mathcal{H}_{31}\hat{x} + \mathcal{H}_{32}\hat{y} + \mathcal{H}_{33}) = \mathcal{H}_{21}\hat{x} + \mathcal{H}_{22}\hat{y} + \mathcal{H}_{23} . \quad (7.17)$$

Lastly, multiplying through by x and y , yields:

$$\mathcal{H}_{11}\hat{x} + \mathcal{H}_{12}\hat{y} + \mathcal{H}_{13} - \mathcal{H}_{31}x\hat{x} - \mathcal{H}_{32}x\hat{y} - \mathcal{H}_{33}x = 0 , \quad (7.18)$$

$$\mathcal{H}_{21}\hat{x} + \mathcal{H}_{22}\hat{y} + \mathcal{H}_{23} - \mathcal{H}_{31}y\hat{x} - \mathcal{H}_{32}y\hat{y} - \mathcal{H}_{33}y = 0 . \quad (7.19)$$

Expressing Eq. 7.18 and Eq. 7.19 in matrix representation, for an arbitrary number of point correspondences, yields:

$$\mathcal{A}\mathbf{h} = \begin{bmatrix} \hat{x} & \hat{y} & 1 & 0 & 0 & 0 & -x\hat{x} & -x\hat{y} & -x \\ 0 & 0 & 0 & \hat{x} & \hat{y} & 1 & -y\hat{x} & -y\hat{y} & -y \\ & & & & \vdots & & & & \end{bmatrix} \begin{bmatrix} \mathcal{H}_{11} \\ \mathcal{H}_{12} \\ \mathcal{H}_{13} \\ \mathcal{H}_{21} \\ \mathcal{H}_{22} \\ \mathcal{H}_{23} \\ \mathcal{H}_{31} \\ \mathcal{H}_{32} \\ \mathcal{H}_{33} \end{bmatrix} = \mathbf{0} . \quad (7.20)$$

Note that each point correspondence populates two rows of $\mathcal{A}$, hence n_p point correspondences yields a $2n_p \times 9$ matrix. We adopt a Least-Squares approach to solve Eq. 7.20.

7.2.5 Least-Squares Solution

We seek the minimum-norm solution to

$$\mathcal{A}\mathbf{h} = \mathbf{0} . \quad (7.21)$$

The trivial solution is simply $\mathbf{h} = \mathbf{0}$. However, this is not necessarily the only solution since any vector in the null space of $\mathcal{A}$ is also a solution by definition. However, it is obviously the solution with the smallest norm. Therefore, $\mathbf{h} = \mathbf{0}$ is the minimum-norm solution to any homogeneous linear system. Albeit correct, this solution is not too interesting. What is desired is a *nonzero* vector $\mathbf{h}$ that satisfies the system (Eq. 7.21) as well as possible. Without any constraints on $\mathbf{h}$, we would obtain the same trivial solution $\mathbf{h} = \mathbf{0}$ again. Hence, for homogeneous linear systems, the definition of a least-squares solution is usually adapted by imposing the constraint

$$\|\mathbf{h}\| = 1 \quad (7.22)$$

on the solution. Unfortunately, the resulting constrained minimization problem does not necessarily yield a *unique* solution.

We make use of the singular value decomposition (SVD) technique to solve the system. We first decompose $\mathcal{A}$ into the following form

$$\mathcal{A} = \mathcal{U}\Sigma\mathcal{V}^T, \quad (7.23)$$

where $\mathcal{U}$ is the left-singular vectors of $\mathcal{A}$ (which are the eigenvectors of $\mathcal{A}\mathcal{A}^T$), $\mathcal{V}$ is the right-singular vectors of $\mathcal{A}$ (which are the eigenvectors of $\mathcal{A}^T\mathcal{A}$), and Σ is the singular values of $\mathcal{A}$ (i.e., the square root of the eigenvalues of both $\mathcal{A}^T\mathcal{A}$ and $\mathcal{A}\mathcal{A}^T$). Note that $\mathcal{U}$ and $\mathcal{V}$ are both orthogonal.

We now seek the unit-norm Least-Squares solution to

$$\mathcal{U}\Sigma\mathcal{V}^T\mathbf{h} = \mathbf{0}, \quad (7.24)$$

subject to the constraint of Eq. 7.22. The unit-norm Least-Squares solution to Eq. 7.24 is the vector $\mathbf{h}$ with $\|\mathbf{h}\| = 1$ that minimizes

$$\|\mathcal{U}\Sigma\mathcal{V}^T\mathbf{h}\|. \quad (7.25)$$

Since orthogonal matrices do not modify the norm of vectors they are applied to, the norm of Eq. 7.25 is equivalent to

$$\|\Sigma\mathcal{V}^T\mathbf{h}\|. \quad (7.26)$$

Let $\mathbf{e} = \mathcal{V}^T\mathbf{h}$, since $\mathcal{V}$ is orthogonal, $\|\mathbf{h}\| = 1$ implies that $\|\mathbf{e}\| = 1$. We thus need to find the unit-norm vector $\mathbf{e}$ that minimizes the norm (squared) of $\Sigma\mathbf{e}$, i.e.,

$$\lambda_1^2 e_1^2 + \dots + \lambda_9^2 e_9^2. \quad (7.27)$$

We can obtain such a vector by concentrating all the (unit) mass of $\mathbf{e}$ where the singular values (λ 's) are smallest, that is by letting

$$e_1 = \dots = e_{9-k} = 0, \quad (7.28)$$

where k is the largest integer such that $\lambda_{9-k+1} = \dots = \lambda_9$, and λ_9 is the last (smallest) singular value of Σ . We have assumed that the singular values appear in decreasing order on the diagonal of Σ , which is usually the case.

From $\mathbf{e} = \mathcal{V}^T \mathbf{h}$, we obtain

$$\mathbf{h} = \mathcal{V} \mathbf{e} = e_1 \mathbf{v}_1 + \dots + e_9 \mathbf{v}_9, \quad (7.29)$$

where $\mathbf{v}_i$ is the corresponding columns of $\mathcal{V}$.

Therefore, all the vectors of the form

$$\mathbf{h} = e_{9-k+1} \mathbf{v}_{9-k+1} + \dots + e_9 \mathbf{v}_9, \quad (7.30)$$

with $e_{9-k+1}^2 + \dots + e_9^2 = 1$ are unit-norm least squares solutions to the homogeneous linear system $\mathcal{A} \mathbf{h} = \mathbf{0}$.

Note:

- when $\lambda_9 > 0$, k is generally equal to 1, as it is highly unlikely that different singular values possess the same numerical value.
- when $\mathcal{A}$ is rank deficient, $k > 1$, this means that we may have numerous singular values equal to zero.

Nonetheless, in any event, if $k = 1$, then the minimum-norm solution is unique and is given by the last singular value of Σ : $\mathbf{h} = \mathbf{v}_9$. For scenarios when $k > 1$, Eq. 7.30 illustrates how to combine the solutions as a linear combination of the last k columns of $\mathcal{V}$.

7.2.6 Random Sample Consensus

In order to estimate the homography between the query and reference image, Eq. 7.20 requires four correct correspondences between the two images. However, in practice, we usually have more than four correspondences with some of them being false matches, i.e. outliers. Solving Eq. 7.20 with even one outlier will result in an erroneous homography estimate. We are now faced with the task of selecting the appropriate correspondences from a set of tentative

correspondences in order to obtain a solution for $\mathcal{H}$. The robust Random Sample Consensus (RANSAC) algorithm [10] addresses this sample selection problem.

The RANSAC algorithm is a simple, yet powerful, statistical method for estimating the parameters of a model given data that might potentially be contaminated by outliers. Not only does RANSAC estimate a global relation that fits the data, but also classifies the data into inliers (data points consistent with the relation) and outliers. Furthermore, RANSAC often finds the correct solution even for data characterized by a high outlier ratio, however, the number of iterations required to do so increases exponentially.

The algorithm itself is based on a hypothesize-and-validate framework. Given a set $\mathcal{P}$ of tentative correspondences, subsets of size n_m are randomly sampled (with replacement) from the data, where n_m is the minimal number of samples required to compute a solution. In our application, $n_m = 4$. Once a model has been hypothesized from this minimal subset, it is then validated against all the data points in $\mathcal{P}$. Thereafter, the support of the model is computed, i.e., the number of data points consistent with the model. This process is repeated until the probability of finding a better model becomes lower than a user-specified probability.

We can compute a theoretical bound on the minimum number of iterations, q , required to ensure that at least one of the selected minimal subsets is outlier-free. Let P_I denote the probability that in some iteration RANSAC selects only inliers from $\mathcal{P}$ when it chooses the n_m points from which the model parameters are estimated. Given the true inlier ratio χ , the probability of selecting an uncontaminated sample of n_m inliers is χ^{n_m} , assuming that the n_m points are selected independently. Consequently, $1 - \chi^{n_m}$ denotes the probability that at least one of the n_m points is an outlier. Therefore, the probability that RANSAC never selects a set of n_m points which are all inliers is $(1 - \chi^{n_m})^q$, which is the same as $1 - P_I$:

$$1 - P_I = (1 - \chi^{n_m})^q . \quad (7.31)$$

Solving for q yields:

$$q = \frac{\log(1 - P_I)}{\log(1 - \chi^{n_m})} . \quad (7.32)$$

Since the true inlier ratio χ is not known a priori, an estimate can be established by using the sample which boasts the largest support. This estimate can then be refined as the algorithm progresses.

7.3 Handling Motion-Blur

In situations where the imaging device is continuously susceptible to movement, motion blur needs to be accounted for. Although we train our features on blurred images, the ability to successfully detect the target depends largely on the ability to identify the keypoints in the first place. Thus failure to identify the keypoints in the runtime images leads to overall performance degradation. Thus, a simple solution to overcome the artifacts induced by motion blur is to downsample the image.

We construct a pyramid of downsampled images of η -levels, each time reducing the downsampled image by a factor of 2. In Sect. 8.2.1, we show that $\eta \leq 3$ yields sufficiently good localization results. Note that all the candidate matches from each pyramid level are combined and given to RANSAC to compute an estimate of the homography matrix.

An added bonus of the pyramid is that it also facilitates the detection of targets at multiple scales. One may choose to implement a more robust scale-invariant keypoint detection algorithm. However, such algorithms are typically too costly to be realized on low computing devices and consequently reduce the overall performance by slowing down the system considerably.

7.4 Conclusion

In this chapter, we have described the locality sensitive hashing technique which we use to match query runtime features to the database features in almost $O(1)$ time. Furthermore, we have derived the Least-Squares solution for estimating the homography relationship between the reference image and query image for planar targets using the Direct Linear Transformation. To deal with outliers, we suggest using the Random Sample Consensus algorithm. Lastly, we have presented a simple, yet powerful solution for accounting for motion blur in situations where the imaging device undergoes continuous motion. To this end, we have proposed using a pyramid scheme which has the added advantage of detecting targets at higher resolutions.

The discussion pertaining to the proposed methodology of our target localization system now ends with the completion of this chapter. In the following chapter we will empirically verify the effectiveness and efficiency of our system.

Chapter 8

Experimental Evaluation

In this chapter, we empirically verify the effectiveness and efficiency of the model generation framework for detecting planar targets. In addition to the three BRIEF descriptors, we also implement the HIP feature descriptor using our model generation framework. The HIP descriptor serves as a benchmark to compare against our BRIEF descriptors. We first conduct an extensive parameter tuning experiment to determine the optimal parameter configuration for the model generation framework. Thereafter, we provide an overall evaluation of three variants of the model generation framework. We then compare our BRIEF descriptors to other well known descriptor-based approaches including SIFT [23] and SURF [6]. Finally, we evaluate our BRIEF descriptors on two external datasets comprising of outdoor scenes.

8.1 Experimental Setup

8.1.1 Dataset

We have collected five datasets for different types of targets with each dataset comprising of around 250 to 300 images. The targets are shown in Figure 8.1(a). The datasets were captured with an LG optimus 2X smartphone camera with a resolution of 480×480 . The camera was rotated by approximately 45° in all directions (i.e. 45° in- and out-of-plane rotation). The scale of the target varied from full resolution (where the target fully occupies the frame) to one third resolution (where the target occupies no more than a third of the frame). Sample images from the datasets are shown in Figure 8.1(b). A majority of the images suffer from further perspective distortions and severe motion blur in some cases.

The ground truth target locations were manually obtained by identifying the four corners



(a)



(b)

Figure 8.1: Dataset used for the evaluation of the model generation framework. (a) Target images (from left to right): advertisement, book cover, football photograph, printed map, and text-based image. (b) Sample images from the datasets.

of the target in each dataset.

8.1.2 Evaluation Metrics

The following performance metrics were used for evaluation: *accuracy*, *precision*, *model size* ($|\hat{F}|$), *file size* (F_s), and *training time*.

The accuracy of a detection is determined by analyzing the maximum error between the estimated target corner locations to the corresponding ground truth corner location. The maxi-

mum error is obtained as follows:

$$\mathcal{E}(\tilde{C}) = \max_j \|C_j - \tilde{C}_j\|, 1 \leq j \leq 4, \quad (8.1)$$

where C_j and $\tilde{C}_j$ denotes the ground truth and estimated location of corner j . The four corners are arranged in the following manner: top left (C_1), top right (C_2), bottom right (C_3), and bottom left (C_4). The target is successfully detected if $\mathcal{E}(\tilde{C}) \leq \delta$. We set δ to 10 to allow for errors up-to 10 pixels in target localisation to account for severely blurred images in the datasets. In these cases, it was nearly impossible to precisely specify the ground truth corners. Furthermore, since we do not refine the estimated homography (as it increases the overall matching rate), setting $\delta = 10$ allows for small errors arising from the homography estimation stage.

