**Doctoral Thesis**

**Affine invariant regions++**

**Author(s):**
Ferrari, Vittorio

**Publication Date:**
2004

**Permanent Link:**
https://doi.org/10.3929/ethz-a-004846641

**Rights / License:**
In Copyright - Non-Commercial Use Permitted
Affine invariant regions++

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Technical Sciences

presented by
VITTORIO FERRARI
Dipl. Computer Science
born 8th September 1976
citizen of Lugano (TI)

accepted on the recommendation of
Prof. Dr. Luc Van Gool, examiner
Prof. Dr. David Lowe, co-examiner

2004
Abstract

Many problems in Computer Vision require the computation of correspondences between images. In order to cope with large differences in viewing conditions, several affine invariant region detectors have been developed in the last few years. These regions automatically adapt their shape so as to cover the same scene surface in any view. This dissertation builds upon existing detectors and develops various novel techniques which extend the power and functionalities of the regions.

The advances relate to different subfields of Vision, and can be summarized as four main contributions. First and foremost, the thesis presents a powerful Object Recognition system capable of working with large amounts of background clutter, severe occlusion, and strong viewpoint and scale changes. It can handle non-rigid deformations, and also finds the contours of the visible parts of the object.

The second innovation consists of a method to obtain region correspondences across several images taken from different viewpoints. These multi-view correspondences are important as they enable the automatic reconstruction of a 3D model given only a few still images. In contrast, traditionally this task requires a complete video as input.

Another branch of the thesis introduces a real-time algorithm which tracks the full affine shape of a region as it evolves through a video, and its application for markerless Augmented Reality. Most prior works instead rely on adding special markers to the scene.

Lastly, a technique to automatically find groups of regions correspondences lying on planar surfaces is presented. This allow to detect planar scene structures and their geometric transformation between views, which in turn can considerably simplify 3D reconstruction procedures, and is useful for robot navigation.
Zusammenfassung

Acknowledgements

Four years is a long period of time. During this period I have met a number of wonderful people who influenced my work in various ways.

First and foremost, I extend my deepest gratitude to Dr. Tinne Tuytelaars, who was always next to me, even thought a thousand kilometers divided our working places. Throughout the whole PhD, her brilliant intellectual support was second only to her amazing capacities to keep up a challenging and exciting working atmosphere, savour the successes, and react positively to defeats.

My heartfelt thanks go to my supervisor, Prof. Luc Van Gool, whose enormous drive was a true inspiration. His expert advice, extensive knowledge of the Computer Vision field, and continuous incentives to improve my work were invaluable.

I am grateful to my co-referee, Prof. David G. Lowe, who accepted to examine the thesis on a short notice, and reported useful suggestions. Also particular thanks go to Profs. Tomas Pajdla (Prague) and Andrew Zisserman (Oxford) for their encouragement and tokens of appreciation of the work.

This thesis certainly wouldn’t have been possible without solid technical facilities, for which I thank the very skilled system administrator of BIWI, Manuel Oetiker. While all my colleagues at BIWI brought a friendly atmosphere, three of them particularly deserve to be mentioned. Matthias Harders, for the countless table-soccer games, nights out in Zuerich and creative conversations. David Serby, who kept me aware of other ways of perceiving research life, proofread various papers, and was a worthy table-soccer opponent. I am indebted to Alexander Neubeck for several valuable discussions about algorithms, and for the cunning speedup to the topological filter algorithm of appendix D.

Finally, this thesis is, if indirectly, the fruit of my mother, who gave me passion, and my father, who taught me patience.
# Contents

Abstract .................................................. i

Zusammenfassung ........................................... ii

Acknowledgments ........................................... iii

List of Figures ........................................... ix

List of Tables ............................................ xiii

1 Introduction ........................................... 1
   1.1 Affine Invariant Regions ..... 1
   1.2 Contributions ...................... 1
   1.3 Organization of the thesis ........ 2

2 Image correspondence and affine invariant regions .... 3
   2.1 The correspondence problem .......... 3
   2.2 Correlation techniques .............. 6
   2.3 Affine invariant regions .......... 7
   2.4 Region extractors .................. 10
   2.5 Matching the regions .......... 13
   2.6 Object Recognition .................. 15

3 Tracking .............................................. 18
   3.1 Introduction ......................... 18
   3.2 Scheme of the tracker .............. 19
   3.3 Snake-based search ................. 21
      3.3.1 Finding the sides’ orientations . 22
      3.3.2 Finding the sides’ lengths .... 24
   3.4 Virtual corner search .............. 25
      3.4.1 Motivation and idea ......... 25
      3.4.2 The algorithm ............ 27
   3.5 2D Augmented Reality ............. 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.1</td>
<td>Geometric transformations</td>
<td>30</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Photometric transformations</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>3D Augmented Reality</td>
<td>32</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Input and notation</td>
<td>33</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Direct placement</td>
<td>34</td>
</tr>
<tr>
<td>3.6.3</td>
<td>User-selected placement</td>
<td>35</td>
</tr>
<tr>
<td>3.7</td>
<td>Parallel tracking*</td>
<td>36</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Motivation and idea</td>
<td>36</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Architecture</td>
<td>37</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Workflow</td>
<td>38</td>
</tr>
<tr>
<td>3.8</td>
<td>Results</td>
<td>40</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Tracking and augmented reality</td>
<td>40</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Parallel tracking*</td>
<td>50</td>
</tr>
<tr>
<td>3.9</td>
<td>Related work</td>
<td>54</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Tracking</td>
<td>54</td>
</tr>
<tr>
<td>3.9.2</td>
<td>Augmented Reality</td>
<td>56</td>
</tr>
<tr>
<td>3.10</td>
<td>Conclusion and outlook</td>
<td>57</td>
</tr>
<tr>
<td>4</td>
<td>Multiple-view correspondences</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>Scheme of the system</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>Building region-tracks</td>
<td>62</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Problem definition</td>
<td>62</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Conflict resolution</td>
<td>63</td>
</tr>
<tr>
<td>4.4</td>
<td>Refinement</td>
<td>66</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Motivation</td>
<td>66</td>
</tr>
<tr>
<td>4.4.2</td>
<td>The Refiner algorithm</td>
<td>66</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Global refinement</td>
<td>69</td>
</tr>
<tr>
<td>4.5</td>
<td>Propagation</td>
<td>69</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Two-view propagation</td>
<td>69</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Global propagation</td>
<td>71</td>
</tr>
<tr>
<td>4.6</td>
<td>Topological filter</td>
<td>72</td>
</tr>
<tr>
<td>4.6.1</td>
<td>The sidedness constraint</td>
<td>73</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Two-view mismatches filter</td>
<td>74</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Properties and advantages</td>
<td>75</td>
</tr>
<tr>
<td>4.6.4</td>
<td>Global mismatches filtering</td>
<td>77</td>
</tr>
<tr>
<td>4.7</td>
<td>Results</td>
<td>78</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Multiple-view correspondences</td>
<td>78</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Topological filter</td>
<td>82</td>
</tr>
<tr>
<td>4.8</td>
<td>Relocation*</td>
<td>85</td>
</tr>
<tr>
<td>4.8.1</td>
<td>The relocation algorithm</td>
<td>86</td>
</tr>
<tr>
<td>4.9</td>
<td>Related work</td>
<td>89</td>
</tr>
</tbody>
</table>
5 Object recognition and segmentation

5.1 Introduction ......................................................... 94
5.2 Soft matches .......................................................... 96
  5.2.1 Tentative matches ................................................. 97
  5.2.2 Refinement and re-thresholding ............................. 98
  5.2.3 Motivation ....................................................... 98
5.3 Early expansion ...................................................... 99
  5.3.1 Coverage of the model image ................................ 99
  5.3.2 Propagation attempt ........................................... 99
  5.3.3 Early expansion ............................................... 100
5.4 Early contraction .................................................. 101
  5.4.1 Surface contiguity filter .................................... 102
5.5 Main expansion ..................................................... 104
5.6 Main contraction ................................................... 105
5.7 Exploring the test image .......................................... 108
5.8 Results ............................................................... 110
  5.8.1 Object recognition ............................................ 110
  5.8.2 Video retrieval ................................................. 115
5.9 Sparsely textured objects* ........................................... 121
  5.9.1 Motivation and idea ........................................... 121
  5.9.2 Approach ....................................................... 124
  5.9.3 Examples ........................................................ 127
  5.9.4 Open issues .................................................... 129
5.10 Related work ....................................................... 131
5.11 Conclusion and outlook ........................................... 131

6 Geometric grouping

6.1 Introduction ........................................................ 134
  6.1.1 Two kinds of grouping ....................................... 134
  6.1.2 Outline of the chapter ........................................ 136
6.2 Coplanarity cues and score ....................................... 136
  6.2.1 Notation ........................................................ 137
  6.2.2 Motion cue ..................................................... 137
  6.2.3 Texture cue .................................................... 138
  6.2.4 Coplanarity score ............................................. 139
6.3 Clique Partitioning approximation ................................ 139
  6.3.1 Grouping as Clique Partitioning ........................... 139
  6.3.2 A fast approximation ........................................ 140
6.4 Texture segmentation*3 ............................................ 142
6.5 Results for coplanar grouping and texture segmentation ....... 145
List of Figures

2.1 Point correspondences ........................................ 4
2.2 Limitations of fixed windows .................................. 7
2.3 Effects of full affine distortion ............................... 9
2.4 Example of viewpoint invariant regions ....................... 11

3.1 Tracker's search space ........................................ 21
3.2 A snake iteration ............................................. 22
3.3 Initializing the polyline snake and the search for \( \mathbf{q} \) ........................................ 24
3.4 Low repeatability of the Harris corner detector .......... 26
3.5 Fitting a straight line segment ............................... 26
3.6 Approximately intersecting segments ....................... 28
3.7 Tracking based on virtual corners ............................ 29
3.8 Computing the texture placement ............................ 31
3.9 Mapping the texture to a video frame ........................ 31
3.10 A region-base and a virtual object .......................... 34
3.11 The tracked regions and the user-defined base .......... 35
3.12 The master-slave parallel tracker ........................... 39
3.13 Tracking and AR results for the Cinema sequence ....... 41
3.14 Tracking and AR results for the Desk sequence .......... 42
3.15 Tracking and AR results for the Electro sequence ....... 43
3.16 Matching two images of the Jennifer scene ............... 44
3.17 Tracking and AR results for the Jennifer sequence .. 45
3.18 Cinema sequence - massive tracking ....................... 47
3.19 Tracking results for the Spaceship sequence ............. 48
3.20 Regions tracked on the Smile sequence .................... 49
3.21 Augmented Smile sequence ................................ 51
3.22 Tracking and AR results for the Smile2 sequence ....... 52
3.23 AR results for the Internet sequence ........................ 53

4.1 Scheme of the multiple-view correspondences system .... 61
4.2 The conflict-resolution algorithm ............................ 64
4.3 Example 2-dimensional non-convex function to be maximized 67
LIST OF FIGURES

4.4 Example effects of the refiner algorithm ........................................... 68
4.5 Propagating a region ........................................................................... 71
4.6 Collective benefits of propagation on multiple incomplete tracks ....... 72
4.7 The sidedness constraint ...................................................................... 74
4.8 Region-tracks through 3 views of the Valbonne scene ....................... 79
4.9 Region-tracks through 3 views of the Birthday scene ......................... 80
4.10 Region-tracks through 4 views of Birthday ......................................... 81
4.11 Region-tracks through 2 views of Birthday ......................................... 81
4.12 Close-up on one region-track from Birthday ....................................... 82
4.13 Close-up on one region-track from a cereal-box ................................ 83
4.14 Test case for testing the robustness of the topological filter ............... 84
4.15 Performance of the topological filter .................................................. 84
4.16 Sensitivity of the topological filter to inaccurate localization ............. 86
4.17 Query point and constraining lines for relocation .............................. 87
4.18 Cutting the relocation polygon ......................................................... 90

5.1 Scheme of the recognition system ....................................................... 96
5.2 A case-study ....................................................................................... 97
5.3 Coverage of the model image and geometry of early expansion ........ 100
5.4 Early expansion on the case-study ...................................................... 102
5.5 Surface contiguity filter (early contraction) ........................................ 103
5.6 Geometry of main expansion .............................................................. 105
5.7 The sidedness constraint used in main contraction ............................. 106
5.8 Sidedness constraints hold also for deformed objects ......................... 107
5.9 Evolution of the number of correct matches for the case-study .......... 109
5.10 Evolution of the percentage of correct matches for the case-study ....... 109
5.11 Evolution of the set of matches .......................................................... 111
5.12 Evolution of the set of matches (continued) ....................................... 112
5.13 Distribution of recognition scores ...................................................... 115
5.14 Object recognition results I ............................................................... 116
5.15 Object recognition results II ............................................................... 117
5.16 Object recognition results III ............................................................. 118
5.17 Video retrieval results I ..................................................................... 122
5.18 Video retrieval results II ................................................................. 123
5.19 An example of sparsely textured object ............................................ 124
5.20 Propagating uniform regions ............................................................. 126
5.21 Detected uniform regions on the Maya book ..................................... 128
5.22 Propagated uniform regions of the Maya book ................................. 129
5.23 Propagated uniform regions of the Pokey object .............................. 130

6.1 Misalignment of line segments connecting two non-coplanar regions .... 139
6.2 Example run of the Clique Partitioning approximation algorithm ....... 142
List of Figures

6.3 The texture segmentation task ........................................ 143
6.4 Comparing color/texture histograms while allowing shifts ........ 144
6.5 Relationship between noise level and error ......................... 147
6.6 Coplanar grouping on the Boxes scene ............................... 148
6.7 False score percentage as a function of \( h_t \) ....................... 149
6.8 Coplanar grouping on the Maison scene .............................. 150
6.9 Coplanar grouping on the Valbonne scene ........................... 151
6.10 Coplanar grouping for the Suchard object ......................... 152
6.11 Segmentation of a texture collage .................................. 153
6.12 Segmentation of a composite texture ................................ 154
6.13 Synthesis of a composite texture ................................... 154
6.14 Landscape segmentation .............................................. 155
6.15 Affine dissimilarity .................................................... 156
6.16 Constructing GAMs on the Felix scene ............................. 158
6.17 Number of incorrect GAMs in function of their size .............. 162
6.18 Percentage of correct GAMs in function of their size ............ 163
6.19 A case where an incorrect match is within an otherwise correct GAM 164
6.20 A large GAM on the Graffiti scene .................................. 164
6.21 GAMs on the Birthday scene ......................................... 165
6.22 Largest GAM on two images of Leo .................................. 165
6.23 The two main GAMs of the Felix scene .............................. 166
6.24 Performance of the GAM-filter and RANSAC-EG on the Felix scene 167
6.25 The three main GAMs of the Foliage scene .......................... 167
6.26 Largest GAM on the Michelle magazine ............................ 168
6.27 Three GAMs on the teddybear ....................................... 169
6.28 Two GAMs on two very different views of Coleo ................. 170
7.1 The Coleo example object ............................................ 178
7.2 Dense matches between two Coleo views ............................ 179
7.3 Forming multi-view tracks ............................................ 180
7.4 Dense tracks through 3 views of Coleo .............................. 180
7.5 Transferring an incorrect GAM ...................................... 181
7.6 Transferring a correct GAM .......................................... 182
7.7 The GAM transfer mechanism ......................................... 184
7.8 Example multi-view integrations for Coleo ......................... 189
7.9 Region matching with a deformed object ............................ 190
7.10 Effects of additional model views on multi-view integration .... 191
7.11 An example integration for the Xmas object ...................... 192
7.12 Another example for Xmas .......................................... 192
7.13 Matching Coleo to a similar object ............................... 193

C.1 Horizontal pass of the fast intensity extrema detector ........... 204
List of Figures

C.2 Vertical pass of the fast intensity extrema detector  .................................. 205
D.1 Cyclic ordering for the fast topological filter .............................................. 208
D.2 Computing the sidedness relations in the first view .................................. 209
D.3 Sidedness relations in the second view ...................................................... 210
List of Tables

3.1 Computation times for the tracker .................................. 46
3.2 Performance of the parallel tracker ................................. 53

4.1 Number of tracks for Valbonne ................................. 79
4.2 Number of tracks for Birthday ................................. 82

6.1 Comparison of LP and our approximation to Clique Partitioning ... 145
6.2 Performance of our Clique Partitioning approximation algorithm ... 146

7.1 Quality of the Genetic Algorithm approximation .................. 195
1

Introduction

1.1 Affine Invariant Regions

Nowadays, an ever growing mass of images is distributed to the public through television, magazines, and the Internet. Additionally, people are producing increasing amounts of imagery with the new generations of high-quality, cheap personal cameras. Computer Vision can automate the access to images, by object recognition and image retrieval, or use them for attractive applications, such as mosaicing and 3D reconstruction.

The task of finding correspondences between images underlies many Computer Vision problems. In recent years, a new exciting research avenue has opened with the development of affine invariant regions. These are image features which automatically adapt their shape to the viewing conditions, so as to cover the same scene surface in any view of a scene. Thanks to this property, affine invariant regions enable to find correspondences between images taken from considerably different viewpoints.

A great share of the research efforts has focused on the reliable detection of the regions, and the invariant description of their image content. In contrast, this thesis develops several novel techniques that apply as subsequent processing steps, after regions have been found by an existing detector. These techniques extend the power and functionalities of the regions.

1.2 Contributions

The main contributions of the thesis are

- A powerful Object Recognition system which actively counters the problems related to the limited repeatability of the region detectors, and the difficulty of
matching, in the presence of large amounts of background clutter and particularly challenging viewing conditions. The system handles non-rigid deformations and produces an approximative segmentation of the object (chapter 5). A technique for exploiting the relationships among multiple model views for better recognition performance is also included (chapter 7 and the second half of chapter 6).

- A method to produce region correspondences across multiple unordered images, all taken from substantially different viewpoints (chapter 4). The method addresses the structural issues inherent to the task and the repeatability problem, and produces many accurate correspondences across several images (for example 10).

- A real-time tracker which recovers the complete affine transformation of a region in every frame of an image sequence. The tracker is complemented by a markerless, uncalibrated Augmented Reality approach, capable of overlaying planar and 3D virtual objects that deform reproducing the effects of viewpoint changes, therefore appearing as if they were part of the physical scene (chapter 3).

- An algorithm to partition a set of region correspondences between two views in groups lying on planar surfaces (first half of chapter 6). The algorithm does not make assumptions about the size or the number of planes in the scene, and works with perspective distortion, discontiguous planar surfaces and considerable amounts of measurement noise.

### 1.3 Organization of the thesis

The structure of the thesis reflects the heterogeneous nature of its main contributions. A brief introduction to the correspondence problem, its numerous applications, and the basic ideas behind affine invariant regions and object recognition is given in chapter 2. The other chapters cover the contributions, as indicated before.

Since every chapter is dedicated to a different problem, it directly includes the related results, a discussion of other research in the particular field, and a conclusion with some ideas for further research. Chapters are relatively self-contained, so that readers interested in only one technique should be able to understand it by considering a single chapter. The last section of every chapter include pointers to publications by the author related to the chapter’s contents. Most chapters have one section marked by a '*' symbol, which covers an additional interesting side-issue, not directly related with the goal of the chapter. Two of these special sections discuss research in early development stage (sections 4.8, 5.9). The acronyms used throughout the thesis are reported in full length in appendix F.
Image correspondence and affine invariant regions

The main issue motivating this thesis, the correspondence problem, is described in this chapter, along with the most relevant recent techniques dedicated to it. This chapter is intended as a review of existing work, preparing the non-specialist reader for the following chapters, and does not contain novel work by the authors. Many of the technical terms used later are introduced here in a natural manner.

2.1 The correspondence problem

Consider two images of the same scene, for example taken from two different viewpoints as in figure 2.1. A natural question is

which features correspond between the images?

The word 'correspond' here means 'are on the same scene spot'. Figure 2.1 shows some correspondences, with points as 'features'. Not every feature of an image need to have a correspondence in the other. Some can be out of the other field of view, or be occluded by other scene features. Sources of appearance variation, other than viewpoint changes, are zoom (change in focal-length), illumination changes and noise inherent to the imaging process. By 'features' researchers usually consider points, line segments or small compact areas, called 'regions'.

This two-view correspondence problem can be generalized by relaxing the assumption that both images show the same scene. Now we have the additional question

are these images of the same scene?

which formulates the basic recognition problem. The correspondence and recognition problems are closely related. Finding correspondences means scene features should
2.1. The correspondence problem

Figure 2.1: Two images depicting different views of the same scene. Points with the same number correspond.

somehow be recognized, and many correspondences might imply the recognition of the scene as a whole.

This general correspondence problem arises in various forms in diverse contexts, and is widely recognized as a fundamental problem of computer vision. The most common problems which are either forms of the correspondence problem, or entail it as a substep, are:

- **Stereo vision.** Given two images, find corresponding features. If the two images are taken from nearby viewpoints, we talk about small-baseline stereo [Zhang et al. 1995]. When the viewpoints are far apart, the problem is referred to as wide-baseline stereo [Pritchett and Zisserman 1998, Tuytelaars and Van-Gool 2000]. Usually complementary goals are to find the epipolar geometry relating the images, in the form of a fundamental matrix [Hartley and Zisserman 2002][p. 262] or, in the small-baseline case, to estimate the relative depth of image points [Fua 1993].

- **Object/scene recognition.** Given one or a few model images of an object or scene\(^1\), determine if it is present in a novel, previously unseen test image. This problem is particularly interesting when there are many test images, in the order of hundreds or thousands, and the object is present only in a small fraction. Object recognition then becomes *search* for a particular object in a database, which is an example of image retrieval [Smeulders et al. 2000]. A novel method for object recognition is proposed in chapters 5, 7. Note that finding feature correspondence between images is not the only way to approach

\(^{1}\text{The problems are equivalent, whether object or scene, so only 'object' will be used.}\)
2.1. The correspondence problem

recognition. Global methods, for example based on color or edge histograms, can achieve recognition without feature correspondence [Swain and Ballard 1991, Murase and Nayar 1995].

- **Video retrieval.** The recognition problem with an image sequence rather than isolated test images. The question is now *which frames of the video contain the object?* [Sivic and Zisserman 2003, Ferrari et al. 2004b]. Techniques can take advantage of the inherent temporal continuity of the video for better performance.

- **Tracking.** Determine the movement of an object through a given image sequence (video). Often following (tracking) the translation of the object is sufficient, like video-surveillance [Niu et al. 2003]. In other applications though, it is necessary to track also the change in size [Collins 2003], or a more complex transformation. An algorithm for tracking the complete appearance deformation of a planar region is presented in chapter 3.

- **3D reconstruction.** Given a video, or even just a few images of an object, reconstruct a 3D model of it. 3D reconstruction methods deduce the structure of the scene and the camera positions by factorizing matrices built from point/line correspondences. Most of the works starts from a video, in which case the correspondences are feature tracks, extending to many frames [Pollefeys et al. 1998, Oliensis and Genc 1999]. Reconstruction from few, widely separated images is quite new and tackled by only very few works [Strecha et al. 2003]. It requires features correspondences across several widely separated viewpoints [Schaffalitzky and Zisserman 2002b, Ferrari et al. 2003]. One method to find such multiview correspondences is proposed in this thesis, in chapter 4.

- **Mosaicing.** A few images of a scene are taken by changing the camera orientation, but keeping its location (projection center) fixed. In the typical example of mosaicing, the user is a tourist taking several pictures of a landscape from a fixed point, say the top of a mountain. The computer is asked to stitch all pictures together into a single, large one showing the whole landscape, by geometrically aligning all images [Brown and Lowe 2003]. This problem requires finding a few correspondences (at least 4) between pairs of images, allowing to compute a homography, which is the geometric transformation relating two images taken from the same point.

- **Robot navigation.** A robot needs to find its way through an environment, whose locations were learnt from images [Goedeme et al. 2004, Tuytelaars et al. 1999]. This is a particular instance of the scene recognition problem, with additional constraints due to knowledge of the robot or the environment (for example, the robot only moves on the horizontal plane [Goedeme et al. 2004]).
2.2 Correlation techniques

Classic techniques, popular until the end of the 90s, tried to find two-view correspondences following a common scheme. In a first stage, interest points were first automatically extracted from the images. Nearly always, these were corner-like structures, found by the Harris corner detector [Harris and Stephens 1988]. In the second stage, correspondences for corners of an image were only searched for among corners of the other image. For this purpose, every corner was characterized by a small window of pixels centered on it. The correspondence search was driven by the correlation between the greylevel patterns of pairs of windows (one in each image). Pairs of corners with highly correlated windows were matched. These techniques were applied to small-baseline conditions, so additional constraints were embedded in the matching process, such as the fact that corresponding corners should lie near the same locations in both images [Zhang et al. 1995]. Since this reduced the number of possible matches, it facilitated the task.

The strategy just described was a reasonable choice. Trying to match all image points at the same time would be computationally unaffordable, so some sample had to be chosen. Corners allow stable and accurate detection, because their neighborhood offer two-dimensional signal variation. Thus they are much better than points on homogeneous areas, or on straight edges. The latter's neighborhood exhibit variation only in the direction perpendicular to the edge, and therefore cannot be localized along it. Moreover, the use of fixed correlation windows was robust enough for small-baseline applications.

The problem gets harder when the two images are taken from considerably different viewpoints or camera orientations. The Harris corner detector is only invariant for points undergoing image translations and rotations, and fixed correlation windows can be used reliably only under image translation. Image rotation arises when the camera rotates about the optical axis (called in-plane rotation, or simply rotation). Already in this case, two windows cannot be directly compared, because the underlying pixels have rotated. Image translation affects windows located on planes parallel to the image plane, when the camera translates parallel to the image plane. When considering general camera translations (viewpoint changes) and rotations, a richer class of image transformation can be observed. Camera rotation about another axis (out-of-plane rotation) causes the image patch around a corner to be stretched along one direction. The same happens to patches on slanted planes during arbitrary camera translations. Moreover, translations along the optical axis, as well as variations of the focal-length, cause patches to change their size (scale change).

\(^2\)two viewpoints in stereo-head setup are related by this kind of motion.
2.3. Affine invariant regions

The desire to deal with wide-baseline conditions pushed researchers to develop new kinds of features. In order to achieve this, the size, shape and orientation of the windows ought to be somehow adapted to the current image scale, camera orientation and viewpoint, so that corresponding windows cover the same physical scene patch. Moreover, the corner detector should be upgraded to be invariant under scale and viewpoint changes as well.

Which properties should the new features (point and window) have? They should be kept small, so that occlusion of part of the scene will not compromise the ones

Figure 2.2: The image patch around the corner is rotated, stretched and scaled by the viewpoint change. However, the two windows do not react to the changes and are therefore misaligned. Fixed windows are not suited for wide-baseline conditions.

Even though the Harris corner detector has proven quite repeatable also for out-of-plane rotations [Schmid and Mohr 1998], it performs poorly with significant scale changes (more than factor 2). Another, even more important problem is the fragility of correlation windows with fixed shape, size and orientation. When the camera undergoes general rotation and/or translation, two such windows will have low correlation even if on corresponding corners, because they will not cover the same physical scene patch (figure 2.2). As a consequence, the classic ‘corner + fixed window’ strategy fails if the two images are taken in wide-baseline conditions. This term, although literally referring only to large viewpoint changes, is commonly used to indicate also changes in camera orientation, and sometimes scale changes due to focal-length variation. This thesis makes no exception, and will use ‘wide-baseline’ in this broad sense.

2.3 Affine invariant regions

The desire to deal with wide-baseline conditions pushed researchers to develop new kinds of features. In order to achieve this, the size, shape and orientation of the windows ought to be somehow adapted to the current image scale, camera orientation and viewpoint, so that corresponding windows cover the same physical scene patch. Moreover, the corner detector should be upgraded to be invariant under scale and viewpoint changes as well.

Which properties should the new features (point and window) have? They should be kept small, so that occlusion of part of the scene will not compromise the ones
in the visible part. Such features are only characterized by a small and compact neighborhood of pixels, so they are called **local features**. This underlines the contrast to **global features**, such as the color histogram, which collect information from the whole image. In wide-baseline conditions it is usual for a camera to see parts of the scene missing in the other. In a similar way, in object recognition, the test images might contain varying backgrounds, which do not appear in the model images. These image parts are called **clutter**, in object recognition. This situation does not cause particular problems to local features, because it only causes spurious features, with no possible correspondence, without affecting the ones on the parts visible in both images.

What can be done to make local features automatically adapt their window to the viewpoint? The crucial observation is that most small image patches cover an approximately **planar** surface of the scene. Since a patch is small with respect to the distance from the optical center, perspective effects can be neglected. In this condition, two images $P_1, P_2$ of a small, planar patch are geometrically related by an **affine transformation**:

$$
\begin{pmatrix}
  x_2 \\
  y_2
\end{pmatrix} = 
\begin{pmatrix}
  a & b \\
  c & d
\end{pmatrix} 
\begin{pmatrix}
  x_1 \\
  y_1
\end{pmatrix} + 
\begin{pmatrix}
  e \\
  f
\end{pmatrix}
$$

where $(x_1, y_1)^T$ is a point of the patch in the first image, and $(x_2, y_2)^T$ its corresponding point in the second image. The 6 parameters $a$ to $f$ completely determine the transformation. If, without loss of generality, we assume that the coordinate frame of the first image is centered on the center of $P_1$, then the parameters have a simple geometric interpretation:

$$
\begin{pmatrix}
  x_2 \\
  y_2
\end{pmatrix} = 
\begin{pmatrix}
  s_x \cos(\theta) + k s_x \sin(\theta) & -s_x \sin(\theta) + k s_x \cos(\theta) \\
  s_y \sin(\theta) & s_y \cos(\theta)
\end{pmatrix} 
\begin{pmatrix}
  x_1 \\
  y_1
\end{pmatrix} + 
\begin{pmatrix}
  t_x \\
  t_y
\end{pmatrix}
$$

the transformation is composed of rotation $\theta$, shear $k$, two translation components $t_x, t_y$, and two scaling components $s_x, s_y$, and has thus 6 degrees of freedom. The two scaling components can be reformulated as absolute ‘scale’ $\sqrt{s_x \cdot s_y}$ and ‘stretch’ $\frac{s_x}{s_y}$ components, rather than two different scaling components, one in the $x$ direction and the other in the $y$ direction. The terminologies are equivalent, and both express the fact that the extent can change either in two directions (‘scale’, like when zooming) or in only one (‘stretch’, occurring for example during out-of-plane rotation).

Figure 2.3 illustrates the effects of all affine components on two synthetic images of a cube. The facet marked with the ‘+’ symbol appears in the right image bigger (scaled) and rotated with respect to the left image, like if the box was turned and the camera zoomed. Moreover, the viewing orientation undergoes out-of-plane rotation, which causes the apparent compression (stretch) of the facet. The last affine component, shear, affects the angle between the sides of the facet, which is no longer
2.3. Affine invariant regions

Figure 2.3: Effects of full affine distortions.

90 degrees in the right image. Shear, like stretching, is induced by out-of-plane rotation.

The above considerations reveal the nature of the desired features: small image patches that can be extracted in an affine invariant manner. Hence, they automatically adapt their affine shape so as to keep on covering the same physical surface patch in any image. These features are called affine invariant regions throughout the rest of the thesis. By automatically it is understood that the region extraction process runs on each image independently, without knowledge of any other image. The shape of a region is determined solely from the local information in its own image.

Clearly, automatic shape adaptation is quite a trick, and therefore not all image locations are suited to breed an affine invariant region. As best formulated by Matas [Matas et al. 2002a], those are distinguished regions, which possess some stable and invariant property which enables their reliable detection over different images. Much like corners are special points because they offer signal variation in two directions, these regions are special because they have some property allowing their viewpoint invariant detection. Several region extractors have been developed in the last few years, differing in the property they look for, and therefore in which image patches they detect. The next section briefly reports the main ideas behind the best known extractors.

Notice that extracting good regions is not enough: we need a way to match them between images. This issue is discussed in section 2.5.

\(^3\)In this thesis, the verb ‘extract’ is preferred over ‘detect’, used by other authors. However, their meaning are considered identical.

\(^4\)The ‘shape, orientation and size’ concept is simply referred to as ‘shape’ in this context, since an affine transformation entails scale change and rotation as well.
2.4 Region extractors

Although different viewpoint invariant extractors employ different algorithms and find different regions, they all share the general ideas and have been designed for the same reasons discussed in the previous section. Since the extraction algorithms are relatively unimportant for the proper understanding of the ideas presented in this thesis, they will only be briefly summarized here. All new techniques described in the next chapters, with the exception of tracking, can work on top of any affine invariant region extractor.

Tuytelaars and Van Gool [Tuytelaars et al. 1999, Tuytelaars and Van-Gool 2000] have proposed two types of regions: parallelogram-shaped and elliptical, both constructed around anchor points. The former are based on two straight edges intersecting in the proximity of a corner (anchor point, extracted by the Harris detector). This fixes a corner of the parallelogram (coined c) and the orientation of its sides. The opposite corner (coined q) is fixed as the extrema of a 2D affinely invariant function of the region's texture (figure 2.4). This automatically gives a region with adequately textured content for matching purposes. One of the contributions of this thesis is a method to track these parallelogram-shaped regions through an image sequence (chapter 3). For the extraction of elliptical regions, intensity extrema of the greylevel image are first computed and serve as anchor points. Next, the system tries to extract a surrounding blob whose contour is formed by extrema of the intensity profile along rays emanating from the anchor point. An ellipse with the same first and second order shape moments is finally fit to the blob (figure 2.4). Elliptical regions do not need prior edge detection, and cover blob-shaped areas, with uniform or gradually shaded content. Because of the need for distinctive texture for reliable matching, the final region is enlarged by a scale factor 2 (this operation preserves affine invariance).

In practice both extraction methods work well for viewpoint changes and have fine-grained sensibility to moderate scale changes. However they do not handle large scale changes (more than factor 2), under which we could observe considerable performance degradation. Hence, we extended them with a simple multi-scale approach, which revealed decisive to achieve robustness to larger scale changes We generate a Gaussian pyramid scale-space and run the extractor independently in each level. All regions extracted at all levels are returned.

5Jiri Matas pointed out that this class of algorithms should be called ‘viewpoint covariant’, rather than ‘invariant’, because the shape of the regions ‘changes with’ the image transformation and does not ‘stay constant’ in spite of it. Although we accept the terminology as more mathematically correct, this thesis uses the expression ‘viewpoint invariant’ because it is more widespread and better conveys the important idea of stability under viewpoint changes.

6As a secondary contribution of this thesis, we have developed a very time-efficient algorithm for the detection of intensity extrema, which is described in appendix C
2.4. Region extractors

Baumberg [Baumberg 2000] describes a region extractor which follows two steps. First, the location and scale of the regions are determined by a multi-scale Harris detector. Then, an adaptive procedure based on the second moment gradient matrix recovers stretch and shear.

The extractor of Mikolajczyk and Schmid [Mikolajczyk and Schmid 2002] follows a similar scheme. However, only initial locations and scales are obtained by a multi-scale Harris detector. During the second step, an iterative procedure modifies the initial location and scale, as well as recovering stretch and shear. This algorithm can work with larger scale changes than [Baumberg 2000].

Matas et al [Matas et al. 2002a] designed maximally stable extremal regions (MSER): connected components of pixels which are all brighter or all darker than any pixel on the region’s contour. The very time-efficient extraction algorithm considers all possible global thresholding of the greylevel image, and retains regions which change only little across a wide range of thresholds ('maximally stable' regions). The MSER extractor achieves scale invariance \(^7\) in a natural way without searching, or even building, a scale-space. Besides being faster, in our opinion this is also more elegant. To ease further manipulations, an ellipse is fit to the detected connected component of pixels, akin to what done by Tuytelaars and Van Gool. In an effort to maximize distinctiveness while still handling occlusions and keep the regions approximately

\(^7\)In the strict sense of scale-space theory, the procedure is not scale invariant, because there is no smoothing involved at any scale. However, it is in the intuitive sense of 'repeatably finding the same region in spite of scale changes'.

Figure 2.4: Example of viewpoint invariant regions extracted with the method of Tuytelaars and Van Gool. The elliptical region is near the spot of figure 2.2. It covers the same physical surface because the shape adapted to the viewing conditions (note the stretching). The effects of image rotation are particularly visible on the parallelogram-shaped region.
2.4. Region extractors

planar, each region is produced in 3 differently sized versions (scaled 1x, 2x and 4x respectively). The matching algorithm then attempts to use the largest ones first. As Lowe [Lowe 2004] pointed out, none of the aforementioned methods is truly affine invariant, as they all start from some location and scale selected in a non-affine-invariant manner. The Matas approach instead, directly extracts the regions, and does so in a full affine invariant manner. While Lowe is right in principle in practice all region types have proven to work well under general affine transformations. The reasons are that the crucial part of the extractor algorithm, the selection of the region’s shape, is well designed and quite accurate in all methods.

Both the Baumberg and the Mikolajczyk approaches are extensions of the Harris corner detector to affine invariance. Hence they are located on corner-like, very textured areas, as opposed to the Tuytelaars’ elliptical and Matas’ MSER regions, which cover blob-like areas. The blob-like and corner-like regions spatially complement each other and a system using both is expected to perform better. Some evidence for the benefits of using multiple region types is already available from various works, such as [Lazebnik et al. 2003] in texture segmentation, [Schaffalitzky and Zisserman 2002b] in multiple-view matching, and [Schaffalitzky and Zisserman 2002a, Sivic and Zisserman 2003] in video analysis.

Lowe [Lowe 1999] invented a detector coined Scale-Invariant Feature Transform (SIFT). In SIFT, a feature’s location and scale are determined by local extrema, over location and scale at the same time, of a Difference-of-Gaussian scale space. Once a feature is detected, its orientation is obtained by the dominant gradient orientation within a circular window. A SIFT feature is only invariant to image similarity transformations, so it is in principle less immune against viewpoint changes than affine invariant features. However, Lowe also proposed a powerful method to describe the content of a SIFT region. The region is partitioned into an array of 4x4 blocks, and an histogram of the gradient orientations, weighted by the gradient magnitudes, is computed for each block. The descriptor is the concatenation of all histograms, and is robust against significant misalignments that might be caused by affine distortions and non-planar regions. As a consequence, the SIFT features can be matched also across viewpoint changes, as demonstrated by Lowe in his most recent paper [Lowe 2004]. The major advantage of SIFT features is their extreme computational efficiency (in the order of one or two seconds for extracting and matching SIFT features between two images).

As a final note, we mention a different approach to wide-baseline matching due to Tell and Carlsson [Tell and Carlsson 2000]. A feature is a Harris corner point, characterized by the intensity profiles on the line segments connecting it to other corners. Matching then proceeds by comparing profiles instead of local regions. The method is very interesting in that it avoids the problem of constructing affine invariant regions. However, the features are no longer existing independently, they are strongly related to each other, making the approach less local. Since it does not
produce a description of the local shape variation, it does not belong to the class of methods which is the subject of this thesis. The techniques proposed in the next chapters are unfortunately incompatible with it.

2.5 Matching the regions

Once invariant regions have been extracted in the images with one of the above methods, the next step is to put them in correspondences between images. This matching process follows a scheme common to most approaches. It is composed of three stages, briefly described next.

Compute descriptors

The idea is to represent the textured content of each region with a relatively small number of measurements (typically less than 100), which concisely describe the region and can be used for comparison when looking for correspondences. We refer to the set of measurements simply as ‘the descriptor’. The descriptor should be invariant to affine geometric transformations, and also to photometric changes. The latter are due to mainly two factors: the variation in the local surface orientation with respect to the light source, and brightness change of the light source itself. Moreover, a good descriptor is distinctive: it preserves the informative characteristics of a region, which allow to distinguish it from another.

The first step towards computing a descriptor is geometric normalization. A region, which in most cases is elliptical, is first transformed into the unit circle. In this way, the affine transformation is removed: corresponding regions in different images are now aligned. To be precise, the regions might still be related by an arbitrary rotation. The most common solution is to use descriptors which are rotation invariant as well. Researches use different descriptors, like combinations of color-based moments [Tuytelaars and Van-Gool 2000], banks of orthogonal complex linear filters [Schaffalitzky and Zisserman 2002b], normalized Gaussian derivatives steered in the gradient direction [Mikolajczyk and Schmid 2002], or spin-images [Lazebnik et al. 2003]. Another approach is to rotate the region towards the direction of the dominant gradient, in a similar way to what done by Lowe (see previous section). In this fashion, non-rotation-invariant descriptors can be used, like the SIFT-descriptor, or even just sampling pixels (recall that the geometric transformation is now completely canceled). A recent work [Mikolajczyk and Schmid 2003] has evaluated various descriptors by applying them to different region types. The best result, regardless of the kind of region, was achieved by computing the SIFT-descriptor after canceling rotation as just mentioned.

Find initial matches
Armed with the region’s descriptors, the system can search for correspondences. Two regions are matched if their descriptors are very similar. The Mahalanobis distance is often used as a similarity measure, because it accounts for the different variances of the descriptor’s components, and their mutual covariances. Notice that comparing descriptors, rather than directly regions, is not only faster, but also potentially more robust, as descriptors often alleviate the effects of noise and misalignments.

In order to avoid the computational expense of computing the similarity for all pairs of descriptors, an indexing mechanism can be employed [Lowe 2004, Schaffalitzky and Zisserman 2002b, Mikolajczyk and Schmid 2002]. Indexing sees descriptors as points in a vector space. It organizes them in a data-structure capable of efficiently finding nearest neighbors, or all points within a certain distance from another. Commonly used indexing data-structures are the binary space partition tree [Schaffalitzky and Zisserman 2002a], and variations of the k-d tree [Lowe 1999, Schmid 1999]. Efficient indexing is particularly useful in problems which involve many images, such as object recognition with a large database of models, and image/video retrieval.

After finding a first set of matches based on the descriptors, several authors apply a simple verification stage which directly compares the pattern of pixels in the matched regions. Typically, only matches with cross-correlation above a certain threshold are kept [Mikolajczyk and Schmid 2002, Tuytelaars and Van-Gool 2000, Schaffalitzky and Zisserman 2002b, Matas et al. 2002a]. This stage removes some incorrectly matched regions which the descriptor could not distinguish because of its lower dimensionality.

**Filter mismatches**

A very common final stage is to filter mismatches (incorrect correspondences) by simultaneously computing the Fundamental Matrix $F$ via the Random-Sampled-Consensus (RANSAC) algorithm [Torr and Murray 1997]. The idea is to find mismatches as correspondences not respecting the epipolar geometry between the images, which is encoded by $F$. Since $F$ is not known yet, RANSAC attempts to estimate it, by looking for large subsets of matches agreeing on a single epipolar geometry.

The algorithm is iterative and can be summarized as follows. At each iteration, 7 matches are randomly sampled and used to estimate a fundamental matrix $F_t$. The number of inliers to $F_t$ are computed among all matches. A match is considered an inlier if the center of a region in an image is close to its associated epipolar line, which is computed by applying $F_t$ to the center in the other image. If the number of inliers found is above a threshold, the algorithm terminates, otherwise a new iteration is started. After a limit of $N$ iterations the algorithm stops and returns the best result so far. The returned inliers are considered as correct matches, while the other
matches as incorrect. This algorithm is referred to as RANSAC-EG throughout the thesis.

RANSAC-EG can succeed only if a sample composed of all correct matches is drawn before N trials. However, probabilistic analysis shows that the number of trials necessary to have a good chance of drawing such an uncontaminated sample is proportional to \(1/r^7\), where \(r\) is the percentage of correct matches in the set as discussed by [Chum et al. 2003]. Hence, when \(r\) is low, the algorithm cannot be used, because it would require a huge number of iterations (for example: 0.5 Million for \(r = 15\%\) correct matches). Moreover, in practical situations, even finding an uncontaminated sample does not lead to the discovery of all correct matches: the inexact localization of the regions, and the minimal sample size, may cause a poor \(F_t\), incapable of agreeing with many of the correct matches [Chum et al. 2003]. This phenomena further restricts the range of cases for which the algorithm can be used.

