
MODEL-BASED VISUAL TRACKING

The OpenTL Framework

GIORGIO PANIN



WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

MODEL-BASED VISUAL TRACKING

MODEL-BASED VISUAL TRACKING

The OpenTL Framework

GIORGIO PANIN



WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2011 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Panin, Giorgio, 1974–

Model-based visual tracking : the OpenTL framework / Giorgio Panin.

p. cm.

ISBN 978-0-470-87613-8 (cloth)

1. Computer vision–Mathematical models. 2. Automatic tracking–Mathematics. 3. Three-dimensional imaging–Mathematics. I. Title. II. Title: Open Tracking Library framework.

TA1634.P36 2011

006.3'7–dc22

2010033315

Printed in Singapore

oBook ISBN: 9780470943922

ePDF ISBN: 9780470943915

ePub ISBN: 9781118002131

10 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE	xi
1 INTRODUCTION	1
1.1 Overview of the Problem / 2	
1.1.1 Models / 3	
1.1.2 Visual Processing / 5	
1.1.3 Tracking / 6	
1.2 General Tracking System Prototype / 6	
1.3 The Tracking Pipeline / 8	
2 MODEL REPRESENTATION	12
2.1 Camera Model / 13	
2.1.1 Internal Camera Model / 13	
2.1.2 Nonlinear Distortion / 16	
2.1.3 External Camera Parameters / 17	
2.1.4 Uncalibrated Models / 18	
2.1.5 Camera Calibration / 20	
2.2 Object Model / 26	
2.2.1 Shape Model and Pose Parameters / 26	
2.2.2 Appearance Model / 34	
2.2.3 Learning an Active Shape or Appearance Model / 37	

- 2.3 Mapping Between Object and Sensor Spaces / 39
 - 2.3.1 Forward Projection / 40
 - 2.3.2 Back-Projection / 41
- 2.4 Object Dynamics / 43
 - 2.4.1 Brownian Motion / 47
 - 2.4.2 Constant Velocity / 49
 - 2.4.3 Oscillatory Model / 49
 - 2.4.4 State Updating Rules / 50
 - 2.4.5 Learning AR Models / 52

3 THE VISUAL MODALITY ABSTRACTION 55

- 3.1 Preprocessing / 55
- 3.2 Sampling and Updating Reference Features / 57
- 3.3 Model Matching with the Image Data / 59
 - 3.3.1 Pixel-Level Measurements / 62
 - 3.3.2 Feature-Level Measurements / 64
 - 3.3.3 Object-Level Measurements / 67
 - 3.3.4 Handling Mutual Occlusions / 68
 - 3.3.5 Multiresolution Processing for Improving Robustness / 70
- 3.4 Data Fusion Across Multiple Modalities and Cameras / 70
 - 3.4.1 Multimodal Fusion / 71
 - 3.4.2 Multicamera Fusion / 71
 - 3.4.3 Static and Dynamic Measurement Fusion / 72
 - 3.4.4 Building a Visual Processing Tree / 77

4 EXAMPLES OF VISUAL MODALITIES 78

- 4.1 Color Statistics / 79
 - 4.1.1 Color Spaces / 80
 - 4.1.2 Representing Color Distributions / 85
 - 4.1.3 Model-Based Color Matching / 89
 - 4.1.4 Kernel-Based Segmentation and Tracking / 90
- 4.2 Background Subtraction / 93
- 4.3 Blobs / 96
 - 4.3.1 Shape Descriptors / 97
 - 4.3.2 Blob Matching Using Variational Approaches / 104
- 4.4 Model Contours / 112
 - 4.4.1 Intensity Edges / 114
 - 4.4.2 Contour Lines / 119
 - 4.4.3 Local Color Statistics / 122

4.5	Keypoints / 126	
4.5.1	Wide-Baseline Matching / 128	
4.5.2	Harris Corners / 129	
4.5.3	Scale-Invariant Keypoints / 133	
4.5.4	Matching Strategies for Invariant Keypoints / 138	
4.6	Motion / 140	
4.6.1	Motion History Images / 140	
4.6.2	Optical Flow / 142	
4.7	Templates / 147	
4.7.1	Pose Estimation with AAM / 151	
4.7.2	Pose Estimation with Mutual Information / 158	
5	RECURSIVE STATE-SPACE ESTIMATION	162
5.1	Target-State Distribution / 163	
5.2	MLE and MAP Estimation / 166	
5.2.1	Least-Squares Estimation / 167	
5.2.2	Robust Least-Squares Estimation / 168	
5.3	Gaussian Filters / 172	
5.3.1	Kalman and Information Filters / 172	
5.3.2	Extended Kalman and Information Filters / 173	
5.3.3	Unscented Kalman and Information Filters / 176	
5.4	Monte Carlo Filters / 180	
5.4.1	SIR Particle Filter / 181	
5.4.2	Partitioned Sampling / 185	
5.4.3	Annealed Particle Filter / 187	
5.4.4	MCMC Particle Filter / 189	
5.5	Grid Filters / 192	
6	EXAMPLES OF TARGET DETECTORS	197
6.1	Blob Clustering / 198	
6.1.1	Localization with Three-Dimensional Triangulation / 199	
6.2	AdaBoost Classifiers / 202	
6.2.1	AdaBoost Algorithm for Object Detection / 202	
6.2.2	Example: Face Detection / 203	
6.3	Geometric Hashing / 204	
6.4	Monte Carlo Sampling / 208	
6.5	Invariant Keypoints / 211	

7	BUILDING APPLICATIONS WITH OpenTL	214
7.1	Functional Architecture of OpenTL / 214	
7.1.1	Multithreading Capabilities / 216	
7.2	Building a Tutorial Application with OpenTL / 216	
7.2.1	Setting the Camera Input and Video Output / 217	
7.2.2	Pose Representation and Model Projection / 220	
7.2.3	Shape and Appearance Model / 224	
7.2.4	Setting the Color-Based Likelihood / 227	
7.2.5	Setting the Particle Filter and Tracking the Object / 232	
7.2.6	Tracking Multiple Targets / 235	
7.2.7	Multimodal Measurement Fusion / 237	
7.3	Other Application Examples / 240	
APPENDIX A:	POSE ESTIMATION	251
A.1	Point Correspondences / 251	
A.1.1	Geometric Error / 253	
A.1.2	Algebraic Error / 253	
A.1.3	2D-2D and 3D-3D Transforms / 254	
A.1.4	DLT Approach for 3D-2D Projections / 256	
A.2	Line Correspondences / 259	
A.2.1	2D-2D Line Correspondences / 260	
A.3	Point and Line Correspondences / 261	
A.4	Computation of the Projective DLT Matrices / 262	
APPENDIX B:	POSE REPRESENTATION	265
B.1	Poses Without Rotation / 265	
B.1.1	Pure Translation / 266	
B.1.2	Translation and Uniform Scale / 267	
B.1.3	Translation and Nonuniform Scale / 267	
B.2	Parameterizing Rotations / 268	
B.3	Poses with Rotation and Uniform Scale / 272	
B.3.1	Similarity / 272	
B.3.2	Rotation and Uniform Scale / 273	
B.3.3	Euclidean (Rigid Body) Transform / 274	
B.3.4	Pure Rotation / 274	
B.4	Affinity / 275	

B.5 Poses with Rotation and Nonuniform Scale / 277

B.6 General Homography: The DLT Algorithm / 278

NOMENCLATURE 281

BIBLIOGRAPHY 285

INDEX 295

PREFACE

Object tracking is a broad and important field in computer science, addressing the most different applications in the educational, entertainment, industrial, and manufacturing areas. Since the early days of computer vision, the state of the art of visual object tracking has evolved greatly, along with the available imaging devices and computing hardware technology.

This book has two main goals: to provide a unified and structured review of this field, as well as to propose a corresponding software framework, the *OpenTL library*, developed at TUM-Informatik VI (Chair for Robotics and Embedded Systems). The main result of this work is to show how most real-world application scenarios can be cast naturally into a common description vocabulary, and therefore implemented and tested in a fully modular and scalable way, through the definition of a layered, object-oriented software architecture. The resulting architecture covers in a seamless way all processing levels, from raw data acquisition up to model-based object detection and sequential localization, and defines, at the application level, what we call the *tracking pipeline*. Within this framework, extensive use of graphics hardware (GPU computing) as well as distributed processing allows real-time performances for complex models and sensory systems.

The book is organized as follows: In Chapter 1 we present our approach to the object-tracking problem in the most abstract terms. In particular, we define the three main issues involved: models, vision, and tracking, a structure that we follow in subsequent chapters. A generic tracking system flow diagram, the main tracking pipeline, is presented in Section 1.3.

The model layer is described in Chapter 2, where specifications concerning the object (shape, appearance, degrees of freedom, and dynamics), as well as the sensory system, are given. In this context, particular care has been directed to the representation of the many possible degrees of freedom (pose parameters), to which Appendixes 8 and 9 are also dedicated.

Our unique abstraction for visual features processing, and the related data association and fusion schemes, are then discussed in Chapter 3. Subsequently, several concrete examples of visual modalities are provided in Chapter 4.

Several Bayesian tracking schemes that make effective use of the measurement processing are described in Chapter 5, again under a common abstraction: initialization, prediction, and correction. In Chapter 6 we address the challenging task of initial target *detection* and present some examples of more or less specialized algorithms for this purpose.

Application examples and results are given in Chapter 7. In particular, in Section 7.1 we provide an overview of the OpenTL layered class architecture along with a documented tutorial application, and in Section 7.3 present a full prototype system description and implementation, followed by other examples of application instances and experimental results.

Acknowledgments

I am particularly grateful to my supervisor, Professor Alois Knoll, for having suggested, supported, and encouraged this challenging research, which is both theoretical and practical in nature. In particular, I wish to thank him for having initiated the Visual Tracking Group at the Chair for Robotics and Embedded Systems of the Technische Universität München Fakultät für Informatik, which was begun in May 2007 with the implementation of the OpenTL library, in which I participated as both a coordinator and an active programmer.

I also wish to thank Professor Knoll and Professor Gerhard Rigoll (Chair for Man–Machine Communication), for having initiated the Image-Based Tracking and Understanding (ITrackU) project of the Cognition for Technical Systems (CoTeSys [10]) research cluster of excellence, funded under the Excellence Initiative 2006 by the German Research Council (DFG). For his useful comments concerning the overall book organization and the introductory chapter, I also wish to thank our Chair, Professor Darius Burschka.

My acknowledgment to the Visual Tracking Group involves not only the code development and documentation of OpenTL, but also the many applications and related projects that were contributed, as well as helpful suggestions for solving the most confusing implementation details, thus providing very important contributions to this book, especially to Chapter 7. In particular, in this context I wish to mention Thorsten Röder, Claus Lenz, Sebastian Klose, Erwin Roth, Suraj Nair, Emmanuel Dean, Lili Chen, Thomas Müller, Martin Wojtczyk, and Thomas Friedlhuber.

Finally, the book contents are based partially on the undergraduate lectures on model-based visual tracking that I have given at the Chair since 2006. I therefore wish to express my deep sense of appreciation for the input and feedback of my students, some of whom later joined the Visual Tracking Group.

GIORGIO PANIN

CHAPTER 1

INTRODUCTION

Visual object tracking is concerned with the problem of sequentially localizing one or more objects in real time by exploiting information from imaging devices through fast, model-based computer vision and image-understanding techniques (Fig. 1.1). Applications already span many fields of interest, including robotics, man-machine interfaces, video surveillance, computer-assisted surgery, and navigation systems. Recent surveys on the current state of the art have appeared in the literature (e.g., [169,101]), together with a variety of valuable and efficient methodologies.

Many of the low-level image processing and understanding algorithms involved in a visual tracking system can now be found in open-source vision libraries such as the *Intel OpenCV* [15], which provides a worldwide standard; and at the same time, powerful programmable graphics hardware makes it possible both to visualize and to perform computations with very complex object models in negligible time on common PCs, using the facilities provided by the *OpenGL* [17] language and its extensions [19].

Despite these facts, to my knowledge, no wide-scale examples of software libraries for model-based visual tracking are available, and most existing software deals with more or less limited application domains, not easily allowing extensions or inclusion of different methodologies in a modular and scalable way. Therefore, a unifying, general-purpose, open framework is becoming a compelling issue for both users and researchers in the field. This challenging

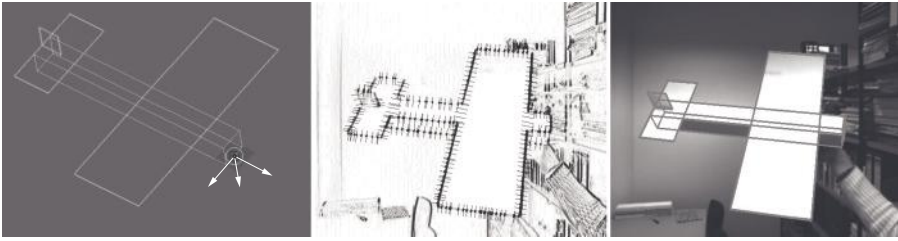


Figure 1.1 Model-based object tracking. *Left*: object model; *middle*: visual features; *right*: estimated pose.

target constitutes the main motivation of the present work, where a twofold goal is pursued:

1. Formulating a common and nonredundant description vocabulary for multimodal, multicamera, and multitarget visual tracking schemes
2. Implementing an object-oriented library that realizes the corresponding infrastructure, where both existing and novel systems can be built in terms of a simple application programming interface in a fully modular, scalable, and parallelizable way.

1.1 OVERVIEW OF THE PROBLEM

The lack of a complete and general-purpose architecture for model-based tracking can be attributed in part to the apparent problem complexity: An extreme variety of scenarios with interacting objects, as well as many heterogeneous visual modalities that can be defined, processed, and combined in virtually infinite ways [169], may discourage any attempt to define a unifying framework. Nevertheless, a more careful analysis shows that many common properties can be identified through the variety and properly included in a common description vocabulary for most state-of-the-art systems. Of course, while designing a general-purpose toolkit, careful attention should be paid from the beginning, to allow developers to formulate algorithms without introducing redundant computations or less direct implementation schemes.

Toward this goal, we begin highlighting the main issues addressed by OpenTL:

- Representing *models* of the object, sensors, and environment
- Performing *visual processing*, to obtain measurements associated with objects in order to carry out detection or state updating procedures
- *Tracking* the objects through time using a prediction–measurement–update loop

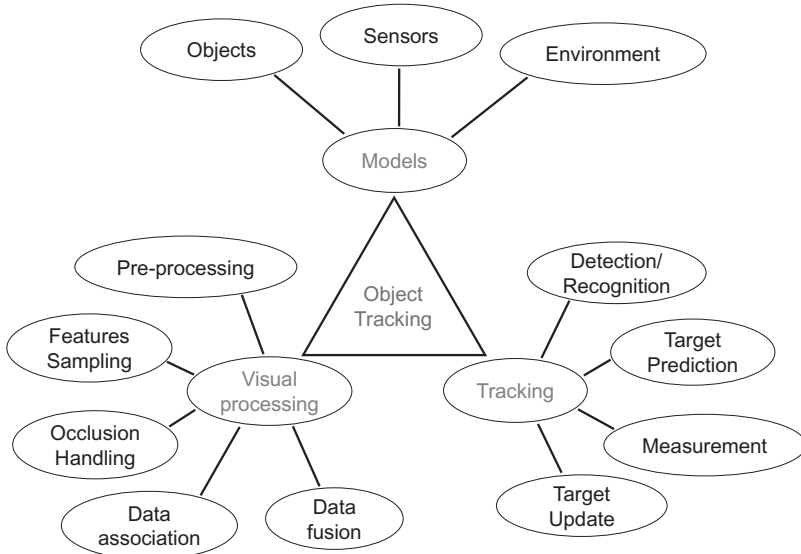


Figure 1.2 Overview of the three main aspects of an object tracking task: models, vision, and tracking.

These items are outlined in Fig. 1.2, and discussed further in the following sections.

1.1.1 Models

Object models consist of more or less specific *prior knowledge* about each object to be tracked, which depends on both the object and the application (Fig. 1.3). For example, a person model for visual surveillance can be represented by a very simple planar shape undergoing planar transformations, and for three-dimensional face tracking a deformable mesh can be used. The *appearance model* can also vary from single reference pictures up to a full texture and reflectance map. Degrees of freedom (or *pose parameters*) define in which ways the base shape can be modified, and therefore how points in object coordinates map to *world coordinates*. Finally, *dynamics* is concerned with a model of the temporal evolution of an object's pose, shape, and appearance parameters.

Models of the sensory system are also required and may be more or less specific as well. In the video surveillance example, we have a monocular, uncalibrated camera where only horizontal and vertical image resolution is given, so that pose parameters specify target motion in *pixel* coordinates. On the other hand, in a stereo or multicamera setup, full calibration parameters have to be provided, in terms of both external camera positions and the

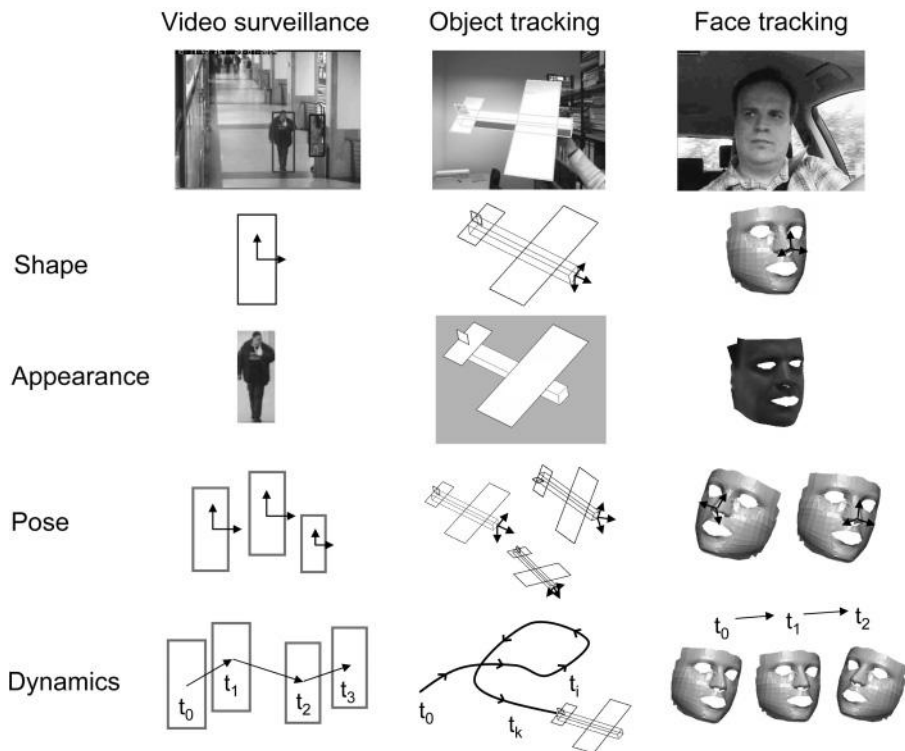


Figure 1.3 Specification of object models for a variety of applications.

internal acquisition model (Chapter 2), while the shape is given in three-dimensional *metric units*.

Information about the *environment* may also play a major role in visual tracking applications. Most notably, when the cameras are static and the light is more or less constant (or slowly changing), such as for video surveillance in indoor environments, a *background model* can be estimated and updated in time, providing a powerful method for detection of generic targets in the visual field. But known obstacles such as tables or other items may also be included by restricting the pose space for the object, by means of penalty functions that avoid generating hypotheses in the “forbidden” regions. Moreover, they can be used to predict external *occlusions* and to avoid associating data in the occluded areas for a given view.¹

¹Conceptually, external occlusions are not to be confused with *mutual occlusions* (between tracked objects) or *self-occlusions* of a nonconvex object, such as those shown in Section 3.2. However, the same computational tools can be used as well to deal with external occlusions.

1.1.2 Visual Processing

Visual processing deals with the extraction and association of useful information about objects from the sensory data, in order to update knowledge about the overall system state. In particular, for any application we need to specify which types of *cues* will be detected and used for each target (i.e., color, edges, motion, background, texture, depth, etc.) and at which level of abstraction (e.g., pixel-wise maps, shape- and/or appearance-related features). Throughout the book we refer to these cues as *visual modalities*.

Any of these modalities requires a *preprocessing* step, which does not depend in any way on the specific target or pose hypothesis but only on the image data, and a *feature sampling* step, where salient features related to the modality are sampled from the visible model surface under a given pose hypothesis: for example, salient keypoints, external contours, or color histograms. As we will see in Chapter 3, these features can also be *updated* with image data during tracking, to improve the adaptation capabilities and robustness of a system.

In the visual processing context, one crucial problem is *data association* or *matching*: assessing in a deterministic or probabilistic way, possibly keeping multiple hypotheses, which of the data observed have been generated by a target or by background clutter, on the basis of the respective models, and possibly using the temporal *state prediction* from the tracker (static/dynamic association). In the most general case, data association must also deal with issues such as missing detections and false alarms, as well as multiple targets with mutual occlusions, which can make the problem one of high computational complexity. This complexity is usually reduced by setting *validation gates* around the positions predicted for each target, in order to avoid very unlikely associations that would produce too-high measurement *residuals*, or *innovations*. We explore these aspects in detail in Chapters 3 and 4.

After data have been associated with targets, measurements from different modalities or sensors must be integrated in some way according to the measurement type and possibly using the object dynamics as well (static/dynamic *data fusion*). Data fusion is often the key to increasing robustness for a visual tracking system, which, by integrating independent information sources, can better cope with unpredicted situations such as light variations and model imperfections.

Once all the target-related measurements have been integrated, one final task concerns how to evaluate the *likelihood* of the measurements under the state predicted. This may involve single-hypothesis distributions such as a Gaussian, or multihypothesis models such as mixtures of Gaussians, and takes into account the measurement residuals as well as their uncertainties (or *covariances*).

As we will see in Chapter 4, the choice of an object model will, in turn, more or less restrict the choice of the visual modalities that can be employed: for

example, a nontextured appearance such as the first two shown in Fig. 1.3 prevents the use of local keypoints or texture templates, whereas it makes it possible to use global statistics of color and edges.

1.1.3 Tracking

When a temporal *sequence* of data is given, we distinguish between two basic forms of object localization: detection and tracking. In the *detection* phase, the system is *initialized* by providing prior knowledge about the state the first time, or whenever a new target enters the scene, for which temporal predictions are not yet available. This amounts to a *global search*, eventually based on the same off-line shape and appearance models, to detect the new target and localize it roughly in pose space. A fully autonomous system should also be able to detect when any target has been lost because of occlusions, or when it leaves the scene, and terminate the track accordingly.

Monitoring the quality of estimation results is crucial in order to detect lost targets. This can be done in several ways, according to the prior models available; we mention here two typical examples:

- *State statistics.* A track loss can be declared whenever the state statistics estimated have a very high uncertainty compared to the dynamics expected; for example, in a Kalman filter the *posterior covariance* [33] can be used; for particle filters, other indices, such as particle *survival diagnostics* [29], are commonly employed.
- *Measurement residuals.* After a state update, measurement residuals can be used to assess tracking quality by declaring a lost target whenever the residuals (or their covariances) are too high.

In the *tracking* phase, measurement likelihoods are used to update overall knowledge of the multitarget state, represented for each object by a more or less generic *posterior* statistics in a Bayesian *prediction–correction* context. Updating the state statistics involves feeding the measurement into a sequential estimator, which can be implemented in different ways according to the system nature, and where temporal *dynamics* are taken into account.

1.2 GENERAL TRACKING SYSTEM PROTOTYPE

The issues mentioned above can be addressed by considering the standard *target-oriented* tracking approach (Fig. 1.4), which constitutes the starting point for developing our framework. The main system modules are:

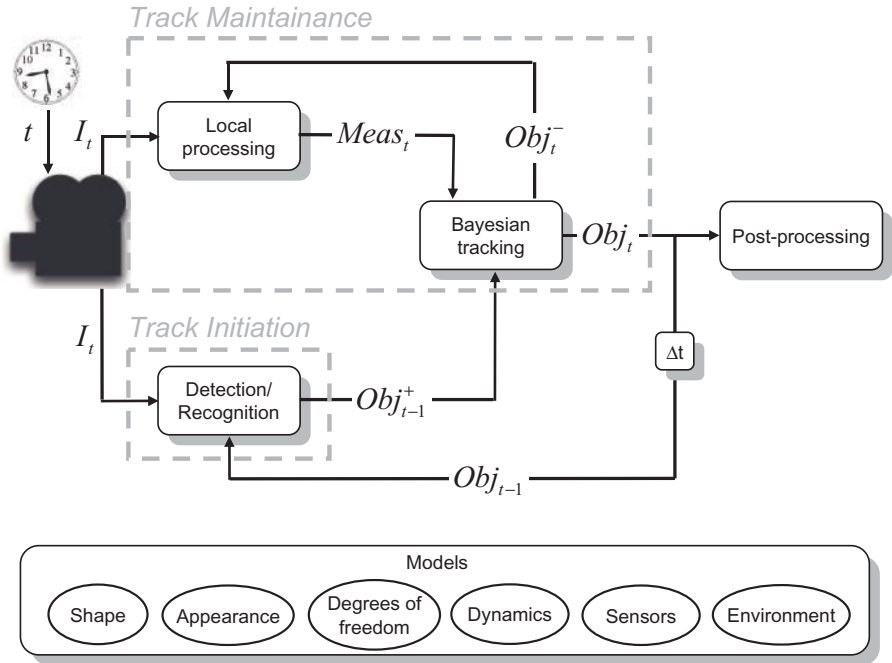


Figure 1.4 High-level view of a target-oriented tracking system.

- *Models*: off-line available priors about the objects and the sensors, and possibly, environment information such as the background
- *Track maintenance*: input devices, measurement processing with local data association and fusion, Bayesian tracking, postprocessing, and visualization of the output
- *Track initiation/termination*: detection and recognition methods for track initialization and termination

In this scheme we denote by Obj a multivariate state *distribution* representing our knowledge of the entire scenario of tracked objects, as we explain in Section 5.1. This representation has to be updated over time using the sensory data I_t from the cameras. In particular, the *track initiation* module processes sensory data with the purpose of localizing new targets as well as removing lost targets from the old set Obj_{t-1} , thus producing an updated vector Obj_{t-1}^+ , while the distribution of maintained targets is not modified. This module is used the first time ($t=0$), when no predictions are available, but in general it may be called at any time during tracking.

The upper part of the system consists of the *track maintenance* modules, where existing targets are subject to *prediction*, *measurement*, and *correction*

steps, which modify their state distribution using the sensory data and models available. In the prediction step, the Bayesian tracker moves the old distributions Obj_{t-1} ahead to time t , according to the given dynamical models, producing the *prior* distribution Obj_t^- .

Afterward, the measurement processing block uses the predicted states Obj_t^- to provide target-associated measurements $Meas_t$ for Bayesian update. With these data, the Bayesian update modifies the predicted prior into the *posterior* distribution Obj_t , which is the output of our system.

In the next section we consider in more detail the track maintenance substeps, which constitute what we call the *tracking pipeline*.

1.3 THE TRACKING PIPELINE

The main tracking pipeline is depicted in Fig. 1.5 in an “unfolded” view, where the following sequence takes place:

1. *Data acquisition.* Raw sensory data (images) are obtained from the input devices, with associated *time stamps*.²
2. *State prediction.* The Bayesian tracker generates one or more predictive *hypotheses* about the object states at the time stamp of the current data, based on the preceding state distribution and the system dynamics.
3. *Preprocessing.* Image data are processed in a model-free fashion, independent of any target hypothesis, providing unassociated data related to a given visual modality.
4. *Sampling model features.* A predicted target hypothesis, usually the average \bar{s}_t^- , is used to sample good features for tracking from the unoccluded model surfaces. These features are *back-projected* in model space, for subsequent re-projection and matching at different hypotheses.
5. *Data association.* Reference features are *matched* against the preprocessed data to produce a set of target-associated *measurements*. These quantities are defined and computed differently (Section 3.3) according

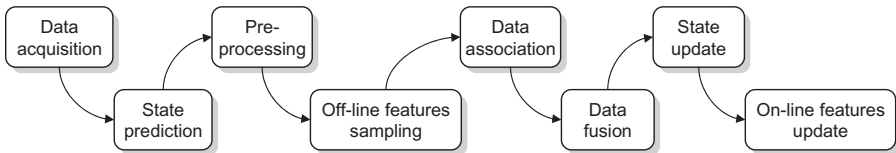


Figure 1.5 Unfolded view of the tracking pipeline.

²In an asynchronous context, each sensor provides independent data and time stamps.

to the visual modality and desired level of abstraction, and with possibly multiple association hypotheses.

6. *Data fusion.* Target-associated data, obtained from all cameras and modalities, are combined to provide a global measurement vector, or a global *likelihood*, for Bayesian update.
7. *State update.* The Bayesian tracker updates the *posterior state statistics* for each target by using the associated measurements or their likelihood. Out of this distribution, a meaningful output-state estimate is computed (e.g., the MAP, or weighted average) and used for visualization or subsequent postprocessing. When a *ground truth* is also available, they can be compared to evaluate system performance.
8. *Update online features.* The output state is used to sample, from the underlying image data, online reference features for the next frame.

An example of a monomodal pipeline for three-dimensional object tracking is shown in Fig. 1.6, where the visual modality is given by local keypoints

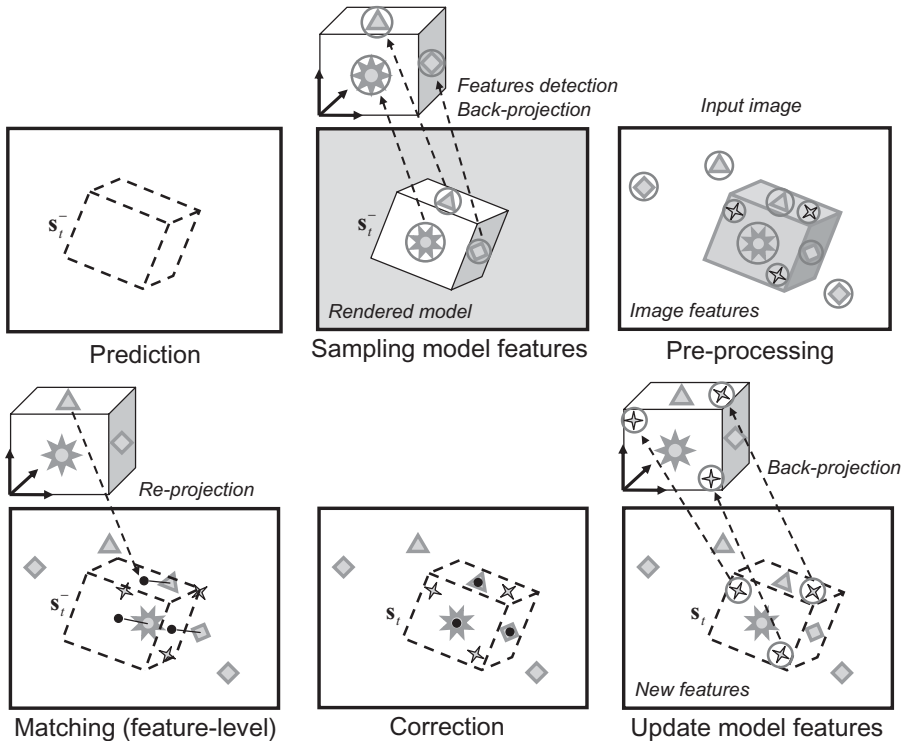


Figure 1.6 Example of a monomodal pipeline for three-dimensional object tracking.

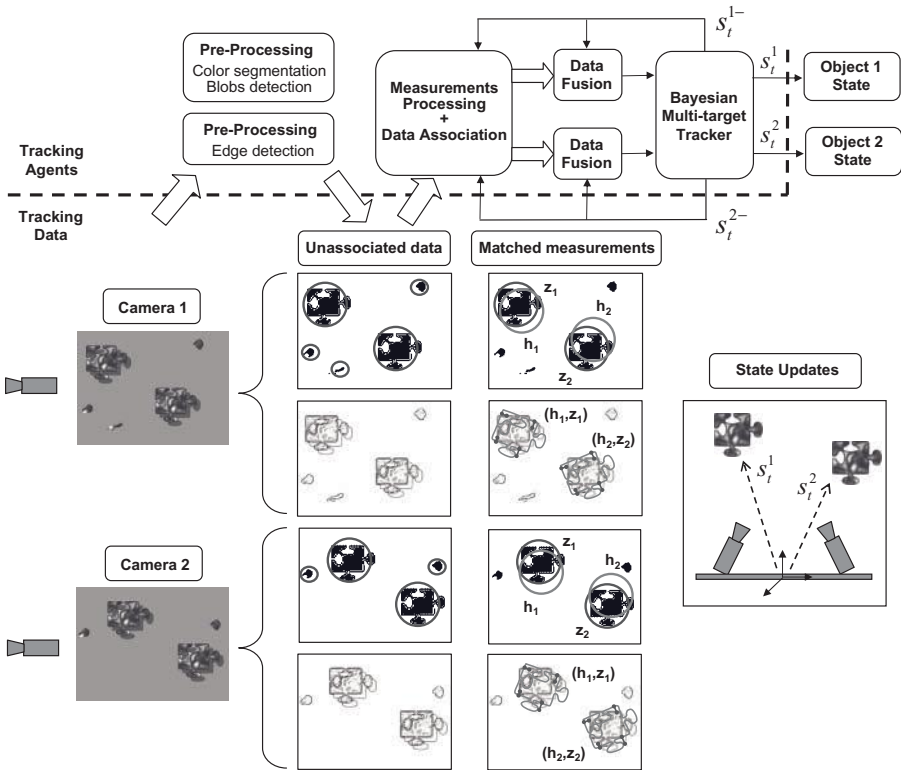


Figure 1.7 Data association and fusion across multiple cameras and modalities.

(Section 4.5). Here, preprocessing consists of detecting local features in the input image, while the average prediction, \bar{s}_t^- , is used to render the object and sample features from the off-line model, by back-projection in object coordinates. Individual hypotheses are used during the matching process, where *re-projected* model features are associated with the nearest-neighbor image data.

After Bayesian correction, residuals are minimized and the output state is estimated; finally, newly detected features (the star-shaped features in the image) are also back-projected onto the object to enrich the model with online data for the next frame. In this sequence we also note how off-line features have a stable appearance, given by the model texture, whereas online features are updated from image data each time, thus making it possible to cope with light variations.

An extension of the pipeline to multimodal/multitarget/multicamera problems is shown in Figs. 1.7 and 1.8. Here, each modality related to each camera provides an independent measurement, $Z_{m,c}^o$, where the three indices refer to

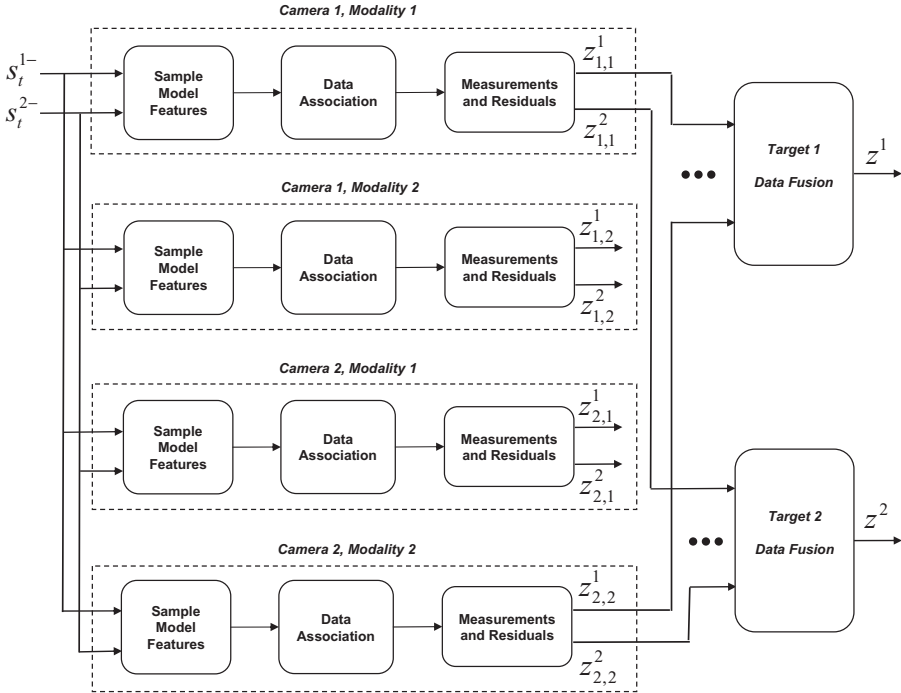


Figure 1.8 Target-oriented measurement processing related to the example of Fig. 1.7.

the object, the modality, and the camera, respectively, while Z^o is the result of data fusion for each target. These operations are computationally quite demanding; therefore, in a complex scenario, parallelizability of these modules may become a critical issue.

CHAPTER 2

MODEL REPRESENTATION

As emphasized in Chapter 1, the OpenTL tracking pipeline operates on the basis of more or less specific, detailed, and complete *prior* information about objects, sensors, and the environment. This information is static and provided offline, and it may be *shared* among subsets of similar targets (e.g., a swarm of similar airplane models) present in the scene.

In our framework, a first crucial task is to represent and store the available priors in a common format, independent of the individual tracking application. This is addressed by the *model layer*, which consists of:

- *Ground shape*: surface geometry, described as a set of *polygonal meshes* in one or more local coordinate systems
- *Ground appearance*: surface appearance, specified by either a set of static reference images or by color, texture, and reflectance maps
- *Degrees of freedom*: which set of pose, deformation, and appearance parameters is going to be estimated during tracking
- *Temporal dynamics*: a probabilistic model of the temporal-state evolution, possibly taking into account mutual interactions in a multitarget scenario
- *Camera model*: available (intrinsic and extrinsic) camera parameters for space-to-image mapping

- *Environment information*: additional information related to the overall scenario: for example, background models for fixed camera views or additional scene items that may interact with the targets

These items may be used in different parts of the tracking pipeline: for example, shape and appearance models are used for visible feature sampling, with pose parameters and camera models for screen projection and back-projection of geometric features, and object dynamics for Bayesian estimation. Environment items may both influence the measurement process, because of partial occlusion of some of the camera views, and interact with the objects, because they occupy part of the parameter space.

2.1 CAMERA MODEL

The camera model defines how points in world coordinates project to individual camera and screen coordinates. To this end, we distinguish between *extrinsic* and *intrinsic* parameters (Fig. 2.6): The former describe the relative position and orientation of a camera in world coordinates, while the latter describe the mapping between three-dimensional camera space and two-dimensional screen coordinates, expressed in *pixels*.

2.1.1 Internal Camera Model

Internal camera parameters provide the *acquisition model*, which is a mapping between three-dimensional camera coordinates and the image plane. Several camera models exist [77, Chap. 6], but we focus primarily on the *pinhole model* (Fig. 2.1). This model is obtained by considering a small hole in the wall of a chamber through which optical rays entering the room are forced to pass, ending on the opposite plane, where an image of the outside world is formed. In the literature, the imaging plane is also called a *retinal plane*, the pinhole

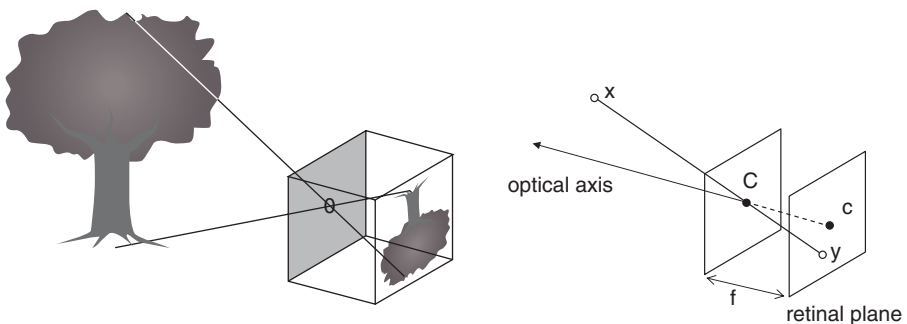


Figure 2.1 Pinhole camera model.

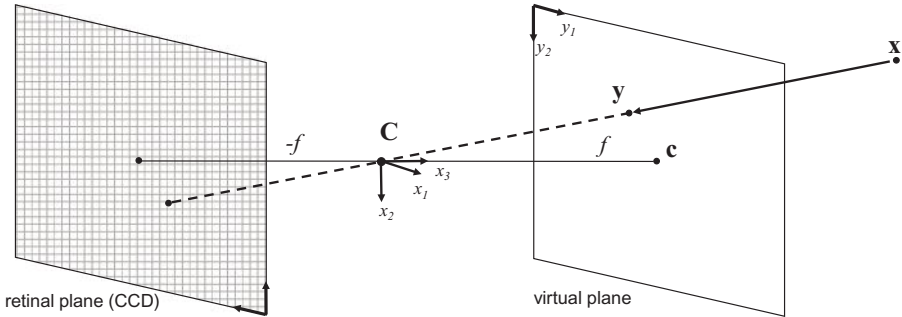


Figure 2.2 Pinhole model using the frontal (or virtual) plane instead of the retinal plane, located on the opposite side of the camera center.

point is the *camera center*¹ C , the main axis orthogonal to the retinal plane and passing through C is the *principal* or *optical axis*, and the intersection of the optical axis with the retinal plane is the *principal point*, c . Finally, the distance between C and c is the *focal length*, usually denoted by f .

However, the retinal image is rotated 180 degrees, so that the image *pixels* acquired by the camera are actually reordered to provide an upright position with the upper-left corner as $(0, 0)$ coordinates, increasing to the right and to the bottom of the image. Therefore, a more convenient way of modeling the acquisition process is to use a *virtual* (or *frontal*) *plane*, which is put in front of the camera along the optical axis, at the same distance f but on the opposite side (Fig. 2.2).

With this choice we can define the origin and the main axes (y_1, y_2) of the image plane, opposite the respective axes of the retinal plane and coherent with the resulting pixel coordinates. Then a natural choice for the camera frame, with origin in C , is given by the axes x_1 and x_2 , respectively, aligned with y_1, y_2 , and x_3 aligned in the optical axis (or *depth*) direction, together giving a *right-handed frame*.

If the coordinates of c and the focal length f are expressed in *metric units* (e.g., meters or millimeters), the camera model is specified by a 3×4 homogeneous projection matrix

$$K = \begin{bmatrix} f & 0 & c_{y_1} & 0 \\ 0 & f & c_{y_2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.1)$$

so that a homogeneous three-dimensional point $\mathbf{x} = (x_1, x_2, x_3, 1)$ projects to $\mathbf{y} = K\mathbf{x}$, which in standard coordinates is given by $(fx_1/x_3 + c_{y_1}, fx_2/x_3 + c_{y_2})$.

¹The effect of an additional *lens* placed between the CCD sensor and the pinhole is basically a *displacement* of C along the optical axis, plus radial distortion and blurring effects.

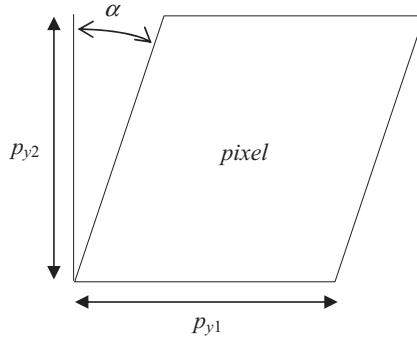


Figure 2.3 Pixel shape and dimensions on the CCD sensor.

This relationship trivially states that for a point lying on the frontal plane ($x_3 = f$), a displacement of 1 unit in the horizontal or vertical direction produces the same displacement on the CCD sensor. In these units, the principal point coordinates are given roughly by the half-sizes of the sensor, and localized more precisely by *calibration* procedures.

However, we are interested in expressing the projected point \mathbf{y} in *pixels*. Therefore, we need to convert this relationship by considering the shape and size of each pixel in the CCD array (Fig. 2.3). In particular, if we denote by p_{y1} and p_{y2} the width and height of a pixel (which may be different if the pixels are not square), we can divide the first row of K by p_{y1} and the second row by p_{y2} , and add a *skew* angle α for nonrectangular pixels (which in most cases is 0), thus obtaining

$$K = \begin{bmatrix} f_1 & \sigma & c_1 & 0 \\ 0 & f_2 & c_2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.2)$$

Here $f_1 = f/p_{y1}$, the normalized focal length in the horizontal direction, can be interpreted as the number of pixels corresponding to a horizontal displacement of 1 unit in metric coordinates for a point lying on the virtual plane; a similar interpretation holds for $f_2 = f/p_{y2}$ but in the vertical direction. $(c_1, c_2) = (c_{y1}/p_{y1}, c_{y2}/p_{y2})$ is the principal point in pixel units, and $\sigma = (\tan \alpha) f_2$ is the corresponding skew factor.

By neglecting σ , we can finally write the projection model in pixel coordinates:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} f_1 \frac{x_1}{x_3} + c_1 \\ f_2 \frac{x_2}{x_3} + c_2 \end{bmatrix} \quad (2.3)$$

which is nonlinear in (x_1, x_2, x_3) .

2.1.2 Nonlinear Distortion

As the focal length of the lens decreases, a linear model such as the pinhole is no longer realistic; the most important deviation is given by *radial distortion*, which causes straight lines in the world to be projected onto curves in the image. This distortion can be modeled as a displacement of the image points by a radial distance (Fig. 2.4), either away or toward the center, respectively, called *barrel* and *pincushion distortion*. It can be incorporated in the camera projection model as follows. Let $\mathbf{x}_c = (x_{c,1}, x_{c,2}, x_{c,3})$ be the camera coordinates of a three-dimensional point and $\tilde{\mathbf{x}} = (x_{c,1}/x_{c,3}, x_{c,2}/x_{c,3}, 1)$ be the *normalized* coordinates of \mathbf{x}_c . In the absence of distortion, the normalized pixel coordinates $\mathbf{y} = (y_1, y_2, 1)$ would be given simply by $\mathbf{y} = K\tilde{\mathbf{x}}$; instead, a radial distortion model first transforms the normalized coordinates $\tilde{\mathbf{x}} = \mathcal{D}(\tilde{\mathbf{x}})$ according to a nonlinear function $\mathcal{D}(\cdot)$ and then applies the calibration matrix to these coordinates, $\tilde{\mathbf{y}} = K\tilde{\mathbf{x}}$.

Nonlinear effects are usually well modeled by purely *radial* terms (up to second order):

$$\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_c \left[1 + k_{r_1}(\tilde{x}_1^2 + \tilde{x}_2^2) + k_{r_2}(\tilde{x}_1^2 + \tilde{x}_2^2)^2 \right] \quad (2.4)$$

where k_{r_1} and k_{r_2} are radial distortion coefficients; the sign of k_{r_1} is positive for a barrel distortion and negative in the other case. Therefore, the projection model becomes $\tilde{y}_1 = c_1 + f_1\tilde{x}_1 + \sigma\tilde{x}_2$ and $\tilde{y}_2 = c_2 + f_2\tilde{x}_2$. By assuming that $\sigma = 0$, we can write it in terms of the undistorted projection \mathbf{y} plus the radial distortion terms

$$\tilde{\mathbf{y}} = \mathbf{y} + (\mathbf{y} - \mathbf{c})[k_{r_1}r^2 + k_{r_2}r^4] \quad (2.5)$$

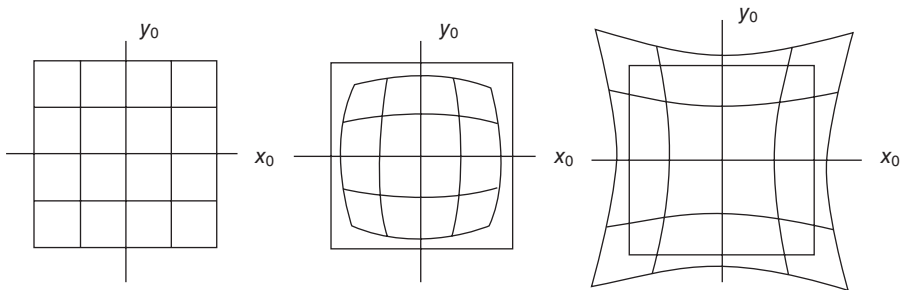


Figure 2.4 Radial distortion. (From [22].)



Figure 2.5 *Left*: image with radial distortion; *right*: corrected image, after estimating the distortion coefficients. (From [101].)

where the radius $r^2 = \tilde{x}_1^2 + \tilde{x}_2^2$ is given in terms of the undistorted, normalized camera coordinates $\tilde{\mathbf{x}}_c$. Figure 2.5 shows an example of radial distortion and correction by inverting the nonlinear warp [eq. (2.5)] and interpolating gray-level values at noninteger pixel coordinates.

2.1.3 External Camera Parameters

The other part of our model concerns the spatial relationship between possibly multiple cameras and the *world reference frame* W (Fig. 2.6). This relationship is expressed by the Euclidean transform

$$T_{c,w} = \begin{bmatrix} R_{c,w} & \mathbf{t}_{c,w} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (2.6)$$

where $R_{c,w}$ is a 3×3 orthogonal rotation matrix and $\mathbf{t}_{c,w}$ is a three-dimensional translation vector, expressing the pose of frame W with respect

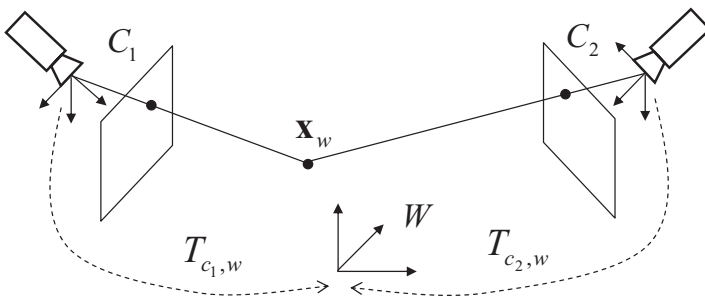


Figure 2.6 Extrinsic and intrinsic camera parameters.

to camera frame C .² Therefore, a world point \mathbf{x} projects to the camera screen by

$$\mathbf{y} = K_c T_{c,w} \cdot \mathbf{x} = K_c^{3 \times 3} [R | \mathbf{t}]_{c,w} \cdot \mathbf{x} = P_{c,w} \cdot \mathbf{x} \quad (2.7)$$

where $P_{c,w}$ is a 3×4 projection matrix and $K_c^{3 \times 3}$ is the left 3×3 submatrix of K_c .