The precision metric represents the inlier ratio which is computed as the fraction of correct matches over the total number of matched points. The number of correct matches are those matches that are geometrically verified based on the known camera positions.

$$\text{Precision} = \frac{\# \text{Correct Matches}}{\# \text{Matches}} \quad (8.2)$$

We allow an error of up-to 5 pixels in identifying correct matches.

The model size metric specifies the average number of features contained in the model, whereas the file size metric relates the actual size of the model on disk. We store the x and y reference position of a feature using two 16-bit unsigned integers. The file size (in bytes) is thus calculated as follows:

$$F_s = |\hat{F}| \times (4 + \nu), \quad (8.3)$$

where ν is the descriptor size in bytes.

The training time metric quantifies the time taken to generate the model and is measured in minutes.

8.2 Parameter Tuning

The parameters that need to be configured for the model generation framework are summarized in Table 8.1. Certain parameters for the viewset generation process can be configured by simply analyzing the deformation space for which the target is to be detected from. These parameters include the rotation ranges (around all three axes) as well as the scale range. If memory is not a constraint, the step sizes ($\Delta\Phi$ and γ) can be set to sufficiently small values resulting in

Process	Symbol	Description
Viewset generation	$[\Phi_{min}^x, \Phi_{max}^x]$ $[\Phi_{min}^y, \Phi_{max}^y]$ $[\Phi_{min}^z, \Phi_{max}^z]$ $\Delta\Phi_x, \Delta\Phi_y, \Delta\Phi_z$ $[S_{min}, S_{max}]$ γ M	out-of-plane rotation range around x axis out-of-plane rotation range around y axis in-plane rotation range about z axis rotation step angles scale range scale step number of virtual images
Stable keypoint identification	Q W M	number of stable keypoints sampling size number of images to generate (global stability)
Spatial clustering	δ_K	min. distance to neighboring keypoint
Orientation clustering <i>Graph-based HCA</i> <i>Probabilistic clustering</i>	ϵ ϵ ρ N_c	max. distance to neighboring vertex max. distance from mode min. probability threshold desired number of clusters
Feature stability verification	Ω_S	stability threshold

Table 8.1: Model generation framework parameters.

a fairly large model database with maximal detection accuracy capability. However, since our focus is on hand-held devices with limited resources, we need to find a reasonable trade-off between the step sizes and the system performance in terms of accuracy, precision, model size, and training time.

The choice of a particular orientation estimation algorithm can potentially bias the selection of the parameters. This is rather undesirable as it is required to find the optimal permutation of parameters for the framework which will maximize the performance for any descriptor given an arbitrary orientation estimator. Hence, we first find the optimal set of values for the framework without the orientation normalization process. Furthermore, by ignoring the orientation normalization process, we effectively reduce the number of parameters that need to be investigated. This is essential since the impact of varying each parameter has an exponential effect on the total number of models that need to be generated and evaluated. In addition, it becomes exceptionally difficult to analyze the results if too many parameters are varied

simultaneously.

For these reasons we divide the parameter tuning experiment into two parts: (i) *without orientation normalization* (Sect. 8.2.1) and (ii) *with orientation normalization* (Sect. 8.2.2). In the first part, the following parameters are investigated: S_{max} , $\Delta\Phi_z$, γ , M , Q , W and Ω_S . In the second part, the orientation normalization parameters are investigated along with the orientation clustering algorithms. The optimal parameter values found in the first part will be used in the latter.

Viewset Generation: Rotation

In most hand-held devices for Augmented Reality applications, the deformation space is typically specified by a rotation range of $[-45^\circ, 45^\circ]$ around all three axes, with the scale of the target ranging from full occupancy of the screen to that of a half. The test datasets described in Sect. 8.1.1 were captured to model this deformation space.

Through the outcomes of numerous experiments, we have observed that virtual images generated by random out-of-plane rotations (around both x and y axes) of no more than 30° are sufficient to detect targets with perspective distortions up to 45° . Hence, we set the out-of-plane rotation parameters as follow $[\Phi_{min}^x, \Phi_{max}^x] = [\Phi_{min}^y, \Phi_{max}^y] = [-30^\circ, 30^\circ]$. The in-plane rotation range as well as step sizes are subjective to the use of orientation normalization.

If orientation normalization is applied, $[\Phi_{min}^z, \Phi_{max}^z] = [-180^\circ, 180^\circ]$ with $\Delta\Phi_z = 360^\circ$. This is because the orientation estimator is responsible for providing invariance to in-plane rotation by estimating the local orientations for each keypoint. A limitation of most cheap orientation estimators is that they are highly susceptible to perspective transformations. To decrease the amount of perspective deformation present in the virtual images, the out-of-plane rotation ranges are partitioned into two bins each, with $\Delta\Phi_x = \Delta\Phi_y = 30^\circ$.

When orientation normalization is not applied, in-plane rotation invariance must be handled by setting $\Delta\Phi_z$ as small as possible to restrict the appearance of the patches in each viewpoint bin. The in-plane rotation range is set to model the expected deformation space, which for the test datasets is simply: $[\Phi_{min}^z, \Phi_{max}^z] = [-45^\circ, 45^\circ]$. Since orientation estimation is not performed, there is no need to worry about the inherent perspective distortions in the virtual images, thus, $\Delta\Phi_x = \Delta\Phi_y = 60^\circ$. It should be noted that more accurate models can be generated by setting the out-of-plane rotation step sizes to smaller values (as will be seen in Sect. 8.3). However, this comes at the cost of significantly longer training times as well as bulkier model databases.

Viewset Generation: Scale

The scale viewpoint bins are configured as follows. Given that the targets in the dataset range from full resolution (480×480) to one third resolution. Therefore, the scale range can be set: $[S_{min}, S_{max}] = [1/3, 1] \times 480 = [160, 480]$. An important aspect worth investigating is the influence of the parameter S_{max} on the system performance. Does training with a larger resolution than that of the expected runtime video resolution offer better results? Or, on the contrary, does training with a smaller resolution yield any gains?

Stable Keypoint Identification

Through preliminary experiments, we were also able to establish that the local stability keypoint selection method is better than the global stability method. We verify this in Sect. 8.2.1. Hence, for both parts of the parameter tuning experiment, the local stability method is used. The spatial clustering parameter is set to: $\delta_K = 2$.

8.2.1 Parameter Tuning: Without Orientation Normalization

Finding the Optimal Configuration

In search of the optimal configuration for the model generation framework parameters (i.e. the configuration with best target detection rate for a given model size), we exhaustively evaluated numerous possible configurations. The values investigated for each parameter are as follows: $S_{max} = \{600, 400, 200, 100\}$; $\Delta\Phi^z = \{5^\circ, 10^\circ, 15^\circ, 20^\circ\}$; $\gamma = \{0.7, 0.8, 0.9\}$; $M = \{50, 100, 200\}$; $Q = \{50, 75, 100\}$; $W = \{1, 2, 3, 4\}$; and $\Omega_S = \{128, 192\}$ (this corresponds to 50% and 75% of the BRIEF descriptor length - which is 256 bits). The total number of possible combinations is therefore 1728 (twice this number for sBRIEF and s+BRIEF). Taking into consideration all four descriptors, a total of 10368 models were generated.

Evaluating over 10000 models is quite a daunting task, hence, for this section only, it was decided to filter the results by applying slightly stricter thresholds for computing the accuracy and precision. For the accuracy, we allowed up to 7 (instead of 10) pixels in error from the ground truth target corners, whereas, for precision, we allowed up to 4 (instead of 5) pixels in error in determining correct matches.

In addition to the model generation framework parameters, we also investigated various matching thresholds ($\mathcal{Z}$) for each descriptor. The $\mathcal{Z}$ values are presented in Table 8.2. For the modified BRIEF descriptors, the matching thresholds are specified in terms of a percentage of the descriptor length as follows. cBRIEF & sBRIEF: $\{15\%, 17.5\%, 20\%, 22.5\%\}$; and

s+BRIEF: $\{2\%, 4\%, 6\%, 8\%\}$. The last column in Table 8.2, $\mathcal{Z}_{\text{best}}$, presents the best matching threshold which was found to yield the most accurate results for a majority of the configurations. Unless otherwise stated, these are the matching thresholds that are used for all the parameter tuning experiments.

Descriptor	$\mathcal{Z}$	$\mathcal{Z}_{\text{best}}$
HIP	3, 5, 7, 10	7
cBRIEF	38, 45, 51, 58	45
sBRIEF	38, 45, 51, 58	45
s+BRIEF	5, 10, 15, 20	15

Table 8.2: Matching thresholds experimented.

Figure 8.2 presents the evaluation results of the various parameter configurations investigated using the best matching thresholds from Table 8.2. The results are shown in terms of a plot of the accuracy as a function of the model size ($|\hat{F}|$) for all four descriptors. Apart from using stricter accuracy and precision thresholds, we further reduced the number of potential candidate parameter configurations by only considering those configurations which resulted in accuracies $\geq 50\%$ and precision $\geq 25\%$. It is evident that, when compared to the HIP descriptor, our BRIEF descriptors yield more precise configurations with more configurations resulting in accuracies $\geq 70\%$.

To identify the optimal configuration, we extracted the common configurations from Fig. 8.2 across all 4 descriptors that resulted in an accuracy $\geq 70\%$ and precision $\geq 30\%$. These configurations are presented in Table 8.3. Immediately we observe that for any given set of configuration values, our BRIEF descriptors outperform the HIP descriptor. The W and Ω_S parameters are not shown in Table 8.3. The stability threshold parameter value, for sBRIEF and s+BRIEF, for all the configurations in Table 8.3 is $\Omega_S = 128$. We observed that for the parameter configurations listed in Table 8.3, all the values for W result in accuracies $\geq 70\%$ and precision $\geq 30\%$. Hence, to avoid redundancy in presentation, the W parameter was omitted. We do however present the results for the various values of W later in this section. Note that the results shown are for $W = 1$.

In an attempt to find the optimal configurations (w.r.t. accuracy vs model size trade-off), we shortlisted the configurations from Table 8.3 by considering only those common configurations for all four descriptors which resulted in accuracies greater than 70% with $|\hat{F}| \leq 4500$. The resulting configurations are highlighted in light gray in Table 8.3. From these configurations, we then selected the *optimal* parameter configuration as the following: $S_{max} = 400$, $\Delta\Phi_z = 20^\circ$, $\gamma = 0.9$, $M = 100$, and $Q = 75$. This configuration ensures that all four descriptors

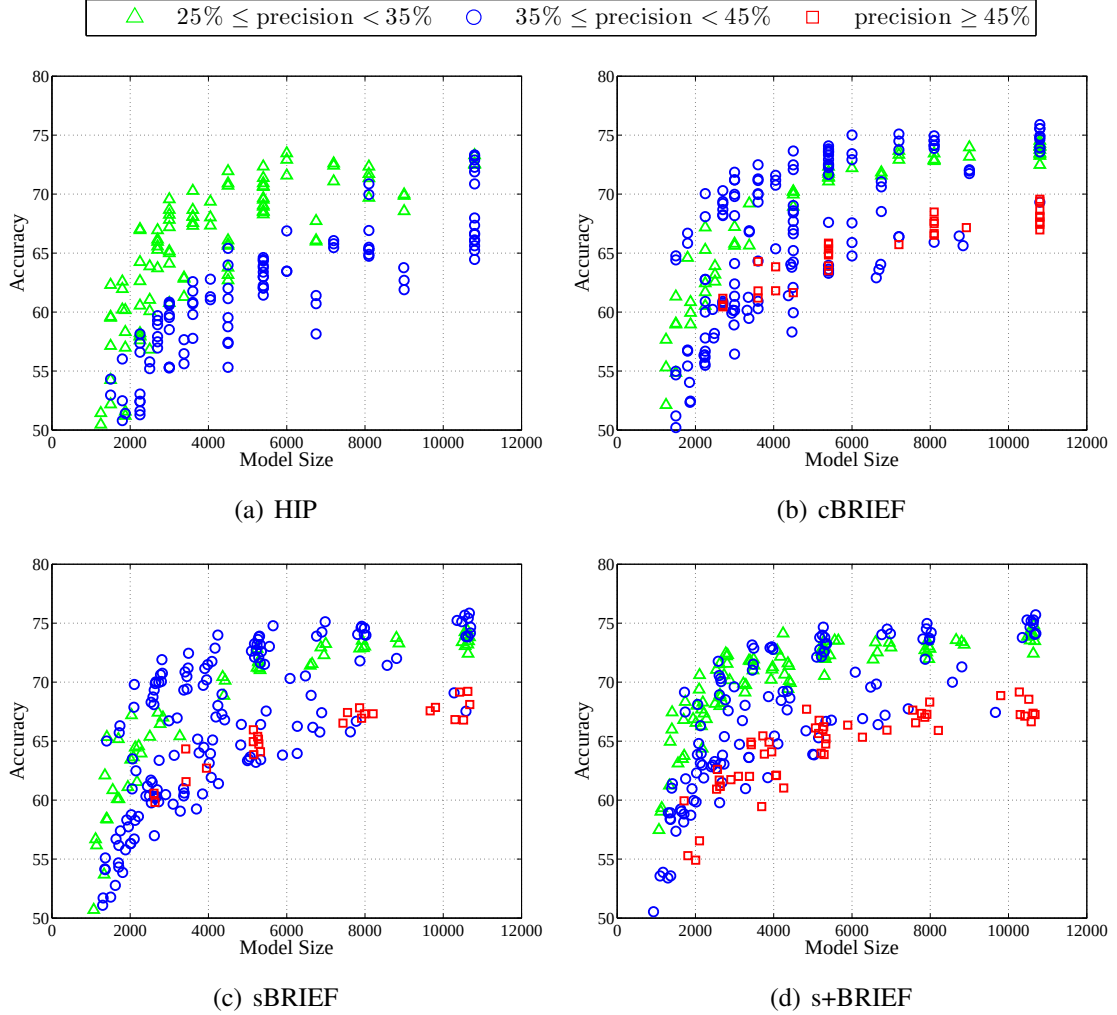


Figure 8.2: Performance evaluation results of the various parameter configurations investigated.

perform similarly in terms of accuracy ($\approx 72\%$) and model size ($|\hat{F}| = 4500$).