In the existing literature, RANSAC is typically used in situations with less than 50% mismatches. Several authors noticed that it fails when the ratio of mismatches is much above this figure [Lowe 2004, Chum et al. 2003, Chen and Meer 2003]. As an indicative, rough approximation we may say that the limits of usability are in the presence of up to 2/3 mismatches. In typical matching problems, this is acceptable, because the regions are sufficiently distinctive to yield only a small fraction of mismatches. However, in particularly difficult cases, there might be a far higher portion of mismatches, which calls for alternative strategies. Also notice that RANSAC-EG enforces rigid-motion through the epipolar geometry, so it is not suited for object recognition systems aiming to deal with deformable objects.

2.6 Object Recognition

Although it tackles several problems, this thesis puts a special weight on Object Recognition (OR). Therefore, we feel it necessary to contextualize our approach within the wide universe of OR. In this section, a very brief classification of OR methods is given. It is by no means complete, and only aims at orienting the reader and providing references to a few particularly significant landmark works in this fascinating and most researched problem.

Researchers divide OR methods between model-based and appearance-based. In the model-based paradigm [Shimshoni and Ponce 1995, Bebis et al. 1995, Lowe 1991, Beis and Lowe 1999], the object is represented by a 3D model of its shape. Recognition of a test image amounts to evaluating if it could be a projection of the model. This evaluation tries to find the object pose which would generate the features observed in the image (typically line segments). Model-based approaches have fallen out of favor mostly because it is impractical to provide the 3D model to the system,
especially for complex-shaped objects. Moreover, it is inherently difficult to interpret geometric features of the test image as projections of a 3D model, in particular when the ambition is to distinguish among many objects.

The philosophy of appearance-based methods differs sharply from the model-based one, and can be summarized as 'just images'. The idea is to model objects from one or more images, without resorting to any 3D model. There are two main kinds of appearance-based approaches: global and local. Global methods build an object representation by integrating information over the entire image. Famous examples are the color histogram, used in the seminal work of [Swain and Ballard 1991], or representations of contours, like the more recent 'shape-contexts' [Belongie et al. 2002] and 'shock-graphs' [Sebastian et al. 2001]. A dissimilarity measure with the histogram (or other representation) computed in the test image provides the recognition criteria. Another, equally important global approach are 'eigenimages' [Murase and Nayar 1995]. A number of full images, taken from many different viewpoints, are injected into a PCA, so as to obtain a small number of 'eigenimages' capturing most of the appearance variation. The object's multiview appearance becomes a low-dimensional manifold into the 'eigenspace' spanned by the eigenimages. Recognition occurs by projecting the test image into a point in the eigenspace and finding the closest manifold.

Unfortunately, any global representation is very sensitive to background clutter in the test image and to partial occlusion of the object. Therefore, global methods only consider test images without background, or necessitate a prior segmentation, a task which has proven extremely difficult, if not impossible. As additional limitation, robustness to large viewpoint changes between the model and test image is hard to achieve, because the global object appearance varies in a complex and unpredictable way (the object’s geometry is unknown). Local methods counter the problems caused by clutter and occlusion by decomposing an image in a collection of relatively small elements, called local features. Since these features are detected based only on local information, occlusion of part of the object does not affect features in the visible part. Moreover, clutter in the test image only result in additional spurious features, without compromising the ones on the object. During recognition, features are detected in the test image, and put in correspondence with the ones from the model image(s). Usually, an additional stage verifies the coherence of the spatial arrangements of matched features, trying to reject incorrect matches and to compute a global score to be used as recognition criteria (typically the number of verified correspondences). Several kinds of local features have been proposed, including edge segments [Pope and Lowe 1996], local contour groups [Selinger and Nelson 1999], and very small multicolored neighborhoods [Matas et al. 2000]. Particularly important for this thesis, is the line of thought started by the influential work of Schmid [Schmid 1996]. A local feature there is an 'interest points' (Harris corners) augmented by a description of its pixel neighborhood. Both the point
2.6. Object Recognition

detector and the descriptor are invariant under rotation. Hence, features are reliably extracted in rotated versions of the model image and effectively matched by directly comparing the descriptors. Within this successful paradigm, there has been a lot of evolution. Feature detectors have appeared [Mikolajczyk and Schmid 2001, Lowe 1999] which are invariant also under scale changes, and they automatically adapt the size of the neighborhood where the descriptor is computed. In this fashion, reliable matching, and hence recognition, can be performed under image similarity transformations. Finally, recognition under general viewpoint changes was recently achieved by several authors [Matas et al. 2002a, Tuylelaars and Van-Gool 2000, Mikolajczyk and Schmid 2002, Baumberg 2000], with detectors adapting the complete affine shape of the neighborhood to the viewing conditions. This last step is particularly significant: even though the global appearance variation of 3D objects is very complex under viewpoint changes, it can be approximated by a simple affine transformation at the local scale. This because the neighborhood (which we call region) is small and hence can be considered approximately planar. A short presentation of the various region detectors was given in section 2.4.
3

Tracking

3.1 Introduction

Small, planar, parallelogram-shaped regions have been proposed by [Tuytelaars et al. 1999] for wide-baseline stereo matching (their definition can be found in section 2.4). A time-efficient approach for tracking this kind of regions through an image sequence, and its application to Augmented Reality are described in this chapter.

The tracker combines two important properties. First, it is computationally efficient, and can track a region in real-time. Second, it tracks the complete affine transformation, therefore putting the region in *full correspondence* across the frames. As output, three accurate point correspondences per tracked region are produced, even in the presence of out-of-plane rotation or scale changes.

Both edges and texture information are exploited in an integrated way, yet the complete region’s contour is not required. The method can withstand fast and irregular motion and can recover from a temporary loss of the region. These properties are based on a particular way of searching the affine transformation space which exploits the nature of the kind of regions considered.

The tracker’s characteristics are very useful in many applications. One of these is Augmented Reality, with its intrinsic real-time requirements and where full affine tracking allows to capture the effects of camera motion. In this chapter we explore how to augment the image sequence with planar and 3D virtual objects. In the planar case, a virtual texture is superimposed on the scene, by rigidly attaching it to a tracked region. The texture moves and changes shape exactly like the region, therefore it deforms reproducing the effects of viewpoint changes, as if it were part of the physical scene. This is possible because the tracker recovers the complete affine transformation of the region between frames. Besides virtual textures, full 3D objects can be inserted in the scene as well, at the mere price of tracking a second region.
In addition to the advantages directly inherited from the tracker, the proposed AR system is also *markerless and uncalibrated*. By markerless we mean that no artificial landmarks or markers are required. Virtual textures/objects can directly be attached to natural scene patches. The method is uncalibrated in the sense that no information about the camera, its motion, or the 3D structure of the scene, is given as input or computed. This is perhaps the most fascinating aspect: virtual textures/objects can be drawn in the scene so as to mimic the viewpoint changes, without knowing anything about the geometry of the scene or the camera viewpoints.

The regions to be tracked and augmented do not necessarily have to be extracted in a viewpoint invariant manner by [Tuytelaars et al. 1999]. The method is able to track any parallelogram-shaped region with two concurrent sides on (short) straight edges. However, attaching a virtual texture to an automatically extracted region enables to combine scene recognition with Augmented Reality. In this scenario, the user would first select a region in a single image, which is also used to model the scene. Later on, a video of the scene is acquired, for example while the user walks through the scene with a head-mounted camera. The system could recognize the scene by matching regions from the model image to the first video frame, find the select region and proceed to track and augment it.

The chapter is organized as follows. The next three sections describe the tracking algorithms. Sections 3.5 and 3.6 explain the Augmented Reality system. A variant of the tracker designed to run on multiple processors is briefly reported in section 3.7. Finally, section 3.8 presents experimental results and section 3.10 closes the chapter with an outlook.

Please note that (sub)sections marked with ¹ are joint work with the author's students Lukas Hohl and Till Quack, while those marked with ² with Michael Gauckler. More details about these works can be found in [Hohl and Quack 2003] and [Gauckler 2002] respectively.

### 3.2 Scheme of the tracker

The key idea for achieving computational efficiency while tracking the full set of affine parameters, is to decompose the affine space in two parts, and consider them sequentially. The tracker first exploits the image edges in order to find the point \( c \) and the orientation of the sides (see figure 2.4 for an illustration of a region). This corresponds to searching for the translation, rotation and shear components of the affine transformation. In the second phase, the image texture is examined, searching for the point \( q \), which corresponds to the two scale components (or scale and stretch, as named in section 2.3).
We have developed two versions of the tracker, which differ in the way they perform the first phase. The first version, which involves the cooperation between the Harris corner detector and a snake, is very efficient and can track a region at about 100 frames-per-second on a normal workstation (see result section 3.8). This version is reliable enough to be used in a number of practical situations. The second version abandons the Harris operator and detects a new type of corners instead. These are more repeatable and are tailored to our needs because they are guaranteed to lie at the intersection of two straight line segments. Although this version of the tracker is computationally more expensive (about 25-30 frames-per-second), it is also more robust and therefore applicable in particularly difficult situations where the first version might fail.

In this section, the common scheme underlying both trackers is introduced. We consider tracking a region $R$ from a frame $f$ to its successor frame $f+1$ in the input image sequence.

The first step is to compute a prediction

$$\hat{R}_{f+1} = A_f \cdot R_f$$

of $R_{f+1}$ using the affine transformation $A_f$ between the previous frame $f-1$ and frame $f$.

Next, we obtain an estimate

$$\hat{a}_{f+1} = A_f \cdot a_f$$

of the region’s anchor point and we define a circular search area $S_{f+1}$ centered on it. The radius $r_{f+1}$ of $S_{f+1}$ is set proportional to the current translational velocity of the region

$$r_{f+1} = \mu d(c_{f-1}, c_f)$$

with $\mu$ a predefined constant (typically $\mu = 2.0$), and $d(.)$ the Euclidean distance between two points. This allows the method to continuously adapt the search range so as to maximize the chances of finding $R_{f+1}$, while keeping it as small as possible (efficiency). Note how this scheme can track regions traveling at any speed, and accommodates accelerations of up to a factor $\mu$ per frame.

The anchor points in $S_{f+1}$ are extracted. These provide potentially better estimates for the region’s location. We investigate the point closest to $\hat{a}_{f+1}$ looking for the target region $R_{f+1}$. The anchor point extraction and investigation procedures differ for the two versions of the tracking algorithm and will be explained in the following sections. Since the anchor points are sparse in the image, the one closest to the predicted location is, in most cases, the correct one. If not, the anchor points are iteratively investigated, from the closest (to $\hat{a}_{f+1}$) to the farthest, until $R_{f+1}$ is found (figure 3.1). Hence, the investigation process runs on a sparse subset of points in $S_{f+1}$. This way of pruning the search space is made possible by the types of regions
3.3. Snake-based search

we consider and allows to keep the radius of $S_{f+1}$ wide enough to ensure tolerance to large image displacements, while staying within the tight time bounds imposed by real-time requirements.

In some cases it is possible that no correct $R_{f+1}$ is found around any anchor point in $S_{f+1}$. This can be due to several reasons, including occlusion of the region, sudden acceleration (the anchor point of $R_{f+1}$ is outside $S_{f+1}$) and failure of the anchor point extractor. When this happens the region's location is set to the prediction ($a_{f+1} = \hat{a}_{f+1}$), and the tracking process proceeds to the next frame, with a larger $S$. In most cases this simple mechanism allows to recover the region in one of the next few frames, while avoiding the computationally expensive process of searching the image further.

The tracking scheme can be summarized as follows:

1. Predict the target region $\hat{R}_{f+1} = A_f \cdot R_f$ and its anchor point $\hat{a}_{f+1} = A_f \cdot a_f$.
2. Define the search space $S_{f+1}$ and extract anchor points.
3. Investigate the closest anchor point searching for $R_{f+1}$.
4. Loop to 3 for the next closest anchor point if needed.

3.3 Snake-based search

In this section, we give the first variant of the procedure to investigate an anchor point, searching for the target region $R_{f+1}$.

The anchor points are Harris corners [Harris and Stephens 1988] extracted in the search area $S_{f+1}$. Given a corner point $h$, the region prediction $\hat{R}_{f+1}$, and the region in the previous frame $R_f$, we want to test if $R_{f+1}$ is anchored to $h$. If this is the case, we also want to construct $R_{f+1}$. The idea is to construct at $h$ the region which best corresponds to $R_f$. 
3.3. Snake-based search

Figure 3.2: A snake iteration. The center vertex $v_c$ can move to any of the 8 neighbors. The vertices $v_1$, $v_2$ can move to the two neighbors in the direction perpendicular to $v_cv_1$.

The process follows two steps. The first tracks two of the straight region sides exploiting the geometric information (edges) of the image, and already yields partial information about $R_{f+1}$. The second step starts from the output of the first, and completes $R_{f+1}$ based on intensity information (texture).

### 3.3.1 Finding the sides’ orientations

In the first step a polyline snake with three vertices recovers two of the sides, but not yet their lengths. We exploit the fact that translating $R_{f+1}$ so that $c = h$ automatically provides an estimation of the sides. We initialize the center vertex $v_c$ of the snake at $h$ and the other two vertices $v_1, v_2$ so that the line segments $v_cv_1$ and $v_cv_2$ have the orientation of the predicted region sides (figure 3.3). The three points are iteratively moved in order to maximize the total sum of gradient magnitudes along the two line segments:

$$E_S(v_c, v_1, v_2) = \sum_{p \in v_cv_1} |\nabla I(p)| + \sum_{p \in v_cv_2} |\nabla I(p)|$$

where $|\nabla I(p)|$ is the image gradient magnitude at pixel $p$.

The snake can only deform by hinging around $v_c$ and the length of the line segments is kept fixed (we are interested in their orientation only). These constraints reduce the number of degrees-of-freedom to four, thereby limiting the search space and improving efficiency.

The optimization process is efficiently implemented by a Dynamic Programming algorithm inspired by [Amini et al. 1990][Ueda and Mase 1992]. The algorithm has a higher probability of being attracted toward contours than the traditional snake implementation [Kass et al. 1988], and it is guaranteed to converge [Amini et al. 1990].
3.3. **Snake-based search**

At each iteration, \( v_c \) can move to one of the 8 neighboring pixels \(^1\), or stay unchanged (figure 3.2). For each of the 9 possible locations \( v'_c \), \( v_1 \) can move only along the line perpendicular to \( v'_c v_1 \) (conservation of length). Therefore, \( v'_1 \) can be in exactly 3 locations for any given \( v'_c \). The same holds for \( v_2 \). Hence, at first sight, at each iteration, computing the best \( v'_c, v'_1, v'_2 \) requires the evaluation of \( 9 \times 3 \times 3 = 81 \) configurations.

However, it is important to observe that moving \( v_1 \) does not affect the segment \( v_cv_2 \), and that there is no segment between \( v_1 \) and \( v_2 \). These observations reveal the structure of the problem and suggest the following optimization:

1. For each combination of the 9 possible \( v'_c \) and the 3 possible \( v'_1 \), we compute the sum of gradient magnitudes along \( v'_c v'_1 \). The best \( v'_1 \) as a function of \( v'_c \) is stored in a table, together with the corresponding sum of gradients.

2. The procedure of point 1 is repeated for the segment \( v_cv_2 \).

3. We select the location of \( v'_c \) which maximizes the total sum of gradients along \( v'_c v_1 \) and \( v'_c v_2 \) by simply looking up the tables computed in points 1 and 2. The corresponding locations of \( v'_1, v'_2 \) are directly obtained from the tables.

This algorithm only explicitly computes \( 9 \times 3 + 9 \times 3 = 54 \) configurations, but is equivalent to the brute-force solution discussed above. The power of this optimization strategy is more evident when considering the case where the segments' lengths are not kept fixed. As \( v_1, v_2 \) can then move freely to 9 locations, the brute-force approach would need to evaluate \( 9^3 = 729 \) configurations, while the optimized approach only \( 9 \times 9 + 9 \times 9 = 162 \). A similar line of reasoning is exploited in [Felzenswalb and Huttenlocher 2000] and generalized to chains of segments of any length, as well as trees.

In practice \( h \) is often very close (a few pixels) to the intersection point \( c \) of the target region sides. Moreover, the initial segments \( v_cv_1 \) and \( v_cv_2 \) are already oriented similarly to the searched region sides. Hence the initialization is often very good and this reduces the number of iterations and the risk of being attracted by nearby distractor edges.

The tracked region sides lift four degrees-of-freedom: the two coordinates of \( c = v_c \) and the orientations of the two sides. These correspond to the translation, rotation and shear components of the affine transformation \( A_{f+1} \) mapping \( R_f \) to \( R_{f+1} \). This is all the information extractable from the geometric features of the image.

\(^1\)The discretization step can be freely chosen. One pixel is considered here, in order to simplify the explanation.
3.3. Snake-based search

3.3.2 Finding the sides’ lengths

The two remaining degrees-of-freedom correspond to the coordinates of the point $q$. These arise from the scale and stretch components of $A_{f+1}$ and are derived from the texture content of the region by the second step of the algorithm.

An initial estimate $\hat{q}$ is obtained by aligning $\hat{R}_{f+1}$ on the structure formed by $v_1, v_c, v_2$, so that $\hat{c} = v_c$ and the sides are oriented like $v_c v_1, v_c v_2$ (figure 3.3). This estimation is refined by moving $\hat{q}$ so as to maximize the similarity between the resulting region $R_{f+1}(\hat{q})$ and the region in the previous frame $R_f$. As a similarity measure we use the normalized cross-correlation (NCC) between $R_f$ and $R_{f+1}(\hat{q})$ after aligning them via $A(\hat{q})$, the affine transformation mapping $R_f$ onto $R_{f+1}(\hat{q})$. Therefore, the objective function to be maximized is:

$$E_C(q) = \text{NCC}(A(q) \cdot R_f, R_{f+1}(q))$$

Notice that this similarity measure is invariant not only under geometric affine transformations, but also under linear transformations of the pixel intensities. This makes the tracking process relatively robust to changes in illumination conditions.

The maximization is carried out by Gradient Descent, initialized on $\hat{q}$, where at each iteration $\hat{q}$ is moved 1 pixel in the direction of maximal increase. Typically $\hat{q}$ is initialized close to the optimum, because most of the variability of the affine transformation is lifted by the sides tracking step. This strongly reduces the risk of converging towards a local maximum and keeps the number of iterations low. Extensive experiments confirm this consideration and indicate that, in most cases, 3 iterations are enough.

At the end of the second step, the most similar region to $R_f$ anchored to $h$ is constructed. This does not mean that it is the correct region though, as $h$ could just be the wrong corner. Hence, as final verification we check if the maximum cross-correlation value is above a predefined threshold (typically 0.9), otherwise the algorithm proceeds to the next closest corner, as explained in section 3.2.
3.4 Virtual corner search

This section explains the second variant of the tracker.

3.4.1 Motivation and idea

The tracking algorithm of the previous section starts by extracting Harris corners (HC) in the search area $S_{f+1}$. If no corner is extracted near the location of point $c$ of $R_{f+1}$, the algorithm fails. Unfortunately HC are not always repeatably extracted along the image sequence. In particular, significant scale changes (more than factor 2) are a considerable problem, as the HC detector is only rotation invariant (as studied by [Schmid and Mohr 1998]). Moreover, only regions with a visually strong corner at point $c$ can be tracked. However, many interesting regions have clearly delineated straight edges near $c$, but only a shallow corner at $c$. This happens when the scene itself offers only a weak corner (e.g. two straight line segments curving at their meeting point) or when the image neighborhood is degraded (e.g. motion blur). Such regions cannot be tracked by the method of the previous section, because their anchoring corner is not extracted repeatably enough.

Figure 3.4 shows an example sequence where HC fails to reliably detect the anchoring corner of the region to be tracked. The geometric transformations are mild, and mostly translations, but the image contrast is low around the desired corner.

In this section, we give an alternative solution for the step described in subsection 3.3.1. The idea is to determine $c$ as the intersection of two extracted straight line segments (or just ‘segments’ for short). More precisely, we are interested in the intersection of the support lines of the two segments (figure 3.5). The segments are found by first running the Canny edge detector [Canny 1986] and then fitting segments to the edgels.

In our experiments, the intersection of the support lines proved much more repeatable and accurate than HC. The reason is that support lines themselves are very stable, because it suffices that a segment is located and oriented somewhat correspondingly. Such segments are robustly extracted because of several reasons.

First, there is no need for absolute accuracy: the segment’s length need not match the previous frame, and its exact localization along its own direction is irrelevant. Errors in both these factors do not affect the support lines. As an interesting consequence, even if two near segments do not really intersect in the image, the intersection of their support lines is stable. In these cases, which are very common, the Harris detector is in trouble, because there is no strong gradient at the desired location (as seen in figure 3.4).
3.4. Virtual corner search

Figure 3.4: Low repeatability of the Harris corner detector. (a) First frame of a sequence, with region to be tracked ('NE') and its anchoring corner (lower right of 'E'). (b) Close-up on the first frame. (c to f) Close-ups on frames 20, 40, 60, 80. The corner is not detected in frames 20 and 60.

Figure 3.5: A straight line segment fit to an image edge, and its support line. The support line completely contains the segment, but has infinite length.

The second reason for the robust behavior is due to the edge detector, which suffers less from the scale-problem, because it looks for one-dimensional structures. Moreover, not the whole edge corresponding to the line segment need to be detected, but just a few approximately aligned pixels somewhere along it, so that a suitable segment can be fit.

In summary, line segments, up to the accuracy with which we need them, are easier to extract than HC.

Besides robustness, another advantage is that there is no need for a strong corner at c. Indeed, there is no need for a corner at all. It suffices that two straight line segments are repeatedly extracted near c. As mentioned before, this allows to track many more regions than with HC.
As a last advantage, this kind of corners are guaranteed to lie at the (approximate) intersection of two segments. In this sense, they are optimal for our task. Many corners of other kinds, which might distract the tracker and increase computation time, are not detected and therefore not considered in the next processing stage (the search for point q).

Because of their nature, this new kind of corners are coined *virtual corners*. In our context, their only disadvantage with respect to HC is the higher computation time needed for their extraction. However, this is still fast enough for the resulting tracker to operate at real-time speed for a region (section 3.8).

### 3.4.2 The algorithm

A tracking algorithm based on the above observations follows. First a brief scheme is given. Each step is explained in more detail afterwards.

1. Detect edges in a square around the search area $S_{f+1}$. Straight line segments are then fit to the detected edgels.
2. Form all pairs of *approximately* intersecting segments.
3. Score the pairs according to the spatial distance of the virtual corner from the predicted $\hat{c}$ and the difference of orientation between the line segments and the predicted region's sides.
4. Filter out pairs which differ too much from the predicted region's sides.
5. Inspect the remaining pairs in order of increasing score. Attempt to construct the missing point $q$ on the geometric structure formed by a pair. Stop when a satisfying pair is found and thus $R_{f+1}$ is completed.

#### 1. Extract segments

The Canny edge detector is applied in a square area which is made a bit larger than the search area $S_{f+1}$, in order to contain all relevant edges. This is important when the real location of point $c$ is near the border of $S_{f+1}$, and thus the region's sides might extend out of it.

Once edgels have been detected, straight line segments are fit to them iteratively by linear regression. Figure 3.7a show the segments extracted on the example of figure 3.4.
3.4. Virtual corner search

2. Form intersecting pairs

All pairs of segments are checked for approximate intersection. Let $s_1$ and $s_2$ be the two segments of a pair, and $x$ the intersection of their support lines. The distance of a segment to $x$ is defined as 0 if $x$ is on the segment, and as the Euclidean distance to the closest endpoint otherwise (figure 3.6). Two segments are deemed approximately intersecting if both are closer than a threshold to $x$. We use 13 pixels as a threshold. All pairs of approximately intersecting segments are selected for further processing. Note that two near, but not really intersecting, segments can be selected as well (figure 3.7b).

3. Score pairs

A score is assigned to each pair, measuring its similarity with the location and orientation of the sides of the predicted region $R_{f+1}$:

$$
\lambda d(\hat{c}, x) + (\theta_1 + \theta_2)
$$

The first term is the spatial distance of the virtual corner from the predicted $\hat{c}$, and relates to the translation component of the affine motion. Let $e_i, i \in 1, 2$, be the farthest endpoint of $s_i$ from $x$. Each $\theta_i$ term is the angular difference between the direction $\hat{c} \rightarrow \hat{p}_i$ and the direction $x \rightarrow e_i$. The segments are associated to the predicted sides so as to minimize the total difference $(\theta_1 + \theta_2)$. Note that each direction ranges in $[0, 2\Pi]$, hence $\theta_i$ range in $[0, \Pi]$. Both $\theta_i$ are related to the rotation and skew components of the motion. $\lambda$ determines the relative weight of the translation component and it is set to $\Pi/100$, thus giving more importance to the rotational component. This is justified as we expect more translation than rotation between two subsequent frames.

The lower the score of a pair, the more it is similar to the expected $R_{f+1}$.
3.4. Virtual corner search

Figure 3.7: Tracking based on virtual corners for frame 20 of the sequence in figure 3.4. a) Extracted straight line segments (black). b) Formed segment pairs (white). c) The predicted sides (white, near c), and the two pairs that remain after the filter (step 4 of the algorithm). All other pairs have segment directions differing too much from the prediction. The pair labeled 1 is inspected first, but the region constructed on it has too low NCC. Pair 2 is then tried and the algorithm successfully terminates.

4. Filter out impossible pairs

Any pair with a $\theta_i > \Pi/4$ is filtered out. This reduces considerably the amount of pairs reaching the final, and most expensive, processing stage, at no cost of quality. Indeed, rotations of more that $\Pi/4$ between two subsequent frames are nearly impossible.

5. Inspect pairs to compute q

Each of the remaining pairs automatically provides an hypothesis for the point c and the orientation of the region's sides. The remaining point q can now be searched for by the algorithm of subsection 3.3.2.

The pairs are listed in order of increasing score. The operation of searching for q is applied for the pair with the lowest score first. If the NCC value for the resulting completed region is above a threshold (typically 0.9), $R_{f+1}$ is considered successfully tracked and the algorithm stops. Otherwise, the next pair is inspected, and so on.

Figure 3.7c shows the two pairs remaining after the filter of point 4, and the order in which they are inspected. The system successfully manages to construct $R_{f+1}$ on the second pair.

Results, including computational times, for the virtual-corners-based tracker and the snake-based tracker are reported in section 3.8.
3.5 2D Augmented Reality

The previous section explained how to track a region through the image sequence. In this section, we explain how to superimpose a planar virtual texture to the scene, by attaching it to a tracked region. The texture changes shape exactly like the region, therefore reflecting the camera motion, and deforming as if it were part of the physical scene. This is possible because the tracker recovers the complete affine transformation of the region between frames.

We recall here the advantages of our approach to augmented reality:

- **Markerless.** No artificial landmarks are required. The virtual object is directly attached to a natural scene patch.

- **Uncalibrated.** No information about the camera, its motion, or the 3D structure of the scene, is given or computed.

- **Real-time.** The system can track and augment a region in real-time, even on modest computers (Ultrasparc-IIIi 440 Mhz).

The next subsection explains how to learn the geometric transformation of the tracked region and apply it to the virtual texture, while subsection 3.5.2 covers photometric changes.

3.5.1 Geometric transformations

The goal is to attach a texture to a tracked region $R$, in every frame of the sequence. The texture will be projected on the same plane as $R$.

The user provides a parallelogram-shaped texture $T$ and a set of three points $\Psi = \{l_1, l_2, l_3\}$ indicating where it should be placed. These points have to be selected in a user-defined reference frame $r$ only (e.g.: first frame) and define a parallelogram-shaped area $\Omega(l_1, l_2, l_3)$ where the texture should be mapped. This gives the user valuable creative freedom, because these points don’t need to coincide with the vertices of $R$. Nevertheless $\Psi$ should be chosen in the proximity of $R$, so as to maximize the quality of the results (the affine transformation of $R$ may only hold in a limited neighborhood). Besides, it must not be forgotten that $T$ will appear in the images as if it was on the same plane as $R$.

We restrict the explanation to a single frame $i$. The algorithm consists of two main steps. The first step constructs the affine transformation $M$ mapping the texture to frame $i$. The second step does the actual texture-mapping. The algorithm is iteratively applied to every frame of the sequence.
In order to construct $M$, we first compute $A$: the affine transformation mapping the region from the reference frame ($R_r$) to the current frame ($R_i$).

The placement of the texture is obtained by warping $\{l_j\}_{j=1,3}$ by $A$ (figure 3.8). By enforcing the correspondence between three texture corners and the obtained points $\{A_l_j\}_{j=1,3}$, we find the desired affine transformation $M$.

It is now possible to continue with the second step: augment the region by mapping $T$ to the image via $M$ (figure 3.9):

$$W(p) = T(M^{-1}p), \quad \forall p \in \Omega(Ml_1, Ml_2, Ml_3)$$

where $W(p), T(p)$ is the color of the augmented region, respectively of the texture, at pixel $p$. Since the region's points are in correspondence along the frames, the texture will be projected always on the same physical surface.

Partially transparent textures are easily integrated in this approach: all pixels of $T$ of a certain predefined color will not be projected on frame $i$. This simple process accommodates for artificial textures of any shape which often blend in much better with the real image. Animated textures are also supported by projecting on each frame a different texture from a user-given texture sequence.

---

**Figure 3.8**: Computing the texture placement in frame $i$. The affine transformation $A$ of $R$ is applied to the user-selected points $l_1, l_2, l_3$

**Figure 3.9**: Mapping the texture $T$ to frame $i$. 

3.6. 3D Augmented Reality

3.5.2 Photometric transformations

In order to achieve a more realistic effect, the virtual texture's colors should also be adapted to the changing environmental conditions. For example, if the region gets darker because it turns away from the light source, the virtual texture should also have darker colors. Besides, other factors like the spectral composition of the light and the position of the light source also influence the colors of the region. We cope with these phenomena in an analogous way to the previous subsection: the photometric transformation of the region is learned from the image and applied to the virtual texture.

In the initialization step, the tracked region \( R_r \) in the reference frame \( r \) is analysed. The average values of the red, green and blue channels are computed \( \mu_r^{\text{red}}, \mu_r^{\text{green}}, \mu_r^{\text{blue}} \).

In each frame \( i \) where the virtual texture is to be overlaid, the channel averages are recomputed for \( R_i \), giving \( \mu_i^{\text{red}}, \mu_i^{\text{green}}, \mu_i^{\text{blue}} \). The ratios of the channel averages give 3 scale factors:

\[
\phi_{\text{red}} = \frac{\mu_i^{\text{red}}}{\mu_r^{\text{red}}} \quad \phi_{\text{green}} = \frac{\mu_i^{\text{green}}}{\mu_r^{\text{green}}} \quad \phi_{\text{blue}} = \frac{\mu_i^{\text{blue}}}{\mu_r^{\text{blue}}}
\]

These factors represent the photometric transformation from \( R_r \) to \( R_i \).

Finally, the RGB values of each pixel of the virtual texture \( T \) are multiplied by the corresponding scale factors. This achieves the desired effect of adjusting the colors of \( T \) to suit the photometric changes of the tracked region.

3.6 3D Augmented Reality

While in the previous section the virtual objects were required to be planar, here a more general technique, capable of dealing with full 3D objects, is introduced. To this end, two regions need to be tracked, and should fulfill the following two requirements:

- **Non-coplanarity.** The two regions should lie on different planes.
- **Proximity.** The two regions should be close to each other.

As they are met quite easily, these requirements do not impose serious restrictions to the system. As a matter of fact, since the tracked regions can be small, it is not difficult to find two nearby non-coplanar regions to be tracked. Besides, only the first requirement is mandatory. The second merely affects the accuracy of the registration of the virtual object.
The approach is still completely uncalibrated. The appearance changes of the 3D object due to the camera motion are entirely deduced solely from the point correspondences within the two tracked regions.

The two regions provide two independent triples of point correspondences along all frames. From these, it is possible to align the real and virtual coordinate systems, hence to bring 3D coordinates of the virtual object in correspondence with 2D image points. The input and notations are formalized in subsection 3.6.1.

In subsections 3.6.2 and 3.6.3, two distinct scenarios are presented. In the simpler one, the virtual object is directly attached to the two tracked regions. The advanced scenario lets the user choose exactly where in the scene to put the virtual object. However, this flexibility doesn’t come for free. As discussed later, the additional information provided by the user is typically not very accurate. This is reflected in inferior registration of the virtual objects in the advanced scenario.

### 3.6.1 Input and notation

Both the simple and the advanced scenario need the following input.

- Two non-coplanar tracked regions. Four points $p'_0, p'_1, p'_2, p'_3$, no three coplanar, are selected from these regions, so that they define the projection of a real world coordinate system (figure 3.10a). We call this the region-base.

- A 3D virtual object to be placed in the scene. We assume it to be represented by point vertices. The bounding box of the object is the smallest parallelepiped that contains the object. It is readily obtained from the vertices, by considering the outermost ones. We select four corners of the 3D bounding-box $P_0, P_1, P_2, P_3$. These define the virtual coordinate base of the object (figure 3.10b).

We use homogeneous coordinates, and denote a 2D image point $p_k$ as $p_k = (x_k, y_k, 1)$, and a 3D point as $P_k = (X_k, Y_k, Z_k, 1)$.

In general a 3D world-point is projected to a 2D image point by a 3x4 projection matrix $\mathcal{P}$.

$$
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
= \mathcal{P} \times
\begin{pmatrix}
  X \\
  Y \\
  Z \\
  1
\end{pmatrix}
$$

\[
\mathcal{P} =
\begin{pmatrix}
  p_{11} & p_{12} & p_{13} & p_{14} \\
  p_{21} & p_{22} & p_{23} & p_{24} \\
  0 & 0 & 0 & 1
\end{pmatrix}
\]
The last line is \((0 0 0 1)\) because we use scaled orthographic projection (no perspective entries in \(P\)).

### 3.6.2 Direct placement

In order to properly draw the 3D virtual object on the scene, we need to find a projection matrix \(P\) mapping the bounding box to the correct location in each frame. In the direct-placement approach, the object is directly mapped to the location of the tracked regions. Hence, we compute \(P\) by enforcing the correspondence between the bounding box corner-points and the region-base points in the image \(P_k \leftrightarrow p'_k\), \(k = 0..3\).

This results in an equation system with 8 equations and 8 unknowns (the entries \(p_{11}, \ldots, p_{24}\) of \(P\)). Each point correspondence provides two equations, e.g. for point \(P_1:\)

\[
\begin{align*}
x'_1 &= p_{11} \cdot X_1 + p_{12} \cdot Y_1 + p_{13} \cdot Z_1 + p_{14} \\
y'_1 &= p_{21} \cdot X_1 + p_{22} \cdot Y_1 + p_{23} \cdot Z_1 + p_{24}
\end{align*}
\]

In every frame where the object should be overlaid, a new projection matrix \(P\) is calculated by solving the linear system. The region-base points \(p'_k\) change from frame to frame and are found by the tracker, while the bounding-box points \(P_k\) stay constant.

Once the projection matrix is obtained, the whole 3D object can be drawn on the image via standard graphics libraries (we used OpenGL [Hohl and Quack 2003]).

---

**Figure 3.10:** (a) Two non-coplanar regions in an image, and the selected region-base points. (b) A pyramidal virtual object (thick) and its bounding-box (thin).
With this simple approach, the object stays accurately registered with the natural scene, but the user is given only limited choice of where to place the object (i.e. only up to the choice of the support regions).

### 3.6.3 User-selected placement

In the advanced approach, the user can choose the exact location where to place the virtual object. Nevertheless, this should be in the proximity of the tracked regions, in order to obtain good registration.

In addition to the two tracked regions and the virtual object, the user must now provide:

- Four image points $p_{0a}, p_{1a}, p_{2a}, p_{3a}$ in a first reference frame $a$. Typically $a$ is the first frame, but any frame can be chosen.

- The corresponding four image points $\{p_{kb}\}_{k=0..3}$ in another frame $b$ (e.g. the last one), showing them under a different viewpoint.

Each set of points specifies the desired location for the projection of the coordinate base of the virtual object in a frame.

Before computing the projection matrix $P$ for a frame $i$, the correspondence between the coordinate base defined by the user and the tracked regions' base needs to be established. This is equivalent to finding the projections of the bounding box points $P_k$ in frame $i$, according to the user choices for frames $a, b$.

Recall that the region-base $\{p'_k\}_{k=0..3}$ consists of four points from the tracked regions. According to this base, any 3D point $X_s, Y_s, Z_s$ projects in the image to:
Consider the case where $X_s, Y_s, Z_s$ are the 3D coordinates of a bounding box point expressed in the 3D coordinate system defined by the tracked regions. These points need to be calculated.

We apply equation (3.3) to each point of the bounding box $P_k$, with the base $\{P_{ka}\}_{k=0..3}$ from the first reference image instead of the region-base $\{P_k\}_{k=0..3}$. This results in 2 equations and 3 unknowns per point (the unknowns are $X_s, Y_s, Z_s, s = 0..3$). Hence the system is under-determined. By using the base from the other reference image $\{P_{kb}\}_{k=0..3}$, we get another 2 equations, with the same 3 unknowns, per point. In total, we now have a system of 4 equations and 3 unknowns, which can be solved. For example, the system for point $P_1$ of the bounding box is:

\[
\begin{pmatrix}
(x'_{1a} - x'_{0a}) & (x'_{2a} - x'_{0a}) & (x'_{3a} - x'_{0a}) \\
(y'_{1a} - y'_{0a}) & (y'_{2a} - y'_{0a}) & (y'_{3a} - y'_{0a}) \\
(x'_{1b} - x'_{0b}) & (x'_{2b} - x'_{0b}) & (x'_{3b} - x'_{0b}) \\
(y'_{1b} - y'_{0b}) & (y'_{2b} - y'_{0b}) & (y'_{3b} - y'_{0b})
\end{pmatrix}
\begin{pmatrix}
X_1 \\
Y_1 \\
Z_1
\end{pmatrix}
= 
\begin{pmatrix}
x_{1a} - x'_{0a} \\
y_{1a} - y'_{0a} \\
x_{1b} - x'_{0b} \\
y_{1b} - y'_{0b}
\end{pmatrix}
\]

The subscripts $a$ and $b$ refer to the two reference images.

Repeating the operation for the three remaining points of the bounding box yields all desired $X_s, Y_s, Z_s, s = 0..3$. These new 3D points can now be used in equation (3.3) to generate a new region-base in frame $i$. These new four image points correspond to the user-selected base (in frames $a, b$) and also to the four points on the 3D bounding box. Hence, the projection matrix for frame $i$ can be computed in the fashion of subsection 3.6.2.

This approach offers high flexibility because the object placement can be freely chosen. However, the user will typically not mark exactly the same points in the two images, especially if the viewpoints are substantially different. This results in less accurate augmentations, compared to the direct method of subsection 3.6.2. Moreover, if the user selects points far from the tracked regions, the registration accuracy degrades further.

### 3.7 Parallel tracking

#### 3.7.1 Motivation and idea

The tracking algorithms described in the previous sections are computationally efficient. A single region can be tracked in real-time (30 frames-per-second) on a
low-class workstation, and a few of them on a standard one. This performance is enough for some interesting applications, like Augmented Reality.

Nevertheless, regions are small, while one might want to have a more global coverage of the scene, as needed in 3D Reconstruction from video [Heyden and Astrom 1997, Pollefeys et al. 1998], or object modeling for recognition. In this case, the scene can be divided in a large number of regions, which are then independently tracked. On one hand, such an approach is robust to occlusions and noisy or featureless image areas, because the loss of a few regions doesn’t affect the others. On the other hand, the regions are tracked independently, so the computational cost grows roughly linearly with their number. In this context, real-time performance for a few regions might no longer be enough. The total processing might still take hours, if not days.

In order to cope with these enormous computational demands, we have developed a parallel-tracker, which uses a pool of processors to track a large number of regions simultaneously. The following subsections explain the main concepts behind the parallel-tracker at a rather abstract level. More details, as well as information related to the processor pool and the program’s implementation, can be found in [Gauckler 2002].

### 3.7.2 Architecture

The organization of the processor pool is described in this subsection, while the next one covers its functioning.

The system is based on the master-slave model. We assume to dispose of $S + 1$ processors. One of the processors is chosen to be the master, while the remaining $S$ processors are used as slaves and actually do the processing work. The master distributes work to the slaves and coordinates their activity. All the information related to the global tracking problem and its current state of processing are kept in the master. The slaves are only aware of the minimum information needed to perform their localized task. The number of slaves $S$, which is bounded by the available hardware resources, is kept fixed during operation and can be chosen at system startup.

The problem of tracking a set of $N$ regions is decomposed in the $N$ subproblems of tracking a single region. In order to simplify the following explanations, we assume to have more regions $N$ than slaves $S$. At runtime, the master gives to each slave regions to track one at a time. A slave performs all steps required to track the assigned region (e.g: local edge detection, virtual corners extraction, q-point finding, etc.).

An alternative to this vertical decomposition, is to decompose the problem horizontally. This approach would start by performing the first tracking step for all regions
(e.g.: edge detection for the whole image), then pass to the next step. All slaves would contribute to solve each step (e.g.: for the first step, by assigning a different image block to each slave).

We have chosen the vertical decomposition. It regards regions as the processing units and has several advantages. First, it makes the system simpler to understand and to implement, starting from the sequential tracker program. Secondly, it simplifies dealing with crashes of the individual slaves. When a slave fails to respond for a certain time interval, the master can just reassign its region to another available slave. In the alternative horizontal decomposition, such robustness would require the design of complex update/recovery rules in order to keep the state consistent. This is due to the fact that slaves depend on the information produced by the others in the horizontal decomposition, while they are independent in the vertical decomposition.

The disadvantage of the chosen decomposition is that some operations (e.g.: corner detection) might be repeatedly performed by different slaves for regions which are close in the image. However, this only marginally affects performance.

3.7.3 Workflow

After appointing a master and preparing the slaves for operation, the system starts to track the regions. The tracking happens frame-by-frame, according to the following scheme.

The master sends to each slave a request to track a region from the previous frame to the current frame. When a slave completes the task, it sends the result to the master. The master collects the region, updates its data-structures, and sends the request to track another region to the slave (figure 3.12).

At some point in time, the master will have sent out all $N$ requests. After this moment, all slaves completing regions stay idle until the last slave is done. Only then the master moves to the next frame and starts distributing regions again. Upon completion of the last frame, the processing stops.

The presented approach keeps the slaves synchronized, in the sense that they all track regions in the same frame. If the whole image sequence is available beforehand, an unsynchronized scheme would be possible. For instance, a region could be assigned to a slave once and for all. The slave could track it until the end of the sequence, and only then report to the master and get a new region to track. Such a scheme would have lower communication cost than the chosen one, and no slave would stay idle between frames. However, our approach has other advantages.

First, regions are dynamically reassigned to potentially different slaves at each frame. When some regions take significantly longer time to track than others, this allows a
better load balancing and therefore leads to a lower total computation time. In fact, in the asynchronous approach, the slave which completes first has to stay idle until the last one is done. The difference in the time required to track a region can be due to the slave being a slower processor, or to the region being harder to track (e.g.: irregular motion, many spurious corners nearby). In our approach, such a slow slave might track only one region in a certain frame, while, at the same time, another slave tracks two or more. The time slice of both slaves is thus well exploited. Especially when $N$ is much larger than $S$, the average time a slave stays idle between frames becomes very small, compared to the working time. Moreover, communication costs on a typical intra-processor network are orders of magnitude inferior to both the tracking costs and the idle periods. These claims are demonstrated in section 3.8.