The entire projection matrix $P_{c,w}$ (or, equivalently, the intrinsic and extrinsic camera matrices), which is the goal of camera calibration, can be obtained in several ways: for example, through the direct linear transform (DLT) followed by a maximum-likelihood refinement by means of a *calibration pattern* and feature correspondences.

We notice here that unlike the camera frame C , world and object frames have no predefined convention, so they can be chosen according to the task of interest. For example, when tracking an object that can move onto a plane but never leave the surface, a natural choice for the x_3 axis (of both world and object frames) is the plane normal direction, with origin on the plane itself, so that the object pose parameters can be given as a purely planar (x_1, x_2) motion.

2.1.4 Uncalibrated Models

For some monocular tracking applications, depth estimation is not required, and the shapes can be defined and projected using only two-dimensional pixel coordinates without reference to metric units. In this case the world reference system can be chosen to coincide with the camera frame, and the projection matrix takes the simple form

$$P_{c,w} = K_c = \begin{bmatrix} 1 & 0 & 0 & r_{y1}/2 \\ 0 & 1 & 0 & r_{y2}/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.8)$$

where the only parameters are the horizontal and vertical resolution (r_{y1}, r_{y2}). Notice, however, that in this case the null column is the *third* instead of the last in eq. (2.2), so that the depth coordinate x_3 is ignored.

Finally, we notice how the nonlinearity of the general projection matrix (2.7) derives from the *zooming* effect of projection rays, which are not parallel but converge into the camera center; to obtain a linear model, an approximation that is often made is the *affine camera*. This is obtained by pushing the camera center back to infinity (i.e., the z component of $\mathbf{t}_{c,w}$)

²The ordering C, W reflects the fact that we need this matrix to project points from world to camera coordinates.

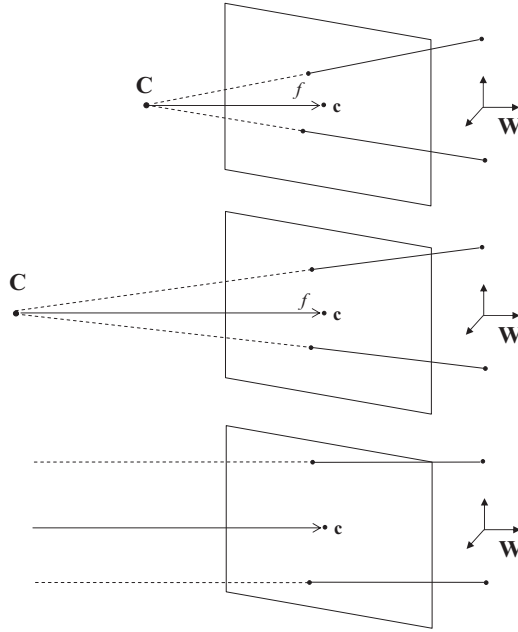


Figure 2.7 Affine camera model (*bottom*) as the limit of a pinhole model, with the camera center at infinity.

while increasing the zooming factor (i.e., the focal length f) by the same amount (Fig. 2.7).

The result is that projection rays become parallel while the frontal plane keeps its position in *world coordinates*. In the limit we have an *affine camera* projection matrix $P_{c,w}$ with focus at infinity. In the most general case, this matrix has the form

$$P_{c,w} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.9)$$

with the property of preserving parallelism (i.e., parallel lines in object space remain parallel on the image). This approximation is valid for small objects compared to their distance to the camera and does not contain any depth or metric information; only pixel coordinates are used.

However, with an affine model we can estimate the approximate size of the object while preserving the linear projection. In fact, this corresponds to the *scaled-orthographic model*, which is obtained by applying the uncalibrated projection (2.8) to a three-dimensional similarity transform for $T_{c,w}$ while including a scale factor a :

$$P_{c,w} = \begin{bmatrix} a\mathbf{r}_1^T & r_{y1}/2 + t_{x1} \\ a\mathbf{r}_2^T & r_{y2}/2 + t_{x2} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (2.10)$$

where \mathbf{r}_1^T and \mathbf{r}_2^T are the first two rows of $R_{c,w}$, and t_{x1} and t_{x2} are the first two components of $\mathbf{t}_{c,w}$; the depth t_{x3} is lost because of the infinite focal length. With 6 degrees of freedom (dof) overall, this model is still less general than eq. (2.9), which has 8 dof and is also nonlinear because of the orthogonal rotation vectors \mathbf{r}_1 and \mathbf{r}_2 that have to be parametrized. Finally, if the scaling factor is fixed at 1, we have an *orthographic projection* with 5 dof.

2.1.5 Camera Calibration

In this section we consider the problem of estimating camera parameters from corresponding geometric entities between world space and image data. In particular, we begin with the direct estimation of the $P_{c,w}$ matrix defined by eq. (2.7). This procedure can be carried out if we have a sufficient number of point correspondences as well as line correspondences, and is also known as *resectioning*. Subsequently, the internal matrix K may be extracted from $P_{c,w}$ by simple decomposition methods.

Following Hartley and Zisserman [77, Chap. 7] we assume that we have N point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{y}_i$, where $\mathbf{y}_i = P\mathbf{x}_i$ for all i , dropping the indices (c,w) for the sake of simplicity. The calibration problem consists of finding P that satisfies as best as possible the equalities in the presence of noise in the data \mathbf{y}_i . In particular, for each correspondence we may consider the *algebraic error*, defined by $\mathbf{y}_i \times P\mathbf{x}_i$. Then if we denote by \mathbf{p}_i^T the i th row of P , it is easy to show that the algebraic error for point i is given by

$$\mathbf{e}_{i,\text{alg}} = \begin{bmatrix} \mathbf{0}^T & -y_{3,i}\mathbf{x}_i^T & y_{2,i}\mathbf{x}_i^T \\ y_{3,i}\mathbf{x}_i^T & \mathbf{0}^T & -y_{1,i}\mathbf{x}_i^T \\ -y_{2,i}\mathbf{x}_i^T & y_{1,i}\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad (2.11)$$

where usually (but not necessarily) $y_{3,i} = 1$ and $\mathbf{p}^T = (\mathbf{p}_1^T, \mathbf{p}_2^T, \mathbf{p}_3^T)$ is the row-wise vectorization of P . We also notice that the three rows of the coefficient matrix in eq. (2.11) are linearly dependent. To remove the redundancy, we can take, for example, the first two rows for each point and stack them together to obtain the *direct linear transform* (DLT) equation

$$\mathbf{e}_{\text{alg}} = \begin{bmatrix} \mathbf{0}^T & -y_{3,1}\mathbf{x}_1^T & y_{2,1}\mathbf{x}_1^T \\ y_{3,1}\mathbf{x}_1^T & \mathbf{0}^T & -y_{1,1}\mathbf{x}_1^T \\ \cdots & \cdots & \cdots \\ \mathbf{0}^T & -y_{3,N}\mathbf{x}_N^T & y_{2,N}\mathbf{x}_N^T \\ y_{3,N}\mathbf{x}_N^T & \mathbf{0}^T & -y_{1,N}\mathbf{x}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix} = \mathbf{0} \quad (2.12)$$

which in the presence of noisy measurements, cannot be satisfied exactly and must be *minimized* with respect to the 12 parameters \mathbf{p} . In particular, since P is defined up to a scale factor, we need at least $5\frac{1}{2}$ -point correspondences in order to estimate the 11 free parameters, meaning that for one of the six points, only one image coordinate needs to be known. In this case, eq. (2.12) has an exact solution, given by the right null space of the $2N \times 12$ coefficient matrix above, which we denote by A .

In the presence of $N \geq 6$ points, and noisy measurements, we can minimize the algebraic error $\|\mathbf{e}_{\text{alg}}\| = \|A\mathbf{p}\|$, subject to an additional normalization constraint such as $\|\mathbf{p}\| = 1$, and the solution is given by the *singular value decomposition* (SVD) of A :

$$A = USV^T \quad (2.13)$$

where:

- U is a square orthogonal matrix $U^T U = I$ with a row size of A , whose columns are the eigenvectors of AA^T .
- V is a square orthogonal matrix $V^T V = I$ with a column size of A , whose columns are the eigenvectors of $A^T A$.
- S is a rectangular matrix with the same size of A , containing on the main diagonal $S_{i,i}$ the square roots of the eigenvalues of $A^T A$ (or AA^T , depending on the smallest between row and column size), and zero elsewhere.

The solution is then obtained by taking the last column of V , corresponding to the minimum singular value S (in this case, \mathbf{v}_{12}). Afterward, the P matrix is reconstructed from the vector \mathbf{p} .

This is not, however, the maximum-likelihood solution, which minimizes the *re-projection* (or *geometric*) error in standard coordinates [77] under the given camera model; therefore, it must be refined subsequently by a nonlinear least-squares optimization. In the latter procedure, the 11 free parameters may also be reduced to a smaller subset if some of them are known or somehow constrained; for example, one may assume equal focal lengths on the two axes, or a zero skew factor.

For the special case of a pinhole model, several algorithms have been proposed to estimate the intrinsic and extrinsic parameters directly, including radial distortion coefficients. For this purpose, we describe here the Zhang calibration method [170, 171]. It consists of two steps: a closed-form solution followed by nonlinear refinement by maximum-likelihood estimation. It

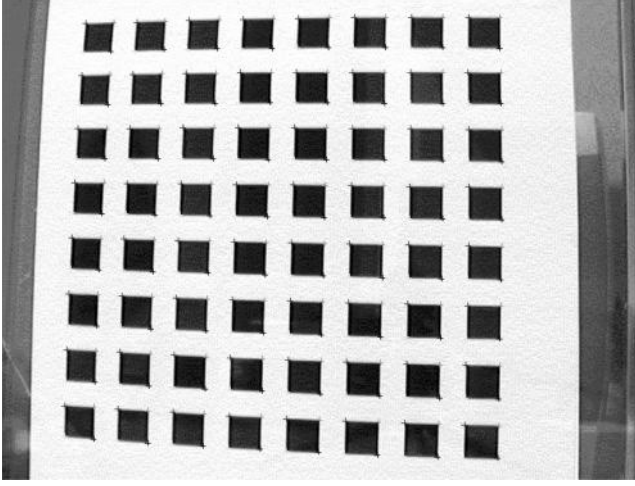


Figure 2.8 Planar calibration pattern, with marked features detected on the image. (From [170]. Copyright © 1999 IEEE.)

requires a planar pattern, such as the one in Fig. 2.8, to be shown to the camera in at least two different orientations.

In particular, let N be the number of points detected on a calibration pattern, and let C be the number of camera views. We consider a world frame solidal with the calibration pattern, with the z axis orthogonal to its plane π , so that a world point on the pattern has coordinates $\mathbf{x}_w = (x_1, x_2, 0, 1)^T$. Therefore, if we denote by \mathbf{r}^i the columns of $R_{c,w}$, the projection equation becomes

$$\mathbf{y} = K \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 0 \\ 1 \end{bmatrix} = K \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \quad (2.14)$$

which is a homography $\mathbf{y} = H_\pi \mathbf{x}_\pi$, with $H_\pi = K \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}$ and \mathbf{x}_π the homogeneous point coordinates on the plane π . This homography can be estimated using the N point correspondences and a maximum-likelihood procedure such as the Levenberg–Marquardt optimization, as explained by Hartley and Zisserman [77, Chap. 4].

The estimated homography defines constraints over the camera parameters

$$[\mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3] = \lambda K \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \quad (2.15)$$

where \mathbf{h}_i are the columns of H_π . In particular, since \mathbf{r}_1 and \mathbf{r}_2 must be orthonormal, we have two equations,

$$\begin{aligned}\mathbf{h}_1^T K^{-T} K^{-1} \mathbf{h}_2 &= 0 \\ \mathbf{h}_1^T K^{-T} K^{-1} \mathbf{h}_1 &= \mathbf{h}_2^T K^{-T} K^{-1} \mathbf{h}_2\end{aligned}\quad (2.16)$$

which provide only *two* constraints on the intrinsic parameters, since a homography has 8 degrees of freedom whereas the extrinsic parameters are 6.

A geometrical interpretation of these constraints is provided by two concepts from projective geometry: the image of the *absolute conic* and the *circular points*. In fact, it can be verified that the model plane is described in camera coordinates by the equation

$$\begin{bmatrix} \mathbf{r}_3^T & \mathbf{r}_3^T \mathbf{t} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = 0 \quad (2.17)$$

where $x_4 = 0$ for points at infinity. The intersection between this plane and the plane at infinity is a line containing the two points $[\mathbf{r}_1^T \ 0]$ and $[\mathbf{r}_2^T \ 0]$ and is therefore given by linear combination

$$\mathbf{x}_\infty = \begin{bmatrix} a\mathbf{r}_1 + b\mathbf{r}_2 \\ 0 \end{bmatrix} \quad (2.18)$$

Next, we consider the absolute conic Ω_∞ , which is a circle of imaginary points on the plane at infinity, with the property of being invariant to similarity transformations [77, Chap. 8.5]. If we compute the intersection of this line with the absolute conic, this requires by definition that $\mathbf{x}_\infty^T \mathbf{x}_\infty = 0$, and therefore

$$(a\mathbf{r}_1 + b\mathbf{r}_2)^T (a\mathbf{r}_1 + b\mathbf{r}_2) = a^2 + b^2 = 0 \quad (2.19)$$

which means that

$$\mathbf{x}_\infty = a \begin{bmatrix} \mathbf{r}_1 \pm i\mathbf{r}_2 \\ 0 \end{bmatrix} \quad (2.20)$$

and their projection on the image plane, up to a scale factor, is

$$\mathbf{y}_\infty = K(\mathbf{r}_1 \pm i\mathbf{r}_2) = \mathbf{h}_1 \pm i\mathbf{h}_2 \quad (2.21)$$

Moreover, we notice that $K^{-T} K^{-1}$ describes the *image* of the absolute conic (IAC) [109], and therefore

$$(\mathbf{h}_1 \pm i\mathbf{h}_2)^T K^{-T} K^{-1} (\mathbf{h}_1 \pm i\mathbf{h}_2) = 0 \quad (2.22)$$

By assigning a value of zero to both the real and imaginary parts of eq. (2.22), we obtain the two constraints (2.16).

If we define the estimation problem in terms of the symmetric matrix $B = K^{-T} K^{-1}$ in the vectorized form $\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]$, we can write

$$\mathbf{h}_i^T B \mathbf{h}_j = \mathbf{v}_{ij}^T B \quad (2.23)$$

where \mathbf{v}_{ij} is a vector containing the products of the entries of \mathbf{h}_i and \mathbf{h}_j , and therefore the two constraints give the linear equation

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0} \quad (2.24)$$

By considering all of the C views, we can stack the respective equations together (after estimating all of the homographies H_∞) and obtain the linear system $\mathbf{A}\mathbf{b} = \mathbf{0}$, with \mathbf{A} a $2C \times 6$ matrix and \mathbf{b} defined up to a scale factor. In particular, for $C \geq 3$, we can solve for \mathbf{b} with the SVD, by taking the singular vector corresponding to the smallest singular value of \mathbf{A} . Afterward, the five parameters in the K matrix are easily computed from B , as shown by Zhang [171].

Extrinsic parameters for each view of the calibration pattern can be recovered using the following formulas:

$$\begin{aligned} \mathbf{r}_1 &= \lambda K^{-1} \mathbf{h}_1 \\ \mathbf{r}_2 &= \lambda K^{-1} \mathbf{h}_2 \\ \mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2 \\ \mathbf{t} &= \lambda K^{-1} \mathbf{h}_3 \end{aligned} \quad (2.25)$$

where $\lambda = 1/\|K^{-1}\mathbf{h}_1\| = 1/\|K^{-1}\mathbf{h}_2\|$. Because of noise in the data, the R matrix is not orthogonal and therefore must be orthogonalized, for example, via the SVD.

This procedure minimizes only an algebraic error measure; therefore, the next step is a MLE by means of nonlinear least squares over the N point correspondences

$$\arg \min_{(K, R_c, \mathbf{t}_c)} \sum_{c=1}^C \sum_{n=1}^N \|\mathbf{y}_{c,n} - \hat{\mathbf{y}}(\mathbf{x}_n, K, R_c, \mathbf{t}_c)\|^2 \quad (2.26)$$

where $\mathbf{y}_{c,n}$ is the observation of point n in image c and $\hat{\mathbf{y}}$ is the projection of model point \mathbf{x}_n according to the camera parameters (K, R_c, \mathbf{t}_c) . All of these quantities are given by nonhomogeneous coordinates.

In particular, for the MLE problem R is parametrized by the rotation vector ρ , through the Rodrigues formula (Appendix B). Nonlinear minimization is carried out again via the Levenberg–Marquardt algorithm, begun from the linear estimate provided by the previous step.

So far, we have assumed a linear projection model. In the case of cameras with significant radial distortion (Section 2.1.2), we can include it in the MLE procedure by estimating the additional parameters k_{r_1} and k_{r_2} of eq. (2.5) after estimating the other parameters, which, if correct, would give us the undistorted pixel coordinates $\hat{\mathbf{y}}$. This leads to the following two equations per point:

$$\begin{bmatrix} (\hat{y}_1 - c_1)r^2 & (\hat{y}_1 - c_1)r^4 \\ (\hat{y}_2 - c_2)r^2 & (\hat{y}_2 - c_2)r^4 \end{bmatrix} \begin{bmatrix} k_{r_1} \\ k_{r_2} \end{bmatrix} = \begin{bmatrix} \tilde{y}_1 - \hat{y}_1 \\ \tilde{y}_2 - \hat{y}_2 \end{bmatrix} \quad (2.27)$$

where $\tilde{\mathbf{y}}$ is the observed point under distortion and $r^2 = (\tilde{x}_1^2 + \tilde{x}_2^2)$ is given in terms of the normalized camera coordinates $(\tilde{x}_1, \tilde{x}_2) = (x_{c,1}/x_{c,3}, x_{c,2}/x_{c,3})$ after the extrinsic transform R, \mathbf{t} .

By taking all of the correspondences, we can stack together the respective equations (2.27) in a $2CN \times 2$ matrix D and solve the system $D\mathbf{k} = \mathbf{d}$, where \mathbf{k} represents the distortion coefficients and \mathbf{d} is a $2CN$ -vector containing the right-hand side of eq. (2.27). The least-squares solution is then given by $\mathbf{k} = (D^T D)^{-1} D^T \mathbf{d}$.

However, this solution assumes that the linear parameters were estimated correctly, although they have been computed under the assumption that $\mathbf{k} = 0$. Therefore, the procedure can be repeated by alternately computing the linear parameters (K, R_c, \mathbf{t}_c) and the distortion coefficients \mathbf{k} until all the parameters converge. We notice that in this phase, only linear methods are adopted, while the maximum-likelihood estimate is deferred to the subsequent step:

$$\arg \min_{(K, k_{r_1}, k_{r_2}, R_c, \mathbf{t}_c)} \sum_{c=1}^C \sum_{n=1}^N \|\mathbf{y}_{c,n} - \tilde{\mathbf{y}}(\mathbf{x}_n, K, k_{r_1}, k_{r_2}, R_c, \mathbf{t}_c)\|^2 \quad (2.28)$$

where the least-squares error is jointly optimized over all parameters. Notice the use of $\tilde{\mathbf{y}}$ for the projection model, given by eq. (2.5). The complete calibration procedure follows.

1. Prepare a planar pattern with known feature points, such as a chessboard.
2. Take C images of the pattern with different orientations.
3. Detect feature points in each image $\mathbf{y}_{c,n}$, corresponding to N given object coordinates \mathbf{x}_n .
4. Use the point correspondences to estimate the planar homographies $H_{\pi,c}$ [eq. (2.14)] between the calibration pattern and each camera view.

5. Alternate the linear estimation of intrinsic [eq. (2.24)] and extrinsic [eq. (2.25)] parameters with the estimation of radial distortion terms [eq. (2.27)] until convergence occurs.
6. For nonlinear refinement, solve the MLE problem [eq. (2.28)] with the Levenberg–Marquardt algorithm, starting with the linear solution.

2.2 OBJECT MODEL

In this section we describe general representations for the object model, encoding all the prior information that concerns its geometric (*shape model*) and photometric (*appearance model*) qualities, as well as their temporal *dynamics*. These prior models will be used later to sample reference features for localization as well as to project local features between object and sensor spaces.

2.2.1 Shape Model and Pose Parameters

Most two- and three-dimensional shapes can be represented in the standard format used in computer graphics as *polygonal meshes* or contours (Fig. 2.9). They can approximate the geometry of any object, ranging from simple polygons up to rigid and articulated bodies with polyhedral as well as smooth surfaces. A polygonal mesh is always defined in a local coordinate system (object frame), usually given by a local origin and two or three axes.

The vertices of the base mesh can move and deform in *world* space according to the respective degrees of freedom that modify the relationship between

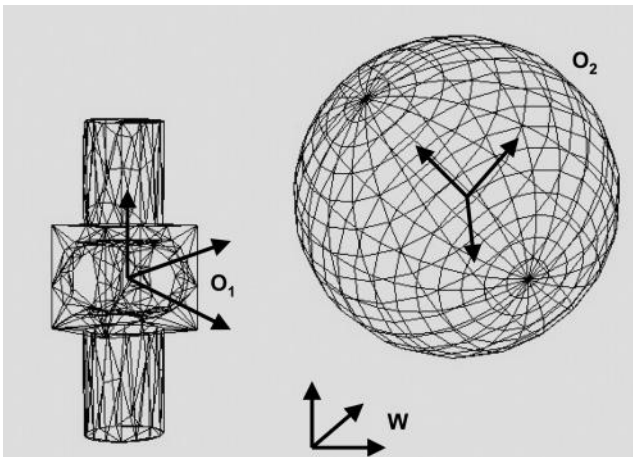


Figure 2.9 Polygonal mesh models and associated object frames.




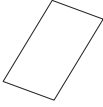
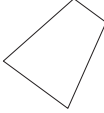
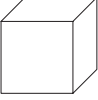
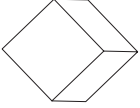

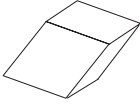
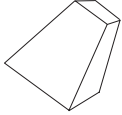
	Base	Euclidean	Similarity	Affine	Homography
2D					
3D					
Invariant properties		Distances Angles Parallel lines Straight lines	Angles Parallel lines Straight lines	Parallel lines Straight lines	Straight lines

Figure 2.10 Single-body projective transforms in two and three dimensions and related invariant properties.

world and object frames. In particular, many single-body three-dimensional transforms (Fig. 2.10) can be represented by a 4×4 homography $T_{w,o}$:

$$\mathbf{x}_w = T_{w,o} \mathbf{x}_o \quad (2.29)$$

which transforms homogeneous body points \mathbf{x}_o into world coordinates, where T belongs to a *submanifold* of the most general space, of dimension 15 since T is defined up to a scale factor; for the sake of uniformity, two-dimensional transforms can also be represented by a 4×4 matrix, with the third row and column both set to 0, which reduces the dimensionality to a maximum of $9 - 1 = 8$ dof. We also notice that, for two-dimensional transforms, only an affine camera can be used, because the depth information x_3 has been suppressed. For example, special Euclidean groups in two and three dimensions, respectively, denoted by $SE(2)$, $SE(3)$, are given by T matrices of the form

$$T = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}' & 1 \end{bmatrix} \quad (2.30)$$

where R is a rotation matrix, which is orthogonal $R^T R = I$ and with positive determinant $|R| = +1$, and \mathbf{t} is a translation vector. Appendix B provides several examples of single-body two- and three-dimensional transforms in more detail.

Each of the manifolds described above has different geometric *invariants*, properties that are preserved under the transformation. For example, Euclidean groups represent motion of rigid bodies, preserving distances between points, as well as parallelism of lines and planes, and angles between lines and planes.

Invariants are very useful properties for object localization, whereby a higher number of degrees of freedom usually implies a lower number of invariants.

Each manifold can also be *parametrized* in different ways with different properties concerning linearity or singular configurations. These parameters are collected into a *pose vector*, hereafter denoted by \mathbf{p} , on which the transformation $T_{w,o}(\mathbf{p})$ depends; without further constraints on the vector \mathbf{p} , its dimension is equal to the manifold dimension, that is, the number of *degrees of freedom* of the object.

Representation *singularities* may arise when the derivatives $\partial T/\partial \mathbf{p}$ (also called *Jacobian matrices*) are not full-rank and may cause significant trouble during the estimation process. For the purpose of tracking, a *singularity-free* representation can be obtained by defining a set of *finite* and *incremental* pose parameters, $(\mathbf{p}, \delta \mathbf{p})$, respectively, where the \mathbf{p} are referred to the previous frame while the $\delta \mathbf{p}$ are estimated by the current tracking loop. In fact, if $\delta \mathbf{p}$ is a different parametrization with respect to \mathbf{p} , it may be defined in a singularity-free way around $\delta \mathbf{p} = 0$, which is always the case for small frame-to-frame motion.

An incremental representation also makes it possible to define the *updating rule* for δp , that is, the way to compute the new pose, \mathbf{p} , from the old one and the increment $(\mathbf{p}, \delta \mathbf{p})$. In the literature, we find primarily the following cases³ (dropping the indices for the sake of simplicity).

- *Additive update:*

$$T_{\text{add}}(\mathbf{p}, \delta \mathbf{p}) = T(\mathbf{p} + \delta \mathbf{p}) \quad (2.31)$$

- *Compositional update:*

$$T_{\text{comp}}(\mathbf{p}, \delta \mathbf{p}) = T(\mathbf{p}) \cdot \delta T(\delta \mathbf{p}) \quad (2.32)$$

The former is intuitive and requires a single parametrization for T while not making any assumptions about the manifold. However, since \mathbf{p} and $\delta \mathbf{p}$ are of the same type, a singularity in \mathbf{p} would result in the same singularity in $\delta \mathbf{p} = 0$. Moreover, computing Jacobian matrices may in some cases be quite complex and depend on the particular parametrization.

On the other hand, the compositional update allows parametrizing $\delta T(\delta \mathbf{p})$ in a different way, by using the *tangent space* representation, which is always singularity-free around $\delta \mathbf{p} = 0$ and easily differentiable for any manifold. A disadvantage, in this case is a less intuitive meaning of δp as well as the requirement for the manifold \mathcal{T} of having the *group* property

$$T = T_1 \cdot T_2 \in \mathcal{T} \quad (2.33)$$

³In Section 4.7.1 we will also see the *inverse* compositional rule, used for invertible planar transforms that preserve the group property.

This is true for most, but not all, of the geometric transforms that can be found in visual tracking applications. Concerning the incremental parameters, Lie algebras [60] provide a unifying representation for the *tangent space* to several transformation groups by means of the *exponential mapping* from the tangent space in $\delta T = I$ onto the manifold T :

$$T(\mathbf{p}, \delta \mathbf{p}) = T(\mathbf{p}) \exp \left(\sum_d \delta p_d G_d \right) \quad (2.34)$$

where G_d are the algebra *generators*. For example, the exponential map to $SE(3)$ is given by

$$T = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}' & 1 \end{bmatrix} = \exp \left(\begin{bmatrix} [\omega]_{\times} & \mathbf{v} \\ \mathbf{0}' & 0 \end{bmatrix} \right) \quad (2.35)$$

where $[\omega]_{\times}$ is the cross-product matrix of the 3-vector ω (Appendix B). The tangent space parameter $\delta p = [\omega, v]^T$ is also called *twist* vector [60] and corresponds to the canonical basis

$$\begin{aligned} G^1 &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, & G^2 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, & G^3 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ G^4 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, & G^5 &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, & G^6 &= \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \quad (2.36)$$

For many problems, the first derivatives of the object-to-world transform are required:

$$J^d(\mathbf{p}) \equiv \left. \frac{\partial}{\partial (\delta p_d)} T(\mathbf{p}, \delta \mathbf{p}) \right|_{(\mathbf{p}, \delta \mathbf{p}=0)} \quad (2.37)$$

where J^d are 4×4 Jacobian matrices for each degree of freedom $d = 1, \dots, D$. Also, Jacobians are computed differently for the two update rules:

$$J_{\text{add}}^d(\mathbf{p}) = \frac{\partial}{\partial (p_d)} T(\mathbf{p}) \quad (2.38)$$

$$J_{\text{comp}}^d(\mathbf{p}) = T(\mathbf{p}) \cdot \left. \frac{\partial}{\partial (\delta p_d)} \delta T(\delta \mathbf{p}) \right|_{\delta \mathbf{p}=0} \quad (2.39)$$

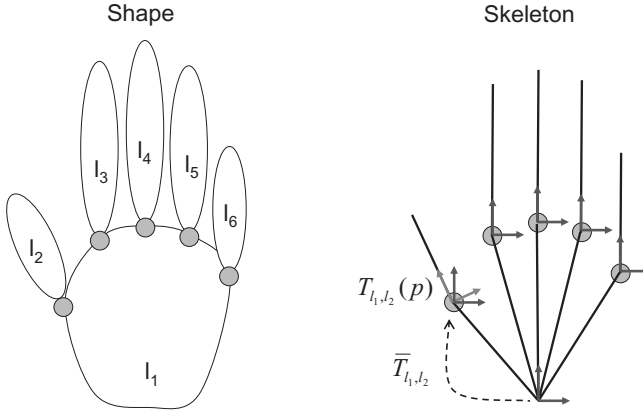


Figure 2.11 Skeleton model with structural interlink transform matrices.

However, in the case of Lie algebras, Jacobians are easily computed using the Lie algebra generators

$$J^d(\mathbf{p}) = T(\mathbf{p})G_d \quad (2.40)$$

In a more general framework, articulated structures can be obtained by defining a *skeleton model*, given by a *tree* of single-body transforms, specifying the intermediate frame locations between connected bodies (Fig. 2.11). These relationships first define the location of each body with respect to its parent (where the parent of the *root* body is W) at *zero* pose, through a set of *reference* transforms between links l_1 and l_2 , which we denote by \bar{T}_{l_1,l_2} . Subsequently, we can add the desired degrees of freedom to this structure by means of the *local* transforms T_{l_1,l_2} , defined by single-body homographies. For example, in Fig. 2.11 we have a structure with two-dimensional translations between palm and fingers, plus local transforms defined by pure rotations around the z axes.

The interlink matrices are shown in Fig. 2.12, and the transformation between two adjacent links $(0, 1)$ is the product of three terms:

1. A *reference* $\bar{T}_{0,1}$, defining the structural transform between links 0 and 1, at zero pose $\mathbf{p} = 0$
2. A *local transform* $T_{0,1}(p)$, the function of a *subset* of the overall pose parameter \mathbf{p}
3. An *incremental transform* $\delta T_{0,1}(\delta\mathbf{p})$, with respect to the local T , that will be estimated during tracking and used to update the local T through one of the rules above

Therefore, the overall world T of frame 1 is computed from the parent node 0 as

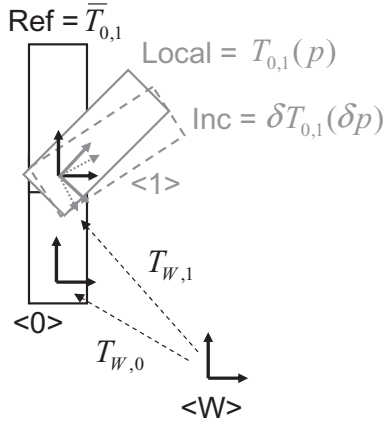


Figure 2.12 Interlink transform matrices for skeleton-based structures.

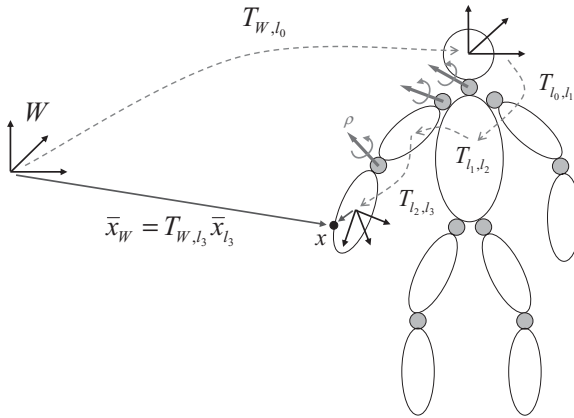


Figure 2.13 Re-projection of points from object-to-world coordinates, through the skeleton transforms.

$$T_{w,1}(\mathbf{p}, \delta\mathbf{p}) = T_{w,0} \cdot (\bar{T}_{0,1} \cdot T_{0,1}(\mathbf{p}) \cdot \delta T_{0,1}(\delta\mathbf{p})) \quad (2.41)$$

Once these matrices have been defined, they can be used to re-project geometric primitives such as points or lines from object-to-world coordinates (Fig. 2.13).

Another multilink structure often encountered in tracking applications is the *active shape model* (ASM) [55], which is a piecewise approximation to deformable surfaces (Fig. 2.14). Each triangle of the mesh represents a surface part, or link, which is transformed according to the motion of its vertices. In particular, moving the vertices induces an *affine* transform for each triangle.

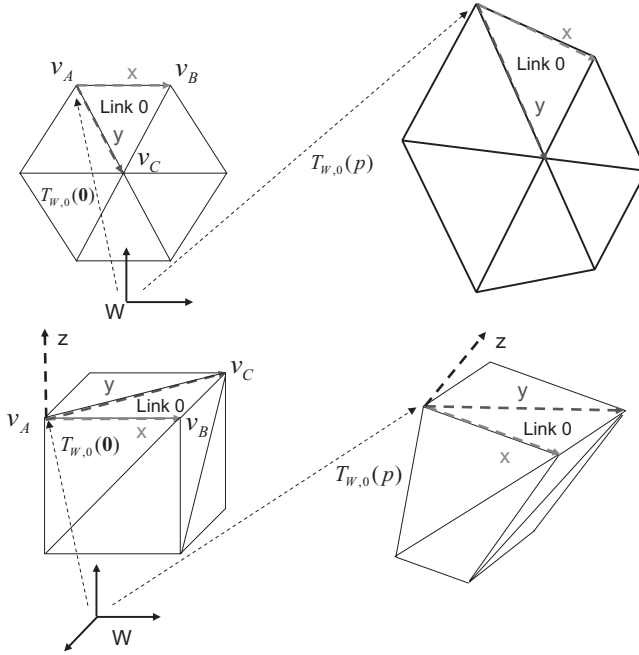


Figure 2.14 Deformable structures in two and three dimensions, approximated by a piecewise affine transform.

By locating the origin of each Cartesian frame on one vertex and the x and y axes on two sides of the triangle, we can easily compute the T matrices between the base shape ($\mathbf{p} = 0$) and the transformed shape, as well as incremental terms δT . In the local reference system, the three vertices of each triangle always have coordinates $(0, 0), (1, 0), (0, 1)$.

Since this deformation induces affine transforms between the world and each triangle, it is also called a *piecewise affine model* [31]. The large number of degrees of freedom, two or three per vertex, can be reduced by means of *subspace projection* methods [54, 55], either defined by the user or learned through a PCA procedure.

We can now define the deformation parameters \mathbf{p} through a linear mapping that generates the corresponding vertices. In particular, let $\mathbf{v}_{0,i}$, $i = 1, \dots, N$ be the average shape in three or two dimensions⁴ at $\mathbf{p} = 0$, and let $\mathbf{v}_{d,i}$, $i = 1, \dots, N$ be the d th deformation vector for each vertex. If we encode the shape in a $3N$ -vector $\mathbf{V}_d = [\mathbf{v}_{d,1}, \dots, \mathbf{v}_{d,N}]^T$, obtained by stacking the vertices of the d th mode together, the ASM defines any shape V as a linear combination (Fig. 2.15):

⁴The planar case is obtained by setting $x_{i,3} = 0, \forall i$.

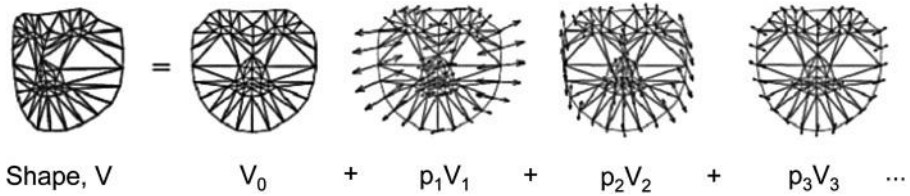


Figure 2.15 Active shape model. The actual shape is obtained by adding a set of linear deformation *modes* to the base vertices. (From [116].)

$$\mathbf{V}(\mathbf{p}) = \mathbf{V}_0 + \sum_{d=1}^D p_d \mathbf{V}_d \quad (2.42)$$

where $D \leq 3N$ (respectively, $2N$) is the number of deformation modes as well as the degrees of freedom of the object. In a more compact form, by defining $\mathcal{V} = [\mathbf{V}_0, \mathbf{V}_1, \dots, \mathbf{V}_D]$ as the *deformation matrix*, we can write

$$\mathbf{V}(\mathbf{p}) = \mathcal{V} \cdot \begin{bmatrix} 1 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} \quad (2.43)$$

Concerning the world transforms $T_{w,l}(\mathbf{p})$ for each triangle l of the mesh, referring again to Fig. 2.14 we can compute them by considering the affine transform induced by the axes $x = \mathbf{v}_{B(l)} - \mathbf{v}_{A(l)}$ and $y = \mathbf{v}_{C(l)} - \mathbf{v}_{A(l)}$ (and $z = x \times y$ for the three-dimensional case) with origin in $\mathbf{v}_{A(l)}$, where $\mathbf{v}_{A(l)}$, $\mathbf{v}_{B(l)}$, and $\mathbf{v}_{C(l)}$ are the three vertices of the triangle. Then, in homogeneous coordinates we have

$$T_{w,l}(\mathbf{p}) = \begin{bmatrix} \mathbf{v}_B - \mathbf{v}_A & \mathbf{v}_C - \mathbf{v}_A & (\mathbf{v}_B - \mathbf{v}_A) \times (\mathbf{v}_C - \mathbf{v}_A) & \mathbf{v}_A \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.44)$$

The term *piecewise affine* derives from the fact that:

- Any point \mathbf{x} belonging to a given triangle l is locally expressed in affine coordinates, given by eq. (2.44) in terms of the respective vertices.
- All of the vertices move according to an affine transform [eq. (2.42)].

It is therefore easy to show that we can rewrite the overall transformation $\mathbf{x}_w = T_{w,l} \mathbf{x}_l$ to world coordinates in a linear form with respect to \mathbf{p} :

$$\mathbf{x}_w = \mathcal{A}_l(\mathbf{x}_l) \mathbf{p} + \mathbf{b}_l(\mathbf{x}_l) \quad (2.45)$$

where \mathcal{A}_l and \mathbf{b}_l are functions of the local coordinates $\mathbf{x}_l \in [0, 1]$. This is very convenient for pose estimation purposes. This structure differs from the

preceding one in that there is no hierarchical tree structure, so that world transforms can be computed independently for each link. Strictly speaking, in this case a compositional update is not defined, since the composition rule does not hold for an arbitrary set of $\delta T_{w,l}$ matrices. In fact, the basic constraint for a set of valid T matrices is that the adjacency of triangles is preserved (i.e., no two triangles are allowed to “split” apart).

2.2.2 Appearance Model

Together with geometric attributes, visual tracking may require a model of the surface appearance, specifying the photometric qualities of the object, under a given viewpoint and light conditions. As can be seen from the literature, appearance models for object tracking can be described in more or less complex ways, ranging from global color and edge statistics up to detailed reflectance maps, accounting for more complex properties such as surface shininess and specularities.

In our framework we classify appearance models into four types, described below.

Key Frames A finite set of *key frames* taken from a range of viewpoints can be used to collect global descriptors such as color histograms in HSV space [53] or histograms of oriented gradients. If key frames are also annotated with pose parameters, more specific features, such as local keypoints and edges, can be sampled and back-projected in object space for localizing the object in intermediate views. More complex techniques such as image-based rendering [90] make it possible directly to obtain intermediate views of the object from the key frames, which can be used for template matching. An example of annotated key frames is given in Fig. 2.16.

Texture Maps A texture map (Fig. 2.17) specifies local reflectance properties in a more detailed and view-independent way, by mapping a texture image onto each vertex of the polygonal mesh, and rendering the surface of any pose using linear, subpixel interpolation of the texture. This procedure is usually accelerated by graphics hardware, also allowing occlusion handling for multiple objects. When a texture map is available, local features such as keypoints, or templates, can be sampled with continuity from the image rendered, providing more reliable matching.



Figure 2.16 Annotated key frames specifying the object appearance at given poses.



Figure 2.17 Texture image mapped onto a polygonal mesh.

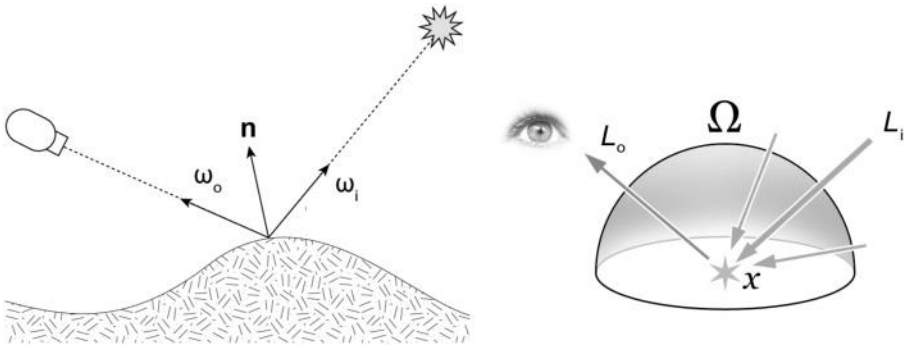


Figure 2.18 Bidirectional reflectance distribution function (BDRF). *Left*: light directions and surface normal (from [2]); *right*: incoming and outgoing radiances in a local hemisphere Ω (from [23]).

Reflectance Map In the most general case, when taking into account the effects of incident light sources, the surface shading and reflectance properties can be modeled by the *bidirectional reflectance distribution function* (BDRF), shown in Fig. 2.18. This is a four-dimensional function $f_r(\omega_i, \omega_o)$ that defines the way that light is reflected from a nontransparent surface. In particular, ω_i and ω_o are the incident and outgoing directions of the light beam (both parametrized by azimuth and elevation angles) referred to the surface normal \mathbf{n} , and the value of f_r defines the ratio between the outgoing and incident *radiances* of the two beams.

When the surface has nonuniform reflectance properties, this function also contains the surface point $f_r(\mathbf{x}, \omega_i, \omega_o)$ in local coordinates; this is also called *spatially varying* BDRF. If, furthermore, we parametrize f_r according to the light *wavelength* λ , we can use it to compute the *rendering equation*,

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (-\omega_i \cdot \mathbf{n}) d\omega_i \quad (2.46)$$

which is an energy conservation law stating that the output radiance L_o in the viewer direction ω_o is the sum of the light emitted (L_e) and the light reflected,

given in turn by the sum of incoming light (L_i) from all directions (ω_i), multiplied by the surface reflection (f_r) and attenuated by the cosine of the incident angle ($\omega_i \cdot \mathbf{n}$). The integral is carried out in a hemisphere Ω defined by \mathbf{x} and its normal \mathbf{n} .

We can parametrize the observed intensity at \mathbf{y} by means of

$$v(\mathbf{y}) = \mathcal{R}_{o,c}(\mathbf{x}_o, \mathbf{n}, \mathbf{a}, T_{c,o}(\mathbf{p})) \quad (2.47)$$

defined per object and camera view, where \mathbf{x}_o is the point on the object surface in local coordinates, \mathbf{n} the normal at \mathbf{x}_o , $T_{c,o}$ the camera-to-object transform (which determines the vector ω_o), and \mathbf{y} the screen projection of \mathbf{x} , so that the color observed depends on both the local reflectance properties at x and the object pose, plus a set of *appearance parameters* \mathbf{a} , including position, intensity, and wavelength of the light sources, which determine the incoming radiance and direction ω_i and, eventually, emitted light L_e .

Active Appearance Model A rather different way of parametrizing the shading variations observed is given by *active appearance models* (AAMs) [54, 116], where an *average texture* A_0 is combined linearly with *texture variations* A_i (see also Fig. 2.19)⁵:

$$A = A_0 + \sum_i a_i A_i \quad (2.48)$$

Although eq. (2.48) may be seen as a special case of eq. (2.47), it does not express directly any underlying physical model of the surface optical properties. As we will see in the next section, an advantage of the AAM is that the basis vectors A_i can be *learned* from example data through PCA analysis, whereby parameters \mathbf{a} are estimated online. Moreover, in the AAM approach the unavoidable *coupling* between the pose and appearance parameters (\mathbf{p}, \mathbf{a}) can also be taken into account, by learning a reduced set of parameters that model the principal *modes* of simultaneous variation.

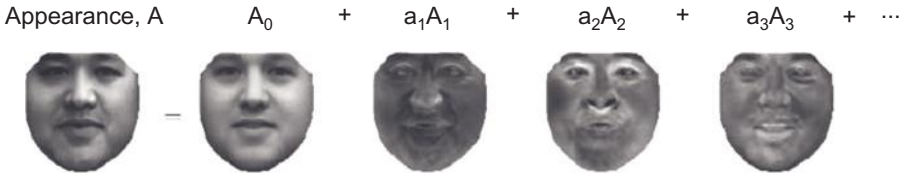


Figure 2.19 Active appearance model. The actual shading is approximated by a linear combination of a base texture and a set of texture variations. (From [116].)

⁵Notice that in this case the reference images A_1, \dots, A_N represent color *variations* from A_0 , and therefore they can also assume negative values.

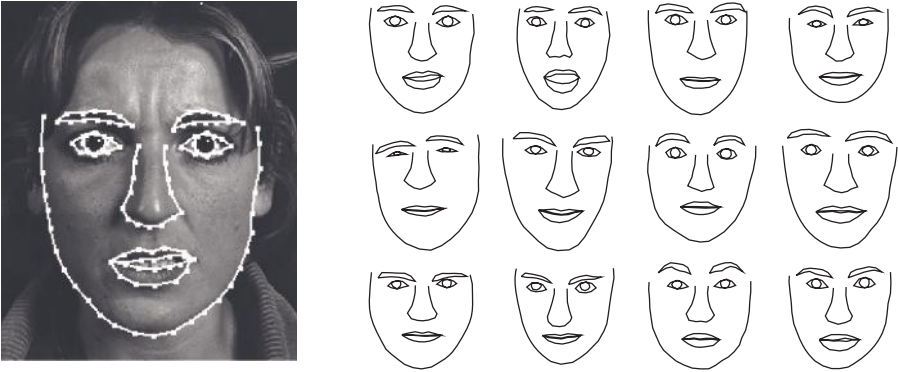


Figure 2.20 ASM training. *Left*: annotated shape; *right*: other examples from the training set. (From [1].)

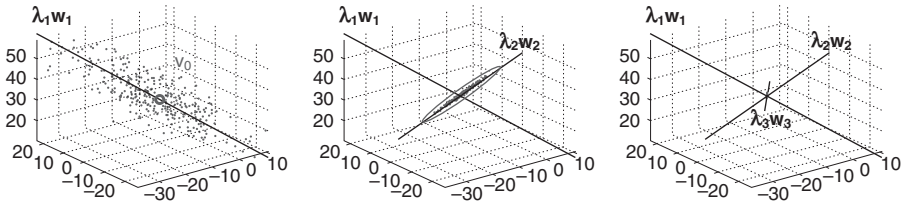


Figure 2.21 PCA procedure.

2.2.3 Learning an Active Shape or Appearance Model

To build an ASM or AAM, a *training* phase is required: for example, through the principal component analysis (PCA) algorithm, explained later. In a first phase, a sequence of example images is provided, where the user shows different poses and expressions of the face in order to cover more-or-less all of the motion and, at the same time, incident light conditions. With this sequence, ground-truth shapes are annotated by hand or, if possible, automated by reliable matching techniques. The left side of Fig. 2.20 shows an example with the vertices of the mesh put into evidence as well as the main contour lines; on the right of the picture, we see a subset of training shapes.

Learning is done via the PCA in the following way: If M is the number of training examples and the model consists of N vertices, one can consider each example as a point \mathbf{v} in a $2N$ -dimensional space, and the M points define a distribution that is usually close to a Gaussian, with a given mean and covariance. Therefore, we can compute an orthogonal basis that expresses the distribution in terms of $2N$ uncorrelated variables (*principal components*), such that the first one corresponds to the direction of maximum variance of the data, the second is the direction of maximum variance after removing the first component, and so on. In this way, one can neglect the last components without significantly affecting the distribution represented, thus reducing the space

dimensionality to N_p , where usually $N_p \ll 2N$. In the example of Fig. 2.21, the third component contains very little information about the data, almost lying on the subspace defined by $(\mathbf{w}_1, \mathbf{w}_2)$.