It is important to note that we select the optimal configuration with our objective set on minimizing the number of features required to learn a general target while maximizing the accuracy. It is obvious that the more features we incorporate into the database, the better the accuracy - however, observing Figure 8.2 we see that the accuracy saturates at 76% for almost 12000 features. Furthermore, we argue that a gain of 4% in accuracy is not worth increasing the model database by almost 8000 features.

The individual *most accurate* parameter configuration for each descriptor is highlighted in

Parameters					HIP		cBRIEF		sBRIEF		s+BRIEF	
S_{max}	$\Delta\Phi_z$	γ	M	Q	Acc	$ \hat{F} $	Acc	$ \hat{F} $	Acc	$ \hat{F} $	Acc	$ \hat{F} $
400	5	0.8	50	50	70.61	5400	72.25	5400	72.08	5285.2	72.55	5285.2
400	5	0.8	50	75	71.35	8100	74.52	8100	74.62	7900.8	74.52	7900.8
400	5	0.8	50	100	71.89	10800	74.81	10800	75.12	10473	75.27	10473
400	5	0.8	100	100	72.56	10800	73.57	10800	73.84	10632.2	74.99	10632.2
400	5	0.8	200	100	70.86	10800	73.94	10800	74.17	10693.6	75.71	10693.6
400	5	0.9	50	50	72.49	10800	73.92	10800	73.87	10577.2	75.19	10577.2
400	5	0.9	100	50	72.89	10800	75.9	10800	75.85	10668.2	74.05	10668.2
400	5	0.9	200	50	72.27	10800	74.64	10800	74.66	10698.4	74.12	10698.4
400	10	0.8	200	100	70.92	5400	73.83	5400	73.9	5304.8	73.78	5304.8
400	10	0.9	50	75	71.71	8100	73.88	8100	74.04	7812.8	73.66	7812.8
400	10	0.9	50	100	73.3	10800	75.55	10800	75.23	10353.6	73.78	10353.6
400	10	0.9	100	50	70.68	5400	73.59	5400	73.84	5290	72.58	5290
400	10	0.9	100	75	72.34	8100	74.94	8100	74.74	7917.6	74.97	7917.6
400	10	0.9	100	100	73.19	10800	75.55	10800	75.68	10549.4	74.66	10549.4
400	10	0.9	200	75	70.87	8100	74.55	8100	74.57	7987.4	73.71	7987.4
400	10	0.9	200	100	73.32	10800	74.92	10800	75.42	10640.4	75.27	10640.4
400	15	0.9	50	75	72.34	5400	72.94	5400	72.63	5098.6	72.12	5098.6
400	15	0.9	50	100	72.41	7200	73.75	7200	73.9	6762.8	74.03	6762.8
400	15	0.9	100	100	71.08	7200	74.49	7200	74.25	6899	74.5	6899
400	15	0.9	200	50	70.28	3600	72.49	3600	72.45	3491.8	72.87	3491.8
400	15	0.9	200	75	71.37	5400	73.55	5400	73.19	5242	73.98	5242
400	15	0.9	200	100	72.61	7200	75.09	7200	75.12	6983	74.11	6983
400	20	0.9	50	75	70.71	4500	72.08	4500	71.75	4088.6	72.67	4088.6
400	20	0.9	50	100	72.91	6000	73.42	6000	71.51	5433.4	72.28	5433.4
400	20	0.9	100	75	71.95	4500	72.48	4500	72.88	4168.4	72.16	4168.4
400	20	0.9	100	100	71.58	6000	72.92	6000	73.03	5557.6	73.53	5557.6
400	20	0.9	200	75	70.91	4500	73.65	4500	73.99	4239.2	74.13	4239.2
400	20	0.9	200	100	73.46	6000	75.01	6000	74.78	5651	73.52	5651
600	5	0.7	50	100	70.1	9000	72.75	9000	72.87	8661.2	73.57	8661.2
600	10	0.9	200	100	70.44	10800	74.53	10800	74.38	10617	74.07	10617

Table 8.3: Resulting configuration values for the various binary descriptors for which the accuracy $\geq 70\%$ and precision $\geq 30\%$. Configurations for which $|\hat{F}| \leq 4500$ are highlighted in light gray. The optimal configuration is shown in bold whereas the individual most accurate configuration for each descriptor is highlighted in dark gray.

dark gray in Table 8.3.

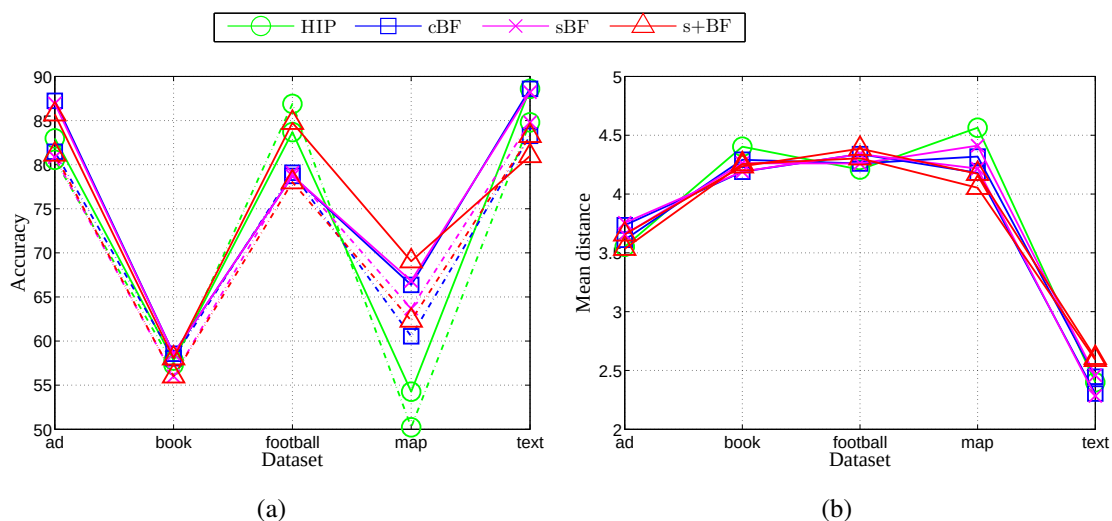


Figure 8.3: Dataset analysis results. The most accurate and optimal configurations are represented by the solid and dashed lines respectively.

Dataset Analysis

Having obtained the optimal and most accurate configurations for each descriptor, we now present an analysis on the individual datasets. Plots of the the mean accuracy and average maximum error (Eq. 8.1) are shown in Fig. 8.3.

On average, apart from the football dataset, our BRIEF descriptors perform the best. Due to the local symmetry in regions around most of the keypoints, the most challenging datasets comprise of the book and map targets. Hence the results are slightly lower for all four descriptors. The HIP descriptor fairs quite poorly on the map dataset, with accuracy $< 55\%$.

The advantage of using the stability vector for training and matching is demonstrated with the map dataset, as we can see that the s+BRIEF descriptor is able to obtain a 70% accuracy. The stability vector in the matching ensures that unstable bits (in the feature) are discarded thereby increasing the distinctiveness of the descriptors. What is also worth noting is that for the text target, s+BRIEF descriptor performs the worst - this could be due to the sparsity and lack of texture in the image which forces many bits to be deemed as unstable.

Overall, for a successful match, it is evident from Figure 8.3 that the maximum error to the ground truth is less than 5 pixels.

Stability Threshold

We had previously mentioned that the optimal stability threshold parameter value was found to be $\Omega_S = 128$ which corresponds to 50% of a 256-bit descriptor. We now present an analysis of the effect of the stability threshold parameter for: $\Omega_S = \{0\%, 50\%, 75\%\}$. Note that for $\Omega_S = 0\%$, the sBRIEF descriptor transforms to the cBRIEF descriptor. Therefore, the cBRIEF descriptor can be regarded as a special case of the sBRIEF descriptor. In addition to the stability threshold parameter, we also investigate the influence of the S_{max} parameter.

We analyze the stability threshold for the following parameter configurations: $S_{max} = \{600, 400, 200, 100\}$; $\Delta\Phi^z = \{5^\circ, 10^\circ, 15^\circ, 20^\circ\}$; $\gamma = \{0.7, 0.8, 0.9\}$; $M = \{50, 100, 200\}$; and $Q = \{50, 75, 100\}$. We set the stratified sampling parameter $W = 1$. The results are shown in Figure 8.4.

We can see that a stability threshold of 50% is the optimal choice for both sBRIEF and s+BRIEF. Although cBRIEF ($\Omega_S = 0\%$) is able to produce a similar accuracy to sBRIEF (at $\Omega_S = 50\%$), it generally requires more features to do so.

As evident from Fig. 8.4, the stability threshold is inversely proportional to the model size. By increasing the stability threshold we increase the rigidity of the criterion for accepting features. Therefore we reject more “unstable” features, which results in a smaller $\hat{F}$. However, having an extremely rigid stability threshold yields poor performances as too many descriptors are discarded, potentially resulting in fewer runtime matches, as is the case of $\Omega_S = 75\%$.

There are several cluster trends visible in Fig. 8.4 which are due to the various S_{max} parameter configurations. We can see that when a model is generated at a maximum resolution significantly lower than that of the runtime video resolution, then the matching performance is severely degraded, as seen for $S_{max} = 200$. We also observe that the best results are obtained for $S_{max} = 400$. This corresponds to a 15% deviation from the maximum video resolution 480×480 .

In general, we have experimentally established that when the runtime video resolution is known a priori, the best matching results are obtained when the target is trained with $S_{max} = \pm 20\%$ of the runtime resolution.

Stable Keypoint Identification

Using the the most accurate and optimal configurations found earlier, we now present an investigation in order to verify which of the two keypoint stability identification methods are better: local stability or global stability. Also included in this investigation is the influence of the stratified sampling parameter. To this end, we evaluate the following values: $W = \{1, 2, 3, 4\}$.

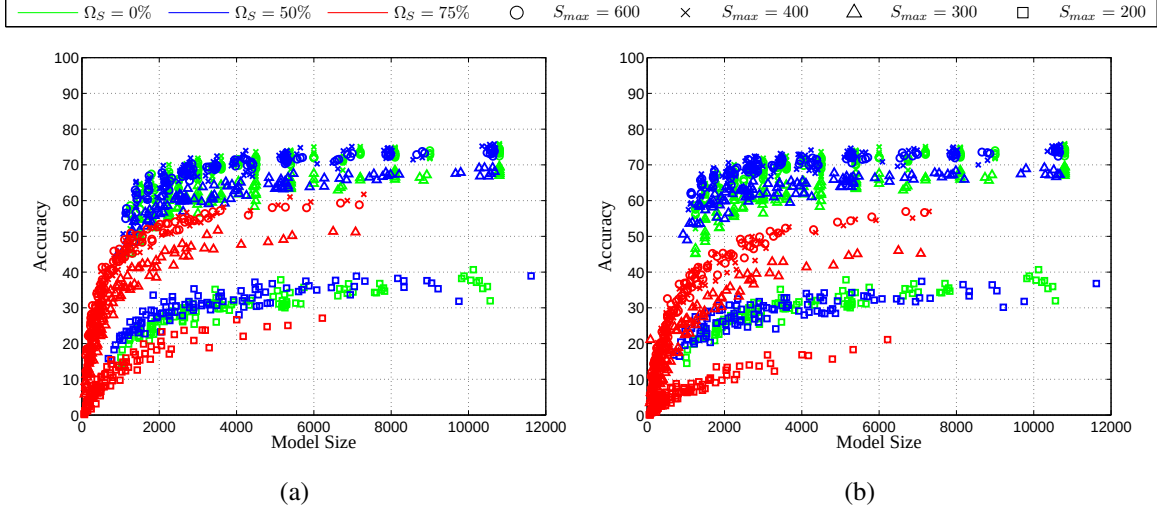


Figure 8.4: Stability Threshold Experiment: (a) sBRIEF, (b) s+BRIEF.

Note that $W = 1$ is a special case of stratified sampling in which the subset is equivalent to that of the population. For the global stability method, we set the number of virtual images to be generated as $M = 1000$. Since we select the same number of stable keypoints ($Q = 75$), the model size remains unchanged. Plots of the accuracy and precision for the various values of W are shown in Fig. 8.5.

The local stability method outperforms the global stability method by almost 20% in accuracy and 10% in precision. The reason why the latter performs poorly can be explained as follows. For the global stability method, partial features are extracted from every image in the viewpoint bin, irrespective of whether the keypoint can be detected or not in a particular image. These features are then used to obtain the final aggregated feature. When too many partial features are aggregated, the final feature tends to be over-smoothed and is thus potentially not distinctive enough. This is where the stability vector becomes useful, as demonstrated by the s+BRIEF descriptor. By exploiting the stability vector during matching (with regard to the stability of each feature element), the s+BRIEF descriptor outperforms all the other descriptors.