Moreover, even assuming that all slaves track at the same speed, if $N$ is not a multiple of $S$, there will be $N - (N \mod S)$ idle slaves in the last cycle of the asynchronized approach. The chosen approach instead nicely balances the load, independently of the values of $S$ and $N$.

The second advantage of our region-distribution strategy is in the robustness to crashes. The system can handle the crash of a slave simply by reassigning its region to the next available slave. The crash is noticed straight away, in the very frame where it happens, and therefore causes the loss of only a little time. In the asynchronous approach instead, the master notices a crash at the very end, and only then can it reassign the region. The work of tracking the region through the whole sequence is lost.

The third advantage is conceptual: if the image sequence is being produced at the time of tracking, the asynchronous approach might not be able to exploit fast slaves.
3.8 Results

3.8.1 Tracking and augmented reality

We present several image sequences demonstrating the qualities of the tracking and augmented reality system proposed in this chapter. The images of the tracked regions are put in complete correspondence along the frames, allowing to define in every frame an affine reference coordinate system where to project the virtual objects (sections 3.5 and 3.6). All presented sequences have a spatial resolution of 720x576 pixels per frame. Many of them can be downloaded from www.vision.ee.ethz.ch/~ferrari/movies.

Cinema

The Cinema sequence (figure 3.13) shows the tracker’s robustness to discontinuous motion. A parallelogram-shaped region on a window of the background building is tracked and two advertising neon-light characters AR are added on the wall. Notice the texture is not projected on top of the region, but to its right, because this better fits in the scene. This is allowed by the mechanism explained in subsection 3.5.1. The AR characters change color, from red to blue, every 20 frames, to simulate a blinking advertising neon-light.

The sequence is taken by a fast moving hand-held camera and therefore contains mostly irregular motion: the region often bruskly changes direction and velocity, making it hard to predict the next location accurately. Moreover, the region moves fast: the displacement between subsequent frames is often large. As a result, the predicted location is often far from the correct one. The region was successfully tracked in every frame despite displacements of up to 22 pixels between subsequent frames and predictions that were off by up to 22 pixels from the target. The irregularity of the motion is illustrated by the regions' trajectory in figure 3.13c.

The physical planar patch covered by the region is accurately tracked along the sequence, as proven by the constantly high cross-correlation score (average 0.97). This is very important for the quality of the final visual impact of the augmented scene, as it directly influences the accuracy by which the artificial texture is mapped into the real environment. As a result, the texture is perceived as rigidly sticking on the building’s wall and relative displacements are hardly noticeable.

The computational performance meets the real-time expectations. Computation times for this and other sequences are summarized in table 3.1 and are discussed below.

Desk

Control of out-of-plane rotation is exemplified in the Desk sequence (figure 3.14). The challenge is posed by the region’s motion, which contains significant rotation
3.8. Results

**Figure 3.13:** Cinema sequence. a) tracked region and AR texture in frame 1. b) texture in frame 74. c) trajectory of the region along the whole sequence (scaled to show a window of 140x220 pixels around the mean location). d) cross-correlation score (y axis) in each frame (x axis). e) velocity (dotted) and prediction errors (in pixels).

around the vertical axis, causing skew and stretch effects in the image. The tracker was able to handle the situation by correctly transforming the 2D region's shape: despite the very different viewpoints of frames 1 and 168, the region is covering the same physical surface. This feature of the tracker allows to attach the BIWI logo to the top-right corner of the poster, and to deform it so as to reproduce the effects of viewpoint changes. This is crucial for the quality of the resulting augmented scene.

**Electro**

The sequence in figure 3.15 illustrates a potential application of the AR system, and it involves an animated virtual texture. The goal is to visually help a worker, who is supposed to wear a head-mounted display, in the task of sticking a lightning metal plate on an electricity control panel. As the panel comes into the field of view, a glowing signal attracts the worker’s attention to the area where the plate has to be stuck. When the worker approaches further, the lightning plate appears exactly where it should be attached, looking like it was already stuck on the panel. As the worker moves, the superimposed plate moves and deforms with the viewpoint, in order to align with the image of the panel. The quality of the augmentation
Figure 3.14: Desk sequence. Left: region in frames 1, 86 and 168. Right: BIWI logo in the same frames.
Figure 3.15: Electro sequence. a) region and signal in frame 1. b) signal in frame 24. c) lightning sign in frame 101 d) lightning in frame 202.

alignment is once again ensured by the accurate correspondences of the tracked region between frames (cross-correlation average 0.94).

The tracker successfully dealt with the out-of-plane rotation of the region and was not distracted by the specular reflections on the metal plate.

Jennifer

The Jennifer sequence shows the combined application of scene recognition, region tracking and augmentation. The left image of figure 3.16 shows the model image, representing the scene, and the region to be augmented (dark parallelogram in middle). The right image is the first frame of the sequence. The system successfully extracts and matches in this frame many regions of the model view, and therefore understands this is the sequence it has to augment. The particular region to be augmented is found in the set of matched regions. This matching process is achieved through the traditional techniques introduced in sections 2.4, 2.5.
3.8. Results

Figure 3.16: Jennifer: scene matching. Left: scene model. Right: first frame of the sequence.

Once the target region is found, a sticker with the text ‘ISAR’ is attached to it and correctly rotated to fit the camera orientation (figure 3.17). Hence, this sequence demonstrates the tracker’s ability to follow in-plane rotations.

Computation times

Table 3.1 reports the computation times for the four sequences presented so far. Each sequence has been tracked first with the method based on the combination of Harris corners and snakes (section 3.3), and then once again with the method based on the virtual corners (section 3.4). All times are in milliseconds (ms) and the experiments have been performed on a workstation with modest computational power (SunSparcIII 1.2Ghz).

The first method (harris + snake) is very efficient and tracks a region in about 10 ms. Hence, it could track 3 to 4 regions at 30 Hz, which is a strong real-time bound.

As expected, the second method (virtual corners) requires more computational power. Nevertheless, it still performs at about 30 Hz for 3 of the sequences, with only Cinema taking 57 ms per frame. The reason lies in the region’s fast (10 pixels per frame in average) and chaotic motion, which makes the search area $S$ (section 3.2) quite large for most frames. In this area $S$, we apply the Canny edge detector, which is the slowest part of the method and significantly slower than corner detection. Nevertheless, it’s important to notice that 57 ms corresponds to 18 frames-per-second, which is a high rate (near real-time) and can still be sufficient for a number of applications.

$^{2}$The values differ from those first reported in [Ferrari et al. 2001a] because another computer (SunSparcII 440 Mhz) was used in that work. Note that the harris+snake method could track a single region in real-time even on this very slow computer.
Figure 3.17: Jennifer sequence. Left: region in frames 1, 30 and 50. Right: ISAR sticker in the same frames.
3.8. Results

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Time (harris + snake)</th>
<th>Time (virtual corners)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinema</td>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>Desk</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>Electro</td>
<td>11</td>
<td>34</td>
</tr>
<tr>
<td>Jennifer</td>
<td>10</td>
<td>37</td>
</tr>
</tbody>
</table>

**Table 3.1:** Average time to track a frame (in milliseconds). The real-time bound is 34 ms (30 Hz).

The total processing time also includes the time used to draw the virtual texture. Currently, this operation is carried out by the main processor. Fortunately this takes only about one millisecond per frame, for image areas of the size of the above examples (about 80x60 pixels). Hence, the complete track-augment cycle can be accomplished in real-time. However, for augmenting much larger areas, a graphic-accelerator board could be necessary in order to keep the texture mapping overhead low.

**Cinema - massive tracking**

Up to now, all presented cases could be dealt with by both tracking methods. However, the method based on virtual corners is more powerful and robust, because it can track many regions that the other method cannot.

For example, figure 3.18 shows 35 regions tracked successfully, and well covering the whole image in the Cinema sequence. In our experiments, we found that many of these regions couldn’t be tracked by the harris+snake approach. For instance, harris+snake fails for regions on the 'CINEMA' characters. In this sequence, the failures are mostly due to the insufficient repeatability of the harris corner detector, as discussed in section 3.4. This problem manifests itself mostly at weak corners, i.e. image areas which are perceived as corners by humans, but have low gradient values. In these cases, the corner detector fails to produce a corner. However, in many such cases, some edges are detected in the surrounding area and the line segments fit to them provide a virtual corner at an adequate location. A secondary problem lies in the harris corners’ inferior accuracy, when compared to virtual corners. This sometimes reveals insufficient as initialization for the polyline snake.

**Spaceship**

Another example of the wider applicability of the virtual-corners tracker is given in figure 3.19. The two regions lie on low contrast areas, especially the region on the left. As a consequence, only weak edges and corners can be observed in the images (figure 3.19 bottom left). Moreover, the region on the right moves away from the camera for a long period, gradually becoming smaller and smaller. This results in the region appearing over a wide range of scales: between the first and last frame there is a scale factor of 4. The virtual corner tracker coped with these challenging
conditions and accurately tracked the region, including adapting its size to reflect the scale variation (figure 3.19 bottom right).

The harris+snake tracker fails for both regions. For the bottom region, the Harris detector proved just not repeatable enough under such large scale variations. As for the left region, the failure is partly due to the edges being too weak for the snake to work properly: even when a corner is detected near the desired location, the snake doesn’t manage to follow the rotation of the sides accurately enough, and gradually drifts away from the correct orientations, never recovering.

The two regions are tracked by the virtual corners approach at a speed close to real-time (25 ms for the left region and 51 ms for the bottom one).

**Smile**

The smile sequence is affected by all the aforementioned difficulties at once. The two tracked regions undergo various transformations while they evolve through the sequence. Initially, the motion is mainly an in-plane rotation (figure 3.20-top). Next, the camera quickly zooms away from the scene causing a rapid scale decrease (there are only 15 frames between frames 21 and 36). Out-of-plane rotation follows (frame 111) and finally the camera quickly pans to the left (frame 145) During the whole sequence the camera moves shakily and irregularly, with sudden alternations of slow and fast motions.

The two regions are tracked by the virtual corners method, which succeeds in updating the regions’ shapes and locations so as to keep them covering the same physical surface in all frames. Once again, the computational performance closely approaches real-time: 35 ms for the region on the horizontal facet of the book, and 43 ms for the region on the vertical facet.
Figure 3.19: Spaceship sequence. Top-left: the two regions in frame 1. Top-right: regions in frame 36. The left region is about to be occluded. Middle: the bottom region tracked in spite of large scale change in frames 86 and 136. Bottom-left: the left region is anchored to a weak corner and weak edges (zoom factor 2 on frame 1). Bottom-right: effects of scale change and accuracy of tracking for the bottom region (zoom factor 4 on frames 1 and 86).
Figure 3.20: Smile sequence (regions). The two tracked regions and the bounding box of the virtual object in 6 frames. Top: frames 1 and 21. Middle: frames 36 and 66. Bottom: frames 111 and 145.
3.8. Results

Based on the point tracks provided by the region, a Buddha-shaped virtual 3D object has been added to the scene (figure 3.21). The virtual object is attached directly to the tracked regions, by the method described in subsection 3.6.2. A particularly nice effect is due to the anti-clockwise viewpoint rotation: the right side of the object, which is initially occluded, becomes visible later in the sequence (starting at frame 66). This underlines how the proposed approach rigidly registers the virtual object with the scene.

A different image sequence of the same physical scene is shown in figure 3.22. This time the user applied the method of subsection 3.6.3 in order to place the Buddha statue in the middle of the book, and to make it bigger. Again, the system correctly overlays the virtual object, in spite of rotations, scale and out-of-plane rotation effects. However, the accuracy is lower than when the object is directly attached to the regions, which diminishes the quality of the augmentation. This is due to the nature of the method, which requires the user to select corresponding points in two substantially different views, an operation which can hardly be performed accurately enough.

As can be seen from these two image sequences, the system only needs each tracked region to be on a locally planar surface, and does not require a large, dominant physical plane to work (in contrast to [Simon et al. 2000]).

Internet

All of the above examples focused on the geometric transformation of the virtual object. In the last sequence, we demonstrate the importance of caring for photometric transformations as well. The icon of a magazine is pasted over a colorful banner in figure 3.23. In the sequence, the banner tilts away from the light source, getting progressively darker. The photometric transformation is automatically learned from the sequence and transferred to the virtual texture by the method of subsection 3.5.2. The augmented sequence appears definitely more realistic than with geometric transformations only.

3.8.2 Parallel tracking*

We briefly report here on the principal results achieved by our parallel tracker implementation.

The hardware architecture of the parallel processing system consists of 20 Alpha 21264 processors, running at 667 MHz. The processors are connected by a 100 Mbit/s fast Ethernet network as well as a Myrinet network providing data rates up to 1280 Mbit/s. The standard Ethernet is used for booting and administrating the machines, and starting the parallel software. Once the parallel program is invoked, the nodes communicate through the faster Myrinet channel.
Figure 3.21: Smile sequence (augmented). The statue of the Buddha moves and deforms according to the viewpoint change. Notice how the right side appears during the sequence, as it becomes visible to the camera.
3.8. Results

The experiments are based on the Cinema sequence of figure 3.18, where 35 regions are tracked across 40 frames. Table 3.2 reports the processing time for an increasing number of regions. The measured time is the total processing time, from the program start, until the last region of the last frame has been tracked. Sets of more than 35 regions are obtained by simply replicating the initial ones.

We are interested in the parallel speedup, i.e.: the amount of time taken by a single slave divided by the time taken by $S = 20$ slaves. The speedup is obviously bounded by 20, which represents a perfectly efficient system. The table shows the speedup to improve together with the problem size $N$ (number of regions). When the problem is sufficiently large ($N = 280$), our system proves very efficient (speedup 15).

The increase in efficiency with the problem size is mainly due to two factors. First, the fraction of the time a slave stays idle between two frames, due to the synchronization explained in subsection 3.7.2, decreases with increasing number of regions.

**Figure 3.22:** Smile2 sequence. Regions and augmentation in four frames of the sequence.
3.8. Results

Figure 3.23: Internet sequence. Left: augmentation in frame 1. Right: augmentation in frame 50. The virtual texture (magazine icon on bottom-left) is drawn darker, like its surroundings. The region (not shown) is at the same location as the texture.

Table 3.2: Performance of the parallel tracker.

<table>
<thead>
<tr>
<th>Number of regions</th>
<th>Time [sec] (20 slaves)</th>
<th>Time [sec] (1 slave)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>49</td>
<td>284</td>
<td>5.80</td>
</tr>
<tr>
<td>70</td>
<td>59</td>
<td>536</td>
<td>9.08</td>
</tr>
<tr>
<td>105</td>
<td>80</td>
<td>800</td>
<td>10.00</td>
</tr>
<tr>
<td>140</td>
<td>105</td>
<td>1085</td>
<td>10.33</td>
</tr>
<tr>
<td>280</td>
<td>145</td>
<td>2198</td>
<td>15.16</td>
</tr>
</tbody>
</table>

tracked per slave ($N/S$ on average). Secondly, in a similar way, the fixed cost of loading the image of the current frame in the slaves loses impact with increasing $N/S$.

The communication time used to circulate region coordinates and tracking requests can be neglected from the analysis, as it has been measured to correspond to about 2% of the total processing time of a slave. This advantage is partly due to the extremely fast Myrinet, but we feel that it might extend to systems equipped with a slower network, because only a small number of very small messages (roughly 30 bytes) are necessary.

This experiment, together with others reported in [Gauckler 2002], confirm the viability of our parallel tracking approach and they show how the design choices of section 3.7 are particularly valid when the problem size $N$ is much larger than the number of slaves $S$. 
3.9 Related work

In this section we discuss similarities and differences to related works, taking into account that the techniques presented in the chapter have been, in large part, published in 2001 [Ferrari et al. 2001a, Ferrari et al. 2001b].

3.9.1 Tracking

A large body of work in visual tracking deals with the problem of determining the pose of a 3D object in an image sequence, given its 3D model as input [Krahnstoever and Sharma 2003, Lepetit et al. 2003]. Other modern works have specialized on high-dimensional, articulated motion, such as the human body [Sminchisescu and Triggs 2001] and hands [Lu et al. 2003, Bray et al 2004]. Although very interesting, these model-based approaches are not be discussed here as they treat a very different problem than ours, which is the tracking of relatively small image regions, without any additional information.

Earlier, classic feature trackers have only considered image translations [Lucas and Kanade 1981]. In more recent years, several real-time region trackers based on the color histogram have appeared, starting from [Birchfield 1998]. Comaniciu [Comaniciu et al. 2000] uses mean-shift optimization to robustly track the region’s translation. The method was recently upgraded to properly track over scale changes [Collins 2003]. As pointed out by Perez [Perez et al. 2002], these methods might be distracted by parts of the background near the target region exhibiting similar colors. Perez proposes to alleviate this problem by a non-deterministic search, based on a particle filter [Isard and Blake 1998], which can temporarily maintain multiple hypothesis. Our approach does not suffer from this problem, as it considers the spatial structure of the region as well. All these methods achieve an excellent computational performance, but they only consider a low dimensional search space for the region transformation, including at most translation and scale. Therefore, they are unable to correctly handle the shearing and stretching effects caused by out-of-plane rotation, because the shape of the tracked region remains the same. Moreover, under in-plane rotation, these trackers simply stay on the target, without recovering the rotation. Our tracker instead, recovers the full affine transformation, and hence produces three accurate point correspondences under general motions. This is very useful when tracking is not done per se, but instead is used as input for applications such as Augmented Reality, structure-from-motion or geometric grouping (e.g. as in chapter 6). On the other hand, the mentioned trackers are more generally applicable than ours, in that they are not limited to planar regions, and have been demonstrated even on deformable objects.
Trackers dealing with the complete set of affine motion parameters have been proposed [Bascle and Deriche 1995, Toklu et al. 1997], but they rely on much heavier techniques which make them slow. Moreover, they often need a large image region, and its complete contour, what strongly limits the number of regions that can be tracked in a scene.

Coming closer to our work in this sense, is the seminal paper by Shi and Tomasi [Shi and Tomasi 1994]. A small image window is tracked by finding the motion parameters that minimize pixelwise dissimilarity between frames. During tracking, a simple translations are considered, while the affine model is only used when necessary to evaluate the quality of tracking. The authors note that estimating the affine deformations in all frames can compromise the quality of the computed displacement, especially with small motions, because of the numerical interaction of all parameters in their minimization algorithm. Thanks to the meaningful decomposition of the affine space in our approach, the computation of the translation and deformation components of the motion do not interfere, and they are not disturbed by small motions. The affine transformation is directly, fully recovered in all frames. Another drawback of [Shi and Tomasi 1994] is the lack of compensation for illumination changes. This can cause failure even under out-of-plane rotation, as demonstrated by [Jin et al. 2001]. Moreover, the minimization technique adopted only allows a small displacement between subsequent frames, as also pointed out by [Kang and Szeliski 1996]. Illumination changes are covered by our tracker (figure 3.23), while tolerance to large displacements (e.g. 20 pixels in figure 3.13) is ensured by a special strategy for pruning the translation subspace (section 3.2). The work of [Jin et al. 2001], which was published roughly at the same time as ours, extends [Shi and Tomasi 1994] to illumination changes, presents a cleaner handling of affine transformations and achieves real-time performance. Thus, it has functionality comparable to our tracker, but only allows smaller inter-frame displacements. On the other hand, it can track any planar region, whereas our method is constrained by the need of two short straight edges.

Another appealing region tracker is the one of [Jurie and Dhome 2001], also published in 2001. The method is based on a brilliant idea: during an off-line stage, a relation between the variation of intensities of the region and the variation of position (motion parameters) is learned by randomly perturbing the region and observing the corresponding changes in brightness. The relation comes in the form of a linear transformation, whose parameters are learned by solving a large linear system by least squares. Once this linear transformation is known, tracking just amounts to computing the difference image between two subsequent frames and then applying the inverse transformation to get the motion parameters. This method is extremely fast (more than real-time) and works under the full affine, or even homographic, model. However, a disadvantage is in the time required by the learning stage. Although the authors do not report computation times, this is likely to be quite slow,
as it involves solving a large linear system (about 100 unknowns and 1000s equations), plus the computation of a number of image differences equal to the number of perturbations, which should be high (1000s) in order to fairly populate the space of expected geometric transformations. Hence, unlike in our approach, tracking does not start right away, but only after an important delay which can be an impediment in some applications (e.g. Augmented Reality).

### 3.9.2 Augmented Reality

Due to the real-time demands of AR, researchers often have to take refuge by putting ‘markers’ or ‘landmarks’ into the scene [Azuma 1997]. They are designed to be easily detectable. Detection may be eased by giving them distinctive photometric characteristics (e.g. colored markers [Ohshima et al. 1998, Okuma et al. 2000], LEDs [Hoff 1998], and retroreflective markers [Dorfmueller 1999, Ribó et al. 2001]) or special shapes (like squares [Ohshima et al. 1998, Tan et al. 2000], circles [Sundareswaran and Behringer 1998], dot codes [Kustaborder and Sharma 1999] or even 2D bar-codes [Rekimoto and Ayatsuka 2000]). As one novelty, our work attempts to use patches of the natural scene as landmarks. The extraction of these landmarks is automatic and robust against changes in viewpoint and illumination. A few patches can be tracked in real-time, where out-of-plane rotations as well as major displacements between frames are allowed. This allows qualitative planar and 3D augmentations in real-time.

The AR approach of this chapter differs with earlier contributions with natural landmarks in that there is no initialization stage based on fiducial markers [Kanbara et al. 2001] or interactively indicated points needed for calibration [Simon et al. 2000, Lepetit et al. 2003], in that the planarity assumption is limited to the small landmark patch instead of an extended part of the image [Simon et al. 2000], in that no 3D model of the target object is required [Uenohara and Kanade 1995, Lepetit et al. 2003, Comport et al. 2003], in that the 3D camera/object motion is not constrained [Broll et al. 2000, Cheng and Robinson 2000], and in that the computations run in real-time on a single, modest processor (see result section).

In the last two years, thanks also to the advent of ever faster computers, more markerless real-time AR systems have appeared. However, they are nearly all model-based [Lepetit et al. 2003, Comport et al. 2003] and/or require that additional information is provided beforehand or computed in an initialization stage (e.g.: user-selected reference images in [Lepetit et al. 2003, Chia et al. 2002], or an external, typically not markerless tracker determining the 3D structure of the scene in [Genç et al. 2002]). Hence, our approach remains attractive because of its simple and comfortable usage, requiring minimal input. Planar virtual textures can be overlaid by just selecting one region, while full 3D virtual objects by selecting two. In either case, no 3D model of any part of the scene, or any other information, is needed.
3.10 Conclusion and outlook

In the first part of this chapter, we have presented a real-time tracker which recovers the complete affine transformation of a region as it evolves through an image sequence. Hence, it provides three accurate point correspondences per tracked region in every frame, even in the presence of image rotation, out-of-plane rotation and scale changes. The tracker is robust because it is able to withstand large displacements between subsequent frames and can recover from a temporary loss of the target region.

A system for markerless uncalibrated Augmented Reality, based on the tracker, is described in the second part of the chapter. As demonstrated in the result section, the system can accurately superimpose planar and 3D virtual objects onto a user-selected part of a moving scene in real-time. The augmentation moves and deforms as if it were part of the physical scene, mimicking the same 3D motion that the regions undergo.

Most of the algorithms of this chapter have been published in 2001 in [Ferrari et al. 2001b, Ferrari et al. 2001a].

The techniques presented in this chapter could be improved and extended in several ways. Some of the known limitations follow.

Nature of the trackable regions

The tracker is specialized for regions near two straight line segments. The presence of the segments is heavily relied upon by the algorithm and allows to properly decompose the search, which in turn results in efficient and accurate affine tracking. These segments can be short, even shorter than the region sides, as their only task is to convey orientations.

However, requiring segments near c limits the spectrum of trackable regions. Many image areas remain beyond reach, even if they contain significant anisotropic 2D texture (i.e.: offer enough information to be tracked unambiguously [Shi and Tomasi 1994]).

A potential solution would be to use a particularly efficient algorithm to search the affine space along its 6 dimensions simultaneously. One possibility could be the refinement algorithm introduced in the next chapter. However, it is not clear how well it would manage to capture the small and fine-grained deformations between subsequent frames. Besides, that algorithm is computationally more expensive, and we expect any 6D-search algorithm to be relatively slow. The viability of using the refiner as a tracker is nevertheless interesting and could be the topic of further research.

Master crashes in the parallel tracker
Although the parallel tracker is robust to crashes of slave processors, it cannot withstand a failure of the master. The crash of the master would entail the breakdown of the whole system.

A fully robust solution would require a decentralized architecture, with no special processor in charge of coordination.

**Occlusions of virtual objects**

The virtual textures and 3D objects are directly overlaid on top of the scene. The approach copes with geometric and photometric transformations (for textures), but not with occlusions of the virtual object by an element of the real scene. In those cases, the virtual object will be erroneously drawn as foreground.
4

Multiple-view correspondences

4.1 Introduction

In chapter 2, the class of wide-baseline stereo (WBS) matching algorithms has been introduced. These methods are capable of finding correspondences between two images taken from substantially different viewpoints. The key idea is to extract local invariant features independently from the two images, then characterize them by invariant descriptors, which are finally used for matching. The power of these approaches lies in two factors. First, local features bring tolerance to occlusions. Second, the construction process and descriptors are invariant under affine deformations, allowing large viewpoint changes, and under linear photometric transformations, allowing changes in illumination.

While these methods have focused on two views, this chapter presents a method for obtaining wide-baseline matches among a larger set of unordered images. The output is a set of region-tracks, each aggregating the image regions of a certain physical surface patch along several views. Wide-baseline N-view matches are precious for a number of applications. In uncalibrated 3D reconstruction, the 3D model of a scene could be generated from still images [Strecha et al. 2003] rather than video [Pollefeys et al. 1998, Oliensis and Genc 1999, Mahamud and Hebert 2000]. In object recognition, the complementary information provided by a small number of model views could be integrated [Lowe 2001, Rothganger et al. 2003, Ferrari et al. 2004a].

In order to build region-tracks, a natural idea is to apply an established WBS algorithm to view pairs and then somehow integrate the outputs. A straightforward implementation is bound to fail, due to two main problems. The first is structural: WBS algorithms return sets of matches between pairs of views. Region-tracks must be derived from this, even in the presence of matching errors, which can lead to contradictory information about their composition. The second problem is statistical: suppose a physical region is visible in N views, there is no guarantee that the image
region will be extracted in all views. Suppose the probability $p$ of extracting the region is the same, and independent, in every view where it is visible. The region will be extracted in its $n$ views with probability $p^n$. Thus, even assuming a perfect matching stage, the probability of a $N$-view track decays exponentially with $N$, and so does the expected number of $N$-view region-tracks. The inevitable matching failures only worsen the situation.

In this chapter we propose a system that tackles these problems. It builds upon a multi-scale extension of the WBS algorithm proposed in [Tuytelaars and Van Gool 2000]. However, these problems are common to all WBS schemes based on independently extracted features 2.4. In practice, given 3 views $v_1, v_2, v_3$, the method typically finds many matches between view pair $(v_1, v_2)$ and view pair $(v_1, v_3)$, but the two sets of matches will usually differ substantially. Hence, a smaller number of regions, typically half, are matched over the three views. When considering more views, the situation deteriorates further (e.g.: only a few, if any, 5-view matches can be found).

The chapter is organized as follows. The next section gives an overview of the system and its main processing stages, which are explained in sections 4.3 through 4.6. Section 4.7 presents experimental results. An additional issue, not part of the processing chain for obtaining multiple-view correspondences, is presented in section 4.8. It is a preliminary attempt to recover the approximate location of a mismatch, based on a set of correct matches. The outlook section 4.10 concludes the chapter with a summary and ideas for further improvements.

### 4.2 Scheme of the system

The system produces multiple-view correspondences starting solely from an unordered set of images. The processing goes through five stages (figure 4.1):

**Pairwise WBS matching**

Since we have no *a priori* ordering on the input image set, the first step is to apply a traditional WBS matching algorithm to all pairs of views. We use the method of Tuytelaars and Van Gool [Tuytelaars and Van-Gool 2000], but any other WBS algorithm based on affine invariant regions will do.

This may seem computationally expensive, but the application is meant for wide baseline conditions, in which case there will only be a limited number of views (typically 5 to 20). Otherwise, other and more appropriate approaches than WBS could be applied. Besides, this allows to use all initial matches, which in turn reflects on the quality of the final results.

This stage outputs separate sets of pairwise matches.
4.2. Scheme of the system

Figure 4.1: Scheme of the multiple-view correspondences system.

**Build region-tracks**

An initial set of clean, disjoint region-tracks is constructed from the large set of all pairwise matches, by the algorithm of section 4.3. This process takes the inevitable presence of matching errors explicitly into account, and solves the structural problem pointed out in the previous section. Note that the number of region-tracks is now finalized. All subsequent processing stages will modify existing region-tracks, by updating, adding or removing the regions composing them.

**Refinement**

The geometric registration between the regions within each region-track is not optimal, because they are extracted independently from each view. In section 4.4, this is improved by a novel way of refining the affine transformations between regions. The higher quality registration better supports the next processing stages.

**Propagation**

Most of the current region-tracks are not complete. A track typically does not cover all views where its physical region is visible yet.

In the propagation stage, unmatched regions are propagated to other views by exploiting the geometric and photometric transformation of nearby matches. Hence, the region-tracks are extended to previously uncovered views (section 4.5).

This step is vital for escaping from the aforementioned statistical trap (section 4.1).
4.3 Building region-tracks

4.3.1 Problem definition

Initially, regions are extracted in all views and the traditional two-view matching algorithm is applied between all pairs of views \((V_i, V_j), i \neq j\). This yields separate sets of pairwise matches, in the form \((A_l, B_m)\), indicating that region \(A\) in view \(V_l\) has been matched to region \(B\) in view \(V_m\).

In order to fulfill the main goal, we need to integrate this information into region-tracks: each aggregating the image regions of a certain physical surface patch along the views. Such a region-track should be in the form \(R = \{A_l, B_m, \ldots, J_z\}\), indicating that regions \(A_l, B_m, \ldots, J_z\) are all in mutual correspondence, and represent a single physical region \(R\) as it is imaged in views \(l, m, \ldots, z\). The region-tracks should be mutually disjoint, i.e.: no two region-tracks should share a common region.

How to get region-tracks out of pairwise matches? Let's consider the graph \(G\) where vertices represent regions and edges are weighted by the following similarity measure between two regions \(A, B\):

\[
\overline{\text{sim}}(A, B) = \text{NCC}(A, B) + (1 - \frac{\text{dRGB}(A, B)}{100})
\]

(4.1)

where NCC is the normalized cross-correlation between the regions' greylevel patterns, dRGB is the average pixel-wise Euclidean distance in RGB color-space after independent normalization of the 3 colorbands (necessary to achieve photometric invariance). \(R, G, B\) range in \([0, 255]\), and \(\text{sim}(A, B)\) is within \([-4.41, 2]\), with 1.0 indicating good similarity. Before computation, the two regions are aligned by the affine transformation mapping \(A\) to \(B\). As observed in various experiments, this mixed measure is more discriminative than NCC alone, which is the most common choice (for example [Tuytelaars and Van-Gool 2000, Mikolajczyk and Schmid 2002, Matas and Obrdzalek 2002]). NCC mostly looks at the structure of the patterns, and
discards valuable color information. For example, a green disc on a red background, and a bright blue disc on a dark blue background would be declared very similar by NCC. The dRGB measure captures complementary properties: it focuses on the color correspondence of pixels but largely ignores the structure of the pattern. It would correctly score low in the previous disc example. However, it would confuse a green disc on a bright green background with a green cross on a bright green background, a difference which would be correctly spotted by NCC. Therefore, by summing these two measures we obtain a score which is good only when both are good, and manage to overcome their (complementary) failure modes.

No edge of $G$ is present between two unmatched regions. Suppose for a moment that the region extraction and two-views matching processes were perfect, so that there are no mismatches and no missing matches. In this ideal case, $G$ is composed of completely connected, disjoint subsets of vertices (cliques). Since each clique corresponds to a region-track, our task is easily solved (figure 4.2a).

Unfortunately real data is plagued by two kinds of errors: mismatches, which insert spurious edges into $G$, and missing matches, which let $G$ lack some correct edges (figure 4.2b). With these errors, $G$ is no longer in a disjoint-cliques form and ambiguities about the composition of the region-tracks arise. This is not structurally acceptable: we want the region tracks to be disjoint and all regions within a track to mutually match each other, so $G$ must be in a disjoint-cliques form.

### 4.3.2 Conflict resolution

Three properties of the desired solution come to help.

- **Transitivity**: matching is transitive: if $(A_i, B_m)$ and $(B_m, C_n)$ are matches then $(A_i, C_n)$ must be a match as well.
- **One-to-one constraint**: a region cannot be matched to two different regions which are in the same view.
- **Rank of similarity**: given the correct match $(A_i, B_m)$ and the mismatch $(A_i, C_m)$, the similarity measure is, almost always, higher for the former.

The rest of this section describes a conflict resolution algorithm (CR) that exploits these properties to bring $G$ into a disjoint-cliques form. CR reconstructs missing edges and discovers spurious edges by the same basic process of **triangulation**: given any two edges $(A_i, B_m)$ and $(B_m, C_n)$, the edge $(A_i, C_n)$ is added to $G$ if not yet present. The two edges generating the new one are its parents. An original edge has no parent (leaf). Two edges $(A_i, B_m)$ and $(A_i, C_m)$ are in **conflict**, as they violate
the one-to-one-constraint. At least one edge in a conflict is caused by a mismatch. Adding an edge can generate a conflict, which is resolved by removing the least weighted edge (i.e.: the match with the lower similarity measure). When an edge is removed also the weakest of its parents is removed, causing a recursive process which backtracks until a leaf is removed. This is justified by the fact that, if an edge is considered a mismatch, then at least one of its parents must be regarded as a mismatch.

CR starts by triangulating the match with highest similarity. This new match is used to triangulate further. CR recurses until no new edge can be constructed with the latest generated one as a parent, or if this is removed as consequence of a conflict. At this point, the match with next highest similarity is triangulated, and so on. Cycles are avoided by not allowing the triangulation of a previously removed edge, thus ensuring CR's convergence. The algorithm terminates when no new edge can be triangulated. At this point $G$ is composed of disjoint cliques.

**Figure 4.2:** a) ideal case. b) real case, dashed edges are mismatches. c) and d) two steps of CR. Thick edges are in conflict.
Figure 4.2 shows an interesting case. Figure 4.2a shows the ideal situation, without mismatches or missing matches. The input given to CR is instead the one of figure 4.2b. First, \((B_2, D_4)\) is triangulated from \((A_1, B_2)\) and \((A_1, D_4)\). This in turn generates \((B_2, C_3)\) (from \((C_3, D_4)\) and \((B_2, D_4)\)). Now \((B_2, C_3)\) conflicts with \((F_2, C_3)\), and the latter is removed as it has lower similarity measure (mismatch, figure 4.2c). Since \((B_2, C_3)\) survived, it combines with \((A_1, B_2)\) to give \((A_1, C_3)\). No triangles can be further constructed based on \((A_1, C_3)\), so original edges are processed. \((F_2, I_4)\) is triangulated from \((E_1, F_2)\) and \((E_1, I_4)\), which then induces \((F_2, H_3)\). A conflict between \((F_2, H_3)\) and \((G_2, H_3)\) is detected and causes the removal of \((F_2, H_3)\) (figure 4.2d). This calls upon the removal of its weakest parent \((F_2, I_4)\), which in turn causes the removal of grandparent \((E_1, I_4)\). Finally, the last possible edge \((G_2, I_4)\) is added and the algorithm terminates on the ideal solution (figure 4.2a).

The example of figure 4.2 shows how CR discovers and solves ambiguities in a conflictual set of matches. The conflicts are often hidden within the dataset, but are triggered by recursive triangulation, and then disambiguated by maximal similarity. CR is guaranteed to find all conflicts, although it is not sure to always solve them correctly: in a few cases the similarity of the mismatch might exceed that of the correct match. Nevertheless, CR's goal is not to find the optimal solution, but instead to efficiently yield a reasonably good, and structurally acceptable, starting point for the further processing stages.

The proposed approach cleans the initial set of matches, yielding a coherent, conflict-free one, respecting the transitivity of matching and the one-to-one constraint. Hence region-tracks are extracted out of a large inconsistent set of pairwise matches (several thousands for 10 views). CR is time-efficient, as it evaluates the similarity measure only when needed, and cautious in that it has no fixed similarity threshold for rejecting a match, but instead only compares similarities, relying on the weaker assumption that a correct match scores higher than a wrong match. This is important because unnecessarily removing matches at the initial stage could compromise the performance of the later processing stages.

The following definitions are used in the rest of the chapter. \(\Gamma\) is the set of all region-tracks. \(R = \{ R_v \}_{v \in vs}\) is a region-track, composed by image regions in views \(vs\). If \(R_v \in R\) we say that region-track \(R\) is present in view \(v\). \(\Phi_{vs} = \{ R \in \Gamma | R_v \in R, \forall v \in vs \}\) is the set of region-tracks present simultaneously in each view \(v \in vs\).
4.4 Refinement

4.4.1 Motivation

Consider two regions $R_1, R_2$ matched between two views. In practical situations $R_1$ is not perfectly registered with $R_2$ as they are independently extracted from the two views. In other words, the two regions often do not cover exactly the same physical surface.

It is desirable to refine this initial match, so as to obtain a more accurate correspondence and to provide better input for the next processing stage (section 4.5).

Formally, $R_2$ should be affinely transformed such that the resulting region maximizes the similarity (4.1) with $R_1$. In this section, we introduce a refiner algorithm capable of searching for (an approximation of) the maximum in a reasonably large range of the 6D affine space, while evaluating the similarity function as few times as possible (computational cost). Besides its importance in the present context, the algorithm will be used also in the propagation phase (section 4.5), and again later for object recognition (chapter 5).

4.4.2 The Refiner algorithm

The affine space is decomposed into the translation $(t_x, t_y)$, scale $(s_x, s_y)$, rotation $(\theta)$ and shear $(h)$ components. We consider searching an hypercuboid $\Omega$ bounded in all dimensions by predefined values $^1$. A point in $\Omega$ is denoted $A = (t_x, t_y, s_x, s_y, \theta, h)$. Let $R_2^c$ be obtained by centering $R_2$ on $(0,0)$, and $c$ the center of $R_2$. The algorithm starts from the identity transformation $A = A_0 = (0, 0, 1, 1, 0, 0)$ and searches for $A_{\text{max}}$ so that

$$A_{\text{max}} = \arg \max_{A \in \Omega} \text{sim}(R_1, AR_2^c + c)$$

Let $A(\text{dim}, \text{val})$ be point $A$ with component $\text{dim}$ set to $\text{val}$. Let $i_d, f_d$ be the bounds of $\Omega$ along dimension $d$ and

$$v_{\text{max}}^d = \max_{v \in [i_d, f_d]} \text{sim}(R_1, A(d, v)R_2^c + c)$$  \hspace{1cm} (4.2)

be the value of the highest similarity induced by affine transformations along the straight line segment starting at $A(d, i_d)$, passing through $A$, and ending at $A(d, f_d)$. Computing (4.2) for all dimensions yields 6 values $v_{\text{max}}^d, d \in 1...6$, from which only the absolute maximum $v_{\text{best}}$ is retained. The procedure thus evaluates similarity on 6 line segments concurrent in $A$, which we call rays. Point $A$ is now moved to the affine transformation inducing $v_{\text{best}}$. The process iterates until stability.

$^1$In all experiments the bounds are: $t_x \in [-14, 14], t_y \in [-14, 14], s_x \in [0.6, 1.8], s_y \in [0.6, 1.8], \theta \in [-\frac{\pi}{4}, \frac{\pi}{4}], h \in [-1, 1]$. Discretization: 2 for translations, 0.1 for scales, $\frac{\pi}{16}$ for rotation, 0.2 for shear.
4.4. Refinement

Figure 4.3: Example 2-dimensional non-convex function to be maximized. Both Gradient descent and our algorithm start from (0,0). Gradient Descent reaches point (3,20), while our method iterates through (0,−19), (15,−19) and then reaches the global maximum (15,−15). Notice the numerous foldings in the landscape.

The proposed algorithm (referred to as RF from now on) only searches a predefined, immutable hypercube $\Omega$ centered at $A_0$. It can be seen as a particular way of stepwise walking in $\Omega$, where each step occurs in a direction parallel to a coordinate axis and can be arbitrary large (within $\Omega$).

RF is motivated by the observation that the similarity function (4.1) generates spaces which are smooth, but highly non-convex, with frequent and diverse foldings. In this situation Gradient Descent (GD) cannot be relied on, because it gets stuck in the local maximum closest to $A_0$. RF does not blindly climb the closest steepest hill, but instead extends its sight along 6 rays, up to the boundaries of $\Omega$. At every iteration, it gets another chance to notice a higher hill elsewhere and it will jump on it. The new hill is climbed until its tip, or until a searching ray intersects a higher hill, and so on. The chances of finding the absolute maximum are much higher than for GD and are fairly high in absolute in the kind of spaces we consider, which are highly non-convex, but also modest in size ($R_2$ initially already roughly corresponds to $R_1$).

Figure 4.3 exemplifies RF’s behaviour in 2 dimensions and shows a typical case where it succeeds while GD fails.

In practical situations RF iterates 3 to 10 times. Given the search space boundaries and discretization steps above, this amounts to 100 to 700 evaluations of the similarity measure. Thus, RF is about as fast as GD and is orders of magnitude faster than exhaustive search (3.5 million for the same parameters) or simulated annealing. This makes RF computationally affordable for our purposes, where thousands of refinements are required (about 10 region matches can be refined in a second on our
4.4. Refinement

Figure 4.4: Examples of the refiner’s power. (a) The small bright region has been artificially deformed to the large one by simultaneous translation \((-5, 0)\), rotation \(\Pi/7\), scale 1.4, 1.3 and shear 0.4. The refined region (small dark) comes very close to the correct solution. (b+c) Two matching regions before refinement. (d) The region of c) refined. (e) Another region. (f) The refined region (bright) is much better aligned with e) than before refinement (darker).

2.4 Gzh test computer). As another advantage, no derivative computations are involved, which brings more stability, and avoids the cost of computing 6-dimensional gradients.

The a priori knowledge about the range of possible deformations, embedded in the boundaries of \(\Omega\), allows to design RF. Indeed, RF only works for a bounded search space. Although this is adequate in our case, it might limit RF’s applicability to other domains.

The proposed algorithm bears resemblance to Powell’s method and Conjugate Gradients [Press et al. 2002, pp. 417-431], because the search occurs along straight lines. However, an important structural difference is that RF does not move through a series of successive line maximizations. Instead, in every iteration a whole set of lines (6, all mutually perpendicular, each in the direction of a coordinate axis) is evaluated, and the move occurs to the collective optimum of all lines. This brings
more power at minimal additional computational cost. Besides, unlike Conjugate Gradients, only function evaluations are required, and not gradient computations. Moreover, the scheme is simpler than Powell’s in that the directions are fixed, predefined, and not necessarily 'conjugate' (non-interfering, see [Press et al. 2002]). This might lead to a suboptimal path for simple functions (e.g. quadratic forms), but it increases the probability of spotting higher hills while climbing the current one.

Various examples of the benefits brought by RF are depicted in figure 4.4.

### 4.4.3 Global refinement

The refinement algorithm works on a pair of regions. The following simple approach is used to optimize the global registration of all regions within a track $R$

1. Each region of $R$ is refined towards each other region of $R$. In other words, each $R_i$ is refined towards all other views $j$ where $R$ is present, and all 'versions' stored.

2. A pivot-view is selected as the one maximizing the sum of similarities to the other views.

3. Set all regions of $R$ as refined towards the pivot-view.

Note that it is unlikely that a view where $R$ is mismatched will be selected as pivot. The chances decrease with the number of views where $R$ is present.