In more detail, the first principal component of the zero-mean data, $\bar{\mathbf{v}}_i = \mathbf{v}_i - \mathbf{v}_0$, is given by

$$\mathbf{w}_1 = \arg \max_{\|\mathbf{w}\|=1} E_i \left\{ (\mathbf{w}^T \bar{\mathbf{v}}_i)^2 \right\} \quad (2.49)$$

where the mean is computed over all data points; subsequently, it can be removed from the data:

$$\hat{\mathbf{v}}_i^1 = \bar{\mathbf{v}}_i - (\mathbf{w}_1^T \bar{\mathbf{v}}_i) \mathbf{w}_1, \quad i = 1, \dots, 2N \quad (2.50)$$

the second component can be found by

$$\mathbf{w}_2 = \arg \max_{\|\mathbf{w}\|=1} E_i \left\{ (\mathbf{w}^T \hat{\mathbf{v}}_i^1)^2 \right\} \quad (2.51)$$

and so on. This procedure is equivalent to finding the eigenvectors of the *data covariance*: If \bar{X} denotes a $2N \times M$ data matrix containing each training example \mathbf{v} column-wise after subtracting the average \mathbf{v}_0 , the data covariance is given by

$$\Sigma = \bar{X} \bar{X}^T = [(\mathbf{v}_1 - \mathbf{v}_0) \quad \dots \quad (\mathbf{v}_M - \mathbf{v}_0)] \begin{bmatrix} (\mathbf{v}_1 - \mathbf{v}_0)^T \\ \vdots \\ (\mathbf{v}_M - \mathbf{v}_0)^T \end{bmatrix} \quad (2.52)$$

and its orthonormal eigenvectors can be computed by the singular value decomposition (SVD)

$$\bar{X} = USV^T \quad (2.53)$$

as the columns of the orthogonal matrix U , while the singular values are the square roots of the principal variances $S_{i,i} = \sqrt{\lambda_i}$, in descending order $\lambda_1 > \lambda_2 > \dots > \lambda_{2N}$. In this way, one can re-project any shape \mathbf{v} onto this basis,

$$\mathbf{p} = U^T (\mathbf{v} - \mathbf{v}_0) \quad (2.54)$$

and use only the N_p most significant components $\mathbf{w}_i, i = 1, \dots, N_p < 2M$, so that

$$\mathbf{v} \approx \mathbf{v}_0 + \sum_{i=1}^{N_p} p_i \mathbf{w}_i \quad (2.55)$$

The first components coincide with the submatrix consisting of the first N_p columns of U . The eigenvalues λ_i represent the distribution of the source data *energy* among the principal components. Therefore, one criterion for deciding how many terms to keep is given by the cumulative energy:

$$e(i) = \sum_{j=1}^i \lambda_j \quad (2.56)$$

for example, choosing the minimum value N_p such that $e(N_p) > 0.9e(2N)$, which is 90% of the total energy. As in the preceding section we denote the training shapes by $\mathbf{V}_i^{\text{train}}, i = 1, \dots, M$, while the learned components are $\mathbf{V}_0, \dots, \mathbf{V}_{N_p}$.

The same procedure can be employed to learn about the appearance model, with the same training sequence. For this purpose, all images must be warped back to the average shape \mathbf{V}_0 by inverting the respective transformation between $\mathbf{V}_i^{\text{train}}$ and \mathbf{V}_0 and obtaining a set of training appearances $A_i^{\text{train}}, i = 1, \dots, M$, each organized into a vector of N pixel intensities. After applying the PCA to this set, we obtain the basis A_0, A_1, \dots, A_{N_a} where A_0 is the average appearance and A_i are the appearance variations, with $N_a \ll N$.

2.3 MAPPING BETWEEN OBJECT AND SENSOR SPACES

In this section we use the object and camera models described previously to map geometric primitives between object and sensor spaces in a multitarget and multisensor scenario (Fig. 2.22). In particular, we describe how to perform *projection* and *back-projection* of points in the respective spaces, as well as the computation of projected *screen Jacobians*, with respect to the pose parameters.

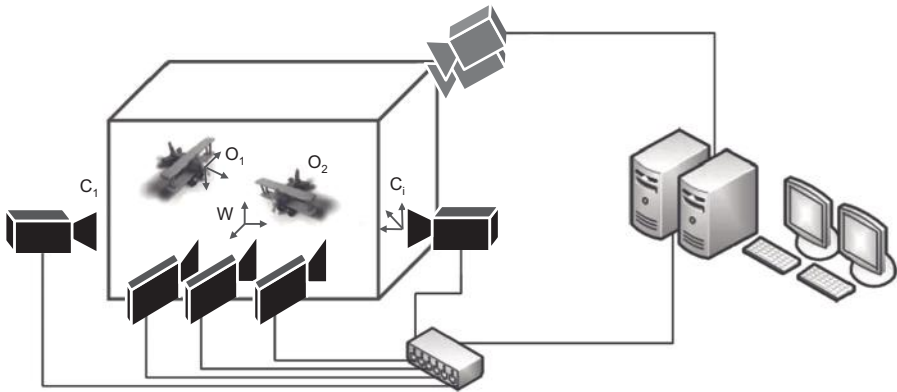


Figure 2.22 World and camera coordinates in a multitarget tracking scenario.

2.3.1 Forward Projection

Mapping points from object o to camera screen c is accomplished by the *warp* function

$$\mathbf{y} = \mathcal{W}_{o,c}(\mathbf{x}, \mathbf{p}^o, P_{c,w}) \quad (2.57)$$

where \mathbf{x} is a point in the local frame of object o , \mathbf{y} is the projection of \mathbf{x} onto camera c screen, \mathbf{p}^o are the pose parameters of o with respect to the world frame W , and $P_{c,w}$ is the camera projection matrix.

All camera models of Section 2.1 are defined by a 3×4 projection matrix $P_{c,w}$. Therefore, the warp function is given by

$$\mathbf{y} = \pi(P_{c,w} T_{w,o}(\mathbf{p}^o) \bar{\mathbf{x}}) \quad (2.58)$$

where $\bar{\mathbf{x}}$ is the object point in homogeneous coordinates ($x_3 = 0$ for the planar case) and

$$\pi(\mathbf{z}) = \begin{bmatrix} z_1 & z_2 \\ z_3 & z_3 \end{bmatrix}^T \quad (2.59)$$

maps from homogeneous to standard coordinates. Although camera parameters are considered constant for object tracking, this formulation can easily be extended to a *navigation* task by also parametrizing the camera matrices $P_{c,w}(\mathbf{p}^c)$ with *camera pose* parameters \mathbf{p}^c , while the objects are in a known world position $\bar{T}_{w,o}$, as well as simultaneous navigation and tracking tasks, where both \mathbf{p}^c and \mathbf{p}^o have to be estimated. We can see from eq. (2.58) that nonlinearity may arise at two stages: the dependence of $T_{w,o}$ on \mathbf{p} and the projection (2.59) for nonaffine cameras.

For estimation and tracking purposes, we also distinguish between finite and incremental pose parameters $(\mathbf{p}, \delta\mathbf{p})$, where the first derivatives of the mapping (2.58) with respect to the incremental parameters are also often required, and are given by

$$\left. \frac{\partial \mathbf{y}}{\partial (\delta p_d)} \right|_{(\mathbf{p}, \delta\mathbf{p}=0)} = \left. \frac{\partial \pi}{\partial \mathbf{z}} \right|_{\mathbf{z}=P_{c,w} T_{w,o}(\mathbf{p}) \bar{\mathbf{x}}} \cdot P_{c,w} \cdot \left. \frac{\partial}{\partial (\delta p_d)} T(\mathbf{p}, \delta\mathbf{p}) \right|_{(\mathbf{p}, \delta\mathbf{p}=0)} \quad (2.60)$$

where

$$\frac{\partial \pi}{\partial \mathbf{z}} = \begin{bmatrix} \frac{1}{z_3} & 0 & -\frac{z_1}{z_3^2} \\ 0 & \frac{1}{z_3} & -\frac{z_2}{z_3^2} \end{bmatrix} \quad (2.61)$$

The last term in eq. (2.60) is the object-to-world Jacobian matrix $J^d(\mathbf{p})$, defined in eq. (2.37), so that

$$\left. \frac{\partial \mathbf{y}}{\partial (\delta p_d)} \right|_{(\mathbf{p}, \delta \mathbf{p}=0)} = \frac{1}{z_3^2} [q_{d,1}z_3 - z_1q_{d,3}; \quad q_{d,2}z_3 - z_2q_{d,3}] \quad (2.62)$$

$$\mathbf{q}_d \equiv P_{c,w} J^d(\mathbf{p}) \bar{\mathbf{x}}$$

where the computation of J^d can be given by either eq. (2.38) or eq. (2.39), according to the update rule, the latter being greatly simplified in the case of Lie algebras (2.40). Finally, we observe that both T and its Jacobians depend only on p and δp , and therefore they can be *precomputed* for a given pose hypothesis before projecting all of the interest points \mathbf{x} .

2.3.2 Back-Projection

Another important issue concerns the inverse projection of points, from sensor to object space, which we call *back-projection*. This is especially important when sampling features from image data (e.g., detected keypoints) back onto the model surface, in order to use them for matching in subsequent frames or at different pose hypotheses.

However, the inverse mapping is not well defined in three-dimensional space, since one dimension has been lost in the image projection, and therefore any point lying on the projection ray may produce the same screen coordinates (Fig. 2.23). This ray is represented, in homogeneous coordinates, by all points $\bar{\mathbf{x}}(\lambda)$ that, for any $\lambda \geq 0$, have the same image $\bar{\mathbf{y}}$:

$$\bar{\mathbf{y}} = P\bar{\mathbf{x}}(\lambda) \quad (2.63)$$

under the camera projection P . This line contains the camera center \mathbf{C} as well as the point on the image plane at a distance f from the center, $\hat{\mathbf{y}} \equiv (y_1, y_2, f)$; therefore, the ray through \mathbf{y} is given by

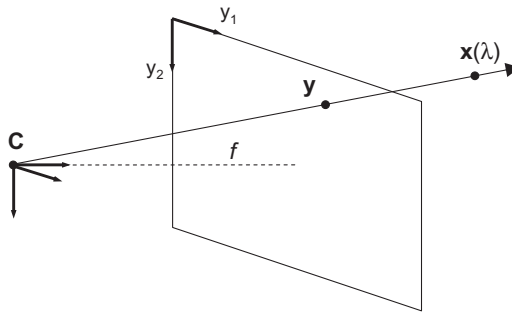


Figure 2.23 Back-projection ray.

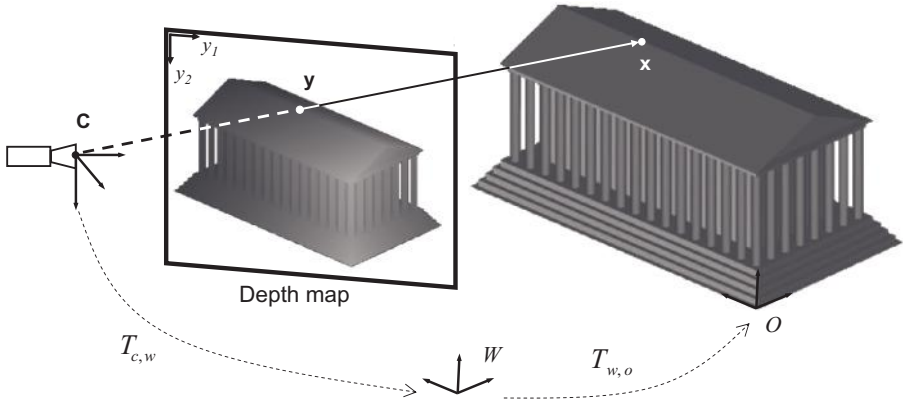


Figure 2.24 Back-projection from screen to object coordinates using the depth map.

$$\mathbf{x}(\lambda) = \lambda \hat{\mathbf{y}} + (1 - \lambda) \mathbf{C} \quad (2.64)$$

Back-projection can be computed if, additionally, we have the *depth* information about \mathbf{x} : that is, x_3 in camera coordinates. This is the case when both the shape model and the pose of the object are known: in fact, a *depth map* can be rendered at the pose specified, where each pixel encodes in a gray-level value the depth of the point projected (Fig. 2.24).

For a pinhole camera, we can separate intrinsic and extrinsic parameters,

$$\mathbf{y} = \pi(K_c T_{c,w} T_{w,o}(\mathbf{p}^o) \bar{\mathbf{x}}) \quad (2.65)$$

and by considering only the intrinsic part K_c , we have

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} f_1 \frac{x_{c1}}{x_{c3}} + c_1 \\ f_2 \frac{x_{c2}}{x_{c3}} + c_2 \end{bmatrix} \quad (2.66)$$

with $\bar{\mathbf{x}}_c = T_{c,w} T_{w,o} \bar{\mathbf{x}}$ expressed in camera coordinates. Since x_{c3} is known, we can recover the other two camera coordinates:

$$\begin{bmatrix} x_{c1} \\ x_{c2} \end{bmatrix} = \begin{bmatrix} \frac{x_{c3}}{f_1} (y_1 - c_1) \\ \frac{x_{c3}}{f_2} (y_2 - c_2) \end{bmatrix} \quad (2.67)$$

Subsequently, we can obtain the object frame coordinates by inverting the extrinsic transform:

$$\bar{\mathbf{x}} = (T_{c,w} T_{w,o}(\mathbf{p}^o))^{-1} \bar{\mathbf{x}}_c \quad (2.68)$$

For affine cameras, the last row of $P_{c,w}$ is $(0, 0, 0, 1)$, and camera depth is not available. In this case we have

$$P_{c,w} T_{w,o}(\mathbf{p}^o) = \begin{bmatrix} A & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.69)$$

with A a 2×3 matrix and \mathbf{t} a 2-vector, so that we can remove $\pi(\cdot)$ from eq. (2.58) and write

$$\mathbf{y} = A(\mathbf{p}^o) \mathbf{x} + \mathbf{t}(\mathbf{p}^o) \quad (2.70)$$

in standard coordinates. This, however, would have infinite solutions in \mathbf{x} and cannot be solved without imposing some further constraint. In particular, if we consider a planar shape, the object frame can always be chosen so that $x_3 = 0$, and the third column of A can be neglected:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = A_{2 \times 2}^{-1}(\mathbf{p}^o) (\mathbf{y} - \mathbf{t}(\mathbf{p}^o)) \quad (2.71)$$

where $A_{2 \times 2}$ is the left block of A .

2.4 OBJECT DYNAMICS

Object dynamics are described by a probabilistic model of the temporal state evolution and can be used in the prediction step of Bayesian tracking [eq. (5.3)]. In the most general case, dynamics may involve pose and appearance parameters as well as their temporal derivatives, and can also model mutual interactions between targets, which we call *ensemble dynamics*. In particular, ensemble dynamics usually contain additional coupling terms with respect to the independent single-target model: for example, mutual repulsion for avoiding overlapping targets, or mutual attraction for modeling group motion.

Dynamics is usually defined from physical laws in terms of continuous-time kinematic variables; however, since visual data are acquired and processed at *discrete time* intervals, for each model we need to define the equivalent, or approximated, discrete-time version. In particular, we denote by t the continuous time, in seconds, sampled at discrete *time stamp* t_k ,

corresponding to the time at which image data have been acquired,⁶ where $k = 0, 1, \dots$.

To define the model, we also need to specify the *state* variables s_k^o for each target, which may contain positional as well as kinematic information. As mentioned above, for simultaneous targets we distinguish between independent and ensemble dynamics, the latter defined over the *ensemble state*:

$$\mathbf{s}_k \equiv (s_k^1, \dots, s_k^O) \quad (2.72)$$

of all objects at time t_k . With this distinction in mind, a discrete model can be given in one of two equivalent forms:

1. Explicit, or *generative model*:

$$s_k^o = f^o(s_{k-1}^o, w_k^o, \tau_k); \mathbf{s}_k = \mathbf{f}(\mathbf{s}_{k-1}, \mathbf{w}_k, \tau_k) \quad (2.73)$$

2. Implicit, or *discriminative model* (also called *predictive prior*):

$$P(s_k^o | s_{k-1}^o, \tau_k); P(\mathbf{s}_k | \mathbf{s}_{k-1}, \tau_k) \quad (2.74)$$

where w_k^o is the *process noise*, with known statistics, and $\tau_k = t_k - t_{k-1}$ is the lag between time stamps; in most cases [33], w is a discrete zero-mean white Gaussian noise with a given covariance matrix $W(\tau)$, increasing with τ . We also notice how each target may have a different dynamics; therefore, the function f is indexed by o as well.

In the following we introduce a set of linear models for a single target. For this purpose we start from the continuous-time version, which involves temporal derivatives $\dot{s}_t = (\dot{\mathbf{p}}_t, \dot{\mathbf{p}}_t, \dot{\mathbf{p}}_t, \dots)$; the linear case is given by the first-order differential equation

$$\dot{s}(t) = A(t)s(t) + B(t)w(t) \quad (2.75)$$

When converted to discrete samples t_k , the sampling interval τ is used in the propagation model; in fact, the evolution from time t_{k-1} to time t_k can be obtained by integration:

$$s(t_k) = F(t_{k-1}, t_k)s(t_{k-1}) + \int_{t_{k-1}}^{t_k} F(t_{k-1}, \tau)B(\tau)w(\tau)d\tau \quad (2.76)$$

where $F(t_{k-1}, t_k)$ is the state *transition matrix*, which under mild conditions on $A(t)$ is given by

⁶For the more realistic case of *asynchronous measurements*, time stamps are referred to the *acquisition time* of the respective sensory data. Therefore, in a fully asynchronous dynamic fusion context [34], the tracker should perform different state predictions for each camera, and perform separate state updates as well.

$$F(t_{k-1}, t_k) = e^{\int_{t_{k-1}}^{t_k} A(\tau) d\tau} \quad (2.77)$$

and for a time-invariant system (where A is independent on t) it becomes the exponential matrix

$$F(t_{k-1}, t_k) \equiv F_k(\tau_k) = e^{A\tau_k} \quad (2.78)$$

In this case, the second term in eq. (2.76), under the assumption of zero-mean white Gaussian noise $w(t)$ with autocorrelation matrix $W(t)$, is also a discrete noise w_k , with zero mean and covariance matrix:

$$Q_k(\tau_k) = \int_{t_{k-1}}^{t_k} e^{(t_k-\tau)A} B W(\tau) B^T e^{(t_k-\tau)A^T} d\tau \quad (2.79)$$

so that the discrete-time model can be written

$$s_k = F_k(\tau_k) s_{k-1} + W_k(\tau_k) w_k \quad (2.80)$$

where $W_k W_k^T = Q_k$ and w_k has unit covariance; this is a Markov process, where both the transition matrix and the noise covariance are parametrized by τ_k . If we also include an *average state* \bar{s} ,

$$s_k - \bar{s} = F_k(\tau_k)(s_{k-1} - \bar{s}) + W_k(\tau_k) w_k \quad (2.81)$$

we can also model constrained motion, such as oscillatory dynamics.

Another class of linear time-invariant models is given by the autoregressive (AR) process of order n :

$$\mathbf{p}_k = F^1 \mathbf{p}_{k-1} + F^2 \mathbf{p}_{k-2} + \dots + F^n \mathbf{p}_{k-n} + W^0 w_k \quad (2.82)$$

which does not involve any physical time lag τ (considered constant) and models the state as the sequence of current and previous poses $s_k = (\mathbf{p}_k, \dots, \mathbf{p}_{k-n-1})$. AR models are used most frequently in tracking because they can approximate most real-world cases and can be *learned* efficiently with linear procedures [144] when only positional information is available from ground-truth trajectories.

In particular, a second-order AR process is given by

$$\begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k-1} \end{bmatrix} - \begin{bmatrix} \bar{\mathbf{p}} \\ \bar{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} F^1 & F^2 \\ I & 0 \end{bmatrix} \left(\begin{bmatrix} \mathbf{p}_{k-1} \\ \mathbf{p}_{k-2} \end{bmatrix} - \begin{bmatrix} \bar{\mathbf{p}} \\ \bar{\mathbf{p}} \end{bmatrix} \right) + \begin{bmatrix} W^0 \\ 0 \end{bmatrix} w_k \quad (2.83)$$

where (F^1, F^2, W^0, \bar{s}) define the model, which is again a Markov process,

$$s_k - \bar{s} = F(s_{k-1} - \bar{s}) + W w_k \quad (2.84)$$

in the state variables

$$s_k \equiv [\mathbf{p}_k \quad \mathbf{p}_{k-1}]^T \quad (2.85)$$

However, since the time lag τ is lost, we cannot cope with asynchronous measurements, which can be acquired at very irregular sampling times, which is particularly the case in visual processing. Therefore, at each time k we can convert the AR model into the real-time form (2.81) by incorporating the τ value inside $F_k(\tau)$ and $W_k(\tau)$ and again use the kinematic state $s_t = (\mathbf{p}_t, \dot{\mathbf{p}}_t)$.

In particular, for a first-order AR model, conversion to real time is trivial:

$$\begin{aligned} F_k &= F^{\text{AR}} \\ W_k &= \tau_k \cdot W^{\text{AR}} \end{aligned} \quad (2.86)$$

where we can also see how the overall process noise covariance Q increases quadratically with τ . In the second-order case, conversion formulas are given by

$$\begin{aligned} \mathcal{H}_k &= \begin{bmatrix} I & \tau_k \\ I & 0 \end{bmatrix} \\ F_k &= \mathcal{H}_k^{-1} F^{\text{AR}} \mathcal{H}_k \\ W_k &= \mathcal{H}_k^{-1} W^{\text{AR}} \end{aligned} \quad (2.87)$$

Figure 2.25 shows some examples generated by the AR models discussed below for one-dimensional pose. For these models, the state probability distribution is a Gaussian $\mathcal{N}(\hat{s}_k, \Sigma_k)$:

$$P(s_k) = \frac{1}{(2\pi)^{nD/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(s_k - \hat{s}_k)^T \Sigma_k^{-1} (s_k - \hat{s}_k)\right) \quad (2.88)$$

where nD is the state dimension, related to the order n and the pose dimension D ; the Gaussian propagates in time according to

$$\begin{aligned} \hat{s}_k - \bar{s} &= F(\hat{s}_{k-1} - \bar{s}) \\ \Sigma_k &= F \Sigma_{k-1} F^T + W W^T \end{aligned} \quad (2.89)$$

where the latter is also known as the *Riccati equation*. We can also see that \bar{s} is, for constrained motion, the *steady-state average*; in fact, if the process matrix F is stable in the limit $k \rightarrow \infty$, the distribution approaches $\mathcal{N}(\bar{s}, \Sigma_\infty)$, where Σ_∞ satisfies

$$\Sigma_\infty = F \Sigma_\infty F^T + W W^T \quad (2.90)$$

Later we provide some well-known examples of AR models.

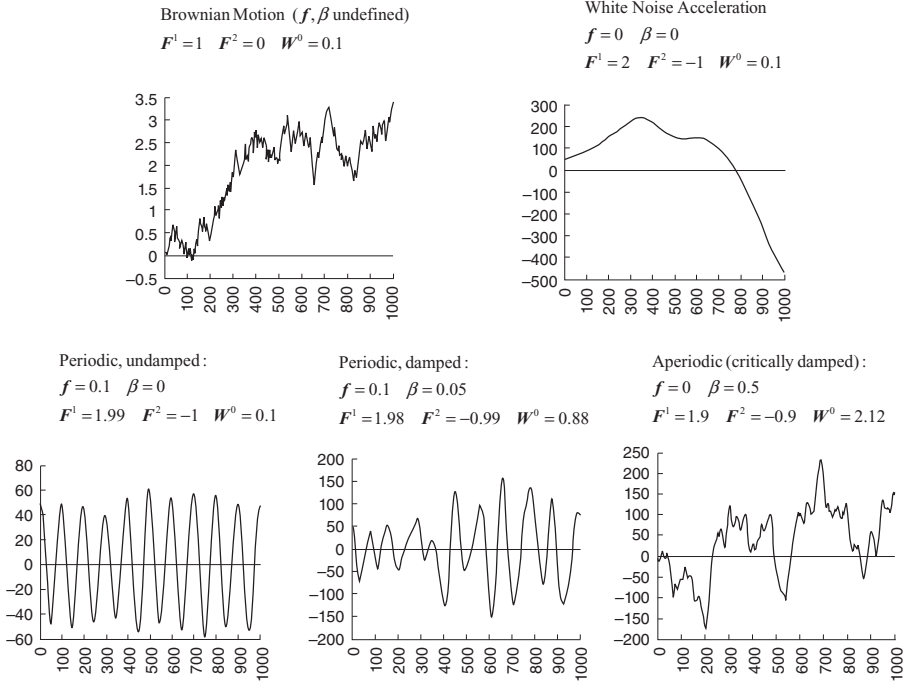


Figure 2.25 Examples of second-order AR models for a one-dimensional state.

2.4.1 Brownian Motion

Unconstrained Brownian motion, or *random walk*, is given by

$$F = F^1 \equiv I, \quad W \equiv W^0 \quad (2.91)$$

Here the state consists of position values only:

$$\mathbf{p}_k = \mathbf{p}_{k-1} + w_k \quad (2.92)$$

and the transition matrix is the identity. In this case we also notice that the average pose $\bar{\mathbf{p}}$ cancels out from eq. (2.83), and the process has no steady state, since although the mean value is constant $\bar{\mathbf{p}}_k = \mathbf{p}_0$, the covariance is unbounded:

$$\Sigma_k = \Sigma_{k-1} + WW^T \quad (2.93)$$

so that $\Sigma_\infty \rightarrow \infty$; it can also be shown that the mean-square radius of the covariance *ellipsoid* [which is given by $\text{tr}(\Sigma)$] grows proportional to the square root of time \sqrt{k} .

This model is less informative for tracking purposes, since it assumes that the velocity is completely random from time to time, which is almost never the case for objects with nonnegligible inertia. However, if the time lag between the measurements τ is very high compared to the object dynamics, any other model reduces to the Brownian model because the velocity will be almost unpredictable from time to time.

Constrained Brownian motion is obtained as follows:

$$F = F^1 \equiv aI; a \leq 1 \quad (2.94)$$

This model can be used to *constrain* the motion to a given bounding volume, where $a=1$ reduces to unconstrained motion. In particular, in the first time steps the behavior is almost the same as that for the unconstrained case, but after some time, the covariance approaches a steady state, given by

$$\Sigma_{\infty} = \frac{1}{1-a^2} WW^T \quad (2.95)$$

A comparison of the constrained and unconstrained cases is shown in Fig. 2.26 for a two-dimensional pose. Through eq. (2.95) one can compute the coefficients a and $W^0 = w_0 I$, yielding the desired *envelope* for the trajectory, as shown by Reynard et al. [144].

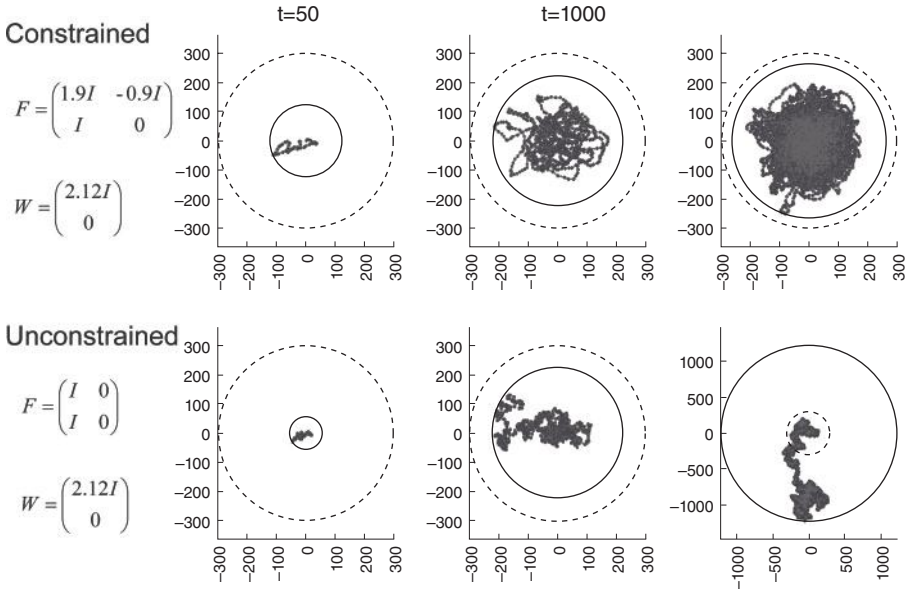


Figure 2.26 Comparison of constrained and unconstrained Brownian motion models.

2.4.2 Constant Velocity

The *constant velocity model*, also called *continuous white noise acceleration* (CWNA [33]) in the unconstrained form, is given by

$$F^1 = 2I, \quad F^2 = -I \quad (2.96)$$

This model reproduces the behavior of a body with nonnegligible mass, where the random acceleration term w represents an acceleration due to external forces. In terms of the real-time model [eq. (2.80)], it can be seen as the equivalent of

$$\mathbf{p}_k = \mathbf{p}_{k-1} + \dot{\mathbf{p}}_{k-1}\tau_k + \frac{1}{2}w_k\tau_k^2 \quad (2.97)$$

where the velocity $\dot{\mathbf{p}}_{k-1}$ is constant over the time interval, plus a random acceleration w_k .

The constrained counterpart is obtained in a manner similar to that for the Brownian case:

$$F^1 = f_1 I, \quad F^2 = f_2 I, \quad W^0 = w_0 I \quad (2.98)$$

where

$$\begin{aligned} f_1 &= 2 \exp(-\beta) \\ f_2 &= -\exp(-2\beta) \\ w_0 &= \rho \sqrt{1 - f_2^2 - f_1^2 - \frac{2f_2 f_1^2}{1 - f_2}} \end{aligned} \quad (2.99)$$

and β and ρ are two additional factors related to the size of the bounding volume. This reduces to the unconstrained model for $(\beta \rightarrow 0, \rho \rightarrow \infty)$.

2.4.3 Oscillatory Model

Next we consider the undamped oscillation model, which reproduces a mass-spring system perturbed by random accelerations (i.e., external forces). The matrices are the same as those defined in eq. (2.98), but with scalar coefficients given by

$$\begin{aligned} f_1 &= 2 \cos 2\pi\nu_0 \\ f_2 &= -1 \\ w_0 &= 1 \end{aligned} \quad (2.100)$$

where ν_0 is the timeless oscillation *frequency* (i.e., given in terms of the integer index k).

We can add a *damping* coefficient,

$$\begin{aligned} f_1 &= 2 \exp(-\beta) \cos 2\pi v_0 \\ f_2 &= -1 \exp(-2\beta) \\ w_0 &= \rho \sqrt{1 - f_2^2 - f_1^2 - \frac{2f_2 f_1^2}{1 - f_2}} \end{aligned} \quad (2.101)$$

where β is the damping factor, reproducing the dynamics of a *mass-spring system* with viscous friction. The additional term ρ is used to regulate the influence of process noise.

2.4.4 State Updating Rules

As explained in Section 2.3, to avoid representation singularities during tracking, we distinguish between *absolute* and *incremental* pose parameters, where the latter are predicted and estimated during tracking and used to update the former. Therefore, we need to keep this distinction while defining the kinematic state, which in discrete time consists of:

- Absolute pose and temporal derivatives: $s_k = (\mathbf{p}_k, \dot{\mathbf{p}}_k, \ddot{\mathbf{p}}_k, \dots)$
- Incremental pose and derivatives⁷: $\delta s_k = (\delta \mathbf{p}_k, \dot{\delta \mathbf{p}}_k, \ddot{\delta \mathbf{p}}_k, \dots)$

and we call δs_k the *incremental state*.

If the updating rule is additive, $(\mathbf{p}, \delta \mathbf{p}) \rightarrow (\mathbf{p} + \delta \mathbf{p}, 0)$, we can rewrite the dynamics in terms of δs_k :

$$\delta s_k = (F_k - I)(s_{k-1} - \bar{s}) + W_k w_k \quad (2.102)$$

and afterward update the state with

$$\begin{aligned} s_k &\rightarrow s_k + \delta s_k \\ \delta s_k &\rightarrow 0 \end{aligned} \quad (2.103)$$

In the compositional case, however, the incremental state cannot be substituted into linear dynamics [eq. (2.81)], since compositional updates are non-linear in \mathbf{p} . Therefore, we can at least *approximate* the model with

$$\delta s_k \approx F_k \delta s_{k-1} + W_k w_k \quad (2.104)$$

⁷Notice that, in general, $\dot{\delta \mathbf{p}}$ refers to the temporal derivative of the incremental pose $d/dt(\delta \mathbf{p})$, not to a finite increment of velocity $\delta(\dot{\mathbf{p}})$. This distinction is relevant for the case of compositional updates.

where no average absolute state \bar{s} is included, limiting the application to unconstrained models only. The updating rule can be applied to the positional part:

$$\begin{aligned} T(\mathbf{p}_k) &\rightarrow T(\mathbf{p}_k) \cdot \delta T(\delta \mathbf{p}_k) \\ \delta \mathbf{p}_k &\rightarrow 0 \end{aligned} \quad (2.105)$$

while temporal derivatives $\dot{\delta \mathbf{p}}_k, \ddot{\delta \mathbf{p}}_k, \dots$ are left unchanged by the update and the absolute kinematic quantities $\dot{\mathbf{p}}_k, \ddot{\mathbf{p}}_k, \dots$ are not estimated explicitly.⁸

As we will see in Chapter 5, the explicit form of dynamics is used by most Bayesian filters: for example, a Kalman filter [163], as well as extended and unscented Kalman filters [87], obtains a single deterministic state *prediction* by setting $w_k = 0$ in eq. (2.102) or eq. (2.104). In particular, for nonlinear models the Jacobian transition matrix

$$F_k \equiv \frac{\partial f}{\partial s}(s_k, w_k = 0, \tau_k) \quad (2.106)$$

may be used to linearize the model as is done, for example, by the extended Kalman filter (Section 5.3.2). For Monte Carlo filters (Section 5.4), the process noise w_k is simulated explicitly, by means of random sampling.

The implicit form $P(s_k | s_{k-1})$, in most cases a Gaussian, makes it possible to evaluate the *predictive prior* for a given state hypothesis s_k , and it can be used for some *importance sampling* schemes [85] as well as by the MCMC filter [92] of Section 5.4.4.

Our strategy for modeling object dynamics thus consists of:

1. Choosing a proper AR model with parameters related to our application, which can be computed or estimated via a learning method from offline sequences, without considering the time lag
2. Measuring each lag τ_k for the current prediction and adapting the AR model to the real-time version involving the kinematic state
3. Applying the predictive model to the incremental state, and afterward updating the state, according to the update rule selected for the pose, additive or compositional

We finally notice that in this framework, the dynamic model must be provided externally. However, when no detailed information regarding the object dynamics is available, a default Brownian model is sufficient for most purposes, where the only parameter required is the process noise covariance, modeled as a white Gaussian uncertainty w_k around the current pose.

⁸In this case one can, instead, compute \dot{T}_k from knowledge of T_k and $\delta \dot{T}_k$, in turn given by $\dot{\delta \mathbf{p}}_k$, and so on.

2.4.5 Learning AR Models

The autoregressive models so far described require the specification of several parameters, such as oscillation frequency, damping coefficient, and size of the constraining envelope. For complex geometric transforms, these variables may differ for each pose parameter and also exhibit some coupling; therefore, an automatic procedure for *learning* the object dynamics is highly desirable.

Such a procedure takes the output of a pretracked input sequence, where the object is exhibiting roughly the same dynamics as those of the test cases, and where pose values \mathbf{p}_k , $k = 1, \dots, K$ are used as ground-truth data,⁹ to fit the best AR model of a given order. In particular, in the second-order case we can start from eq. (2.84), which is repeated here:

$$s_k - \bar{s} = F(s_{k-1} - \bar{s}) + Ww_k \quad (2.107)$$

where $s_k = (\mathbf{p}_k, \mathbf{p}_{k-1})^T$, $\bar{s} = (\bar{\mathbf{p}}, \bar{\mathbf{p}})^T$, and the quantities F^1 , F^2 , W^0 , and \bar{s} are to be estimated. This equation can also be written in the form

$$s_k = F s_{k-1} + D + W w_k \quad (2.108)$$

where the relationship between the two representations is expressed by

$$(I - F^1 - F^2)\bar{\mathbf{p}} = D \quad (2.109)$$

The learning problem can be formulated as a maximum-likelihood estimation (MLE), where the log likelihood of the sequence \mathbf{p}_k observed is maximized over the dynamic parameters. For this purpose, in the following we outline the approach described in more detail by Blake and Isard [43, Chap. 11].

Starting from the one-dimensional case, if we assume temporarily that the mean value is $\bar{\mathbf{p}} = 0$, then the scalar quantity $p_k - F^1 p_{k-1} - F^2 p_{k-2}$ is a Gaussian $\mathcal{N}(0, (W^0)^2)$. Since w_k are also independent, the likelihood of the sequence observed is expressed by

$$P(p_0, \dots, p_K | F^1, F^2, W^0) \propto \prod_{k=3}^K \exp \left[-\frac{1}{2} \left(\frac{p_k - F^1 p_{k-1} - F^2 p_{k-2}}{W^0} \right)^2 \right] \quad (2.110)$$

and the negative log-likelihood, up to an additive constant, is

$$-\log P = \frac{1}{2(W^0)^2} \sum_{k=3}^K (p_k - F^1 p_{k-1} - F^2 p_{k-2})^2 + (K-2) \log W^0 \quad (2.111)$$

⁹Notice that pose *measurements* \mathbf{p}_k are assumed noise-free, whereas the only random variable here is the *process* noise.

which is quadratic in F^1 and F^2 for any value of W^0 . The MLE is then obtained by first minimizing the sum of squares with respect to F^1 and F^2 , and subsequently minimizing eq. (2.111) over W^0 .

This is done more efficiently by first computing the *autocorrelation* coefficients r_{ij} , defined by

$$r_{ij} \equiv \sum_{k=3}^K p_{k-i} p_{k-j} \quad (2.112)$$

for $i, j = 0, 1, 2$. Then it can be shown that the MLE parameters \hat{F}^1 and \hat{F}^2 are found by solving the system

$$\begin{aligned} r_{02} - \hat{F}^2 r_{22} - \hat{F}^1 r_{12} &= 0 \\ r_{01} - \hat{F}^2 r_{21} - \hat{F}^1 r_{11} &= 0 \end{aligned} \quad (2.113)$$

following which \hat{W}^0 is given by

$$\hat{W}^0 = \sqrt{\frac{1}{K-2} (r_{00} - \hat{F}^2 r_{20} - \hat{F}^1 r_{10})} \quad (2.114)$$

For the multidimensional case, if we include the mean value \bar{s} in the estimation process, we can write the log likelihood of the ground-truth sequence in terms of the matrices F^1, F^2, W^0 , and D :

$$\begin{aligned} -\log P(\mathbf{p}_1, \dots, \mathbf{p}_K | F^1, F^2, W^0, D) \\ = \frac{1}{2} \sum_{k=3}^K \left| (W^0)^{-1} (\mathbf{p}_k - F^1 \mathbf{p}_{k-1} - F^2 \mathbf{p}_{k-2} - D) \right|^2 - (K-2) \log \det W^0 \end{aligned} \quad (2.115)$$

which is quadratic in F^1, F^2 , and D .

The solution, similar to the unidimensional case, is obtained by defining the autocorrelation matrices

$$R_i \equiv \sum_{k=3}^K \mathbf{p}_{k-i} \mathbf{p}_{k-i}^T, \quad R_{ij} \equiv \sum_{k=3}^K \mathbf{p}_{k-i} \mathbf{p}_{k-j}^T, \quad R'_{ij} \equiv R_{ij} - \frac{1}{K-2} R_i R_j^T \quad (2.116)$$

and subsequently estimating the AR matrices:

$$\begin{aligned} \hat{F}^2 &= (R'_{02} - R'_{01} R'^{-1}_{11} R'_{12}) (R'_{22} - R'_{21} R'^{-1}_{11} R'_{12})^{-1} \\ \hat{F}^1 &= (R'_{01} - \hat{F}^2 R'_{21}) R'^{-1}_{11} \\ \hat{D} &= \frac{1}{K-2} (R_0 - \hat{F}^2 R_2 - \hat{F}^1 R_1) \end{aligned} \quad (2.117)$$

as well as the mean vector (if required)

$$\hat{\mathbf{p}} = (I - \hat{A}^2 - \hat{A}^1)^{-1} \hat{D} \quad (2.118)$$

and finally, the process noise gain W^0 is the matrix square root of

$$\hat{Q}^0 = \frac{1}{K-2} (R_{00} - \hat{A}^2 R_{20} - \hat{A}^1 R_{10} - \hat{D} R_0^T) \quad (2.119)$$

where $\hat{Q}^0 = \hat{W}^0 \hat{W}^{0T}$.

CHAPTER 3

THE VISUAL MODALITY ABSTRACTION

In this chapter we present the main “backbone” of the OpenTL framework: a common abstraction for all visual modalities, allowing seamless data fusion at different levels, in order to build an arbitrary measurement *processing tree* for the tracking pipeline. Here static and dynamic data association and fusion take place as well as mutual occlusion handling for simultaneous target hypotheses (*ensemble states*), when depth information is available. For such a complex task, efficient, feature sampling, as well as matching procedures, can be implemented on the GPU.

By following the pipeline flow of Fig. 1.5, we show that any visual modality can be derived from a common abstraction, consisting of the five steps shown in Fig. 3.1. As described in Section 3.3, data association can also be differentiated according to the type of the resulting output: pixel, feature, or object level. This abstraction encompasses a wide variety of cues and methods and provides a high degree of freedom for choosing new combinations, limited only by the imagination of the developer.

3.1 PREPROCESSING

The preprocessing step consists of operations related to the particular modality, not involving a *target* hypothesis, which means that it should depend neither

- *Preprocessing.* This involves target-independent image processing, providing unassociated data. Depending on the modality, these data may be at the pixel or feature level.
- *Sample off-line features.* Select good features visible from each camera under a *prior* ensemble state hypothesis s_k^- (usually, the average \bar{s}_k^-), from the off-line shape and appearance model.
- *Matching (pixel/feature/object) level.* Associate model data (both off-line and online) with preprocessed data under a predicted state hypothesis; the output consists of target-related *measurements* at the level desired (Section 3.3). In this step, static data fusion may also take place.
- *Likelihood.* For each target, evaluate the likelihood of the output measurement.
- *Update online features.* Collect image features under a *posterior* ensemble state hypothesis s_k (usually, the average \bar{s}_k), and back-project them in object space.

Figure 3.1 Main abstraction for visual modality processing.

on object models nor on state values. More precisely, this requirement also concerns an *indirect* dependency: If two visual modalities are connected in cascade, and the second modality performs target-independent processing onto data which, however, have been obtained from the previous modality after model matching, this operation has an indirect dependence on targets, and therefore it can no longer be considered preprocessing.

For example, a blob tracker requires detecting connected components from a presegmented binary image, which may have been obtained with model-based pixel-level matching, such as pixels from a blue target model. Therefore, in this case, blob detection can no longer be considered preprocessing. If, instead, a model-free color or motion segmentation has produced the binary image, that was a preprocessing step, and blob detection can also be considered preprocessing.

Generally speaking, the output of preprocessing can be pixel-level data, such as pixel-wise color space conversion, as well as a set of detected features such as edges or keypoints. Subsequently, these data have to be matched with the target model at the pixel or feature level. In particular, if preprocessing produces features, only feature- or object-level matching can be performed, whereas pixel-level output can be matched with the target at any level. This corresponds to the principle of nonincreasing information in the matching process, which can only operate a *synthesis* of the given data to a higher abstraction level, but obviously cannot create new data by “downgrading” the level.

The use of graphics hardware for general-purpose computing, also called GP-GPU, can largely accelerate preprocessing by implementing these procedures on the *fragment shaders* of the GPU-rendering pipeline, which are parallel computing units acting on local regions of the output frame buffer.

3.2 SAMPLING AND UPDATING REFERENCE FEATURES

Reference features can either be sampled from the base shape and appearance model (*off-line* features) or from image data (*online* features), and subsequently *back-projected* onto the model surface so that they can be reprojected at different hypotheses and used for matching and pose estimation.

The two sampling procedures take place at different stages of the pipeline.

- Off-line features are selected from the shape and appearance model *before* matching, that is, under a *prior* ensemble state hypothesis \mathbf{s}_k^- , assuming that predicted states are close to the current view. In most cases, this involves *rendering* the model into an off-screen image, detecting features, and back-projecting them in object space, eventually using the depth buffer.
- Online features are, instead, updated from the current image *after* state update, that is, under a *posterior* state \mathbf{s}_k , because posterior states are matched to the image data, in the ideal case giving the correct back-projection, in the absence of estimation errors.

Depending on the modality and on the visibility conditions of the shape, these procedures can sometimes be performed once before tracking (e.g., for planar objects), whereas in other cases they must be repeated more or less often for different views, in the worst case at every frame. In any case, being expensive procedures, they are usually performed for a *single* ensemble state, usually coincident with the *average* of the distribution $\bar{\mathbf{s}}_k^-$ (respectively, $\bar{\mathbf{s}}_k$).

Combining both types of features provides more stability and robustness [157]: in fact, off-line features are stable but cannot adapt to the environmental conditions, such as variable lighting or external occlusions, whereas online features are always up-to-date but may accumulate errors during the back-projection because of state estimation errors. Therefore, the former may give *jittering* pose estimates, while the latter may suffer from *drift* phenomena. However, fusion of off-and online features has a synergic effect, which improves tracking quality; in fact, off-line features prevent drift, in turn allowing safe back-projection of the online features, contributing overall to more precise and robust results.

To provide a few examples: Foreground color statistics can be collected and updated from the last estimation result, as well as sampled from the appearance model rendered. A background model for segmentation can also be updated during time and kept along with the initial model, which acts as a constant reference for regions that have erroneously been updated with moving objects. The same holds for shape statistics used for foreground blob matching, where the current blob shape can be refined online, while the off-line model may provide a coarse but constant version (e.g., a simple rectangular shape for representing a person).

Concerning the detection procedure, every visual primitive (i.e., edge points, contour segments, local keypoints, templates, histograms, etc.) can be encoded in a more-or-less complex *descriptor*, which corresponds to specific geometric and/or photometric attributes.

- *Geometric descriptor.* The location, size, and shape of the feature can be defined in both image and object space, where the correspondence for single points is given by the warp function of Section 2.3.
- *Photometric descriptor.* This includes appearance-related attributes such as the gray-level pattern of a keypoint, or a multiresolution template, color histograms, and so on. They can be sampled from the appearance model and eventually modified by the appearance parameters.

We identify and provide in the lower layer of our architecture specific data structures for visual primitives, which can be common to more processing modalities. For example, different contour-matching algorithms, such as the CCD algorithm [135] or edge-based trackers [61, 84], require sampling visible contour points and related screen normals, while computing different likelihood functions. Also in this case, the GPU can provide very efficient sampling for different modalities, under a given ensemble state and camera view, assisted by the *z-buffer* rendering facilities. At the same, it makes it possible to handle self-occlusions for nonconvex and articulated objects, as well as mutual occlusions between targets.

For this purpose, scene rendering can be run off-screen onto a *texture memory*, producing silhouette lines, full color or texture images, or feature points obtained with more complex operations, such as silhouette sampling [148]. An example of the latter is shown in Fig. 3.2, where visible object silhouettes are sampled uniformly in image space, with a constant number of points independent on the viewing angle or distance, also taking care of mutual occlusions.

Moreover, subsequent data association can also be implemented on the GPU, thus saving much of the data *back-transfer* overhead, which usually constitutes a severe bottleneck for the system; for example, a scalar likelihood value could be returned directly for each state hypothesis instead of the full vector of matching residuals. For both sampling and visualization purposes, multiple rendering *contexts* must be instantiated, taking into account each camera position and internal parameters, so that the entire scene can be rendered separately from each view.

All of these functionalities are implemented in OpenTL using the OpenGL Shader Language (GLSL [147]), providing a general-purpose and hardware-independent code, with the exception of a few extensions, such as the *occlusion query* [18] and *integer texture* [16], which are vendor independent, however, and available as part of the OpenGL 2.1.1 standard.

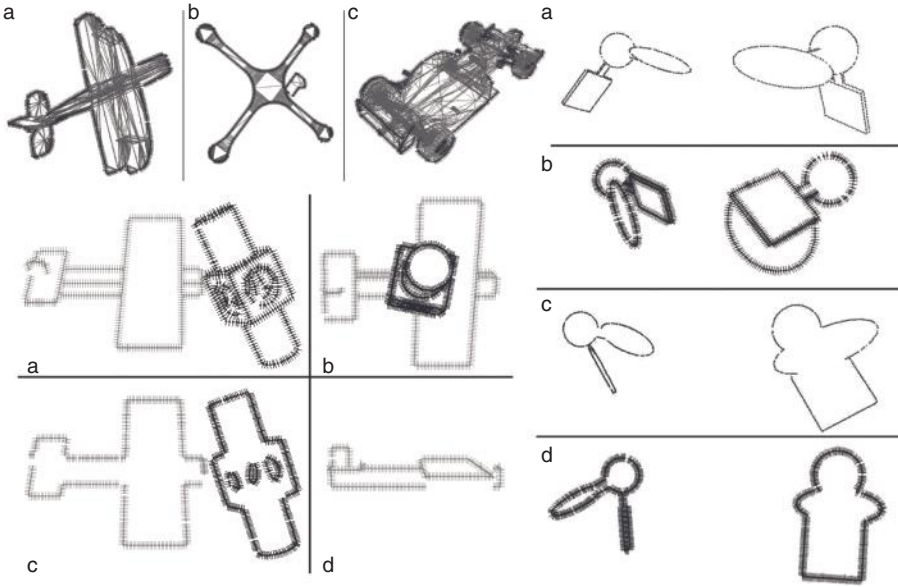


Figure 3.2 GPU-assisted visibility testing, with contour points and normals sampled from CAD models. (From [148].)