Since the targets (in their respective datasets) are not occluded, the effect of stratified sampling is not that evident in Fig. 8.5. However, in the event where certain parts of the target are occluded, stratified sampling (with an applicable W) guarantees that the model database $\hat{F}$ contains features extracted from all segments of the target, thereby strengthening the system's robustness to occlusion. Nonetheless, we can see that (for the given test datasets) the optimal sampling sizes for each descriptor is: HIP ($W = 4$), cBRIEF & sBRIEF ($W = 3$), and s+BRIEF ($W = \{2, 3\}$).

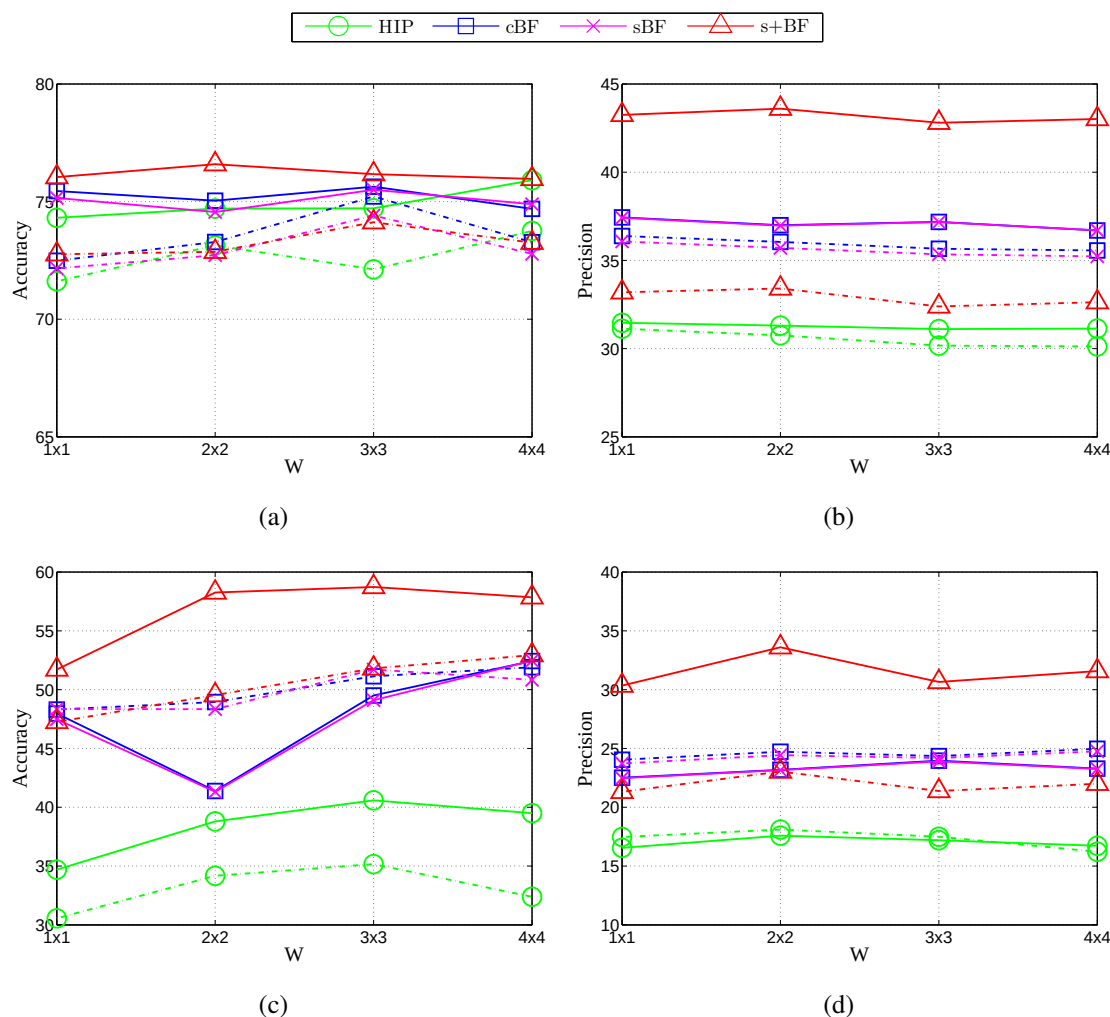


Figure 8.5: Local vs Global Stability Experiment: (a,b) Local stability; (c,d) Global stability. The most accurate and optimal configurations are represented by the solid and dashed lines respectively.

Runtime Parameters

Matching Threshold

Table 8.2 presents the matching thresholds investigated for each descriptor. Using the optimal and most accurate configurations found, we now demonstrate the effect of each matching threshold on the respective descriptors. Since the matching threshold parameter does not influence the model size, we illustrate the matching performance as a plot of the accuracy vs precision as shown in Fig. 8.6.

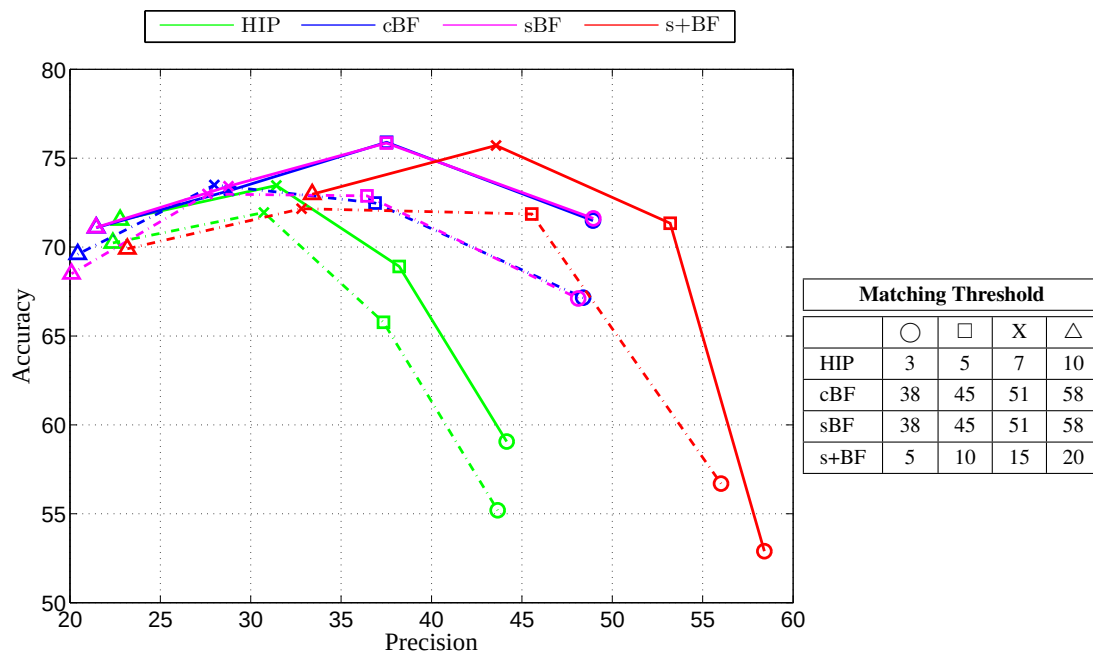


Figure 8.6: Matching Threshold Experiment. The most accurate and optimal configurations are represented by the solid and dashed lines respectively.

From the results obtained, it is evident that the best matching thresholds for both configurations for each descriptor lie in the following intervals:

1. HIP : $\mathcal{Z} \in [5, 10]$,
2. cBRIEF : $\mathcal{Z} \in [38, 51]$,
3. sBRIEF : $\mathcal{Z} \in [38, 51]$, and
4. s+BRIEF : $\mathcal{Z} \in [10, 20]$,

Note that for all four descriptors, the lower the matching threshold is, the better the precision. A more precise matching scheme ensures a higher signal to noise ratio (SNR) in the matched correspondences. A higher SNR is desirable as it implies that there are less outliers, and thus consequently facilitating a faster convergence rate for the RANSAC algorithm.

Pyramid Levels

In order to address motion blur in runtime images (without using a scale-invariant keypoint detector), we had suggested using an η -level pyramid of downsampled images in Sect. 7.3.

This experiment serves to investigate the appropriate number of pyramid levels (η) to compute as well as the effect of each pyramid level.

Hence, once again, using the most accurate and optimal configurations for all four descriptors, we investigate the following runtime scenarios:

- matching with one pyramid level at the original frame resolution (no down-sampling),
- matching with one pyramid level at half the frame resolution (50% down-sampling),
- matching with one pyramid level at a quarter of the frame resolution (75% down-sampling),
- matching with two pyramid levels (level 1 = original frame resolution, level 2 = $1/2$ frame resolution), and
- matching with three pyramid levels (level 1 = original frame resolution, level 2 = $1/2$ frame resolution, level 3 = $1/4$ frame resolution).

Since neither the value of η nor the resolution of the runtime images influences the model size, we present the matching performance as a plot of the accuracy vs precision as shown in Figure 8.7. We can see that 2 pyramid levels are sufficient to obtain very good matching results. Furthermore, at a resolution of 480×480 , a 3-level pyramid does not provide any significant gain in accuracy or precision to that of 2-levels. The reason why an accuracy of 0% is obtained at a scale of $1/4$ is due to the fact that at 25% of a 480×480 image (i.e. resolution = 120×120), the image is potentially too small to establish sufficient matches.

The advantage of using a 2-level pyramid scheme at runtime is visually illustrated in Fig. 8.8 using the optimal configuration for the cBRIEF descriptor. This image was captured while the camera was being moved and thus suffers from motion blur. In Fig. 8.8(a) cBRIEF could only identify 4 correct matches with a single pyramid (at full resolution) with too many false matches (outliers), preventing RANSAC from correctly estimating the pose. However, when matched with 2 levels, cBRIEF could establish more than 20 correct matches, and thus allowing RANSAC to compute the pose correctly as shown by the red bounding box in Fig. 8.8(b).

Figure 8.9 presents a break-down of the time (per frame in milliseconds) required to perform keypoint detection, feature extraction, feature matching and pose estimation for each descriptor-pyramid level combination for the most accurate and optimal configurations. Note that the matching rates are presented for un-optimized implementations. We can see that more than 60% of the time spent in detecting a target is spent on the pose estimation step, specifically the RANSAC algorithm. Therefore, significantly better matching rates can be immediately obtained by replacing RANSAC with a more efficient homography estimator such as

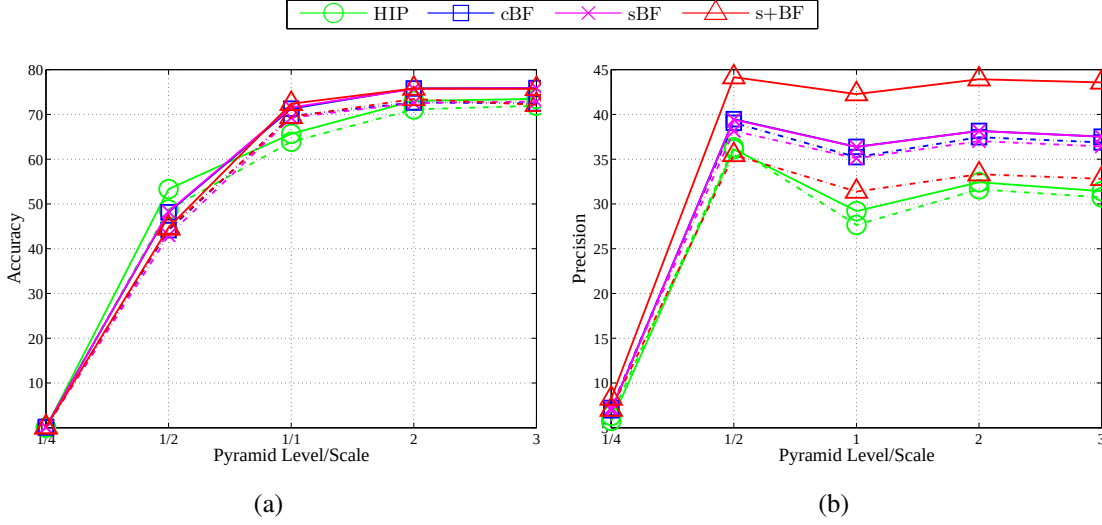


Figure 8.7: Pyramid Experiment. The most accurate and optimal configurations are represented by the solid and dashed lines respectively. (a) Accuracy vs Pyramid Level / Scale. (b) Precision vs Pyramid Level / Scale.

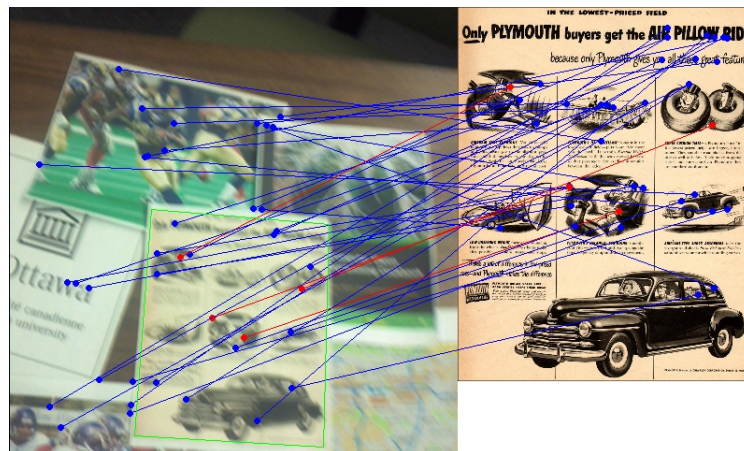
ARSAC [32]. Nonetheless, even on the un-optimized code, we do not take more than 8ms (average worst case) to detect, extract and match features, and an additional 5 to 10 ms for pose estimation.