This simple approach is independent of the order in which the views are considered and makes all regions within a region-track well globally aligned. A high quality registration is important for the forthcoming propagation stage.

All region-tracks $R \in \Gamma$ produced by the conflict-resolution algorithm (section 4.3) are processed in the manner explained in this subsection.

### 4.5 Propagation

#### 4.5.1 Two-view propagation

As pointed out in section 4.1, many regions of a view $V_1$ do not get matched to another view $V_2$ even though the feature is visible in the image. This might happen because either the corresponding region has not been extracted, or the matching failed.
This section describes an approach for exploiting the information supplied by a correct match in order to generate many other correct matches. Consider a region $C_1$ in $V_1$ without matching region in $V_2$ (candidate region) and a set $\Psi = \{S_1^i\}$ of matched support regions in an area around $C_1$. The key observation is that if $S_1^i$ and $C_1$ lie on the same physical surface (e.g.: a facet of an object), then they will probably be mapped to $V_2$ by similar affine and photometric transformations.

This observation is exploited by the following procedure (illustrated in figure 4.5). For every support $S_1^i \in \Psi$ do:

1. Compute the affine transformation $A_1^i$ mapping $S_1^i$ to $S_2^i$ in view $V_2$.

2. Compute the color transformation $T_{RGB}^i = \{s_R, s_G, s_B\}$ between $S_1^i$ and $S_2^i$. This is composed by the scale factors on the three colorbands. Each can be computed efficiently as ratio of the mean colorband intensity in $S_1^i$ over the mean intensity, of the same colorband, in $S_2^i$.

3. Project $C_1$ to view $V_2$ via $A^i : C_1^i = A_1^i C_1$. We refer to $C_1^i$ as a propagation attempt.

4. Evaluate the similarity between $C_1^i$ and $C_1$ after applying the color transformation $T_{RGB}^i$

\[
sim_i = \text{sim}(C_1, C_1^i, T_{RGB}^i) = \text{NCC}(T_{RGB}^i C_1, C_1^i) + (1 - \frac{\text{dRGB}(T_{RGB}^i C_1, C_1^i)}{100})
\]

Applying $T_{RGB}^i$ allows to use the unnormalized color-distance $\text{dRGB}$ on the raw image pattern, because photometric changes (e.g. reduced lighting) are now compensated for. This is an advantage over $\text{dRGB}$ as it provides more discriminative power.\(^3\)

We retain $C_2^{\text{best}}$, with $\text{best} = \arg \max_i \sim_i$, the attempt that best matches $C_1$ and refine it with the algorithm of section 4.4, yielding $C_2^{\text{ref}}$. This refinement step adapts $C_2^{\text{best}}$ to the local surface orientation and counters perspective effects (the affine approximation is only valid locally). Indeed, the candidate and the correct support have only similar, not identical, affine transformations. By this mechanism, the method nicely deals with the common case in which support and candidate lie on a curved surface (figure 4.5).

$C_2^{\text{ref}}$ is considered successfully propagated if $\sim(T_{RGB}^i C_1, C_2^{\text{ref}}) > t_{\text{prop}}$ ($t_{\text{prop}} = 1.0$ in our experiments). Note that refinement tends to raise the similarity of correct propagation attempts much more than the similarity of mispropagated ones. The resulting increase in the separation between the distributions of the similarity values, renders the last thresholding step remarkably more effective after refinement.

\(^2\)In all experiments a circle of radius $\frac{1}{5}$ of the image size (i.e. max(width, height)) has been used.

\(^3\)The function $\text{sim}(A, B, T_{RGB})$ will be used again in the recognition chapter 5.
unmatched candidate

propagation attempt

support match

Figure 4.5: A candidate region is propagated via the affine transformation $A$ of a support match. Notice how refinement adapts the shape to the different surface orientation of the candidate. The combination of propagation attempts and refinements results in a flexible and powerful tool.

The usefulness of exploiting previously established geometric transformations was also noted by [Schaffalitzky and Zisserman 2002b]. However, our approach generates a new region in $V_2$ which might not have been originally extracted. This differs from [Schaffalitzky and Zisserman 2002b], where the geometric transformation is only used to guide the search for further matches. This is important as it helps solving the main problem exposed in section 4.1: the quick drop of the probability that a region is extracted simultaneously in several views. Indeed propagation strongly increases the chances that a region will be put in correspondence, as it suffices that any nearby region undergoing a similar image transformation is correctly matched.

The proposed procedure is robust, since several supports attempt to propagate the candidate, and only a single correct one is required. Any mismatched support in $\Phi$ is unlikely to disturb, as it produces an incorrect $C_2^i$ with a low similarity $sim$. The same holds for inadequate supports, like regions lying on very differently oriented surfaces (e.g.: two different facets of a cube).

### 4.5.2 Global propagation

For every pair of views $l,m,l \neq m$, the propagation algorithm is applied to all candidate regions $\Phi_l \setminus \Phi_m$ which are present in $l$, but not in $m$, using as support
4.6. Topological filter

In the previous sections, the region-tracks have been first formed out of pairwise matches, then their geometric alignment has been enhanced, and finally they have been extended to cover more views. However, the region-tracks might still contain some mismatches. These are due to original mismatches that survived the conflict-resolution stage and occasional mispropagations.

Figure 4.6: Cooperation between two incomplete tracks. a) The situation before propagation. The tracks R, Q are incomplete as they are not present in all views 1, 2, 3. b) Propagation from view 2 to view 1. Track R acts as a support to propagate track Q. c) Q supports the propagation of R to view 3. At this point, both tracks are complete.

the regions $\Phi_{im}$ already matched between the two views. Everytime a region $R_i$ is successfully propagated to view $m$, it is added to region-track $R_i$.

Propagation does not generate new region-tracks, but extends currently existing ones. As a consequence, the region-tracks grow larger, as they are present in more views. The connectedness between any subset of views $\mathcal{v}$, i.e.: the amount $|\Phi_{\mathcal{v}}|$ of region-tracks present simultaneously in each $v \in \mathcal{v}$, is significantly increased (as shown in section 4.7). Note that a single propagation creates several new pairwise matches, as all regions in the track are implicitly assumed matched to the newly added region. These transitive propagations actively contribute to increase the inter-view connectedness.

In order to better understand the large benefits of the strategy, one should think about its collective effects on a large set of incomplete tracks (figure 4.6). Suppose that two physical regions R, Q are both visible through views 1, 2, 3, but R is matched only between views 1, 2, and Q in views 2, 3. None of the tracks is complete. Let’s imagine that both regions lie on the same surface, somewhat near to each other. When propagating from view 2 to view 1, the track R will act as a support to propagate track Q. Later, track Q will help track R extending to view 3. As a total result, after the whole propagation process, both tracks will cover views 1, 2, 3. This example illustrates how the incomplete tracks cooperate, helping each other becoming as complete as possible.

4.6 Topological filter
In this section, we tackle this problem with a method for filtering mismatches based on a topological constraint for triples of region matches.

The word topological is used here in a broad sense, to underline that the method does not rely on numerical distances, but on qualitative arrangements of points instead.

The next subsection introduces the property on which the filter is based, while the following three subsections discuss the filter itself for the two-view case, its properties and advantages, and the filter for the all-view cases respectively.

### 4.6.1 The sidedness constraint

Consider a triple \((R^1_v, R^2_v, R^3_v)\) of regions in a view \(V_1\) and their corresponding regions \((R^2_w, R^3_w, R^3_w)\) in another view \(V_2\). Let \(c^i_v\) be the center of region \(R^i_v\). The function

\[
\text{side}(R^1_v, R^2_v, R^3_v) = \text{sign}((c^2_v \times c^3_v)c^1_v)
\]

(4.3)

takes value \(-1\) if \(c^1_v\) is on the right side of the directed line \(c^2_v \times c^3_v\), going from \(c^2_v\) to \(c^3_v\), or value \(1\) if it’s on the left side.

The equation

\[
\text{side}(R^1_v, R^2_v, R^3_v) = \text{side}(R^2_w, R^3_w, R^3_w)
\]

(4.4)

states that \(c^1\) should be on the same side of the line in both views (figure 4.7).

The sidedness constraint (4.4) holds for all correctly matched triples of coplanar regions, because in this case property (4.3) is viewpoint invariant. The constraint is valid also for most non-coplanar triples. A triple for which equation (4.4) does not hold is said to violate the constraint. A violation is due to any of two cases. First, at least one of the three regions is mismatched. Second, the regions are not coplanar and there is important camera translation in the direction perpendicular to the 3D plane containing their centers (parallax-violation). This can create a parallax effect strong enough to move \(c^1\) to the other side of the line. Nevertheless, this phenomenon typically affects only a small minority of triples. Since the camera can only translate in one direction between two views, the resulting parallax can only corrupt few triples, because those on planes oriented differently will not be affected.

The region matches violate or respect equation (4.4) independently of the order in which they appear in the triple. The three points should be cyclically ordered in the same orientation (clockwise or anti-clockwise) in the two images in order to satisfy (4.4).

The benefits of analyzing topological configurations of points and lines was already noted by Carlsson in 1996 [Carlsson 1996]. They are used in the wide-baseline stereo context in 2002 by Tell and Carlsson [Tell and Carlsson 2002], as a means for guiding the matching process.
4.6. Topological filter

4.6.2 Two-view mismatches filter

A triple including mismatched regions has higher chances to violate the sidedness constraint. When this happens we can only conclude that probably at least one of the matches in the triple is incorrect, but we do not yet know which. While one triple is not enough to decide, this information can be recovered by considering all triples simultaneously. By integrating the weak information each triple provides, it is possible to robustly discover mismatches. The key idea is that we expect incorrectly located regions to be involved in a higher share of violations.

The constraint is checked for all unordered triples \((R_i, R_j, R_k), R_i, R_j, R_k \in \Phi_{12}\) (recall that \(\Phi_{12}\) is the set of all regions present in both \(V_1, V_2\)). The share of violations for a region \(R_i \in \Phi_{12}\) is:

\[
\text{err}_{\text{topo}}(R_i) = \frac{1}{v} \sum_{R_j, R_k \in \Phi_{12}\setminus R_i, j > k} |\text{side}(R_i, R_j, R_k) - \text{side}(R_i, R_j, R_k)|
\]

(4.5)

Where \(v = (n-1)(n-2)/2\), with \(n = |\Phi_{12}|\), is the maximum number of violations any region can be involved in. Hence, \(\text{err}_{\text{topo}}(R_i) \in [0,1]\) because it is normalized by \(v\).

The filtering algorithm starts from the set of matches \(\Phi_{12}\), and then iteratively removes one match at a time as follows:

1. (Re-)compute \(\text{err}_{\text{topo}}(R_i)\) for all \(R_i \in \Phi_{12}\).
2. Find the worst match \(R_w\), with \(w = \arg \max \text{err}_{\text{topo}}(R_i)\)
3. If \(\text{err}_{\text{topo}}(R_w) > 0.15\), \(R_w\) is removed. \(R_w\) will not be used for the computation of \(\text{err}_{\text{topo}}\) in the next iteration. Iterate to 1.
   If \(\text{err}_{\text{topo}}(R_w) \leq 0.15\), or if all regions have been removed, then stop.
At each iteration, the most probable mismatch $R^w$ is removed, and $\text{err}_{\text{topo}}$ is recomputed at the next iteration only based on the remaining regions. During the first iterations several mismatches are still present. Therefore, even correct matches might have a moderately large error, because they take part in triples including mismatches. However, the mismatched regions will have an even larger error, because they will be involved in the very same triples, plus many other violating ones. Hence, the worst mismatch $R^w$, the region which is located in $V_2$ farthest from where it should be, is expected to have the largest error. After removing $R^w$ all errors decrease, including the errors of correct matches, because they are involved in less triples containing a mismatch. Again, the match with the largest error is very probably a mismatch, and it is removed. After several iterations, ideally only correct matches are left. Since these are involved in only a few violations, due to occasional parallax-violations, the algorithm stops.

Various experiments on artificial and real configurations confirmed these theoretical considerations. In section 4.7 the algorithm is shown to be very robust, by being capable of telling correct matches apart in sets with up to 75% mismatches.

When all matches are uniformly random distributed over the two images, $\text{err}_{\text{topo}}$ has expected value 0.5 for all. This, together with experimental measurements on mismatch-free configurations, helped us selecting the removal threshold 0.15.

In order to achieve good computational performance, it is wise to store the terms of the sum in function (4.5) during the first iteration. In the following iterations, the sum can be quickly recomputed by retrieving and adding up the necessary terms. This makes the computational cost almost independent of the number of iterations.

At first glance, the method seems to have complexity $O(n^3)$, because the number of triples grows with the cube of the number of region matches. Nevertheless, it can be implemented to run in $O(n^2 \log(n))$ by constructing, for each point, a list with a cyclic ordering of all other points. This faster algorithm is described in appendix D.

### 4.6.3 Properties and advantages

The proposed filter has a number of attractive properties, and offers several advantages over detecting outliers to the epipolar geometry through RANSAC [Fischler and Bolles 1981], which is traditionally used in the matching literature [Tuytelaars and Van-Gool 2000, Mikolajczyk and Schmid 2002, Matas et al. 2002a, Schaffalitzky and Zisserman 2002b, Schaffalitzky and Zisserman 2002a]. In the following, we refer to it as RANSAC-EG.

- *It allows for non-rigid deformations.* Although this feature is not necessary in this chapter, where we consider multiple viewpoints of a static scene, it will come out to be precious in chapter 5, for recognizing deformable objects.
4.6. **Topological filter**

- *It has only one, intuitive parameter.* It is easy to predict the effects of the removal threshold, because it has a clear meaning. On the other hand, RANSAC-EG requires the user to set two parameters, both causing problems. None of them affects our approach.

The first is the width of the band around the epipolar line, used as acceptance criteria. Only if a point lies within the band it is considered to be an inlier. When there is a large scale change (more than 2) between the two images, this causes trouble, as the localization errors gets magnified. The problem cannot be solved by merely increasing the bandwidth, as this implies accepting more outliers in cases with small scale changes. Thus, this parameter is a form of scale-selection, which is unfortunately unknown a priori (this drawback of RANSAC is also noted by [Chen and Meer 2003]).

As a second parameter, RANSAC-EG requires the maximal number of iterations (section 2.5). This is equivalent to providing the maximal ratio of mismatches, which is again unknown a priori, and varies from case to case. Moreover, the number of iterations required to have a good chance of success grows very quickly with the percentage of mismatches. More precisely, it is proportional to \(1/(1 - m)^7\), where \(m\) is the percentage of mismatches, therefore putting a clear limit on the range of solvable cases (as also pointed out by [Chum et al. 2003]).

- *It is less sensitive to inaccurate locations.* The regions’ centers need not be exactly localized, because \(\text{err}_\text{topo}\) varies smoothly for a region departing from its ideal location. As a result, the algorithm performance is not affected by perturbations of the region’s locations. This feature is particularly precious in the presence of large scale changes, or not completely planar regions, where the localization errors become more important.

In RANSAC-EG the point must lie within a band around the epipolar line. The width of the band is a user parameter, and is usually set tight, to avoid accepting too many outliers. Worse yet, inaccurate localization of some regions might compromise the quality of the fundamental matrix, and therefore even cause rejection of many accurate regions [Zhang et al. 1995].

In the result section 4.7 an experiment is shown comparing the performance of our approach and RASANC-EG as a function of localization inaccuracy.

- *It is more discriminative for nearly planar scenes.* Nearly planar scenes are a degenerate case for fitting epipolar geometry. The fundamental matrix has too many degrees of freedom which causes overfitting of the data. The epipolar lines tend to go in ‘random’ directions, therefore allowing malicious mismatches to slip through the filter. This problem could be tackled by including a model-selection stage (e.g. [Torr 1998]), but only at the price of a more complex algorithm and additional parameters to set.
Our filter instead is completely unaffected by planar scenes. Actually, it even works better than in the general 3D case, because the possibility of parallax-violations is eliminated.

Some of these advantages are not crucial for the application described in this chapter. Indeed, the results of section 4.7 can be achieved also by replacing this step with a standard RANSAC-EG filter, at the price of some loss in quality. However, the topological filter will prove very important in the next chapter, where deformable objects and large scale changes will be an issue.

### 4.6.4 Global mismatches filtering

The topological filter is applied to all pairs of views \(I, m, l \leq m\) yielding a set \(\{(R_i, R_m)\}\) of mismatches per view-pair. A single mismatch still does not tell which one of \(R_i\) or \(R_m\) is wrong. However, this information can be inferred from the set of mismatches related to a single region-track. Removing either \(R_i\) or \(R_m\) suffices to eliminate the mismatch. It is desirable to remove only one, to avoid over-impoverishing the track. The following algorithm removes the minimum number of regions from each region-track, so that all mismatches found by the all applications of the filter are eliminated.

For each region-track \(R\) do:

1. Collect all mismatches pairs related to \(R\).

2. For each view \(t\) in which \(R\) is present, count the number of times \(R_t\) is part of a mismatch.

3. Remove from the region-track the view \(w\) with the highest count. This action automatically eliminates all mismatches involving view \(w\) from the list of point 1. The track is no longer present in view \(w\).

4. Iterate to point 2. Stop when the list of mismatches is empty.

This approach eliminates all detected mismatches, while minimally reducing the region-tracks, thus enhancing quality without sacrificing inter-view connectedness.

4The filter is symmetric in the views.
4.7 Results

The methods and algorithms proposed in the chapter are experimentally validated in this section. The main part (next subsection) presents results on the computation of multiple-view correspondences from unordered image sets, while subsection 4.7.2 investigates the degree of robustness to mismatches and localization inaccuracy of the topological filter.

4.7.1 Multiple-view correspondences

The multiple-view correspondence scheme has been tested on several sets of images (scenes). We report here results for two scenes, both processed with the same set of parameters reported in the chapter.

Valbonne

In the first example 10 images\footnote{The Valbonne is a standard test set, and the 10 images correspond to ids 1,2,3,5,9,11,12,13,14,15 of the Oxford distribution: \url{www.robots.ox.ac.uk/~vgg/data/valbonne}} from the Valbonne scene (figure 4.8), were processed. This image set poses several challenges, like significant viewpoint differences and uniform colors and textures.

Table 4.1 shows the number of region-tracks present in various subsets of views (‘final’ column). Note the high entries for 4 views and more, indicating strong connectedness between views (e.g.: about 200 4-view matches). The statistical problem stated in section 4.1, namely the rapid decrease of the chance of obtaining a \( N \)-view track, with increasing \( N \), has been remedied: the number of \( N \)-view matches gracefully decreases (close to linearly) with increasing \( N \) and corresponds well with failing visibility. The ‘CR’ column reports the number of \( N \)-view matches just after forming the tracks, via the algorithm proposed in section 4.3. At this stage, the tracks are still vulnerable, and their number decays exponentially (e.g.: 51 2-view matches in (9,11) roughly half in (9,11,12) and only 3 in (3,5,9,11,12,13)). However, the first stage’s main goal is to solve the structural problem, by formatting the data in region-tracks. It is the combined effect of refinement, propagation and topological filtering that then counters the statistical problem, as demonstrated in the ‘final’ column of the table. The increase in the number of \( N \)-view matches is strong, and many of them now exist even when none could be found in the original data (from 9 views on). The number of \( N \)-view matches now decreases roughly linearly with \( N \). The values in table 4.1 compare favorably against the ones reported in [Schaffalitzky and Zisserman 2002b], corroborating the effectiveness of our approach (further discussion about this work can be found in section 4.9).
4.7. Results

Figure 4.8: Valbonne. 230 3-view matches through views 9, 12, 14.

Table 4.1: Number of tracks for Valbonne.

<table>
<thead>
<tr>
<th>views</th>
<th>final</th>
<th>CR</th>
<th>views</th>
<th>final</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 11</td>
<td>340</td>
<td>51</td>
<td>all 10 views</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>9 11 12</td>
<td>295</td>
<td>26</td>
<td>9 12 14</td>
<td>230</td>
<td>13</td>
</tr>
<tr>
<td>5 9 11 12</td>
<td>204</td>
<td>9</td>
<td>3 9</td>
<td>273</td>
<td>39</td>
</tr>
<tr>
<td>3 5 9 11 12 13</td>
<td>153</td>
<td>3</td>
<td>3 5 9</td>
<td>254</td>
<td>21</td>
</tr>
<tr>
<td>3 5 9 11 12 13 14</td>
<td>100</td>
<td>2</td>
<td>3 5 9 11</td>
<td>208</td>
<td>7</td>
</tr>
<tr>
<td>2 3 5 9 11 12 13 14</td>
<td>69</td>
<td>1</td>
<td>3 5 9 11 15</td>
<td>132</td>
<td>5</td>
</tr>
<tr>
<td>1 2 3 5 9 11 12 13 14</td>
<td>29</td>
<td>0</td>
<td>2 3 5 9 11 15</td>
<td>124</td>
<td>3</td>
</tr>
</tbody>
</table>

The correctness of the tracks $\Phi_{vs}$ present simultaneously in a set of views $vs$ is evaluated as $1 - \frac{errors}{|\Phi_{vs}|(|vs|-1)}$, where $errors$ is the total number of incorrectly located regions, computed over all tracks and views. The errors have been determined by manual inspection of the results. This measure ranges from 0 (all mismatches) to 1 (perfect tracks), and takes into account that a track can be partially correct, if only some of its regions are erroneously located. Figure 4.8 shows 230 tracks (96% correct), distributed over the whole church, in views (9, 12, 14).

Birthday

The Birthday scene features a more complex geometry than Valbonne and more diverse textures. Eight very different viewpoints serve as input (ids 1 to 8). Figure 4.9 shows 135 tracks (95% correct) on 3 views. Two of the views are taken from almost opposite directions (7, 8). Nevertheless, the matches cover the commonly visible parts of the scene well. The telephone, and the picture of the girl above it, are affected by strong out-of-plane rotation, but still have several correct matches.
region-tracks (98% correct) are present in another four images (figure 4.10). Note the quality of the tracks on the mousepad, which undergo strong image scaling and rotation (views 2, 6), and the lack of 4-view matches on the book. This is correct, as it is not visible in view 4.

Robustness to scale changes is demonstrated in figure 4.11, where viewpoint 5 is significantly closer to the scene than 4. Nevertheless, the two cameras see largely the same part of the scene, and the system produces 178 tracks (97% correct), densely covering the images.

Table 4.2 summarizes the large increase in N-view matches brought by the method. The number of N-view matches decreases close to linearly with N. This is particularly significant in this scene, where the parts visible in all N-views quickly diminish with N.

The accuracy of the registration within a region-track is exemplified in figure 4.12. The ellipse's shape and orientation cover the same physical surface in all views, ac-
4.7. Results

Figure 4.10: Birthday. 124 tracks through views 1, 2, 4, 6. Top: views 1, 2. Bottom: views 4, 6.

Figure 4.11: Birthday. 178 tracks present in views 4, 5. Left: view 4. Right: view 5.
4.7. Results

Figure 4.12: Close-up on 5 views of a Birthday region-track. Note the good accuracy of the registration, in spite of widely varying viewing conditions (mostly viewpoint, but also scale and illumination). As a reference point, observe the part where the ellipse touches the cherry.

Table 4.2: Number of tracks for Birthday.

<table>
<thead>
<tr>
<th>views final CR</th>
<th>views final CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 6 241 48 all 8 views 46 0</td>
<td></td>
</tr>
<tr>
<td>1 4 6 185 14 2 8 170 25</td>
<td></td>
</tr>
<tr>
<td>1 2 4 6 124 4 2 3 8 161 16</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 6 108 2 2 3 7 8 120 3</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 6 8 81 1 2 3 4 7 8 93 1</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 6 7 8 79 0 1 2 3 4 7 8 81 0</td>
<td></td>
</tr>
</tbody>
</table>

accurately deforming to fit the viewpoints. This entails rotation, skew and anisotropic scale changes, well covering the affine spectrum. Figure 4.13 depicts another interesting case, where a region gradually rotates over almost 180 degrees out-of-plane during 5 views. These come from a scene featuring a 19 views tour around a cereal box. In total, the region has a 8-view track. The good alignment is made possible, and computationally affordable, by the new optimization technique presented in section 4.4.

4.7.2 Topological filter

Among the properties of a mismatch filter, two are of particular interest for the application of multiple-view correspondence search. The first is the capacity of telling correct matches apart in sets containing many mismatches. The second is the sensitivity to inaccurate localization of the regions. A good filter should not reject
4.7. Results

Figure 4.13: Close-up on 5 views from a cereal-box track, spanning almost the whole 180 degrees of out-of-plane rotation.

a match just because it is a few pixels off the ideal location. Moderately inexact localization (about 2 to 4 pixels) is common, and it can become serious (5 to 20 pixels) in the presence of large scale changes, not completely planar regions (e.g.: the regions on the Birthday hat), or with significant image degradation (motion blur, etc.). The following paragraphs present an experimental assessment of these two issues for the topological filter of section 4.6. Both tests are performed with the same parameters. Note that another important feature of the filter, the capacity to handle non-rigid deformations, will be discussed in the next chapter, in the context of deformable object recognition.

Robustness to mismatches

The robustness to mismatches has been evaluated on view pair (1, 2) of Birthday. We manually selected 100 correct matches spread over the whole scene (figure 4.14). The test was carried out as follows. Successive sets with progressively more mismatches were generated by moving increasingly more regions of view 2 to uniformly spread random locations. In the first step, no regions were moved, in the second 5, in the third 10, and so on until all 100 regions are mismatched in the last step. The percentage of mismatches is later referred to as noise-level. We applied the topological filter at every step, and measured the percentage of correct matches that were rejected (false-negatives) and mismatches that were accepted (false-positives).

As can be seen from figure 4.15a, the filter performs well in false-positives terms and accept a constant rate of approximately 10% mismatches, almost independently of the overall percentage of mismatches. Notice that the percentage is constant, not the absolute number, which grows roughly linearly.

Figure 4.15b shows the behavior of the filter with regard to false-negatives. It works perfectly until 75% noise-level, as it does not remove any correct match. After this point, the performance does not gently degrade, but it sharply breaks down instead (with 90% mismatches, 80% of the correct matches are lost).
4.7. Results

Figure 4.14: 100 matches between views 1 (left) and 2 (right) of Birthday serve as a basis for testing the robustness to mismatches of the topological filter.

<table>
<thead>
<tr>
<th>false-positives</th>
<th>false-negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentage of mismatches</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.15: Performance of the topological filter. a) false-positives as a function of the noise-level. b) false-negatives.

The presented experiment suggests that our topological filter is robust, as it can reliably operate in the presence of considerably high amounts of mismatches (about up to 75%).

In order to contain the necessary space, we report systematic evaluation on only one representative pair of images, but similar effects were observed on several other scenes.

Sensitivity to localization inaccuracy

Here we additionally compare the performance of our topological filter to RANSAC-EG, which is the traditional algorithm used for filtering mismatches. The same 100 regions in birthday’s views (1, 2) are used again to evaluate sensitivity to localization inaccuracy. Every region’s location in view 2 was perturbed by randomly translating it within a circular neighborhood of increasing radius, from 0 pixels up to 42, in steps of 3 pixels (figure 4.16a). The radius is referred to as perturbation-radius from now on. Unlike in the previous test, here all regions’ locations are altered in every step,
but the magnitude of the disturbance grows with every step. Both the topological filter and RANSAC-EG were applied at every step, and false-negatives measured (there are no false-positives, as all regions are ideally correct matches).

The topological filter withstood the test very well, by correctly keeping all matches until the perturbation-radius reached 18 pixels. (figure 4.16b). Moreover, only 4% false-negatives occur with the perturbation-radius as large as 30 pixels. In contrast, RANSAC-EG performs much worse, thereby confirming the expected sensitivity to exact localization. Already with 9 pixels perturbation-radius, 12% correct matches are rejected. The results degrade roughly linearly in the further steps, and half of all matches are lost at 30 pixels perturbation-radius (10 times worse than the topological filter). In this test, RANSAC-EG ceases to be useful starting from perturbations of up to 12 pixels, where 20% of correct matches are incorrectly removed.

The reasons for such diverse behavior of the two filters are to be sought in the profoundly different nature of the information they exploit. In the topological filter, the region’s center coordinates are only used to measure sidedness relationships. Hence, the number of sidedness constraints a region violates varies slowly and smoothly when it departs from its ideal location. As a result, the algorithm is not disturbed by moderate perturbations of the regions’ locations, and its performance degenerates gently for larger ones.

In RANSAC-EG instead, the point must lie within a band around the epipolar line in order to be accepted. The width of the band is a user parameter, and it is usually set tight, to avoid accepting too many outliers (we have set a relatively wide value for this experiment. Enlarging it further would worsen the false-positive results in the presence of real mismatches). Thus, there is an abrupt cutoff effect: a region is rejected as soon as it is located outside the band. Worse yet, inaccurate localization of some regions might compromise the quality of the fundamental matrix, and therefore even cause rejection of many accurate regions.

4.8 Relocation*

In this section we consider another application of the sidedness constraint. Note that this is not part of the processing chain for obtaining multiple-view correspondences, but only an additional interesting issue.

Suppose we are given a set of matches between two views, and we know one of them is wrong. Given the location of the mismatch (of a query point in general) in the first view, we would like to know where it should go in the second view.

Unfortunately the problem cannot be solved exactly. The best we can do is constrain the query point to lie on a straight line. Indeed, if at least 7 correct matches are
4.8. Relocation

Given, the fundamental matrix can be computed, and used to obtain the epipolar line corresponding to the query point.

Nevertheless, in the next subsection we show how to exploit the sidedness constraints imposed by the correct matches, in order to get the approximate location of the query point in the second view. The location is approximate in two ways. First, it comes in the form of a convex polygon, in which the point should lie. Secondly, depending on the scene structure, the query point and the camera viewpoints, the accuracy might be perturbed by parallax-violations.

In this fashion, it is possible to relocate a mismatched region (point) into a convex polygon. As this is usually quite small, the point is better localized than on an epipolar line. Moreover, by considering the intersection between the epipolar line and the polygon, we could localize the point up to a (short) line segment.

Approximate relocation can be used as initialization for re-matching: the search for a region's correspondence can be reduced to a small portion of the image, thereby increasing the chances of success. An attractive application are scenes containing some repeated pattern, which are typically hard to match by local descriptors only. In this scenario, the reliable part of the scene would provide some correct matches, which could then be used as a basis for relocating the repeated parts. Re-matching would then proceed within the relocation polygons.

4.8.1 The relocation algorithm

The relocation algorithm takes as input a set of region matches between two views, and a query region in the first view only. In a first preprocessing step, the region

Figure 4.16: a) 20 of the 100 basis regions (bright), and their perturbed version translated within a circle of radius 15 pixels (dark). b) false-negatives as a function of the perturbation-radius.
4.8. Relocation*

Figure 4.17: a+b) Query mismatched region in the first and second views, with query point (center; the region is on the telephone in the left view, and on the top-left corner in the right view). c+d) Constraining lines. Each passes through the centers of a pair of reference regions.

matches are passed through the topological filter of section 4.6, with the termination threshold set to 0. Hence, the resulting matches are fully topologically consistent, in that they do not entail any violation of the sidedness constraint. In the following, these will be named reference matches, and the center of the query region will be referred to as the query point $q_1$.

The goal is to find the locus of points in the second view where the correspondence of the query point can lie, in order to respect all sidedness constraints imposed by the reference matches. Since each constraint determines on which side of a line the point can be, it limits the solution within a half-plane. Thus, the locus of points respecting all constraints is the intersection of a set of half-planes, which is a convex polygon [Glicksman 2001].

We explain the algorithm on the example of figure 4.17. Among the 25 initial matches, we selected one mismatch as query (figure 4.17a+b). Since the region is
very large in the first view, and covers a non-planar part of the scene, it is mismatched to a whitish, locally similar region at the top of the second view. Applying the topological filter, with termination threshold set to 0, selects \( r = 9 \) reference matches. Given this input, the algorithm proceeds as follows:

1. **Find the constraints.** Compute all sidedness constraints imposed by the reference matches. There is one constraint for every unordered pair of matches. The constraint is formed by a directed straight line \( l \) and a binary sidedness \( s \) ('to the left' or 'to the right' of the directed line). The line \( l \) goes from the center of the first region to the center of the second region. The side of \( q_1 \) w.r.t to \( l \) gives \( s = \text{sign}(l q_1) \). Note that swapping the order of the reference matches in a pair just leads to a superfluous, equivalent constraint (both \( l \) and \( s \) are multiplied by \(-1\)). Therefore, there are \( r(r - 1)/2 \) constraints (\( 9 \times 8/2 = 36 \) in the example, figure 4.17c+d).

2. **Initialize polygon.** The output polygon is initialized to the rectangle covering the whole second view.

3. **Cut polygon.** One by one, all the sidedness constraints computed in point 1 are transferred to the second view. With each constraint added, the polygon is cut so as to contain only points within the half-plane defined by the constraint. Figures 4.18a+b shows the polygon after adding the first and the second constraint of our example. Note how the order in which the constraints are added has no influence on the final result. After adding each constraint the polygon is reduced to the intersection of the previous polygon with the constraining half-plane. The final polygon is the intersection of the initial rectangle with all half-planes (figure 4.18c).

In our example, the query point (upper part of the telephone in the first view) is nicely relocated from its erroneous position to a fairly small triangle, containing the correct location (figure 4.18c). In the general case, the size of the produced polygon, and therefore the accuracy of relocation, depends on the number of reference matches, and how they are spread in the images. As a rule of thumb, the more uniform the spread, the better the results.

Although the method works well on a number of examples, there is no absolute guarantee that the polygon will contain the correct location (the guarantee exists only for planar scenes). Depending on the scene structure and camera viewpoints, this could lie elsewhere in the image, because strong parallax effects might deteriorate the reliability of some constraints. In practice we observed the correct location to be either in the produced polygon, or in one adjacent to it. The latter is still a helpful indication, and sufficient for the re-matching purpose discussed above. The issues of polygon size and correctness, as well as the re-matching application, are
very interesting and would deserve a thorough investigation, but they are currently not settled yet.

4.9 Related work

The problem of finding feature correspondences across multiple views has been tackled with considerable success for the case of image sequences, where there is a clear temporal ordering and only small differences between subsequent images. Examples of such tracking approaches include [Shi and Tomasi 1994] and our own, presented in chapter 3. Other methods, still restricted to small inter-image differences, are extensions of two-view stereo matching algorithms to multiple views [Fua 1995, Roy and Cox 1998].

The case of large difference between images, mostly due to wide changes in viewpoint and camera orientation, has been catered for by several wide-baseline stereo algorithms in the recent years (for example [Tuytelaars and Van-Gool 2000, Mikolajczyk and Schmid 2002, Matas et al. 2002a, Baumberg 2000], see sections 2.4, 2.5). However, these schemes only deal with two views.

The general problem of finding feature correspondences across multiple, unordered wide-baseline views is quite new and largely unexplored. We could only find very few works dedicated to the problem, which we discuss next.

In [Rothganger et al. 2003] a sparse 3D model of an object is reconstructed, given a limited number of views taken from significantly different viewpoints. As intermediate step they describe a simple strategy to generate the necessary multiview matches. Affine invariant regions are extracted in each view and then matched between successive pairs of views. Pairwise matches sharing a common region are implicitly chained up, thereby forming matches across multiple views. This strategy avoids the structural problem of deriving multiview tracks from potentially contradictory pairwise matches, but only because it exploits an a priori given ordering of the views, allowing to consider only successive view pairs. Moreover, since nothing is done to counter the problem of the rapid decay of probability of an N-view track, the generated matches are likely to span only a few views. Nevertheless, this is sufficient for the purpose of [Rothganger et al. 2003], because the geometric constraints necessary for 3D reconstruction can be imposed already on small subsets of matches between 3, or even just 2 views. The authors then use points common to overlapping subsets to register successive partial reconstructions in a common global frame. The focus of [Rothganger et al. 2003] is more on 3D reconstruction and object recognition, while the multiple-view matching is seen as a side-issue.

---

6 This is a "**" section, covering additional material which has not yet reached full maturity.
Figure 4.18: Cutting the polygon. a) The initial rectangle around the image is cut by the first constraint. b) The second constraint (dashed) cuts the polygon (thick) further. The thin line shows the part of the polygon in a) being cut off. c) Final result, after enforcing all constraints. The correct correspondence for the query point lies within the polygon.
Our approach is more deeply involved in producing multiview matches. It assumes no ordering on the image set, actively counters the rapid decay problem, and thus manages to provide a number of reliable and long tracks, extending over many views, starting from a general set of unordered images. We believe that plugging the output of our approach into the reconstruction method of [Rothganger et al. 2003] would result in more complete and accurate 3D models.

The work that comes closest to ours is [Schaflitzky and Zisserman 2002b], referred to as SZ02 henceforth. To the best of our knowledge, it is the only other work focusing on finding multiple-view wide-baseline correspondences for affine invariant regions, starting from a set of unordered images. It is therefore discussed in some detail. The two approaches differ both in the global strategy and in the component algorithms. SZ02 introduces a novel method for describing the region's texture content with the useful property that Euclidean distance between descriptors directly predicts a lower bound on the sum-of-squared-differences (SSD) between regions. Our approach makes no contribution in this sense.

The initial matches are obtained in SZ02 with an indexing scheme where regions are represented by the new descriptors. This is very efficient, but the output is not, in general, properly structured into multiview tracks, and is typically contaminated with a large number of incorrect matches. In contrast, we match all pairs of images independently and then apply a new, principled mechanism for the integration of all pairwise matches in proper multiple-view tracks, accounting for mismatches and guaranteeing the structural correctness of the solution (see section 4.3).

The geometric registration of matches is refined by standard Levenberg-Marquardt in SZ02, whereas we propose a novel optimization technique, which is computationally efficient and avoids local maxima, because it is specialized to maximize a non-convex function, over a given, clearly bounded domain (see section 4.4).

The necessity for increasing the number of matches after the initial matching stage is felt also by SZ02. However, they use the affine transformation of established correspondences only to guide the search for further matches, which might have been missed e.g. due to inexact region localization. In our approach instead, the transformation is used to generate a new region, which is not necessarily originally extracted (section 4.5). This is crucial to counter the main problem exposed in the introduction: the probability of extracting the same region in Nviews quickly drops with N. The newly created region is additionally refined, so as to adapt to the local surface orientation and to help deciding whether it forms a correct match.

As last algorithmic difference, in SZ02 mismatches are searched with the well known method of detecting outliers to the epipolar geometry via RANSAC, while we designed a new filter, based on topological relationships, with some interesting advantages (section 4.6).
On the only example processed by both approaches (the Valbonne image set), ours produced more tracks. More importantly, the number of final $N$-view tracks decreases slowly and roughly linearly with $N$. In contrast, the values reported in SZ02 exhibit much steeper (and non-linear) decrease. Although these results are too limited for drawing conclusions, they certainly corroborate the effectiveness of our approach.

The main advantage of SZ02 is being linear in the number of views, whereas our method is quadratic. This is obtained by organizing the images in a tree, with adjacent ones having many matches. The matches growing and filtering stages then run only on pairs of adjacent images. Although theoretically important, and making SZ02 faster than our method, this has only moderate practical impact. There are two reasons for this. First, a limited number of wide-baseline views (typically 5 to 20) suffice to model a scene. Second, image pairs with few matches are processed very quickly anyway, thus they only cause very modest increase of the total computation time.

For the sake of completeness, we briefly mention the following approaches based on contours, rather than regions. Although much less related, they are equally interesting. Schmid and Zisserman describe an approach to match line segments [Schmid and Zisserman 1997] and curve fragments [Schmid and Zisserman 1998], although limited to 2 or 3 views. Curve fragments multiview matching and reconstruction is performed by the impressive, very recent work of [Kahl and August 2003]. The method assumes the relative camera motions to be known and curves to be visible in at least 3 or 4 views.

4.10 Conclusion and outlook

This chapter presented a method for obtaining multiple-view correspondences. The experiments confirm that the approach can generate a large number of high quality region tracks starting from an unordered set of images taken from substantially different viewpoints. The tracks strongly connect the views and provide a good coverage of the commonly visible parts of the scene. This is particularly important in object-recognition, where more coverage means a more complete description of the object. Accuracy is more an issue in 3D-reconstruction and has also been improved. The method works in conjunction with any wide-baseline stereo method based on local invariant regions.

Some individual components, like the refiner and the topological filter, are interesting in their own right, and can be used profitably beyond the scope of this chapter. The refiner algorithm has been integrated in some recent works by the University
of Oxford [Sivic and Zisserman 2004, Everingham and Zisserman 2004]. The propagation idea will be at the heart of the object recognition technique of the next chapter.

The approach presented in this chapter has been published in 2003 in [Ferrari et al. 2003].

The most interesting enhancement to the system would be to improve computational efficiency. Most processing stages (matching, propagation, etc.) run over all unordered pairs of \( N \) input images. While this is not a major problem, because only a small number of images (8 to 20) suffice to cover a scene, it would be preferable to have linear processing time in \( N \).

A potential solution could be to adopt the strategy of [Schaffalitzky and Zisserman 2002b], where the images are ordered in a tree structure based on the number of initial pairwise matches, so that images with many matches are adjacent. The further processing stages could then be applied only on pairs of adjacent images. However, there is the danger of losing considerable quality, especially due to the reduced chances given to propagation for completing tracks (section 4.5).
Object recognition and segmentation

5.1 Introduction

As discussed in section 2.6, object recognition (OR) approaches based on local invariant features have become increasingly popular in the recent years [Lowe 1999, Mikolajczyk and Schmid 2002, Tuytelaars and Van-Gool 2000, Rothganger et al. 2003, Obrdzalek and Matas 2002]. Let us briefly recall the common basic scheme behind these approaches, and their most important properties. Initially local features are extracted independently from both a model and a test image, then they are characterized by invariant descriptors, which are finally used for matching. The success of these approaches is twofold. First, the feature extraction process and description are viewpoint invariant. Secondly, local features bring tolerance to clutter and occlusion, de facto removing the need for prior segmentation. In this respect, global methods are a step behind, regardless if based on the contour [Cyr and Kimia 2001], color histogram [Swain and Ballard 1991], or complete images [Murase and Nayar 1995].

In spite of their success, the robustness and generality of these approaches are limited by the repeatability of the feature extraction, and the difficulty of matching correctly, in the presence of large amounts of clutter and challenging viewing conditions. Large scale or viewpoint changes considerably lower the probability that any given model feature is re-extracted in the test image (e.g.: figure 5.2a). Simultaneously, occlusion reduces the number of visible model features. The combined effect is that only a small fraction of model features has a correspondence in the test image. This fraction represents the maximal number of features that can be correctly matched. Unfortunately, at the same time extensive clutter gives rise to a large number of non-object features, which disturb the matching process. As a final outcome of these combined difficulties, only a few, if any, correct matches are produced. Because these often come together with many mismatches, recognition tends to fail.
5.1. Introduction

Figure 5.2a shows a challenging example, which will be used as case-study throughout the chapter. There is a large scale change (factor 3.3), out-of-plane rotation, extensive clutter and partial occlusion. All these factors make the life of the feature extraction and matching algorithms hard. Indeed, the approach of [Tuytelaars and Van-Gool 2000] fails badly, producing only 5 matches, all incorrect. Lowering the matching threshold does not help, as it results in 19 mismatches, and still no correct match being generated.

Even in easier cases, to suit the needs for repeatability in spite of viewpoint changes, only a sparse set of distinguished features [Obrdzalek and Matas 2002] are extracted. As a result, only a small portion of the object is typically covered with matches. Densely covering the visible part of the object is desirable, as it increases the evidence for its presence, which results in higher discriminative power. Moreover, it would allow to find the contours of the object, rather than just its location.