3.3 MODEL MATCHING WITH THE IMAGE DATA

The next processing step involves projecting model features in image space through the warp function at predicted state s^- and performing an association (or *matching*) with the image data. *Data association* [34] in a general context consists of formulating one or more hypotheses about which object, or object part, generated the data observed, eventually including missing detections (object parts without associated data) and false alarms, originating from background clutter. The result of a data association process is a set of *measurements* that we define as a 5-tuple:

$$Z \equiv \langle z, h, H, e, R \rangle_X \quad (3.1)$$

where X is the *level* of the output (pixel, feature, or object level), (z, h) are matched image–model pairs, possibly with multiple hypotheses, e the respective residuals or *innovations*, R the *measurement noise* covariances, and H the expected measurement Jacobians:

$$H \equiv \frac{\partial h}{\partial s} \quad (3.2)$$

The H matrices can be required by the tracker (e.g., by the EKF) as well as for maximum-likelihood pose estimation. Computing H requires the screen point derivatives from the warp function: for example, given by the respective Lie algebras, in terms of the incremental pose δp , as described in Section 2.3.

Dropping the object and modality indices, two equivalent forms for the measurement model may be required by a given Bayesian tracker:

- Explicit form: $z_k = h(s_k, e_k, v_k)$
- Implicit form (likelihood): $P(z_k | s_k)$

In the first form, v_k is the measurement noise, with covariance R_k , usually diagonal or block diagonal, that can be provided externally or updated online, depending on the modality. The second form is the *likelihood* of the observation z given s , and alone can be used for maximum-likelihood parameter estimation, such as LSE optimization for Gaussian models (Section 5.2).

Often, in computer vision applications a *multihypothesis* association is a more realistic and robust model, in the presence of clutter background and missing detections, which implies a non-Gaussian likelihood with multiple *modes*, such as a Gaussian mixture. Moreover, in analogy with the data fusion terminology [71], Sec. 4.4, we also classify our measurements into three levels (Fig. 3.3):

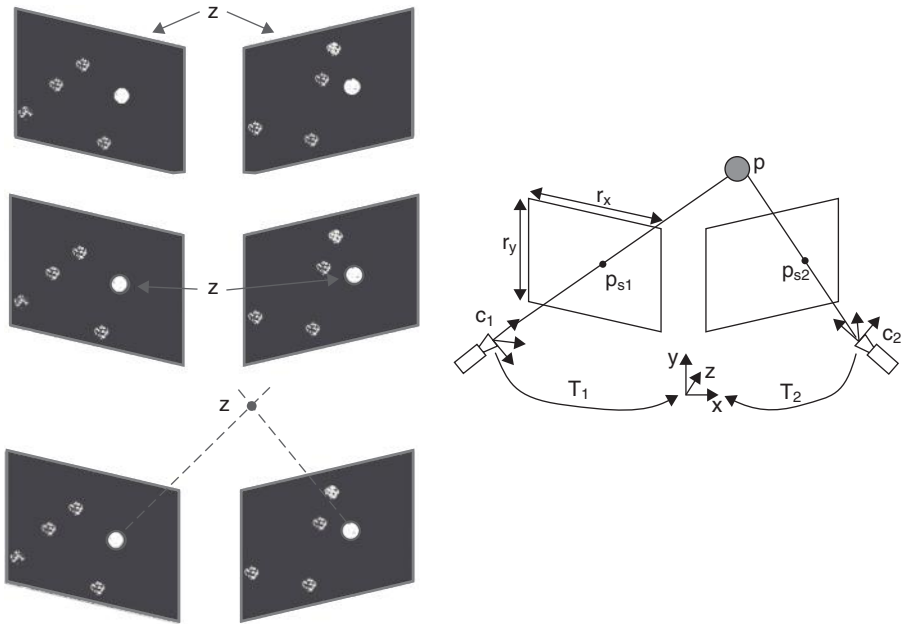


Figure 3.3 Three possible visual processing levels for the same task.

1. *Pixel level.* Z consists of pixel-wise data such as binary maps or vector fields, where (z, h) are observed and expected maps, e is an image or a global, scalar residual, and R is the covariance of e , either pixel-wise or scalar. If e is an image, an *image Jacobian* $H_d = \partial h / \partial s_d$ may be defined for each state variable d .
2. *Feature level.* Z is a set of associated model and image features. Here h is a vector containing the expected *descriptors* h_i and z a corresponding set of image features that may be associated on a single-hypothesis basis z_i or multihypothesis basis $z_{ij}, j = 1, \dots, J$, also including the case of missing detection $J = 0$, where h_i is associated with an empty set. H is a set of matrices with the derivatives of each element in h_i with respect to the state. Residuals may be given by vectors $e_{ij} = z_{ij} - h_i$, with the respective covariance matrices R_i , or by a global distance $e = d(z, h)$ in some metric space, with scalar covariance.
3. *Object level.* (h, z) are, respectively, predicted and estimated states (s^-, s^*) or, more commonly, just poses ($\mathbf{p}^-, \mathbf{p}^*$); therefore, $H = I$ or $H = [I \ 0]$, while the residual is simply the difference $e = z - h$, and the measurement covariance R is defined for each state parameter. An object-level measurement is usually obtained from a local, maximum-likelihood estimation procedure, starting from the predicted state s^- , where the likelihood may be at either the feature or pixel level, or a combination (as described in Section 3.4).

In the example of Fig. 3.1, matching at the pixel level involves comparing the projected *object shadow* h at pose s^- with the segmented camera images z ; at the feature level, a reference blob h is warped and matched to the nearest-neighbor blob z in the respective image; at the object level, the three-dimensional pose $z = \mathbf{p}^*$ can be estimated by means of *triangulation* from the associated blobs, after feature-level matching, and compared with the predicted pose $h = \mathbf{p}^-$.

A nice analogy can be drawn between two basic approaches for the analysis of fluid mechanics (see, e.g., [35]) and the first two measurement levels. Referring to Fig. 3.4, a pixel-level measurement corresponds to the *Eulerian*

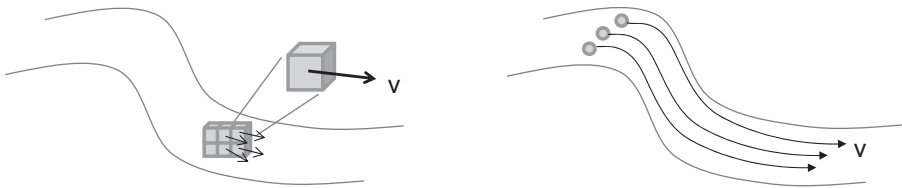


Figure 3.4 Analogy with two approaches in fluid mechanics. *Left:* Eulerian approach: local velocities are observed through a small volume, equivalent to a pixel-level measurement. *Right:* Lagrangian approach: tracking massless particles for computing the flow lines is equivalent to feature-level measurements.

approach, where fluid analysis is carried through small observation volumes (or *voxels*) with a fixed location in space and concerns local properties such as velocity, pressure, and temperature. Instead, a feature-level measurement corresponds to the *Lagrangian approach*, where massless, small moving particles are tracked into the fluid in order to compute the flow trajectories. In the latter, features may be lost and resampled from time to time, according to the visibility conditions of the surface from each viewpoint. As explained in the following, the choice of output level is related to the data fusion methodology as well as the likelihood function desired.

3.3.1 Pixel-Level Measurements

When the measurement is obtained pixel-wise, h and z are, respectively, the pixel maps expected and observed. We mention here a few examples. A foreground segmentation (Fig. 3.5) provides a binary map that can be matched to the object *shadow* expected, representing an ideal segmentation without model errors or background clutter. An optical flow field [80] can also be matched with the flow expected, obtained at a given object pose *and* velocity. Finally, when a fully textured model is available, the surface can be rendered directly and matched against the underlying color pixels.

Pixel-level measurements have the advantage of being more informative and generally more precise than feature-level measurements; on the other hand, they involve expensive and nonlinear computation, which, however, can be accelerated greatly by graphics hardware, using, for example, the OpenGL shader language [147].

In the case of binary images, we consider each pixel \mathbf{y} as a measurement location with only two possible responses $z(\mathbf{y}) = \{0, 1\}$. Therefore, we can define the following cases (Fig. 3.6, left):

- False alarm: $h(\mathbf{y}) = 0, z(\mathbf{y}) \neq 0$, occurring with probability λ
- Missing detection: $h(\mathbf{y}) \neq 0, z(\mathbf{y}) = 0$, occurring with probability α
- Correct match: $h(\mathbf{y}) = z(\mathbf{y})$, occurring with probability $1 - \alpha - \lambda$



Figure 3.5 Example of pixel-level measurement. *Left*: expected object shadow; *middle*: foreground segmentation on the real image; *right*: residual image. (From [100].)

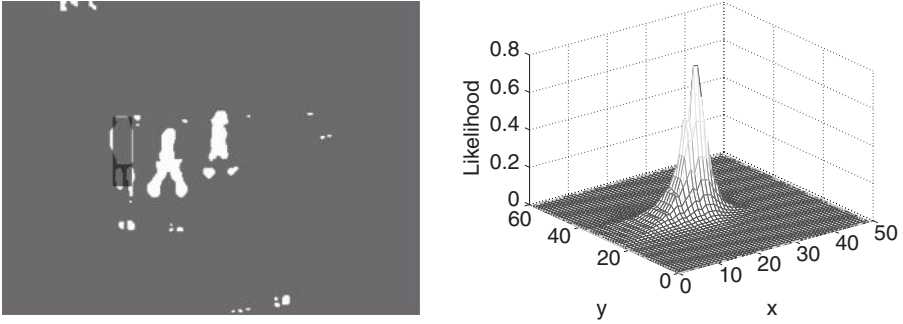


Figure 3.6 Pixel-level likelihood. *Left:* residual image for a given pose hypothesis; *right:* plot of the likelihood for different values of horizontal and vertical translation.

With these values we may define a scale-normalized likelihood [136]:

$$P(z|s) = \frac{1}{K} \cdot \alpha^{|\text{MD}|/N_{\text{obj}}} \lambda^{|\text{FA}|/N_{\text{obj}}} (1 - \alpha - \lambda)^{(N - |\text{MD}| - |\text{FA}|)/N_{\text{obj}}} \quad (3.3)$$

$$K = \lambda^{(\max|\text{FA}| - N_{\text{obj}})/N_{\text{obj}}} (1 - \alpha - \lambda)^{(N - \max|\text{FA}| + N_{\text{obj}})/N_{\text{obj}}}$$

In this formula, N is the image size, $\text{MD}(s)$ the subset of undetected pixels, and $\text{FA}(s)$ the number of false alarms observed under the state hypothesis s . The term N_{obj} is the object size (i.e., the number of pixels under the shadow h expected, which provides scale normalization, in conjunction with an additional factor K , by modifying the covariance of the three terms with respect to the object displacement; in fact, an object with half the size will exhibit a half variation in both the MD and FA error terms.

While modeling a pixel-level likelihood, other issues may arise because of the spatial closeness of neighboring pixels, which invalidate the assumption of mutual independence between measurements. However, in most cases, neglecting this effect does not degrade the tracking performance, while keeping the likelihood function well shaped and peaking around the correct pose (Fig. 3.6, right).

Furthermore, in the case of pixel-level measurements, the two parameters λ and α can be estimated and updated over time, thus providing more robustness and adaptivity to challenging conditions such as a variable clutter density or partial occlusions. For this purpose, a simple adaptation scheme can be used: Under the current pose estimate, the missing detection α and false alarm λ rates are estimated by the two regions defined by the object shadow, foreground and background:

$$\begin{aligned}\alpha &= \frac{\text{no. missing pixels in the foreground region}}{\text{area of foreground region}} \\ \lambda &= \frac{\text{no. detected pixels in the background region}}{\text{area of background region}}\end{aligned}\quad (3.4)$$

Other examples of pixel-level measurements, involving scalar or vector fields, are dense optical flow maps, silhouette matching with distance transform, and motion history images; these modalities are presented in Chapter 4.

3.3.2 Feature-Level Measurements

A feature-level measurement is obtained by defining one or more *cues* of a given type, such as contour points, local keypoints, and color statistics, and are usually supposed to be statistically independent. A basic distinction here concerns local vs. global features:

- *Local features* cover small areas such as surface patches or line segments, and they are detected individually and matched to the corresponding model features. Examples include keypoints, contour points, and lines.
- *Global features* describe the *entire* visible surface of an object, or a large part of it, such as an entire link of an articulated structure, matched to the image data in the underlying region. Examples include color statistics, histograms of oriented gradients, template maps, or global shape descriptors such as spatial blob *moments*.

Each local feature may provide one or more association hypotheses (h_i, z_{ij}) , $j = 0, \dots, J_i$, where h_i is the expected value and z_{ij} are the associated data in a variable number J_i , possibly also $J_i = 0$. This results in a set of residuals e_{ij} and measurement covariances R_i , assumed to be the same for all hypotheses:

$$(e_{ij}, R_i), \quad j = 1, \dots, J_i \quad (3.5)$$

The corresponding likelihood typically has the form of a product of Gaussian mixtures, plus uniform clutter noise [84]:

$$P(z|s) \propto \prod_{i=1}^N \left[\alpha + \frac{1-\alpha}{\lambda J_i (2\pi)^{D/2} \sqrt{|R_i|}} \sum_{j=1}^{J_i} \exp\left(-\frac{1}{2} e_{ij}^T R_i^{-1} e_{ij}\right) \right] \quad (3.6)$$

with D the dimension of the feature descriptor, α the *missing detection probability* $P(J=0)$, and λ the *false alarm density* (i.e., the average rate of clutter-

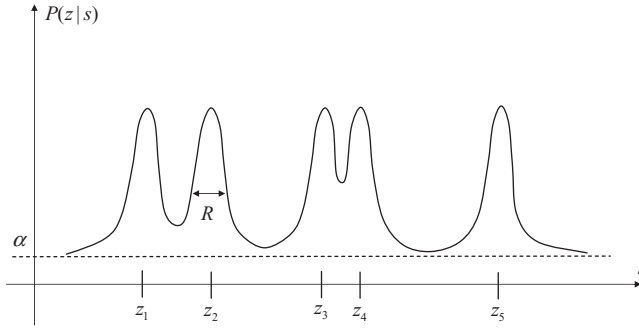


Figure 3.7 Multihypothesis association likelihood (monodimensional case).

originated features). An example for a monodimensional measurement with $h(s) = s$ is given in Fig. 3.7.

This model is usually employed by Monte Carlo filters, which can also deal with non-Gaussian densities. The purely Gaussian likelihood can be obtained as a special case of Eq. (3.6) by imposing $\alpha = 0$ and $J_i = 1$ for all i , typically with a *nearest-neighbor* approach:

$$e_i = e_{ij^*} : j^* = \arg \min_j \|e_{ij}\| \quad (3.7)$$

To select the associated hypotheses, it would be too complex to consider all possible combinations for all model features. However, during tracking we have the prior-state distribution, and we can therefore easily rule out “impossible” associations, which are too far away from the predicted location and have almost zero probability of occurrence.

This leads to the concept of *validation gate* [34], a region in *measurement* space in the neighborhood of the predicted measurement, containing the only data that can be associated with this target. Validation gates can be fixed in advance by giving a maximum search distance, or adapted during tracking. In the latter case, by assuming Gaussian models for both process and measurement noise, at time k we can linearize the measurement model through the Jacobian matrices $H_{k,i}$ and compute the *innovation covariance* for each feature i :

$$S_{k,i} = H_{k,i} \Sigma_k^- H_{k,i}^T + R_{k,i} \quad (3.8)$$

which is the covariance matrix of the innovation $e_{k,i} = z_{k,i} - h_{k,i}(s_k^-)$, taking into account both the uncertainty of $z_{k,i}$ given by $R_{k,i}$ and the state prediction given by Σ_k^- .

According to $S_{k,i}$, any data z_k having a Mahalanobis distance from the expectation $h_{k,i}(s_k^-)$ higher than, say, 3.0 has a very low probability of being

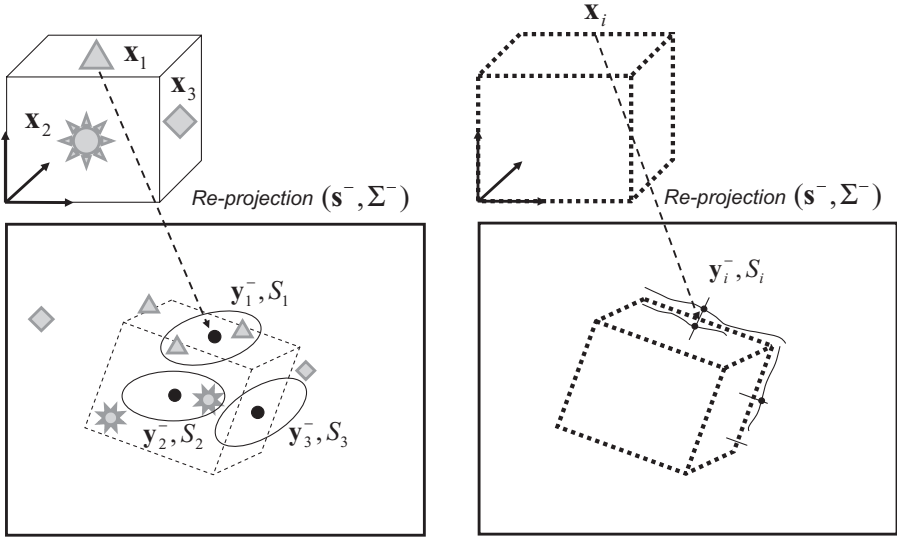


Figure 3.8 Validation gates for re-projected features according to the innovation covariance S_i . *Left*: bidimensional gates for local keypoints; *right*: monodimensional gates for contour points, matched along the normals.

associated with it. The validation gate is then given by an ellipsoid representing the volume containing most of its probability density.¹

Figure 3.8 provides an example of validation gates computed for individual features being matched by re-projection from the prior distribution $(\bar{s}_k, \bar{\Sigma}_k)$. On the left, we show the two-dimensional validation gate for local keypoints (Section 4.5), while on the right a monodimensional gate is shown, matched only along the normals (Section 4.4.1). We can observe how the number of measurements z falling into the validation gate of each re-projected feature is not constant, and the gate can also be empty.

In the context of feature association, we finally make an important distinction between matching by *re-projection* or by *tracking*. Referring to Fig. 3.9, after sampling model features, in the subsequent frame we can associate image data in two ways:

1. By *re-projection*. Model features are re-projected while looking for the nearest neighbor to the location predicted.
2. By *tracking*. Features from the previous frame are matched directly to the current frame without using the model.

¹In the case of particle filters, a single-state hypothesis is considered as a *Dirac* with zero covariance. Therefore, one should use $R_{k,i}$ instead of $S_{k,i}$, while the apparently smaller search range is compensated for by the fact that many particles s_n are distributed around \bar{s} with the prior covariance.

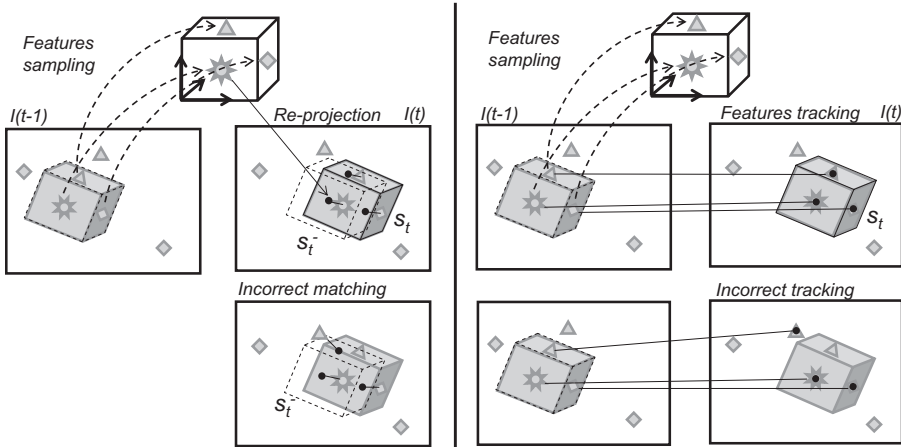


Figure 3.9 Left: feature matching by re-projection; right: feature tracking (corresponding to the *optical flow* modality).

As we can see, re-projection is a *model-driven association* that depends on the particular state hypothesis s_t^- and may fail if false alarms occur in the neighborhood of the re-projected location. On the other hand, feature tracking is purely *data driven* and can fail if local tracking is incorrect; according to our abstraction, feature tracking can be considered to be a *preprocessing* step.

We also notice that the latter approach corresponds to a sparse *optical flow*; in fact, since image-to-image correspondences are established, for each feature an additional estimate of its *velocity* in image space is provided which can be put into correspondence with the projected object velocities at the given state. In most of the examples of Chapter 4, we will see feature-matching procedures by re-projection, apart from the optical flow modality of Section 4.6.2. However, notable examples of feature tracking are also known in the literature, such as the *moving edges approach* [44] and any *blob tracking algorithm*, such as the *level sets method* (Section 4.3.2).

3.3.3 Object-Level Measurements

An object-level measurement is a direct estimate of the object state. This type of measurement amounts to a localization of the object near the pose predicted, and generally is obtained with a maximum-likelihood estimation (MLE), where the likelihood may be a lower-level one (pixel or feature) as well as any combination of multimodal and multisensor likelihoods.

As can easily be understood, this is in general a nonlinear optimization problem, for which a recursive algorithm must be used which may produce multiple local maxima (i.e., a multihypothesis measurement) around the pose predicted, which is taken as the “initial guess,” $\mathbf{p}^0 = \mathbf{p}_t^-$ for the local search. This

problem will be considered in more detail in the context of each modality, of static data fusion, of state-space estimation (Section 5.2), and in Appendix A. In this section we note only how this type of procedure may be used at any level of the processing tree, so that intermediate object-level measurements can be carried out before the Bayesian state update, which takes place at the end of the pipeline.

3.3.4 Handling Mutual Occlusions

When two or more targets are being tracked simultaneously, they may overlap in some of the camera fields of view, resulting in a mutual occlusion; this causes part of a target with a higher camera depth to disappear from the field of view, and the missing data should be considered properly by the feature sampling and data association procedures. In particular, we consider pixel- and feature-level occlusions separately.

In the pixel-level case, measurement data are given by images: for example, a color segmentation, based on the respective object statistics, such as histograms or GMMs (Section 4.1), so that each pixel can be labeled according to a target identifier (ID), with 0 representing the background (Fig. 3.10). The same labeling is applied when computing the image expected in a simultaneous target hypothesis or ensemble state, and the two images have to be compared to evaluate the joint residual and its likelihood.

Therefore, occlusion is dealt with by the multilabel shadow h (Fig. 3.11), which is unique only when depth information is available in the camera model. Instead, when depth is missing we need to generate a multihypotheses expectation, where overlapping pixels in the occluding shadows are classified as either of the two targets, with given probabilities that are usually equal, in the absence of prior information. Therefore, multiple residuals e_{pix} are produced and are combined when evaluating the likelihood of the ensemble state \mathbf{s}^- :

$$P(z|\mathbf{s}^-) = \frac{1}{2} [P(z|\mathbf{s}^-, A) + P(z|\mathbf{s}^-, B)] \quad (3.9)$$

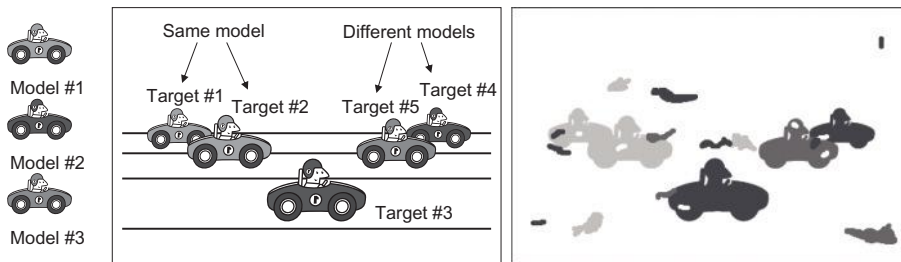


Figure 3.10 Multitarget color segmentation. *Left*: object models; *middle*: image with occlusions between targets; *right*: multilabel color segmentation. Occlusion between targets 1 and 2 cannot be solved, because they have the same model.

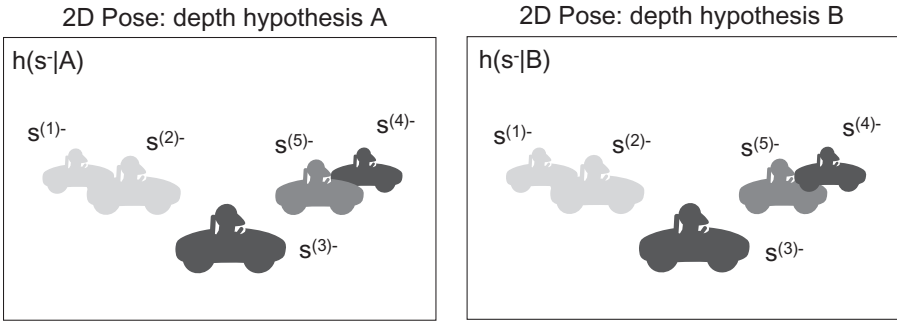


Figure 3.11 Pixel-level matching between the expected image (ensemble shadow) and the color segmentation of Fig. 3.10. In absence of depth information, two hypotheses concerning targets 4 and 5 are generated. Targets 1 and 2 have the same object model; therefore, data in the common region have the same likelihood.

where A and B are the two association hypotheses generating the respective image h . This scheme easily generalizes to more than two occluding targets. We also notice that if two targets with the same color model are occluding, only one hypothesis h is generated.

In the feature-level case, we have a similar distinction between poses with and without depth; here occlusion may also be handled by the feature *sampling* procedure, operating on the average ensemble state \bar{s} , by considering the relative object depths for sampling only in unoccluded areas. The bottom-left part of Fig. 3.2 illustrates an example of multitarget sampling of contour points. The matching procedure will re-project the available features.

An alternative method is to handle occlusion during matching, by avoiding data association in the occluded areas (Fig. 3.12). In this case, occlusion is

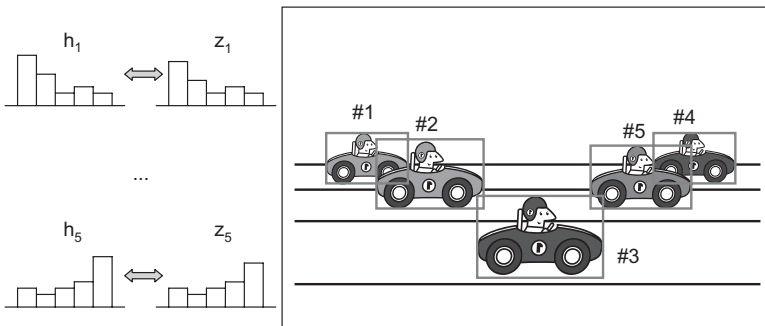


Figure 3.12 Feature-level matching in the presence of occlusions. Color histograms (z) are compared to the expected model histograms (h), where the sampling of z is done only in the unoccluded areas.

handled while computing the residuals for each target, as in the pixel-level example. Also, a combination of both strategies can be thought of (i.e., handling occlusions during sampling *and* data association). If depth is not available, also at the feature level, multiple hypotheses have to be considered, generating a different set of visible features and associated data.

3.3.5 Multiresolution Processing for Improving Robustness

Another important aspect of multimodal visual processing is that the procedures defined above can be carried out at different *resolutions* (or *scales*), to improve robustness. In fact, almost any feature detection and matching procedure selects relevant properties that are more or less related to the average *size* of the object being sought. For example, when detecting foreground blobs, regions with too large or too small an area are usually discarded, and the contours extracted can be a precise approximation of the projected shape, or a coarse but more regular shape, where high-curvature details are lost. The same holds for edge detection, which can be performed naturally at multiple scales by the Marr–Hildreth detector (Section 4.4.1) by means of the filter variance σ . In this case, coarse scales only retain the edges of large blobs with less precision, whereas fine scales detect all object contours and also small details in the scene. Scale-invariant keypoints also perform multiscale processing, but the scale selection is part of the detection procedure.

Generally speaking, any method that performs some *blurring* of the cost function for computing the likelihood, such as the CCD algorithm of Section 4.4.3 or image pyramids in template matching, can be considered a multiresolution matching. Multiresolution approaches have the advantage of a more flexible model representation and generally more robust localization and matching when the estimation is performed in a coarse-to-fine way, especially during maximum-likelihood estimation of object-level measurements.

3.4 DATA FUSION ACROSS MULTIPLE MODALITIES AND CAMERAS

Going back to the example in Figs. 1.7 and 1.8, multiple cameras and visual modalities provide a large data set for each target o , $(Z_{1,b}^o, \dots, Z_{M,C}^o)$, possibly at different processing levels. To obtain a unique measurement z for the Bayesian update, *data fusion* must be performed [51, 71], providing a *global* target measurement Z^o and the related likelihood.

Proper fusion of multisensory and multimodal data can greatly enhance the overall performance of a tracking system [34]. This is particularly crucial for computer vision applications, where different cues and viewpoints provide complementary information about the same object and contribute to a stable and unambiguous localization. For this purpose, we describe here some possible fusion techniques, differentiated according to the use of multiple cameras

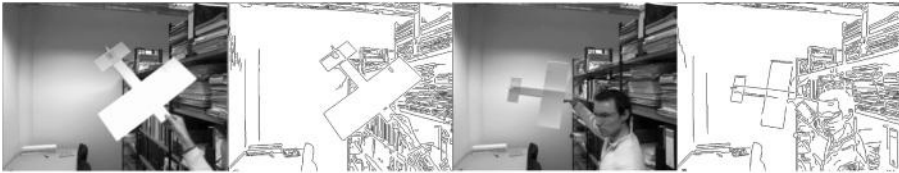


Figure 3.13 Color statistics deal efficiently with situations showing a significant difference between foreground and background regions (*left frame*), but may fail when a weak separation is present (*right frame*). On the contrary, an intensity-based tracker works well when luminance edges can be detected reliably (*right edge map*) but may be challenged by a locally cluttered background (*left edge map*). (From [137].)

and/or modalities, the use of temporal dynamics (static vs. dynamic fusion), and the processing level of the output.

3.4.1 Multimodal Fusion

Multimodal fusion combines different visual modalities related to the same set of targets in order to exploit complementary information and improve robustness. For example (Fig. 3.13), if we know that the object has a distinctive color model as well as distinctive shape contours, by combining both cues we can cope with situations when the background contains similar colors *or* edges, but not both, with respect to the target [137].

3.4.2 Multicamera Fusion

Multicamera fusion is a special case of *multisensor* data fusion. Several sensors, located in different positions and orientations, collect information from different areas of the same object. This can be crucial for tasks such as hand tracking, where the fingers often occlude each other or are occluded by the palm.

Moreover, multicamera tracking often is the only solution for providing full three-dimensional estimation when the shape is very small compared to the camera resolution and object depth. For example, when tracking a ball in sport games, a single view is not sufficient to estimate its depth, whereas two or more views can tackle the problem.

In a multicamera context, however, we need to distinguish between complementary and redundant configurations. A *complementary* setup (Fig. 3.14, top) consists of almost nonoverlapping views where the object to be tracked can be seen completely only by one camera at a time. This scheme is useful when the camera field of view is small compared to the overall state space. For example, in video surveillance of a large building, most cameras cover disjoint areas corresponding to rooms, floors, and so on.

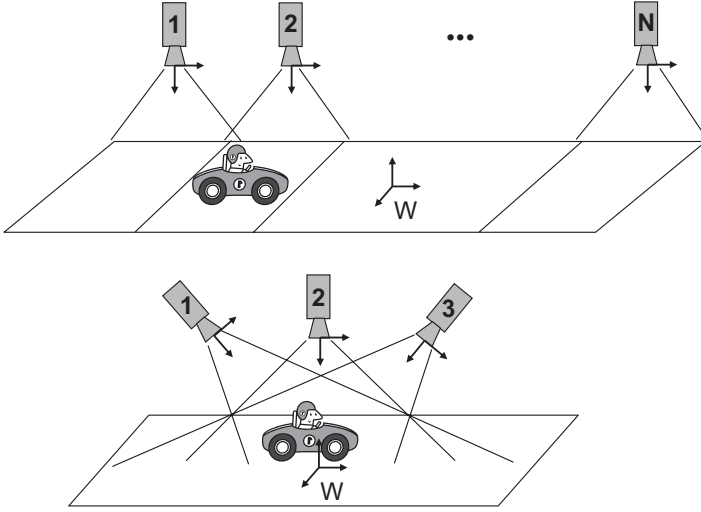


Figure 3.14 Complementary vs. redundant multicamera setup for object tracking.

In this case, each camera takes care of a disjoint subset of targets, and the system cannot take any advantage of multisensory data fusion; moreover, an additional *handover* mechanism is required when a target crosses from one field of view to the adjacent one, which can be realized in a *distributed* way by means of a management *server*, taking care of the target assignment and identity maintainance among local trackers [27]. For this configuration, synchronous image acquisition is not required, since each camera has no data to share or process together with its neighbors, apart from the brief handover time, where the last position estimated for one target is passed on to the next camera as an initial guess for its tracker.

In a *redundant* configuration (Fig. 3.14, bottom), in contrast, multiple fields of view overlap to a large extent, so that the object can be seen simultaneously from different viewpoints. This scheme improves the robustness and accuracy of localization because multiple simultaneous sensory data are integrated about the same target but usually requires that all the cameras be more or less synchronized in order to achieve a consistent estimation.

3.4.3 Static and Dynamic Measurement Fusion

Static fusion computes a global measurement for target o at time k from the individual measurements from C cameras and M modalities:

$$Z_k^o = \mathcal{F}(Z_{k,1,1}^o, \dots, Z_{k,M,C}^o) \quad (3.10)$$

This requires that we compute the combined residual e_k^o as well as the related covariance R_k^o or, equivalently, the combined likelihood $P(Z_k^o|s_k)$, for the Bayesian update.

By omitting the o and k indices, we can write

$$\begin{aligned} h &= \mathcal{F}(h_{1,1}, \dots, h_{M,C}) \\ z &= \mathcal{F}(z_{1,1}, \dots, z_{M,C}) \\ e &= z - h \\ H &= \sum_{m=1}^M \sum_{c=1}^C \left. \frac{\partial \mathcal{F}}{\partial h_{m,c}} \right|_h \cdot H_{m,c} \end{aligned} \quad (3.11)$$

where \mathcal{F} is the fusion strategy, and the joint covariance R , depending on the complexity of \mathcal{F} , may be computed separately in each case from the individual covariances $R_{m,c}$, or may be specified as additional information.

As already mentioned, static data fusion requires *synchronous* image acquisitions, since the data must be consistent with one another. In this context, the maximum acceptable time lag also depends on the object dynamics; in fact, fast motion imposes stricter requirements on the temporal consistency.

Dynamic fusion simply amounts to “stacking” all of the measurements into hybrid-level vectors, which are fed to the Bayesian filter:

$$\begin{aligned} h &= [h_{1,1}, \dots, h_{M,C}] \\ z &= [z_{1,1}, \dots, z_{M,C}] \\ e &= [e_{1,1}, \dots, e_{M,C}] \\ H &= [h_{1,1}, \dots, h_{M,C}] \\ R &= \text{blockdiag}(R_{1,1}, \dots, R_{M,C}) \end{aligned} \quad (3.12)$$

In this case, the filter will actually perform data fusion, also integrating the dynamical prior, for the posterior state update. A dynamic scheme may also be implemented *sequentially* where measurements are integrated one after the other [34, 164], and in that case also handle asynchronous data, provided that each state prediction is carried out according to the respective time stamp (Section 2.4).

For a Kalman filter, this is provably the optimal fusion strategy [34]. For Monte Carlo filters, dynamic fusion provides a global likelihood, which for the same assumption of independent sensors and modalities simply amounts to multiplying the respective likelihoods:

$$P(Z|s) = \prod_{m,c} P(Z_{m,c}|s) \quad (3.13)$$

Other dynamic fusion schemes are possible, such as *ICondensation*: In [85], color blobs provide the *importance distribution* from which to sample new particles and eventually reinitialize the tracker, whereas the weights are updated through a more precise, contour-based likelihood.

Following our multilevel scheme, if we denote by (P, F, O) the level of each measurement $Z_{m,c}$ as well as the desired output Z , we provide some common examples of fusion schemes.

Pixel-Level Fusion: $P + P + \dots \rightarrow P$ Static fusion of pixel-wise maps obtained from the same camera can be carried out in several ways, such as:

- *Weighted average.* The combined measurement is obtained for each pixel (x, y) by

$$z_c(\mathbf{y}) = \sum_m w_m z_{m,c}(\mathbf{y}), \quad \sum_m w_m = 1 \quad (3.14)$$

with proper weights w_m , which can also be adapted in time according to the estimated reliabilities.

- *Voting.* A voting scheme for binary maps consists of assigning the pixel response 0/1 to the target or to the background, according to the percentage of *agreements* between the modalities; for example, the following thresholds are typically employed:

$$z_c(\mathbf{y}) = \begin{cases} 1 & \text{if } \sum_m z_{m,c}(\mathbf{y}) = M & (\text{unanimity}) \\ 1 & \text{if } \sum_m z_{m,c}(\mathbf{y}) > \frac{2}{3} M & (\text{Byzantine}) \\ 1 & \text{if } \sum_m z_{m,c}(\mathbf{y}) > \frac{1}{2} M & (\text{majority}) \\ 1 & \text{if } \sum_m z_{m,c}(\mathbf{y}) > N & (N\text{-out-of-}M) \end{cases} \quad (3.15)$$

- *Bayesian.* This scheme is referred to a *prior confidence* assigned to each modality, which has nothing to do with the Bayesian tracking definition (where the prior is based on temporal dynamics) and is realized by

$$z_c(\mathbf{y}) = P(o_y | z_{1,c}, \dots, z_{M,c}) \propto \prod_m P(z_{m,c} | o_y) \cdot P_c(o_y) \quad (3.16)$$

where $P(z_{m,c} | o_y)$ are individual likelihoods under the hypothesis that object o is present at \mathbf{y} and $P(o_y)$ is the prior probability of this event.

This term can be useful, for example, for excluding areas that contain known background regions, where the object cannot be found in any case [$P(o_y) = 0$], or to focus on areas where the object can be found with a higher probability.

Concerning different cameras, a static pixel-level fusion may be applied only if the images have been registered into a common reference system: for example, by performing dense stereo-disparity computation at each frame. In this way, pixels corresponding to the same three-dimensional point can be related to one another for computing the joint measurement.

Feature-Level Fusion: $F + F + \dots \rightarrow F$ Feature-level measurement can always be combined dynamically, whereas static fusion is seldom possible, because of the variable nature and specificity of each primitive; however, some cases can be thought of as being like elliptical blobs detected from different segmentation maps, which can be merged into an average blob with respect to position, size, and so on.

Hybrid, with Upgrade to Object Level: $P + F + O + \dots \rightarrow O$ In dynamic fusion, multiple levels result in a hybrid data set Z or joint likelihood, used to update the state distribution. In a static context, we can, instead, generalize the unimodal object-level measurement defined in Section 3.3. A joint maximum-likelihood pose estimation can be carried out from more measurements of any level:

$$z = \mathbf{p}^* = \arg \max_{\mathbf{p}} \prod_{m,c} P(z_{m,c} | \mathbf{p}) \quad (3.17)$$

In the case of Gaussian likelihoods, this amounts to a nonlinear least-squares estimation, as explained in Section 5.2.

Object-Level Fusion: $O + O + \dots \rightarrow O$ As a special case of eq. (3.17), multiple independent pose estimates $z_{m,c} = \mathbf{p}_{m,c}^*$ can easily be integrated into a joint linear LSE; for example, if the state consists only of pose variables, we have

$$\begin{aligned} h &= \mathbf{p}^- \\ z &= R \left(\sum_{m,c} R_{m,c}^{-1} \mathbf{p}_{m,c}^* \right) \\ e &= z - h \\ H &= I \\ R &= \left(\sum_{m,c} R_{m,c}^{-1} \right)^{-1} \end{aligned} \quad (3.18)$$

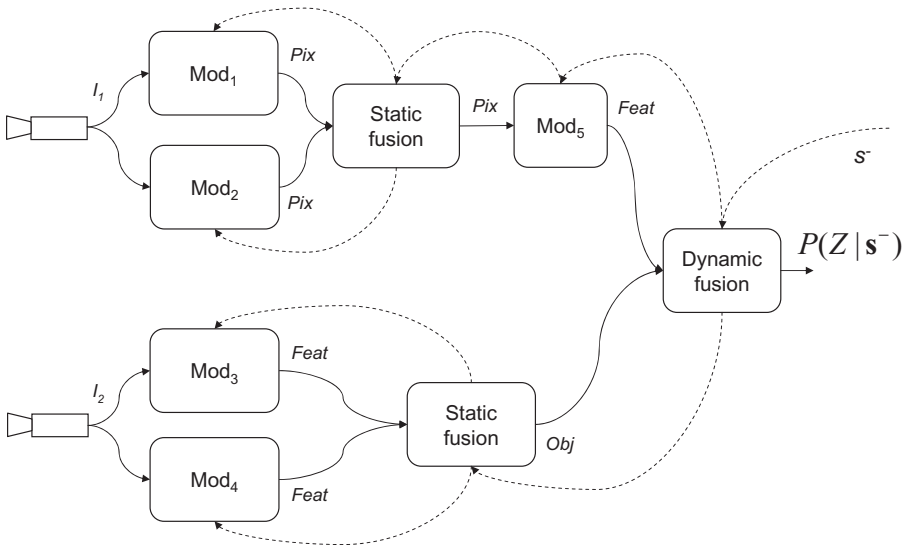
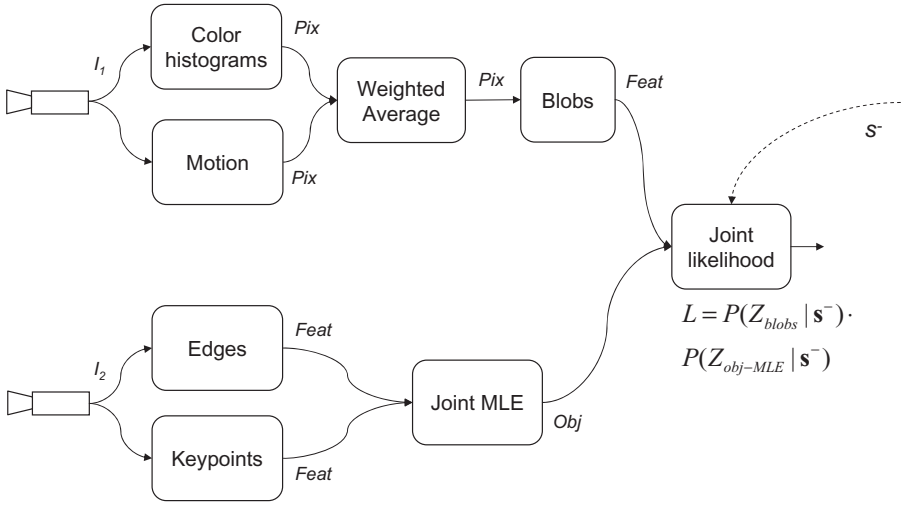


Figure 3.15 Multimodal and multicamera data fusion. For a given state hypothesis s^- , the pipeline methods of Fig. 1.5 (preprocessing, features sampling, matching, and update) are called *recursively*, starting from the root node.

3.4.4 Building a Visual Processing Tree

In real-world applications, multiple modalities may be connected both in cascade and in parallel, to form a visual *processing tree*, where the *root* provides the integrated measurement for each target, which can be fed to any of the Bayesian filters described in Chapter 5. For example, suppose that we wish to realize the scheme described in Fig. 3.15(a); two calibrated cameras provide synchronous images I_1 and I_2 , where I_1 is processed by color and motion segmentation, both returning gray-level saliency maps, followed by blob detection, which produces a feature-level output. Instead, I_2 is processed by a contour-matching algorithm and a keypoint detector, both providing local features and residuals. Subsequently, the first two measurements are combined pixel-wise through a weighted average, while the other two are jointly upgraded to the object level with a maximum-likelihood estimation procedure that finds a local optimum p^* starting from the predicted state s^- . Finally, we perform dynamic fusion of the independent camera measurements, by computing their joint likelihood.

This scheme can be abstracted in terms of modalities and data fusion blocks, as shown in Fig. 3.15(b). In particular, we notice how the static fusion blocks can also be derived from our modality abstraction by defining the fusion procedure as a *matching* function, with input from the respective *children* modalities and output of the desired level. Instead, the dynamic fusion block has a different semantics, because it does not perform any further data processing but multiplies the respective likelihoods (or equivalently, *stacks* the two measurements into one), letting the Bayesian update be responsible for the integration. The overall workflow of the pipeline, described in Fig. 1.5 and exemplified in Fig. 1.6, is then accomplished by calling any of the preprocessing, feature sampling, matching, and updating methods in a *recursive* fashion, starting from the root of the processing tree.

CHAPTER 4

EXAMPLES OF VISUAL MODALITIES

Many different visual modalities and feature-processing methods can be used for the same object-tracking task. In this chapter we provide a variety of examples, following the abstraction of Chapter 3 and organized in the following way (Fig. 4.1):

- *Global statistics.* Reference statistics (e.g., a color distribution) are collected from the visible object surface, and matched to the image statistics under the region of interest.
- *Background model.* A reference pixel-wise model of the background can be matched with the current image in order to remove data that should not belong to any object.
- *Blobs.* Connected components are clustered from a given saliency map and matched to the model silhouette projected.
- *Contours.* Sharp intensity, color, and texture transitions in the image are matched to the model contours projected.
- *Motion.* A motion field detected in the image (scalar or vector) is compared to the expected motion field generated by the object.
- *Keypoints.* Distinctive gray-level patches are detected and matched to those sampled from the model surface.
- *Template.* A fully textured model is rendered and matched to the underlying pixel colors.

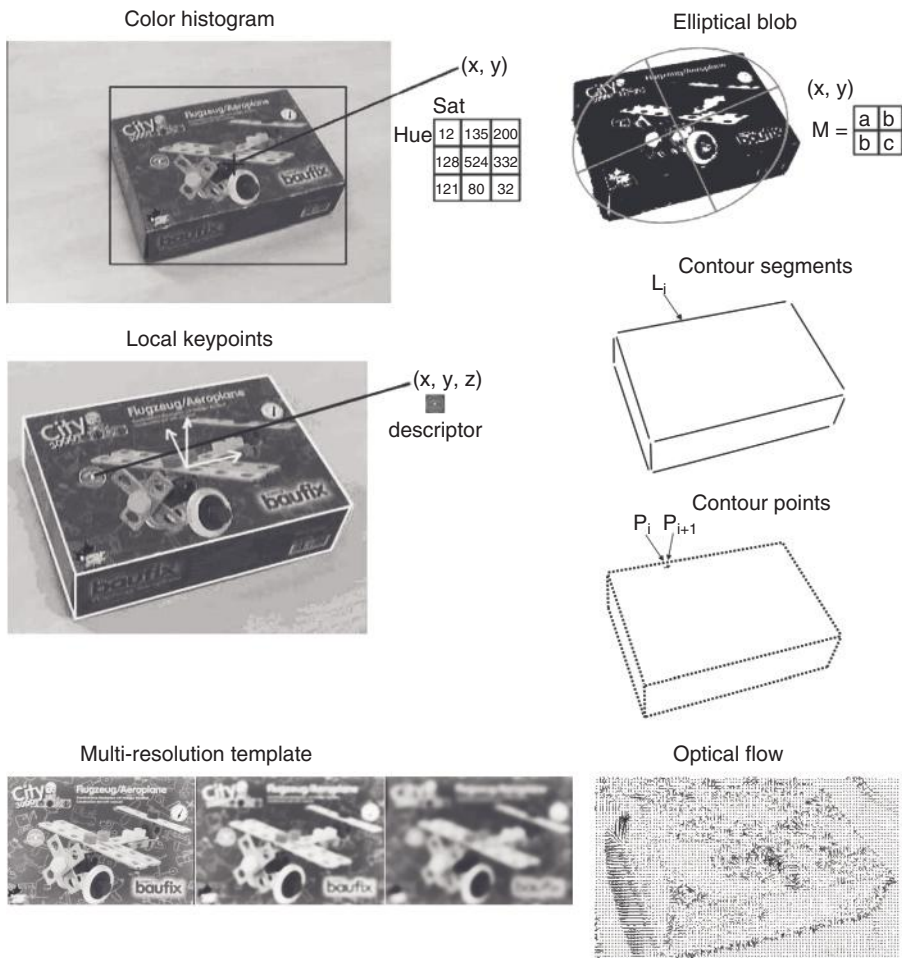


Figure 4.1 Visual modalities for object tracking.

Following our abstraction for visual modalities (Fig. 3.1), these definitions will also be differentiated according to the processing level, defined in Section 3.3. As we can see, for each modality the specific model and image features need to be defined in order to describe the matching procedure and likelihood function.

4.1 COLOR STATISTICS

Color statistics represent the distribution of image color under the region of interest where the object is sought. For the purpose of model-based tracking, this region can be obtained by projecting the shape at a given pose hypothesis



Figure 4.2 RGB color space decomposition. (From [24].) (*See insert for color representation of the figure.*)

p, and computing its silhouette or simply a two-dimensional bounding box. Color pixels are collected from this region and accumulated in a global descriptor that represents the underlying *distribution*, characterizing the object from that point of view; this method therefore requires a more-or-less specific appearance model, such as key frames.

We note that a color model depends on the viewpoint, since the object surface may show different colors. The same reasoning applies to other global descriptors, such as histograms of edge orientations or texture. Moreover, this distribution must be defined in the proper *color space*, which may be different from the standard red–green–blue (RGB), and may also be limited to fewer than three channels, or to the luminance channel only.

4.1.1 Color Spaces

The first issue in color-based tracking is proper representation of the color space: a mathematical model describing how perceptually different colors can be represented by t -uples of numbers, typically three values. For example, since any visible color can be obtained by combining a given amount of red, green, and blue, the RGB color space (Fig. 4.2) can be represented by an orthogonal Cartesian frame, where the three axes represent the respective percentage of each component, normalized from 0 to 1.