8.2.2 Parameter Tuning: With Orientation Normalization

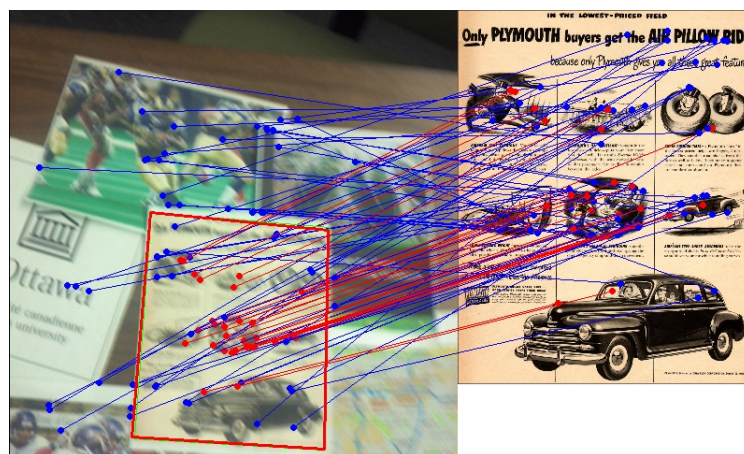
Now that we have established the optimal parameter configuration for the model generation framework, we can complete the parameter tuning experiment by optimizing the parameters for the orientation normalization process. Furthermore, we are also interested in finding out which of the two orientation clustering algorithms provides the best performance gains w.r.t. accuracy and precision. To this end, we evaluate both algorithms on two different types of orientation estimators. The first being the more reliable intensity centroid estimator [33], and the second comprising of the faster yet slightly inconsistent, Taylor's orientation assignment method [39]. Both of these methods are described in Sect. 5.1.2.

Orientation Estimation Analysis

Before we can proceed to optimize the parameters of the orientation clustering algorithms, we must first investigate the performance of the orientation estimators on the five test targets



(a)



(b)

Figure 8.8: Successful target detection using cBRIEF with 2-pyramid levels. (a) Matching with 1 pyramid level (at full resolution). (b) Matching with 2 pyramid levels. The green and red bounding boxes represent the ground truth and estimated target location respectively. The red and blue lines indicates true and false matches respectively.

(Fig. 8.1(a)). The results of this investigation will provide valuable insight when analyzing the orientation clustering algorithms in subsequent sections.

To measure the consistency of the orientation estimators for each target, we generate an artificial viewset (of the target) comprising of 8 viewpoint bins, each with 500 virtual images. The viewpoint bins are specified by random out-of-plane rotations, around both x and y axes, from $[-30^\circ, 30^\circ]$ and random in-plane rotations from $[-180^\circ, 180^\circ]$. The scale of each viewpoint bin ranges from $[0.3, 1]$ with an incremental step size of 0.1.

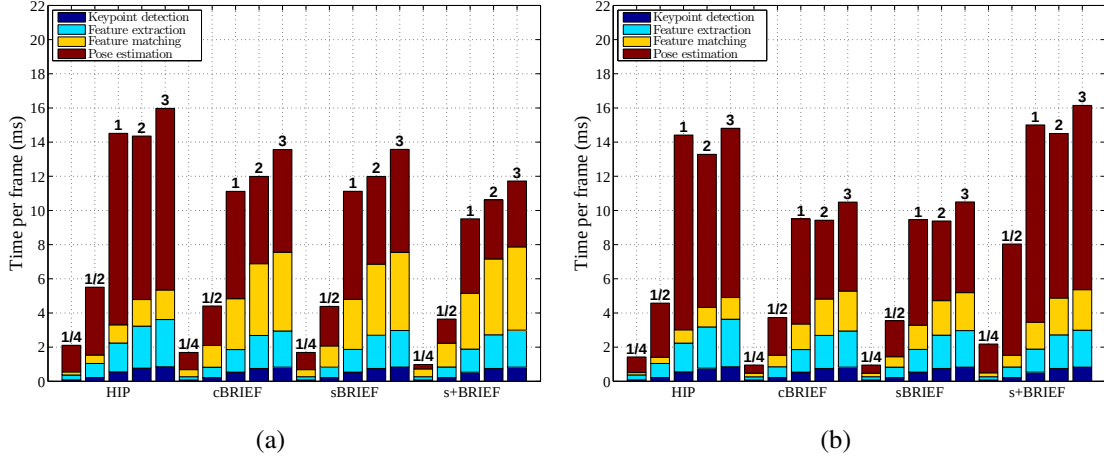


Figure 8.9: Time per frame (milliseconds) for the different components during matching at various pyramid levels: (a) most accurate configuration (b) optimal configuration.

The orientation estimators are evaluated as follows. After detecting the keypoints from each virtual image, we apply spatial clustering to identify the 100 most locally stable keypoints from each viewpoint bin (with $W = 4$). We then compute the orientation for all the keypoints (from each stable keypoint group) and project them on to the reference image. The mean deviation from the average orientation of each stable keypoint group is then computed from the reference image. From these mean deviation values, we then determine the overall mean deviation, μ_d , per viewpoint bin. Figure 8.10 presents the plots of μ_d for both orientation estimators for all five targets.

We observe a larger variance in the values of μ_d for Taylor's orientation estimator as compared to the intensity centroid orientation estimator. This is expected as we have seen from the results of Sect. 5.1.4 that Taylor's orientation estimator is more inconsistent as compared to the intensity centroid method. Note that a larger deviation from the mean orientation of each stable keypoint group, implies a larger variation in the appearance of the patches from which the features are extracted. We can therefore anticipate to see the orientation clustering algorithms having a greater influence on the system performance when Taylor's orientation assignment method is applied.

Optimizing the Orientation Clustering Parameter (ϵ)

The main parameter that needs to be optimized for both clustering algorithms is ϵ . For the Graph-based hierarchical clustering algorithm, ϵ specifies the maximum deviation from the

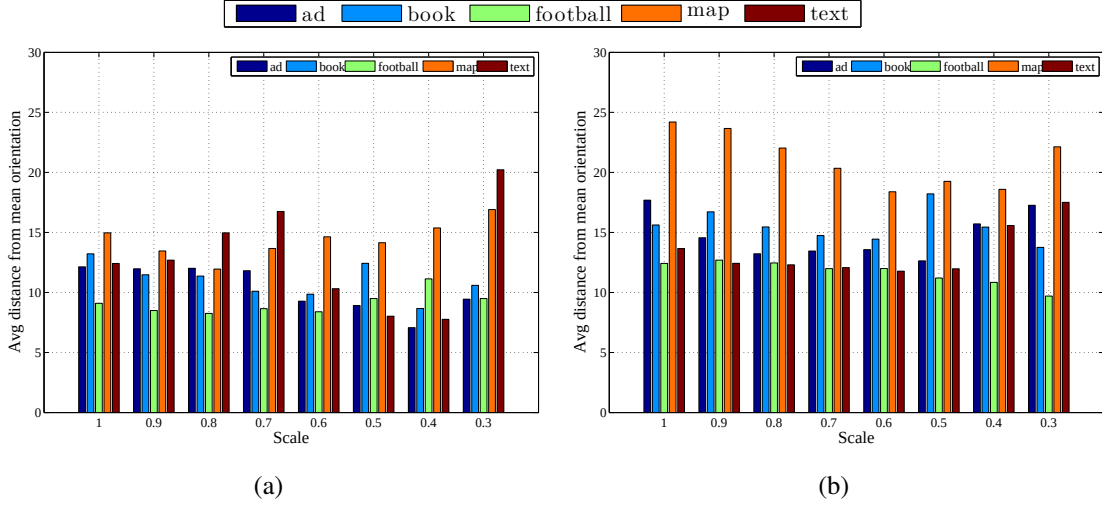


Figure 8.10: The mean deviation from the average orientation per stable keypoint group for each viewpoint bin (μ_d). (a) Intensity centroid orientation estimator. (b) Taylor's orientation estimator. Note that the x -axis is labeled by the scale since it is the only parameter that specifies the viewpoint bin.

medoid of a cluster to all its nodes. Whereas for the Probabilistic clustering algorithm, ϵ specifies the maximum deviation from the mode of the cluster to all its nodes. Taking note of the fact that if ϵ is set too high, the advantage of orientation clustering is lost. Hence, we investigate the following set of values for both algorithms: $\epsilon = \{10^\circ, 15^\circ, 20^\circ, 30^\circ\}$.

The Probabilistic clustering algorithm has two additional parameters that need to be configured: ρ and N_c . The parameter ρ specifies the minimum probability threshold for accepting a cluster. Therefore, the number of clusters found is dependent on ρ . The lower ρ is, the likelihood of retaining more clusters is high, conversely, a high value for ρ ensures less, yet better quality clusters. The parameter N_c specifies the desired number of regions we wish to search, We set these parameters to the following values: $\rho = 0.3$ and $N_c = 7$.

The remaining model generation framework parameters, presented in Table 8.4, are set to the optimal parameter configuration found in Sect. 8.2.1.

The evaluation results of the various ϵ configurations for both orientation estimators are shown in Fig. 8.11. The average model size for each descriptor are as follows:

- HIP & cBRIEF: $|\hat{F}| \in [3500, 3600]$, and
- sBRIEF & s+BRIEF: $|\hat{F}| \in [2800, 3200]$.

It is evident from Fig. 8.11 that the intensity centroid orientation estimator yields more pre-

Parameter	Value
Φ_x, Φ_y	$[-30^\circ, 30^\circ]$
$\Delta\Phi^x, \Delta\Phi^y$	30°
Φ_z	$[-180^\circ, 180^\circ]$
$\Delta\Phi^z$	360°
$[S_{min}, S_{max}]$	$[160, 400]$
γ	0.9
M	100
Q	75
W	4
Ω_S	128

Table 8.4: Parameter configuration values for the parameter tuning experiment (with orientation normalization).

cise results than Taylor’s orientation assignment method. Although both clustering algorithms demonstrate similar performances, the Graph-based HCA tends to yield higher precision gains. As we had predicted earlier in the orientation estimation section, both clustering algorithms tend to have a significant impact on the more inconsistent orientation estimator, i.e. Taylor’s method.

It is interesting to note that the ϵ parameter does not influence cBRIEF and sBRIEF that much. However we do see that a large ϵ yields slightly lower accuracy and precision results. From this observation, we can infer that the orientation clustering stage can potentially be omitted for both these descriptors. Furthermore, apart from decreasing the model size (by rejecting unstable features), the sBRIEF feature does not offer any significant performance gain over the cBRIEF descriptor.

Nonetheless, from the results obtained, it is evident that the ϵ parameter greatly influences the precision of the system, especially for the HIP and s+BRIEF descriptors. For both orientation clustering algorithms, a smaller ϵ reduces the variation in the orientation clusters, thereby increasing the similarity in the appearances of the image patches encompassing the keypoints. This results in the extraction of similar partial features and consequently resulting in more stable and representative aggregated features. A configuration yielding a higher precision is generally preferred as it ensures a greater inlier to outlier ratio, allowing the RANSAC algorithm to estimate the homography in fewer iterations. Table 8.5 presents the optimal ϵ parameter values for each descriptor. These values were selected to maximize both the accuracy and precision, with more priority given to the precision performance factor.

Figure 8.12 presents a break-down of the time (per frame in milliseconds) required to

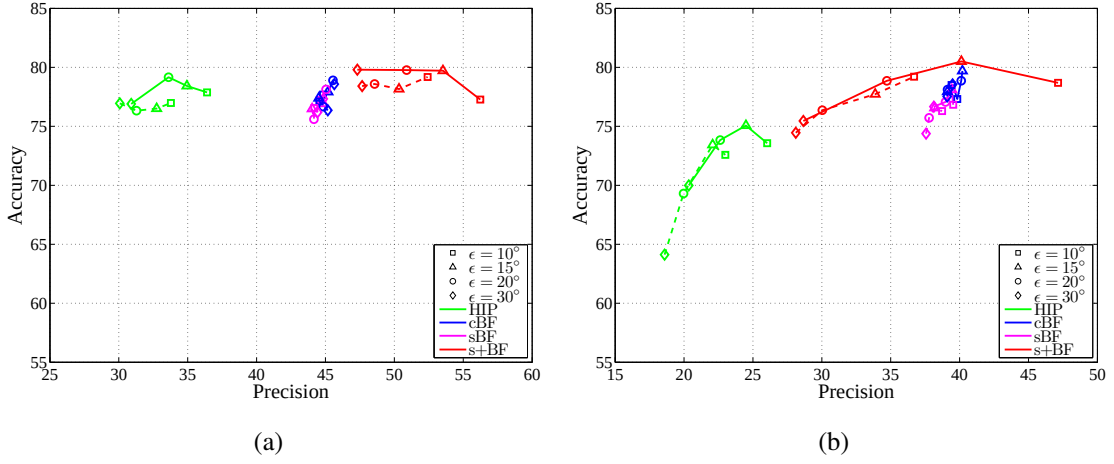


Figure 8.11: Performance evaluation results of the orientation clustering algorithms. (a) Intensity centroid orientation estimator, (b) Taylor's orientation estimator. Solid and dashed lines represent the Graph-based hierarchical clustering and Probabilistic clustering algorithms respectively.