In this chapter, we tackle these problems by no longer relying solely on matching viewpoint invariant features. Instead, we propose to anchor on an initial set thereof, and then look around them trying to construct more matching features. As new matches arise, they are exploited to construct even more, in a process which gradually explores the test image, recursively constructing more and more matches, increasingly farther from the initial ones. As the number and extent of matched features increases, so does the information available to judge their individual correctness. Gradually the system’s confidence in the presence of the object grows.

Figure 5.1 shows a scheme of the new approach. We build upon a multi-scale extension of the affine invariant region extractor of [Tuytelaars and Van-Gool 2000] (section 2.4). An initial large set of unreliable region correspondences is generated through a process tuned to maximize the number of correct matches, at the cost of producing many mismatches (section 5.2). Additionally, we generate a grid of circular regions homogeneously covering the model image (coined coverage regions). The core of the method iteratively alternates between expansion phases, where correspondences for these coverage regions are constructed, and contraction phases, which attempt to remove mismatches. In the first expansion phase (section 5.3), we try to propagate the coverage regions based on the geometric transformation of nearby initial matches. By propagating a region, we mean constructing the corresponding one in the test image. The propagated matches and the initial ones are then passed through a novel local filter, during the first contraction phase (section 5.4). The processing continues by alternating faster expansion phases (section 5.5), where coverage regions are propagated over a larger area, with contraction phases based on a global filter (section 5.6). The filter exploits both topological arrangements and appearance information, and tolerates non-rigid deformations. During the expansion phases, the shape of each new region is adapted to the local surface orientation, thus allowing the exploration process to follow curved surfaces and deformations (e.g. a folded magazine). At each iteration, the presence of the newly
5.2 Soft matches

The first stage in the object recognition scheme is to compute an initial set of region matches between a model image $I_m$ and a test image $I_t$.

The region extraction algorithm [Tuytelaars and Van-Gool 2000] is applied to both images independently, producing two sets of regions $\Phi_m, \Phi_t$. Test regions $\Phi_t$ are matched to model regions $\Phi_m$ in two steps, described in the next two subsections. The matching procedure allows for soft matches, i.e.: more than one model region is matched to the same test region, or vice versa. The reasons for this choice are given in the last subsection.

**Figure 5.1:** Scheme of the system.

Propagated matches helps the filter to take better removal decisions. In turn, the cleaner set of matches makes the next expansion more effective. As a result, the amount, and the percentage, of correct matches grows every iteration. The algorithm is getting a clearer idea about the object's presence and location. The two closely cooperating processes of expansion and contraction gather more evidence about the presence of the object and separate correct matches from wrong ones at the same time. This results in the simultaneous recognition and segmentation of the object. By constructing matches for the coverage regions, the system succeeds in covering also image areas which are not interesting for the feature extractor or not discriminative enough to be correctly matched by traditional techniques.

The basic advantage of the approach is that each single correct initial match can expand to cover a contiguous surface with many correct matches, even when starting from a large number of mismatches. This leads to filling the visible portion of the object with matches. Some interesting direct advantages derive from it. First, robustness to scale, viewpoint, occlusion and clutter are greatly enhanced, because most cases where the traditional approach generated only a few correct matches are now solvable. Secondly, discriminative power is increased, because decisions about the object's identity are based on information densely distributed over the entire portion of the object visible in the test image. Thirdly, the approximate boundary of the object in the test image is directly suggested by the final set of matched regions (section 5.8). Fourthly, non-rigid deformations are explicitly taken into account.
5.2. Soft matches

Figure 5.2: a) Case-study. The cereal box appears partially occluded and much smaller in a heavily cluttered test image (bottom). b) A close-up with 3 initial matches. The two model regions on the left are both matched to the same region in the test image. Note the small occluding rubber on the spoon.

5.2.1 Tentative matches

For each test region $T \in \Phi_t$ we compute the Mahalanobis distance of the invariant descriptors to all model regions $M \in \Phi_m$:

$$d_M(f_M, f_T) = \sqrt{(f_M - f_T)^T C^{-1} (f_M - f_T)}$$

Where $f_M$ and $f_T$ are the feature vectors, and $C$ the covariance matrix (previously estimated from tracking data).

The appearance similarity measure $\overline{\text{sim}}(T, M)$ is computed between $T$ and each of the $10$ closest model regions. The measure was introduced in section 4.3 as equation 4.1. It is a linear combination of grey-level normalized cross-correlation (NCC) and average Euclidean distance in RGB space, after geometric and photometric normalization. This mixture is more discriminant than NCC alone, while keeping invariance to brightness changes (e.g.: due to dimming illumination or change of surface orientation w.r.t. to the light source).
5.2. Soft matches

Each of the 3 test regions most similar to $T$ above a low threshold $t_1$ are considered tentatively matched to $T$. Repeating this operation for all regions $T \in \Phi_t$, yields a first set of tentative matches.

At this point, every test region could be matched to either none, 1, 2 or 3 model regions.

5.2.2 Refinement and re-thresholding

Since all regions are independently extracted from the two images, the geometric registration of a correct match might not be optimal, which lowers its similarity. The registration of the tentative matches is refined using the algorithm of section 4.4, that efficiently looks for the affine transformation that maximizes the similarity. After refinement, the similarity is re-evaluated and only matches scoring above a second, higher threshold $t_2$ are kept. Refinement tends to raise the similarity of correct matches much more than that of mismatches. The increased separation between the similarity distributions makes the second thresholding more effective. At this point, about 1/3 to 1/2 of the tentative matches are left.

5.2.3 Motivation

The obtained set of matches usually still contains soft matches, i.e. more than one region in $\Phi_m$ corresponding to the same region in $\Phi_t$, or vice versa. This contrasts with classic matching methods [Obrdzalek and Matas 2002, Tuytelaars and Van-Gool 2000, Mikolajczyk and Schmid 2002, Baumberg 2000, Lowe 2004], where one region in an image is matched to at most one region in the other image, but there are two good reasons for it. First, the scene might contain repeated, or visually similar elements. Secondly, large viewpoint and scale changes cause loss of resolution which results in a less accurate geometric correspondence and a lower similarity. When there is also extensive clutter, it might be impossible, based purely on local appearance [Schaffalitzky and Zisserman 2002a], to decide which of the best-3-matches is correct, as several competing regions might appear very similar, and score higher than the correct match. A classic 1-to-1 approach may easily be distracted and fail to produce the correct match.

The proposed process outputs a large set of plausible matches, all with a reasonably high similarity. The goal is to maximize the amount of correct matches, even at the cost of accepting a substantial fraction of mismatches. This is important in difficult cases, when only a few model regions are re-extracted in the test image, because each correct match can start an expansion which will cover significant parts of the object.
Figure 5.2a shows the case-study, for which 3 correct matches out of 217 are found (a correct-ratio of 3/217). The large scale change (factor 3.3), combined with the modest resolution (720x576), causes heavy image degradation which corrupts edges and texture. In such conditions only a few model regions are re-extracted in the test image and many mismatches are inevitable. In the remainder of the chapter, we refer to the current set of matches as the configuration $\Gamma$.

How to proceed? Global, robust geometry filtering methods, like detecting outliers to the epipolar geometry through RANSAC [Torr and Murray 1997] fail, as they need a minimal portion of inliers of about 1/3 [Lowe 2004, Chum et al. 2003]. Initially, this may very well not be the case. Even if we could separate out the few correct matches, they would probably not be sufficient to draw reliable conclusions about the presence of the object. In the following sections, we explain how to gradually increment the number of correct matches and simultaneously decrease the number of mismatches.

### 5.3 Early expansion

#### 5.3.1 Coverage of the model image

We generate a grid $\Omega$ of overlapping circular regions densely covering the model image $I_m$ (figure 5.3a). In the implementation used for the result section 5.8 the grid is composed by a first layer of regions of radius 25 pixels, spaced 25 pixels, and a second layer with radius 13 pixels and spaced 25 pixels. No regions are generated on the black background. According to various experiments, this choice of the parameters is not crucial for the overall recognition performance. The choice of the exact grid pattern, and consequently the number of regions in $\Omega$, trades segmentation quality for computational cost, and should be selected based on the user’s desires.

At this point, none of the regions in $\Omega$ is matched to the test image $I_t$. The expansion phases will try to construct in $I_t$ as many regions corresponding to them as possible.

#### 5.3.2 Propagation attempt

We now define the concept of propagation attempt which is the basic building-block of the expansion phases and will be used later. Consider a region $C_m$ in model image

1. These values are appropriate for an image of 720x576 pixels. They are proportionally adapted for images of other sizes.

2. Propagation attempts were first introduced in section 4.5.
5.3. Early expansion

Figure 5.3: a) The homogeneous coverage Ω. b) A support region (dark), associated sectors (lines) and candidates (bright).

Im without match in the test image It and a nearby region Sm, matched to St. If Cm and Sm lie on the same physical facet of the object, they will be mapped to It by similar affine transformations. The support match (Sm, St) attempts to propagate the candidate region Cm to It as follows:

1. Compute the affine transformation A mapping Sm to St.
2. Project Cm to It via $C_t = AC_m$.

The benefits of exploiting previously established geometric transformations was also noted by [Schaffalitzky and Zisserman 2002b].

5.3.3 Early expansion

Propagation attempts are used as a basis for the first expansion phase as follows. Consider as supports $\{S_i = (S_{m_i}, S_{i})\}$ the soft-matches configuration $\Gamma$, and as candidates $\Lambda$ the coverage regions $\Omega$. For each support region $S_{m_i}$ we partition $I_m$ into 6 circular sectors centered on the center of $S_{m_i}$ (figure 5.3b).

Each $S_{m_i}$ attempts to propagate the closest candidate region in each sector. As a consequence, each candidate $C_m$ has an associated subset $\Gamma_{C_m} \subset \Gamma$ of supports that will compete to propagate it. For a candidate $C_m$ and each support $S_i$ in $\Gamma_{C_m}$ do:

1. Generate $C_i^j$ by attempting to propagate $C_m$ via $S_i^j$.
2. Refine $C_i^j$. If $C_i^j$ correctly matches $C_m$, this adapts it to the local surface orientation (handles curved and deformable objects) and perspective effects (the affine approximation is only valid on a local scale).
3. Compute the color transformation $T^{i}_{\text{RGB}} = \{s_R, s_G, s_B\}$ between $S^i_m$ and $S^i_t$. This is specified by the scale factors on the three colorbands.

4. Evaluate the quality of the refined propagation attempt, after applying the color transformation $T^{i}_{\text{RGB}}$

$$sim_i = \text{sim}(C^i_m, C^i_t, T^{i}_{\text{RGB}})$$

Applying $T^{i}_{\text{RGB}}$ allows to use the unnormalized similarity measure $\text{sim}$, because color changes are now compensated for. This provides more discriminative power over using $\text{sim}$.

We retain $C^{\text{best}}_i$, with $\text{best} = \arg\max_i sim_i$, the best refined propagation attempt. $C_m$ is considered successfully propagated to $C^{\text{best}}_i$ if $sim_{\text{best}} > t_2$. Note that $t_2$ is the threshold used already for the initial matching (section 5.2). This is a consistent choice, as the origin of a matching region, whether obtained from a viewpoint invariant extractor or from the propagation procedure, makes no difference. This procedure is applied for all candidates $C^i_m \in \Lambda$.

Most support matches may actually be mismatches, and many of them typically lie around each of the few correct ones (e.g.: several matches in a single soft-match, figure 5.2b). In order to cope with this situation, each support concentrates its efforts on the nearest candidate in each direction, as it has the highest chance to undergo a similar geometric transformation. Additionally, every propagation attempt is refined before evaluation. Refinement raises the similarity of correctly propagated matches much more than the similarity of mispropagated ones, thereby helping correct supports to win. This results in a limited, but controlled growth, maximizing the chance that each correct match propagates, and limiting the proliferation of mispropagations. The process also restricts the number of refinements to at most 6 per support (contains computational cost).

For the case-study, 113 new matches are generated and added to the configuration $\Gamma$. 17 of them are correct and located around the initial 3 (figure 5.4a). The correct-ratio of $\Gamma$ improves to $20/330$ (figure 5.4b), but it is still very low.

### 5.4 Early contraction

The early expansion guarantees high chances that each initial correct match propagates. As initial filter, we discard all matches that did not succeed in propagating any region. The correct-ratio of the case-study improves to $20/175$ (no correct match is lost), but it is still too low for applying a global filter. Hence, we have developed the following local filter.

---

3The measure $\text{sim}$ was introduced in section 4.3, while $\text{sim}$ in section 4.5
5.4.1 Surface contiguity filter

A local group of regions in the model image have uniform shape, are arranged on a grid and intersect each other with a specific pattern. If all these regions are correctly matched, the same regularities also appear in the test image, because the surface is contiguous and smooth (regions at depth discontinuities cannot be correctly matched anyway). This holds for curved or deformed objects as well, because the affine transformation varies slowly and smoothly across neighboring regions. (figure 5.5a). On the other hand, mismatches tend to be located elsewhere in the image and to have different shapes.

We propose a local filter based on this observation. Let \( \{ N^i_m \} \) be the neighbors of a region \( R_m \) in the model image. Two regions \( A, B \) are considered neighbors if they intersect, i.e. if Area(\( A \cap B \)) > 0. Only neighbors which are actually matched to the test image are considered. Any match \( (R_m, R_t) \) is removed from \( \Gamma \) if
5.4. Early contraction

Figure 5.5: Surface contiguity filter. a) The pattern of intersection between neighboring correct region matches is preserved by transformations between the model and the test images, because the surface is contiguous and smooth. b) The filter evaluates this property by testing the conservation of the area ratios.

\[
\sum_{\{N_m\}} \left| \frac{\text{Area}(R_m \cap N_m^i)}{\text{Area}(R_m)} - \frac{\text{Area}(R_t \cap N_t^i)}{\text{Area}(R_t)} \right| > t_s
\]  

(5.1)

with \(t_s\) some threshold. The filter, illustrated in figure 5.5b, tests the preservation of the pattern of intersections between \(R\) and its neighbors (the ratio of areas is affine invariant). Hence, a removal decision is based solely on local information. As a consequence, this filter is unaffected by the current, low overall ratio of correct matches.

Shape information is integrated in the filter, making it capable of spotting insidious mismatches which are roughly correctly located, yet have a wrong shape. This is an advantage over the (semi-)local filter proposed by [Schmid 1996], and later also used by others [Schaffalitzky and Zisserman 2002a, Sivic and Zisserman 2003], which verifies if a minimal amount of regions in an area about \(R_m\) in the model image also match near \(R_t\) in the test image.
5.5 Main expansion

The input regions need not be arranged in a regular grid, the filter applies to a general set of (intersecting) regions. Note that isolated mismatches, which have no neighbors in the model image, will not be detected. The algorithm can be implemented to run in $O((|\Gamma| + x) \log(|\Gamma|))$, with $x \ll |\Gamma|^2$ the number of region intersections, as shown in appendix B.

Applying this filter to the case-study brings the correct-ratio of $\Gamma$ to 13/58, thereby greatly reducing the number of mismatches.

5.5 Main expansion

The first early expansion and contraction phases brought several additional correct matches and removed many mismatches, especially those that concentrated around the correct ones. Since $\Gamma$ is cleaner, we can now try a faster expansion.

All matches in the current configuration $\Gamma$ are removed from the candidate set $\Lambda \leftarrow \Lambda \setminus \Gamma$, and are used as supports. All support regions $S_m$ in a circular area around a candidate $C_m$ compete to propagate it:

1. Generate $C_t^i$ by attempting to propagate $C_m$ via $S^i$.
2. Compute the color transformation $T_{RGB}^i$ of $S^i$.
3. Evaluate $\text{sim}_i = \text{sim}(C_m, C_t^i, T_{RGB}^i)$.

We retain $C_t^{best}$, with $\text{best} = \arg\max_i \text{sim}_i$ and refine it, yielding $C_t^{ref}$. $C_m$ is considered successfully propagated to $C_t^{ref}$ if $\text{sim}(C_m, C_t^{ref}) > t_2$ (figure 5.6). This scheme is applied for each candidate.

In contrast to the early expansion, many more supports compete for the same candidate, and no refinement is applied before choosing the winner. However, the presence of more correct supports, now tending to be grouped, and fewer mismatches, typically spread out, provides good chances that a correct support will win a competition. In this process each support has the chance to propagate many more candidates, spread over a larger area, because it offers help to all candidates within a wide circular radius. This allows the system to grow a mass of correct matches. Moreover, the process can jump over small occlusions or degraded areas, and costs only one refinement per candidate.

For the case-study, 185 new matches, 61 correct, are produced, thus lifting the correct-ratio of $\Gamma$ up to 74/243 (31%, figure 5.11 bottom row).

\footnote{In all experiments the radius is set to 1/6 of the image size (i.e. max(width, height)).}
5.6 Main contraction

At this point the chances of having a sufficient number of correct matches to try a global filter are much better. In contrast to the local filter of section 5.4, the following global filter is capable of finding also isolated mismatches. The algorithm extends the topological filter of section 4.6 to include also appearance similarity. We give here only a summary, and discuss the appearance extension. The interested reader is invited to read section 4.6 for more details.

Figure 5.7 illustrates the property on which the filter is based. The center of a region \( R^1 \) should be on the same side of the directed line going from the center of a second region \( R^2 \) to the center of a third region \( R^3 \) in both the model and test images (noted side\((R^1, R^2, R^3)\)). This sidedness constraint holds for all correctly matched triples of coplanar regions and also for most non-coplanar ones. It does not hold for non-coplanar triples in the presence of strong parallax in a few cases, coined parallax-violations.

A triple including any mismatched region has higher chances to violate the constraint. When this happens, it indicates that probably at least one of the matches is incorrect, but it does not tell which one(s). However, mismatches can be robustly discovered by integrating the weak information provided by each triple: we check the constraint for all unordered triples and we expect wrong matches to be involved in a higher share of violations:

\textbf{Figure 5.6:} a) a candidate (thin) and 2 of 20 supports within the large circular area. b) the candidate is propagated to the test image using the affine transformation \( A \) of the support on the right (thick). Refinement adapts the shape to the perspective effects (brighter). The left support in a) is mismatched to a region not visible in this close-up.
Figure 5.7: Sidedness constraint. $R^1$ is on the same side of the line in both images.

$$\text{err}_\text{topo}(R^i) = \frac{1}{v} \sum_{R^j, R^k \in \Gamma \setminus R^l \setminus R^j} |\text{side}(R^i_m, R^j_m, R^k_m) - \text{side}(R^i_t, R^j_t, R^k_t)| \quad (5.2)$$

with $v = (n-1)(n-2)/2$, $n = |\Gamma|$. $\text{err}_\text{topo}(R^i) \in [0,1]$ because it is normalized w.r.t. the maximum number of violations $v$ any region can be involved in.

As an extension to section 4.6, the topological error share (5.2) is combined with an appearance term, giving the total error:

$$\text{err}_\text{tot}(R^i) = \text{err}_\text{topo}(R^i) + (t_2 - \text{sim}(R^i_m, R^i_t))$$

The filtering algorithm goes as follows:

1. (Re-)compute $\text{err}_\text{tot}(R^i)$ for all $R^i \in \Gamma$.
2. Find the worst match $R^w$, with $w = \arg\max_i \text{err}_\text{tot}(R^i)$
3. If $\text{err}_\text{tot}(R^w) > 0$, remove $R^w \leftarrow (\Gamma \setminus R^w)$, and iterate to point 1, else stop.

The idea of the algorithm is that at each iteration the most probable mismatch $R^w$ is removed and the error of correct matches decreases, because they are involved in less triples containing any mismatch. After several iterations, ideally only correct matches are left and the algorithm stops.

The second term of $\text{err}_\text{tot}$ decreases with increasing appearance similarity, and it vanishes when $\text{sim}(R^i_m, R^i_t) = t_2$, the matches acceptance threshold. The removal criteria $\text{err}_\text{tot} > 0$ expresses the idea that topological violations are accepted up to the degree to which they are compensated by high similarity. This helps finding mismatches which can hardly be judged by only one cue. A typical mismatch with
5.6. Main contraction

Figure 5.8: Sidedness constraints hold also for deformed objects. The small arrows indicate 'to the right' of directed lines $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A$.

similarity just above $t_2$, will be removed unless it’s perfectly topologically located. Conversely, correct matches with $\text{err}_{\text{topo}} > 0$ due to parallax-violations are in little danger, because they typically have good similarity. Including appearance makes the filter more robust to low correct-ratios, and remedies the drawback (parallax-violations) of the purely topological filter.

The proposed method offers several advantages over rigid-motion filters, traditionally used in the matching literature, e.g.: detecting outliers to the epipolar geometry through RANSAC [Torr and Murray 1997]. While these are discussed in subsection 4.6.3, it is worth discussing here the most significant one for object recognition: the filter allows for non-rigid deformations, like the bending of paper or cloth. This is possible because the structure of the spatial arrangements, captured by the sidedness constraints, is stable under deformations, unlike quantitative measurements of distances. Like shown figure 5.8, sidedness constraints are still respected even in the presence of substantial deformations. Other filters, which measure a geometrical distance error from an estimated model (e.g.: homography, fundamental matrix) would fail in this situation. In the best case, they would discard several correct matches. Worse yet, in many cases, the deformations would disturb the estimation of the model parameters, resulting in a largely random behavior. A remarkable strength of the proposed filter is that it does not try to capture the transformations of all matches in a single, overall model, but it relies instead on measuring simpler, weak properties, involving only three matches each. The discriminative power is then obtained by integrating over all measurements, revealing their strong, collective information.
Topological configurations of points and lines are also used in [Tell and Carlsson 2002], which enforces the cyclic ordering of line segments connecting corners as a mean for steering the matching process.

On the case-study, the filter starts from 74/243 and returns 54/74 (figure 5.12 top), which is a major improvement. 20 correct matches are lost, but many more mismatches (149) are removed. The further processing will recover the correct matches lost and generate even more.

5.7 Exploring the test image

The processing continues by iteratively alternating main expansion and main contraction phases.

1. Do a main expansion phase. All current matches $\Gamma$ are used as supports.
   This produces a set of propagated region matches $\Upsilon$, which are added to the configuration: $\Gamma \leftarrow (\Gamma \cup \Upsilon)$.

2. Do a main contraction phase on $\Gamma$. This removes matches from $\Gamma$.

3. If at least one newly propagated region survives the contraction, i.e. if $|\Upsilon \cap \Gamma| > 0$, then iterate to point 1, after updating the candidate set to contain $\Lambda \leftarrow (\Omega \setminus \Gamma)$, all original candidate regions $\Omega$ which are not yet in the configuration. Stop if no newly propagated regions survived, or if all regions $\Omega$ have been propagated (i.e. if $\Omega \subseteq \Gamma$).

In the first iteration, the expansion phase generates some correct matches, along with some mismatches, thereby increasing the correct-ratio. The first main contraction phase removes mostly mismatches, but might also lose several correct matches: the amount of noise (percentage of mismatches) could still be high and limit the filter’s performance. In the next iteration, this cleaner configuration is fed into the expansion phase again which, less distracted, generates more correct matches and fewer mismatches. The new correct matches in turn help the next contraction stage in taking better removal decisions, and so on. As a result, the amount, percentage and spatial extent of correct matches increase at every iteration, reinforcing the confidence about the object’s presence and location (figures 5.9 and 5.10). The two goals of separating correct matches and gathering more information about the object are achieved at the same time.

Correct matches erroneously killed by the contraction step in an iteration get another chance during the next expansion phase. With even fewer mismatches present,
5.7. Exploring the test image

Figure 5.9: Evolution of the number of correct matches for the case-study. The number increases at every iteration (compare the points after each contraction phase). The sawtooths are due to the alternation of expansion (increase) and contraction (decrease).

Figure 5.10: Evolution of the percentage of correct matches (correct-ratio). This steady growth best illustrates the increasing confidence of the system in the presence of the object. Note that the method achieves the enormous improvement of passing from 1.4% after soft-matching, to 91.8% after the last iteration.

...they are probably regenerated, and this time have higher chances to survive the contraction (higher correct-ratio, more positive evidence present).

Thanks to the refinement, each expansion phase adapts the shape of the newly created regions to the local surface orientation. Thus the whole exploration process follows curved surfaces and deformations.

The exploration procedure tends to 'implode' when the object is not in the test image, typically returning only very few matches. Conversely, when the object is present, the approach fills the visible portion of the object with many high confidence matches. This yields high discriminative power and the qualitative shift from only detecting the object to knowing its extent in the image and which parts are occluded. Recognition and segmentation are two aspects of the same process.

In the case-study, the second main expansion propagates 141 matches, 117 correct, which is better than the previous 61/185. The second main contraction starts from
171/215 and returns 150/174, killing a lower percentage of correct matches than in the first iteration. After the 11th iteration 220 matches cover the whole visible part of the object (202 are correct). Figures 5.11 and 5.12 depict the evolution of the set of matches $\Gamma$ along all phases of the proposed scheme, from the initial matches, through the early expansion and contraction, all the way up the iterative alternation of main expansion and main contraction, until the final set of matches. The correct matches gradually cover more and more of the object, while mismatches decrease in number. The set of matches gets progressively cleaner. The system successfully reversed the situation, by going from only very few correct matches in a large majority of mismatches, to hundreds of correct matches with only a few mismatches. This effect is crucial for the enhanced power of the system: particularly hard cases, beyond the reach of traditional methods, can now be solved, because only very few correct initial matches suffice for reliable recognition, even when starting from a large majority of mismatches. Each single correct initial match can expand to cover a contiguous surface, so typically just one correct match per facet of the object is enough.

Notice that the final segmentation is quite accurate. In particular the small rubber, occluding the spoon, has been correctly left out (figure 5.12, bottom).

### 5.8 Results

The image exploration scheme is evaluated in two scenarios, each involving different kinds of data. The first is the classic object recognition situation, where a pool of known model objects is searched for in a set of test images (next subsection). Both model and test images are taken by the author of this thesis. The second scenario is video retrieval, the problem of automatically finding a specific object or scene in a video (subsection 5.8.2). The input video streams are real news broadcasts from the RTBF Belgian television channel.

#### 5.8.1 Object recognition

The dataset for the classic object recognition scenario consists of 9 model objects and 23 test images. In total, the objects appear 43 times, as some test images contain several objects. To facilitate the discussion, the images are referred to by their coordinates as in figures 5.14, 5.15, 5.16. There are 3 planar objects, each modeled by a single view, including a Kellogs box \(^5\) and two magazines, Michelle (figure b2) and Blonde (analog model view). Two objects with curved shapes, Xmas (h2) and Ovo (f1), have 6 model views. Leo (d1), Car (g1), Suchard (e2) feature more complex 3D shape and have 8 model views. Finally, one frontal view models the

---

5The kellogs box is used throughout the chapter as a case-study. It is not discussed again here.
Figure 5.11: Evolution of $\Gamma$. Left: correct matches, Right: mismatches.
Figure 5.12: Evolution of $\Gamma$ (continued).
last 3D object, *Guard* (a1). Multiple model views are taken equally spaced around the object (all model views are reported in appendix E). The contributions from all model views of a single object are combined by superimposing the area covered by the final set of matched regions (to find the contour), and by summing their number (recognition criteria). All images are shot at a modest resolution (720x576) and all experiments are conducted with the same set of parameters. In general, in the test cases there is considerable clutter and the objects appear smaller than in the models (all model images have the same resolution as the test images and they are shown at the same size).

Tolerance to deformations is shown in a2, where *Michelle* is simultaneously strongly folded and occluded. The contours are found with a good accuracy, extending to the left until the edge of the object. Note the extensive clutter. High robustness to viewpoint changes is demonstrated in b1, where *Leo* is only half visible and captured in a considerably different pose than any of the model views, while *Michelle* undergoes a very large out-of-plane rotation of about 80 degrees. *Guard*, occluding *Michelle*, is also detected in the image, despite a scale change of factor 3. In e1, *Leo* and *Ovo* exhibit significant viewpoint changes, while *Suchard* is simultaneously scaled by factor 2.2 and 89% occluded. This very high occlusion level makes this case challenging even for a human observer. A scale change of factor 4 affecting *Suchard* is illustrated in f2. In figure g2, *Xmas* is divided in two by a large occluder. Both visible parts are correctly detected by the presented method. On the right side of the image, *Car* is found even if half occluded and very small. *Car* is also detected in spite of a considerable viewpoint change in h1. The combined effects of strong occlusion, scale change and clutter make i2 an interesting case. Note how the boundaries of *Xmas* are accurately found, and in particular the detection of the part behind the glass. As a final example, 8 objects are detected at the same time in d2 (for clarity, only 3 contours are shown). Note the correct segmentation of the two deformed magazines and the simultaneous presence of all the aforementioned difficulties.

Figure i1 presents a close-up on one of 93 matches produced between a model view of *Xmas* (left) and test case i2 (right). This exemplifies the great appearance variation resulting from combined viewpoint, scale and illumination changes, and other sources of image degradation (here a glass). In these cases, it is very unlikely for the region to be detected by the initial region extractor, and hence traditional methods fail. This figure also illustrates the accuracy of the correspondences generated by the expansion phases.

As a proof of the method's capability to follow deformations, we tried to process the case in c2 starting with only one match (dark). 356 regions, covering the whole object, were produced. Each region's shape fits the local surface orientation (for clarity, only 3 regions are shown).
The discriminative power of the system was assessed by processing all pairs of model-object and test images, and counting the resulting number of region matches. The highest ROC curve in figure c1 depicts the detection rate versus false-positive rate, while varying the detection threshold from 0 to 200 matches. The method performs very well, and can achieve 98% detection with 6% false-positives. For comparison, we processed the dataset also with 4 state-of-the-art affine region extractors [Obrdzalek and Matas 2002, Mikolajczyk and Schmid 2002, Baumberg 2000, Tuytelaars and Van-Gool 2000], and described the regions with the SIFT [Lowe 2004] descriptor 6, which has recently been demonstrated to perform best [Mikolajczyk and Schmid 2003]. The matching is carried out by the 'unambiguous nearest-neighbor' approach 7 advocated in [Baumberg 2000, Lowe 1999]: a model region is matched to the region of the test image with the closest descriptor if it is closer than 0.7 times the distance to the second-closest descriptor (the threshold 0.7 has been empirically determined to optimize results). Each of the central curves in i1 illustrates the behavior of a different extractor. As can be seen, none is satisfactory, which demonstrates the higher level of challenge posed by the dataset and therefore suggests that our approach can broaden the range of solvable Object Recognition cases. Closer inspection reveals the source of failure: typically only very few, if any, correct matches are produced when the object is present, which in turn is due to the lack of repeatability and the inadequacy of a simple matcher under such difficult conditions. The important improvement brought by the proposed method is best quantified by the difference between the highest curve and the central thick curve, representing the system we started from [Tuytelaars and Van-Gool 2000] (labeled 'TT00 org' in the plot). The other curves of the plot correspond to the different extractors as follows. 'KM02' to [Mikolajczyk and Schmid 2002]; 'TT00' to [Tuytelaars and Van-Gool 2000]; 'Baum00' to [Baumberg 2000]; 'Mat02' to [Obrdzalek and Matas 2002].

Figure 5.13a shows a histogram of the number of final matches (recognition score) output by our system. The scores assigned when the object is in the test image (positive cases) are much higher than when the object is absent (negative cases). This very good separation brings discriminative power and is due to the combination of two effects. First, the exploration process tends to implode in negative cases, because the expansion phases can do very little and the contraction phases eat up all the matches. Conversely, the method fills the object with matches when it is present, as expansions can prosper on much fertile surface. As a comparison

6 All region extractors and the SIFT descriptor are implementations of the respective authors. We are grateful to Jiri Matas, Krystian Mikolajczyk, Andrew Zisserman, Cordelia Schmid and David Lowe for providing the programs.

7 We have also tried the standard approach, used in [Obrdzalek and Matas 2002, Mikolajczyk and Schmid 2002, Tuytelaars and Van-Gool 2000, Mikolajczyk and Schmid 2003], which simply matches two nearest-neighbors if their distance is below a threshold, but it produced slightly worse results.
5.8. Results

Figure 5.13: Distribution of scores (in percentage). a) The new method. The scores of positive (bright) and negative cases (dark) are well separated. 57% of the negative cases are in the leftmost bin, and have less than 20 matches. 34% even have exactly 0 matches. b) Traditional matching of the regions of [Obrdzalek and Matas 2002], based on the SIFT descriptor. Positive and negative cases cannot be reliably distinguished. 59% of the negative cases, and 30% of the positive ones, are in the leftmost bin.

with the traditional methods, the standard matching of regions of [Obrdzalek and Matas 2002], based on the SIFT descriptor, yields two hardly separable distributions (figure 5.13b), and hence the unsatisfactory performance in the ROC plot. Similar histograms are produced based on the other feature extractors [Mikolajczyk and Schmid 2002, Baumberg 2000, Tuytelaars and Van-Gool 2000].

5.8.2 Video retrieval

Scenario

In the video retrieval scenario, we tackle the problem of automatically finding a specific object or scene in a video. The object is only given as delineated by the user in one, or a few, model images. This is useful in the context of video annotation: instead of having a person manually marking every occurrence of interesting objects in the video, the system could do this largely automatically, resorting to human intervention only to delineate each object once.

Traditionally, in such content-based image retrieval task the object is represented by global features, which collect information across the whole image. Examples of such features are histograms of color and texture, or edge-signatures [Smeulders et al. 2000]. In contrast, using local invariant features brings robustness to background clutter and partial occlusions. Moreover, viewpoint changes are allowed and the search focuses on a specific object, rather than on mere similar appearance (e.g.:
Figure 5.14: Object recognition results I (see main text for discussion).
Figure 5.15: Object recognition results II (see main text for discussion).
Figure 5.16: Object recognition results III (see main text for discussion).
Results

Adding the techniques presented in this chapter, on top of the standard region extraction and matching scheme, brings the aforementioned advantages: robustness to scale, viewpoint, occlusion and clutter is enhanced; the approximate boundaries of the object are found; non-rigid deformations are supported. These help to cope with the challenging conditions of real-world video material, such as television news broadcast. Unlike in the object recognition setting, such a video usually offers only low quality images, affected by compression artifacts, motion blur, and interlacing effects.

In [Sivic and Zisserman 2003] another region-based system for video object retrieval is presented. However, it focuses on different aspects of the problem, namely the organization of regions coming from several shots, and weighting their individual relevance in the wider context of the video. At the feature level, their work still relies solely on regions from standard extractors.

Approach

Due to the different nature of the data, the system differs in some points from the object recognition one. The input consists of a model-image containing the object to be annotated delineated by the user, and a test video stream where to search for the object. The model-image does not necessarily have to come from the test video.

The processing is divided in learning and recognition. During learning, regions are extracted from the object part of the model-image (again, we use [Tuytelaars and Van-Gool 2000], but any affine invariant region extractor is suitable). The recognition phase goes through the following stages:

1. **Video segmentation.** The input video is segmented into shots, and a few representative keyframes are selected in each shot. The video is sampled so that subsequent keyframes are significantly different, but still adequately cover the whole content of the shot. This operation is performed by the algorithm of [Osian and Van-Gool 2003]. Further stages only inspect the keyframes.

2. **Region extraction.** Regions are extracted in all keyframes, with the algorithm of [Tuytelaars and Van-Gool 2000].

3. **Keyframes exploration.** This stage is applied for all keyframes. For each keyframe, its regions are first matched to the model regions, and then a variant of the image exploration algorithm is applied.

4. **Detection.** The object is detected in every keyframe with more than a predefined amount of matches (after the exploration stage).

The image exploration scheme applied in point 3 is a simplified version of the one presented in this chapter. Specifically, it uses a simple one-to-one nearest neighbor
Results

Experiments were conducted on news broadcast material from the RTBF Belgian television channel. The data comes from 4 different news report videos, captured on different days, each of about 20 minutes. Keyframes were obtained through the algorithm of [Osian and Van-Gool 2003]. The image quality is quite low: the keyframes have low resolution (672x528) and many of them are visibly affected by compression artifacts, motion blur and interlacing effects. We selected 13 objects, including locations, advertising products, logos and football shirts, and delineated each in one representative keyframe. Each object is searched in the keyframes of the video containing its model-image. On average, an object is searched in 325 keyframes, and occurs 7.4 times. The number of 'negatives', i.e.: keyframes not containing an object, is therefore much greater than the number of positives, which allows to collect significant statistics. A total of 4236 (object,keyframe) image pairs have been processed.

Figures 5.17 and 5.18 show some examples of successful detections. A large piece of quilt decorated with various flags is found in figures 5.18c+d in spite of non-rigid deformation, occlusion and extensive clutter. Notice the completeness of the segmentation (the right part is self-occluded in the model and, correctly, left undetected in the keyframe, as is the part occluded by the RTBF logo).

An interesting application is depicted in figures 5.17a+b+c. The shirts of two football teams are picked out as query objects (figure 5.17a) The system is then asked to find the keyframes where the first team (Dexia) is playing, and where the other team (Fortis) is playing. In figure 5.17c the Fortis shirt is successfully found in spite of moderate rotation and motion blur. Both teams are identified in figure 5.17b, even if the shirts appear much smaller and the Dexia player is turned 45 degrees (viewpoint change on the shirt).

Robustness to large scale changes and occlusion is demonstrated in figure 5.17e, where the UN council, modeled in figure 5.17d, is recognized while enlarged by a scale factor 2.7, and heavily occluded: only 10% of the model image is visible. Equally intriguing is the image of figure 5.17f, where the UN council is seen from an opposite viewpoint. The large painting on the left in the model image is about the only thing still visible in the test keyframe, where it appears on the right side. The system managed to match the whole area of the painting, which suffers from out-of-plane rotation, and thus retrieve the UN council.
As a last example, a room with Saddam Hussein is found in figure 5.18b (model in 5.18a). The keyframe is taken under a different viewpoint and corrupted by considerable motion blur.

The retrieval performance is measured by the detection rate and false positive rate, averaged over all 13 objects. For an object, the detection rate is the number of correct detections divided by the total number of times the object occurs in the video (to keep results fair, detections of model-keyframes are not counted). The false positive rate refers to the number of wrong detections over the number of negatives. An object is detected if the number of final matches, divided by the number of model coverage regions (see subsection 5.3.1), exceeds 10%. The system performs well, by achieving an average detection rate of 82.4%, for a false-positive rate of 3.6%. As a comparison, we repeated the whole experiment with [Tuytelaars and Van-Gool 2000], the method we started from. It only managed a 33.3% detection rate, for a false-positive rate of 4.6%, showing that our approach can substantially boost the performance of standard affine invariant matching schemes.

It takes on average 2.16 minutes to process a (object,keyframe) pair on a common workstation (2.4 Ghz PC). While this is not particularly fast, and far slower than real-time, it is still a reasonable computational requirement for off-line processing. In this scenario, the system is run beforehand on many potentially interesting objects, and user-queries are processed in real-time based on the pre-computed annotations (like in [Sivic and Zisserman 2003]).

The results confirm the viability of our approach for retrieving objects in the challenging, real-world conditions of news broadcast video data. The method is very effective against viewpoint and scale changes, occlusion, clutter and is robust to moderate amounts of image degradation, like motion blur and compression artifacts. Moreover, deformable objects are taken into account and the approximate contours of the object are produced.

5.9 Sparsely textured objects*

This section covers a preliminary method for addressing the problem of sparsely textured objects. This method is not part of the Object Recognition scheme presented in the rest of the chapter. It is a separate, additional issue.

5.9.1 Motivation and idea

The techniques of this chapter allow to substantially boost the performance of traditional region matching schemes, in that their power and functionality (segmentation)
Figure 5.17: Video retrieval results I. The parts of the model-images not delineated by the user are blanked out. The sizes have not been altered: the model images are shown at the same scale as the test keyframes. Details in the text.
are enhanced. However, the scope remains limited to objects where a large portion of the surface is covered by texture. Indeed, the initial region extraction phase does not respond to uniform areas, and propagation is unreliable there. While objects featuring only a few textured spots, spread over much larger uniformly colored surfaces, could sometimes be recognized based purely on the former, it would be much more discriminative to have a dedicated mechanism for this situation.

In this section we present a method for explicitly dealing with such sparsely textured objects, like the handkerchief-distributor of figure 5.19a. In a nutshell, the idea is to automatically detect uniform regions in the model image, and then propagate them to the test image via the affine transformations of regular region matches (subsection 5.9.2). The intuition behind this can be explained on the example of figure 5.19a. The large whitish area has in itself only little distinctive power: detecting another whitish area in a test image would not be a decisive indication.
5.9. Sparsely textured objects*

Figure 5.19: Handkerchief-distributor. a) Model image. Note the CWS logo on the right. b) Three uniform regions extracted by the color segmentation algorithm.

of the object's presence. However, suppose that we first find the CWS logo, and then note that there is a whitish area around it, exactly like observed in the model image! The uniform area would definitely be a contributing evidence, and the total confidence in the presence of the object would be greater than with either cue alone (only CWS logo, or only whitish area). When associated to relatively distinct textured spots, the uniform areas become a relevant cue.

In the next subsection the method is explained, and illustrated on the handkerchief-distributor example. The technique is not fully mature, yet sufficiently developed to prove its value on a few examples (subsection 5.9.3). Open issues are discussed in subsection 5.9.4. Note that completely uniform objects, like balloons, remain out of reach for the system, and should be treated with techniques based on contours [Cyr and Kimia 2001, Selinger and Nelson 1999].

5.9.2 Approach

The goal is to somehow 'match' uniform areas between the model and the test image. Unfortunately these contain very little useful information: besides their color, only the shape (contour) is available. However, the latter is difficult to extract accurately under large viewpoint changes, and hard to exploit in the presence of partial occlusions.

It is only natural to propose a solution along the lines of the successful propagation paradigm. Like in the previous sections, regular region matches play the role of supports, while this time the candidates are uniformly colored regions automatically extracted in the model image. We now give a brief sketch of the approach. Each phase is explained in more detail afterwards.

---

*This is a "*" section.
1. **Extract uniform regions.** Uniformly colored regions are extracted in the model image by a color segmentation algorithm.

2. **Match regular regions.** The usual small, textured, viewpoint invariant regions are matched between the model and the test image.

3. **Propagate uniform regions.** Much like in the usual propagation mechanism, the affine transformations of matched regular regions are used to predict the location and shape of the uniform regions in the test image. A pixel-wise verification step follows and allows to score the propagation attempts.

1. **Extract uniform regions**

The model image is first Gaussian filtered, in order to smooth out small irregularities in the uniform image areas. Next, uniform regions are grown starting from a set of seeds, which lie on a grid homogeneously sampling the image.

The region growing process is started from each seed, one at the time, and consists of a flood-fill algorithm. After initializing the output region to contain only the seed pixel, the algorithm recursively inspects the 4 neighbors, adding a pixel to the region if its color is roughly the same as the seed's. More precisely, the two colors must be closer than 50 in Euclidean RGB space. Upon completion of a growing process, the region is kept if larger than 3000 pixels. Pixels assigned to such regions are not reused when growing from the other seeds. Each region is stored as the set of composing pixels and their mean color (region color from now on).

Note that in typical model images, a large uniform region is detected on the background. This is discarded as it is known not to belong to the object.

This simple approach effectively detects most of the significant uniform regions of the model image. Note that the exact shape of the produced regions might depend on the order in which the seeds are processed (in our case, the lexicographic order), especially when the image contains shaded areas. However, this is irrelevant in our context, as our modest goal is not to segment the image in some semantically optimal way, but simply to quickly get some approximate partitioning of sufficiently large untextured areas. This is sufficient for our recognition purposes.

The algorithm extracts three regions from the model image of the handkerchief-distributor (figure 5.19b).

2. **Match regular regions**

---

9Recall that in this thesis the pixels colorbands range in [0,255].
5.9. Sparsely textured objects*

Figure 5.20: The uniform region is propagated to the test image (b) via the affine transformation $A$ of the support region on the CWS logo.