Many different color spaces have been defined, both according to psycho-physical or technical representation criteria, along with the *mapping* between spaces. When dealing with devices such as printers or monitors, color spaces also take into account the difference between the human perception and the device representation capabilities given by the *gamut*, a specific subset of visible colors that can actually be represented and processed by the device.

In computer vision, the choice of color space may be crucial, since any segmentation or target localization procedure may benefit greatly from a well-behaved distribution of the color features, a distribution that allows easy classification of target pixels among different objects, including the background.

We begin by considering the human perception of color, which is based on the response of three types of receptors in the retina, called *cones*. Each cone is denoted either S, M, or L (short, medium, or long wavelength), each of which has a different range of sensitivity to the wavelengths of the visible light spectrum, depicted on the left in Fig. 4.3. The S-cones are more sensitive to high frequencies, corresponding to violet light, while the M- and L-cones are more

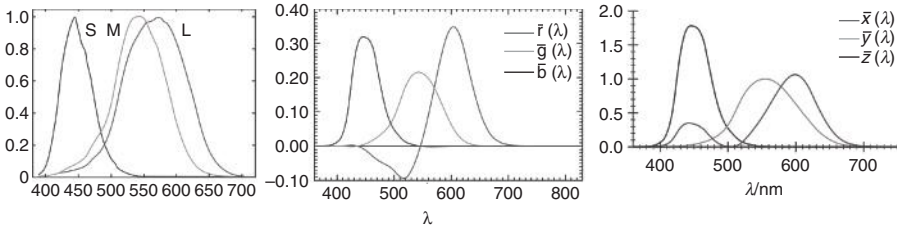


Figure 4.3 Left: tristimulus curves for the three types of retinal cone cells (from [8]); middle: CIE-RGB responses; right: CIE-XYZ responses (from [6]). All wavelengths are expressed in nanometers.

sensitive to green and yellow-green light (but not red, as is often thought), respectively. For light containing multiple frequencies (i.e., not a “pure” color), the overall response is *integrated* over wavelengths, producing the output known as *tristimulus* values.

Therefore, the perceived color space is *three-dimensional* despite the infinite dimensions that would be necessary to describe an arbitrary periodic signal, through the Fourier analysis. In fact, two light waves with different spectral power distribution, which produce the same tristimulus values, are indistinguishable to the human eye, a phenomenon also known as *metamerism*. Moreover, since the response curves overlap to a large extent, some tristimulus values cannot be produced by any visible light; for example, no light can stimulate only the M-cones without at the same time stimulating the L-cones.

One of the first color spaces developed following this theory was the *CIE-RGB space* [86], proposed by the International Commission on Illumination (also known as the ICI). It was derived from experimental results combined with the RGB tristimulus curves shown in the middle of Fig. 4.3, which are slightly different from the actual responses of the retinal cones because the red response is shifted to 700 nm. Therefore, it has less overlap with the green response and also contains *negative* values in a certain range. If $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, and $\bar{b}(\lambda)$ are the responses at each wavelength λ , and $I(\lambda)$ denotes the spectral power of incident light, then

$$R = \int_0^\infty I(\lambda) \bar{r}(\lambda), \quad G = \int_0^\infty I(\lambda) \bar{g}(\lambda), \quad B = \int_0^\infty I(\lambda) \bar{b}(\lambda) \quad (4.1)$$

where the curves are normalized to cover the same amount of spectral energy:

$$\int_0^\infty \bar{r}(\lambda) = \int_0^\infty \bar{g}(\lambda) = \int_0^\infty \bar{b}(\lambda) \quad (4.2)$$

One important property of this representation is given by *Grassmann’s law* of optics, which can be expressed as follows: If a test color is obtained by mixing two components, and each component can be expressed as a combination of

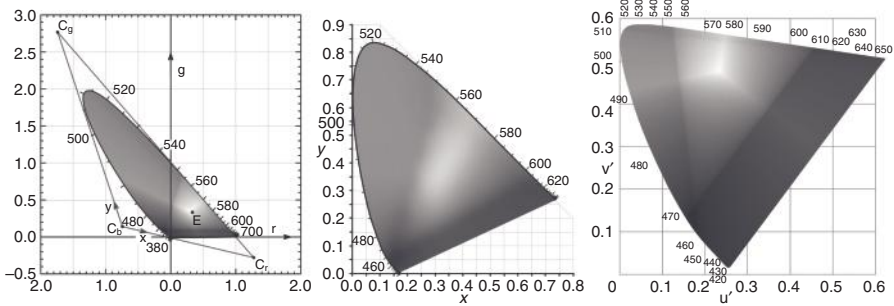


Figure 4.4 *Left: CIE-r-g chromaticity diagram; middle: CIE-x-y diagram, obtained after normalizing the gamut (left triangle) with a linear mapping (from [6]); right: the $(u'v')$ chromatic space attempts to obtain a perceptually uniform color distribution (from [5]). (See insert for color representation of the figure.)*

primary colors (R_1, G_1, B_1) , (R_2, G_2, B_2) , the primaries of the mixed color are obtained by *adding* the respective primaries $R = R_1 + R_2$, and so on. In this space, one can define the *chromaticity* of a color as

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B} \quad (4.3)$$

where it can be shown that (r, g) are sufficient to identify the color perceptually after removing its *brightness*. The chromaticity diagram is shown on the left in Fig. 4.4; the outer contour, also known as the *monochromatic locus*, contains pure colors with only one spectral component, denoted by the respective wavelength number, while the central point E is the *white spot*, or *achromatic light*. The triangle shown contains the range of colors that are usually representable by a visualization device such as a monitor.

Because of the property described above, the CIE-RGB is a *linear* color space, which can be transformed into a similar space by a linear mapping. This leads to the CIE-XYZ representation (Fig. 4.4, middle), where the tristimulus curves $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ satisfy these properties:

- The new responses are nonnegative everywhere.
- The $\bar{y}(\lambda)$ curve exactly matches the *luminous efficiency* function (or *luminance*) $V(\lambda)$, defined by the CIE in 1924 as an experimental curve reporting the variation of the perceived brightness of colors with the wavelength.
- For the white spot, the values should be $X = Y = Z = 1/3$.
- The range of visible colors, or *gamut*, in the new chromaticity plane (x, y) should be normalized to the triangle $(0, 0)$, $(1, 0)$, $(0, 1)$.

These considerations lead to the following linear mapping between the two spaces:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.0 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (4.4)$$

Another desirable property of a color space is *perceptual uniformity*: A given amount of change in a color value should result in a proportional amount of the perceived variation. The *XYZ* representation still has the disadvantage of not being perceptually uniform, as can be seen from its chromaticity diagram. Therefore, in 1976 the CIE proposed a simple modification providing more uniformity, as well as a better approximation of the luminance component, in terms of the perceived brightness. This was the *CIE-L*a*b** (or *CIE-LAB*) color space, designed to approximate human vision.

In this space, L^* is the *lightness* of a color, ranging from 0 (black) to 100 (diffuse white, nonspecular), while a^* is the position between green and red/magenta, respectively, from negative to positive values, and b^* the position between blue and yellow, again from negative to positive. The reason for the $*$ symbol is to distinguish these components from *Hunter's* (L, a, b) components (see [82] for more details).

In particular, the nonlinear relations between L^* , a^* , and b^* should mimic the nonlinear response of the human eye, so that the perceptual difference between two colors in this space can be well approximated by the Euclidean distance in three-dimensional space. Conversion formulas between *XYZ* and $L^*a^*b^*$ are given below.

1. Forward transformation:

$$\begin{aligned} L^* &= 116f(Y/Y_n) - 16 \\ a^* &= 500[f(X/X_n) - f(Y/Y_n)] \\ b^* &= 200[f(Y/Y_n) - f(Z/Z_n)] \end{aligned} \quad (4.5)$$

where X_n , Y_n , and Z_n are the tristimulus values for the reference white point (n stands for *normalized*), and

$$f(t) = \begin{cases} t^{1/3}, & t > t_0 = (6/29)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29}, & \text{otherwise} \end{cases} \quad (4.6)$$

In this formula, the $f(t)$ function has two domains, to avoid an infinite slope at $t = 0$, and the linear part, defined below a threshold t_0 , matches both the value and slope of the upper part at t_0 .

2. *Reverse transformation.* Let $f_y \equiv (L^* + 16)/116$, $f_x \equiv f_y + a^*/500$, and $f_z \equiv f_y - b^*/200$. Then:

$$\begin{aligned} \text{If } f_x > \delta, \text{ then } X &= X_n f_x^3, \text{ else } X = (f_x - 16/116) 3\delta^2 X_n \\ \text{If } f_y > \delta, \text{ then } Y &= Y_n f_y^3, \text{ else } Y = (f_y - 16/116) 3\delta^2 Y_n \\ \text{If } f_z > \delta, \text{ then } Z &= Z_n f_z^3, \text{ else } Z = (f_z - 16/116) 3\delta^2 Z_n \end{aligned} \quad (4.7)$$

where $\delta = (6/29)$. A related space is the 1976 *CIE-L*u*v**, which also attempts to achieve perceptual uniformity but with a different representation of the chromaticity, shown on the right in Fig. 4.4. In this space, the range for L^* is the same as before, while the chromatic components u^* and v^* vary between ± 100 for typical images; nonlinear conversion formulas from *XYZ* may be found in Scharda's book [149]. We notice that the diagram shown actually concerns the intermediate variables (u', v') , linearly related to the (u^*, v^*) quantities, after subtracting the prespecified white point coordinates (u'_n, v'_n) . The Euclidean distance can also be used here, to measure color differences.

Perceptually uniform spaces are well suited for color segmentation and model-matching algorithms because they result in well-distributed *clusters* of color pixels, as can be seen from the example in Section 4.1.4 (Fig. 4.11, top).

A variant of the RGB color space often used in computer vision, which attempts to achieve more intuitive coordinates, is the hue–saturation–value (HSV) representation. This space is obtained by converting the RGB Cartesian cube into cylindrical coordinates, corresponding roughly to three perceptual properties of a color: its identity, or *hue*; its “purity,” or *saturation*; and its *lightness*, or *value*.

In reference to Fig. 4.5, this transformation is achieved by tilting the RGB cube around the origin $(0, 0, 0)$ (corresponding to black), so that the main diagonal of the cube becomes vertical and the main axis of this cylinder

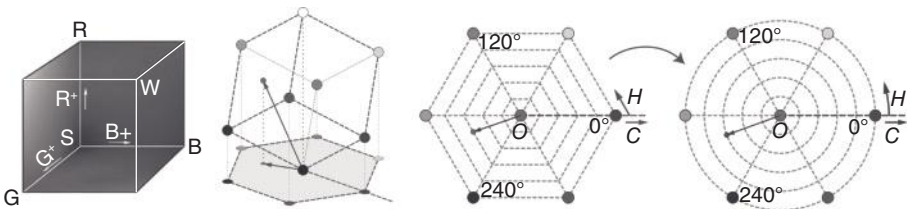


Figure 4.5 Conversion from RGB to HSV. *Left to right:* the RGB cube; tilting the cube to set the vertical axis to the main diagonal; a slice of the cube orthogonal to the main axis which provides hue and saturation, expressed in *hexagonal* coordinates; the same chromaticity plane, in a *circular* representation (angle and radius) obtained after warping the hexagonal coordinates. (From [9].)

contains all gray values, from black to white. Then, by slicing the cube with a plane orthogonal to the main axis, one obtains the chromaticity plane, which contains all colors distributed into a hexagon.

Therefore, conversion from RGB to HSV is done piecewise:

$$\begin{aligned}
 C &= \max(R, G, B) - \min(R, G, B) \\
 H' &= \begin{cases} \text{undefined,} & \text{if } C = 0 \\ \frac{G-B}{C} \bmod 6, & \text{if } M = R \\ \frac{B-R}{C} + 2, & \text{if } M = G \\ \frac{R-G}{C} + 4, & \text{if } M = B \end{cases} \\
 H &= 60^\circ \cdot H' \\
 V &= \max(R, G, B) \\
 S &= \begin{cases} 0, & \text{if } C = 0 \\ \frac{C}{V}, & \text{otherwise} \end{cases}
 \end{aligned} \tag{4.8}$$

where the saturation component is the chroma scaled by the lightness, so that the HSV color space describes all visible colors in the following intervals: $S \in [0, 1]$, $V \in [0, 1]$, and $H \in [0^\circ, 360^\circ]$.

The value component (corresponding roughly to the intensity) is given by the height of the plane; the hue is given by the angle of a vector pointing to the color in the chromaticity plane; and the *chroma* component, related to the saturation, is the distance of the color from the main axis. However, notice that these coordinates are referred to the *hexagonal* representation, not to the circular one shown in Fig. 4.5, obtained by warping the hexagonal coordinates back in Cartesian space.

4.1.2 Representing Color Distributions

A simple example of color statistics can be given by the description “The object surface, from this viewpoint, shows 70% green and 30% blue pixels.” This type of description may, for example, be encoded into a color *histogram*, defined in a suitable color space (Fig. 4.6). Histograms are discrete, nonparametric representations of probability distributions in which the only parameter to be provided is the number of cell subdivisions for each dimension.

On the other hand, a parametric representation encodes the color distribution by means of a *likelihood* function:

$$P(\mathbf{c}|\theta) \tag{4.9}$$

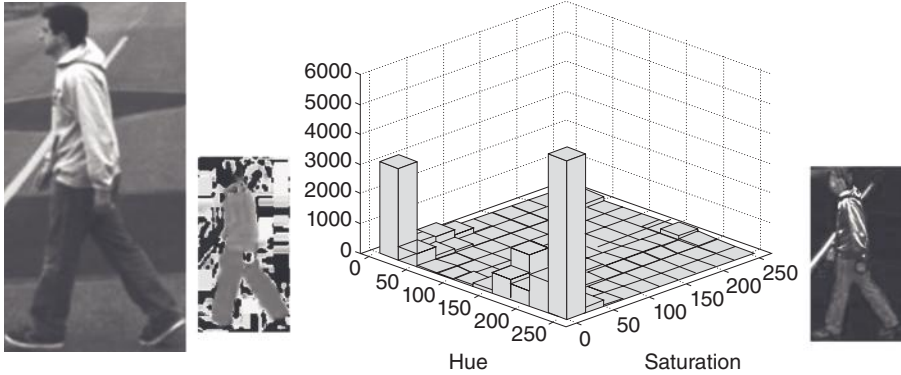


Figure 4.6 Hue-saturation color histogram. *Left:* original RGB image; *right:* histogram of hue-saturation color channels.

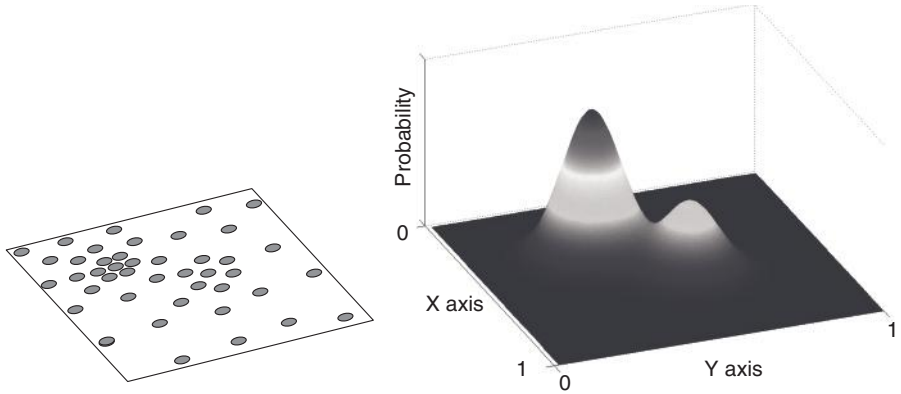


Figure 4.7 Mixture of Gaussians for parametric representation of a probability distribution.

parametrized by a vector θ , where \mathbf{c} is the color of a given pixel, using one, two, or three channels. According to the choice of P and θ , more or fewer details of the distribution can be captured; if P is also differentiable, derivative-based methods for maximum-likelihood estimation can be employed.

A typical example of a parametric model is the mixture of Gaussians (Fig. 4.7),

$$P(\mathbf{c}) = k \sum_{i=1}^M w_i \exp\left(-\frac{1}{2}(\mathbf{c} - \bar{\mathbf{c}}_i)^T \mathcal{C}_i^{-1}(\mathbf{c} - \bar{\mathbf{c}}_i)\right) \quad (4.10)$$

where \mathbf{c}_i and \mathcal{C}_i are the mean and covariance matrix of each component, w_i are the weights (with $\sum_i w_i = 1$), and k is a normalization constant, so that P integrates to 1 over the color space. In this case, the parameters are

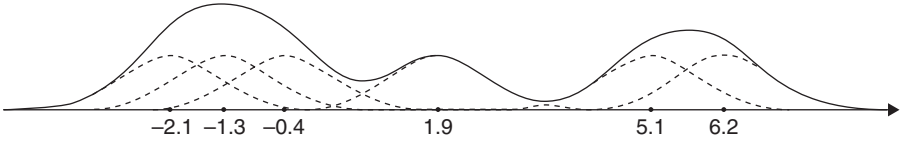


Figure 4.8 Kernel density estimation.

$$\theta = (M, \mathbf{c}_1, C_1, w_1, \dots, \mathbf{c}_M, C_M, w_M) \quad (4.11)$$

which includes the number of components M of the mixture. For a given sample set, estimating all of these parameters amounts to another maximum-likelihood problem, which is a nontrivial task that can be performed via the expectation-maximization (EM) algorithm¹ [40,59].

Within the nonparametric class, *kernel-based* methods provide a more flexible and differentiable representation. If c is one-dimensional (Fig. 4.8), the kernel representation is given by

$$P(c|\theta) = \frac{1}{N} \sum_{i=1}^N \frac{1}{h_i} k\left(\frac{c - c_i}{h_i}\right) \quad (4.12)$$

In this formula, k is the kernel, which has a maximum value in 0 and quickly decays in a neighborhood of the origin; the parameters h_i , called *bandwidths*, regulate the width of the kernel around each color sample c_i . The number of sample points N also defines the representation, so that $\theta = h$ is the only parameter, although quite critical, to be specified.

In a multidimensional space, the kernel representation generalizes to

$$P(\mathbf{c}) = \frac{1}{N} \sum_{i=1}^N \frac{1}{\det H_i} \mathcal{K}(H_i^{-1}(\mathbf{c} - \mathbf{c}_i)) \quad (4.13)$$

where the multivariate kernel \mathcal{K} is obtained as the product of univariate kernels:

$$\mathcal{K}(\mathbf{c}) = \prod_{d=1}^D k(c_d) \quad (4.14)$$

with D the space dimension. If all H_i are equal, with diagonal matrices $H_i = hI$, the multivariate kernel is *radially symmetric*, and the probability density function (pdf) becomes

¹Special care has to be taken when estimating M , which can be done via another maximum-likelihood, or minimum-entropy, criterion.

$$P(\mathbf{c}) = \frac{1}{Nh^D} \sum_{i=1}^N k\left(\left\|\frac{\mathbf{c} - \mathbf{c}_i}{h}\right\|\right) \quad (4.15)$$

Several choices of \mathcal{K} (or, equivalently, k) can be made, usually satisfying the following properties:

- Exponential decay: $\lim_{\|\mathbf{x}\| \rightarrow \infty} \|\mathbf{x}\|^D \mathcal{K}(\mathbf{x}) = 0$, or compact support: $\|\mathbf{x}\| > t \Rightarrow \mathcal{K}(\mathbf{x}) = 0$
- Normalization: $\int_{\mathbb{R}^D} \mathcal{K}(\mathbf{x}) d\mathbf{x} = 1$
- Symmetry: $\int_{\mathbb{R}^D} \mathbf{x} \mathcal{K}(\mathbf{x}) d\mathbf{x} = 0$

For example, the following radially symmetric kernels satisfy the foregoing conditions:

- Uniform kernel:

$$\mathcal{K}_U(\mathbf{x}) = \begin{cases} c_D, & \|\mathbf{x}\| \leq 1 \\ 0, & \|\mathbf{x}\| > 1 \end{cases} \quad (4.16)$$

- Epanechnikov kernel:

$$\mathcal{K}_E(\mathbf{x}) = \begin{cases} \frac{1}{2} c_D^{-1} (D+2) (1 - \|\mathbf{x}\|^2), & \|\mathbf{x}\| \leq 1 \\ 0, & \|\mathbf{x}\| > 1 \end{cases} \quad (4.17)$$

- Gaussian (normal) kernel:

$$\mathcal{K}_N(\mathbf{x}) = (2\pi)^{-D/2} e^{-\frac{1}{2}\|\mathbf{x}\|^2} \quad (4.18)$$

In particular, in the Gaussian case, $h = \sigma$ is the standard deviation, so that eq. (4.12) represents an equally weighted but nonparametric mixture of Gaussians, apart from the only scalar value of h , since the means are determined by the sample set and the covariances are fixed in advance.

A radially symmetric multivariate kernel [eq. (4.15)] can also be expressed in the form

$$\mathcal{K}(\mathbf{x}) \propto k_p(\|\mathbf{x}\|^2) \quad (4.19)$$

where $k_p(x)$ is a *profile* function that has to be defined only for positive values $x \geq 0$; for example:

- Epanechnikov profile:

$$k_{p,E}(\mathbf{x}) = \begin{cases} 1-x, & 0 \leq x \leq 1 \\ 0, & x > 1 \end{cases} \quad (4.20)$$

- Gaussian profile:

$$k_{p,N}(\mathbf{x}) = \exp\left(-\frac{1}{2}x\right), \quad x \geq 0 \quad (4.21)$$

yield the kernels (4.17) and (4.18), respectively.

4.1.3 Model-Based Color Matching

According to the density representation chosen, model-based color matching can be performed in different ways at both the pixel and feature levels. At the pixel level, supervised color segmentation is carried out by classifying individual pixels as target or background. A binary classification can be obtained by computing the likelihood of each pixel \mathbf{x} being generated by target o (with $o=0$ denoting the background), and thresholding this value:

$$o_{\mathbf{x}}^* = [P(\mathbf{c}_{\mathbf{x}}|o) - P_{\min}]^+ \quad (4.22)$$

where the target assignment o^* can be encoded in a pseudocolor label. A *fuzzy* classification map can also be built by assigning to each pixel the maximum likelihood $P(\mathbf{c}_{\mathbf{x}}|o^*)$, encoded in a gray-level map, this time providing a separate map for each target. Figure 4.9 shows an example of single-target histogram *back-projection*. The model histogram, represented by the rectangular patch, is used in order to compute the pixel likelihoods shown on the right.

Feature-level matching can be obtained by direct comparison of the statistics expected and those observed, which is the global feature of interest. For the histogram representation in particular, the Bhattacharyya coefficient B and distance D are typically used:

$$\begin{aligned} D(q^*, q) &= \sqrt{1 - B(q^*, q)} \\ B(q^*, q) &= \sum_b \sqrt{q_b^* q_b} \end{aligned} \quad (4.23)$$

In this formula, q^* is the reference histogram and $q(s)$ is the histogram observed, from image pixels underlying the hypothesis s . The sum is performed over the histogram bins in the given color space.

A geometric interpretation of the Bhattacharyya coefficient is shown in Fig. 4.10: Given two histograms p and q , by considering the vectors $p' = (\sqrt{p_1}, \dots, \sqrt{p_n})$ and $q' = (\sqrt{q_1}, \dots, \sqrt{q_n})$, with unit Euclidean norm,

$$B(p, q) = \cos \theta(p', q') = \frac{p'^T q'}{\|p'\| \|q'\|} = \sum_i \sqrt{p_i q_i} \quad (4.24)$$



Figure 4.9 Back-projection of an image on the model color histogram. *Left*: original image and model; *right*: back-projection on the model histogram. (From [45].)

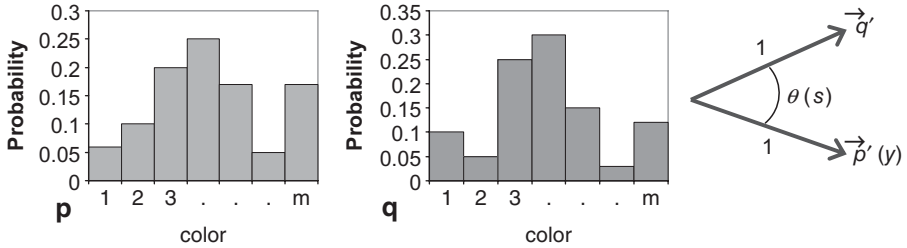


Figure 4.10 Representation of the Bhattacharyya distance between color histograms.

and therefore B is the *divergence* between p' and q' , represented by the angle θ . This is just one example of a divergence measure between distributions; another well-known measure is the *Kullback–Leibler divergence*, derived from information theory [56] (see also Section 4.7.2).

For parametric densities, if we consider the single-Gaussian case, where the reference distribution $g^* \sim \mathcal{N}(\mu^*, \Sigma^*)$ must be matched to the Gaussian observed, $g \sim \mathcal{N}(\mu, \Sigma)$, the Bhattacharyya coefficient is given by

$$B(g^*, g) = \frac{1}{8}(\mu^* - \mu)^T \left[\frac{\Sigma^* + \Sigma}{2} \right]^{-1} (\mu^* - \mu) + \frac{1}{2} \ln \frac{\left| \frac{\Sigma^* + \Sigma}{2} \right|}{\sqrt{|\Sigma^*|} \sqrt{|\Sigma|}} \quad (4.25)$$

Image patches can be localized by minimizing one of the distances defined above, by means of a suitable optimization algorithm.

4.1.4 Kernel-Based Segmentation and Tracking

The *mean-shift* approach consists of finding the *modes* of a kernel density estimate, and can be used both for solving unsupervised clustering problems

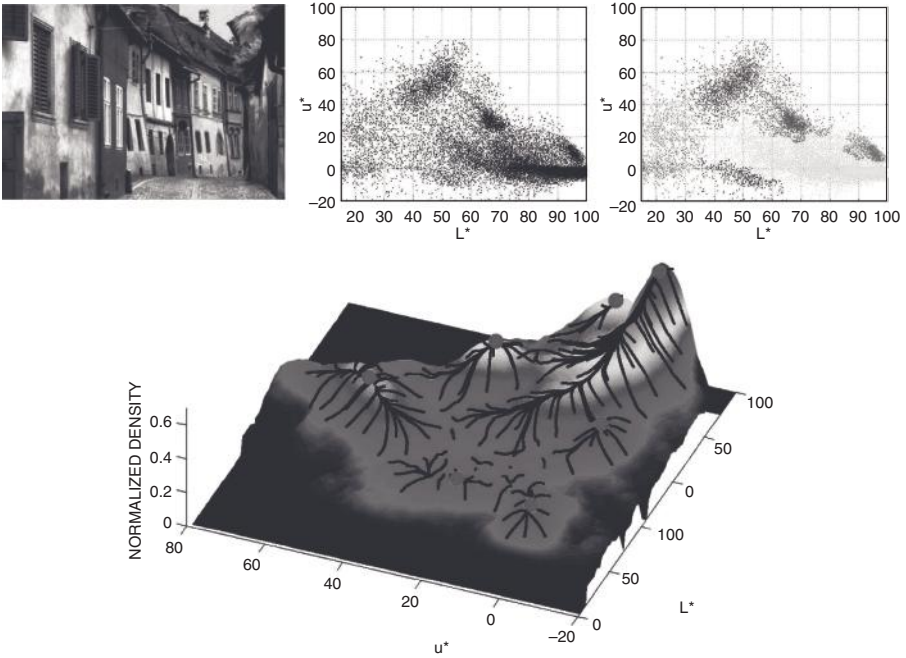


Figure 4.11 Color segmentation by mean-shift clustering in $L^*u^*v^*$ color space. *Top left*: original image; *top center*: distribution of color pixels, transformed in $L^*u^*v^*$ space; *top right*: result of mean-shift clustering; *bottom*: modes of kernel-based likelihood, found by mean-shift optimization. (From [52], Copyright © 2002 IEEE.)

[52] such as color segmentation (top of Fig. 4.11) and for object tracking purposes [53]. The basic idea consists of looking for local maxima of this distribution, starting from each kernel point, and performing a gradient ascent optimization, which usually converges in a few steps (Fig. 4.11, bottom).

By assuming a radially symmetric kernel, the density can be optimized by gradient descent by computing the mean-shift vector,

$$m_h(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{x}_i g\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x} \quad (4.26)$$

where $g = -k'_p$ is the first derivative of the profile function. As shown by Comaniciu and Meer [52], the mean-shift vector always points in the direction of maximally increasing density and, for an appropriate choice of h , has a fast convergence from any sample point to the local optimum. Mean-shift segmentation considers each pixel as a D -dimensional feature, consisting of color and

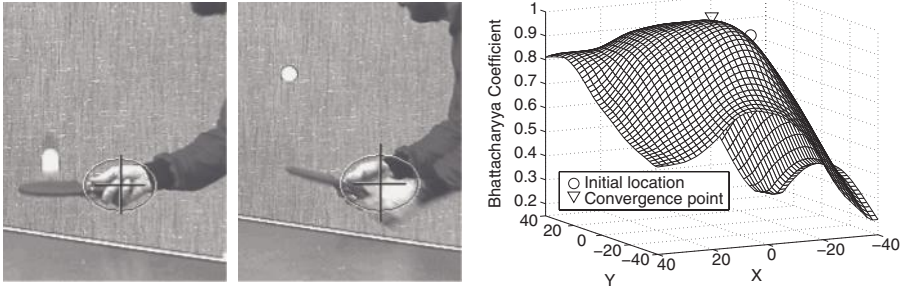


Figure 4.12 Mean-shift optimization for blob tracking. (From [53]. Copyright © 2003 IEEE.)

image position as well as other properties, such as texture or motion, that may characterize the region being sought, and therefore forming a distinctive *cluster* in feature space. Subsequently, this region can be used as a blob measurement (Section 4.3) in order to detect or track an object by matching the re-projecting shape.

The same mean-shift algorithm can also be applied to two-dimensional *blob tracking* [53], where a global feature such as a rectangular patch undergoes a pure translational motion, eventually followed by scale adjustments, where the descriptor is updated from frame to frame. This corresponds to a feature-tracking approach, as discussed in Section 3.3.2. For this purpose we can formulate the kernel likelihood in the two-dimensional parameter space (t_{y1}, t_{y2}) and then localize the new blob position by local gradient ascent, starting from the position predicted (Fig. 4.12).

To go into greater detail, if we write the Bhattacharyya coefficient in terms of the position \mathbf{t} of the window,

$$B(\mathbf{t}) = \sum_b \sqrt{q_b^* q_b(\mathbf{t})} \quad (4.27)$$

we can linearize it around \mathbf{t}_0 :

$$B(\mathbf{t}) \approx \sum_b \sqrt{q_b^* q_b(\mathbf{t}_0)} + \sum_b \frac{1}{2} \sqrt{\frac{q_b^*}{q_b(\mathbf{t}_0)}} q'_b(\mathbf{t}_0) \cdot (\mathbf{t} - \mathbf{t}_0) \quad (4.28)$$

where q'_b is the first derivative of bin $q_b(\mathbf{t})$ in the current histogram with respect to \mathbf{t} . This quantity can, in turn, be obtained by linearization:

$$q_b(\mathbf{t}) - q_b(\mathbf{t}_0) \approx q'_b(\mathbf{t}_0) \cdot (\mathbf{t} - \mathbf{t}_0) \quad (4.29)$$

and therefore

$$B(\mathbf{t}) \approx \frac{1}{2} \sum_b \sqrt{q_b^* q_b(\mathbf{t}_0)} + \frac{1}{2} \sum_b q_b(\mathbf{t}) \sqrt{\frac{q_b^*}{q_b(\mathbf{t}_0)}} \quad (4.30)$$

The next step toward a kernel-based formulation consists of replacing the standard histogram contributions to $q_b(\mathbf{t})$,

$$q_b(\mathbf{t}) = \frac{1}{n} \sum_{i: \text{bin}(\mathbf{y}_i)=b} (1) \quad (4.31)$$

where the sum is performed over pixels \mathbf{y}_i whose color matches the bin b , and n is the overall number of pixels, with a *fuzzy binning*,

$$q_b(\mathbf{t}) = C_h \sum_{i: \text{bin}(\mathbf{y}_i)=b} k_p \left(\left\| \frac{\mathbf{t} - \mathbf{y}_i}{h} \right\|^2 \right) \quad (4.32)$$

where the contribution of \mathbf{y}_i is *weighted* by its distance to the center, through a given kernel profile $k_p(\cdot)$. Then eq. (4.30) becomes

$$B(\mathbf{t}) \approx \frac{C_h}{2} \sum_{i=1}^n w_i k_p \left(\left\| \frac{\mathbf{t} - \mathbf{y}_i}{h} \right\|^2 \right) \quad (4.33)$$

with the sum performed over all pixels, and

$$w_i = \sqrt{\frac{q_{\text{bin}(\mathbf{y}_i)}^*}{q_{\text{bin}(\mathbf{y}_i)}(\mathbf{t}_0)}} \quad (4.34)$$

are the respective weights, computed after accumulating the histograms $q_b^*, q_b(\mathbf{t}_0)$. Optimization of B is then performed by the weighted mean shift

$$\Delta \mathbf{t} = \frac{\sum_{i=1}^n w_i \mathbf{t}_i g \left(\left\| \frac{\mathbf{t} - \mathbf{y}_i}{h} \right\|^2 \right)}{\sum_{i=1}^n w_i g \left(\left\| \frac{\mathbf{t} - \mathbf{y}_i}{h} \right\|^2 \right)} - \mathbf{t} \quad (4.35)$$

which generalizes eq. (4.26) to a weighted sample set.

4.2 BACKGROUND SUBTRACTION

When the camera position is constant in time, a single image of the background provides very valuable information: This is the case, for example, in indoor or

outdoor video surveillance, where no prior models of people are available, so that the presence of people can be deduced mostly by their difference with respect to the known background.

Several well-known approaches to background subtraction can be found in the literature, including efficient implementations based on graphics hardware [69]. Review papers on the subject are available (e.g., [38,142,143]), as well as comparative evaluation studies of several algorithms on experimental data sets (e.g., [139]).

Most of these algorithms are based on a *learning* phase, where the background model is built pixel-wise from one or more example images, where no foreground objects are present, producing a statistical model of local properties such as color (Fig. 4.13), texture, or edges, which may be represented pixel-wise or as background features.

At the pixel level, the complexity of this representation may range from simple classification criteria such as thresholding image differences, up to complex and computationally expensive statistics, using mixtures of Gaussians, local histograms, and so on. Therefore, for real-time applications, a trade-off between speed and precision of the subtraction algorithm must be chosen.

As an example, we mention two color background subtraction algorithms:

- *Preprocessing.* The incoming image is transformed into the color space used for matching (e.g., from RGB to HSV). Unsupervised segmentation may take place here (Section 4.1.4).
- *Sampling model features.* Expected off-line color statistics are computed by rendering the model at the pose predicted or by using the nearest key frame and collecting color values.
- *Matching*
 1. Pixel level. Individual pixels are classified according to the statistics expected, off-line and online, and provide a pixel-wise data association, binary or gray level for a single target, or multilabel for multiple targets. The residual is evaluated as a pixel-wise difference with the object shadow expected.
 2. Feature level. Color statistics are collected from the image under the object region at the given pose hypothesis and compared with the statistics expected, by means of a suitable distance (e.g., the Bhattacharyya coefficient).
 3. Object level. When the pixel- or feature-level likelihood is represented by a kernel density, the mean-shift algorithm provides an MLE that finds the closest peak to the target location predicted in pose space.
- *Update model features.* The online color histogram is collected from the image, under the region at the updated pose, and used for the next frame estimation.

Figure 4.13 Color statistics modality.

1. The method described in [173] uses an adaptive Gaussian mixtures model in which the number of components per pixel is estimated and adapted online, as well as the mixture parameters.
2. The fast GPU implementation [69], based on the work of Mester et al. [118], uses a fixed background model (i.e., not adapted over time), obtained from a single reference image.

Both methods can be considered state of the art, and turn out to be very effective for object detection and tracking, in particular the second, in the absence of significant light variations, such as indoor environments under artificial lighting. Figure 4.14 shows an example of the GPU-based procedure, where we notice how light reflections on the metal parts, such as the path of a toy car, may give false positives; however, the main object blob is segmented correctly. Figure 4.15 outlines the steps for a color-based implementation.

As mentioned earlier, there are many other ways to employ information from the background of a static camera; in fact, any visual modality may implement its own background subtraction, which consists in the abstract procedure:

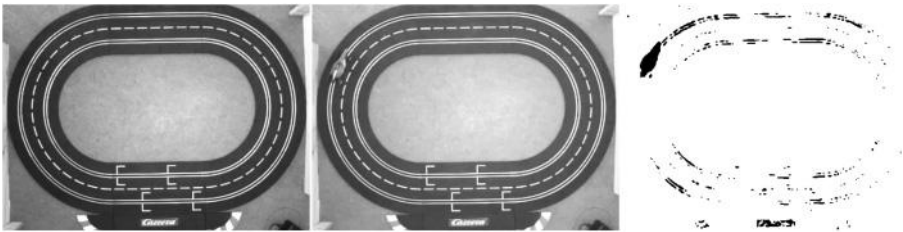


Figure 4.14 GPU-based foreground segmentation. *Left*: background reference image; *middle*: camera frame; *right*: segmentation result. (From [69].)

- *Preprocessing*. Perform color conversion, retaining only the relevant image channels.
- *Sampling model features*. A reference pixel-wise statistics is built off-line from a reference image or sequence of images.
- *Matching*
 1. Pixel level. Image pixels are compared against the model and classified as foreground or background according to their likelihood.
 2. Feature level (none, in the color-based modality)
- *Update model features*. Background statistics can be updated online from image data, with a slower time constant with respect to the average object dynamics, possibly avoiding known foreground regions, under the targets estimated.

Figure 4.15 Background subtraction using color statistics.



Figure 4.16 Subtraction of background edges using the Canny detector. *Left to right:* background model; background edges; current image; remaining edges after background subtraction.

- *Off-line.* From the background images, learn relevant features such as edges and keypoints.
- *Online.* Match them to the current image data; either remove them from the preprocessed data before object matching or include them as a “negative likelihood” term.

Concerning the matching procedure, the former choice corresponds, for example, to the color pixel subtraction or to the edge subtraction shown in Fig. 4.16. The latter consists of keeping the background features and computing, instead, the *likelihood ratio*,

$$L = \frac{P(Z^{\text{obj}}|s)}{P(Z^{\text{bg}}|s)} \quad (4.36)$$

where Z^{obj} and Z^{bg} are, respectively, the associated object and background features under the hypothesis s . In this way, a hypothesis that also matches background features or pixels is penalized accordingly.

Finally, we note how two simultaneous background models can be kept for more robustness: the *off-line* or *static model*, built from prior images, and the *online* or *dynamic model*, which is kept up to date with the current images, after removing data in areas of known objects. In fact, the former may be complete and stable but not up to date, whereas the latter is more recent but less complete, since areas covered by tracked objects must be excluded, or data from nontracked objects may be included erroneously. Once again, fusion of these two models (e.g., by weighted average of residuals) can achieve the benefits of each.

4.3 BLOBS

A distinctive feature often used to localize an object is a descriptor of the *shape* of the region underlying its camera projection, referred to as the *silhouette*, *shadow*, or *blob*. In general, this region has a highly variable shape, possibly consisting of disconnected components, and can be characterized through

global properties such as the area or the perimeter of its boundary, or through more detailed properties such as the location of corners or holes, or high curvature values, up to the entire region itself.

We can further split blob-based methods into two classes: blob detection or local optimization. The former consists of detecting all blobs in the current image and matching them using one of the descriptors hereafter presented, either with the re-projected object shadows or with the previous image blobs, in either case by looking for the nearest neighbors (as in Fig. 3.9). The latter, instead, consists of a continuous adaptation of the shape to the current image data, by means of local optimization of a cost function, which may be initialized as well from the re-projected model blob or from the blob of the previous image; due to the high degree of freedom of the shape, these approaches usually have a *variational* formulation.

If matching is performed frame to frame, the resulting shapes can be used subsequently to recover its pose parameters: for example, by means of a silhouette *back-projection* in world space [155] or a maximum-likelihood optimization that minimizes the re-projection error. In any case, using multiple views may be crucial, to solve the localization ambiguity given by the shadow of a nonconvex three-dimensional object.

4.3.1 Shape Descriptors

As we have seen, the goal of image segmentation is to identify connected regions that most probably belong to a single object or to a physically distinguishable part of an object, because they show relatively uniform visual properties such as uniform color, texture, or motion field. In particular, the output of image segmentation is a binary or n -ary classification that assigns a given pixel to the objects of interest or to the background, possibly based on prior models of the object and/or the background. This map can be refined through *morphological* operations such as erosion or dilation, used to remove noise, and finally, to detect clusters of connected pixels, or blobs.

Figure 4.17 shows the result of blob detection on a foreground map, followed by further *blob grouping* that connects blobs belonging to the same target, in this case a human silhouette, on the basis of prior shape attributes such as the expected size and height of the region.² A general blob-based target detector is described in more detail in Section 6.1; here we concentrate more on feature description and a matching procedure.

Blobs that have been detected can be matched to the model silhouette expected at the pose predicted by defining suitable shape descriptors and the likelihood function with single or multiple association hypotheses. In the

²Although not strictly necessary, preclustering of features may result in improved tracking; in this context, pose prediction can also be used to guide the clustering procedure, which may also be called *dynamic* clustering.

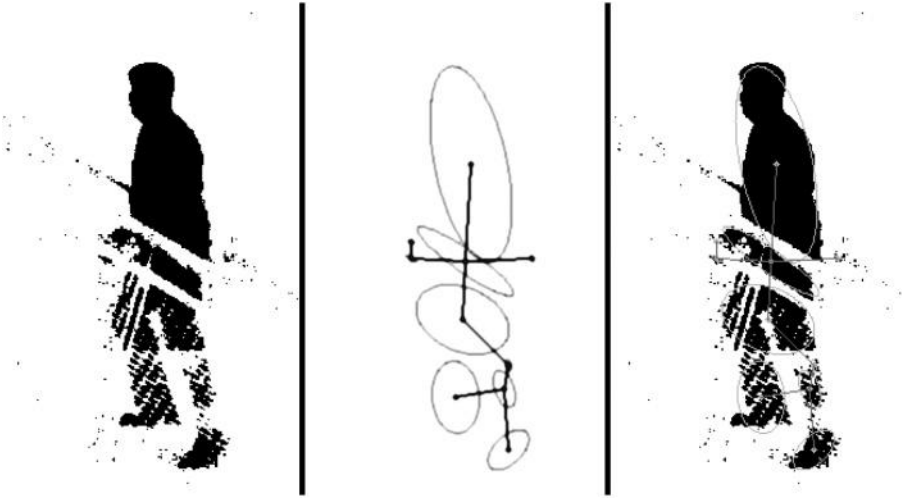


Figure 4.17 Blobs detected after background subtraction and clustered using the minimum spanning tree approach. (From [98].)

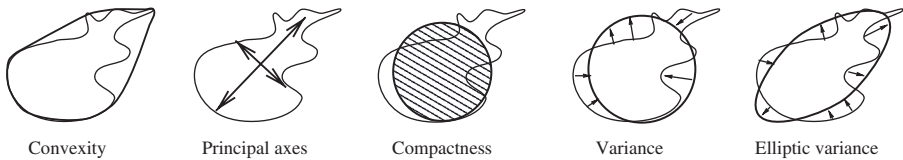


Figure 4.18 Simple shape descriptors. (From [141].)

literature, many examples of shape description and matching are known, usually classified as *contour-* and *region-based*.

Simple descriptors encode properties of the spatial distribution of the silhouette; they may be given by geometric properties (Fig. 4.18) such as the following [141,146]:

- *Area*: the number of pixels covered by the region.
- *Perimeter*: the number of pixels of the boundary.
- *Orientation*: the direction of the principal axis (i.e., the main eigenvector of the *covariance matrix*).
- *Compactness (or circularity)*: how “closely packed” the shape is (i.e., the ratio between the square perimeter and the area). Its minimum value is obtained by a circle, equal to 4π .
- *Eccentricity*: the ratio between the length of the two principal axes (i.e., the ratio of the covariance eigenvalues), which is also minimal for a circle.

- *Rectangularity*: the ratio between the area of the object and the area of its minimum bounding box (possibly rotated), which reveals how close the object is to a rectangle.
- *Convexity*: the ratio between the perimeter of the object and the perimeter of its *convex hull* or, equivalently, between the two areas.
- *Ellipticity (or elliptic variance)*: the similarity of the shape with the closest ellipse, in terms of the average distance between the respective contour points.

A more general descriptor is given by *moments*, which decompose the overall region, seen as a two-dimensional piecewise constant function, into an infinite series of elementary functions that can be used to reconstruct the shape up to a certain level of detail given by the number of coefficients retained. In principle, one would require infinite terms for encoding the shape completely. In practice, only low-order terms are retained; higher-order terms are more influenced by noise and are therefore less reliable for matching.

This decomposition can be carried out according to purely mathematical (such as Taylor or Fourier series) or more biologically inspired criteria, producing descriptors with different properties: robustness to noise and model imperfections, generality of application with respect to the shape, viewpoint invariance, and computational complexity. These coefficients can be put into a global descriptor, possibly with different geometric *invariance* properties such as invariance to roto-translation or scale. As a general rule, invariant moments are desirable for detection when the actual pose is completely unknown, but less desirable for tracking, where a close prediction is already available. In fact, enforcing too much invariance leads to false positives: for example, the “ellipticity” property defined above is invariant to anisotropic scale, and therefore any ellipse is matched to a circle.

A well-known descriptor dates back to the vision-inspired *Hu moments* [81], which are scale-, rotation-, and translation-invariant. For this purpose, we first define the *raw moments* of order $p + q$ of a region $I(x, y)$ by

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q I(x, y) dx dy \quad (4.37)$$

where in the binary case, $I(x, y) = 1$ inside the region and 0 elsewhere; for a discrete image, M_{pq} is computed by

$$M_{pq} = \sum_x \sum_y x^p y^q I(x, y) \quad (4.38)$$

It can be shown that if $I(x, y)$ is piecewise continuous and the support region has a finite extension, we can compute M_{pq} for any order, and the set of all moments uniquely defines $I(x, y)$. In particular, the *area* of the region is given by M_{00} , and the *centroid* is $(\bar{x}, \bar{y}) = (M_{10}/M_{00}, M_{01}/M_{00})$.

Afterward, we can define the *central moments*

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y) dx dy \quad (4.39)$$

which can also be computed directly from the raw moments, using linear formulas. These moments are invariant to translation but not to rotation and scale. Next, we compute the *scale-invariant* moments,

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\left(1 + \frac{p+q}{2}\right)}} \quad (4.40)$$

and finally, we define the *Hu moments*:

$$\begin{aligned} I_1 &= \eta_{20} + \eta_{02} \\ I_2 &= (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\ I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\ I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\ I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ &\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ I_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\ I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ &\quad + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \end{aligned} \quad (4.41)$$

which are also invariant to rotation.

Zernike moments are another similarity-invariant, generally more robust representation, based on Taylor expansion of complex functions and therefore not biologically inspired. Their use for shape description and matching has been described in the literature [93,94]. In polar coordinates (ρ, θ) , the Zernike radial polynomials of order p are defined as

$$R_{p,q}(\rho) = \sum_{s=0}^{(p-|q|)/2} (-1)^s \frac{(p-s)!}{s!((p+|q|)/2-s)!((p-|q|)/2-s)!} \rho^{p-2s} \quad (4.42)$$

with the constraints

$$\begin{aligned} p &\geq 0 \\ q &> 0 \\ p - |q| &\text{ is even} \\ |q| &\leq p \end{aligned} \quad (4.43)$$

The Zernike complex basis functions, defined over the unit disk, are given by

$$V_{p,q}(\rho, \theta) = R_{p,q}(\rho) \exp(jq\theta), \quad \rho \leq 1 \quad (4.44)$$

and finally, the Zernike moments are defined by

$$Z_{p,q} = \frac{p+1}{\pi} \iint_{\text{unit disk}} V_{p,q}^*(\rho, \theta) I(\rho, \theta) \quad (4.45)$$

where $V_{p,q}^*$ is the complex conjugate of $V_{p,q}$.

The magnitudes of $Z_{p,q}$ are invariant to rotation [93], robust to noise and small shape variations, nonredundant because the basis vectors are orthogonal, and *multiresolution*, because the low-order moments represent the global shape of the pattern while the higher-order moments represent the detail. This leads to a very effective description of the content of the blob region compared to the moments described previously.

The choice of the maximum order p determines the specificity of the descriptor $z = (Z_{0,0}, \dots, Z_{p,q})$ as well as the overall number of components. In particular, the number of indices q available for a given p , $Z_{p,q}$, increases with p , as shown in Table 4.1.

Matching can be performed by direct comparison of two descriptors: for example, in the weighted L_1 norm

$$e = \sum_i w_i |h_i - z_i| \quad (4.46)$$

where h and z are the descriptors expected and observed, respectively, and w_i are weights that reflect the variance of each moment with respect to the

TABLE 4.1 The 36 Zernike Moments up to Order 10

p	Moments	Number of Moments
0	$Z_{0,0}$	1
1	$Z_{1,1}$	1
2	$Z_{2,0}, Z_{2,2}$	2
3	$Z_{3,1}, Z_{3,3}$	2
4	$Z_{4,0}, Z_{4,2}, Z_{4,4}$	3
5	$Z_{5,1}, Z_{5,3}, Z_{5,5}$	3
6	$Z_{6,0}, Z_{6,2}, Z_{6,4}, Z_{6,6}$	4
7	$Z_{7,1}, Z_{7,3}, Z_{7,5}, Z_{7,7}$	4
8	$Z_{8,0}, Z_{8,2}, Z_{8,4}, Z_{8,6}, Z_{8,8}$	5
9	$Z_{9,1}, Z_{9,3}, Z_{9,5}, Z_{9,7}, Z_{9,9}$	5
10	$Z_{10,0}, Z_{10,2}, Z_{10,4}, Z_{10,6}, Z_{10,8}, Z_{10,10}$	6

Source: Data from [94].