Descriptor	Optimal ϵ -value	
	Graph-based HCA	Probabilistic Clustering
HIP	15°	15°
cBRIEF	20°	10°
sBRIEF	15°, 20°	10°
s+BRIEF	15°	10°

Table 8.5: The optimal ϵ parameter values of the orientation clustering algorithms for each descriptor.

perform keypoint detection, feature extraction, feature matching and pose estimation for the Graph-based HCA for both orientation estimators: $\epsilon = \{10^\circ, 15^\circ, 20^\circ\}$. Note that the matching rates are presented for un-optimized implementations. Since a higher precision enables the RANSAC algorithm to estimate the homography in fewer iterations, slightly faster matching rates are obtained with $\epsilon = 10^\circ$. In general, of the two orientation estimators, the average pose estimation time for the intensity centroid estimator is relatively lower. However, there are two arguments in support of applying Taylor's method for orientation estimation on low-powered devices. Firstly, due to the simplicity of the algorithm, highly optimized assembly code can be exploited to estimate the orientations in virtually no time. Secondly, it is possible to reduce the pose estimation time significantly by using a more efficient homography estimator such as ARSAC [32]. Hence, for the rest of the experiments involving orientation estimation, we

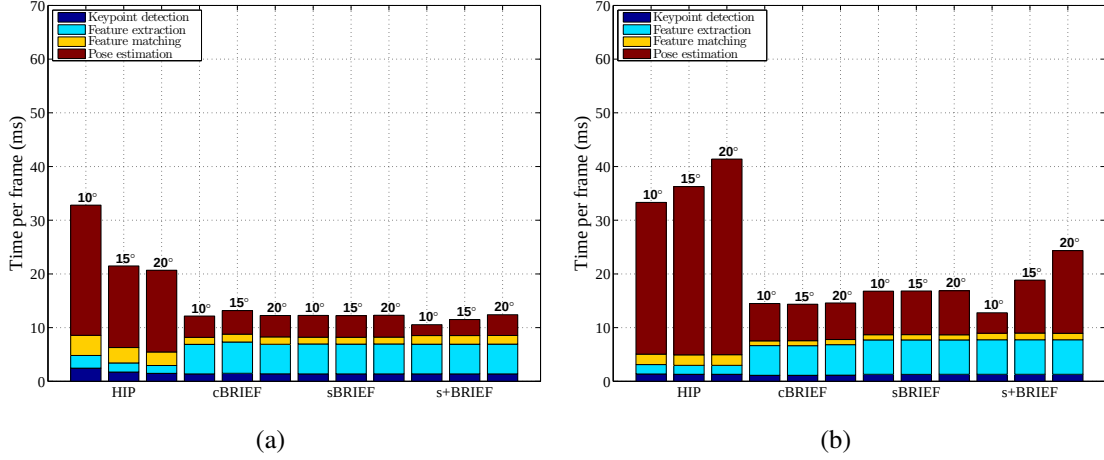


Figure 8.12: Time per frame (milliseconds) for the different components during matching for the Graph-based Hierarchical Clustering Algorithm: $\epsilon = \{10^\circ, 15^\circ, 20^\circ\}$. (a) Intensity centroid orientation estimator. (b) Taylor's orientation estimator.

make use of Taylor's method keeping in mind that better results are possible by applying a more consistent orientation estimator such as the intensity centroid estimator.

8.3 Overall Model Generation Comparison

Up until now, we have focused our attention on identifying the optimal parameter configuration values for the model generation framework. Although we have been able to observe the performance of the system with respect to the accuracy, precision and model size, we have not yet formally investigated the costs involved in generating the models. Of particular concern to us is the actual file size of the model (on disk) as well as the average time required to generate a model (i.e. training time). In addition, we have also not yet investigated whether orientation normalization, specifically orientation clustering, provides any advantage to the overall performance of the model generation framework.

Taking the above into consideration, we now present a thorough investigation comprising of three possible variants of the model generation framework. In the first variant (A), orientation normalization is not performed, therefore, $\Delta\Phi^z$ needs to be as small as possible to limit the variation in patch appearance. From the first part of the parameter tuning experiment, the optimal value was found to be $\Delta\Phi^z = 20^\circ$. For the second variant (B), partial orientation normalization is performed by omitting the orientation clustering stage. In partial orientation normalization, we simply estimate the keypoint orientation and rotate the image patch accord-

ingly in order to extract rotationally invariant features. The third variant (C) comprises of the full model generation framework with orientation normalization. For variants B and C, $\Delta\Phi^z = 360^\circ$. Since variants B and C achieve invariance to the full range of in-plane rotations, in order to ensure that each variation is compared in complete fairness, the in-plane rotation range for variant A is set to: $[\Phi_{min}^z, \Phi_{max}^z] = [-180^\circ, 180^\circ]$. For the orientation normalization component, we apply Taylor’s orientation assignment method with the Graph-based hierarchical clustering algorithm for orientation estimation and clustering respectively. The values for ϵ are configured to that of Table 8.5.

The other framework parameters are configured to the optimal parameter configuration found throughout the parameter tuning experiments: $M = 100$, $Q = 75$, $W \in [3, 4]$, and $\Omega_S = 128$. With regard to the scale viewpoint parameters, we had previously established that $\gamma = 0.9$ yields the optimal results. However, at this scale step, 12 bins are required to cover the scale range $S \in [0.3, 1]$ resulting in larger model databases. We observed a similar performance by using $\gamma = 0.85$ instead, which requires only 8 bins to cover the same scale range. Thus reducing the model database size as well the training time.

In addition to comparing the different variations of the model generation framework, we also investigate the influence of the out-of-plane rotation step angle as follow: $\Delta\Phi^x = \Delta\Phi^y = \{30^\circ, 60^\circ\}$ with $[\Phi_{min}^x, \Phi_{max}^x] = [\Phi_{min}^y, \Phi_{max}^y] = [-30^\circ, 30^\circ]$. Both these configurations are indexed as 1 and 2 respectively.

Since the performance of the cBRIEF and sBRIEF descriptors have been quite similar throughout the experiments, we only consider the cBRIEF descriptor. We evaluate each variation configuration against the five datasets presented in Sect. 8.1.1 for the HIP, cBRIEF and s+BRIEF descriptors. The evaluation results are presented in Fig. 8.13.

Overall, we can see that the best accuracy and precision values are obtained for variant A1 for all the descriptors. However, we also observe that for this case, it takes approximately 8 minutes on average to generate a model for the full range of in-plane rotations $[-180^\circ, 180^\circ]$ with over 40K features in the model (1.5 MB to 3 MB). Not too far behind, the second best results are achieved by variant C1 (orientation normalization with $\Delta\Phi^x = \Delta\Phi^y = 30^\circ$). With C1, we lose less than 5% in accuracy and precision to that of A1, however, we are able to generate the models in much less than a minute (more than 90% less than A1). On average, models generated using variant C1 requires less than 2500 features to achieve over 75% accuracy.

On another note, from the evaluation results in Fig. 8.13, it is evident that our BRIEF descriptors are able to outperform the HIP descriptor in both accuracy (over 5%) and precision (over 20%). Furthermore, the advantage of applying the feature stability verification process, during model generation and runtime matching, is demonstrated by the s+BRIEF descriptor.

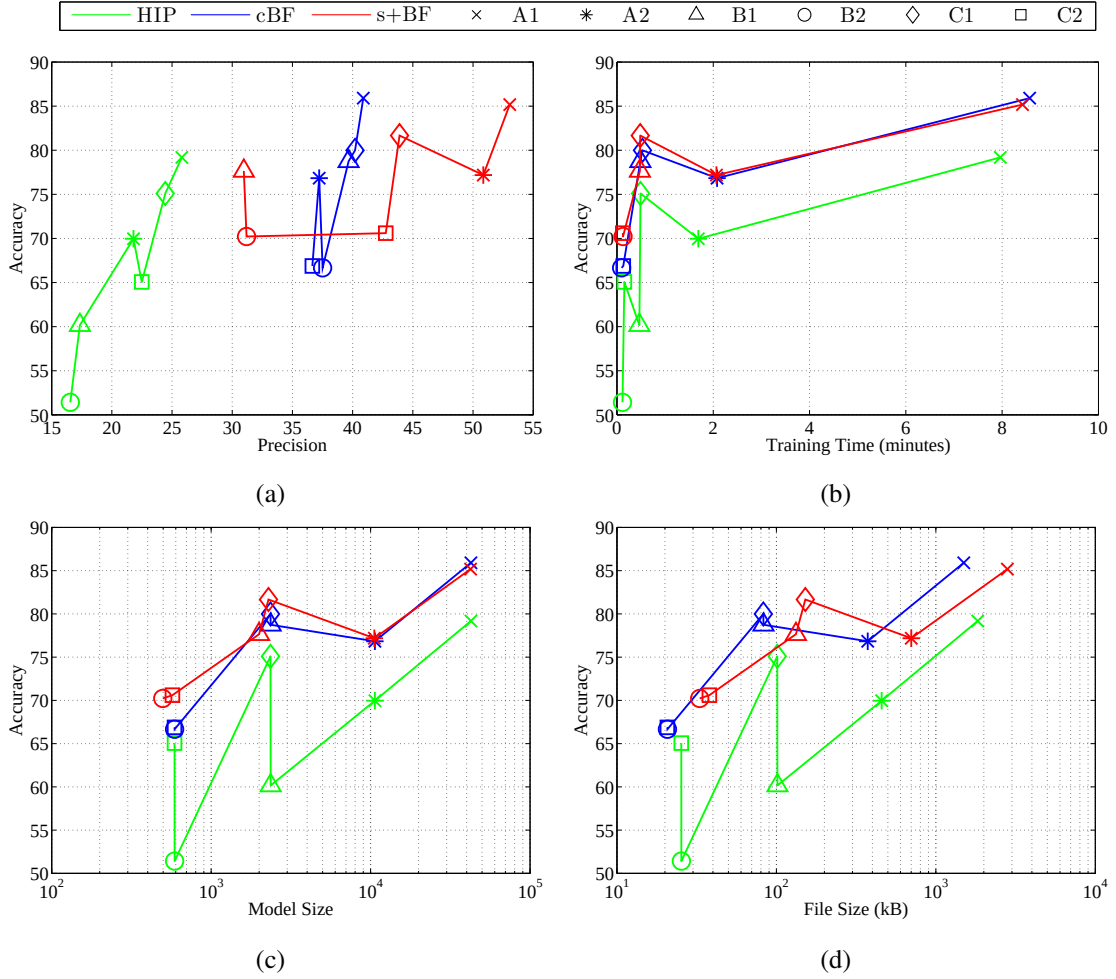


Figure 8.13: Evaluation results of the three variants of the model generation framework. A_i - no orientation estimation, B_i - orientation estimation without orientation clustering, C_i - orientation estimation and clustering, $i = 1 : \Delta\Phi^x = \Delta\Phi^y = 30^\circ$, and $i = 2 : \Delta\Phi^x = \Delta\Phi^y = 60^\circ$.

For the same configuration, for variants A and C, s+BRIEF boasts of a gain in precision of up to 15% greater than cBRIEF. A sample of the detection results on the five test datasets for the s+BRIEF descriptor is shown in Fig. 8.14.

We can see that for descriptors like HIP and s+BRIEF, which are strongly dependent on the patch appearances of the keypoints, orientation clustering assists in limiting the appearance variation resulting in improved performances. Although the s+BRIEF descriptor is constructed using the same approach as cBRIEF, we note that the final matching performance is dependent on the stability vector. The stability vector is in essence appearance dependent i.t.o. repeatabil-

ity of binary tests which influences the entropy. Thus, we can deduce that for descriptors (like BRIEF) which are less sensitive to appearance variation, orientation clustering is not necessary as the small deviation in estimated orientations are handled by the descriptor itself.

In summary, the orientation normalization stage proves to be an important component of the model generation framework. Not only does it produce significantly smaller model sizes, with fairly accurate and precise detection results, but also ensures that the models are generated in less than a minute. Note that this is an impressive contribution, as Taylor and Drummond [39] acknowledge that they require at least 20 minutes to generate a model using their training framework for their HIP descriptor.

8.4 Comparison to Other Descriptors

In this experiment, we evaluate our improved BRIEF descriptors against the original BRIEF [8] algorithm in addition to other well-known descriptor-based approaches, including SIFT [23], SURF [6], ORB [36], and FREAK [3] as implemented in the OpenCV library (version 2.4.3). We utilize the recommended default settings for each descriptor. However, to achieve scale-invariance for the ORB descriptor, we apply a pyramid of 8 levels and detected 2000 keypoints instead of the default 500.

SIFT and SURF are matched with the approximate nearest neighbour FLANN matcher, whereas a brute-force search is applied to BRIEF, ORB, and FREAK, with an emphasis on accuracy rather than speed. Matches that yield a distance larger than $3\times$ the best match are rejected.

The models for HIP, cBRIEF, and s+BRIEF are generated using the same parameter configuration as variant C1 in Sect. 8.3. The accuracy and precision results of the evaluation are shown in Fig. 8.15(a). It is evident that our BRIEF descriptors are able to achieve a similar performance to SIFT while significantly improving the accuracy (by 50%) and precision (by 25%) of the original BRIEF descriptor.

Figure 8.15(b) presents a break-down of the time (per frame in milliseconds) required to perform keypoint detection, feature extraction, feature matching and pose estimation for all the descriptors. Note that, for un-optimized implementations of our BRIEF descriptors, we are able to obtain a similar accuracy with regard to an optimized SIFT implementation at a tenth of the processing time. Interestingly, SIFT is more optimized than SURF in the current OpenCV implementation, hence it yields smaller computational times compared to SURF.



Figure 8.14: Sample detection results on the five test datasets for the s+BRIEF descriptor (trained with variant C1).