The usual textured affine invariant regions are extracted in both the model and the test images. The experiments in the next subsection used the extractor of [Tuytelbauers and Van-Gool 2000], but any other will do. Matching proceeds in the standard way, as described in section 2.5.

Figure 5.20b shows a test image containing the handkerchief-distributor. It appears very slanted, rotated and considerably larger than in the model image. Four matches, one correct, are produced.

3. Propagate uniform regions

After passing the test image through a Gaussian filter, the following scheme is applied for each uniform region $U_m$ from point 1.

In turn, each regular match $R$ from point 2 acts as support and attempts to propagate $U_m$:

1. Project every pixel of $U_m$ to the test image via the affine transformation $A$ of $R$.

2. For each projected pixel it is verified if its color in the test image roughly matches $U_m$'s color $U_c$. More precisely, the pixel color $P_c$ must fulfill:

$$\text{dist}(P_c, s \cdot U_c) < s \cdot 50$$

where $s$ is the brightness scale factor change of $R$ between the model and test images.

This approach is very helpful as it adjusts the expected pixel color to the brightness change observed in the support. Moreover, the tolerance threshold also varies with the expected brightness change. This enhances reliability,
because pixels surrounding the support will only be positively verified if they exhibit $U_m$'s hue and follow the support's brightness change. This way, an incorrectly matched support has reduced chances of producing a high scoring propagation attempt. Moreover, the threshold also varies with $s$, thus preventing easy mismatching of dark areas.

3. The number of positively verified pixels (area of the accepted region) is the score of the propagation attempt.

The propagation attempt with the highest score is kept. If its positively verified area is more than 500 pixels, then $U_m$ is considered successfully propagated to the test image.

Much like in the usual propagation, the competition among the supports effectively limits the damage caused by mismatches. A remarkable difference is that only a portion of a candidate region $U_m$ is typically propagated. Another, less important difference, is that the propagated uniform region is not in an exact, 1-to-1 point correspondence with $U_m$ throughout its whole area. The affine transformation of the support match might be inaccurate for distant points of a large uniform region. Moreover, $U_m$ could even be non-planar. Nevertheless, this is not crucial: for recognition it is only relevant to know if and how much of $U_m$ is also present in the test image. This information is conveyed by the very fact that the region has been propagated, and by its area.

Figure 5.20 shows how a correct support match on the CWS logo successfully propagates a nearby whitish uniform region of the handkerchief-distributor. This realizes the ideas of the first subsection: it is possible to find correspondences between uniform areas, by associating them to regular textured regions. Such a uniform region correspondence then becomes a relevant cue, indicating the presence of the object.

### 5.9.3 Examples

Besides the handkerchief-distributor, we present here two additional examples.

**Maya**

The cover of the manual of the popular computer graphics software Maya features two small logos and a short text over three main uniform areas (figure 5.21). In the test image (figure 5.22) the book is imaged from a very different viewpoint, is partially occluded and surrounded by clutter. The textured parts (logos and text) are quite distinctive, and 9 (of 14) correct regular region matches are found. However, 9 matches represent a somewhat weak indication for a solid final recognition decision. Fortunately, the method of this section manages to 'match' all 3 uniform regions,
5.9. Sparsely textured objects

Figure 5.21: Maya. a) Model image. b) The three principal uniform regions have been successfully detected.

Based on the affine transformation of 3 previously matched supports (figure 5.22). The resulting set of matches, considered as the union of regular and uniform regions, completely covers the object, and therefore constitutes much more complete evidence than regular matches alone.

It is interesting to notice the case of figure 5.22c. Part of the black uniform region on the bottom of the book is propagated over the occluding fluorescent-marker, which is also mostly black. This example shows the limits of the approach: when an area with the expected color lies where the geometric transformation of a support predicts it, its pixels will be (erroneously) positively verified. While in this case this does no harm for the recognition purpose, problems might arise when mismatched supports accidentally map to an appropriately colored area.

Pokey

The last example is a smiley-puppet named Pokey. It is mostly just yellow, but for two black elliptical eyes and a curved mouth (figure 5.23). The zones around the eyes and mouth provide the 'textured' parts for obtaining regular region matches. As depicted in figure 5.23, the test image is heavily cluttered and Pokey is half hidden behind various objects. The main uniform region extracted by the segmentation phase, the yellow body, is correctly propagated thanks to a correct match on the right eye (figure 5.23b). The total matched surface now more clearly indicates the presence of the object. However, another uniform region, on the mouth, is mispropagated to the black area on the left of the test image, because a mismatched support accidentally mapped it there. Although this clearly shows the limits of the proposed approach, it is important to recall the highly challenging task (to match uniform regions, which have very little intrinsic information). How reliably this can be achieved, is still an open question.
5.9. Open issues

The approach for sparsely textured objects presented in this section still needs some research before being consistently integrated in the object recognition system. The main open issues are:

- How to integrate the propagated uniform regions with regular matches? The two sources of information need to be combined into a single, meaningful total recognition score. How much should uniform regions weigh, relative to regular regions? Should it depend on the size? A satisfactory scheme is yet to be found.

- Larger scale experiments are needed to assess the degree of reliability of propagated uniform regions. In which cases, and how frequently, significantly large areas are mispropagated because a mismatched support accidentally happens to map to an appropriately colored surface?
5.9. Sparsely textured objects

Figure 5.23: Pokey. a) The model image, with a uniform region on the yellow body. b) The body is correctly propagated to the test image, thanks to the support on the eye. c+d) The mouth is erroneously propagated to the black area on the left.

- The current geometric propagation model involves one affine transformation, from a single support match. When the candidate uniform region is large, or significantly curved, this is inaccurate and might incorrectly alter the score of the propagation attempt. It would be interesting to use several affine transformations, depending on the pixel being verified. For example, the closest support to the pixel could be taken. In this fashion, the propagation process could dynamically 'bend', according to the surface orientation embedded in the support's geometric transformation. As a major downside, this idea assumes several correct supports within the candidate region, an assumption failing exactly in the relevant cases, where nearly the whole object is uniform.
5.10 Related work

Our approach belongs to the category of OR based on local appearance, which is introduced in section 2.6. Since it extends any OR approach which matches affine invariant regions between images, it is strongly related to this class of methods (defined in chapter 2). The novelties and improvements brought by our approach are enumerated in the introduction section 5.1 and demonstrated in the result section 5.8.

Beyond the realm of local invariant features, there are a few works which are related to ours, in that they also combine recognition with segmentation. Leibe and Schiele [Leibe and Schiele 2003] present a method to detect an unknown object instance of a given category and segment it from a test image. The category (e.g. 'cows') is learned from example instances (images of particular cows). However, the method does not support changes in camera viewpoint, orientation or scale. In [Yu et al. 2002], low-level grouping cues based edge responses, high-level cues from a part detector and spatial consistency of detected parts, are combined in a graph partitioning framework. The scheme is shown to recognize and segment a human body in a cluttered image. However, the part detectors need a considerable number of training examples, and the very 'parts' to be learned are manually indicated ('head', 'left arm', etc.). Moreover, there is no viewpoint, orientation or scale invariance.

While we believe our approach to be essentially new, some components are clearly related to earlier research. The filter in section 5.6 is constructed around the sidedness constraint, which has been studied by Carlsson [Carlsson 1996]. A similar constraint, testing the cyclic ordering of points, was used in wide-baseline matching in [Tell and Carlsson 2002]. Moreover, the 'propagation attempt' at the heart of the expansion phases is an evolution of the idea of 'growing matches' as proposed by [Pritchett and Zisserman 1998, Schaffalitzky and Zisserman 2002b, Schaffalitzky and Zisserman 2002a].

5.11 Conclusion and outlook

We have presented an approach to object recognition capable of solving very challenging cases. Its power roots in the 'image exploration' technique. Every single correct match can lead to the generation of many correct matches covering the smooth surface on which it lies, even when starting from an overwhelming majority of mismatches. Hence, the method can boost the performance of any algorithm which provides affine invariant regions correspondences, because very few correct initial matches suffice for reliable recognition. Moreover, the approximate boundaries of the object are found during the recognition process, and non-rigid deformations are explicitly taken into account, two features lacking in competing approaches

The approach has been published in [Ferrari et al. 2004c], and the application to video-retrieval in [Ferrari et al. 2004b].

Although we believe the points mentioned above to be a significant improvement over existing methods, our approach is not without drawbacks. In the following, we discuss some of these.

**Computational expense**

The method requires more computation than traditional schemes based only on region extraction and matching. In the current implementation, a 2.4 Ghz computer takes about 4-5 minutes, in average, to process a pair of model and test images. Although we plan a number of efficiency enhancements to reduce this figure, the method is unlikely to reach the excellent speed of other systems (in particular the system of Lowe [Lowe 2001, Lowe 2004] which is reported to perform recognition within seconds).

**The loss of indexing**

Our approach abandons the paradigm where only the descriptor vectors of the regions are compared. Since such a paradigm enables recognition by indexing (section 2.5), regions of the test image can be very efficiently matched simultaneously to training images of all modeled objects. Afterwards, the decision is typically taken very quickly by counting the number of matches to each model and possibly applying a final geometric verification stage. In contrast, our scheme compares image parts outside the initially extracted regions, and therefore it requires direct evaluation of pairs of model and test image. Indexing could be used to substantially boost the initial matching, but the method then still resorts to pairwise image comparison in the exploration phase.

**Uniform objects**

Much like the other OR schemes based on invariant regions, ours is best suited for very textured objects. Objects with only partial texture can often still be recognized, when a sufficient number of matches covers the textured part. In section 5.9 we have presented an early attempt to extend the system to objects with only a few textured spots, spread over much larger uniformly colored surfaces. Nevertheless, this is not fully mature yet. More importantly, purely uniform objects (e.g. a coffee cup) cannot be dealt with and seem out of the reach of this kind of approaches. They should be addressed by techniques based on global contours [Cyr and Kimia 2001]. Alternatively, for geometrically complex uniform objects, like bicycles, the approach of [Selinger and Nelson 1999] or the recent method of Mikolajczyk et al., working with local edge-based regions, could be applied [Mikolajczyk et al. 2003].
Independent use of multiple model views

The method of this chapter uses the model views independently, regardless of the fact that there might be significant overlap among the parts of the objects they see. In chapter 7 we present a method for exploiting the relationships between multiple model views, for further improving recognition performance.
6

Geometric grouping

6.1 Introduction

6.1.1 Two kinds of grouping

Besides the traditional matching techniques reviewed in section 2.5, also the region tracking, multiple-view matching and image exploration methods presented in the previous chapters are sources of region correspondences between two widely separated views. In the case of tracking, these would be two distant frames of the video.

In this chapter, we turn our attention to a higher-level task: grouping such two-view matches according to some geometric criteria.

We focus on two kinds of groupings. The first kind, coplanar grouping aims at finding coplanar groups. Each group corresponds to a distinct physical plane, includes all regions lying on that plane, but no other region. Formally, all regions within a class are coplanar, while no two regions from different classes are. The method proposed in this chapter is completely automatic: neither the size nor the number of classes has to be given as input. Moreover, it works well in the presence of perspective distortions, discontiguous planar surfaces and considerable amounts of measurement noise. Planes can be in any relative position, including parallel and incident.

Automatically detecting planar structures is useful in a number of areas. In 3D reconstruction planes play an important role, especially when buildings are present and in indoor scenes. Parallax-based scene analysis methods [Irani et al. 1998, Criminisi et al. 1998] and some special reconstruction techniques [Rother and Carlsson 2001] need a reference plane. Navigation systems need to find free floor space.

In the second kind of grouping, a group is coined group of aggregated matches (GAM). A GAM is a set of region matches between two images, which are distributed over a smooth surface of the object/scene. The GAM idea is similar in
spirit to the work of Selinger and Nelson [Selinger and Nelson 1999], who advocate the benefits of an intermediate perceptual grouping level between primitives and views. Unlike in their work, here the primitives being grouped are region matches, rather than contour fragments. Moreover, GAMs are inherently a two-view concept, whereas contour fragments are defined in individual views.

A set of matches, including an arbitrary amount of mismatches, can be partitioned into GAMs. The obtained GAMs have two fundamental properties. First, a GAM is most often 'pure', i.e.: composed only of correct matches or only of mismatches. Secondly, the number of matches in a GAM relates to its probability of being correct. If a GAM is composed of many region matches (typically more than 5-10), it is very probably correct, whereas if it has only a few, it is usually incorrect. When evaluating the correctness and structure of sets of matches, it is convenient to reason at the higher perceptual grouping level that GAMs offer: no longer consider unrelated region matches, but the collection of GAMs instead. Hence, GAMs become the atomic unit, and their size is a precious piece of information. Moreover, the computational complexity of a problem can be reduced, because there are considerably fewer relevant GAMs than region matches.

Noticing the differences between the two kinds of groupings brings a better insight in their nature and function. As a first difference, GAMs can collect regions covering a planar, but also a curved or smoothly deformed surface. Although this is geometrically more general than coplanar grouping, it does not fulfill its aims. As a second difference, in coplanar grouping the geometric interpretation of classes is everything: the final goal is to cluster regions lying on the same plane. In GAMs instead, what matters are the two fundamental properties, because GAMs are meant as an intermediate representation to facilitate the operation of higher-level algorithms. The two fundamental properties impose a preliminary form of order on an otherwise chaotic large set of unreliable matches. Special GAM-based algorithms can be designed to take advantage of this order. For example, GAMs are useful in the context of robust wide baseline stereo matching, allowing the design of a two-view filter which is largely insensitive to the percentage of mismatches (section 6.7). Another application is the integration of multiple-model views for object recognition, which will be discussed in the next chapter. The geometric interpretation of GAMs as smooth surfaces is secondary, and is an elegant by-product rather than a crucial feature.
6.1.2 Outline of the chapter

The two kinds of grouping are algorithmically independent.

Coplanar grouping

For coplanar grouping, a coplanarity score is computed for every pair of region matches (section 6.2). The score, which combines a motion and a texture cue, provides information about the possible coplanarity of two regions. All scores are then used to weigh a graph, whose partitioning yields the coplanarity classes (section 6.3). The partitioning algorithm is time-efficient and robust to considerable amounts of uncertainty in the pairwise scores. It is also demonstrated on a texture segmentation application (section 6.4). Results for both coplanar grouping and texture segmentation are shown in section 6.5.

GAMs

Section 6.6 presents an efficient algorithm to partition sets of region matches into GAMs. The produced GAMs are qualitative, in the sense that they closely approximate the two fundamental properties. In section 6.7, we propose a very robust GAM-based two-view filter, capable of recovering correct matches in sets containing up to 96% mismatches. The quality of the GAM decomposition and the two-view filter are evaluated in section 6.8. Another application of GAMs, multiple-view integration for object recognition, will be the subject of the next chapter.

Section 6.10 concludes the chapter with a summary and an outlook. Sections marked with 3 are joint work with Geert Caenen and Alexey Zalesny. More details can be found in [Caenen et al. 2002, Zalesny et al. 2002].

6.2 Coplanarity cues and score

Given a set of region matches between two views, we want to partition it into disjoint coplanarity classes. Each class corresponds to a distinct plane in 3D space and all regions in it are coplanar.

The basic grouping unit is the pair: with the affine region model providing three independent coplanar points per region, a pair of regions is the smallest set sufficient for considering general planar motions (homographies). Hence, from now on, perspective effects are fully taken into account. This is important, since the affine approximation is only valid on a local scale.

In a first step a coplanarity score is assigned to each pair of regions (this section), based on a motion and a texture cue computed from the point correspondences embedded in the matches. This score summarizes the cues and conveys information about the probability that the two regions lie on the same plane.
All scores are used to weigh the edges of a graph, which is then partitioned by an effective \textit{Clique Partitioning} approximation algorithm (section 6.3), yielding the coplanarity classes. The Clique Partitioning formulation allows to exploit the intrinsic transitivity of the coplanarity property to resolve grouping ambiguities arising from noisy scores. It offers an elegant approach for the treatment of general grouping problems. We introduce a simple, but very fast and effective, polynomial time heuristic for its solution.

The input region matches can come from any source. In particular, region tracking over an image sequence, conventional two-view matching, and the image-exploration matching of chapter 5, are all valid inputs, as demonstrated in the result section 6.5.2.

\subsection{Notation}

The input consists of region matches between two images, and the basic grouping unit is a pair thereof. Let \( R, S \) denote any two region matches. Consider three points \( r_1^i, r_2^i, r_3^i \) characterizing \( R \) in image \( i \), with \( i \in \{1, 2\} \). If \( R \) is parallelogram-shaped, these are three corners (all but \( q \), figure 2.4). In the elliptical case, these are the center \( c \) and two points on the contour. Note that these characteristic points have previously been put in correspondence by the matching algorithm. \( \{r_i^p\}_{p=1..3} \) completely define \( R \) in image \( i \) and the correspondences between \( \{r_i^p\} \) and \( \{r_j^p\} \) implicitly encode the affine transformation of \( R \) between the image. We assume analogous definitions for \( S \).

Next, we introduce two numerical coplanarity cues that will later be integrated into a single coplanarity score.

\subsection{Motion cue}

The first cue is purely based on the motion of \( R \) and \( S \) between the first and the second image. We compute by least squares approximation the 2D homography \( H \) that best maps the 6 points in the first image (the set \( \{r_1^p\}_{p=1..3} \cup \{s_1^p\}_{p=1..3} \)) to their corresponding points in the second image (the set \( \{r_2^p\}_{p=1..3} \cup \{s_2^p\}_{p=1..3} \)). If \( R, S \) are coplanar, \( H \) correctly describes the motion of both regions. We measure this via the mean of the projection errors:

\[ c_m = \frac{1}{6} \sum_{p=1}^{3} (d(Hr_1^p, r_2^p) + d(Hs_1^p, s_2^p)) \]  

(6.1)

where \( d(p_1, p_2) \) is the Euclidean distance between points \( p_1, p_2 \). Assuming noise-free data, if \( R, S \) are coplanar then \( c_m = 0 \); \( c_m \) is related to the difference between
6.2. Coplanarity cues and score

The position and orientation of the $R$ plane and the $S$ plane in 3D space. Hence we use expression (6.1) as a cue about the potential coplanarity of two regions: the smaller $c_m$, the higher the chances of $R, S$ being coplanar.

6.2.3 Texture cue

While the motion cue is based completely on local information, the second cue takes a larger view and considers the image data between $R$ and $S$. The idea is to check if $R, S$ are coplanar and located on a contiguous, unoccluded physical planar surface in 3D space. In order to take into account a small, but representative, sample of the surface between $R$ and $S$ we consider three lines connecting corresponding characteristic points of the two regions. To keep the notation simple, we restrict the explanation to only one line.

Consider the line $l_1$ connecting the first characteristic points $r_1, s_1$ of the two regions in the first image. Divide $l_1$ in $s = \frac{d(r_1, s_1)}{m}$ segments of equal length $m$, denote them $\{l_1^i\}_{j=1..s}$. Let $\{l_2^j\}_{j=1..s}$ be a list of segments in the second image, whose coordinates are obtained by projecting $\{l_1^i\}_{j=1..s}$ via $H$ (defined above). We are interested in the similarity between corresponding segments in the two frames, and in particular in the least similar one:

$$\min_{j=1..s} \text{NCC}(l_1^i, l_2^j) \quad (6.2)$$

where NCC($l_1^i$, $l_2^j$) is the value of the normalized cross-correlation of the intensity profile on line segment $l_1^i$ with the one on $l_2^j$. Only if $R, S$ are coplanar and located on a contiguous, unoccluded planar surface, will all segments score well. If $R, S$ are not coplanar, segments close to the region may still score well: $H$ describes the motion in that zone best, and probably the neighboring area is planar. Nevertheless, central segments will tend to be misaligned as $H$ can not correctly describe their motion, and therefore have low scores (figure 6.1). Taking the least scoring segment ensures the detection of exactly those significant cases.

We define the second coplanarity cue $c_t$ as the average of expression (6.2) over the three lines connecting $r_1^1$ with $s_1^1$, $r_1^2$ with $s_1^2$ and $r_1^3$ with $s_1^3$. Coplanar pairs located on a discontinuous planar surface (e.g.: the surface is interrupted between the two regions) will tend to have a low $c_t$: clearly, this should not be interpreted as an indication that two regions are not coplanar. Hence, we use $c_t$ only to increment the total coplanarity score. Nevertheless the role of this cue must not be underestimated, as it can substantially reinforce the total score of a significant portion of the coplanar pairs, hence helping the grouping algorithm (next section).
6.3. Clique Partitioning approximation

6.3.1 Grouping as Clique Partitioning

Taken individually, the coplanarity scores are unreliable because they arise from very limited, noisy information. In practice it happens that a coplanar pair has \( w < 0 \) (false negative), or the contrary (false positive). Nevertheless, taken altogether, the scores carry reliable information about the correct grouping. We want to be robust to misleading local information by exploiting the transitivity of coplanarity: if \( R, S \) are coplanar and \( S, T \) too, then \( R, T \) must be coplanar\(^1\). How can transitivity help

\(^1\)Coplanarity is reflexive, symmetric and transitive (an equivalence relation).
**6.3. Cliques Partitioning approximation**

Consider a scene with three regions. Let $w_{ij}$ be the score of the pair $(i,j)$ composed of the $i$th and $j$th region. Given the scores $w_{12} = 9$, $w_{13} = 7$, $w_{23} = -3$, and the transitivity property, the best choice is to group the three regions together ($w_{23}$ is a false negative score). Next, we formulate the coplanar grouping problem so as to exploit transitivity to detect and avoid false scores.

We construct a complete graph $G$ where each vertex represents a region and edges are weighted with the coplanarity scores. We partition $G$ into completely connected disjoint subsets of vertices (cliques) so as to maximize the total score on the remaining edges (Clique Partitioning, or CP). The transitivity property is ensured by the clique constraints: every two vertices in a clique are connected, and no two vertices from different cliques are connected. Hence, the generated cliques correspond to the best possible coplanar grouping (given the cues).

The CP formulation of coplanar grouping is made possible by the presence of positive and negative weights: they naturally lead to the definition of a best solution without the need of knowing the number of cliques (planes) or introducing any artificial stopping criteria like in other graph-based approaches based on strictly positive weights [Shi and Malik 1997, Aslam et al. 2000]. On the other hand, our approach needs a parameter $h_t$ that determines the splitting point between positive and negative scores. But, in our context, this parameter is easily determined and experiments show the optimal solution of CP to be generated for a wide range of $h_t$ (section 6.5).

CP can be solved by Linear Programming [Graham et al. 1995] (LP). Let $w_{ij}$ be the weight of the edge connecting $(i,j)$, and $x_{ij} \in \{0,1\}$ indicate whether the edge exists in the solution. The following LP can be established:

\[
\begin{align*}
\text{maximize} & \quad \sum_{1 \leq i < j \leq n} w_{ij} x_{ij} \\
\text{subject to} & \quad x_{ij} + x_{jk} - x_{ik} \leq 1, \quad \forall 1 \leq i < j < k \leq n \\
& \quad x_{ij} - x_{jk} + x_{ik} \leq 1, \quad \forall 1 \leq i < j < k \leq n \\
& \quad -x_{ij} + x_{jk} + x_{ik} \leq 1, \quad \forall 1 \leq i < j < k \leq n \\
& \quad x_{ij} \in \{0,1\}, \quad \forall 1 \leq i < j \leq n
\end{align*}
\]

The inequalities express the clique constraints (transitivity), while the objective function to be maximized corresponds to the sum of the intra-clique edges.

### 6.3.2 A fast approximation

Unfortunately CP is a NP-hard problem [Graham et al. 1995]: LP (6.4) has worst case exponential complexity in the number $n$ of vertices (regions), making it impractical for large $n$. The challenge is to find a practical way out of this complexity trap. The correct partitioning of the example in figure 6.2 is $\{\{1,3\}, \{2,4,5\}\}$. A
simple greedy strategy merging two vertices \((i, j)\) if \(w_{ij} > 0\) fails because it merges \((1, 2)\) as its first move. Such approach suffers from two problems: the generated solution depends on the order by which vertices are processed and it looks only at local information.

We propose to do better than the simple greedy strategy and approximate Clique Partitioning with the following polynomial-time iterative heuristic. The algorithm starts with the partition

\[
\Phi = \{\{i\}\}_{1 \leq i \leq n}
\]

composed of \(n\) singleton cliques each containing a different vertex. The function

\[
m(c_1, c_2) = \sum_{i \in c_1, j \in c_2} w_{ij}
\]

defines the cost of merging cliques \(c_1, c_2\). We consider the functions

\[
\begin{align*}
    b(c) &= \max_{t \in \Phi} m(c, t) \\
    d(c) &= \arg \max_{t \in \Phi} m(c, t)
\end{align*}
\]  

representing, respectively, the score of the best merging choice for clique \(c\) and the associated clique to merge with. We merge cliques \(c_i, c_j\) if and only if the three following conditions are met simultaneously

\[
d(c_i) = c_j, \quad d(c_j) = c_i, \quad b(c_i) = b(c_j) > 0
\]  

In other words, two cliques are merged only if each one represents the best merging option for the other and if merging them increases the total score. At each iteration the functions \(b(c), d(c)\) are computed, and all pairs of cliques fulfilling the criteria are merged. The algorithm iterates until no two cliques can be merged.

At each iteration, the function \(m\) can be progressively computed from its values in the previous iteration. The basic observation is that for any pair of merged cliques \(c_k = c_i \cup c_j\), the function changes to

\[
m(c_i, c_k) = m(c_i, c_j) + m(c_i, c)\]

for all \(c_i \not\in \{c_i, c_j\}\). This strongly reduces the amount of operations needed to compute \(m\).

Figure 6.2 shows an interesting case. In the first iteration \(\{1\}\) is merged with \(\{3\}\) and \(\{4\}\) with \(\{5\}\). Notice how \(\{2\}\) is, correctly, not merged with \(\{1\}\) even though \(m(\{1\}, \{2\}) = 3 > 0\). In the second iteration \(\{2\}\) is correctly merged with \(\{4, 5\}\), resisting the (false) attraction of \(\{1, 3\}\) \(\{b(\{1, 3\}, \{2\}) = 1, d(\{1, 3\}) = \{2\}\}\). The algorithm terminates after the third iteration because \(m(\{1, 3\}, \{2, 4, 5\}) = -3 < 0\). The second iteration shows the power of CP. Vertex 2 is connected to ill-signed edges \((w_{12}\) is false positive, \(w_{25}\) is false negative). Given vertices \(\{1, 2, 3\}\) only, it
6.4. Texture segmentation

is not possible to derive the correct partitioning \( \{\{1, 3\}, \{2\}\} \); but, as we consider also vertices \( \{4, 5\} \), the global information increases and CP arrives at the correct partitioning.

The proposed heuristic is order independent, takes a more global view than a direct greedy strategy, and resolves several ambiguous situations while maintaining polynomial complexity. Analysis reveals that the exact amount of operations depends on the structure of the data, but it is about \( 3n^2 \) in the average case (appendix A reports the full analysis). Moreover, the operations are simple: only comparisons and sums of real values (no multiplication or division is involved).

In the first iterations, being biased toward very positive weights, the algorithm risks to take wrong merging decisions. Nevertheless our particular merging criterion ensures this risk to quickly diminish with the size of the cliques in the correct solution (number of regions in a plane) and at each iteration, as the cliques grow and increase their resistance against spurious weights. Moreover, in our application, very positive scores arise only when both cues score well and are therefore much more reliable than negative scores, which are often due to large homography errors due to measurement noise. In summary, the algorithm uses reliable data as seeds, and then proceeds to the robust construction of the correct solution by filtering out spurious data. The performance of the algorithm is addressed in section 6.5, which analyzes its speed, the quality by which it approximates CP, and the amount of noise on the graph’s edges it can withstand.

6.4 Texture segmentation

The Clique Partitioning approximation algorithm is not restricted to the application of coplanar grouping. It can be used for other problems where pairwise compatibly
values between the entities to be grouped can be calculated. In this section, we
tackle one such problem, texture segmentation. Given an image containing multi¬
ple textures, like figure 6.3, we would like to segment it into areas with the same
texture (in this case the flowerbed, the mountain, the trees and the sky). More
formally, we wish to partition the set of pixels in subsets each containing all pix¬
els of a texture, and no pixels of another texture. This is useful in the context of
texture synthesis, where the spatial layout of a composite texture could be learned
automatically [Caenen et al. 2002], and then used as basis to generate novel, visually
similar instances [Zalesny et al. 2002].

How to apply the CP approximation algorithm to achieve texture segmentation ?
Just like in the coplanar grouping application regions are grouped into planes, here
pixels are grouped into textures. Hence, the idea is to build a graph where each
vertex represents a pixel and each edge is weighted by a similarity score between
a pair of pixels. Once such graph is built, it can be fed to the CP approximation
algorithm, which will yield the intended texture segmentation, with each output
clique corresponding to a different texture. The similarity score between two pixels
$p_i, p_j$ should quantify the similarity between the textures at $p_i, p_j$. In practice, for
each pixel we summarize in a histogram the color and structure (Haar-wavelets) of
a small circular neighborhood around it. The Bhattacharyya coefficient [Comaniciu
et al. 2000] is then used to compare the histograms of a pair of pixels.

Unfortunately, directly comparing the histograms leads to problems when a pixel is
near the border between two textures (figure 6.4). In this case, the histogram of the
pixel mixes information from two different textures, and therefore it is quite different
from the histograms of pixels from either texture. As a result, the boundaries could
be poorly located, and spurious intermediate texture classes covering borders could
be produced by the segmentation algorithm. In order to avoid this problem, the
6.4. Texture segmentation

Figure 6.4: Left: direct comparison between two pixels. The dashed circles indicate the support of the histograms. Right: comparison with shifts, avoiding problems near texture borders.

The histogram of the first pixel is compared to those of several pixels in a neighborhood of the second, searching for the best matching histogram (highest Bhattacharyya coefficient). The search for this location is efficiently carried out by the mean-shift algorithm [Comaniciu et al. 2000], thus avoiding exhaustive search. The search is also performed over different scales, to cater for perspective effects and the like that may occur within a texture (e.g.: the flowerbed in figure 6.3).

To reduce computational cost, the similarity scores are only computed for a subsample of the image pixels (on a regular grid). After segmenting this sample, a full resolution segmentation map is obtained by assigning each pixel to the texture class of the neighboring sample pixel with the most similar histogram. Since there are still about 1000 pixels in the subsample, the time-efficient approximation of CP proposed in the previous section is crucial. As experimentally demonstrated in section 6.5, our algorithm can deal with problems of this size in only a few seconds, while the full solution cannot be computed, because of its exponential complexity.

CP needs positive and negative edge weights (similarity scores), indicating that pixels are likely to belong together or the opposite. The absolute value of the weight is a measure of confidence. Hence, we subtract a predefined constant value from the Bhattacharyya coefficient, which initially ranges in [0, 1] (the value is kept the same in all experiments). It is important to notice that this fixed threshold allows to correctly segment various images, with different, and unknown, numbers of textures. The number of textures is not given to the system, just like the number of planes in coplanar grouping isn’t. Instead, this would be required by the popular k-means clustering algorithm. Actually, our threshold is a form of 'perceptual scale' factor, and allows the user to express what should be considered to be perceptually similar, and therefore belong to the same texture class. Several example results are reported in subsection 6.5.3.

Since the texture segmentation scheme is marginal to this thesis, it has been only quickly summarized here. More detail can be found in [Caenen et al. 2002].
6.5 Results for coplanar grouping and texture segmentation

6.5.1 Performance of the CP approximation

When considering the heuristic approximation to Clique Partitioning, three questions are especially relevant:

- how much noise on the edge weights (i.e. uncertainty on the similarity scores) can it withstand?
- how well does it approach the true solution of CP (obtained by solving LP (6.4))?
- how fast is it?

We tested both LP and the heuristic on random instances of the CP problem. Graphs with a priori known, correct partitioning were generated. Their sizes differed in that both the number of cliques and the total number of vertices were varied (all cliques had the same size). Intra-clique weights were uniformly distributed in $[-a, 9]$ with a real number, while inter-clique weights were uniformly distributed in $[-9, a]$, yielding an ill-signed edge percentage of $\frac{a}{a+9}$. This noise level could be controlled by varying the parameter $a$. Let the difference between two partitionings be the minimum number of vertices that should change their clique membership in one partitioning to get the other. For example, the difference between $\{\{A, B\}, \{C, D, E\}, \{F, G\}\}$ and $\{\{A, C\}, \{D, E\}, \{B, F, G\}\}$ is 2, because $C$ should be moved to the second clique, and $B$ to the first, to make the two partitionings equal. The quality of the produced partitionings is evaluated in terms of average percentage of misclassified vertices: the difference between the produced partitioning and the correct one, averaged over 100 instances and divided by the total number of vertices in a single instance.

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Cliques</th>
<th>Diff %</th>
<th>Err % LP</th>
<th>Err % Approx</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>3</td>
<td>0.53</td>
<td>6.8</td>
<td>6.93</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0.5</td>
<td>2.92</td>
<td>3.08</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>0.05</td>
<td>2.19</td>
<td>2.14</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>0.2</td>
<td>1.13</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Table 6.1: Comparison of LP and our approximation. The noise level is 25%. Diff % is the average percentual difference between the partitionings produced by the two algorithms. The two Err columns report the average percentage misclassified vertices for each algorithm.
6.5. Results for coplanar grouping and texture segmentation

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Cliques</th>
<th>Noise level</th>
<th>Err % Approx</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>4</td>
<td>25</td>
<td>0.33</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>25</td>
<td>0.1</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>33</td>
<td>2.1</td>
</tr>
<tr>
<td>120</td>
<td>5</td>
<td>36</td>
<td>1.6</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>40</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 6.2: Performance of the CP approximation algorithm on various problem sizes.

Table 6.1 shows a comparison between our approximation to CP and the optimal solution computed by LP on various problem sizes, with constant noise level set to 25% ($a = 3$). In all cases the partitionings produced by the two algorithms are virtually identical: the average percentual difference is very small as shown in the third column of the table. Due to the very high computational demands posed by LP, the largest problem reported here has only 24 vertices. Beyond that point, computation times run into the hours, which we consider as too impractical. Note that the average percentage of misclassifications quickly drops with the size of the cliques.

Table 6.2 summarizes the performance of our approximation for larger problem sizes. Given 25% noise level, the average error already becomes negligible with clique sizes between 10 and 20 (less than 0.5%). In problems of this size, or larger, the algorithm can withstand even higher noise levels, still producing high quality solutions. In the case of 1000 vertices and 10 cliques, even with 40% noise level ($a = 6$), the algorithm produces solutions which are closer than 1% to the correct one. This case is of particular interest as its size is similar to the typical texture segmentation problems.

The heuristic algorithm is very fast: it completed these problems in less than 0.1 seconds, except for the 1000 vertices one, which took still only 4 seconds on the average. Times are measured on a modest 1.4 Ghz computer. The ability to deal with thousands of vertices is important for texture segmentation, as every pixel to be clustered will correspond to a vertex. Even in coplanar grouping, there can easily be a hundred regions, which is already out of reach for the LP solution. A complexity analysis of the approximation algorithm reveals that the number of operations is only quadratic in the number of vertices (appendix A). This, combined with the fact that the operations are very simple (comparisons and sums), explains the high speed of execution.

Figure 6.5 shows the average error for a problem with 100 vertices and 5 cliques as a function of the noise level ($a$ varies from 3 to 5.5). Although the error grows faster than linearly, and the problem has relatively small size, the algorithm produces high quality solutions in situations with as much as 36% of noise.
6.5. Results for coplanar grouping and texture segmentation

Figure 6.5: Relationship between noise level and error, for a 100 vertices, 5 cliques problem. The average percentage of misclassified vertices (Y-axis) is still low with as much as 36% noise level (X-axis).

These encouraging results show CP’s robustness to noise and support our heuristic as a good approximation. Cliques in these experiments were only given the same size to simplify the discussion. The algorithm itself can deal with differently sized cliques.

6.5.2 Coplanar grouping

The coplanar grouping techniques have been tested on several scenes. Four representative cases are discussed here. The parameters were kept the same for all experiments and set to $h_t = 2.0$ for the homography error threshold, and to 0.6 for the minimal NCC value before the texture cue is included in the total coplanarity score, as in equation (6.3).

Boxes

The first example consists of a table with many planar objects tracked over a short sequence by the method of chapter 3. 12 regions have been tracked: 3 on the Artificial Intelligence book in the middle of the scene (AI plane), 4 on the magazines lying on the table (Table plane), and 5 on the two cardboard boxes on the background (Box plane). The first and last image of the sequence (figure 6.6) are used to compute the cues. The scene presents several challenges: the Table and Box planes are not visible as contiguous planar surfaces, the Table plane shows some perspective effect, the AI plane is slanted and adjacent to the table. Moreover, the camera moves only little during the sequence, resulting in a tight baseline and therefore little motion information for our cues (parallax). However, it is the small region at the bottom of AI that makes this scene particularly challenging. Since it is located very close to the Table regions and the two planes orientations (at the local scale) are similar, its motion is compatible with the Table motion, and the coplanarity
6.5. Results for coplanar grouping and texture segmentation

Figure 6.6: First and last frame of test sequence for coplanar grouping. Notice the moderate baseline

scores are positive whereas they should be negative. Based on this local information only, we could not separate it from the Table regions.

Despite these difficulties, the algorithm produced the correct grouping, and exploited the attraction of the topmost AI regions to bring the bottom one in place. The bottom-AI region is more attracted to the top-AI regions than to the Table ones. Therefore, it is grouped with them in the first iterations. Once together, the 3 AI regions actually score negatively with the table regions. In summary, the algorithm managed to take a more global view on the data.

The correct solution for the whole scene was produced for the homography error threshold $h_t$ ranging in $[1.4, 2.3]$, which is a considerably wide interval, considering that the homography error for all pairs formed by regions in AI and Table ranges from 0.6 to 3.0. For $h_t < 1.4$ the AI plane was split (i.e.: not all AI regions were grouped together), and for $h_t > 2.3$ AI was merged with Table. As shown by figure 6.7, picking values for $h_t$ far from the optimum causes a deterioration of the coplanarity scores, resulting in an increase of false scores (both false positive and false negative scores). Nevertheless, the algorithm resisted even significant increases: for $h_t = 2.3$, 24% of the coplanarity scores are false, with the grouper still delivering the correct partitioning.

The range of $h_t$ yielding the correct solution is even wider in simpler cases: considering only the AI and Box regions, this is $h_t \in [1.4, 4.5]$. For $h_t = 4.5$, 32% of the scores are false. Again, this is a wide interval, given that the homography errors for this scene range in $[1.18, 5.22]$. These figures show Clique Partitioning’s ability to extract the correct information in the presence of considerable amounts of noise. We verified the correctness of the clique partitioning solutions provided by our heuristic
6.5. Results for coplanar grouping and texture segmentation

by solving the problem also with LP. In both cases (complete scene, only AI and Box), for all values of $h_t$, LP generated the same solutions.

Experiments on several scenes, including the 4 reported here, reinforce the observations about the $h_t$ ranges for which the correct solution is produced (more statistics are presented for the second scene). This suggests the algorithm is little sensitive to the exact choice of $h_t$. Setting $h_t = 2.0$ resulted in the correct partitioning of all reported scenes.

Maison

While the region correspondences in the previous example were obtained by tracking over an image sequence, figure 6.8 shows a case with 12 conventional two-view matches, i.e. correspondences between regions produced by an affine invariant region extractor (chapter 2). Just like tracked regions, this kind of correspondences is also a valid input for coplanar grouping. However, due to the view-independent extraction, the accuracy of the point correspondences might not always be very good. As additional challenges, the main plane is composed of two spatially disjoint parts (the left and right blocks), and it is parallel to the other plane (on the middle block).

The algorithm dealt well with the situation and correctly returned a group including all regions on the left and right blocks, and another group with regions on the middle block. The correct solution is found for a wide range of $h_t$ (any $h_t \in [1.6, 6.3]$).

Valbonne

Figure 6.9 shows coplanar grouping on two views of the Valbonne church, which belongs to the most used test imagery for wide-baseline stereo (for example in [Mikolajczyk and Schmid 2002, Schaffalitzky and Zisserman 2002b, Ferrari et al. 2003, Matas et al. 2002b]). The viewpoint change is considerable and more important than in the previous two examples. The input correspondences are 48 conventional matches, including 11 gross mismatches. The total running time of the coplanar
6.5. Results for coplanar grouping and texture segmentation

Figure 6.8: The two groups found for Maison.

grouping algorithm is about 3 seconds, nearly all used to compute the texture-cue (section 6.2). Less than 0.1 seconds are required for the heuristic CP to solve this 48-vertices case. In contrast, the full LP solution could not be computed, as it takes already more than 6 hours for a graph with 30 vertices.

The three largest groups detected by the proposed method correspond to actual scene planes. The two adjacent tower walls are correctly separated in two groups, and the lower wall on the left, which stands a little deeper, in another. Another group of two correct matches is near the lower-left corner of the tower. They are at a different depth than the leftmost wall and the ‘plane’ they lie on, somewhat between the sign and the wall, has a different orientation.

None of the 11 gross mismatches has been mixed with any of the presented planes. With the only exception of one group of two, all matches not included in the 4 presented groups have been left isolated (i.e. a singleton ‘group’ each). This is good, because it means that mismatches are not aggregated into spurious, non-existing planes. Nevertheless, it is interesting to look at a third kind of matches, in-between correct ones and gross mismatches, which, up to one exception, have also been left isolated (figure 6.9-right). These matches are roughly correctly located, but do not accurately cover the same physical surface in both views. Most of them are non-planar regions, for which the system cannot do better. Their wrong shape causes large homography errors which in turn lead to negative scores and finally isolation. For many of these cases, isolation is the right answer, like the one lying in-between the two sides at the top of the tower. In some other cases however, like the region at the bottom, there is a plane which seems natural to join, in spite of matching inaccuracy. It is an interesting, though still open, question how to include such low accuracy matches while still preserving sharp discriminative power between different planes. In only one case, one such inaccurate match is grouped with the
6.5. Results for coplanar grouping and texture segmentation

The four main groups found in Valbonne are shown in the left and middle images of Figure 6.9. The tower is grouped with the lower wall on the left. A few isolated matches are also shown on the right.

Suchard

As a last example, we consider a pair of model and test images from the object recognition chapter 5. The object is seen under two very different viewpoints, and is partially occluded in the heavily cluttered test image. This time, the matching is carried out by the image-exploration algorithm of chapter 5. 128 matches, densely covering the object, are produced and passed to the coplanar grouper (this case cannot be tackled without the heuristic CP, as it is too large for the LP solution).
6.5. Results for coplanar grouping and texture segmentation

Figure 6.10: Groups found in Suchard.

model

main two groups

some matches left isolated
The method succeeds in putting matches from each facet of the object in a separate group (figure 6.10-middle). Note how the two parts of the wider facet, divided by the occluding plush leopard, are detected as being on the same plane. Combining the image-exploration technique with the coplanar grouper in this fashion allows to actually ‘segment out’ planes, rather than only finding sparse subsets of coplanar regions.

Mismatches and particularly inaccurate matches, some located on the edge between the two facets, have been left out of the main groups (figure 6.10-bottom). In most cases, they have been isolated in singleton groups.
6.5. Results for coplanar grouping and texture segmentation

Figure 6.12: Segmentation of a composite texture.

Figure 6.13: Original image and synthesis.