Figure 4.19 Matching shapes with Zernike moments. The shape of a bat undergoing variations due to the viewpoint (*left*) is matched successfully to the top-left shape on the right. (From [94].)

variability of the shape. The sum is carried out over all components of the descriptor (e.g., 36) if the maximum order is $p = 10$.

Figure 4.19 shows an example of blob matching using Zernike moments. The shape of a bat, as seen from different viewpoints, undergoes complex transformations, including scale, rotation, and perspective distortions, yet it is matched successfully to its reference model in the database. Another way to characterize the blob shape is by using the boundary line given by a sequence of N pixels $(y_1(u), y_2(u))$, with $u = 1, \dots, N$ the curvilinear coordinate.

Similar to the situation in a region descriptor, the boundary can also be encoded into a set of scalar values, defining geometric properties such as local curvatures, corners, and arc lengths. Among them we find a robust, multiresolution approach known as the *curvature scale space*, proposed by Mokhtarian and Mackworth [124] and applied in more recent work (e.g., [122,123,125]). In this case, the descriptor is obtained by smoothing the contour at multiple scales (Fig. 4.20) and finding the local maxima of curvature zero crossings (Fig. 4.21),

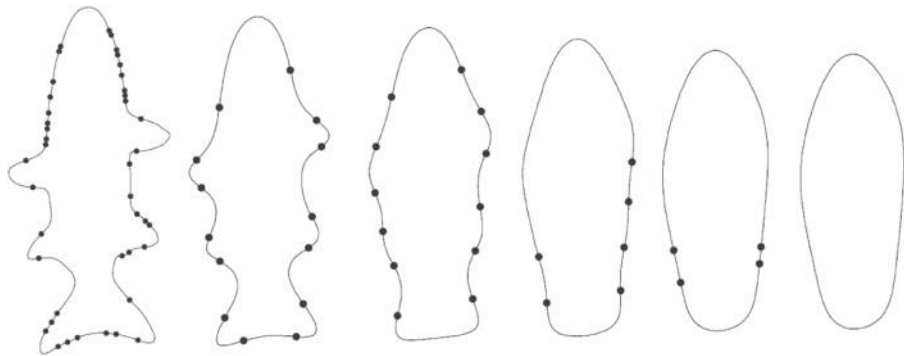


Figure 4.20 Smoothing the curvature of the object to find curvature zero crossings at multiple resolutions. (From [123].)

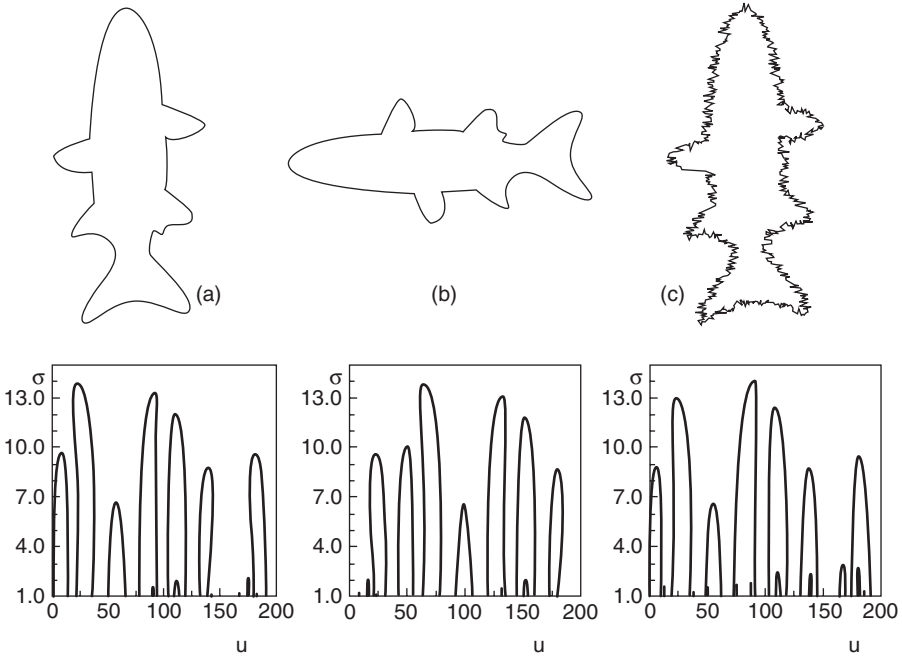


Figure 4.21 The curvature scale-space representation is rotation invariant and robust to noise. (From [123].)

which are invariant to scale, rotation, and translation and relatively robust to noise.

In particular, at each scale σ we find a different set of points that with increasing σ will merge pairwise into one and then disappear; the merge points, shown in Fig. 4.21, provide the CSS descriptor:

$$\text{CSS} = \{(2213.8), (9113.3), (10,912.0), (89.7), (1819.6), (1398.8), (576.7)\} \quad (4.47)$$

given in terms of the curvilinear parameter $(u_{i_1}, \dots, u_{i_M})$ at the respective scales.

Before matching, the u_i coordinates are normalized to 1 by dividing them by N , which provides invariance to scale, and are sorted according to their σ values; one more step in achieving orientation invariance is to *align* the CSS of the two shapes to be matched, since a change in orientation causes a *shift* of the CSS. The reason for sorting the peaks according to scale is that only peaks at higher σ are reliable for matching; the high-resolution peaks may arise because of noise.

A summary of blob matching by detection is given in Fig. 4.22.

- *Preprocessing.* Detect connected components from a pre-segmented image, and encode them into a proper descriptor based on region or contour properties.
- *Sampling model features* (not necessary, since the blob expected must be re-computed each time for matching)
- *Matching*
 1. Pixel level (not present, since blobs are detected at the feature level)
 2. Feature level. Match the current blobs either with the re-projected object shape or with the previous image blobs.
- *Update model features* (none)

Figure 4.22 Blob matching by detection.

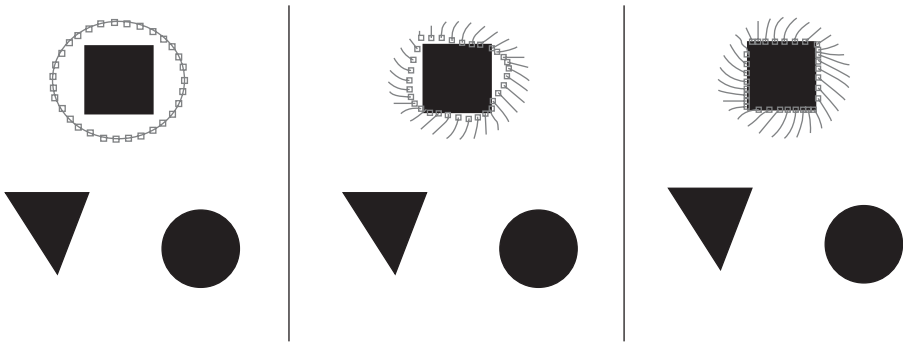


Figure 4.23 Nonparametric contour fitting with *snakes*. *Left*: initialization; *right*: convergence result.

4.3.2 Blob Matching Using Variational Approaches

In Section 4.3.1 we dealt with blob matching by means of global shape properties of the region or its contour, extracted after image segmentation and clustering. Another class of methods consists of tracking the region directly in the gray-level image by means of local optimization of a cost function defined over all possible shapes and initialized by a prior contour.

If the initialization is provided by the previous frame contour, this is another example of feature-level tracking (Section 3.3.2), which we may also call *contour flow*, in analogy to the local, pointwise optical flow of Section 4.6.2. Contour optimization is a *variational* problem in which the goal is to locate a function \mathcal{C} with virtually infinite degrees of freedom.

In this context, we first consider the classical *snakes* approach [91], which uses a generic deformable contour, defined as a two-dimensional parametric curve $\mathbf{y}(u)$ with $u \in [0, 1]$, and define the contour residual as a general *energy functional* to be minimized (Fig. 4.23):

$$E_{\text{snake}} = \int_0^1 [E_{\text{int}}(\mathbf{y}(u)) + E_{\text{img}}(\mathbf{y}(u)) + E_{\text{ext}}(\mathbf{y}(u))] du \quad (4.48)$$

which consists of three terms: an internal regularization term E_{int} , favoring smooth contours without corners or discontinuities; a feature-related term E_{img} , which leads toward image features (e.g., local gradients); and an external force E_{ext} , which may lead toward the prior curve \mathbf{y}_0 or push away from known obstacles.

The internal model energy, in turn, consists of first- and second-order derivatives of the parametrized curve,

$$E_{\text{int}}(\mathbf{y}(u)) = \alpha \left| \frac{\partial \mathbf{y}}{\partial u} \right|^2 + \beta \left| \frac{\partial^2 \mathbf{y}}{\partial u^2} \right|^2 \quad (4.49)$$

where α enforces a short curve and β enforces a low curvature at each point. The image energy is usually defined by intensity gradients, which may be convolved with a Gaussian for a larger convergence area:

$$E_{\text{img}}(\mathbf{y}) = -\|G * \nabla I(\mathbf{y})\|^2 \quad (4.50)$$

where $*$ is the convolution operator and the negative sign is used to *maximize* gradient; or equivalently, by zeroing the absolute value of Laplacian, again convolved with a Gaussian:

$$E_{\text{img}}(\mathbf{y}) = \|G * \nabla^2 I(\mathbf{y})\|^2 \quad (4.51)$$

Finally, the external energy can be given by the distance to the corresponding point on a prior curve, \mathbf{y}_0 :

$$E_{\text{ext}}(\mathbf{y}(u)) = \|\mathbf{y}(u) - \mathbf{y}_0(u)\|^2 \quad (4.52)$$

By considering a discrete set of points \mathbf{y}_i along the curve, eq. (4.48) can be rewritten in the discrete form

$$E_{\text{snake}} \approx \sum_{i=1}^N E_{\text{int}}(\mathbf{y}_i) + E_{\text{img}}(\mathbf{y}_i) + E_{\text{ext}}(\mathbf{y}_i) \quad (4.53)$$

where the internal energy is approximated by finite differences,

$$E_{\text{int}}(\mathbf{y}_i) = \alpha \frac{|\mathbf{y}_i - \mathbf{y}_{i-1}|^2}{\Delta u^2} + \beta \frac{|\mathbf{y}_{i+1} + \mathbf{y}_{i-1} - 2\mathbf{y}_i|^2}{\Delta u^4} \quad (4.54)$$

and Δu is a fixed sampling interval for the curvilinear coordinate.

As already emphasized, this formulation implies a shape-free approach to tracking single-contour objects, where the only “prior” is given by regularization terms such as E_{img} and E_{ext} but not from a predefined shape. In the

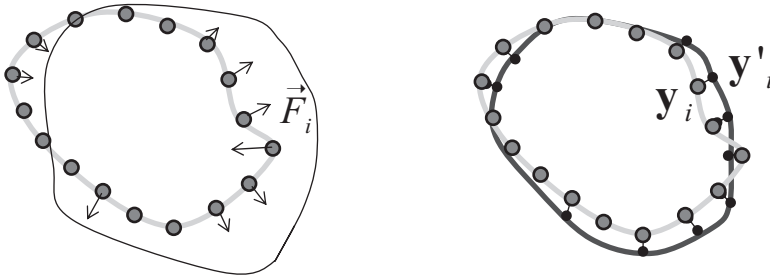


Figure 4.24 Gradient descent for snakes evolution. *Left*: force field; *right*: a step of gradient descent.

discrete formulation, energy optimization can be performed by gradient descent if we define the *pose* of the contour as $p_{\text{snake}} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$, which is a $2N$ -dimensional vector of free parameters. We also notice here the difference between this approach and the *active shape* model of Section 2.2.1. For active shapes, the contour model is constrained to a reduced set of degrees of freedom, providing the deformation *modes* of a prior shape, whereas here the curve is freely estimated, with additional regularization terms.

In particular, the gradient is given by a *force field* applied to each contour point (Fig. 4.24, left) where each term \mathbf{f}_i drives the contour point independently to its destination on the image, according to the derivative of the three terms; we also notice how the high-curvature point on the right side is pulled back, since the internal energy prevails over the image energy there. We can therefore update the contour points according to

$$\mathbf{y}'_i = \mathbf{y}_i + k\mathbf{f}_i, \quad i = 1, \dots, N \quad (4.55)$$

and stop at a local minima of E_{snake} , when $\mathbf{f}_i \approx 0, \forall i$. More complex and robust optimization approaches can be found in the literature, such as dynamic programming [28], with better convergence properties.

However, as pointed out by Cremers et al. [57], the snakes formulation has major drawbacks, including a complex contour representation that often requires *re-parametrization* to avoid overlapping points during evolution of the contour, as well as not easily being capable of handling topological changes in the curve: for example, if the blob splits into disconnected regions, or two regions merge into one. Moreover, due to the use of local gradients, many local minima cause the optimization to fail when the initial hypothesis is not sufficiently close to the actual contour.

A more robust and flexible approach to contour tracking is provided by *level sets*. This approach is somehow related to the approach described above, but deals with the contour evolution problem in a different way, by considering the contour \mathcal{C} as the *level set* of a continuous function ϕ :

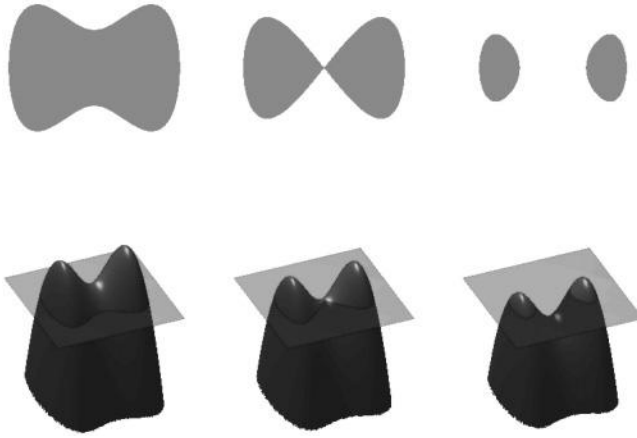


Figure 4.25 Representation of the level sets method. (From [13].)

$\mathcal{C} = \{(y_1, y_2) : \phi(y_1, y_2) = 0\}$, so that its temporal evolution is obtained by considering ϕ as a function of *time*, $\mathcal{C}_t = \{(y_1, y_2) : \phi(y_1, y_2, t) = 0\}$. Figure 4.25 illustrates this idea; we can also see that if the topology of the curve changes (e.g., by splitting), this is handled much more easily by moving the function ϕ in space rather than moving and re-parametrizing the contour.

The main difference between the two approaches can also be related to Fig. 3.4. The snakes algorithm is a *Lagrangian* approach, where control points on the contour are *tracked* from frame to frame, whereas level sets are a *Eulerian* approach, which recomputes the entire function ϕ pixel-wise, afterward localizing the contour as a “by-product” given by the pixels where $\phi = 0$, whereas the blob is defined by $\phi > 0$ (or $\phi < 0$).

To formulate the problem, we first reconsider the evolution of a contour \mathcal{C}_t in the variational framework of snakes:

$$\frac{\partial \mathcal{C}}{\partial t} = -\frac{\partial E(\mathcal{C})}{\partial \mathcal{C}} = v\mathbf{n} \quad (4.56)$$

where $E(\mathcal{C})$ is the energy defined by eq. (4.48), whose variation with respect to the contour gives the force (i.e., the *speed* v of each point along the respective normal \mathbf{n}). Then the corresponding level-set equation is obtained by considering the related function $\phi(\mathbf{y}, t)$, originating at the contour \mathcal{C}_t , whose total temporal derivative at any contour point and any time must be zero:

$$\frac{d}{dt} \phi(\mathcal{C}_t, t) = \nabla \phi \cdot \frac{\partial \mathcal{C}}{\partial t} + \frac{\partial \phi}{\partial t} = 0 \quad (4.57)$$

and therefore, letting $\mathbf{n} = \nabla \phi / \|\nabla \phi\|$, we have

$$\frac{\partial \phi}{\partial t} = -\|\nabla \phi\| v(\phi) \quad (4.58)$$

Equation (4.58) is a particular form of *Hamilton–Jacobi* partial differential equation (PDE), and can be solved by numerical methods. However, this equation is only specified along the contour, and therefore any numerical implementation requires that the right-hand side be extended to some points away from the contour.

In the original formulation of Osher and Sethian [129], the regularization term in the velocity was chosen to be $v = v_0 + \varepsilon \kappa(\phi)$, where κ is the mean curvature of the level set, given by

$$\kappa(\phi(\mathbf{y})) = \operatorname{div} \left(\frac{\nabla \phi}{\|\nabla \phi\|} \right) = \frac{\phi_{y_1 y_1} \phi_{y_2}^2 - 2\phi_{y_1} \phi_{y_2} \phi_{y_1 y_2} + \phi_{y_2 y_2} \phi_{y_1}^2}{(\phi_{y_1}^2 + \phi_{y_2}^2)^{3/2}} \quad (4.59)$$

where $\phi_{y_1 y_2}$ are second-order partial derivatives of ϕ . The role of κ is the same as that of the internal energy for snakes, controlling the smoothness of the curve; the constant term v_0 , although not strictly necessary, is used to improve convergence speed and robustness, also with a proper choice for the ε coefficient.

To add the image-related term, the *geometric level sets* approach [112] substitutes for the image gradient, $\|\nabla I\|$, a more general *stopping function*, $g(I)$, which is positive and decreasing with the gradient:

$$g(I) = \frac{1}{1 + \|\nabla I\|^2} \quad (4.60)$$

and therefore

$$\frac{\partial \phi}{\partial t} = -g(I(\mathbf{y}))(\kappa(\phi) + v_0)\|\nabla \phi(\mathbf{y})\| \quad (4.61)$$

In most cases, ϕ is initialized by the signed *distance transform* of the initial blob, which for each pixel provides the distance from the nearest contour point, with a negative sign inside the contour and a positive sign outside (Fig. 4.26).

Another way to derive a level sets equation is given by formulating the variational problem of minimizing $E(C)$ into another variational problem $E(\phi)$, with respect to the level sets function ϕ . This approach leads to the Euler–Lagrange equation,

$$\frac{\partial \phi}{\partial t} = -\frac{\partial E(\phi)}{\partial \phi} \quad (4.62)$$

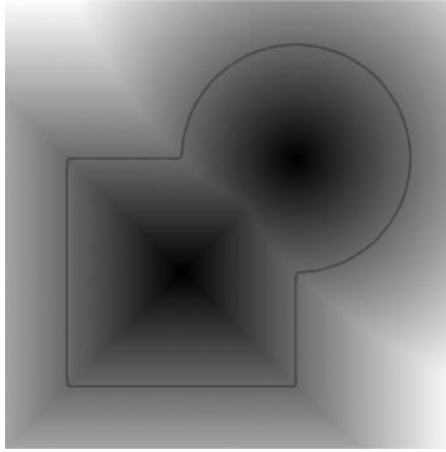


Figure 4.26 Level sets function: signed distance transform from the contour hypothesis. (From [14].)

where the definition of $E(\phi)$ is not unique and therefore leads to different evolution equations. In particular, in the *geodesic* active contours [49], ϕ is evolved according to the following rule:

$$\frac{\partial \phi}{\partial t} = \|\nabla \phi\| \operatorname{div} \left(g(I) \frac{\nabla \phi}{\|\nabla \phi\|} \right) = g(I) \|\nabla \phi\| \kappa(\phi) + \nabla g(I) \cdot \nabla \phi \quad (4.63)$$

which is the same as eq. (4.61), with the additional term $\nabla g(I) \cdot \nabla \phi$. The name *geodesic* has been adopted because the underlying energy function can also be seen as the length of the contour in a *Riemannian* metric space, induced by the image intensity.

So far, the energy function has been defined on the basis of the blob *boundary*, localized into high-intensity gradients, exhibiting convergence problems similar to those of the traditional snakes formulation. Another way to formulate the cost function is based, instead, on *region* properties, where the energy function is defined in the interior and exterior areas with respect to the blob. These approaches attempt to provide an active image *segmentation*, where the properties of interest may be a combination of color, texture, motion, and others.

Following Cremers et al. [57], to define the problem we consider a partition of the image area Ω into a disjoint set of regions $\mathcal{P}(\Omega) = \{\Omega_1, \dots, \Omega_n\}$, where $\Omega = \cup_{i=1}^n \Omega_i$; in the special case of two regions (i.e., object and background), the problem is known as *two-phase segmentation*.

Then, for a given image I , an optimal partition \mathcal{P} can be obtained by maximizing the *a posteriori* probability $P(\mathcal{P}(\Omega)|I)$ by means of Bayes' rule:

$$P(\mathcal{P}(\Omega)|I) = P(I|\mathcal{P}(\Omega))P(\mathcal{P}(\Omega)) \quad (4.64)$$

where the prior term $P(\mathcal{P}(\Omega))$ can be defined in a general-purpose fashion: for example, as a minimizer of the contour length,

$$P(\mathcal{P}(\Omega)) \propto e^{-\nu|\mathcal{C}|}, \quad \nu > 0 \quad (4.65)$$

with $|\mathcal{C}|$ the total length of the partition boundary $\mathcal{C} \equiv \partial\Omega$. By assuming independence of statistics in the various regions, we can also decompose the likelihood into

$$P(I|\mathcal{P}(\Omega)) = \prod_{i=1}^n P(I|\Omega_i) \quad (4.66)$$

where each term expresses the probability of the image data I observed under the region Ω_i . These data are given by pixel-wise image properties, usually modeled as the outcome of an independent and identically distributed random process $P_i(I)$, so that the likelihood of region Ω_i is given by

$$P(I|\Omega_i) = \prod_{\mathbf{y} \in \Omega_i} (P_i(I(\mathbf{y})))^{d\mathbf{y}} \quad (4.67)$$

for all pixels in the region, and the exponent volume $d\mathbf{y}$ is meant to ensure the correct limit for $d\mathbf{y} \rightarrow 0$. Then the MAP estimation problem becomes an energy minimization problem if we take the negative logarithm of both sides in eq. (4.64):

$$E(\{\Omega_1, \dots, \Omega_n\}) = -\sum_i \int_{\Omega_i} \log P_i(I(\mathbf{y})) d\mathbf{y} + \nu|\mathcal{C}| \quad (4.68)$$

The subsequent step consists of specifying the pixel-wise likelihood P_i , which can be parametric or nonparametric. In particular, parametric models express the statistics via a predefined form: for example, a Gaussian with parameters θ_i :

$$E(\{\Omega_i, \theta_i\}_{i=1, \dots, n}) = -\sum_i \int_{\Omega_i} \log P(I(\mathbf{y})|\theta_i) d\mathbf{y} + \nu|\mathcal{C}| \quad (4.69)$$

In this case, the standard optimization approach consists of alternately minimizing E with respect to Ω_i or θ_i , while keeping the other variable fixed, until convergence.

The former problem, that of finding the optimal partition satisfying the expected statistics θ_i , amounts to minimizing

$$\hat{E}(\{\Omega_i\}) \equiv \min_{\{\theta_i\}} E(\{\Omega_i, \theta_i\}) = - \sum_i \int_{\Omega_i} \log P(I(\mathbf{y})|\hat{\theta}_i) d\mathbf{y} + \nu|C| \quad (4.70)$$

formulated in the variational context of level sets, with the corresponding Euler–Lagrange equations. The latter problem, that of estimating the optimal statistics $\hat{\theta}_i$ for a given partition, is solved by

$$\theta_i = \arg \min_{\theta} \left(- \int_{\Omega_i} \log P(I(\mathbf{y})|\theta) d\mathbf{y} \right) \quad (4.71)$$

corresponding to simple averaging operations.

The partition \mathcal{P} can easily be embedded in a level sets function ϕ : for example, in the two-phase approach [50], $\Omega_1 = \{\mathbf{y} : \phi(\mathbf{y}) \geq 0\}$ and $\Omega_2 = \{\mathbf{y} : \phi(\mathbf{y}) < 0\}$, so that the boundary $\partial\Omega$ is defined by $\phi(\mathbf{y}) = 0$. This can be expressed formally by means of the Heaviside function,

$$H(\phi) \equiv \begin{cases} 1, & \text{if } \phi > 0 \\ 0, & \text{else} \end{cases} \quad (4.72)$$

and the energy [eq. (4.69)] can be expressed by a single integral over the entire image:

$$E(\phi, \{\theta_i\}) = \int_{\Omega} -H(\phi) \log P(I|\theta_1) - (1 - H(\phi)) \log P(I|\theta_2) + \nu |\nabla H(\phi)| d\mathbf{y} \quad (4.73)$$

where the term $|\nabla H(\phi)|$ is nonzero only on the boundary $\phi = 0$. This minimization problem can be solved by gradient descent on ϕ , by means of

$$\frac{\partial \phi}{\partial t} = \delta(\phi) \left(\nu \operatorname{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) + \log \frac{P(I|\theta_1)}{P(I|\theta_2)} \right) \quad (4.74)$$

where $\delta(\phi)$ is the delta function (i.e., the derivative of H , which is nonzero only when $\phi = 0$); in practice, the discontinuous H is approximated by a smooth version H_ε , so that its derivative δ_ε is nonzero in a small interval, where the level sets function will be updated at each step t (a procedure called *narrow-band optimization* [26]).

Concerning the optimal parameters [eq. (4.71)], by adopting the Gaussian parametrization

$$P(I(\mathbf{y})|\mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-(I-\mu_i)^2/2\sigma_i^2}, \quad i = 1, 2 \quad (4.75)$$

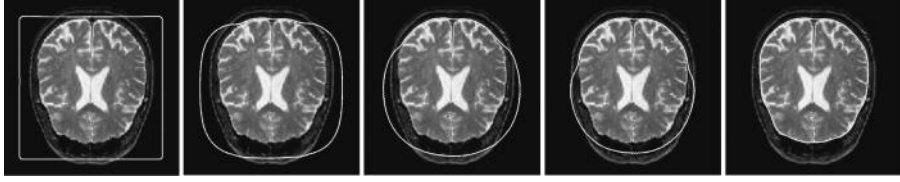


Figure 4.27 Level sets segmentation on a medical image. (From [14].)

- *Preprocessing.* Perform local contour optimization: for example, with snakes of level sets.
- *Sampling model features* (not defined; see Fig. 4.22)
- *Data association*
 1. Pixel level. The localized blob corresponds to a binary image that can be matched on the pixel level with the object shadow predicted.
 2. Feature level. Here the “features” would be the regions enclosed by the contours that can be matched by extracting global properties such as those of Section 4.3.1.
- *Update model features* (not defined; see Fig. 4.22)

Figure 4.28 Blob matching with variational approaches.

they are given simply by the sample mean and covariance over the region of interest Ω_i :

$$\begin{aligned}
 \mu_1 &= \frac{1}{a_1} \int H(\phi) I(\mathbf{y}) d\mathbf{y}, & \sigma_1^2 &= \frac{1}{a_1} \int H(\phi) (I(\mathbf{y}) - \mu_1)^2 d\mathbf{y} \\
 \mu_2 &= \frac{1}{a_2} \int (1 - H(\phi)) I(\mathbf{y}) d\mathbf{y}, & \sigma_2^2 &= \frac{1}{a_2} \int (1 - H(\phi)) (I(\mathbf{y}) - \mu_2)^2 d\mathbf{y}
 \end{aligned} \tag{4.76}$$

where $a_1 = \int H(\phi) d\mathbf{y}$ and $a_2 = \int (1 - H(\phi)) d\mathbf{y}$ are the areas of the two regions.

In Fig. 4.27 we can see a result of level sets segmentation applied to a medical image showing a slice of the human brain. The initial contour is arbitrary to a large extent. The main abstraction for variational blob matching is summarized in Fig. 4.28.

4.4 MODEL CONTOURS

Model contours are another widely used feature for tracking; for this purpose, many methods have been developed, and a good review has been provided by Blake and Isard [43]. Basically, object contours are distinctive lines that will probably be detected because they produce sharp optical transitions of inten-

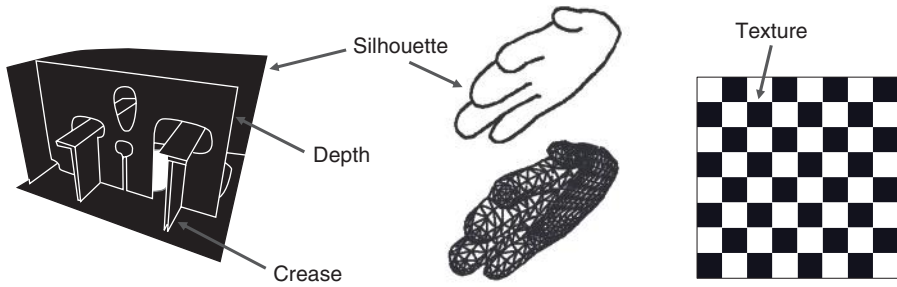


Figure 4.29 Candidate model contours for object detection.

sity, color, or texture from a given camera viewpoint. These transitions may arise as a consequence of specific properties of the model lines (Fig. 4.29):

- *Silhouette edges*: located on the “horizon” of the object surface
- *Texture edges*: located in correspondence to sharp variations in the reflectance properties
- *Crease edges*: located on discontinuities of the surface orientation
- *Depth edges*: located on depth discontinuities within the object region, caused by self-occlusion of nonconvex parts

All of the above can automatically be selected from any viewpoint and used to localize the object by matching with the corresponding image data. In this process, background clutter usually provides challenging situations, producing many additional edges that make data association highly ambiguous. Whenever possible, background edge subtraction may largely alleviate the problem.

Moreover, an image edge may also arise because of *false features* (Fig. 4.30) such as shadows or light spots, as well as depth discontinuities between two different objects in three-dimensional space. These features are not to be associated with any physical object, nor with the background.

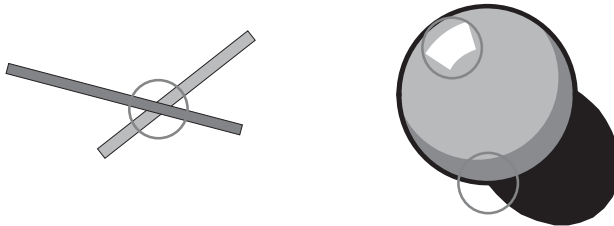


Figure 4.30 False features arising from projective and lighting effects.

Various strategies are used to track object contours, each constituting a visual modality of its own. In the following we show some examples of contour data association schemes.

4.4.1 Intensity Edges

On the image side, intensity edges can be defined as sharp *luminance* transitions observed along connected pixel lines. Such discontinuities can be detected using first- or second-order derivatives, respectively, looking for maxima or zero crossings, followed by more or less complex classification and grouping techniques.

Concerning the first type of detector, horizontal and vertical Sobel masks approximate the first x and y derivatives of a Gaussian kernel:

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad g_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (4.77)$$

which provide a smooth approximation to the image gradient; its magnitude and orientation are computed, respectively, by

$$\begin{aligned} m &= \sqrt{g_x^2 + g_y^2} \\ \theta &= \text{atan2}(g_y, g_x) \end{aligned} \quad (4.78)$$

The Canny detector looks for edges as local maxima of the gradient magnitudes. For this purpose it first applies a small Gaussian filter as a preprocessing step for noise removal, followed by Sobel filtering, nonmaxima suppression, and finally, *thinning*, to obtain 1-pixel-width edges (Fig. 4.31). This detector may produce open and disconnected edge lines, and works directly at fine resolution; therefore, it cannot be used in a multiscale context. Nevertheless, it produces a good map with precise edge localization, providing a good choice for its parameters (Fig. 4.32).

A traditional second-derivative edge detector is the Marr–Hildreth [115] technique, which connects sequences of image edges of 1-pixel width by looking for zero crossings of the Laplacian of Gaussian (LoG) response (Fig. 4.33):

$$\text{LoG}(x, y, \sigma) = -\frac{1}{\pi\sigma^4} \left(1 - \frac{x^2 + y^2}{2\sigma^2} \right) e^{-(x^2 + y^2)/2\sigma^2} \quad (4.79)$$

which is a more biologically motivated approach, related to the early visual processing occurring in the human retina [114]. By applying the LoG operator to the image I , we can compute it in two steps:

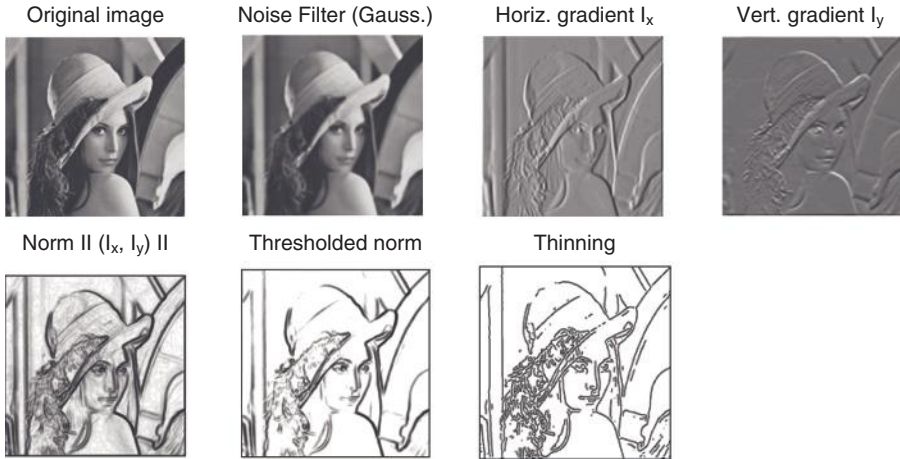


Figure 4.31 Processing steps of the Canny edge detector.

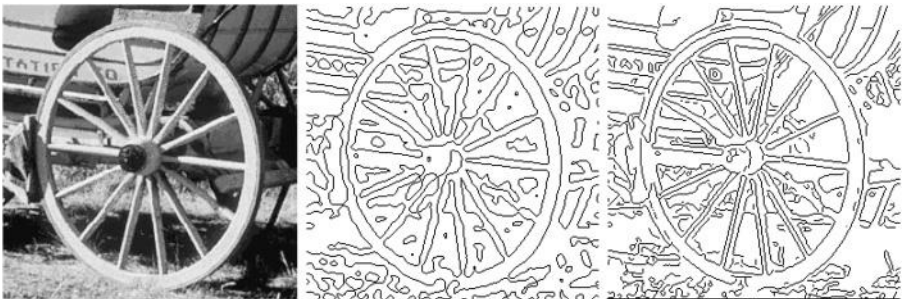


Figure 4.32 Comparison of the Marr–Hildreth edge detector with $\sigma = 2$ (*middle*) and Canny with $\sigma = 1$ (*right*).

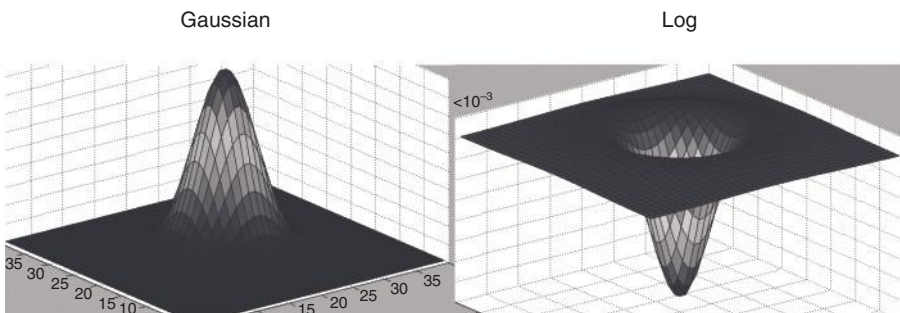


Figure 4.33 Two-dimensional Laplacian of Gaussian.

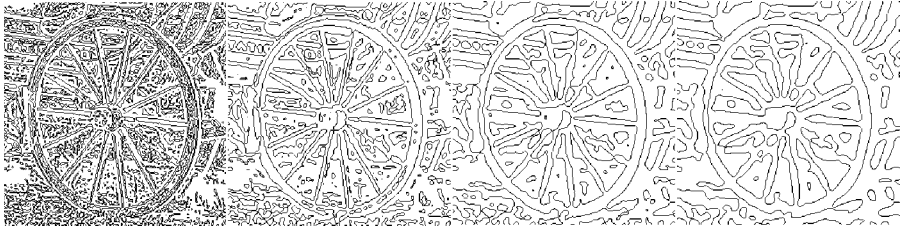


Figure 4.34 Marr–Hildreth edges, detected at different scales. *Left to right:* $\sigma = 1, 2, 3, 4$.

$$\text{LoG}(x, y, \sigma) = \nabla_{x,y}^2 (G(x, y, \sigma) * I(x, y)) \quad (4.80)$$

where $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$ is the Laplacian operator, $G(x, y)$ the two-dimensional Gaussian function, and $*$ the convolution operator. In the discrete case, both operations are performed with finite pixel *masks*, usually rounded off to integer values; the *separability* property of the Gaussian kernel $G(x, y) = G(x)G(y)$ for any σ allows very efficient implementation by means of two monodimensional filters, respectively, along the rows and columns of the image.

Unlike the Canny detector, this method provides connected and closed image contours located roughly on the boundaries of image structures at different *scales*, obtained by varying the size σ of the LoG kernel. In particular, at higher scales all small-sized blobs are increasingly suppressed (Fig. 4.34). On the other hand, because of the Gaussian smoothing, at coarse scales the object contours are also poorly localized; in particular, high-curvature edges such as corners are “rounded off”. Therefore, this method provides a multi-scale approach to edge detection and object tracking as well: By proceeding from coarse to fine resolution, we can obtain more robustness and larger convergence regions for both image-space and state-space search algorithms.

The standard output of both detectors is a scalar binary map where edge pixels are set to 1. However, the direction and magnitude of an intensity edge is also valuable information that should be retained for object tracking, because if the pose hypothesis is correct, the gradient onto a sharp edge should possess *normal* orientation to the projected contour as well as significant magnitude. In fact, many small blobs on the low-scale edge detection (Fig. 4.34, left) can be removed simply by thresholding their magnitude.

After image edges have been detected, the next step is to match them with the relevant model features; first we consider the case of individual model points sampled along the visible silhouette. In our abstraction, pixel-level data association with the edge map is carried out using the Chamfer *distance transform* [64]; the map on the right of Fig. 4.35 represents, for each pixel, the distance to the closest edge in the Canny map, which is model-free and therefore part of the preprocessing step.

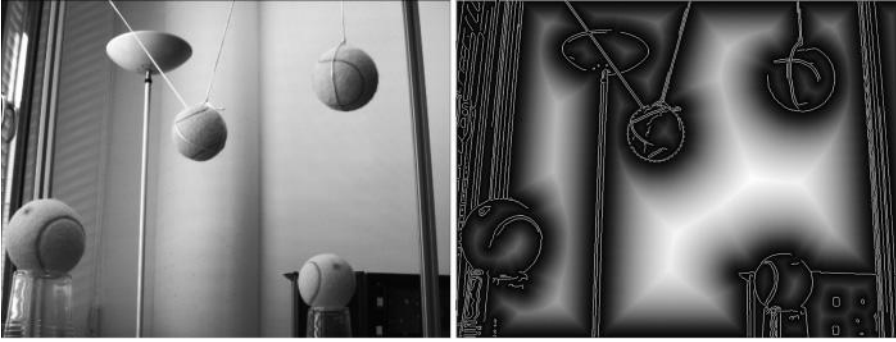


Figure 4.35 Chamfer distance transform on the Canny edge map. It can be used for pixel-level contour matching.

The likelihood of a pose hypothesis can be computed by drawing the projected object contours into another image, adding the squared values of the distance transform on the respective pixels, and afterward, normalizing by the number of pixels in order to obtain the SSD cost function. The best state hypothesis in Fig. 4.35, for a simple spherical model, projects onto a circle that matches the ball in the image that has the lowest average DT. In this context, multiresolution can be obtained by performing multiscale edge detection (e.g., with the Marr–Hildreth operator) before computing the distance transform.

As for the measurement expected, it is an ideal, noise-free, precise image of the object contours at the given pose hypothesis, which can be obtained by means of nonphotorealistic rendering (NPR) techniques and implemented efficiently on graphics hardware [99]. More complex methods can also be mentioned here. Figure 4.36 shows an example of NPR silhouette rendering with the *dual-space* method [78] for increasingly complex models.

At the feature level, edge association is obtained by sampling a uniform set of contour points at the pose predicted, back-projecting them in model space, and looking for corresponding image edges along the re-projected contour points and *normals*, up to a reasonable search distance d , which constitutes the validation gate [Fig. 4.37(a)]. To keep uniform properties of the matching process independent of the viewpoint or the shape complexity, we should also sample equispaced points in image space but not necessarily in object space, and for computational efficiency, keep the amount of points independent on the scale.

Overall, such a sampling procedure is a complex task that can, however, be implemented efficiently on graphics hardware, as mentioned in Section 3.2. Matching may result in zero, one, or multiple hypotheses [Fig. 4.37(b)]; this procedure has been used in a number of successful systems [61,76,84].

For each contour point, in the case of multiple associations, we assume that at most one is target generated and that the others must arise from clutter

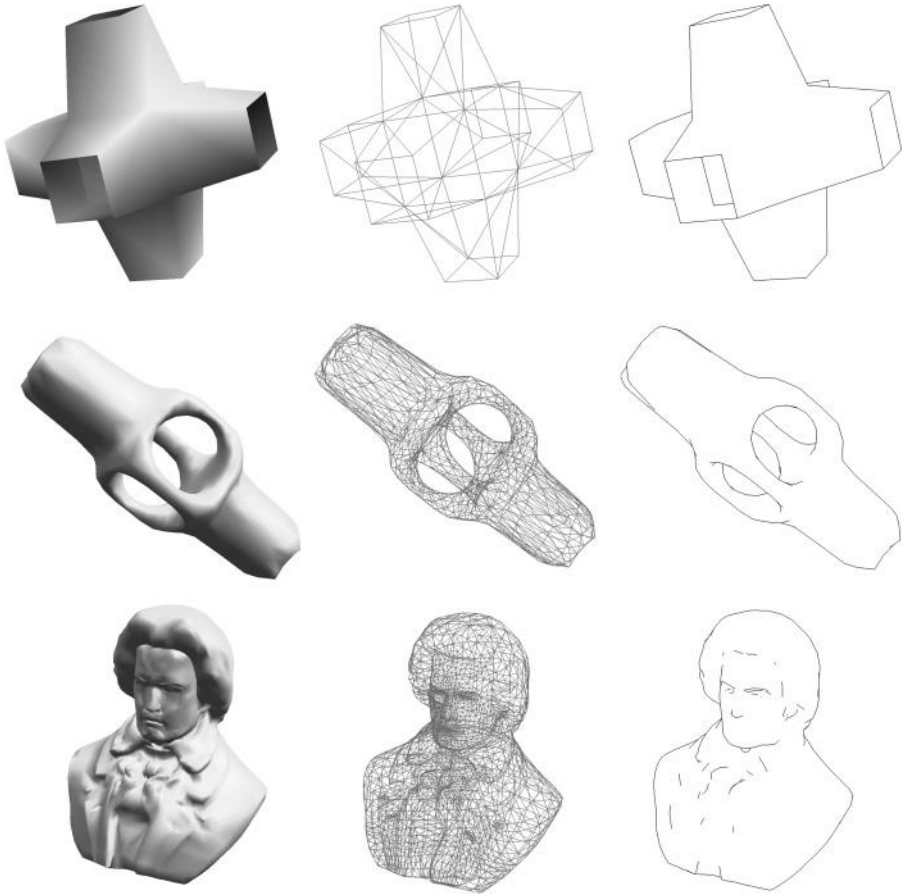


Figure 4.36 Silhouette rendering using the dual-space method. *Left:* rendered surface; *middle:* wireframe mesh; *right:* model contours. Models were rendered using silhouette detection software [25]. (From [78].)

background, including the case of missing detection (i.e., that all associated edges are clutter generated). Therefore, we have two schemes:

1. *Single-hypothesis matching.* Only one data item per feature is retained (i.e., the best, or the nearest neighbor); the others are discarded as false alarms. This gives a Gaussian likelihood that can be handled, for example, by an extended Kalman filter.
2. *Multihypothesis matching.* All of the hypotheses are retained, including the misdetection case occurring with probability α , and a multimodal likelihood must be used.

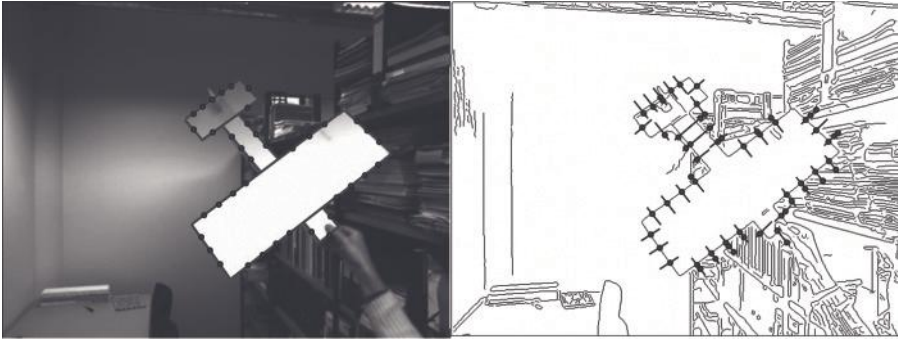
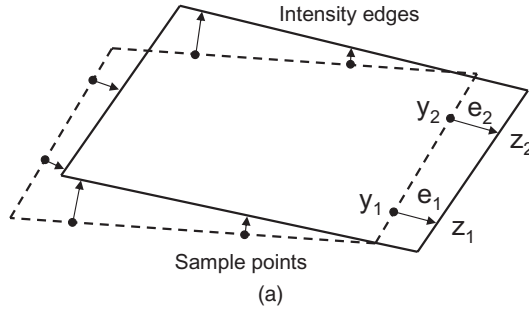


Figure 4.37 Feature-level contour matching. (a) Contour point matching along the normals. (b) *Left*: re-projected points under a given pose hypothesis; *right*: multiple matches, detected on the Canny edge map. (From [136].)

For the second case, a standard assumption is that false alarms are generated according to a Poisson process and that their location is distributed uniformly in feature space, whereas the target-originated measurement is modeled by the usual zero-mean Gaussian noise. These assumptions correspond to the multimodal likelihood defined in eq. (3.6). With a Gaussian likelihood, a maximum-likelihood estimation for obtaining object-level measurements corresponds to the standard Gauss–Newton method, also requiring first-order derivatives, as described in Section 5.2.

A summary of the intensity edges modality is given in Fig. 4.38.

4.4.2 Contour Lines

Another way of exploiting information from the edge map is given by *line features*. In this case, geometric primitives such as line segments or curves are detected and matched to the corresponding model primitives. Line features can be detected from the binary edge map in different ways. One class of methods performs a feature-space transform of the edge map, where line

- *Preprocessing.* Edge detection is performed on the gray-level image, and the preprocessed data are given in the form of a binary edge map, possibly also including gradient magnitudes and directions. For pixel-level matching, the chamfer distance transform is also computed.
- *Sampling model features.* For feature-level matching, object contours are sampled uniformly in image space at the average pose and that predicted and back-projected in object space.
- *Data association*
 1. Pixel level. Draw the expected edge map (or *sketch*) at the pose predicted with non-photorealistic rendering techniques. The contours predicted can be matched pixel-wise using the distance transform.
 2. Feature level. Sampled model points are re-projected and screen normals are computed in order to search and match the corresponding edges; this can give single- or multihypothesis likelihood.
 3. Object level. Perform a maximum-likelihood estimate by means of Gauss–Newton optimization, possibly with robust improvements.
- *Update model features* (none, because a complete CAD model already contains all of the edge features required).

Figure 4.38 Intensity edges modality.

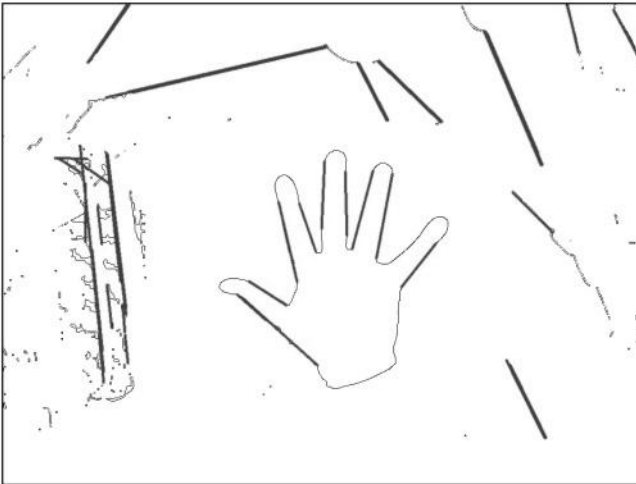


Figure 4.39 Line segments detected via the Hough transform. (From [132].)

detection takes place more easily; other methods attempt directly to *group* edge points into line primitives.

In the first case, the Hough transform [62] provides a histogram of likelihoods in the parameter space of line features, given by their direction and distance to the origin (θ, d), but possibly also including the endpoints, so that features can be detected as local maxima after the voting procedure (Fig. 4.39).