8.5 Evaluation on Outdoor Scenes

We now evaluate our modified BRIEF descriptors on medium-to-high resolution images of outdoor scenes from the freely available Oxford dataset¹. In particular, we make use of the *graffiti*

¹<http://www.robots.ox.ac.uk/~vgg/research/affine/>

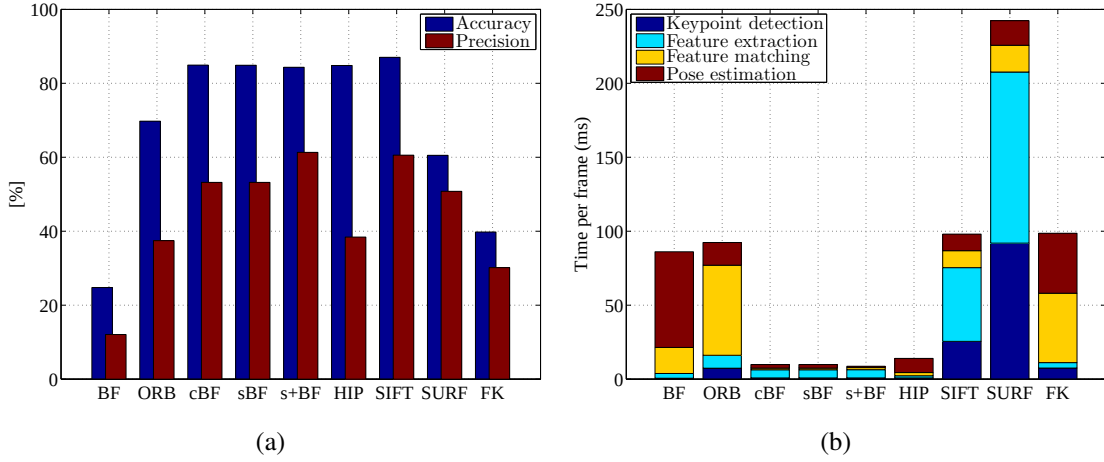


Figure 8.15: Comparison of our BRIEF descriptors to other approaches. (a) Accuracy and Precision, (b) Time per frame (milliseconds) for the different components during matching. BF and FK refer to BRIEF and FREAK respectively.

and *boat* datasets. The *graffiti* dataset contains viewpoint changes, whereas the *boat* dataset contains zoom and rotation changes. Both datasets possess scale and lighting variations.

The models for HIP, cBRIEF, and s+BRIEF are generated using the same parameter configuration as variant C1 in Sect. 8.3. However, for the *graffiti* dataset, we set $\Phi_x = \Phi_y = [-60^\circ, 60^\circ]$ with $\Delta\Phi^x = \Delta\Phi^y = 30^\circ$ in order to train for the viewpoint changes inherent in the test sequences.

Most descriptor-based approaches perform exceptionally well on targets which are deformed by changes in scale as well as in-plane rotation only. However, the true challenge is presented in scenarios in which a target is deformed by out-of-plane viewpoint changes of more than 40° . Hence, we first investigate the performance of our model generation framework, specifically our BRIEF descriptors, on the *graffiti* dataset - which includes viewpoint changes up to 50° .

For this evaluation, we compute the recall (number of correct matches / number of correspondences) vs 1-precision curve (number of false matches / number of matches) for all the descriptors. We compare our results to the following descriptor-based approaches: SIFT [23], ORB [36], and FREAK [3].

The evaluation results are presented in Fig. 8.16. For small viewpoint changes ($< 30^\circ$), the descriptor-based approaches tend to produce slightly better results. This is due to the fact our trained BRIEF descriptors have much more keypoints to match, hence, they yield a larger false positive rate. However, as can be seen for larger viewpoint changes, our BRIEF descriptors

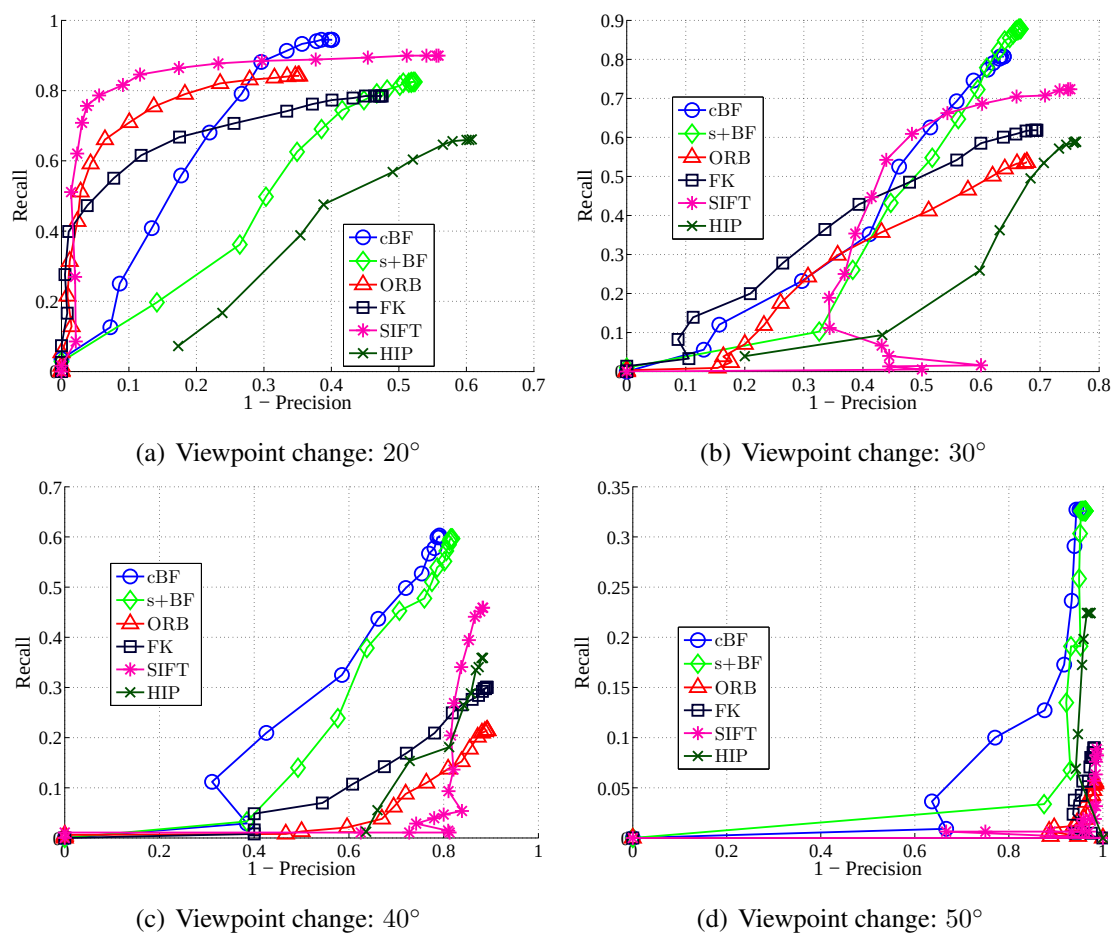


Figure 8.16: Evaluation results showing precision-recall curves (of all detection, extraction and matching stages jointly) for cBRIEF, s+BRIEF, ORB, FREAK, SIFT and HIP. Results are shown for various viewpoint changes of the graffiti dataset. Note that 1200 keypoints were extracted from each image sequence. BF and FK refer to BRIEF and FREAK respectively.

tend to provide the best results.

The visual matching results for cBRIEF on both datasets are illustrated in Fig. 8.17 and Fig. 8.18.

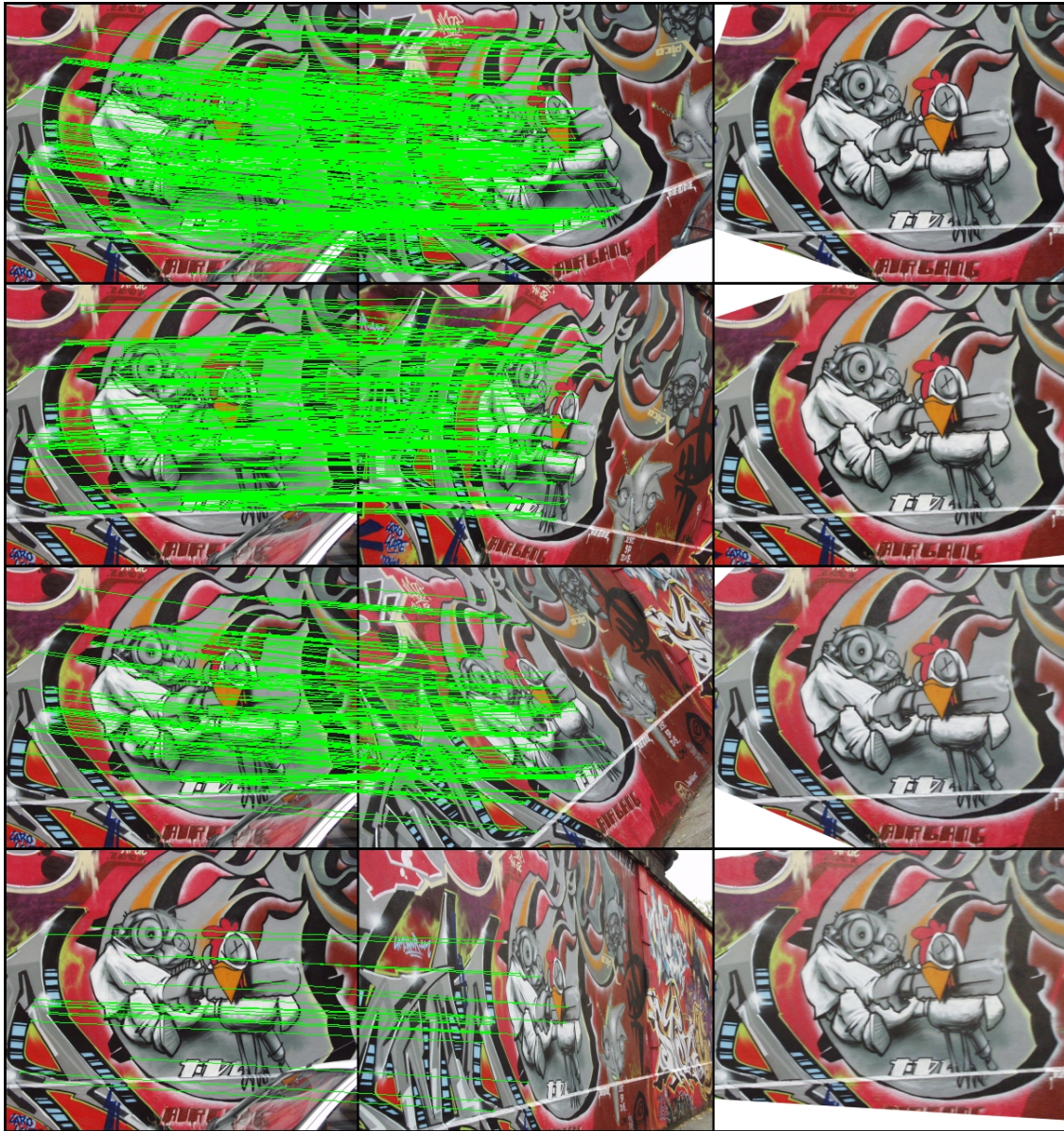


Figure 8.17: Matching results for cBRIEF on the graffiti dataset. The images in the second column are matched to those in the first. The last column is the resulting warp of the second image onto the first.

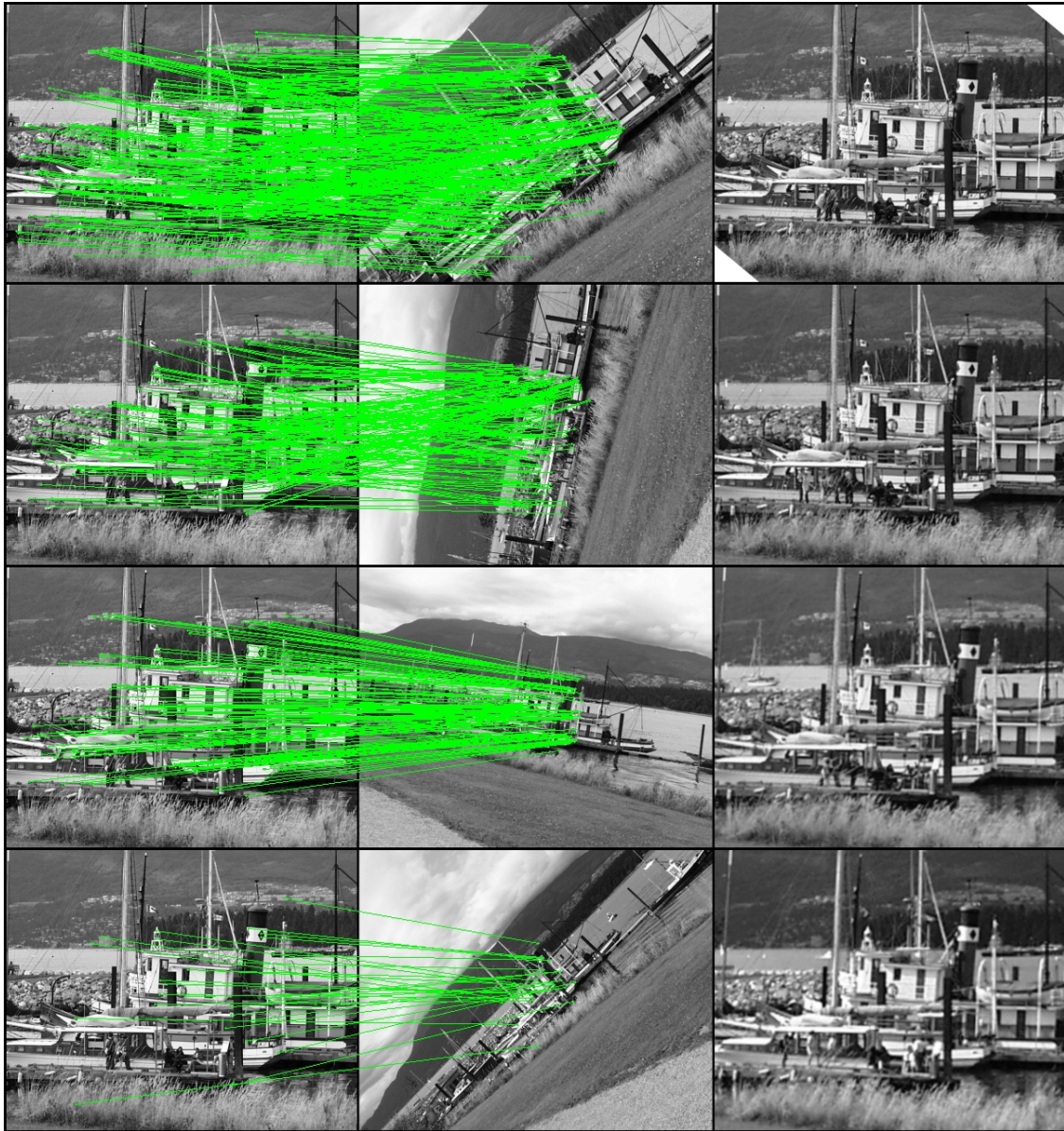


Figure 8.18: Matching results for cBRIEF on the boat dataset. The images in the second column are matched to those in the first. The last column is the resulting warp of the second image onto the first.