6.5.3 Texture segmentation

The texture segmentation scheme was tested on both composite textures and textured landscapes. Figure 6.11 shows a well-known texture collage which was processed in three ways. In (b) histograms were compared without allowing 'shifted' matches. (c) shows the segmentation using mean shift to achieve an optimal match, which clearly solves the problems at boundaries between textures. Segmentation (d) was computed with the popular normalized cuts algorithm [Shi and Malik 1997], available on the web at http://www.cs.berkeley.edu/~doron/software/ncuts/

along with a standard set of parameters. Our method (c) performs best, by detecting the correct number of textures, and determining the covered area to a good accuracy. The source image was downloaded from

www.utia.cas.cz/user_data/haindl/virtuous/dema/demtexseg.html

Figures 6.5.3 shows the segmentation of a composite texture and figures 6.3, 6.5.3, of two landscapes. Notice how the whole flowerbed of figure 6.3 is detected as a single
6.6 Constructing GAMs

The previous sections focused on the coplanar grouping problem. It is now time to consider the second kind of grouping: GAMs ('Group of Aggregated Matches'; see the introduction section 6.1 for an informal definition).

This section describes an incremental grouping algorithm to partition a set of two-view matches into GAMs. The important design issue is to guarantee a close approximation of the ideal fundamental properties (see section 6.1).

6.6.1 Affine dissimilarity

The grouping process is driven by the similarity between the geometric (affine) transformations that map the regions from one view to the other. Consider three points on each region: the center $p_0$ and two more points $p_1, p_2$ on the boundary. These points have previously been put in correspondence by the matching algorithm. The following function measures to which degree the affine transformation of a region match $R$ is also valid for another match $Q$ (figure 6.15):

$$D(R, Q) = \frac{1}{6} \left( \sum_{i=0.2} \| A_{p_0 \rightarrow p_1} Q^i_1 - Q^i_2 \| + \sum_{i=0.2} \| A_{p_2 \rightarrow p_1} Q^i_2 - Q^i_1 \| \right)$$  (6.8)
6.6. Constructing GAMs

Figure 6.15: Affine dissimilarity. d is one term in function (6.8).

where $A_{R_a\rightarrow R_b}$ is the affine transformation mapping $R$ from view $a$ to view $b$, and $R_v^i$ is point $p_i$ of region $R$ in view $v$. By averaging over the two regions, we obtain the affine dissimilarity

$$D_A(R, Q) = \frac{1}{2}(D(R, Q) + D(Q, R))$$

between (the affine transformations of) $R$ and $Q$. This measure is symmetric in the regions and in the views. This brings stability and helps dealing fairly with large scale changes. Two region matches have a high affine dissimilarity if either is a mismatch, or if they lie on different surfaces.

6.6.2 Constructing GAMs

The matches are partitioned by the following algorithm, which starts from a single match and then grows a GAM by iteratively adding matches. The algorithm starts with the set $\Omega$ of region matches.

1. A region match is removed from $\Omega$ and used to create a new GAM $\Gamma$.

2. Search $\Omega$ for a region with affine dissimilarity to the GAM below a certain threshold \(^2\). The search proceeds from the closest to the farthest to the GAM, according to the spatial distance:

$$\sum_{R \in \Gamma} d(R_1^0, Q_1^0) / |\Gamma|$$

\(^2\)This is the only parameter, and it is set to 1/15 of the image size.
This is the average Euclidean distance (d) of a region Q to the regions composing the GAM, measured in the first view. The affine dissimilarity between a region Q and the GAM $\Gamma$ is:

$$\sum_{R \in \Gamma} w_R D_A(R, Q)$$

This is the weighted mean of the affine dissimilarities to each region in the GAM, with weights $w_R$

$$w_R = \frac{1}{d(R^0_1, Q^0_1)^2 \sum_{S \in \Gamma} \frac{1}{d(S^0_1, Q^0_1)^2}}$$

set inversely proportional to the square of the distances between the regions.

3. As soon as a suitable region is found, it is added to the GAM and the search stops. The region is removed from $\Omega$, and the algorithm iterates to 2. If no such region is found, the current GAM is closed. The algorithm goes back to 1, where a new GAM is created and then grown. The process terminates when $\Omega$ is empty.

Figure 6.16 shows an example scene (Felix). Matches A, B, C, D, E, F are distributed over the curved magazine surface, while G, I, J over the planar plate on the left of the image. Region H, covering the 'a' of 'Happy' in the left image, is mismatched to the 'a' of 'Birthday' in the right image. Note that the correct corresponding region is not visible in the right image. The algorithm starts by creating a GAM containing region A alone. In the next iteration, the nearest region B is added to the GAM, and in the subsequent iterations, regions C, D, E, F are added one at the time, in this order. No other region has a sufficiently similar affine transformation, so the GAM $\{A, B, C, D, E, F\}$ is closed and a new GAM formed by region G is started. Region I is first added. Then, the nearest region H is a mismatch and has a quite dissimilar affine transformation, so it doesn't join the GAM in the second iteration. Instead, J is picked up, and the second GAM is closed as $\{G, I, J\}$. Finally, H is put in a singleton GAM, and the construction algorithm terminates.

The algorithm groups two regions in the same GAM if they have a similar affine transformation or if there is some region with coherent intermediate affine transformation spatially located between them. In other words, the affine transformation can vary gradually from a region to the next within a GAM. Hence, a GAM can cover not only a planar, but also a curved or even a continuously deformed surface (like bending of paper or cloth). The fact that the method doesn't prescribe a fixed neighborhood area where to grow renders it capable of grouping also spatially sparse and discontiguous subsets of correct matches.
6.6. Constructing GAMs

Figure 6.16: Felix scene. Top: 9 Matches, with two clearly identifiable groups. Bottom: Close-up on match H. The region on the ‘a’ of ‘Happy’ is mismatched to ‘Birthday’. The GAM constructor successfully finds the two groups and isolates the mismatch in a third, singleton one (see text).

In principle, the composition of a produced GAM might depend on the choice of its first region in step 1. However, the near-to-far growing order and the distance-based weighting make the algorithm highly order independent. This is confirmed by experiments on various scenes, where the composition of the GAMs was stable (variations of about 1%) in spite of random permutations of the input regions.

6.6.3 Fundamental properties

The GAM decomposition has two fundamental properties:

- It is unlikely for mismatches to form large GAMs (typically of more than 5 matches). Mismatches have independent, random affine transformations, uniformly spread in the large 6D affine transformation space. Thus, the more mismatches you consider, the less likely they will respect the constructor’s criteria, that their affine transformations vary slowly and gradually from a region to the next. A set of mismatches has widely varying, inconsistent transformations. More precisely, the probability that $N$ mismatches are grouped in the same GAM is expected to decrease roughly exponentially with $N$. On
the other hand, several correct matches lying on the same surface will form a larger GAM, because of their coherent affine transformations.

- A GAM is most often composed of either only correct matches or only mismatches. The reasons again lie in the randomness of mismatches' transformations. Suppose a correct GAM is being grown, and at some iteration the algorithm has to decide if to add a nearby mismatch. It is unlikely that this will happen as the mismatch has a low chance to offer a suitable affine transformation. Even in this unfortunate case, the probability to add a second mismatch are again equally low. The total probability quickly drops with the number of added mismatches. As a result, correct GAMs are composed of correct matches only, or they contain only very few mismatches (typically 1 or 2).

As a combined effect of the two properties, mismatches are scattered over many small GAMs, while correct matches concentrate in a few larger GAMs. This brings a major advantage to organizing individual matches into GAMs: if a GAM contains many matches we know it is very probably correct. GAMs containing only a few regions are most of the time mismatches, and sometimes they are minor groups of correct matches located on a small, or difficult to match, surface.

Apart from carrying information about correctness, the sizes of GAMs correlate with relevance: the larger a GAM is, the more important it is, because it covers a larger part of the scene/object.

The above properties are confirmed to hold, even when starting from a large majority of mismatches, in the experimental result section 6.8. The two fundamental properties are the reason of existence of GAMs and enable the design of powerful new algorithms.

6.7 Two-view filtering

6.7.1 Motivation and idea

As an example of the usefulness of GAMs, this section explores their application to the problem of robust two-views matching. Another interesting application is Object Recognition from multiple model views and will be discussed in the next chapter.

Suppose we have computed a set of matches containing a large amount of mismatches, say more than 80%, and we want to tell the correct matches apart. Unfortunately, the widely used RANSAC Epipolar Geometry (RANSAC-EG) filter [Torr
and Murray 1997] performs poorly in the presence of more than about 60% mismatches. We illustrate here a GAM-based filter which can handle a much higher percentage of mismatches, and is in fact largely insensitive to it. Section 6.8 shows that the filter is capable of distinguishing correct matches in sets containing up to 96% mismatches.

By using the GAM constructor of section 6.6 we can partition the input matches into GAMs. Each GAM is most often composed of only correct matches or only mismatches. Hence, rather than deciding on which matches to keep, we can decide on which complete GAMs to keep. GAMs are seen as the new atomic units. Moreover, not all GAMs are equally important: a GAM composed of many regions (typically more than 5) is very likely to be correct. GAMs containing only a few regions are most of the times composed of mismatches, but sometimes they are minor groups of correct matches. Therefore, one can design powerful filters by relying more on the larger GAMs and using these to validate the smaller ones. Next, we introduce such a filter, and demonstrate its tolerance to extreme amounts of mismatches in section 6.8.

### 6.7.2 Filtering algorithm

The algorithm starts by constructing GAMs from the input matches. Next, the GAMs are sorted according to their size, from the ones containing the most regions, to the ones containing the least. In the first iteration, a fundamental matrix is fit to all regions in the first (largest) GAM. Then, the number of inliers to this fundamental matrix are computed, among all matches within the first GAM. The first GAM is implicitly assumed correct. In the second iteration, a new fundamental matrix is fit to the first and the second largest GAMs. If there are more inliers to this new matrix than there were before, then the second GAM is deemed correct. All iterations have this general form. In the \( k \)th iteration, a fundamental matrix is fit to all GAMs previously deemed correct plus the \( k \)th one. If the number of inliers to this fundamental matrix exceeds the maximum observed so far, the \( k \)th GAM is considered correct. The inliers are computed among the matches within all GAMs previously deemed correct, plus the \( k \)th one. The algorithm iterates until it meets the first singleton GAM. At this point, all singleton GAMs which are inliers to the latest fundamental matrix are considered correct. All region matches within all GAMs deemed correct are returned as correct matches.

The power of this simple algorithm lies in the order in which the GAMs are inspected. In the first few iterations, the algorithm is likely to meet only correct matches, and safely builds a solid fundamental matrix. In the later iterations, this helps deciding whether smaller GAMs are correct. It is important to note that erroneous decisions matter less in the later iterations, as fewer matches are at stake. Note how the atomicity of GAMs is respected: either all matches of a GAM are accepted, or none.
The method has only two parameters: the maximal distance to the epipolar line, used to determine the inliers, and the affine dissimilarity threshold used in the GAM constructor. RANSAC-EG instead requires the maximal number of iterations as one of the parameters. This is equivalent to providing the maximal ratio of mismatches, which is unknown a priori, and varies from case to case.

6.8 Results for GAMs

The GAM methods proposed in this chapter are validated on three sets of experiments. The first constructs GAMs on several pairs of images, in order to show that the fundamental properties stated in section 6.6 hold. Next, the GAM-based two-view filter is demonstrated in subsection 6.8.2. In the last subsection, examples of GAMs built from dense matches, produced by the image exploration technique of the previous chapter, are shown to nicely cover large curved and deformed surfaces.

6.8.1 Fundamental properties of GAMs

GAMs have two fundamental properties, introduced in section 6.6:

- It is unlikely for mismatches to form large GAMs. If a GAM is composed by many matches, then they are very probably correct.
- Most often, a GAM is composed of either only correct matches or only mismatches.

These properties are essential because GAMs are useful only if higher-level algorithms can rely on them. In order to assess their validity, we have run the GAM constructor algorithm on 14 pairs of images and carefully collected statistics about sizes and composition of the resulting GAMs. The image pairs come from diverse sources and contain planar, curved, as well as deformed surfaces. Seven pairs are wide-baseline stereo (WBS) cases, including three classic examples used in many papers. The other seven pairs are object recognition cases, with the first image being a model view and the second a test image. The two kinds of data differ in several aspects. The test images of the object recognition cases contain substantial parts missing in the associated model images, where the object is pictured in isolation (see previous chapter). This results in large amounts of clutter regions which disturb the matching process. Additionally, the scale change and occlusion levels are higher than in the WBS cases. On the other hand, some WBS pairs feature a more complex geometry, with many fragmented surfaces, in contrast to the often compact
objects in the recognition pairs. Six of the object recognition cases come from our
dataset (subsection 5.8.1), while one is the teddybear used in the independent work
of [Rothganger et al. 2003].

The region correspondences are obtained by one-to-one matching for the WBS cases,
and by soft-matching for the object recognition cases (see section 5.2 for details on
soft-matching). In no case the exploration technique (chapter 5) has been used, as it
would make the conditions easier, because it would create a high ratio, and a large
amount, of correct matches.

In total there are 2253 matches, which have been partitioned into 1428 GAMs. 1378
of them are formed purely from mismatches, while there are 50 GAMs containing
all 415 correct matches. We call the former incorrect GAMs and the latter cor-
rect GAMs. Since the overall ratio of correct matches is only 18.4%, the collected
statistics are particularly relevant and truly summarize the behavior of the GAM
constructor. In the next subsection the test conditions are pushed even further,
with an experiment where GAMs are constructed from sets augmented with large
amounts of artificially created mismatches, reaching a ratio of correct matches as
low as 4%.

Figure 6.17 plots the number of incorrect GAMs as a function of their size. The
exponential decrease is very apparent. There is only one incorrect GAM of size 6,
one of size 7, and none larger than 7. These numbers confirm the first fundamental
property: it is very unlikely for mismatches to be grouped into large GAMs.

In order to validate the second property, we have counted the GAMs composed by
either only correct matches or only mismatches. These are the 96.4% of all non-
singleton GAMs (the property trivially holds for singleton GAMs, so they have been
omitted from this statistic). The property is also almost fulfilled by the remaining
3.6% of GAMs, as they contain all correct matches, but one (2.4%) or two (1.2%).
Even in these rare cases, the mismatches were lying pretty close to their correct
location (see figure 6.19). No GAM mixed more than two mismatches with a correct match, therefore meeting the expectations formulated in section 6.6.

The relation between the size of a GAM and its probability of being correct is illustrated in figure 6.18, which plots the percentage of correct GAMs of size $N$, for various $N$. One can clearly see that the chances that a GAM is correct quickly grow with its size. Note that all GAMs containing more than 7 regions are correct (figure 6.17), and that some correct matches are grouped into small GAMs.

The informative power of the two properties is very important and can be illustrated as follows. Suppose we discard all GAMs of size 7 or less. Although simplistic, this strategy results in the elimination of nearly all mismatches, with the loss of only 17.8% of the correct matches. In practice, one is more careful, and tries not to lose the precious fraction of correct matches within the smaller GAMs. This line of reasoning suggests the proper use of GAMs, like in the two-view filter of section 6.7: large GAMs are trusted being correct and are relied upon to validate the smaller ones.

The crux of the matter is that the two fundamental properties impose a preliminary form of order on an otherwise chaotic large set of unreliable matches. Special GAM-based algorithms can then be designed to take advantage of this order, like the aforementioned two-view filtering algorithm, and the multiple-view integration of the next chapter.

Figures 6.20 through 6.22 show some examples GAMs on 3 of the 14 image pairs. The first example is the well-known Graffiti from INRIA, introduced in their 2002 paper [Mikolajczyk and Schmid 2002] and used in other works since [Xiao and Shah 2003]. The constructor algorithm grouped in a single GAM 71 matches spread over the whole wall, despite evident perspective effects (figure 6.20). Two views of the Birthday scene, used to demonstrate multi-view correspondences in section 4.7, are shown in figure 6.21. The largest GAM groups 23 matches on the round box in the background, while a smaller GAM with only 5 matches is found on the diskette.
6.8. Results for GAMs

Figure 6.19: An unfrequent case where a mismatch has been put into a correct GAM. Nevertheless, the region is mismatched near the correct corresponding location. This situation is caused by the repetitive patterns covering the object.

Figure 6.20: Graffiti scene. A large GAM formed by 71 matches covering the whole wall, effectively bridging the significant perspective effect. For clarity, only region centers are shown.

Although relatively small, the latter GAM is correct as well. This example draws attention to the exact information GAMs provide: while in practice a large GAM always contains correct matches, there is no guarantee that all correct matches will be found in large GAMs! If one cares about finding all correct matches, small GAMs should be inspected as well. The pivotal advantages of using GAMs are that large GAMs can be relied upon, and that each GAM can be safely considered in its entirety, without looking at the correctness of each and every match individually. As a last example, a GAM formed by 13 matches covering the whole round head of a toy leopard is shown in figure 6.22. The toy leopard is also used in the object recognition database of section 5.8. In total, only 17 of the 155 initial matches are correct (11%, a few of them are on the body). The GAM grouper succeeds in picking out the 13 matches on the head, without being confused by the large amount of mismatches.
6.8. Results for GAMs

Figure 6.21: Birthday scene. The 23 matches on the round box in the background (white dots), are grouped in a different GAM than the 5 matches on the diskette (black dots).

Figure 6.22: Leo, object recognition example. The largest GAM nicely covers the whole round head.

6.8.2 GAM-based two-view filter

In order to test the GAM-based two-view filter algorithm explained in section 6.7, we take a closer look at two image pairs from the previous subsection.

Felix

The first pair is shown in figure 6.23. There are 122 initial matches, 98 of which are correct. The GAM grouper puts 96 of the correct matches into the two largest GAMs (sized 59 and 37 respectively). The mismatches are spread over 22 singleton GAMs and only one GAM of size 2. Note how a correct GAM covers a curved surface (magazine)

In order to test the robustness of the GAM constructor and two-view filtering algorithms, we have added an increasing amount of randomly generated mismatches to
6.8. Results for GAMs

Figure 6.23: The two main GAMs of the Felix scene. One GAM covers a curved surface (magazine). For clarity, only some matches are shown for each GAM.

the initial set of matches. We added from 300 to 2100 random mismatches in 7 steps, therefore raising the total percentage of mismatches from the initial 20% gradually up to 96%. At each step, we recomputed the GAMs and applied the filter. As shown in figure 6.24, the filter performs very well, steadily returning at least 98% of the correct matches, also when as much as 96% of mismatches contaminate the input. At the same time, the percentage of accepted mismatches stays approximately constant at 2.5%. The composition of the two largest GAMs remains identical in the first 5 steps, with up to 1500 added mismatches, and changes only marginally in the last two steps (difference of a few matches). The largest incorrect GAM never contains more than 4 matches. This corroborates further the two fundamental properties, as it shows they hold largely independently of the amount of mismatches in the input set.

As a comparison, figure 6.24 shows the performance of RANSAC-EG as well. Even though it works very well in the initial case, returning all correct matches but one, its performance drops sharply with increasing amounts of mismatches. Already when adding 300 mismatches, only 51% of the correct matches are returned. This value continuously decreases until about 8% in the last two steps. An approximately constant rate of about 4.5% mismatches are accepted.

Foliage

The foliage scene in figure 6.25 illustrates a practical application of the new filter. Severe matching ambiguities arise due to the very similar structures repeated many times in the images. This results in many more mismatches than in usual scenes. The initial matches set contains 25 correct matches out of 206 (12%). Under these conditions, RANSAC-EG fails. However, the largest 3 GAMs are correct and contain
6.8. Results for GAMs

Figure 6.24: Percentage of correct matches returned by the filters for the Felix scene (figure 6.23). The GAM-based filter works always well, even with the lowest percentage of correct matches. In contrast, the performance of RANSAC-EG quickly degrades.

Figure 6.25: The three main GAMs of the Foliage scene. Note the presence of many similar repeated elements, due to the similar colors and shapes of the leaves.

23 correct matches (the 3 GAMs have respectively 11, 8 and 4 matches). The mismatches are scattered in mostly singleton GAMs, with the largest incorrect GAM having only size 3. Our filter returns 24 correct matches and 5 mismatches, therefore qualitatively solving the problem. The observed weakness of RANSAC-EG is in agreement with the independent observations in [Chum et al. 2003] and [Lowe 2004].

6.8.3 GAMs from expanded matches

Until now, GAMs were made of conventional matches, i.e. correspondences established between regions produced by an affine invariant region extractor run on each image independently. However, nothing prevents feeding the GAM constructor with the output of the exploration algorithm of chapter 5. In this case there are many more matches, typically densely covering the parts of the object/scene visible in both images. When applied to this input, the GAM decomposition is most mean-
6.8. Results for GAMs

Figure 6.26: A very large GAM covers the whole unoccluded part of the magazine, in spite of deformation, occlusion and clutter. The contour of the matches composing the GAM is shown.

Figure 6.26: A very large GAM covers the whole unoccluded part of the magazine, in spite of deformation, occlusion and clutter. The contour of the matches composing the GAM is shown.

meaningful, because the GAMs typically cover larger areas, even if curved, deformed, or seen under very different viewpoints. Indeed in many of these cases only a modest amount of correct conventional matches are available, therefore providing the GAM constructor with insufficient prime matter.

We present three examples. A single, very large GAM, composed of 167 matches, covers the whole unoccluded surface of the magazine in figure 6.26. The GAM constructor managed to group matches from the whole, strongly folded surface, with affine transformations varying gradually over a wide range.

The Teddybear of [Rothganger et al. 2003] is shown under two considerably different viewpoints in figure 6.27. There is a GAM on the head, and another on the left arm (which is on the right in the image). The third GAM covers both the right arm and the right paw, as they undergo similar geometric transformations. While usually each correct GAM corresponds to an object facet, this is not a crucial feature, rather an elegant by-product. Two facets undergoing a similar transformation will be merged in the same GAM, for example when the camera translation is only modest w.r.t their orientations. However, what matters is the promotion of the two fundamental properties, and therefore it is important for correct GAMs to be as large as possible, in order to better stand out. From this point of view, a single surface should not be split up into several GAMs, and it is actually better when many correct matches lying on two facets are merged into one large, robust GAM.

Figure 6.28 presents Coleo, a plush toy with leopard texture and a complex shape composed by several curved surfaces. Despite the very different viewpoints, the exploration algorithm produced about 120 correct matches, densely covering the parts visible in both views. The two GAMs correspond well to the principal contiguous surfaces, which are the head (37 matches) and the back-arm complex (81 matches). The close-ups in the bottom of figure 6.28 show some of the matches among the
Figure 6.27: Three GAMs on the teddybear. A GAM correspond to the head (bright), another to the left arm (dark thin) and the last to the right arm/paw (dark thick).

back-arm GAM. The regions are all circles of the same size in the left image, because they are part of the homogeneous coverage generated in subsection 5.3.1. The contiguous variation of the regions’ shapes in the right image mirrors the changes in affine transformation due to the varying surface orientation. The range of the transformations is very wide, going from the small, vertical, y-stretched regions on the left of the arm, to the large, horizontal, x-stretched ones on the right of the back (near the black label). Nevertheless, the GAM grouper succeeded in putting all these matches together to form a large GAM, exploiting the fact that the transformation changes gradually from a region to the next.

The characteristics of GAMs presented here will be most useful for integrating contributions from multiple model views in the object recognition scheme of the next chapter.

6.9 Related work

6.9.1 Coplanar grouping / Plane detection

We are not aware of other work on grouping affine invariant regions into coplanar classes. However, the idea is related to the general problem of plane detection, which we discuss in the sequel. Our method has been published in 2001 [Ferrari et al. 2001b] and works with two, potentially widely separated, views.

Despite the fact that planes are very useful in various problems of Computer Vision, surprisingly few methods have been proposed for their automatic detection. Some authors reconstruct scene points in 3D, before grouping them into planes [Bartoli
Figure 6.28: Top: Two GAMs on two very different views of Coleo. Bottom: Close-up on some of the matches of the back-arm GAM. The geometric transformations vary over a wide range, but change gradually among spatially neighboring regions.

2001, Imiya and Fermin 1999]. Unfortunately, they use more than two images during reconstruction. Moreover, plane detection should be achieved as a preprocessing step, before reconstruction, so as to be useful to methods able to exploit the constraints imposed by planarity [Rother and Carlsson 2001, Vidal and Oliensis 2002].

The opposite approach is taken by 'direct' methods, which estimate planar motion directly from the pixels of two images, without even matching features. They minimize the SSD of pixel intensities w.r.t. the unknown parameters of the homography aligning the images. Typically, the SSD is computed over the entire image [Bergen et al. 1992]. Problems due to multiple motions in the image (different planes) are countered by first estimating a simple translational model. This usually locks on the 'dominant' plane, which is then segmented out, and on which a full homography model is estimated. The pixels of this plane are removed and the procedure iterates [Irani et al. 1994]. Direct methods suffer from several drawbacks. First of all they assume the existence of a dominant plane, covering the majority of the image. With several equally large planes, the estimation breaks down. Moreover,
these methods rely on the brittle assumption that corresponding pixels have identical intensity, violated in wide-baseline conditions or with changing illumination. Finally, the minimization follows a similar vein as the computation of optical flow, and works only with small inter-image transformations. Even though repeating the minimization from coarse to fine resolutions [Bergen et al. 1992, Irani et al. 1994, Zelnik-Manor and Irani 2000] helps dealing with larger displacements, the method fails in general wide-baseline conditions, which include large variation of all homography parameters (due to out-of-plane rotation, scale, etc.).

The method of [Sinclair and Blake 1996] (S96) detects planes by grouping point correspondences. Quintuples are formed containing each a different input point and its four nearest neighbors. A quintuple is deemed coplanar if the two projective invariants it defines hold. In this case, a homography is fit to the quintuple and used to predict the motion of other points. All points moving like predicted are declared coplanar and added to the group. The method iteratively re-estimates the homography based on the larger group, and uses it to include more points. The method tends to include in a plane A several points lying on another plane B, especially near their intersection. This happens because the points might agree with the current homography of A. Once included, they skew the newly estimated homography into accepting even more points of B. The authors tentatively propose a final stage to keep only points found to be exclusively on one plane. Nevertheless, this way several points remain unclassified and the problem of coplanar points lost due to the perturbed homography remains. In our approach, points near intersections are often correctly classified, because they are more attracted (coplanarity scores) by points on their own plane (as in figure 6.6). Our grouping process via the CP approximation takes a more global view, and 'grows' all planes at the same time, evaluating all possible plane memberships for each feature. Moreover, there is no danger of skewed homographies, as there are only measurements between pairs of features. As another shortcoming of S96, some planes might not be discovered, because no point is coplanar with its four nearest neighbors. This happens with non-compact planes, or when the matches are sparse (as in figure 6.8). Such a situation does not disturb our approach, because the coplanarity cues are evaluated for all pairs of features. A pair suffices to estimate an homography, because we exploit the fact that an affine region match provides correspondence between three independent coplanar points. Evaluating all possible projective invariants could not be done by S96 because there are too many combinations of 5 points. As last, possibly most important limitation, S96 inputs point correspondences, which can be obtained reliably only with small inter-image differences (the paper uses points tracked through an image sequence). The attentive reader might argue that S96 can be readily upgraded to wide-baseline conditions by feeding it with the point correspondences produced by an affine region matcher. However, the method would then see the 3 point correspondence in an affine region as unrelated, and would also continue to suffer from the other
mentioned drawbacks. Our algorithm instead is especially designed for the different problem structure induced by the implicit 'pre-grouping' of the 3 points in an affine region.

In [Lourakis et al. 2002] another interesting method is described (published after ours, in 2002). Matches between line segment and point features are first computed by conventional, small-baseline, correlation-based techniques. The Fundamental Matrix \( F \) is estimated from the matched points, and the epipoles derived from it. By combining \( F \), the epipoles and a pair formed by a line and a point match, one can compute the homography induced by the 3D plane defined by the 3D line and 3D point. The homographies are computed for all point-line pairs, and used to transfer all features. The best homography is chosen, correctly transferring the largest number of features (its inliers). After re-estimating the homography from all its inliers, these are removed and the process iterates. Problems near plane intersections are alleviated by a final stage, reassigning each feature to the homography that best transfers it. This approach needs to compute \( F \) and accurately extract the epipoles. This fails in the quite common case where all features lie on one plane, and with independently moving objects. Our approach instead does not compute \( F \) and works reliably in these situations. Moreover, [Lourakis et al. 2002] uses conventional feature matching, and is only demonstrated on small-baseline stereo pairs. Providing it with affine region matches would violate the nature of the approach, which is specialized on combinations of lines and points. Finally, our region grouping by CP finds all planes at the same time, thus avoiding the bias towards large planes hidden in the iterative voting mechanism of [Lourakis et al. 2002].

An additional difference to the two approaches just discussed, our method uses also photometric information, in addition to geometric information (with the 'texture cue' in section 6.2).

### 6.9.2 GAMs / two-view filtering

The GAM decomposition is novel in its attempt to group affine region matches lying on smooth surfaces. However, finding GAMs is not a goal per se. GAMs might be better related to the research world by considering the higher-level algorithms they enable. We discuss here one of them, the two-view filter, while we leave the multiple-view integration for object recognition to the next chapter.

The task of recovering the Fundamental Matrix and simultaneously rejecting mismatches (outliers) is usually given to robust statistical estimators. The three classic techniques are M-estimators, Least-Median-of-Squares (LMedS) and Random-Sampled-Consensus (RANSAC). M-estimators tolerate well inaccurately localized matches, but only a minor amount of real mismatches. LMedS is more robust, and can resist to up to 50% mismatches [Zhang et al. 1995]. RANSAC is by far the most
widely used technique of the recent years. There is no exact limit in the percentage of mismatches it can work with, but researches agree it should not be used beyond roughly 2/3 (67%) [Lowe 2004, Chum et al. 2003].

The two-view filter we propose in this chapter can work with much higher amounts of mismatches, and we demonstrate it on a case with up to 96%. The basic reason is that, while RANSAC has minimal chances to get a mismatch-free sample, our two-view filter directly sees subsets of matches which are likely to be correct, because they form the largest GAMs. In more theoretic words, all matches are equiprobably correct during RANSAC's sampling, whereas they are not during our filtering process, because the GAM decomposition provides an implicit estimation of which matches are more probably correct. This non-uniform distribution is the key to the filter's success. Recently, an improved version of RANSAC, tuned to use the full affine transformation between matches, has been proposed [Chum et al. 2003], and demonstrated on scenes with up to 88% mismatches. Yet, to the best of our knowledge, no method for estimating the Fundamental Matrix and filtering mismatches has been shown robust to as high ratios of mismatches as ours can deal with (we demonstrate it up to 96% in the result section).

Although the last claim might seem very strong, its generality is moderated by three issues. First, although the ratio of correct matches can be extremely low, the location of the matches is slightly constrained. Indeed, the scene should contain at least some smooth surface, to host groups of at least 3-4 matches. This condition is fulfilled by the large majority of scenes, but it is in principle less general than what RANSAC could handle (for example: there are only 9 correct matches, each on a different small surface). Second, unlike RANSAC, our GAM-based two-view filter is not a general model-fitting paradigm. It can only be applied to two-view matches, e.g. to find Fundamental Matrix, or a homography. Third, it explicitly uses the whole affine transformation between matches, so it cannot be used when the input are simple point correspondences (the same holds for [Chum et al. 2003]). Nevertheless, considering the growing interest in the community for affine invariant regions, we believe our filter to be useful and attractive for a broad audience.

As final remark, we mention the work of [Tuytelaars et al. 2002], which groups affine regions of a single image forming symmetric or repetitive patterns. The work is profoundly different as it doesn't look for planes or smooth surfaces, and operates in one image, but it is related in that it also finds groupings of affine invariant regions.

### 6.10 Conclusion and outlook

In the first part of the chapter, we have presented a method to partition a set of two-view matches into coplanarity classes. The experimental section shows the
validity of the coplanar grouping approach and its most important properties. It is capable to distinguish different planes even given small motion between the two views, or when regions from different planes lie in close proximity. Moreover, it works in the presence of perspective effects, discontiguous planes, parallel planes and noisy input information due inevitable imaging effects and to inexact region correspondences. A qualitative and very time-efficient approximation algorithm for the Clique Partitioning problem has been introduced, and applied also for texture segmentation.

The GAM concept, a construction algorithm and its application to robust two-view filtering have been the object of the second part of the chapter. As confirmed by the experimental section, GAMs are capable of covering planar, curved or smoothly deformed surfaces, and possess two fundamental properties. First, correct matches and mismatches are separated into different GAMs. Second, there is a strong relation between the amount of matches in a GAM and its probability of being correct. In particular, mismatches form GAMs composed of at most a few regions. These properties hold largely independently of the amount of mismatches in the input set, and reveal precious for the design of higher-level algorithms. As an example, a GAM-based filter is presented which strongly outperforms RANSAC-EG and is robust to very high amounts of mismatches. The filter assumes the presence of some smooth surface, which holds for a very broad range of scenes. The next chapter will explore the use of GAMs for integrating multiple model-views in the object recognition context.

This coplanar grouping method was published in 2001 in [Ferrari et al. 2001b], while the GAM constructor and the GAM-based two-view filter appeared in [Ferrari et al. 2004a].

Some directions of future improvement are:

**Pixelwise plane segmentation**

Although the approach groups regions into coplanarity classes, it does not tell anything about image areas outside the regions. It would be interesting to compute a global homography for a coplanarity class, and then use it within a change detection scheme in order to decide which other image pixels belong to the same plane. In this fashion the result would be upgraded to a *piecewise planar segmentation* of the images.

Something similar is achieved by using matches from the image-exploration technique as input, because they typically densely cover the scene parts visible in both images. However, the change-detection approach could deliver more fine-grained segmentations and would work also with conventional matches, thus cutting the computationally more expensive image-exploration short.
Symmetry of coplanarity cues

Unlike the affine dissimilarity cue of GAMs, the coplanarity cues introduced in section 6.2 are not symmetric in the views. While this did not cause problems in our experiments, it is expected to be potentially troublesome in the presence of large scale changes, and it is arguably unelegant. One easy way to symmetrize the cues is to first recompute them with the role of the two images reversed, and then average the result with the normal cues.
Integrating multiple-view for object recognition

7.1 Introduction

The powerful image-exploration technique of chapter 5 has a quite simplistic approach for using multiple model views. Each view is matched to the test image independently, and the total number of matches is used as recognition criteria. No attempt is made to make the model views cooperate, even if usually there are important overlaps between the object parts seen by each view. It would certainly be better to exploit these relationships so as to achieve a real integration.

In this chapter, we present an approach which effectively integrates the contributions of multiple model views. The main ingredient of the approach is the partitioning of a set of region matches between two images into groups of aggregated matches (GAM, introduced in the previous chapter). Recall that GAMs enjoy two fundamental properties. First, the matches in a GAM are most often all correct, or all incorrect. Second, it is very unlikely for mismatches to form large GAMs (i.e. composed of many matches). Hence, the size of a GAM informs about the probability of it being correct. Large GAMs (more than 5-10 matches) are very probably correct. A GAM usually covers a smooth surface of the object/scene, though this is not a crucial feature. It can extend to multiple surfaces when the camera translation is only small with respect to their 3D orientation. Because of these properties, it is convenient to reason in terms of GAMs, rather than individual matches. Our multiple view integration scheme relates GAMs arising from different model views, and considers them as atomic units, without descending to the matches level.

The object recognition system of chapter 5 is upgraded to include multiple-view integration as follows. During modeling, the model views are densely connected by a number of region-tracks. Each region-track connects the image regions of a certain physical surface patch across the views (section 7.2). At recognition time, we match
7.2 Modeling

Before the system is ready for recognition, a preliminary modeling stage is necessary to capture relationships among the model views. These relationships are modeled by a dense set of region-tracks. Each such track is composed by the image regions of a single physical region along the model views in which it is visible. The set of tracks should densely connect the model views, because they will be used during recognition in order to establish connections among GAMs matched from different model views to the test image (section 7.3).

This section explains how to build the model region-tracks, starting from the bare set of $M$ unordered model images. First, dense two-view matches are produced between all pairs of model images (subsection 7.2.1). All pairwise sets of matches are then integrated into a single multi-view model (subsection 7.2.2). This process can be regarded as a specialized, dense counterpart of other sparse multi-view matching schemes, like the one presented in chapter 4, or the ones in papers [Schaffalitzky and Zisserman 2002b, Ferrari et al. 2003].

Figure 7.1 presents all model views of Coleo, the example object which will be used throughout the chapter.

7.2.1 Dense two-view correspondences

A dense set of region correspondences between any two model views $v_i, v_j$ is obtained using a simplified variant of the image-exploration technique of chapter 5. More precisely, it uses a simple one-to-one nearest neighbor approach for the initial matching
Figure 7.1: The Coleo example object. The eight model views are taken about every 45 degrees during a complete tour around the vertical axis.

instead of the soft-matching phase, and there are no 'early' phases (sections 5.3, 5.4). The system directly goes to the 'main' phases after the initial matching (sections 5.5, 5.6). The use of this faster, and less powerful version is justified because matching model views is usually easier than matching to a test image (compare the test cases in section 5.8). Typically, there is no background clutter in the model images, and the object appears at approximately the same scale in all model views.

Let's recall that the image-exploration technique focuses on constructing correspondences for many overlapping circular regions, arranged on a grid completely covering the first model view \( v_i \) (these are called coverage regions, see subsection 5.3.1). The procedure yields a large set of reliable region correspondences, densely covering the parts of the object visible in both views (figure 7.2). Please note that the image-exploration matcher is not symmetric in the views, as it tries to construct
Figure 7.2: Two-view matches produced by matching view 4 to view 5. In spite of the considerable viewpoint change, there are 145 matches densely covering all parts visible in both views, including the left paw, arm and tail.

correspondences in the second view, for the coverage regions of the first view (we say that it matches \( v_i \) to \( v_j \), noted \( v_i \rightarrow v_j \)).

The method also works well in the presence of smooth non-rigid deformations, like folding and bending. Although this usually does not happen during modeling, it is an important feature at recognition time.

### 7.2.2 Dense multi-view correspondences

Once two-view region correspondences have been produced for all ordered pairs of model views \( (v_i, v_j), i \neq j \), they can be organized into multi-view region tracks. When matching a view \( v_i \) to any of the other model views, we always use the same set of coverage regions. Therefore, each coverage region, together with the regions it matches in the other views, induces a region track (figure 7.3). Note that if a region is matched from view \( v_i \) to view \( v_j \), and also from view \( v_i \) to view \( v_k \), then it is implicitly matched between \( v_j \) and \( v_k \) as well, because it will be part of the same track. These transitive matches actively contribute to the inter-view connectedness, as they often link parts of the object that are harder to match directly.

The final set of region tracks constitutes our object model. Figure 7.4 shows all 3-view tracks passing through views 4, 5, 6, after building the model from all 8 views.

### 7.3 Recognition

Given a test image, the system should determine if it contains the modeled object. The first step is to match each model view of the object to the test image separately.
7.3. Recognition

Figure 7.3: a) Coverage regions for model view 5. b) One of the coverage regions. c+d) the corresponding regions constructed by the image-exploration algorithm in views 4 and 6. These direct matches $5 \rightarrow 4$ and $5 \rightarrow 6$ induce a three-view track across views 4, 5, 6. Hence, the transitive match $4 \rightarrow 6$ is implied.

Figure 7.4: 242 3-view tracks through views 4, 5, 6. The tracks densely connect the views. Note that there are more tracks here as mere 2-view matches in figure 7.2. This happens because here also matches from all other pairwise matching processes are included (for example $6 \rightarrow 4$ and $6 \rightarrow 5$).
7.3. Recognition

Figure 7.5: A correct GAM (head), matched from view 3, and an incorrect one (paw) from view 4. The GAM on the paw is transferred from model view 4 to model view 3 (arrow) via the model's connections, prior to computation of the compatibility measure.

For this purpose, the image-exploration algorithm of chapter 5 is used again, this time in its complete version. Each resulting set of region matches is then partitioned into GAMs, via the algorithm of section 6.6. When applied to these dense matches, the GAM decomposition is most meaningful. Each correct GAM then usually corresponds to (part of) an object facet (figures 7.5, 7.6 show some GAMs; only the contour of each GAM is shown).

However, at this stage, there is no guarantee that all GAMs are correct. As a result, there usually are some inconsistencies in the set of GAMs. For instance, a GAM correctly matches the head of the object in figure 7.5 from model view 3 to the test image. Furthermore, there is another GAM erroneously matching the paw in model view 4 to the chest in the test image. Since the model views are interconnected by the model tracks, the model knows the correspondences among the regions on the paw in model views 3 and 4. Therefore it considers that the second GAM matches the chest in the test image to the paw in model view 3. Now both GAMs match model view 3 to the test image, and their (geometric) inconsistency can be measured and discovered.

Just as it finds conflicting GAMs, the system can notice GAMs that are compatible (figure 7.6). This is a good reason for considering them as more reliable and therefore to reinforce the system's belief in the presence of the object. This leads to the
Figure 7.6: Two compatible (and correct) GAMs. The nose GAM (black) is initially matched from model view 8, and is transferred to model view 1 (arrow). Note how the other GAM (white) is very large and covers the head, arms and chest. A GAM can extend over multiple facets when the combination of viewpoints and surface orientations induce the region matches to have smoothly varying affine transformations even across facets edges. In these cases, the resulting GAMs are larger and therefore more reliable and relevant.

main advantage in evaluating GAM compatibilities: the reliability of the recognition decision is enhanced, because higher scores can be assigned in positive cases (i.e.: when the object is in the test image). As a secondary advantage, incorrect GAMs can be detected and removed, thus improving the segmentation.

In this section, we explain how to realize these ideas. But before going into the details, we sketch the overall procedure. For each pair of GAMs, we compute a compatibility score, based on the geometric consistency of their relative arrangement. In simple cases, the two GAMs are matched from the same model view and the score can be directly computed. In the more interesting cases where each GAM is from a different model view, we first have to transfer one of the GAMs to the model view of the other. This is made possible by the connections embedded in the model tracks. Next, the pairwise scores are integrated in a single configuration score. This varies as a function of the configuration, the subset of all GAMs which are considered correct. The process favors highly compatible subsets containing large GAMs. This is justified because larger GAMs are more likely to be correct (as discussed in section 6.8). A genetic algorithm is used to maximize the configuration
score. The maximum yields the final recognition score and reveals which GAMs are deemed incorrect. The recognition score increases in the presence of compatible GAMs, thereby improving recognition performance.

The final recognition score, and the decisions to remove GAMs, are based on a global analysis of the situation. This considers simultaneously relationships among all pairs of GAMS, coming from all model views. It is computationally feasible because there are much less GAMs (a few tens) than region matches (hundreds to thousands). This is an advantage of reasoning on the higher perceptual grouping level offered by GAMs. The system no longer needs to consider each single region individually, but it can rely on a meaningful organization instead.

The following subsections describe the elements of the above strategy in more detail.

### 7.3.1 GAM transfer

Consider a GAM matched from a model view $v_i$ to the test image, and another GAM matched from a different model view $v_j$. Before computing the compatibility score for this GAM pair, they must be put in a common model view. Only then the geometrical coherence of their relative arrangement can be evaluated.

A GAM is transferred from $v_i$ to $v_j$ as follows (figure 7.7):

1. Determine the set of model regions $\Lambda$ covering the same part of $v_i$ as the GAM. This is implemented by selecting the model regions which strongly overlap (more than 70%) with the image area covered by the union of the GAM's regions. Remove from $\Lambda$ all regions which are not part of a model track passing through $v_j$. The model can now predict the location and shape of the GAM in $v_j$.

2. Compute the affine transformations mapping each region of $\Lambda$ from $v_i$ to $v_j$.

3. Project each GAM region to $v_j$ via the affine transformation of the nearest region of $\Lambda$. We have established a region-to-region correspondence for the GAM between the test image and model view $v_j$.