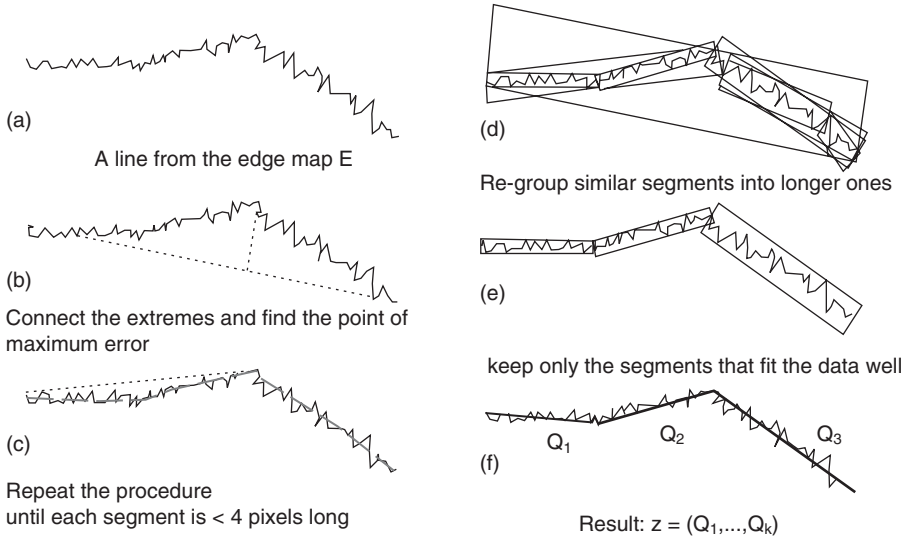


Figure 4.40 Incremental procedure for grouping edge points into connected segments. (Adapted from [104].)

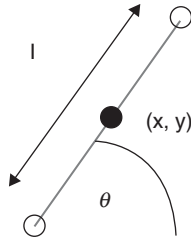


Figure 4.41 Simple descriptor for segment features.

An example of the second type is given by the method described in [104,107], where edge pixels are clustered into *segments* via an incremental least-squares fitting (Fig. 4.40), followed eventually by perceptual grouping criteria such as collinearity and parallelism.

After line primitives have been detected, corresponding model lines can be matched and residuals can be computed in feature space using a suitable error metric. In the case of segments, the feature can be parametrized by a simple descriptor,

$$\ell \equiv (x, y, \theta, l) \quad (4.81)$$

which contains information about the midpoint, orientation, and length (Fig. 4.41). A metric in this space can be defined [97] as the Mahalanobis distance between the descriptors,

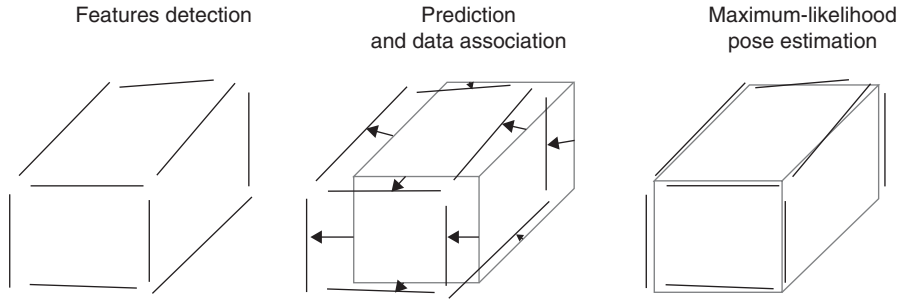


Figure 4.42 Line features matching and pose estimation via least-squares optimization.

- *Preprocessing.* Line detection is performed by means of one of the methods mentioned above, producing line or segment descriptors.
- *Sampling model features.* Visible contour lines are selected from the model silhouette rendered, at the pose predicted.
- *Data association*
 1. Pixel level (not defined, since preprocessed data are in feature space)
 2. Feature level. Lines observed in the neighborhood of the model are matched by using the metric (4.82).
 3. Object level. This is standard LSE, weighted by the R matrices.
- *Update model features* (none, for the same reasons as discussed for Fig. 4.38)

Figure 4.43 Contour lines modality.

$$d_R^2(\ell_1, \ell_2) \equiv (\ell_1 - \ell_2)^T R^{-1} (\ell_1 - \ell_2) \quad (4.82)$$

where the 4×4 residual covariance R is usually diagonal and gives a different level of confidence about the parameters; in fact, the length l usually has a higher uncertainty than the x and y values, while the direction θ is quite reliable.

The same residual metric is also used for data association (Fig. 4.42, middle) by setting the validation gate for each feature; for more robustness, it can eventually be adapted to reflect each feature strength and be updated over time. All of the association issues mentioned in the preceding section, such as missing detections and multiple hypotheses, apply here as well.

A summary of the contour lines modality is given in Fig. 4.43.

4.4.3 Local Color Statistics

The *contracting curve density* (CCD) algorithm [74,135] is another contour-based modality looking for optimal separation of *local color statistics* along

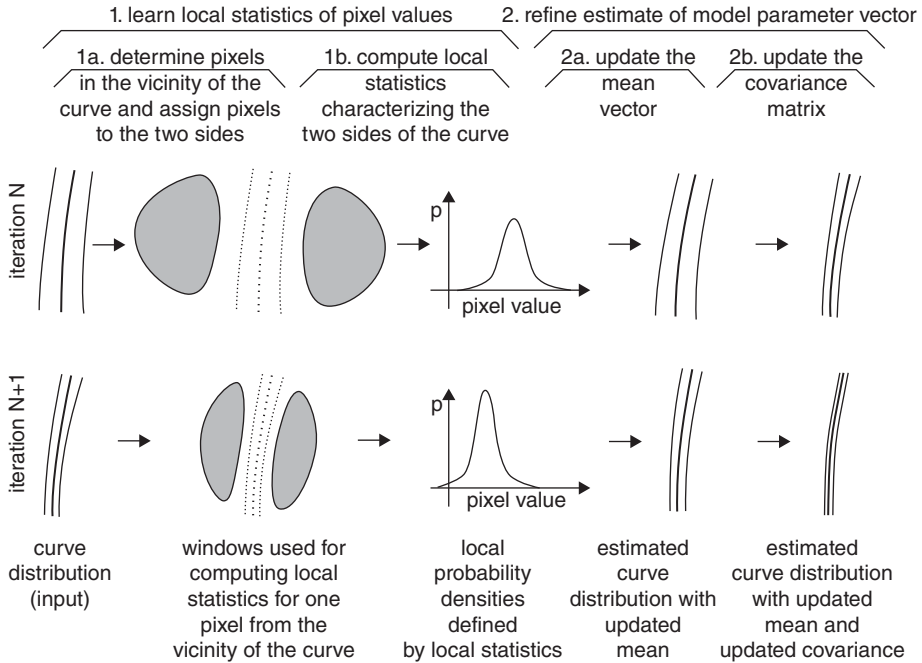


Figure 4.44 Contracting curve density (CCD) algorithm. (From [74].)

the contour. For object localization, separation takes place between the object and the background, across the projected model contours under a given pose hypothesis. For this purpose, a *color classification* likelihood is defined and measured through a set of data points sampled uniformly from the contour projected. Therefore, this modality shares the contour points sampler of Section 4.4.1.

The CCD algorithm basically iterates two steps (Fig. 4.44) for maximizing a monomodal separation likelihood, akin to expectation maximization [59]:

1. Compute local color statistics on the two sides of the contour by collecting pixels in local areas near each contour position.
2. Use the same points to carry out a Gauss–Newton pose update to maximize the classification likelihood according to the respective statistics.

During the two-step process, the resolution of local statistics, given basically by the area of pixels collected, decreases with exponential decay, thus providing a *multiresolution* approach, with improved robustness and precision. To describe the algorithm, we follow our *modality* abstraction directly and provide details about the main parts, in particular the feature-level matching.

In the CCD algorithm the preprocessing step is absent, since color data are used directly in the feature-level matching in order to collect local

statistics and optimize the pose. However, a color-space conversion can also be performed if it is statistically more convenient for the classification likelihood.

Sampling model features consists, as already mentioned, of collecting a uniform set of contour points and normals in the image. We notice that if data fusion is carried out using the contour points modality, the sampling procedure can be performed only once since the two classes will share the same re-projected points.

Feature-level matching involves the following substeps:

1. Collect color pixels along the normal to each contour point, on both sides.
2. Compute the online local statistics from these data.³
3. For the same pixels, compute the assignment errors, which are measurement residuals, to the respective side of the curve, as well as their Jacobians and covariance matrices.

Hereafter, we consider these three steps in more detail. In the original formulation [74], statistics were collected from weighted areas around each contour point, on either side (Fig. 4.44 left). To simplify the computation, as also suggested by Hanek and Beetz [74], we can first sample points along the respective *normals*, separately collect statistics, and afterward *blur* each one with its neighbors (Fig. 4.45). This is fully equivalent to the initial procedure, but computationally more convenient.

From each contour position \mathbf{y}_i , J foreground and background color pixels are collected along the normals \mathbf{n}_i up to a distance L , and second-order moments are estimated for the foreground and background sides (F and B), respectively):

$$\begin{aligned}
 v_i^{0,B/F} &= \sum_{j=1}^J w_{ij} \\
 v_i^{1,B/F} &= \sum_{j=1}^J w_{ij} I(\mathbf{y}_i \pm Ll_j \mathbf{n}_i) \\
 v_i^{2,B/F} &= \sum_{j=1}^J w_{ij} I(\mathbf{y}_i \pm Ll_j \mathbf{n}_i) I(\mathbf{y}_i \pm Ll_j \mathbf{n}_i)^T
 \end{aligned} \tag{4.83}$$

where $l_j \equiv j/J \in [1/J, 1]$ is the normalized contour distance and the \pm sign is referred to the respective side ($-$ for the background); image values I are

³If a textured model is available, *off-line* reference statistics at the pose predicted can also be built, using a rendered image of the object. These statistics can be combined with the online counterpart in the classification process.

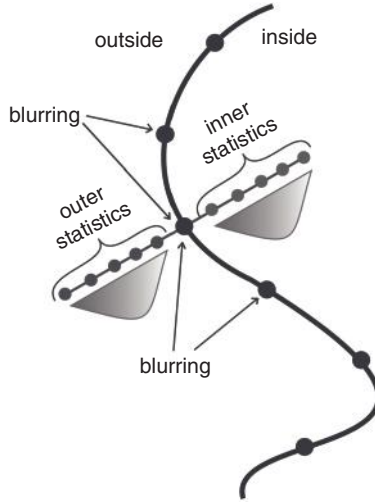


Figure 4.45 Sampling pixels along the normals for collecting local color statistics.

three-channel RGB colors. The local weights w_{ij} decay exponentially with the normalized distance l_j , thus giving a higher level of confidence to pixels near the contour.

Afterward, single-line statistics [eq. (4.83)] are *blurred* along the contour in order to distribute them onto local areas:

$$\tilde{v}_i^{o,B/F} = \sum_{i'} \exp(-\lambda \|\mathbf{y}_i - \mathbf{y}_{i'}\|) v_{i'}^{o,B/F}, \quad o = 0, 1, 2 \quad (4.84)$$

with a blurring coefficient λ , eventually using image coordinates scaled to the size of the object projected, in order to achieve scale invariance. Finally, the area statistics are normalized:

$$\bar{I}_i^{B/F} = \frac{\tilde{v}_i^{1,B/F}}{\tilde{v}_i^{0,B/F}}, \quad \bar{R}_i^{B/F} = \frac{\tilde{v}_i^{2,B/F}}{\tilde{v}_i^{0,B/F}} \quad (4.85)$$

providing two-sided local RGB means \bar{I} and 3×3 covariance matrices \bar{R} .

The third step involves computing residuals and Jacobian matrices for the Gauss–Newton pose update. For this purpose, the pixel colors observed, $I(\mathbf{y}_i + L\mathbf{n}_i)$, are classified as being in the foreground region, according to the statistics collected using eq. (4.85) under a *fuzzy* membership rule $a(\cdot)$:

$$a(l) = \frac{1}{2} \left[\operatorname{erf} \left(\frac{l}{\sqrt{2}\sigma} \right) + 1 \right] \quad (4.86)$$

which becomes a sharp $\{0, 1\}$ assignment for $\sigma \rightarrow 0$. Fuzzy pixel assignment is then accomplished by mixing the two statistics accordingly:

$$\begin{aligned}\hat{I}_{ij} &= a(l_j) \bar{I}_i^F + (1 - a(l_j)) \bar{I}_i^B \\ \hat{R}_{ij} &= a(l_j) \bar{R}_i^F + (1 - a(l_j)) \bar{R}_i^B\end{aligned}\quad (4.87)$$

and color residuals are given by

$$e_{ij} = I(\mathbf{y}_i + L\mathbf{n}_i) - \hat{I}_{ij} \quad (4.88)$$

with measurement covariances \hat{R}_{ij} .

By organizing the residual \mathbf{E} and its covariance matrix $\mathbf{R} = \text{blockdiag}(\hat{R}_{ij})$ in vector form, we express the CCD likelihood as a Gaussian,

$$P_{\text{CCD}}(z|s) \propto \exp\left(-\frac{1}{2} \mathbf{E}^T \mathbf{R}^{-1} \mathbf{E}\right) \quad (4.89)$$

which *contracts* after each pose update by reducing the search length L through a simple exponential decay.

Finally, the $3 \times n$ derivatives of e_{ij} can be computed by differentiating eq. (4.87) with respect to pose parameters p (Section 2.3)⁴:

$$H_{ij} = -\frac{\partial \hat{I}_{ij}}{\partial p} = -\frac{1}{L} (\bar{I}_i^F - \bar{I}_i^B) \cdot \frac{\partial a}{\partial l_j} \cdot \left(\mathbf{n}_i^T \frac{\partial \mathbf{y}_i}{\partial p} \right) \quad (4.90)$$

which are stacked together in the measurement Jacobian \mathbf{H} . Overall, the dimension of the residual is given by $6NJ$, with N the number of contour points.

We note how the feature-level measurement can in principle be used within any type of filter, such as a particle filter, where the likelihood is evaluated for each state hypothesis. However, because of its computational complexity, this measurement is generally used for a maximum-likelihood optimization by performing a Gauss–Newton loop with the data defined above, which in this case is a *multiresolution* approach.

A summary of the CCD modality is given in Fig. 4.46.

4.5 KEYPOINTS

When a textured model is available, local keypoints provide a very useful cue for object tracking. A keypoint consists basically of a photometric pattern,

⁴Following Panin et al. [135], we neglect the dependence of R_{ii} on p while computing the Jacobian matrices.

- *Preprocessing* (none; color pixels are used directly)
- *Sampling model features*. The sampling procedure of Fig. 4.38 applies here as well.
- *Data association*
 1. Pixel level (not defined, since preprocessed data are in the feature space)
 2. Feature level. Local color statistics are computed *on the fly* and used to classify color pixels on both sides of the contour; the result is a set of *color residuals* and related 3×3 covariances and Jacobian matrices.
 3. Object level. A maximum-likelihood estimate is obtained by nonlinear LSE, weighted by the inverse covariances.
- *Update model features* (not defined)

Figure 4.46 CCD modality.

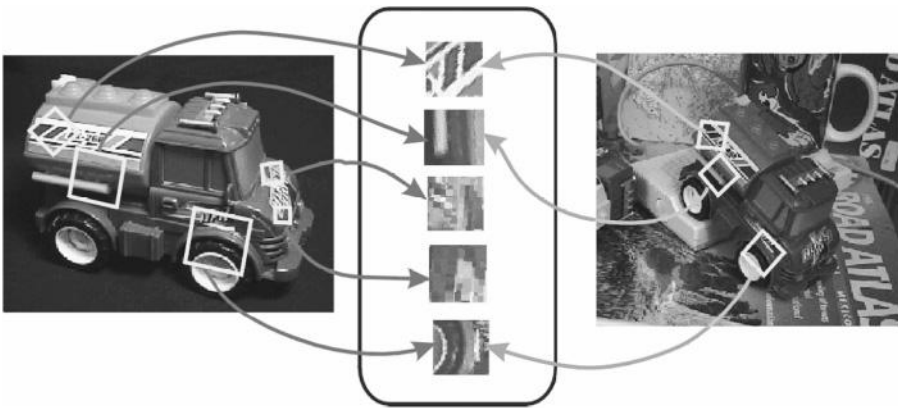


Figure 4.47 Keypoint detection and matching between two views of an object. *Middle*: a few keypoints detected from the reference view (*left*) and matched to the current viewpoint (*right*).

collected around a surface pixel, that can be identified and located with good reliability from different viewpoints (Fig. 4.47).

To be reliable for tracking, local keypoints should satisfy some geometric and photometric properties:

- They should be *invariant* to some extent, in order to be matched from different viewpoints and lighting conditions.
- They should have a *distinctive* pattern; for this reason, they should also not be too close in space, so that they belong to differently textured regions.

- Similar to contour points, they should correspond to real object features and not to light spots, shadows, or false-depth intersections.
- They should be located either on a locally *planar* area or on a pointwise discontinuity such as a corner or junction, but never along an edge, which would prevent unambiguous localization.

All of these aspects determine the *quality* of a keypoint for tracking.

4.5.1 Wide-Baseline Matching

When matching keypoints between two different views, one important distinction can be made between *short-* and *wide-baseline* matching. In fact, for a viewpoint change the almost planar pattern of a keypoint undergoes a transformation, modeled by a planar *homography*, with a maximum of 8 degrees of freedom. This is given by the relationship between the projections of a plane in two views, under perspective cameras (Figure 4.48).

In this example, the two external camera matrices are given by R_1, \mathbf{t}_1 and R_2, \mathbf{t}_2 , and the internal ones are K_1, K_2 , so that the two projections of a space point \mathbf{x} are given by $\mathbf{y}' = K_1[R_1|\mathbf{t}_1]\mathbf{x}$ and $\mathbf{y}'' = K_2[R_2|\mathbf{t}_2]\mathbf{x}$, respectively. By considering the plane π in world coordinates, with orientation \mathbf{n} and distance d to the origin, we can express the relationship between its projections \mathbf{y}' and \mathbf{y}'' onto the two views by

$$H_\pi = K_1(\delta R - \delta \mathbf{t} \mathbf{n}'^T / d') K_2^{-1} \quad (4.91)$$

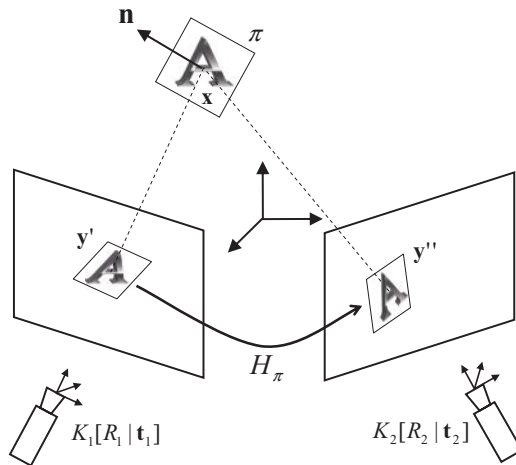


Figure 4.48 Homography between two views induced by a three-dimensional plane. Each point \mathbf{y}' on the left-plane projection corresponds to a point $\mathbf{y}'' = H_\pi \mathbf{y}'$ in the second image. The homography H_π can be computed from the two camera matrices.

which is also called *induced homography* from a three-dimensional plane, where

$$\begin{aligned}\delta R &= R_2 R_1^T \\ \delta \mathbf{t} &= -\delta R \mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{n}' &= R_1 \mathbf{n} \\ d' &= d - \mathbf{t}_1^T \mathbf{n}'\end{aligned}\tag{4.92}$$

When matching different views from the same camera, the intrinsic matrix is supposed to be constant: $K_1 = K_2 = K$. The induced homography H_π obviously depends on π , so it must be computed separately for each keypoint. For a short baseline such as two subsequent frames of a sequence, the two views are quite close to each other. Then H_π can be approximated by a simpler transformation, such as affine (6 dof), similarity (4 dof), or even pure translation (2 dof). This simplifies the localization procedure greatly, but at the same time does not require a high-quality pattern, therefore allowing the use of more keypoints.

For a wide baseline, a full 8-dof deformation or, in the best case, an affine approximation must be computed if the keypoint descriptor has no strong invariance properties. Moreover, since the two views are very different, usually the corresponding pattern may also change because of different shading due to the light incidence. Therefore, the feature descriptor should be robust to the following distortions:

- Planar rotation
- Scale, due to depth variation
- Shading, due to brightness and contrast variations
- Perspective distortion, due to out-of-plane rotations

This can be achieved by *warping* the feature descriptor to the pose predicted, as shown in Fig. 4.48, or by transforming the gray-level pattern into a feature space, where the invariance properties mentioned above are enforced, so that the descriptor does not change significantly.

For object localization we also distinguish between the two main tasks of tracking and detection, here implying the use of different keypoints; in fact, during tracking a state prediction for re-projecting the keypoint and its pattern is available, whereas during detection this is not possible. In the following we provide two well-known examples of local keypoints suitable for the respective localization problems.

4.5.2 Harris Corners

A first idea for selecting keypoints is to look for points that possess simple geometric properties, such as being located on corners or on junctions along

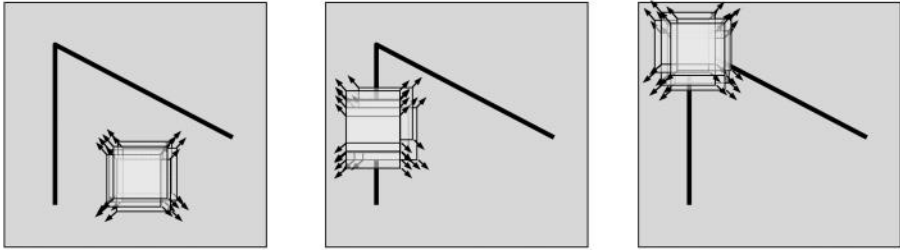


Figure 4.49 Corner detection: The local intensity pattern exhibits a strong variation in any direction.

the surface of an object. These points can be detected robustly from different views with a well-defined position, which are two essential requirements. Corners, in particular, can be defined as points that lie at the intersection of two edges or, equivalently, that show two dominant edge directions: that is, sharp variations of the image pattern in a local neighborhood (Fig. 4.49). Therefore, we can measure this property by means of a sliding window, through the *autocorrelation* function, which is the sum of squared differences (SSD) between the image patch at a point (y_1, y_2) and its shifted version at $(y_1 + \delta y_1, y_2 + \delta y_2)$:

$$E(y_1, y_2, \delta y_1, \delta y_2) = \sum_{i=-n}^n \sum_{j=-n}^n w(i, j) [I(y_1 + i, y_2 + j) - I(y_1 + \delta y_1 + i, y_2 + \delta y_2 + j)]^2 \quad (4.93)$$

where n is the half-size of the window, which is an odd number, $2n + 1$. In this formula, all numbers are integer, and the weighting function $w(i, j)$ is a radially symmetric kernel around zero, normalized to $\sum_{i=-n}^n \sum_{j=-n}^n w(i, j) = 1$. This provides an isotropic weighting, so that pixels in the center are considered more reliable than peripheral pixels in terms of the *corneriness* property. For a small window of a few pixels, eq. (4.93) can be linearized around (y_1, y_2) :

$$I(y_1 + \delta y_1, y_2 + \delta y_2) \approx I(y_1, y_2) + I_{y_1}(y_1, y_2) \delta y_1 + I_{y_2}(y_1, y_2) \delta y_2 \quad (4.94)$$

where I_{y_1}, I_{y_2} are the first image derivatives, and therefore

$$E(y_1, y_2, \delta y_1, \delta y_2) \approx \sum_{i=-n}^n \sum_{j=-n}^n w(i, j) [I_{y_1}(y_1 + i, y_2 + j) \delta y_1 + I_{y_2}(y_1 + i, y_2 + j) \delta y_2]^2 \quad (4.95)$$

which can also be written in matrix form,

$$E(y_1, y_2, \delta y_1, \delta y_2) = [\delta y_1 \quad \delta y_2] Z(y_1, y_2) \begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} \quad (4.96)$$

where Z is the autocorrelation matrix:

$$Z(y_1, y_2) = \sum_{i=-n}^n \sum_{j=-n}^n w(i, j) \begin{bmatrix} I_{y_1}^2 & I_{y_1} I_{y_2} \\ I_{y_1} I_{y_2} & I_{y_2}^2 \end{bmatrix} = \begin{bmatrix} \langle I_{y_1}^2 \rangle & \langle I_{y_1} I_{y_2} \rangle \\ \langle I_{y_1} I_{y_2} \rangle & \langle I_{y_2}^2 \rangle \end{bmatrix} \quad (4.97)$$

Therefore, the autocorrelation function E can be approximated locally by a quadratic form, where the cornerness property is expressed directly in terms of the two eigenvalues of Z , λ_1 and λ_2 , where $\lambda_1 \geq \lambda_2 \geq 0$. By referring to the three cases in Fig. 4.49, we have:

- *Flat region.* Both eigenvalues are small (i.e., there is little variation of E in both directions), and for any shift $(\delta y_1, \delta y_2)$ the window exhibits a similar pattern.
- *Edge.* One of the two eigenvalues is much larger than the other, so there is a strong variation across the edge but not along the edge.
- *Corner.* Both eigenvalues are large (i.e., there is a strong variation of E in any direction).

In the work of Harris and Stephens [75], the measure of the cornerness property is given by

$$R = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 = \det(Z) - \kappa \text{trace}^2(Z) \quad (4.98)$$

in terms of the determinant and trace of Z , so that the eigenvalues must not be computed explicitly. κ is an empirical constant, usually set to $\kappa = 0.04$. With this index, a corner is characterized by $R \gg 0$, a flat region by $R \ll 0$, and an edge by $R \approx 0$; therefore, by setting a suitable threshold on R , one can localize regions around corner points, refined subsequently by nonmaxima suppression. Figure 4.50 shows an example of the detector workflow.

The Harris detector is invariant to planar roto-translation and to some extent to light changes. In fact, if the keypoint is rotated, the eigenvectors of Z rotate as well but the eigenvalues do not change, therefore R is the same; moreover, it can be shown that R depends linearly on the image gradients I_{y_1}, I_{y_2} , so that a linear change in the image $I' = aI + b$ causes a proportional change $R' = aR$, and the same local maxima can be detected provided that the threshold is adjusted properly (Fig. 4.51, top).

Unfortunately, this detector is not invariant to *scale*, since the cornerness property depends strongly on window size, which should match the size of the corner (Fig. 4.51, bottom), which is not known in advance. Some works (e.g., [120]) discuss how to achieve invariance to affine transforms at the price of a

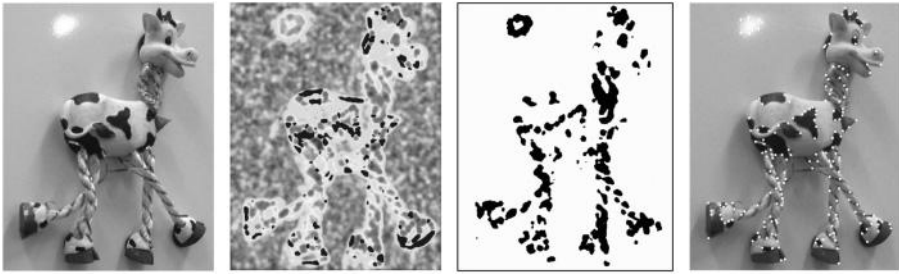


Figure 4.50 Harris–Stephens corner detection. *Left to right:* original image; corner response (dark pixels represent negative values); positive response blobs after thresholding; corner points detected after nonmaxima suppression.

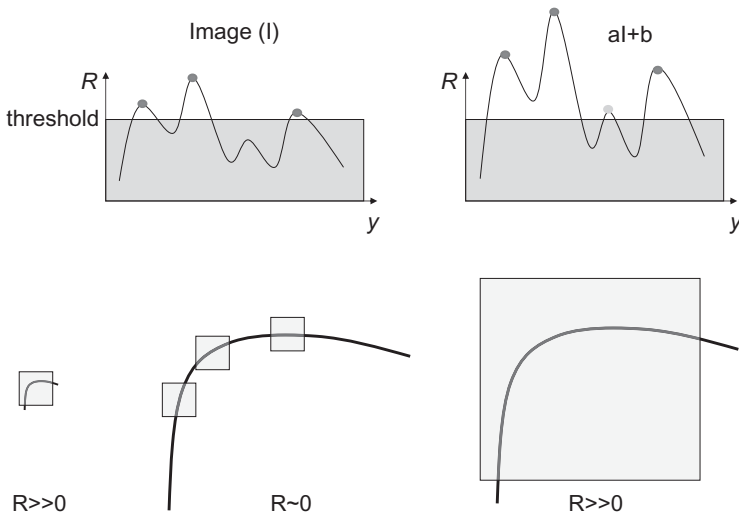


Figure 4.51 *Top:* invariance of Harris corner detector to lighting; *bottom:* the detector is not invariant to scale.

higher computational complexity. In the next section we describe instead a well-known scale-invariant detector and the related description and matching procedures, achieving near real-time performance. However, once again we remark that in the context of object tracking, the Harris detector is fast and reliable, especially for short-baseline matching.

Keypoint matching by re-projection was illustrated in Fig. 1.6 and is summarized in Fig. 4.52. Other examples of the use of Harris keypoints are frame-to-frame feature tracking for sparse optical flow computation (Section 4.6.2), and stereo correspondence problems [77].

- *Preprocessing.* Keypoints are detected in the current frame.
- *Sampling model features.* The textured model is rendered at the pose predicted, and keypoints are detected and back-projected in object space; these are off-line model features. To save computation, this step can be done off-line by rendering a set of *key frames* and sampling features, followed by selecting the features from the closest view.
- *Data association*
 1. Pixel level (not present, since preprocessing produces a feature-level output)
 2. Feature level. Model keypoints (off-line and online) are re-projected in the image and matched to the keypoints detected. For wide-baseline matching a further *warp* of the pattern is required, while online keypoints are matched on a short baseline.
 3. Object level. After feature matching, MLE pose estimation from point correspondences can be carried out.
- *Update model features.* Keypoints detected on the object surface that have not been matched to any model feature are back-projected under the estimated pose and will be used for the next frame.

Figure 4.52 Harris keypoints modality: matching by re-projection (see also Fig. 1.6).

4.5.3 Scale-Invariant Keypoints: Detection, Description, and Matching

The scale-invariant features transform (SIFT) [106,172] is a method for detecting distinctive keypoints that can be found and matched across different translation, planar rotation, scale, illumination, and to a lesser extent, perspective transforms. In the current literature, many other examples of invariant keypoints, such as the speeded-up robust features method (SURF) [36], can be found. However, for reasons of space in the following we describe only the traditional SIFT algorithm; for more information we refer readers to two surveys [121,154].

In SIFT, scale invariance is achieved through the use of a *scale-space* representation [103]. This representation is based on the notion of multiscale (or multiresolution) analysis of an image, where the gray-level pattern is filtered by Gaussian kernels of increasing size (Fig. 4.53); in particular, the size σ of the kernel constitutes the scale in a three-dimensional space (x, y, σ) , where



Figure 4.53 Multiscale representation of an image.

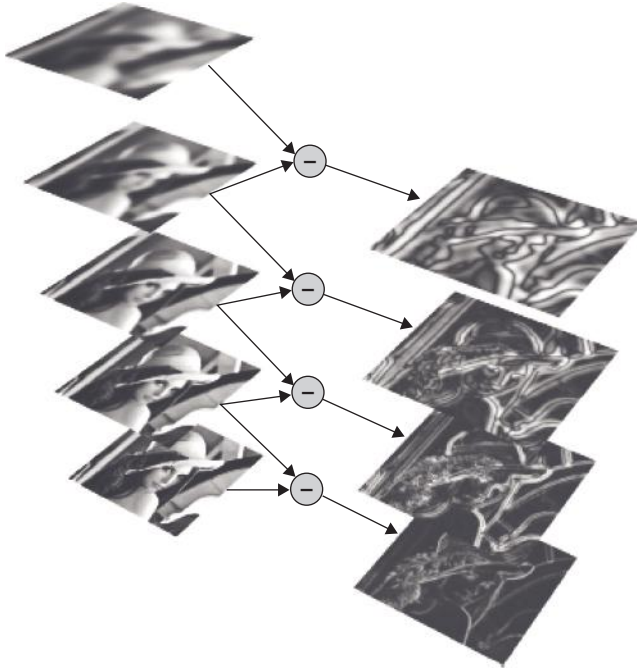


Figure 4.54 Difference-of-Gaussians pyramid.

for increasing scale, small details are subsequently suppressed and only the main features (or *blobs*) are detected.

If the kernel used for the scale-space representation is the normalized Laplacian of Gaussian, which is the same as that used by the Marr–Hildreth edge detector of Section 4.4.1, we have the N-LoG *pyramid*; this pyramid can be approximated efficiently by computing the difference of Gaussians (DoG) at adjacent scales (Fig. 4.54):

$$\sigma^2 \nabla^2 \mathcal{N}(x, y, \sigma) \approx \mathcal{N}(x, y, k\sigma) - \mathcal{N}(x, y, \sigma) \quad (4.99)$$

where \mathcal{N} is the Gaussian distribution, with standard deviation σ and k a suitable factor, usually $k = 1.4$, on which the quality of approximation also depends.

A main result from scale-space theory is that salient keypoints can be detected as local maxima of the absolute values of the N-LoG, or DoG representation in both space *and* scale (Fig. 4.55).

To keep constant the size of the feature detected despite scale variations, the images are also subsampled by a factor of 2 at each *octave* (i.e., each doubling of the scale σ). Since the scale increases logarithmically, $\sigma_{k+1} = k\sigma_k$ with a factor $k < 2$, subsampling occurs only at some steps in the pyramid.

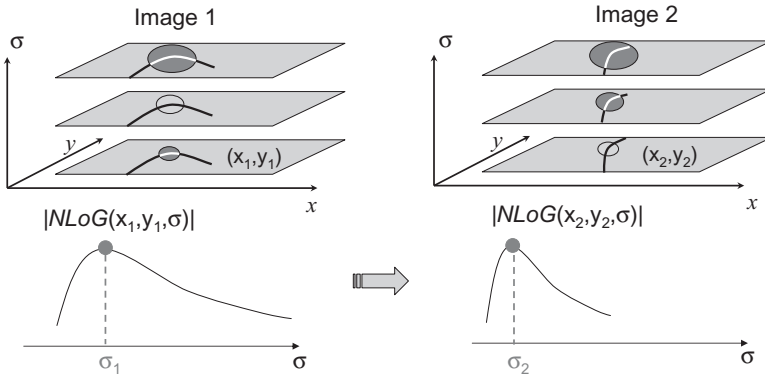


Figure 4.55 Invariant keypoints are found as local maxima, in space and scale, of the absolute values of the N-LoG image pyramid.

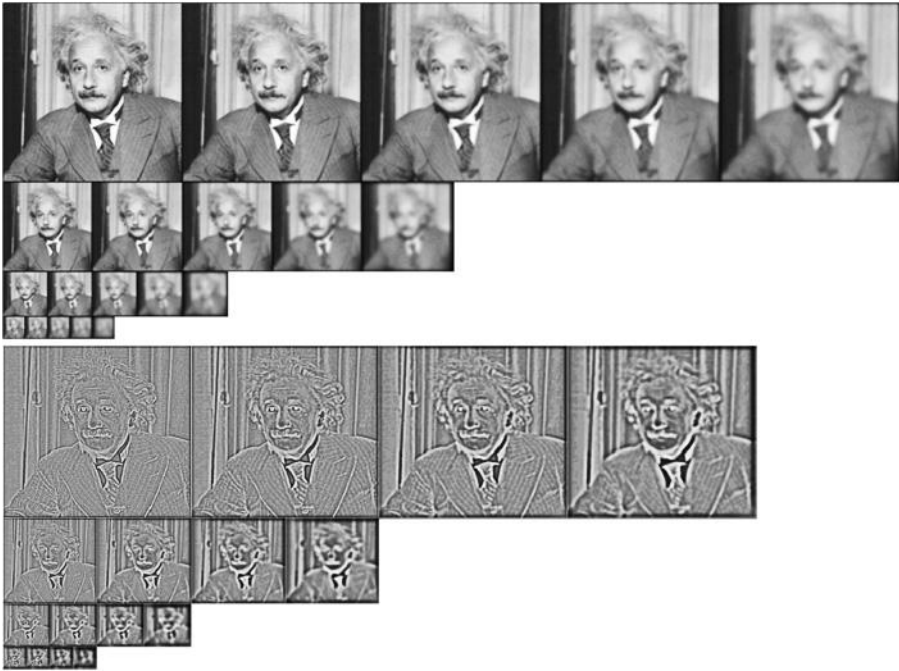


Figure 4.56 Top rows: Gaussian pyramid subsampled at scale *octaves*; bottom rows: related difference-of-Gaussians pyramid.

Figure 4.56 shows an example, with subsampling at octaves, and the related difference of Gaussians.

Nonmaxima suppression is subsequently applied to the *absolute* DoG pyramid by using a $3 \times 3 \times 3$ mask. At double scales, keypoints also are almost

double in size, but because of subsampling the size of the mask can be kept constant.⁵ Local maxima are then refined, with subpixel and subscale accuracy, by least-squares fitting of a quadratic form in three-dimensions, and by localizing the maximum. This can be accomplished as described by Brown and Lowe [48], by expanding the scale-space function $D(\mathbf{x})$ with $\mathbf{x} = (x, y, \sigma)$ in a neighborhood of the interest point \mathbf{x}_0 :

$$D(\mathbf{x}) \approx D(\mathbf{x}_0) + \frac{\partial D}{\partial \mathbf{x}} \delta \mathbf{x} + \frac{1}{2} \delta \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \delta \mathbf{x} \quad (4.100)$$

where $\delta \mathbf{x} = \mathbf{x} - \mathbf{x}_0$ is the offset (with $\{+1, 0, -1\}$ values), and the derivatives are computed in \mathbf{x}_0 by numerical approximation with finite differences between neighbors.

The local maximum is given by

$$\hat{\mathbf{x}} = \mathbf{x}_0 - \frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}} \quad (4.101)$$

and the maximum value is

$$D(\hat{\mathbf{x}}) = D(\mathbf{x}_0) + \frac{1}{2} \frac{\partial D}{\partial \mathbf{x}}^T (\hat{\mathbf{x}} - \mathbf{x}_0) \quad (4.102)$$

This value is also used to suppress weak maxima⁶ (Fig. 4.57, top), typically with $|D(\hat{\mathbf{x}})| < 0.03$. As we can see from the bottom of Fig. 4.57, after planar rotation and scaling of the original picture, most of the same keypoints are found, with the related scale and orientation values.

The two feature sets can now be matched, provided that their pattern is also represented by an invariant *descriptor*. In particular, the descriptor used by Lowe [106] is based on *histograms of oriented gradients* (HOGs), obtained by computing the image gradients over the feature window and histogramming the orientations after zeroing the main component. Since gradients are related only to local variations of illumination, not to the absolute values, the average brightness is removed from the descriptor.

To achieve rotation invariance, a global HOG is first computed over a window around \mathbf{x}_0 at the related scale (Fig. 4.58, top) and the main component

⁵A tricky situation occurs between two scales because of subsampling, which is solved by re-sampling one of the adjacent images (either the higher- or lower-resolution image) to the size of the other, in order to apply the mask.

⁶In this context, keypoints located on *edges* also have to be removed; this can be accomplished as well by computing the *eigenvalues* of the Hessian matrix of D , as described by Lowe [106].

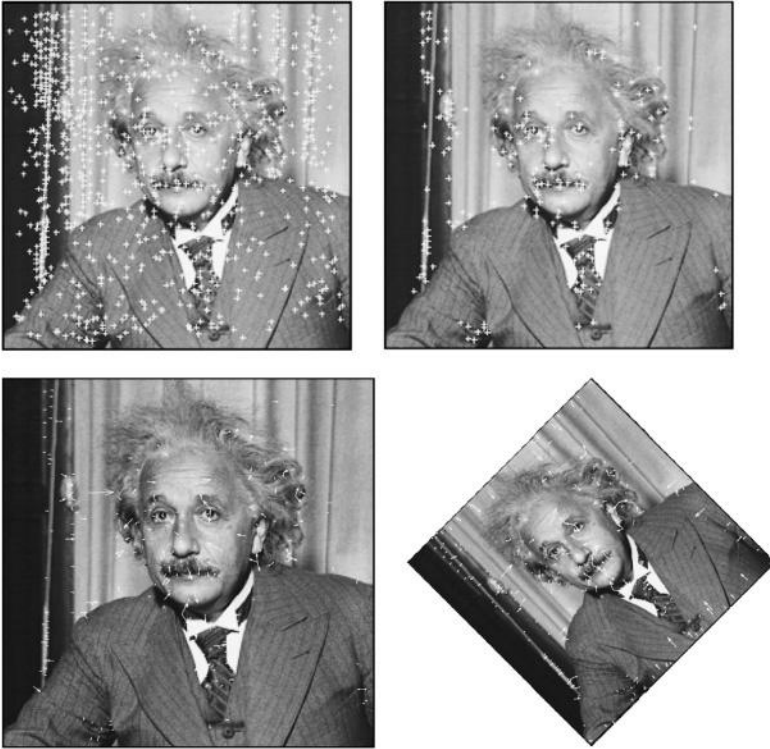


Figure 4.57 *Top left:* keypoints detected after nonmaxima suppression; *top right:* removing weak keypoints; *bottom:* invariance SIFT keypoints to scale and orientation. Most of the same keypoints can also be found in the second view, with the new scale and orientation.

θ is removed by rotating back all gradients by θ . The histogram bins are also weighted by the magnitude of gradients, since small gradients are affected by noise, as well by the distance from the center of the window, with a Gaussian kernel. In case of two significant orientations, θ_1 and θ_2 , both are retained and the keypoint is duplicated with the respective θ .

Afterward, the window is split into subblocks and multiple weighted HOGs are computed, with a reduced number of bins, typically eight per window, because of the reduced sample size (Fig. 4.58, bottom). If the window consists of 16×16 pixels and is subdivided into 4×4 blocks, there will be $8 \cdot 16 = 128$ elements in the feature descriptor plus the four entries x , y , σ and θ . Histogram descriptors are also *normalized* to 1 to achieve invariance to linear *contrast* variations, which affect only the gradient magnitudes. These descriptors can be used for matching across different images by minimizing Euclidean distances.

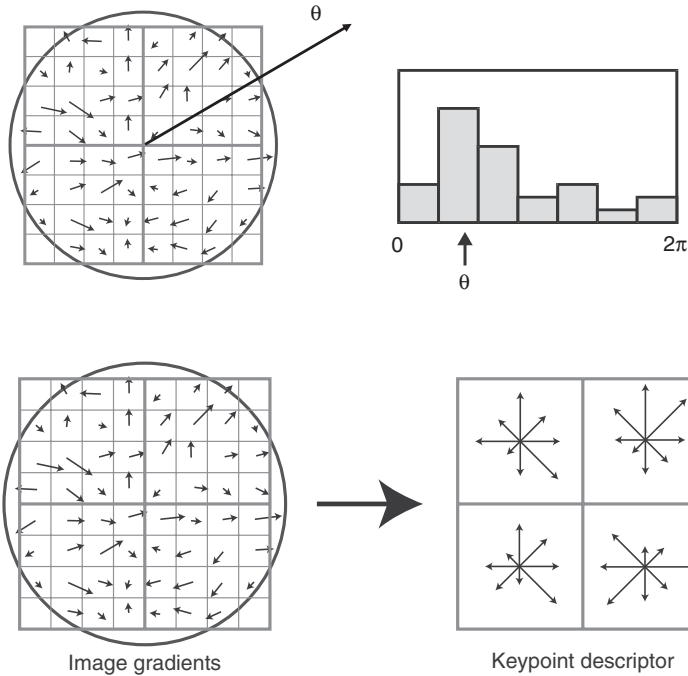


Figure 4.58 *Top*: computing the histogram of oriented gradients over a window; *bottom*: the SIFT descriptor is a set of local HOGs. (From [106].)

4.5.4 Matching Strategies for Invariant Keypoints

Invariant features provide a good example of *static* data association, where no prediction is required in order to perform matching. In particular, for SIFT keypoints, matching is carried out simply by comparing the HOGs, so that, in principle, no information about the feature location, rotation, or scale is required.

Although the SIFT descriptor is relatively distinctive and robust, if no prediction of pose parameters is available, the nearest-neighbor search will have a combinatorial complexity, even in the presence of a moderate set of points (e.g., around 1000), since all possible pairings must be evaluated; therefore, a more efficient scheme than exhaustive search must be adopted. In [106] Lowe proposed the *best-bin-first* (BBF) approach [37], which consists of an approximate search scheme, returning with high probability, although not 100%, the nearest-neighbor matchings.

The BBF approach consists basically of a modified search ordering for the better known *k-d tree* (*k*-dimensional tree) algorithm [83] such that bins in feature space are searched in the order of their closest distance from the query location. A *k-d tree* is a space-partitioning data structure for organizing points

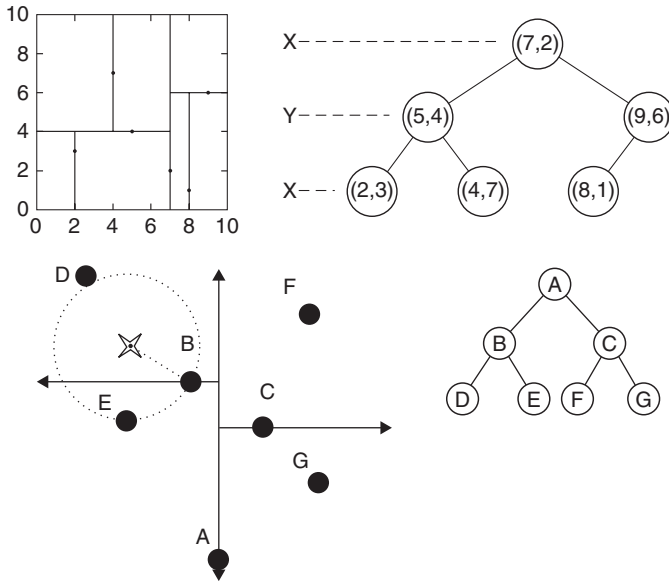


Figure 4.59 Example of a k -d tree (with $k = 2$). *Top left*: space partitioning with median hyperplanes; *top right*: resulting tree; *bottom*: nearest-neighbor search. The cross-shape represents a query point, and the circle radius is given by the best neighbor currently found.

in k -dimensional space, which facilitates searches involving a multidimensional *key*, or descriptor, such as range searches and nearest-neighbor searches.

This type of partitioning can also be seen as a special case of *binary space partitioning* (BSP) trees [67], which is a method for recursively subdividing a space into convex sets by hyperplanes, which give rise to a representation of the scene by means of a tree data structure. In a k -d tree (Fig. 4.59, top) every node is a k -dimensional point, associated with a splitting *hyperplane*, which divides the space \mathfrak{R}^k into two subspaces: All points to the left of the hyperplane represent the left subtree of that node, and points to the right of the hyperplane represent the right subtree.

In particular, given N sample points in \mathfrak{R}^k , the first cutting hyperplane is chosen along the dimension i , where they show the largest variance, and its position is given by the *median* of all points m along that dimension. Information about the splitting hyperplane of each partition (or *bin*) is stored in the tree node. The same procedure is applied recursively to the two subspaces generated by this node and terminates at the *leaf* nodes, where only one point, the median, is present. Overall, this procedure creates a tree with depth $d = \lceil \log_2 N \rceil$. The leaf nodes of a k -d tree partition the sample set completely, in such a way that small bins are created in regions where points have a higher density, and vice versa.

Using this structure, a nearest-neighbor (NN) search for a query point, corresponding to the cross shape on the bottom of Fig. 4.59, can be made efficiently in a two-step procedure:

1. By traversing the tree once downward, from the root to a leaf, we can locate the bin containing the query point, which probably (but not certainly) contains its best NN, and store this leaf node as the current estimate of NN, with distance d_{curr} .
2. Afterward, the tree is traversed upward, from leaf to root, pruning entire branches of the tree if their node is at a distance from the query higher than d_{curr} while exploring downward branches that have not already been traversed, and updating d_{curr} whenever a better NN is found on a leaf.

The pruning condition corresponds to seeing if a given hyperplane intersects the hypersphere of radius d_{curr} centered in the query point. The search is terminated as soon as all of the unexplored branches have been pruned or, equivalently, as soon as the root node is reached again. This procedure becomes less effective in high-dimensional spaces, because the number of bins adjacent to the first that may be visited is high. However, a k-d tree is not suitable for efficient determination of the nearest neighbor in high-dimensional spaces such as the SIFT descriptor.

As a general rule, the number of points in the data should be $N \gg k$; otherwise, most of the points in the tree will be evaluated during the search, and the efficiency will not be better than an exhaustive search. If this is not the case, an approximate solution can be obtained by stopping the search as soon as a given number of leaf nodes have been visited while also paying attention to the order in which leaves are visited. This leads to the BBF method, and a speed-up about two orders of magnitude over the exhaustive search (for more details, see [37]).

4.6 MOTION

When the object moves with respect to the camera, or the camera moves in the scene, the projection of visible surfaces onto the screen changes over time. Therefore, the relative spatial velocity results in local variations of pixel intensities that can be described by a *motion field*. Motion fields can be computed in a variety of ways, ranging from simple temporal image differences up to sparse or dense vector fields (*optical flow*). According to the desired amount of information, as well as accuracy and robustness, the complexity of such algorithms is also widely different. We consider here two distinct motion field modalities: motion history images and optical flow.

4.6.1 Motion History Images

Motion can be detected on a pixel basis: first, as the amount of variation of gray-level values over time. This provides a coarse estimation of the image-

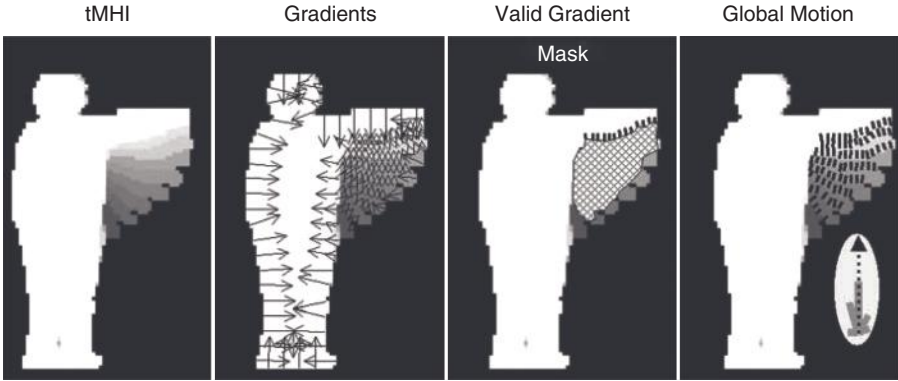


Figure 4.60 Motion history image. *Left:* temporal motion history image; *center left:* gradients of the tMHI; *center right:* mask for removing gradients; *right:* global motion vector. (From [46].)

projected *velocity* of the underlying items, which can be due to a moving object or to the *egomotion* of the camera. The absolute image difference provides the overall *magnitude* of image velocity but not the direction of motion. Nevertheless, this information can be useful for segmenting moving shapes from the background.