8.6 Conclusion

In this chapter, we have empirically verified the effectiveness and efficiency of our proposed model generation framework for detecting planar targets. In order to find the optimal parameter configuration for the model generation framework, we have conducted an extensive parameter tuning experiment. The corresponding optimal values for both orientation normalization and non-orientation normalization variants of the framework are summarized in Table 8.6.

Overall, we found that the local stability measure for identifying stable keypoints leads to significantly better performances as compared to the global stability measure. Furthermore, we were able to empirically demonstrate that orientation normalization is beneficial in the training framework as it greatly reduces the training time by a considerable factor. A major advantage of the orientation normalization stage is the reduction in the size of the model database. Since the features are rotation invariant, we can generate a model with much less features. Of the two orientation clustering algorithms developed, the Graph-based hierarchical clustering algorithm boasts better results over the Probabilistic clustering algorithm.

In the last part of the chapter, we have demonstrated results superior in speed and accuracy to state-of-the-art fast descriptors such as ORB [36], and FREAK [3] and stronger robustness to large viewpoint changes when compared to SIFT [23] and SURF [6].

Process	Symbol	Optimal Value / Range
Viewset generation	$[\Phi_{min}^x, \Phi_{max}^x]$ $[\Phi_{min}^y, \Phi_{max}^y]$ $[\Phi_{min}^z, \Phi_{max}^z]$ $\Delta\Phi_x, \Delta\Phi_y$ $\Delta\Phi_z$ $\frac{S_{min}}{S_{max}}$ γ M	$[-30^\circ, 30^\circ]$ $[-30^\circ, 30^\circ]$ $[-180^\circ, 180^\circ]$ 30° $20^\circ^{(1)}, 360^\circ^{(2)}$ $\frac{1}{3}$ 0.85 100
Stable keypoint identification	Q W	75 $[3, 4]$
Spatial clustering	$\delta_{\mathcal{K}}$	2
Orientation clustering <i>Graph-based HCA</i>	ϵ	$[15^\circ, 20^\circ]$
Feature stability verification	Ω_S	$\Omega_S = 128$

Table 8.6: Summary of the optimal configuration for the model generation framework parameters. Notes: ⁽¹⁾ no-orientation normalization, ⁽²⁾ with orientation normalization.

Chapter 9

Conclusion

In this thesis we have presented a novel learning-based approach, i.e. model generation framework, for localizing planar targets in real-time on power-constrained devices such as mobile phones. The framework comprises of three primary steps: viewset generation, stable keypoint identification, and feature extraction. The framework is generic and can readily be applied to an arbitrary feature descriptor in order to increase its robustness to both affine and perspective deformations. To complement the model generation framework, we have also developed an efficient runtime matching system which boasts of matching rates over 60Hz. We have successfully demonstrated the effectiveness of our system by realizing three extensions to the BRIEF descriptor [8], namely cBRIEF, sBRIEF and s+BRIEF, and have shown them to yield comparable robustness to state-of-the-art descriptors like SIFT [23] and SURF [6].

Our training framework is able to effectively generate models for targets in less than a minute. This is one of the most significant contributions of our research, as current methods require at least 20 minutes [39] to achieve respectable performances. Our reduction comes from: (i) the way we synthesize the viewset images, and (ii) applying orientation normalization with cheap orientation estimators. Instead of generating viewsets using the conventional affine transformation, we apply a perspective transformation which allows us to sample the deformation space in a more controlled way requiring less images per viewset. Orientation normalization allows us to obtain invariance to in-plane rotation deformations. In addition, it has the effect of reducing the number of viewpoint bins required which in turn decreases the overall model database size.

The amount of rotation invariance offered by orientation normalization is determined by the consistency of the orientation estimator: although an accurate estimator is desirable, it adds too much computational burden on the power-constrained device, thus preventing real-

time matching. On the other hand, cheap orientation estimators are generally sensitive to noise and pose a threat of negating the advantage offered by orientation normalized features. To solve this problem, we have applied orientation clustering to group similar orientation values for features which are in spatial proximity to one another. The objective of the orientation clustering process is to limit the appearance variation of the region encompassing the groups of features thus, allowing for better descriptors to be extracted. We have developed and compared two types of orientation clustering algorithms: a Graph-based hierarchical clustering algorithm and a Probabilistic clustering algorithm. We have found them both to be useful, however, of the two, the Graph-based hierarchical clustering algorithm yields better clusters which results in improved performances.

To ensure that we only add meaningful and reliable features to the model database, we have developed a novel method for verifying the stability of features in terms of their entropy. The feature stability verification component enables us to reject features which are too inconsistent or encode little information about the keypoint neighborhood. By storing the stability vector associated with each feature, we exploit the knowledge of which elements of the feature are stable for establishing reliable correspondences at runtime. This helps in increasing the inlier to outlier ratio by discarding false matches. However, this gain comes at the expense of a model database which is almost double in size (since the stability vector must be stored in addition to the feature vector).

On a final note, there are two factors which ultimately determine the system performance, namely the repeatability of the keypoint detector and the “descriptiveness” of the descriptor. Ideally, it is desired to have a fast keypoint detector, however, such a detector generally compromises speed for repeatability. If similar real-world points cannot be consistently detected, then even though we might have extracted features for those points in our model database, those features may never be matched. The descriptiveness of the descriptor is a significant contributory factor which directly influences the overall success of our proposed framework. Although our system is designed to increase the robustness of (relatively weak) descriptors to various deformations such as rotation, scale and perspective distortions, using an arbitrarily designed descriptor which does not uniquely describe the local neighborhood of the keypoint results in more false matches. Such a non-representative descriptor yields higher outlier ratios which results in more processing time for estimating the pose.

9.1 Future Work

One of the main constraints of our system is that it is currently restricted to localizing planar targets only. It would be worthwhile to extend the model generation framework to handle 3D objects with non-planar surfaces. A possible solution can be to generate a 3D model of the object, and then apply the framework on partially overlapping sections of the model. The sections are to be selected as such that they are locally planar. This allows us to create a rich database of features comprising of the entire object. This solution can be regarded as a starting point, however, more efficient solutions should be sought.

Our emphasis in this thesis has been primarily on the learning aspect. We have achieved fairly good performances. However, in the future, we would like to extend our work by investigating various approaches to further improve the matching rates. This would involve researching various indexing techniques including sparse distribution-based projections. Also worth investigating is the effect of computing index values from patches that are aligned to the local orientation of the keypoint. One would have to measure the orientation-alignment and interpolation costs involved in doing so. Alternatively, a lookup table of pre-computed rotated indexing patterns corresponding to discretized angle increments may be constructed to mitigate these additional runtime costs. However, this solution is memory demanding and may not be well suited for devices with limited memory depending on the size of the indices as well as the angle sampling rate.

Another limitation of the current system is that it is designed to detect a single target at a time. To detect multiple targets, the system must adopt a linear approach and search each target model database sequentially. This is obviously an unattractive solution as the number of models to be searched increases, the matching rate would decline accordingly. An interesting multi-target matching technique worth exploring in the future would comprise of both trees and hashing data structures.

Lastly, the training algorithm is required to run on a desktop machine. A desirable characteristic of the system would be to learn objects on the device itself. This has numerous advantages such as cutting down Internet usage costs (sending an object image / receiving a model database from the server). Also, the ability to learn targets on the device itself is useful in areas where Internet might not be available.

Bibliography

- [1] M. Agrawal, K. Konolige, and M. Blas. Censure: Center surround extremas for realtime feature detection and matching. In *European Conference on Computer Vision*, 2008.
- [2] Sharat Saurabh Akhoury and Robert Laganière. Training binary descriptors for improved robustness and efficiency in real-time matching. In *International Conference on Image Analysis and Processing (ICIAP)*, September 2013.
- [3] Alexandre Alahi, Raphal Ortiz, and Pierre Vandergheynst. FREAK: Fast Retina Keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. CVPR 2012 Open Source Award Winner.
- [4] Mitsuru Ambai and Yuichi Yoshida. Card: Compact and real-time descriptors. In *Proceedings of the 2011 International Conference on Computer Vision, ICCV '11*, pages 97–104, Washington, DC, USA, 2011. IEEE Computer Society.
- [5] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural Comput.*, 9(7):1545–1588, October 1997.
- [6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.
- [7] Matthew Brown, Richard Szeliski, and Simon Winder. Multi-image matching using multi-scale oriented patches. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, CVPR '05, pages 510–517, Washington, DC, USA, 2005. IEEE Computer Society.
- [8] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV, ECCV'10*, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

- [9] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.
- [10] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
- [11] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *Int. J. Comput. Vision*, 94(3):335–360, September 2011.
- [12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
- [13] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.
- [14] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.
- [15] Marko Heikkilä, Matti Pietikäinen, and Cordelia Schmid. Description of interest regions with local binary patterns. *Pattern Recogn.*, 42(3):425–436, March 2009.
- [16] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.
- [17] Yan Ke and Rahul Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'04, pages 506–513, Washington, DC, USA, 2004. IEEE Computer Society.
- [18] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(9):1465–1479, 2006.
- [19] V. Lepetit, J. Pilet, and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 244–250, 2004.

- [20] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision, ICCV '11*, pages 2548–2555, Washington, DC, USA, 2011. IEEE Computer Society.
- [21] Tony Lindeberg. Feature detection with automatic scale selection. *Int. J. Comput. Vision*, 30(2):79–116, November 1998.
- [22] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision - Volume 2, ICCV '99*, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.
- [23] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.
- [24] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases, VLDB '07*, pages 950–961. VLDB Endowment, 2007.
- [25] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *European Conference on Computer Vision (ECCV'10)*, September 2010.
- [26] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, October 2005.
- [27] Hans Moravec. Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. In *tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University*. 1980.
- [28] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM J. Img. Sci.*, 2(2):438–469, April 2009.
- [29] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, July 2002.
- [30] M. Özuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2007.

- [31] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.
- [32] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *Proceedings of the 10th European Conference on Computer Vision: Part II*, ECCV '08, pages 500–513, Berlin, Heidelberg, 2008. Springer-Verlag.
- [33] Paul L. Rosin. Measuring corner properties. *Comput. Vis. Image Underst.*, 73(2):291–307, February 1999.
- [34] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515 Vol. 2, 2005.
- [35] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Proceedings of the 9th European conference on Computer Vision - Volume Part I*, ECCV'06, pages 430–443, Berlin, Heidelberg, 2006. Springer-Verlag.
- [36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.
- [37] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *Int. J. Comput. Vision*, 37(2):151–172, June 2000.
- [38] Stephen M. Smith and J. Michael Brady. Susan a new approach to low level image processing. *Int. J. Comput. Vision*, 23(1):45–78, May 1997.
- [39] Simon Taylor and Tom Drummond. Multiple target localisation at over 100 fps. In *Proceedings of the British Machine Vision Conference*, pages 58.1–58.11. BMVA Press, 2009. doi:10.5244/C.23.58.
- [40] Simon Taylor, Edward Rosten, and Tom Drummond. Robust feature matching in 2.3 μ s. In *IEEE CVPR Workshop on Feature Detectors and Descriptors: The State Of The Art and Beyond*, June 2009.
- [41] Tomasz Trzcinski, Christos Marios Christoudias, Pascal Fua, and Vincent Lepetit. Boosting binary keypoint descriptors. June 2013.

- [42] Tomasz Trzcinski and Vincent Lepetit. Efficient discriminative projections for compact binary descriptors. In *Proceedings of the 12th European conference on Computer Vision - Volume Part I*, ECCV'12, pages 228–242, Berlin, Heidelberg, 2012. Springer-Verlag.
- [43] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, pages 125–134, 2008.
- [44] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proceedings of the third European conference on Computer Vision (Vol. II)*, ECCV '94, pages 151–158, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.