When transferring a GAM, it is like making a model-based prediction. The pairwise compatibility score (next subsection) evaluates to which degree the two GAMs are consistent with this prediction. This idea is essential: in this way the system exploits the relationships among the model views, in order to conclude more than what is possible from the mere collection of all GAMs. During modeling, the system learned the structure of the object in the form of region tracks, and it brings this insight to bear at recognition time by imposing order on the GAMs.
7.3. Recognition

b) overlapping model regions c) transfer one region to view 1

Figure 7.7: An example of the GAM transfer mechanism. a) The GAM to be transferred, which is originally matched from view 3 to the test image. b) The set \( \Lambda \) of overlapping model regions. c) One of the GAM regions (white) is transferred from view 3 to view 1, via the affine transformation of the closest region of \( \Lambda \) (black). We now know the correspondence between view 1 and the test image.

Note that a GAM cannot be transferred if the model regions it covers in view \( v_i \) are not visible in view \( v_j \) (\( \Lambda \) is empty). In these cases, the compatibility score is not computed, and a neutral score is assigned instead. Note that both GAMs could still be correct, but the object parts they cover might not be visible in view \( v_j \) at the same time.

7.3.2 Pairwise compatibility score

We evaluate here the geometric consistency of a pair of GAMs. Both GAMs are matched between the test image and a model view \( v_i \). If at least one of the two GAMs is incorrect (i.e.: composed of mismatches), we wish this measure to be low.

The compatibility score is based on the sidedness constraint for unordered triples of region matches, introduced in subsection 4.6.1. The center of the first region should be on the same side of the directed line going from the center of the second region to the center of the third region, in both the model and the test images. This holds for all coplanar triples of correct matches and also for most non-coplanar ones.

We check the constraint for all triples formed by a region from a GAM and two regions from the other GAM. The percentage of triples respecting the constraint is our choice for the compatibility score of the GAM pair.
The central idea is that if a region is picked from an incorrect GAM, we expect that most of the triples in which it takes part violate the constraint. Note that no triple is composed of regions from a single GAM. This preserves the quality of the measure when exactly one of the GAMs is correct. In these cases, most triples based only on the correct GAM will respect the constraint, and would therefore falsely raise the score.

The proposed score tolerates a substantial amount of non-rigid deformation. This preserves the system's capability of recognizing deformable objects. Moreover, it is insensitive to inaccurately localized region matches, because the amount of triples violating the constraint grows slowly and smoothly with a region departing from its ideal location (see subsection 4.7.2). This might happen, for instance, when a region is bridging a non-planar part of the object, or in the presence of large scale changes, or with all kinds of image degradation (motion blur, etc.).

The score can penalize conflicting GAMs, but also assign high scores to compatible pairs of GAMs. Although the score is based on comparing region matches, it captures the compatibility of the GAMs as a whole.

### 7.3.3 Configuration score

The compatibility scores are computed for all pairs of GAMs, and are combined here in a single configuration score.

The compatibility scores fall within the range \([0,1]\). Based on a threshold \(t\), we linearly transform the interval \([0,t]\) to \([-1,0]\) and the interval \([t,1]\) to \([0,1]\). The resulting values then fall in the range \([-1,1]\). In all experiments, the same threshold \(t = 0.2\) splits the original range into positive and negative parts. Positive scores now indicate that two GAMs are likely to belong together, while negative ones indicate incompatibility.

Let a configuration \(C\) be a subset of the available GAMs. What is the score of a configuration? It should be high when containing large, mutually compatible GAMs. It should be lower in the presence of incompatible ones. These two forces, pairwise corroboration and individual size, are combined into the following configuration score:

\[
S(C) = \sum_{P \in C} \left( \text{Size}(P) + \sum_{Q \in C \setminus P} (\text{Comp}(P,Q) \cdot \text{Size}(Q)) \right)
\]  

(7.1)

The number of regions in GAM \(P\) is denoted \(\text{Size}(P)\), while \(\text{Comp}(P,Q) \in [-1,1]\) are the pairwise compatibility scores. We are interested in the maximum value of
7.3. Recognition

S(C), and in the configuration for which it occurs. The maximum value is used as recognition criteria, to decide whether the object is in the test image. As argued before, more trust is given to the larger GAMs (first summation term). The second term makes the contribution of each GAM heavily dependent on its compatibility with the others, especially the larger ones. A GAM whose negative compatibilities lower S will be left out. Smaller GAM can also be part of the maximum configuration, depending on how compatible they are with the others.

An important effect of the second summation term is that the total score can be much higher than the mere sum of the sizes of all correct GAMs. This reflects the key idea that compatible configurations are worth more because they more reliably indicate the presence of the object. This increases the separation between scores in positive and negative cases, thus improving discriminative power.

The GAMs not selected by the best configuration are deemed incorrect and are removed. Note how this decision is based on a global analysis. Typically, several incorrect GAMs are detected thanks to their incompatibility with GAMs matched to other model views. Such a case couldn’t have been discovered by looking at the GAM’s model view in isolation. This is another benefit of our proposal for integrating multiple model views. In analogy with section 6.7, each GAM is treated as an atomic unit.

7.3.4 Maximization by Genetic Algorithm

We now need to find the configuration which maximizes function (7.1). Unfortunately, we can’t try them all out, as there are $2^n$ possible configurations of $n$ GAMs. Moreover, a function in the form of (7.1) cannot be maximized by graph-cuts methods (e.g. [Shi and Malik 1997]), as shown by [Kolmogorov and Zabih 2002].

We designed a Genetic Algorithm (GA) to find an approximation of the solution. GAs offer an elegant and flexible framework for optimizing functions of any form. In this context, we represent a configuration by a binary indicator vector $I$ of length $n$. If $I(p) = 1$, the $p$th GAM is in the configuration. The fitness function $F(I)$ is defined equivalent to $S(C)$. The GA follows several steps:

1. **Initialize.** Create a random, uniformly distributed population of binary $n$-vectors. The size of this population is $l = \text{ceil}(\sqrt{2n})^2$. Since this enforces $\sqrt{l}$ to be an integer, it simplifies the later crossover.

2. **Fitness.** Evaluate the fitness function $F(I)$ for each individual. Stop if the best individual is identical as in the previous generation (not tested the first time).
3. *Crossover*. Consider the best $\sqrt{l}$ individuals. Derive the next generation by crossing over all pairs of them. Crossing over two individuals means keeping the identical bits and randomly choosing the different bits. This amounts to producing $l - \sqrt{l}$ new individuals, and copying the current best $\sqrt{l}$.

4. *Mutation*. Each bit of each individual in the new population is switched with probability 0.1. This avoids that the algorithm explores only the part of the search space spanned by the best individuals.

5. *Iterate*. Iterate to point 2.

In section 7.4 we demonstrate experimentally the effectiveness of the GA in maximizing function $(7.1)$, and its computational efficiency. One of the reasons for this good performance is in the nature of the optimization problem itself. In the vast majority of cases where the object is in the test image, the GAMs sizes are very non-uniformly distributed, with some large GAMs, and a greater number of smaller ones. Moreover, the value of function $(7.1)$ raises more when large GAMs are in $C$, and even much more with compatible large GAMs. As a result, the search space, while being huge, has a strong non-flat shape, and usually features high peaks for $C$ containing at least some of the largest GAMs. These characteristics significantly ease the task of the GA.

### 7.4 Results

The next three subsections illustrate and discuss the behavior of the proposed multiple-view integration scheme. After presenting some positive cases (the object is in the test image), we turn to negative cases, and conclude with a closer look at the performance of the GA maximization of the total score function.

#### 7.4.1 Positive cases

**Coleo**

The object of figure 7.1, *Coleo*, serves as main experimental item. It features a complex geometry composed by several curved surfaces. Moreover, it is covered by ambiguous texture, formed by many small variations on the same basic pattern, which challenge the matching process. The model is built from only 8 views, taken at 45 degrees intervals, all at the same height, during a tour around the vertical axis.

On the example of figures 7.5 and 7.6, the system initially produces 33 GAMs by matching each model view to the test image, with the image-exploration method
7.4. Results

of chapter 5. Only 9 of the GAMs are correct, but 4 of them are very large (more than 60 matches) and contain the majority of the correctly matched regions. The techniques of this chapter select 10 GAMs in the configuration with the maximal score. All 9 correct GAMs are included, while all but one of the 24 erroneous GAMs are successfully detected and discarded. The final recognition score is 1770, which is three times as much as the total amount of matches within the correct GAMs (596). Hence the confidence about the presence of the object is significantly boosted, compared to the simpler approach taken in chapter 5, which just accumulates the number of matches from all model views as score. Moreover, when the object is not in the test image, the confidence score is decreased (see next subsection). As combined effect, the scores assigned in the two cases are more widely separated, which leads to enhanced discriminative power. Figure 7.8a shows the final segmentation, as the total area covered by the 10 selected GAMs. Note the complete and accurate contours.

A challenging case is shown in figure 7.8c. The viewpoint is almost completely from above, and remarkably different from any model view. The object appears twice smaller than in the model views, and is partially occluded by a ball (head) and a plush wildcat (front). 37 GAMs are initially produced, out of which 5 are correct and quite large (43 matches in average, with only one GAM with less than 10). Most of the 32 wrong ones are composed by only few matches. Our method selects all 5 correct GAMs, and 3 small incorrect ones, thereby effectively removing the large majority of mismatches (93%). The recognition score is 581, which is 2.6 times the number of matches among all correct GAMs (216). Note the good quality of the segmentation, which includes even parts of the tail and the left paw. The system has overcome the aforementioned difficulties. Figure 7.8d shows some of the removed GAMs.

The case in figure 7.8b demonstrates recognition in presence of non-rigid deformations. One arm is raised (on the left in the image), the paws face each other and the chest is being compressed. Nevertheless, the system could identify the object (configuration score 1270), and included in the final segmentation also the arm and the chest. The paws were unfortunately missed, probably because too occluded (right paw) and turned so as to hide the bottom part, mostly visible in the model views (left paw). A closer look at the deformed chest allows to better appreciate the behavior of the image-exploration technique (figure 7.9). The pressure applied by the finger causes significant distortions of the texture pattern (top row of the figure). The system copes with this situation by altering the shape of each region in the test image, so as to mirror the wide variation of the local surface orientation (bottom-right). In this fashion, not only the deformed area is covered in the final segmentation, but the individual region matches provide reliable pointwise correspondences throughout.
Figure 7.8: Positive cases for Coleo. a) The example used in the chapter. b) Deformed case. The raised arm and the deformed chest are successfully detected. The minor background blobs are due to a few incorrect GAMs in the selected configuration. c) A challenging case with viewpoint remarkably different from any model view. d) Some of the removed GAMs for the case c).

Effects of additional model views

Although the above reported cases are solved satisfactory based on 8 model views, it is interesting to inspect the effects of including more model views. Figure 7.10-top shows 2 of the 4 additional model views, which are taken from above the object at 90 degrees intervals. Matching also these new model views to the test image of figure 7.8c results in a total of 60 GAMs, including 9 correct. 8 correct GAMs, and 10 incorrect ones, are selected by the best configuration, giving a score of 2498, almost 5 times the total size of correct GAMs (511). Not only the score is much higher than when using 8 model views (581), but especially the ratio to the number of correct matches is larger (2.6). The configuration score grows faster than linearly with the number of compatible GAMs. This realizes the idea that since compatible GAMs reveal consistent hypothesis, the system’s confidence should quickly grow with them. When more model views are available, their larger overlap leads to a
greater number of GAMs and a higher degree of their mutual corroboration. More model views means more cooperation and the proposed approach can effectively measure it. Besides, the segmentation also marginally improves, e.g. it now covers the left paw and more of the head (figure 7.10-bottom).

While including 10 incorrect GAMs might seem a lot at first sight, it must be noted that the other 41 incorrect ones are filtered out. Moreover, these contain only a few matches each (3.8 in average) and their total size makes up only 11% of the mismatches within all 51 incorrect GAMs. Hence, the system behaves well also on the point of removing incorrect GAMs.

**Xmas**

The cylinder-shaped box *Xmas*, also used in section 5.8, constitutes a second validation object. The model is build from 10 views, 6 of which are taken from section 5.8.
While the additional 4 views are not necessary for recognition or qualitative segmentation of the presented cases, they have been included in order to better demonstrate the benefits of multiple-view integration.

The first, fairly simple, example case is pictured in figure 7.11. There are 126 matches in all correct GAMs, and the final score is 620, nearly 5 times as high. An example of two compatible GAMs, matched from very different model views are shown in the bottom row of the figure. Thanks to the rich connections established during modeling, a positive compatibility score is measured for this pair. Including both GAMs helps towards a complete final segmentation.

The second, more challenging, example sees Xmas smaller, occluded and encircled by many clutter objects (figure 7.12). Once again, the score of the produced configuration (1145) is several times greater (5.1) than the number of matches in the correct GAMs (224).
7.4. Results

Figure 7.11: An example for the Xmas object. All 8 correct GAMs are returned. The blob on the lower left is the only accepted incorrect GAM.

Figure 7.12: Another, more challenging, example for Xmas.
7.4. Results

Figure 7.13: Matching Coleo to a similar object. Left: all 122 matches. Right: 42
matches retained by the selected configuration.

7.4.2 Negative cases

The previous examples confirmed that, in positive cases, much higher scores are
produced when using the multiple-view integration approach. However, this is only
useful if it does not happen with negative cases as well! Only then the separation
between the two cases is increased.

Fortunately, the configuration score handles adequately also negative cases: the
pairwise compatibilities are likely to be negative, which leads to higher score with
less GAMs. As a result, some GAMs are not included in the 'best' configuration and
the total score is lower than the sum of all matches. This consideration is confirmed
by several experiments, where the produced scores are lower than the sum of all
matches (about 50%), or at most equivalent. The latter happens when there are
only very few matches and the multiple-view integration scheme cannot measure
GAMs' compatibilities, or in the infrequent cases where some of these happen to be
positive (cooperating mismatches).

For the sake of experimentation, figure 7.13 shows an particular kind of negative case,
where we tried to match the Coleo model (figure 7.1) to a test image containing Leo,
a plush leopard with similar texture and shape. Attracted by the locally similar
structures, image-exploration produced 122 matches, a number that might indicate
object presence, depending on the detection threshold. Nevertheless, the multiple-
view integration approach noticed the inconsistencies among the GAMs and only
kept 6 of the 24 GAMs, totaling 42 matches for a score of 52. Thus, 2/3 of the
(mis)matches have been removed and the recognition score lowered to 40% (from
the 122 initial matches).
7.4.3 Performance of GA

The performance of the Genetic Algorithm (GA) used to maximize the configuration score is considered in this subsection. Two points are especially relevant:

- **Efficiency.** How fast is it? What is the largest problem (number of GAMs) that can still be dealt with exhaustive search? Is GA still efficient for even larger problems?

- **Quality.** How well does it approximate the real optimum?

In order to address these issues, we have analyzed two cases, by first processing them with both the exhaustive search algorithm and the GA, and then comparing the produced solutions. The quality of the GA solution $I_{GA}$ is measured in terms of misclassified GAMs, i.e. the number of GAMs belonging to the real optimum configuration $I_{opt}$ which are not in $I_{GA}$ and vice-versa. Recall that $I_{opt}, I_{GA}$ are binary indicator vectors (subsection 7.3.4), with $I(p) \in \{0,1\}$ indicating whether the $p$-th GAM belongs to the configuration. Hence

$$e = \sum_{p=1..n} \text{abs}(I_{opt}(p) - I_{GA}(p))$$

is the number of misclassified GAMs of $I_{GA}$ ($n$ is the number of GAMs). Since GA is a randomized algorithm, this error measure is averaged over 100 trials.

The two cases considered are the one of figure 7.11 and figure 7.10. The former has 12 GAMs, 8 correct, and the latter has 60 GAMs, 9 correct. Unfortunately the exhaustive search solution cannot be computed for the latter case, as it needs more than 1 hour for a 20 GAM case, and the time is proportional to $2^n$ for $n$ GAMs. Instead, we select the 9 correct GAMs and the 11 largest incorrect ones, so as to form a fair case which contains many incorrect GAMs, and has about the largest size (20) which is still computable by exhaustive search (takes less than 2 hours).

Note that no other case with these characteristics was readily available (either too large or too small).

Table 7.1 reports the results of the investigation. In both cases, the GA approximates very well the real solution, with less than 1 misclassified GAM on average. Moreover, even in the trials where the optimum is not reached exactly, the misclassified GAM(s) is small, so only very few actual matches are misclassified on average. The time measurements demonstrate the need for an approximation algorithm: while the exhaustive search solution is computed within seconds for the 12 GAM case, more than 1 hour is necessary for 20 GAMs. As time requirements grow exponentially, the real optimum cannot be computed in many practical cases, including most of those presented above. The GA approximation instead, runs in some seconds also for the 20 GAM case, and in less than a minute for the full 60 GAM case (unoptimized Matlab implementation).
### 7.5. Related work

If we take a step back from local invariant regions, and look at the wider world of appearance-based Object Recognition, we find much research on modeling 3D objects using multiple training viewpoints. In the ‘aspect graphs’ paradigm [Cyr and Kimia 2001, Weinshall and Werman 1997, Koenderink and van Doom 1979] model views with similar contours are clustered into aspects (compact regions on the viewing sphere). Each aspect is then represented by a single general view (often called ‘prototype’, or ‘characteristic view’). These prototypes can then be seen as a sufficient representation of the object and matched to a test image during recognition [Cyr and Kimia 2001].

The famous work of [Murase and Nayar 1995] injects a large number of complete images of the object into a PCA, so as to obtain a small number of ‘eigenimages’ capturing most of the appearance variation. The object’s multiview appearance becomes a low-dimensional manifold in the ‘eigenspace’ spanned by the eigenimages. Recognition then amounts to projecting the test image into a point in the eigenspace and finding the closest manifold.

Other authors integrate in a single model the curved contours of a smooth 3D object seen in multiple views. In [Vijaykumar et al. 1995] invariants are computed for contour curves in training views coming from an image sequence. Recognition is performed by computing invariants for curves of the test image and index them in a model database. In [Sullivan and Ponce 1998] silhouettes are extracted from a few registered training images and used to create a triangular spline model which captures the topology and rough shape of the object. This kind of approaches is particularly suited for smooth objects (e.g. vases, statues) with no prominent textural features, for which methods based on invariant regions would fail.

A large number (more than 1000) of training views are organized in a global topological chart by [Wieghardt et al. 2002], based on a similarity measure between pairs of views computed by elastic-graph matching [Lades et al. 1993]. The output is

<table>
<thead>
<tr>
<th>Case</th>
<th>GAMs</th>
<th>Real opt</th>
<th>GA opt</th>
<th>Miscs GAMs</th>
<th>Miscs matches / %</th>
</tr>
</thead>
<tbody>
<tr>
<td>figure 7.11</td>
<td>12</td>
<td>620</td>
<td>605</td>
<td>0.63</td>
<td>2.99 / 2.1%</td>
</tr>
<tr>
<td>figure 7.10</td>
<td>20</td>
<td>1519</td>
<td>1479</td>
<td>0.54</td>
<td>7.35 / 1.1%</td>
</tr>
</tbody>
</table>

**Table 7.1:** Quality of GA approximation. **GAMs** is the number of GAMs; **Real opt** is the optimum of the total score function, as computed by exhaustive search; **GA opt** is the optimum computed by the GA, averaged over 100 trials; **Miscs GAMs** is the average number of misclassified GAMs; **Miscs matches / %** is the average number of matches within the misclassified GAMs / this number divided by the total number of matches in all GAMs.
a consistent 'chart' reflecting the neighboring relationships on the viewing sphere. Although the representation is appealing, the authors do not propose a recognition scheme. Pope and Lowe [Pope and Lowe 1996] model the appearance variation of an object on two levels. Large variations are accounted for by dividing the set of training views into clusters of similar views. The smaller variations within each cluster are then modeled by probability distributions over the presence, location and attributes of image features. At recognition time, the model is employed to constrain matching to a test image. Only straight and curved line segments are considered as features, but the approach could possibly be adapted to include invariant regions.

The works discussed above do not represent images as sets of local invariant regions, and, with the exception of [Pope and Lowe 1996], they do not seem adaptable to this paradigm. When turning our attention to local invariant regions, we notice that the large majority of works focus on one model image, or use multiple model images just independently, without an effort to relate them or exploit their interplay [Matas and Obrdzalek 2002, Obrdzalek and Matas 2002, Ferrari et al. 2004c, Schmid 1999, Schmid and Zisserman 1997].

Only very few earlier works try to capture and exploit the relationships among the model views. In [Lowe 2001], similar model views are clustered, and links are made between corresponding features in adjacent clusters. By following the links, a feature from the test image votes for the view to which is matched, and for the adjacent ones. The system gains robustness, because the votes are not dispersed among neighboring model views. In comparison to that work, we believe that our approach offers deeper integration among the model views. Multiple views actively cooperate: by reciprocally (in)validating GAMs arising from different views, they corroborate, or inhibit, the hypothesis of correspondence among parts of the object surface they represent. Moreover, the system arrives at a global recognition score, based on all GAMs and their mutual compatibility as expressed by the model views. This score grows in presence of compatible GAMs, thereby explicitly taking into account that hypothesis shared by multiple model views more reliably indicate the presence of the object. The very organization of region matches into GAMs, which become the new unit of reasoning, is a difference and novelty of our approach.

In [Rothganger et al. 2003], a high degree of multiple-view integration is reached by building a 3D model of the object, prior to recognition. The method imposes two-view and multiview geometric constraints on subsets of matches, and obtains partial reconstructions by factorization. These partial reconstructions are then registered in a global frame by aligning points common to overlapping subsets. In contrast, our method does not build a 3D model. This has the advantage that the selection of model views is far less constrained. Indeed, not all features need to be visible in at least two or three views (but this overlap is exploited whenever it occurs). Moreover, there is no danger of degenerate cases such as views showing only a single planar part. As an additional advantage, our method does not make rigidity assumptions and is
7.6. Conclusion and outlook

This chapter proposed a method to improve the performance of object recognition systems by exploiting the relationships between multiple model views. The result section has demonstrated the method by extending our image-exploration scheme, but we believe the strategy might be applied to other recognition approaches based on local features.

The discriminative power is enhanced due to the higher scores in positive cases, and the segmentation quality improves due to the removal of spurious region matches. Multi-view integration is achieved without rigidity assumptions, and without constructing a 3D model. Since the techniques of this chapter are based on the GAM representation, it shares part of the merit. GAMs are potentially valuable in several contexts of computer vision. This was also demonstrated in the previous chapter with a powerful GAM-based two-view filter. In a sense, GAMs also form an alternative to the elusive concept of 'object parts', in that they offer a perceptual unit between the local features and the global object.

The presented method has been published in [Ferrari et al. 2004a].

One mostly interesting extension to our strategy for multiple-view integration is to make it more active. In the proposed approach the model views are first all matched to the test image separately, while the integration happens only afterwards. This is possibly suboptimal, because the system knows the model connections already before being presented a test image, and therefore can establish topological relationships among them. Hence, it would be computationally profitable to design a more active system, which would start by matching to a single view only and then exploit the model connections to decide if and which other model view to try out. For example, if there are a large number of matches to the first view, it might be superfluous to match also to neighboring views. If there are no matches to the first view, it might be good to try out the opposite view.
The computational complexity of the Clique Partitioning algorithm of section 6.3 is studied here.

Let $N$ be number of vertices in the graph. We consider the complexity as the sum of two terms. The first is the amount of comparisons needed to find the best merging choice for each clique (as in equations (6.5) and (6.6)). The second is the amount of sums needed for merging pairs of cliques satisfying conditions (6.6), via the update equation (6.7). To simplify the analysis, we consider only upper bounds, so the actual number of operations might be somewhat lower than what reported below.

Apart from $N$, the other important factor for the running time is the share $s \in [0, 1]$ of cliques that get merged at each iteration. For example, if $s = 1/2$, then $1/4$ of the cliques merge with another $1/4$ of the cliques, while the remaining $1/2$ of the cliques do not merge. Remember that, at each iteration, each clique either merges with exactly one other clique, or remains untouched. The factor $s$ depends on the structure of the data in the particular problem instance, and is unknown beforehand. Thus, we express the following analysis as a function of $N$ and $s$.

At each iteration, $sN$ cliques merge, and there are $\frac{s}{2}N$ merge operations. In the first iteration, the cost to find all best merging options is (bounded by) $N^2$. From equation (6.7), the cost of each merging operation is of $N - 2$ sums. As said before, to simplify the discussion, we slightly round up the cost to exactly $N$. Hence, the total cost of the first iteration is

$$N^2 + \frac{3}{2}N$$

In the second iteration, there are $(1 - \frac{s}{2})N$ cliques left. Thus, there are $\frac{s}{2}(1 - \frac{s}{2})N$ merging operations. The cost of finding the best merging options is the square of the
number of cliques left: \((1 - \frac{s}{2})N\)^2. The cost of each merging operation is \((1 - \frac{s}{2})N\). Hence, the total cost of the second iteration is

\[
\left(\left(1 - \frac{s}{2}\right)N\right)^2 + \frac{s}{2} \left(1 - \frac{s}{2}\right)N \cdot \left(1 - \frac{s}{2}\right)N
\]

All iterations have the same form. In the \(i\)th iteration there are \((1 - \frac{s}{2})^{i-1}N\) cliques left. The comparisons to find the best merging options are \(((1 - \frac{s}{2})^{i-1}N)^2\). The cost of a merge is \((1 - \frac{s}{2})^{i-1}N\), and there are \(\frac{s}{2}(1 - \frac{s}{2})^{i-1}N\) merges. The total cost of the \(i\)th iteration is thus

\[
\left(\left(1 - \frac{s}{2}\right)^{i-1}N\right)^2 + \frac{s}{2} \left(1 - \frac{s}{2}\right)^{i-1}N \cdot \left(1 - \frac{s}{2}\right)^{i-1}N
\]

After some algebraic manipulations, this formula simplifies to

\[
\left(1 - \frac{s}{2}\right)^{2i} \frac{4 + 2s}{(2 - s)^2} N^2
\]

The first two factors depend by \(s\) and \(i\), but not by \(N\), so we can define an auxiliary function \(h(s, i)\) as

\[
h(s, i) = \left(1 - \frac{s}{2}\right)^{2i} \frac{4 + 2s}{(2 - s)^2}
\]

and rewrite the cost of the \(i\)th iteration as

\[
h(s, i)N^2
\]

The total cost of the whole algorithm is the sum of this expression over all iterations. The number of iterations is at most \(N\), because at each iteration the number of cliques diminishes. Since \(N < \infty\), the total cost of all iterations is bounded by

\[
\sum_{i=1}^{\infty} h(s, i)N^2 = -\frac{(1 - \frac{1}{2}s)^2 (4 + 2s)}{(2 - s)^2 (-s + \frac{1}{4}s^2)} N^2
\]

Again, term in front of \(N^2\) does not depend by \(N\), only by \(s\), so we define

\[
k(s) = \frac{(1 - \frac{1}{2}s)^2 (4 + 2s)}{(2 - s)^2 (-s + \frac{1}{4}s^2)}
\]

and conclude that the total cost of the CP approximation is bounded by

\[
k(s)N^2
\]

It is therefore quadratic in the number of vertices in the graph \(N\). The constant (with respect to \(N\)) \(k(s)\) depends on the structure of the data. In the best case, when all cliques get merged in all iterations, \(s = 1\) and \(k(1) = 2\). In the average
case, only half of the cliques merge in each iterations, while the other half does not: 
\[ s = 0.5 \] and \[ k(0.5) \approx 2.9. \]

Note that \( k(s) \) is approximately linear in \( s \), so even in a particularly bad case it has 
still a small value. For example, when \( s = \frac{1}{16} \), that is just 1 clique every 16 merge 
in an iteration, \( k(s) \approx 16.8. \)

In conclusion, the complexity of the clique partitioning approximation is quadratic 
in \( N \), and has a small constant for any reasonable assumption about the structure 
of the data.
Complexity of surface contiguity filter

The filter of section 5.4 involves finding which pairs of regions intersect in an image, and their intersection area. The simplest implementation is to consider each pair of regions and compute their intersection area via some function $f$. This brute force approach clearly requires $O(n^2)$ time, where $n$ is the number of regions.

In practical situations only a small minority of the pairs intersect, so it is definitely suboptimal to invoke $f$, which is a time-expensive function with elliptical regions, for each pair. A better idea is to check if the bounding-boxes of the two regions intersect before applying $f$, because two regions can only intersect if their bounding-boxes intersect. This approach is much faster, because testing for bounding-box intersection is computationally very cheap, and $f$ is used only a small number of times. Nevertheless, all pairs of regions are tested, and thus the complexity is still $O(n^2)$. In the sequel, we present a simple method which avoids testing all pairs, and thus has lower complexity.

The idea is to exploit an existing Computational Geometry algorithm for finding intersecting line segments. Given $n$ line segments, the algorithm finds all pairs of intersecting segments in $O((n + x) \log(n))$, where $x$ is the number of intersecting pairs. The running time thus does not only depend on the size of the input $n$, but also on the size of the output $x$. However, in practice $x$ is much smaller than the number of all pairs $\frac{n(n-1)}{2}$, hence the speedup. The algorithm is not reported here, but can be found in [de Berg et al. 2000][pp. 20-28], or any other standard computational geometry textbook.

Since our problem deals with regions, it needs some adaptation before the mentioned algorithm can be used. The scheme of the method follows.

1. Replace each region by the four line segments composing its bounding box.

B
2. Find all intersections among these line segments with the fast algorithm discussed above.

3. Each intersecting pair of segments correspond to a pair of intersecting bounding boxes. Hence, the problem of finding which bounding boxes intersect is solved in $O((n + x) \log(n))$. Having actually $4n$ segments does not change the complexity. It only increases the running time by a constant factor.

4. Apply $f$ to pairs of regions whose bounding boxes intersect. Since $f$ takes some constant time $k$ for a pair, the time needed for this step is $kx$, which grows slower than the $x \log(n)$ term. Hence, the total complexity is

$$O((n + x) \log(n))$$

with $x \ll n^2$. 
A fast Intensity Extrema detector

The extraction of the elliptical regions proposed by Tuytelaars and Van Gool [Tuytelaars et al. 1999, Tuytelaars and Van-Gool 2000] requires the detection of local extrema of the image intensities (section 2.4). A pixel is considered an extremum if its greylevel is the highest or lowest in a square neighborhood of side $M$ centered on it. The simplest way to find intensity maxima is to directly compare every pixel with its $M^2 - 1$ neighbors. This requires a number of operations in the order of $O(PM^2)$, where $P$ is the number of pixels in the image. In order to obtain stable and meaningful extrema, $M$ needs to be relatively large: in the implementation of this thesis $M = 21$. Because of this, the total cost of the simplest algorithm is very high: for $M = 21$, more than $400P$ operations are necessary.

This section presents an algorithm to find intensity maxima in time linear in $P$, and independent of $M$, and therefore very fast in practice. The following sections describe the algorithm and analyze its complexity. The minima can be found easily by applying the same algorithm after negating the intensity values.

Step 1: horizontal pass

In the first step the image is scanned horizontally and two lookup tables are prepared. Each pixel of the first lookup table is made to point at the pixel with highest intensity within an horizontal neighborhood of width $M$, centered on the pixel (figure C.1). This maximal intensity is stored in the second lookup table.

What is the cost of this step? When processing a new pixel, the neighborhood shifts 1 pixel to the right. The algorithm reads the intensity of the rightmost pixel in the neighborhood, which is the only ‘new’ one. At this point, there are three possible situations:

- In the first case, the new pixel has higher intensity than the maximum of the previous neighborhood. When this happens (probability $\frac{1}{M+1}$), the algorithm

\[1\text{The processing order is irrelevant, it can be rightwards or leftwards.}\]
Figure C.1: Horizontal pass. The lookup table at the location of the current pixel contains a pointer to the brightest pixel in the horizontal neighborhood. The intensity of this brightest pixel is stored in a second lookup table.

knows the new pixel to be the maxima of the current neighborhood as well, and no further operation is required.

• In the second case, the new pixel has lower intensity than the previous maximum, and the latter is the leftmost pixel of the previous neighborhood. The maximum of the current neighborhood lies within its leftmost $M - 1$ pixels, but the algorithm does not know exactly where. All pixels of the current neighborhood must be read again, which costs $M$ operations. This case also has probability $\frac{1}{M+1}$.

• In the third case, the new pixel has lower intensity than the previous maximum, but this is not the leftmost pixel of the previous neighborhood, i.e. it lies within the current neighborhood. Therefore, the algorithm directly knows it to be the maximum of the current neighborhood. This case occurs with probability $\frac{M-1}{M+1}$.

The expected average cost for a pixel is thus

$$\frac{1}{M+1} + M \frac{1}{M+1} + \frac{M-1}{M+1} = \frac{2M}{M+1}$$

giving a total cost for $P$ pixels

$$P \cdot \left( \frac{2M}{M+1} \right) = \frac{2PM}{M+1}$$
Step 2: vertical pass

In this step, the image is scanned vertically and each pixel is processed as follows. First it is checked if the pixel is the maximum of its own horizontal neighborhood, by a single table lookup. If it is not, the pixel cannot be the maximum of its square neighborhood, hence the algorithm stops. If the pixel is the maximum in its horizontal neighborhood, then it is checked if it has higher intensity than the maximum of another row of its square neighborhood (figure C.2). This can be tested with just another table lookup: it suffices to read the intensity value written in the second lookup table, at the location of the central pixel of the row. If the test is not passed, then the current pixel cannot be the maximum of its square neighborhood, and the algorithm stops. If the test is passed, the other rows are tried one at a time, until all $M$ rows are checked, or until a test fails. If even the last test is passed, the pixel is finally deemed a maximum. Notice that the order in which the rows are tested has no effect on the outcome.

The cost of this vertical pass is very low on the average, because the probability that the $i$th row is checked quickly decreases with $i$. A pixel has $\frac{1}{M}$ probability of having the highest intensity in its own row, and probability $\frac{1}{iM}$ of having the
highest intensity of rows 1,.., i (there are iM pixels in the first i rows). Hence, the probability of a pixel being the maximum of rows 1,.., i, but not of row i + 1 is

\[ \frac{1}{iM} - \frac{1}{(i + 1)M} \]

The cost of checking rows 1,.., i is i. By computing the mean cost, weighted by the corresponding probability, we obtain the expected cost for a pixel

\[ \frac{1}{M} \left( \frac{M - 1}{M} + 2 \left( \frac{1}{M} - \frac{1}{2M} \right) + \cdots + M \left( \frac{1}{(M - 1)M} - \frac{1}{M^2} \right) + \frac{M}{M^2} \right) \]

After some algebraic manipulations, the cost reduces to

\[ \frac{1}{M} \left( 1 + \sum_{i=2}^{M} \frac{i}{i - 1} \right) \]

The term \( \sum_{i=2}^{M} \frac{i}{i - 1} \) equals 2 for \( i = 2 \), and is < 2 for \( i = 3, \ldots, M \). Hence

\[ \frac{1}{M} \left( 1 + \sum_{i=2}^{M} \frac{i}{i - 1} \right) < \frac{1}{M} \left( 1 + (M - 1) \cdot 2 \right) < 2 - \frac{1}{M} \]

which is always < 2 for any M, giving a total cost for P pixels of less than 2P.

**Total complexity**

The sum of costs of the horizontal and vertical steps gives a complexity of \( O(P) \), which has the main advantage of being independent of M. Moreover, only very few image reading operations are necessary per pixel, plus 2 writing operations for preparing the lookup tables.

The expectations of high computational efficiency are met in practice. The intensity extrema (both maxima and minima) are found in about half a second on a 1280x960 pixels image with \( M = 21 \). This figure stays approximately stable for a wide range of values of M (we tried from \( M = 11 \) to \( M = 101 \)).
Complexity of the topological filter

At every iteration, the topological filter verifies the sidedness constraint for all triples of region matches, and computes the number of violated constraints in which each match participates (sections 4.6 and 5.6). We present \(^1\) here an algorithm to achieve this in \(O(n^2 \log(n))\), with \(n\) the number of matches, rather than in \(O(n^3)\). The computational speedup is very high, nearly of factor \(n\). For large \(n\) (i.e. \(n > 100\)), this has dramatic impact on the processing time.

The key observation is that the values of the sidedness (4.3) are not independent for each triple, but interrelated instead. Hence, knowing the sidedness of a triple brings information about the sidedness of all other triples sharing at least a common point.

Since only region centers are used, we consider the input to consist of a set of \(n\) matched point pairs \(\Phi = \{P_1^i \leftrightarrow P_2^i\}_{i=1..n}\) between two views \(V_1, V_2\). The algorithm finds the number of violated constraints in which a given point match \(S \in \Phi\) is involved. It must be run separately for each point match of \(\Phi\).

**Step 1: Cyclic ordering**

The first step sorts all points \(P_1^i \neq S_1\) in order of increasing angle of the direction \(S_1 \rightarrow P_1^i\) (figure D.1). The angles increase in the counter-clockwise orientation. Since we are interested in the cyclic ordering only, the actual angles can be measured with respect to any fixed direction (e.g. the x-axis). The cost of measuring the angle is \(O(1)\) for each point, while the cost of sorting all points is \(O(n \log n)\) using quick-sort [Cormen et al. 1996]. Hence this first step has complexity \(O(n \log n)\).

The operation is repeated for the second view, and the ordering stored separately. In the following, we will consider points \(P_1^i\) as ordered.

**Step 2: Sidedness in the first view**

This step computes the sidedness of all points with respect to the directed line going from \(S_1\) to any other point \(P_1^i \neq S_1\).

\(^1\)The speedup algorithm discussed in this appendix has been designed by Alexander Neubeck.
Figure D.1: A set of $n = 8$ points. Points 1 to 7 are ordered with respect to point $S$. Notice how the ordering is cyclic: there is no real 'first point', and point 1 is the successor of point 7.

The algorithm starts with any $i$ and evaluates the sidedness of point $P_i^{i+1}$, coming after $P_i^i$ in the ordering computed during step 1. If it is on the left side of the line, then the next point's sidedness is evaluated. The procedure iterates until it meets a point on the right side of the line. Thanks to the ordering, all other points are known to lie on the right side (figure D.2a).

Now, the point coming after $P_i^i$ takes up its role, and the algorithm determines the sidedness of all points with respect to the line going from $S_i$ to $P_i^{i+1}$. This task is not performed from scratch, but is carried out incrementally from the outcome of the previous one! Point $P_i^i$ is now on the right side of the line, and all other points which were on the left side stay on the left side. However, there might be some new points on the left side, as the line has rotated counter-clockwise (figure D.2b+c). The sidedness of the next point, after the last one known to be on the left side, is thus evaluated. If it is on the left side, then the next point is considered, and so on until a point on the right side is found.

The procedure is repeated using as second point of the line every $P_i^i$, resulting in the computation of the sidedness of all points with respect to every line through $S_i$. In total, at most $2(n-1)$ sidedness relations are evaluated! Since each sidedness set with respect to any line is in fact an interval in the ordering, it is represented simply by its first and last point. The total cost of incrementally updating all intervals is of $O(n)$.

Step 3: Sidedness in the second view and number of respected constraints

The third step evaluates the sidedness relations in the second view and counts the number of respected constraints in which $S$ takes part. During this step, the algo-
Figure D.2: Computing the sidedness relations in the first view. a) Points 2, 3, 4 are on the left side of the line. b) Point 1 is now on the right side of the current line (solid), while points 3, 4 are still on the left. These facts are known already from a), so the algorithm only needs to consider the sidedness of points starting from 5 (i.e. the first point on the right of the previous line, drawn dashed). After testing point 6, the procedure stops, and knows points 6, 7 to be on the left side. c) In a similar way, when considering the line to point 3, only the sidedness of points 6, 7 need be computed.

The algorithm keeps a list of all points $P^t$ in an order statistic tree [Cormen et al. 1996], or OST for short.

The procedure starts from any point $P_2$ other than $S_2$, considers the line going from $S_2$ to $P_2$, and iteratively evaluates the sidedness of the successors of $P_2$ in the ordering computed during step 1. As soon as a point on the right side is found, the procedure stops. All visited points are on the left side, and are set to 1 in the OST.

In step 2, we have computed which interval of points in $P^t_1$ are on the left side of the line $S_1 \rightarrow P_1$ in the first view. Therefore it suffices to sum the entries of OST over the interval to know the number of respected constraints involving the current line (figure D.3a).

The algorithm continues in a similar way as step 2, and considers the line going from $S_2$ to the successor of $P_2$. As $P_2$ is now on the right side, its entry in the OST is set to 0. The points that were on the left before stay on the left, hence their status in the OST remains unchanged. One by one, the sidedness of each next point is evaluated and the OST entry is set to 1, until a point lying on the right side is met (figure D.3b). The number of respected constraints is now increased by the sum of the OST over the interval of points which are on the left side on the corresponding line in the first view.

Repeating the procedure using as second point of the line every point readily gives the total number of respected constraints involving $S$. This requires at most setting twice to 1 and once to 0 each of the $n - 1$ entries of the OST, and one interval-sum for each of the $n - 1$ points. Setting one entry, or summing over an interval, costs
**Figure D.3:** Sidedness relations in the second view. a) Points 2, 4 are on the left of the line from $S$ to 1 in the second view, while points 2, 3, 4 are on the left side of the corresponding line in the first view (figure D.2a). Entries 2, 4 are set to 1 in the OST, and the sum over the interval 2, 3, 4 gives the number of respected constraints involving the line $S \rightarrow 1$. b) Points 3, 4, to the left of the current line, are set to 1 in the OST. The sum is computed over interval 3, 4, 5, composed of points to the left of the line in the first view.

$O(\log n)$ using an order statistic tree. Hence, the total cost of this third step is $O(n \log n)$.

**Total complexity**

By running the above algorithm for each point match in the input set $\Phi$, one iteration of the topological filter is performed in $O(n^2 \log n)$. However, only the total number of respected constraints is returned, and not which triples respect it. Therefore, the algorithm must be rerun from scratch at every iteration. It is no longer possible to quickly recompute function 4.5 by retrieving and adding up the terms related to the remaining matches. The total computational cost, over all iterations, seems to slip back to $O(n^3 \log n)$. It intuitively seems possible to somehow store and recycle the computed constraints, so as to keep the total complexity below $O(n^3)$, but, at the time of this writing, we did not find a technique yet.
Model images for the Object Recognition experiment

All model views of all the objects in the database used in section 5.8.1 are included here. The purpose is to allow a complete evaluation of the imaging condition changes with respect to the test images. Note that the size of all model images is the same as the size of all test images (720x576).
Acronyms

The full version of acronyms used throughout the thesis can be found in the list below.

CR: The Conflict Resolution algorithm of section 4.3
HC: Harris Corners
GAM: Group of Aggregated Matches
CP: Clique Partitioning
GD: Gradient Descent
LMedS: Least Median of Squares
LP: Linear Program(ming)
MSER: Maximally Stable Extremal Regions
OR: Object Recognition
PCA: Principal Component Analysis
RANSAC: RAndom SAmpled Consensus
RANSAC-EG: ditto, applied to the recovery of the Epipolar Geometry
RF: the Refinement algorithm of section 4.4
SIFT: Scale-Invariant Feature Transform
SFM: Structure-From-Motion
SSD: Sum of Squared Differences
WBS: Wide-Baseline Stereo
Bibliography


Curriculum Vitae

Personal Data

Name          Vittorio Ferrari
Date of birth 8 September 1976
Place of birth Lugano, Switzerland
Citizenship   Swiss

Education

1982 – 1987  Primary school, Lugano and Massagno
1987 – 1991  Secondary school, Lugano
1991 – 1995  High school, Lugano
            Graduation: Matura (Typus C)
1995 – 2000  Studies of Computer Science with a minor in Philosophy at the University of Fribourg
            Graduation: Dipl. Informatik
2000 – 2004  Doctoral student at ETH Zurich, Department of Information Technology and Electrical Engineering, Computer Vision Lab

Occupations

1998 – 1999  Teaching under-assistant at the University of Fribourg, Department of Computer Science
2000 – 2004  Teaching and research assistant at ETH Zurich, Department of Information Technology and Electrical Engineering, Computer Vision Lab