Motion is detected primarily on the boundary of objects, where sharp intensity variations occur in both space and time. However, textured surfaces also offer strong cues for motion detection. Motion silhouettes can also be computed by accumulating binary segmentation masks (e.g., foreground segmentation) over time [46,58]. This method consists of superimposing the most recent silhouettes onto the older ones (Fig. 4.60) and computing the image gradients; following this, gradients in the interior of the temporal motion history image (tMHI) are retained, while on the boundary they are discarded, and the overall amount of motion can be computed as shown on the right in Fig. 4.60.

The tMHI contains the time stamps (per pixel) of the current and old silhouettes. Denoting by $S(x, y, t)$ the current silhouette, at time t we have

$$\text{tMHI}(x, y, t) = \begin{cases} t, & \text{if } S(x, y, t) = 1 \\ 0, & \text{if } S(x, y, t) = 0 \text{ and } \text{tMHI}(x, y) < (t - \tau_m) \end{cases} \quad (4.103)$$

where τ_m is the maximum delay before removing an old silhouette from the tMHI; in this way, the most recent silhouette always receives a higher value than those of the old ones. After masked gradients in the horizontal and vertical directions $I_x(x, y)$ and $I_y(x, y)$ have been computed using a Sobel filter, the respective orientations are given by $\phi(x, y) = \text{atan2}(I_y, I_x)$. Then the global orientation $\bar{\phi}$ can be computed following Bradski and Davis [46], by first

- *Preprocessing*: The motion history image is computed from the current and previous frames.
- *Sampling model features* (not defined, since this is a pixel-level measurement)
- *Data association*
 1. Pixel level. The motion image expected can be computed from a full state (position and velocity) hypothesis, or simply by the binary silhouette expected, and compared to the motion image observed.
 2. Feature level (not defined; see above)
 3. Object level. Minimize the overall image residual with a local optimization algorithm.
- *Update model features* (not defined; see above)

Figure 4.61 Motion history images.

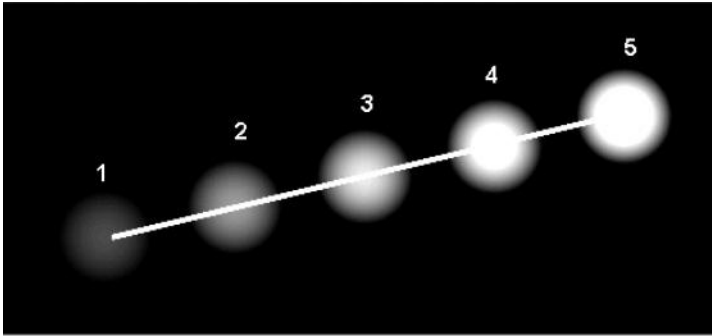


Figure 4.62 Optical flow vector of a moving object in a video sequence. (From [21].)

taking a histogram of oriented gradients and computing the peak value as a reference, ϕ_{ref} . Afterward, we have

$$\bar{\phi} = \phi_{\text{ref}} + \frac{\sum_{x,y} \text{angDiff}(\phi(x, y), \phi_{\text{ref}}) \cdot \text{tMHI}(x, y)}{\sum_{x,y} \text{tMHI}(x, y)} \quad (4.104)$$

where $\text{angDiff}(\phi_1, \phi_2)$ is the smallest angle between ϕ_1 and ϕ_2 , with a sign. A reference angle ϕ_{ref} is needed because of difficulties while averaging circular image measurements.

A summary of motion history images is given in Fig. 4.61.

4.6.2 Optical Flow

Optical flow is a vector field representing projected velocities of real scene points, as reconstructed from pixel intensity variations over space and time from two subsequent images (Fig. 4.62). Optical flow has been shown to be

strongly present in human and animal vision, which further motivates its use for machine vision and object tracking. By recalling our basic camera mapping [eq. (2.57)], we have

$$\dot{\mathbf{y}} = \frac{d}{dt} \mathcal{W}(\mathbf{x}, \mathbf{p}(t)) = \sum_d \frac{\partial}{\partial p_d} \mathcal{W}(\mathbf{x}, \mathbf{p}) \dot{p}_d \quad (4.105)$$

At the pixel level, the instantaneous velocities $\dot{\mathbf{y}}$ are in turn related to the variations observed in the color pattern $I(\mathbf{y})$, which can be put into mathematical relationship with the motion field, and used both for object localization and for shape reconstruction and scene modeling. However, the latter relationship is nontrivial as to both correct formulation and solution, and requires some crucial assumptions to hold, at least approximately. In particular, the most common and basic assumption is called *brightness constancy* between consecutive images:

$$I(\mathbf{y}, t) = I(\mathbf{y} + \delta\mathbf{y}, t + \tau) \quad (4.106)$$

where pixel \mathbf{y} at time t *corresponds* to pixel $\mathbf{y} + \delta\mathbf{y}$ in the next frame (i.e., they are projections of the same physical point \mathbf{x} in the scene). This assumption is in principle satisfied only by points lying on *Lambertian* surfaces in the scene, with a distant and constant light source, without shadows cast on them and with a pure planar translation between views; in fact, relaxing any of these assumptions would modify the local *shading* to some extent. However, provided that both τ and $\delta\mathbf{y}$ are small enough, most points in real-world scenes [eq. (4.106)] are satisfied *approximately*, at least enough to make optical flow techniques work.

In the Lucas–Kanade method [108], we compute a first-order Taylor approximation in both space and time:

$$I(\mathbf{y} + \delta\mathbf{y}, t + \tau) \approx I(\mathbf{y}, t) + \nabla I^T(\mathbf{y}, t) \delta\mathbf{y} + \tau I_t(\mathbf{y}, t) \quad (4.107)$$

where $\nabla I = (I_{y_1}, I_{y_2})$ is the spatial image gradient and I_t is the temporal derivative, both evaluated at \mathbf{y} . Therefore, eq. (4.106) becomes

$$\nabla I^T(\mathbf{y}, t) \delta\mathbf{y} + \tau I_t(\mathbf{y}, t) = 0 \quad (4.108)$$

which relates space and time variations of image intensities, and therefore is also called a *gradient constraint equation*. In this equation, the space derivative can be evaluated by means of gradient masks (e.g., Sobel), while the temporal derivative can be approximated by *image differencing*, $I_t \approx [I(\mathbf{y}, t + \tau) - I(\mathbf{y}, t)]/\tau$.

Unfortunately, solving this equation results in an underconstrained problem, with one equation in two unknowns $\delta\mathbf{y}$, which provides only the *normal component* of the flow to the line defined by ∇I :

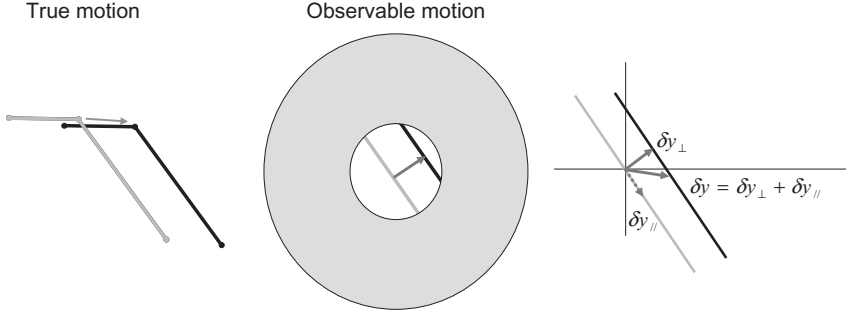


Figure 4.63 The aperture problem: Only the normal component of the flow vector f_{\perp} is observable through a small aperture.

$$\delta \mathbf{y}_{\perp} = -\frac{I_t}{\|\nabla I\|} \frac{\nabla I}{\|\nabla I\|} \quad (4.109)$$

This is also known as the *aperture problem* (Fig. 4.63).

One method of constraining the problem further is to compute the flow of more pixels in a small window around \mathbf{y} , and imposing on all of them, along with the brightness constancy assumption, sharing of the *same* displacement $\delta \mathbf{y}$. This is the Lucas–Kanade method, resulting in the set of equations

$$\begin{bmatrix} I_{y_1}(\mathbf{y}^1) & I_{y_2}(\mathbf{y}^1) \\ \vdots & \vdots \\ I_{y_1}(\mathbf{y}^N) & I_{y_2}(\mathbf{y}^N) \end{bmatrix} \begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{y}^1) \\ \vdots \\ I_t(\mathbf{y}^N) \end{bmatrix} \quad (4.110)$$

for the given neighborhood $\mathbf{y}^1, \dots, \mathbf{y}^N$ around \mathbf{y} (usually, $N = 5 \cdot 5 = 25$ pixels). The least-squares solution to this overconstrained problem is provided by the *normal equations*

$$(A^T A) \delta \mathbf{y} = A^T \mathbf{b} \quad (4.111)$$

where A and \mathbf{b} are the two coefficient matrices in (4.110). Equation (4.110) can also be written as

$$\begin{bmatrix} \sum_i I_{y_1}^i I_{y_1}^i & \sum_i I_{y_1}^i I_{y_2}^i \\ \sum_i I_{y_1}^i I_{y_2}^i & \sum_i I_{y_2}^i I_{y_2}^i \end{bmatrix} \begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} = - \begin{bmatrix} \sum_i I_{y_1}^i I_t^i \\ \sum_i I_{y_2}^i I_t^i \end{bmatrix} \quad (4.112)$$

where the sums are performed over all pixels \mathbf{y}^i in the window. The solution is then



Figure 4.64 Sparse Lucas–Kanade flow vectors. (From [41]. Copyright © 2008 IEEE.)

$$\delta \mathbf{y} = (A^T A)^{-1} A^T \mathbf{b} \quad (4.113)$$

Matrix $A^T A$ in eq. (4.112), also known as the *autocorrelation matrix* of the window, represents the “strength” of the feature point being tracked from frame I_t to frame $I_{t+\tau}$ (see also Section 4.5). We notice how the optical flow solution [eq. (4.113)] does not exactly solve the brightness constancy assumption because of the linearization in eq. (4.107), but in fact corresponds to a single step of *Gauss–Newton optimization*, searching locally for the window with the smallest brightness difference between the two frames. Therefore, by iterating eq. (4.113) we can move the window to $\mathbf{y} \rightarrow \mathbf{y} + \delta \mathbf{y}$ and solve the L-K equations again.

In this process, a big computational saving is achieved by keeping the spatial gradient ∇I (and therefore the matrix A) *constant* because, due to the same brightness constancy assumption, no significant changes in the *slope* of the intensity values should be observed between \mathbf{y} and $\mathbf{y} + \delta \mathbf{y}$. Because of its local formulation, the L-K method produces a sparse flow field (Fig. 4.64) that can reliably be computed only for strong local keypoints, where the eigenvalues of $A^T A$ are high, and the system (4.113) can reliably be solved.

A variational approach to computing dense flow fields is the Horn–Schunk method [80], which imposes regularization constraints to a *global* energy minimization problem:

$$E = \iint [(I_{y_1} v_1 + I_{y_2} v_2 + I_t)^2 + \alpha^2 (|\nabla v_1|^2 + |\nabla v_2|^2)] dy_1 dy_2 \quad (4.114)$$

where $\mathbf{v} = (v_1, v_2)$ is the velocity field at \mathbf{y} (denoted previously by $\delta \mathbf{y}$) and α is a regularization coefficient, enforcing smooth variations of the flow \mathbf{v} over the

image. This constraint is obviously violated on the object boundaries, where the velocity field has a discontinuity, but on most of the image it holds true.

Minimizing (4.114) is a variational problem that can be solved by means of the Euler–Lagrange equations. If we denote the gradients of the two components of the velocity field by $\nabla v_1 = (v_{1,y_1}, v_{1,y_2})$ (and similarly for v_2), we have

$$\begin{aligned} \frac{\partial L}{\partial v_1} - \frac{\partial}{\partial y_1} \frac{\partial L}{\partial v_{1,y_1}} - \frac{\partial}{\partial y_2} \frac{\partial L}{\partial v_{1,y_2}} &= 0 \\ \frac{\partial L}{\partial v_2} - \frac{\partial}{\partial y_1} \frac{\partial L}{\partial v_{2,y_1}} - \frac{\partial}{\partial y_2} \frac{\partial L}{\partial v_{2,y_2}} &= 0 \end{aligned} \quad (4.115)$$

where L is the integrand inside the energy equation (4.114); this gives

$$\begin{aligned} I_{y_1} (I_{y_1} v_1 + I_{y_2} v_2 + I_t) - \alpha^2 \nabla^2 v_1 &= 0 \\ I_{y_2} (I_{y_1} v_1 + I_{y_2} v_2 + I_t) - \alpha^2 \nabla^2 v_2 &= 0 \end{aligned} \quad (4.116)$$

where $\nabla^2 \equiv \partial^2/\partial y_1^2 + \partial^2/\partial y_2^2$ is the Laplace operator. In practice, this is approximated by finite differences $\nabla^2 v_1 \approx \bar{v}_1(y) - v_1(y)$, where \bar{v}_1 is a weighted average of v_1 , computed in a neighborhood of y , and similarly for v_2 .

Therefore, eq. (4.116) can also be written as

$$\begin{aligned} (I_{y_1}^2 + \alpha^2) v_1 + I_{y_1} I_{y_2} v_2 &= \alpha^2 \bar{v}_1 - I_{y_1} I_t \\ I_{y_1} I_{y_2} v_1 + (I_{y_2}^2 + \alpha^2) v_2 &= \alpha^2 \bar{v}_2 - I_{y_2} I_t \end{aligned} \quad (4.117)$$

which is linear in v_1 and v_2 and can be solved for each pixel. However, since the solution depends on neighboring values of the flow field, it must be repeated once the neighbors have been updated. Hence, the following iterative scheme⁷ is adopted:

$$\begin{aligned} v_1^{k+1} &= \bar{v}_1^k - \frac{I_{y_1} (I_{y_1} \bar{v}_1^k + I_{y_2} \bar{v}_2^k + I_t)}{\alpha^2 + I_{y_1}^2 + I_{y_2}^2} \\ v_2^{k+1} &= \bar{v}_2^k - \frac{I_{y_2} (I_{y_1} \bar{v}_1^k + I_{y_2} \bar{v}_2^k + I_t)}{\alpha^2 + I_{y_1}^2 + I_{y_2}^2} \end{aligned} \quad (4.118)$$

where the superscript $k+1$ denotes the next iteration, computed from the result of iteration k .

An advantage of the Horn–Schunck algorithm is that it yields a high-density vector field (Fig. 4.65); that is, the flow information missing in inner parts of

⁷This method is inspired from the *Jacobi method* for determining the solutions of a system of linear equations, with the largest absolute values in each row and column dominated by the diagonal element.

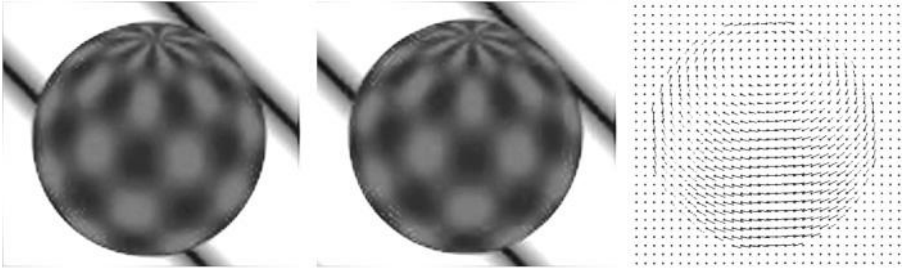


Figure 4.65 Dense optical flow from a synthetic image sequence of a sphere. (From [11].)

homogeneous objects is *filled in* from the motion boundaries by the regularization constraints. On the negative side, it is more sensitive to noise than are local methods such as the Lucas–Kanade method, and, of course, computationally more complex.

The optical flow modality produces information at the pixel level (dense map) or feature level (sparse motion vectors) which can be matched directly to the model flow expected, obtained from the object shape at a given state hypothesis. In this case, obviously the state includes position *and* velocity; therefore, this modality cannot be used, for example, with a simple Brownian dynamics (Section 2.4), where the object velocity is unpredictable. Moreover, a motion-based modality alone may lead to error accumulation, or pose *drift*, since there are no absolute reference features during matching; only relative displacements between frames are considered. Therefore, it is usually combined with other modalities, such as contours.

A summary of optical flow modality is given in Fig. 4.66

4.7 TEMPLATES

Template-based tracking consists of using all of the surface information from a textured object as a *global feature* for matching. When the appearance model is represented by a texture map (Fig. 2.17), it can be rendered from any viewpoint and compared directly with the underlying image pixels. This is a pixel-level measurement that exploits the largest amount of information available. However, it may also result in a high computational complexity, which usually involves computing the Jacobian images for each parameter (for example, in a Gauss–Newton optimization), which in turn requires a *back-projection* of all visible pixels onto the object surface.⁸ For two-dimensional models, speed-up

⁸More efficiently, Jacobians can be approximated on GPU *shaders* by computing screen derivatives of vertices and interpolating them as vertex *attributes*, without the need for back-projection but at the expense of a large back-transfer of data from the graphics card memory.

- *Preprocessing.* A dense or sparse flow field is computed from current and previous images. For a sparse field, feature points are *tracked* from the preceding frame, including replacement of lost features because of occlusions, lighting, or fast motion.
- *Sampling model features* (not available, since features are tracked from frame to frame; therefore, no texture model is required)
- *Data association*
 1. Pixel level. For a dense field, matching can be done between the flow expected and the flow observed, induced by the object pose and velocity.
 2. Feature level. Data association has already been carried out in the preprocessing step. The residual is the difference between re-projected flow vectors (positions and velocities).
- 3. Object level. The pose can be estimated using Gauss–Newton estimation, after matching sparse or dense flow fields.
- *Update model features.* Newly detected features from preprocessing are *back-projected* onto the model at the pose estimated.

Figure 4.66 Optical flow modality.

can be achieved using the *inverse-compositional* approach proposed by Baker and Matthews [30], which we describe in the next section, or by the efficient second-order minimization approach (ESM) [39]; the former saves significant computational time by *precomputing* image Jacobians, while the latter achieves higher convergence rates, due to a better approximation of the Hessian matrix, still without using second derivatives.

On the other hand, a *sparse* set of control points, sampled and back-projected over the model surface, can be used as a feature-level measurement. In this case the template descriptor is given by the gray-level value of each model point, and state-space derivatives can be computed by re-projection of the same points at different poses. In this approach, back-projection is required only when sampling the visible points, which, as for any modality, can be accomplished once per frame or even less, without affecting the precision significantly. When dealing with a large set of points, a more convenient technique is *subsampling*, selecting a different subset of points every time, for better convergence of the optimization algorithm, which, however, should be able to cope with a noisy cost function (also called *stochastic optimization*).

Both class of approaches can be improved further by building image and texture *pyramids* and by matching at multiple resolutions. If we compare this modality with the local keypoints, we notice how a larger class of objects which do not possess reliable features or which have a high curvature surface but nevertheless show an overall distinctive texture, can be localized successfully. The most notable example is given by human faces (Fig. 4.67): in fact, apart from small areas such as eyes, lips, and eyebrows, only a few keypoints, and very few internal edges, can be matched with good precision. A texture map may overcome these limitations by using the *entire* surface and its shading.

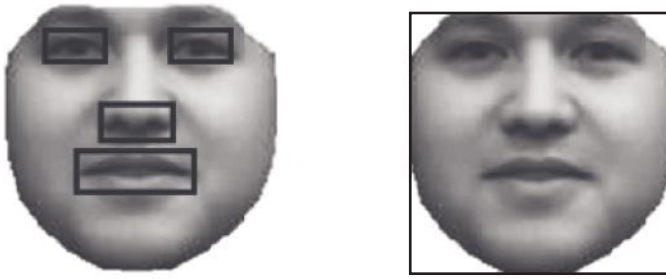


Figure 4.67 *Left: facial features for tracking; right: full template model.*

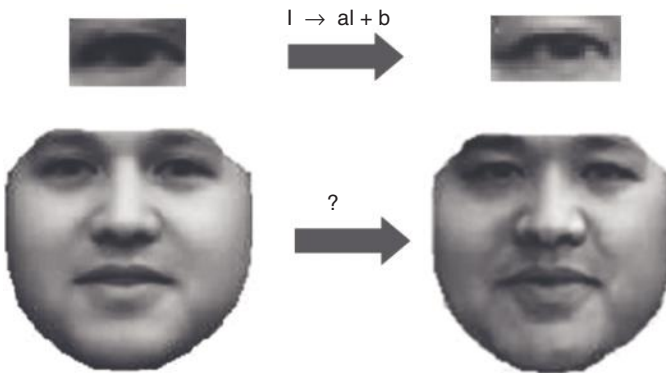


Figure 4.68 *Top: A local keypoint undergoes an approximately linear appearance variation. Bottom: For the global template, the shading variation is usually complex and nonlinear.*

However, templates also have their weaknesses; in fact, apart from a higher computational complexity, one notable problem is the overall appearance variation, due to illumination conditions. Generally speaking, all visual features are more or less affected by light, concerning both the local intensity and the incidence direction, which poses the most difficult challenges to the robustness of computer vision algorithms.

However, in most circumstances an illumination change results in locally smooth shading variations, with the exception of boundary or edge points. In particular, a local keypoint covers only a small area, which therefore has an approximately flat curvature; in this case, the pattern undergoes an approximately linear change (Fig. 4.68). In this case, the normalized cross-correlation index [eq. (4.137)] provides invariance to linear changes, and allows successful matching.

Unfortunately, the same cannot be said for the overall surface, which exhibits a complex and strongly nonlinear variation of shading under different incident light directions. To add robustness to these variations, several methods



SSD	40.5226	58.6129	66.5071	27.3357	25.3888
NCC	0.4365	0.3746	0.0540	-0.2037	0.2817
MI	0.6942	0.4384	0.3591	0.2466	0.1616

Figure 4.69 Comparison between three similarity measures for template matching in the presence of light and shading variations. *Top row*: base image and gray-level pattern variations due to different light or poses; *second row*: gray-level *cooccurrence* matrices of gray levels; *bottom*: corresponding SSD, NCC, and mutual information similarity measures. (From [134].)

have been considered in the literature, which we can classify briefly into two categories:

1. Replacing the LSE (or NCC) correlation with more robust similarity functions, able to cope to some extent with nonlinear shading variations
2. Explicitly modeling and estimating the shading by including *appearance parameters* in the state variables

The first class of methods includes, for example, mutual information [133,134], which has been used successfully for multimodal medical image alignment [111,159]. This index is theoretically able to enforce general *consistency* between the template values observed and the underlying image, thus also working in the presence of a highly nonlinear correlation. The existence of a relationship between two shading patterns on the same surface, assuming mild conditions on the reflectance properties, is also used in *photometric stereo* methods for surface reconstruction [167].

In Fig. 4.69 we can see a comparison of the three indices. Standard SSD is not robust to light variations, and NCC can only model a linear correlation between template and image. However, a strong but nonlinear dependence

can still be observed when the pose is constant, also under different conditions of illumination, for which MI shows more reliable and smoother behavior. This method is explained in more detail in Section 4.7.2.

The second class of methods includes the AAM (Section 2.2.2) [54,116,168] as well as nonlinear models of appearance variation, defined by local or global parameters [150]. Linear appearance models are provided by a basis image and a set of variation *modes*, obtained from a general training method such as the PCA, from a set of annotated pictures. In this case, usually both the shape and appearance models are trained on the same sequence, and their parameters form an *augmented state* vector that may include *coupling* effects. In fact, a shape deformation cannot be completely independent of the shading variation observed.

Figure 4.70 summarizes the modality abstraction for texture templates. In the following, we consider the methods described above in more detail.

4.7.1 Pose Estimation with AAM

With an active shape and appearance model, we can track by estimating the AAM *state* for the camera image at time k , I_k , where the state is defined by $\mathbf{s}_k^{\text{AAM}} \equiv (\mathbf{p}_k, \mathbf{a}_k)$, including both pose and appearance parameters, with a total of $N_p + N_a$ degrees of freedom. Figure 4.71 shows an example of joint estimation for a two-dimensional piecewise affine model. As we can see from the right side of the figure, if a corresponding three-dimensional mesh is available,

- *Preprocessing*. Usually nothing, because image data are used directly for matching. However, in a multiresolution context, building the image pyramid takes place as a preprocessing step.
- *Sampling model features*. For feature-level matching, the model is rendered at the average pose predicted, and template points are sampled from the image and back-projected in object space.
- *Data association*
 1. Pixel level. The image expected is obtained by rendering the model, and the residuals are computed pixel-wise. Image Jacobians may also be computed, by back-projecting all pixels in model space and then computing the re-projected derivatives; or by vertex interpolation using GPU shaders.
 2. Feature level. Sampled control points can be used for template matching with re-projection and Jacobian computation.
 3. Object level. A maximum-likelihood estimation procedure, possibly with robust indices such as a mutual information index or speed-ups such as the inverse-compositional approach, is used.
- *Update model features*. Online features can be sampled from image data and back-projected after Bayesian update.

Figure 4.70 Template matching modality.

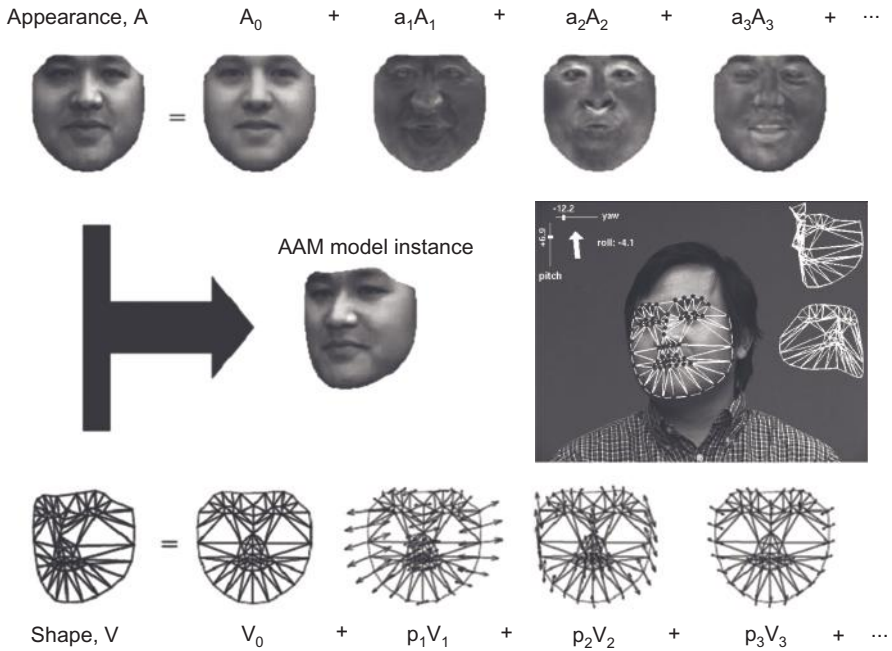


Figure 4.71 Template-based tracking with ASM/AAM. Pose and appearance parameters in the current image are estimated simultaneously by least-squares optimization. (From [116].) The result, as shown on the right, can be upgraded to a three-dimensional pose estimated by vertex correspondences. (From [168].)

the estimation result can be *upgraded* to a three-dimensional pose via LSE point correspondences or, in the presence of a dynamical model, with an extended Kalman filter [168].

By defining the expected appearance $A(\mathbf{x}; \mathbf{a})$, with \mathbf{x} a model point in two-dimensional,⁹ the state estimation problem can be formulated as a standard least-squares estimation (LSE), where the cost function is defined by

$$E(\mathbf{a}; \mathbf{p}) = \sum_{\mathbf{x}} \|I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A(\mathbf{x}; \mathbf{a})\|^2 \quad (4.119)$$

which is a pixel-level residual of gray values. The sum is performed over all pixels covered by the shape rendered at pose \mathbf{p} . In the case of partial occlusions, the influence of outliers can be reduced by using an M-estimator $\rho(\cdot)$ [such as eqs. (5.15) and (5.16)], so that

⁹In principle, the LK algorithm can also be applied to three-dimensional models, but not its fast variants. Therefore, here we concentrate on two-dimensional AAMs.

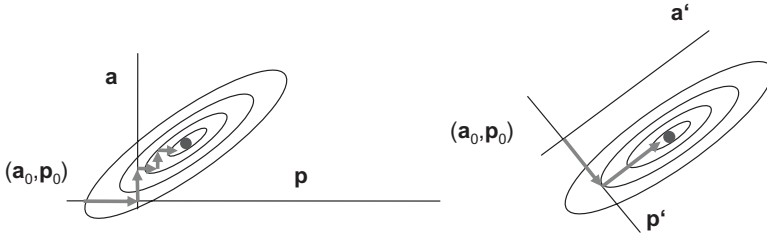


Figure 4.72 *Left:* Alternate optimization of pose and appearance parameters may lead to slow convergence. *Right:* By choosing the right orthogonal subspaces, optimization can be performed in two steps.

$$E(\mathbf{a}; \mathbf{p}) = \sum_{\mathbf{x}} \rho(\|I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A(\mathbf{x}; \mathbf{a})\|) \quad (4.120)$$

which is solved by a reweighted Gauss–Newton scheme where the weights are given by $w(\mathbf{x}) = \rho(\|e(\mathbf{x})\|) / \|e(\mathbf{x})\|^2$.

However, in both cases we notice that the LSE problem has the form

$$\mathbf{s}^* = \arg \min_{(\mathbf{a}; \mathbf{p})} E(\mathbf{a}; \mathbf{p}) \quad (4.121)$$

where the state parameters \mathbf{a} and \mathbf{p} are split between the two terms in eq. (4.119). A first, intuitive idea would be to perform the minimization alternately between \mathbf{a} and \mathbf{p} , by fixing one of them and updating the other via standard LSE. This approach has the potential drawback of requiring many search loops before converging to the joint optimum \mathbf{s}^* (Fig. 4.72, left side). However, for the particular case of AAM, by choosing the correct subspace decomposition we can achieve optimization in only two steps (Fig. 4.72, right side). Before dealing with this issue, we consider the pose and appearance optimization subproblems.

Pose estimation with a constant appearance A_0 is also known as the *Lucas–Kanade algorithm*, which we encountered in Section 4.6.2 when dealing with keypoint tracking for computing the sparse optical flow. Unlike local features that undergo simple transformations such as pure translation or affinity, in the case of a global template there are two additional issues to deal with: (1) a very large set of points \mathbf{x} , and (2) a large parameter space \mathbf{p} , with a complex warp. Therefore, to achieve real-time performance, the choice of optimization method must be made carefully.

Considering the second issue, the warp function in this case is a *piecewise affine* mapping of the type described in Section 2.2.1 (Fig. 2.15). Each triangle l of the base mesh is a *link* of a deformable structure, undergoing a local affine transform, with origin in one of the vertices $\mathbf{v}_{A(l)}$ and the two axes given by the sides of the triangle $\mathbf{v}_{B(l)} - \mathbf{v}_{A(l)}$, $\mathbf{v}_{C(l)} - \mathbf{v}_{A(l)}$. As we mentioned earlier, this

Iterate: at pose \mathbf{p} , do

1. Warp each template point \mathbf{x} , and get image gray values $I(\mathcal{W}(\mathbf{x}; \mathbf{p}))$.
2. Compute the error image $A_0(\mathbf{x}) - I(\mathcal{W}(\mathbf{x}; \mathbf{p}))$.
3. Also compute the warped image gradients $\nabla I(\mathcal{W}(\mathbf{x}; \mathbf{p}))$.
4. Evaluate the warp Jacobians $\partial \mathcal{W} / \partial \mathbf{p}$ at $(\mathbf{x}; \mathbf{p})$.
5. Compute the *steepest-descent* images, $\nabla I \cdot \partial \mathcal{W} / \partial \mathbf{p}$.
6. Accumulate the Gauss–Newton matrix:

$$G(\mathbf{p}) = \sum_{\mathbf{x}} \left(\nabla I \cdot \frac{\partial \mathcal{W}}{\partial \mathbf{p}} \right)^T \left(\nabla I \cdot \frac{\partial \mathcal{W}}{\partial \mathbf{p}} \right)$$

7. Accumulate the error gradient:

$$\mathbf{g} = \sum_{\mathbf{x}} \left(\nabla I \cdot \frac{\partial \mathcal{W}}{\partial \mathbf{p}} \right)^T [A_0(\mathbf{x}) - I(\mathcal{W}(\mathbf{x}; \mathbf{p}))]$$

8. Compute the pose update, $\delta \mathbf{p} = G^{-1} \mathbf{g}$.
9. Update the pose $\mathbf{p} \rightarrow \mathbf{p} + \delta \mathbf{p}$ until $\|\delta \mathbf{p}\| < \varepsilon$.

Figure 4.73 Lucas–Kanade algorithm for template matching.

model is linear in \mathbf{p} and therefore computationally convenient for screen re-projection and Jacobians.

The pose estimation problem,

$$\mathbf{p}^* = \arg \min_{\mathbf{p}} \sum_{\mathbf{x}} \|A_0(\mathbf{x}) - I(\mathcal{W}(\mathbf{x}; \mathbf{p}))\|^2 \quad (4.122)$$

is, however, nonlinear because of the image pattern $I(\mathbf{y})$, and can be dealt with by means of standard Gauss–Newton optimization, where the error is linearized at each pose starting from the initial guess \mathbf{p}_0 , leading to the Lucas–Kanade algorithm of Fig. 4.73.

In particular, steps 6 and 7 obtain the Gauss–Newton matrix G and the gradient \mathbf{g} of the cost function, respectively, which are used to update the pose. Steps 3 and 4 result from the linearization of the error function (4.119) around a given pose \mathbf{p} :

$$E(\mathbf{p} + \delta \mathbf{p}) \approx \sum_{\mathbf{x}} \left\| I(\mathcal{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathcal{W}}{\partial \mathbf{p}} \delta \mathbf{p} - A_0(\mathbf{x}) \right\|^2 \quad (4.123)$$

The warp Jacobian is computed at $(\mathbf{x}; \mathbf{p})$, and the term $\nabla I(\mathcal{W}(\mathbf{x}; \mathbf{p}))$ is given by first computing the image gradient ∇I in its own frame, and subsequently, warping this image according to $\mathcal{W}(\mathbf{x}; \mathbf{p})$.

In the formulation above, we can see how every step must be repeated at each pose hypothesis, leading to a high computational complexity because of the very large data set \mathbf{x} . To cope with this complexity, several approaches have been proposed in the literature (e.g., [31,39,70]). For the sake of simplicity we consider here two basic variants of the Gauss–Newton algorithm: the compositional and inverse-compositional approaches, the latter providing a high-speed improvement.

Following Baker and Matthews [31], the standard Lucas–Kanade method may also be called *forwards-additive*:

- *Forwards*. The direct warp function $\mathcal{W}(\mathbf{x}; \mathbf{p})$ is used while formulating the cost function [eq. (4.119)].
- *Additive*. The Gauss–Newton update $\delta \mathbf{p}$ is *added* to the previous pose (step 9 of the algorithm).

The first modification, which we discussed in Section 2.2.1 [see eq. (2.32)] for the case of homogeneous transforms, consists of substituting the additive rule for the compositional rule:

$$E(\mathbf{p}) = \sum_{\mathbf{x}} \|I(\mathcal{W}(\mathcal{W}(\mathbf{x}; \delta \mathbf{p}); \mathbf{p})) - A_0(\mathbf{x})\|^2 \quad (4.124)$$

where the warp is *composed* with itself,¹⁰ so that the meaning of $\delta \mathbf{p}$ at iteration k is

$$\mathcal{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathcal{W}(\mathbf{x}; \mathbf{p}) \circ \mathcal{W}(\mathbf{x}; \delta \mathbf{p}) \quad (4.125)$$

which means that after computing the increment $\delta \mathbf{p}$, we need to find the new value for \mathbf{p} satisfying eq. (4.125) for any \mathbf{x} . As we have seen, composition amounts in most cases to a *product* of transformation matrices, and it can be employed only if the manifold has the *semigroup property* (i.e., the warp is closed under matrix product) and if at $\mathbf{p} = \mathbf{0}$ we have the *identity warp*, $\mathcal{W}(\mathbf{x}; \mathbf{0}) = \mathbf{x}$.

With this definition one can easily verify that the error linearization becomes

$$E(\mathbf{p} + \delta \mathbf{p}) = \sum_{\mathbf{x}} \left\| I(\mathcal{W}(\mathbf{x}; \mathbf{p})) + \nabla I(\mathcal{W}) \frac{\partial \mathcal{W}}{\partial \mathbf{p}} \bigg|_{(\mathbf{x}; \mathbf{0})} \delta \mathbf{p} - A_0(\mathbf{x}) \right\|^2 \quad (4.126)$$

where the two principal differences with the original formulation are the use of $\nabla I(\mathcal{W})$, which is the gradient of the warped image $I(\mathcal{W}(\mathbf{x}; \mathbf{p}))$, where the warp is applied to the image *before* computing its gradient, and computation

¹⁰Notice how our former definition does not require the composition of the warp with itself, but more in general with any other parametrization of the same manifold.

of the warp Jacobian $\partial\mathcal{W}/\partial\mathbf{p}$ in $(\mathbf{x}; \mathbf{0})$, which does not depend on the pose and therefore can be computed off-line. However, the computational cost is almost the same as that of the original formulation, because the most expensive steps remain computation of the Hessian and steepest-descent images, which have to be repeated at each pose. Concerning convergence properties, it has been shown [31] that the two approximations are equivalent to the first order if, in addition, the warp is invertible, that is, the transformation manifold has a *group* structure.

In the latter case, the Lucas–Kanade algorithm can be improved dramatically by using the *inverse-compositional formulation*, where the inverse of the incremental warp is used to perform the update

$$\mathcal{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathcal{W}(\mathbf{x}; \mathbf{p}) \circ \mathcal{W}(\mathbf{x}; \delta\mathbf{p})^{-1} \quad (4.127)$$

which corresponds to the error function

$$E(\mathbf{p}) = \sum_{\mathbf{x}} \|A_0(\mathcal{W}(\mathbf{x}; \delta\mathbf{p})) - I(\mathcal{W}(\mathbf{x}; \mathbf{p}))\|^2 \quad (4.128)$$

where the roles of the template and image are exchanged, since the pose increment $\delta\mathbf{p}$ is on the side of the template A_0 . Therefore, the linearization becomes

$$E(\mathbf{p} + \delta\mathbf{p}) = \sum_{\mathbf{x}} \left\| A_0(\mathcal{W}(\mathbf{x}; \mathbf{0})) + \nabla A_0 \frac{\partial\mathcal{W}}{\partial\mathbf{p}} \Big|_{(\mathbf{x}; \mathbf{0})} \delta\mathbf{p} - I(\mathcal{W}(\mathbf{x}; \mathbf{p})) \right\|^2 \quad (4.129)$$

so that now the Gauss–Newton update $\delta\mathbf{p} = G^{-1}\mathbf{g}$ is given by

$$G = \sum_{\mathbf{x}} \left(\nabla A_0 \frac{\partial\mathcal{W}}{\partial\mathbf{p}} \right)^T \left(\nabla A_0 \frac{\partial\mathcal{W}}{\partial\mathbf{p}} \right) \quad (4.130)$$

$$\mathbf{g} = \sum_{\mathbf{x}} \left(\nabla A_0 \frac{\partial\mathcal{W}}{\partial\mathbf{p}} \right)^T [I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})] \quad (4.131)$$

where all terms are evaluated in $(\mathbf{x}; \mathbf{0})$ apart from the error image¹¹ $[I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})]$, which depends on \mathbf{p} . This allows us to move the most expensive computation off-line, as outlined in Fig. 4.74.

The equivalence of inverse-compositional formulation with the other two approximations has been shown by Baker and Matthews [31] up to the first-order approximation in $\delta\mathbf{p}$, as well as its convergence properties from experimental results. Therefore, whenever applicable this algorithm is highly

¹¹Notice the opposite sign with respect to the forwards algorithm.

Off-line

1. Compute the template gradient ∇A_0 at each point \mathbf{x} .
2. Compute the Jacobian $\partial \mathcal{W} / \partial \mathbf{p}$ at $(\mathbf{x}; \mathbf{0})$.
3. Compute the steepest-descent images, $\nabla A_0 \partial \mathcal{W} / \partial \mathbf{p}$.
4. Compute the Gauss–Newton matrix G from eq. (4.130).

Iterate. At pose \mathbf{p} , do

1. Warp each template point \mathbf{x} and get image gray values $I(\mathcal{W}(\mathbf{x}; \mathbf{p}))$.
2. Compute the error image $I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})$.
3. Accumulate the error gradient \mathbf{g} from eq. (4.131).
4. Compute the pose update $\delta \mathbf{p} = G^{-1} \mathbf{g}$.
5. Update the warp with the inverse-compositional rule $\mathcal{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathcal{W}(\mathbf{x}; \mathbf{p}) \circ \mathcal{W}(\mathbf{x}; \delta \mathbf{p})^{-1}$ until $\|\delta \mathbf{p}\| < \varepsilon$.

Figure 4.74 Inverse-compositional formulation of the Lucas–Kanade algorithm.

recommended for its computational efficiency, leading to more than real-time performances. In particular, the ideas above have been applied to face tracking by Matthews and Baker [116].

Subsequently, appearance can be estimated using eq. (4.119), and unlike the pose estimation, it is a linear problem in the respective parameter \mathbf{a} , which can be solved in one step. However, to solve the entire problem in only two steps, without need of further iterations, we need to reparametrize the problem by means of the appearance *subspace projection method* described below.

By considering the complete minimization problem, we can write

$$\mathbf{s}^* = \arg \min_{(\mathbf{a}; \mathbf{p})} \sum_{\mathbf{x}} \|I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A(\mathbf{x}; \mathbf{a})\|^2 = \arg \min_{(\mathbf{a}; \mathbf{p})} \left\| \mathbf{A}_0 + \sum_i a_i \mathbf{A}_i - \mathbf{I}(\mathbf{p}) \right\|^2 \quad (4.132)$$

where the last term contains all images in vectorized form, with length equal to the number of template points \mathbf{x} , and $\mathbf{I}(\mathbf{p})$ is the warped image at pose \mathbf{p} . We now consider the space $\text{span}(A_i)$ spanned by the appearance variations A_i , such that after removing the average A_0 , each valid appearance according to the AAM model belongs to this space as well as its orthogonal complement, $\text{span}(A_i)^\perp$. Then we can decompose the error vector into orthogonal components by re-projecting it onto the two subspaces, and write

$$\|\mathbf{E}(\mathbf{a}; \mathbf{p})\|^2 = \left\| \mathbf{A}_0 + \sum_i a_i \mathbf{A}_i - \mathbf{I}(\mathbf{p}) \right\|_{\text{span}(A_i)}^2 + \left\| \mathbf{A}_0 + \sum_i a_i \mathbf{A}_i - \mathbf{I}(\mathbf{p}) \right\|_{\text{span}(A_i)^\perp}^2 \quad (4.133)$$

In the second term, the A_i disappear because they belong to the other subspace,

$$\|\mathbf{E}(\mathbf{a}; \mathbf{p})\|^2 = \left\| \mathbf{A}_0 + \sum_i a_i \mathbf{A}_i - \mathbf{I}(\mathbf{p}) \right\|_{\text{span}(\mathbf{A}_i)}^2 + \|\mathbf{A}_0 - \mathbf{I}(\mathbf{p})\|_{\text{span}(\mathbf{A}_i)^\perp}^2 \quad (4.134)$$

so that it has no dependence on the appearance parameters \mathbf{a} . The first term, $\mathbf{E}_{\text{span}}(\mathbf{a}, \mathbf{p})$, still depends on both parameters, but it has a global optimum, $\mathbf{E}_{\text{span}}(\mathbf{a}^*, \mathbf{p}) = \mathbf{0}$, for *any* pose \mathbf{p} and a corresponding value \mathbf{a}^* that can be found in one step by solving the linear LSE

$$a_i^* = \sum_{\mathbf{x}} A_i(\mathbf{x}) [I(\mathcal{W}(\mathbf{x}; \mathbf{p})) - A_0(\mathbf{x})], \quad i = 1, \dots, N_a \quad (4.135)$$

which in vector form is the scalar product of the respective appearance with the error image, $a_i^* = \mathbf{A}_i \cdot [\mathbf{I}(\mathbf{p}) - \mathbf{A}_0]$.

Therefore, with this decomposition the full LSE optimization can be performed in two steps:

1. Minimize the error projected onto $\text{span}(\mathbf{A}_i)^\perp$ using the Lucas–Kanade algorithm or any of its variants.
2. Solve for the appearance parameters using eq. (4.135).

The former step simply amounts to projecting both the error image *and* the steepest-descent images onto $\text{span}(\mathbf{A}_i)^\perp$ before performing the Gauss–Newton updates; therefore, it comes at a reasonable computational cost.

4.7.2 Pose Estimation with Mutual Information

An alternative approach for robust template matching is to substitute for the cost function more general correlation or similarity indices, able to cope with nonlinear shading variations as well as outliers. In this way, the appearance model can be kept simple by using only one reference texture with a more or less arbitrary lighting, while the similarity function still exhibits a local maximum at the correct pose.

To achieve this degree of generality, we reconsider the example in Fig. 4.69: The first index is the standard SSD, which is the sum of differences between the gray values expected and those observed, at the re-projected pixel locations, corresponding to the observation model

$$\min_{(z_i, h_i)} \sum_i (z_i - h_i)^2 = 0 \Leftrightarrow z = h \quad (4.136)$$

where z is corrupted by a Gaussian zero-mean independent noise variable. The NCC index, instead, provides the maximum response when

$$\max_{(z_i, h_i)} \sum_i \frac{(z_i - \bar{z})(h_i - \bar{h})}{\sigma_z \sigma_h} = 1 \Leftrightarrow z = ah + b \quad (4.137)$$

where \bar{z} and σ_z are the mean and standard deviation of z_i , and similarly for h , which holds for any values of a and b , therefore enforcing any linear relationship, such as global contrast and brightness variation. Going beyond linear relationships, mutual information (MI) [56] allows *any* functional relationship f that may exist between two different shadings of the same surface:

$$\max_{(z_i, h_i)} \text{MI}[(z_1, h_1), \dots, (z_n, h_n)] = M \Leftrightarrow z = f(h) \quad (4.138)$$

where M depends on the dimension of the joint probability distribution $P(z, h)$. This index can be defined, in a statistical sense, as a *dependency* between the two variables; in fact, if a relationship holds, then $P(z|h)$ is a Dirac.¹² In the case of discrete variables such as quantized gray values, MI is given by

$$\text{MI}(z; h) = \sum_{z=0}^{255} \sum_{h=0}^{255} P(z, h) \log \frac{P(z, h)}{P(z)P(h)} \quad (4.139)$$

where h and z are in the range $[0, 255]$, $P(z, h)$ is their joint probability of occurrence, and $P(z) = \sum_h P(z, h)$ and $P(h) = \sum_z P(z, h)$ are the marginal probabilities, both obtained from the joint distribution.

The meaning of MI is, basically, the amount of *information* that one variable provides about the other, and of course it is defined in a symmetric way, $\text{MI}(z; h) = \text{MI}(h; z)$. In fact, it can be written as

$$\text{MI}(z; h) = H(z) - H(z|h) \quad (4.140)$$

which is the difference between the marginal *entropy* $H(z)$ and the conditional entropy $H(z|h)$, where the entropy of a random variable

$$H(z) = - \sum_z P(z) \log P(z) \quad (4.141)$$

is precisely the amount of *uncertainty* about its outcome in a single random trial, and it is maximum when the variable is distributed uniformly, $P(z) \propto c$. Instead, the conditional entropy is the uncertainty about z *after* knowing the outcome of h on which z may depend, and of course, it cannot be higher than the prior entropy. Therefore, we have two extreme cases:

¹²This relationship may also be one to many and in any case has a *finite* number of possible z for a given h (i.e., a sum of Dirac distributions).

- If z and h have no dependence, knowledge of h does not reduce the uncertainty about z , and $\text{MI}(z; h) = 0$.
- If z and h have complete dependence (i.e., knowing h tells everything about z), $\text{MI}(z; h)$ is maximum.

Another way to see why MI measures the dependency between two variables is given by comparing the joint probability $P(z, h)$ with the product of marginals $P(z)P(h)$. Only for independent variables will they be equal, and, in fact, MI measures the *distance* between these distributions by means of the *Kullback–Leibler divergence*:

$$\text{MI} = D_{KL}(P(z, h) \| P(z)P(h)) \quad (4.142)$$

where D_{KL} is defined by

$$D_{KL}(P(x) \| Q(x)) \equiv \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)} \quad (4.143)$$

and $x = (z, h)$ is the variable of interest. Obviously, $D_{KL} = 0$ iff $P(x) = Q(x)$, $\forall x$.

As we can see, computing MI requires two steps instead of the single sum of SSD or NCC:

1. Estimating the joint distribution $P(z, h)$
2. Computing MI through eq. (4.139)

and for many optimization approaches, we also need to compute derivatives of this index with respect to the incremental pose parameters.

Computing the joint distribution can be done in several ways, and here we refer to the method [134] where the estimation is carried out by means of histograms with a *fuzzy binning procedure*. Basically, the procedure consists of collecting all gray-value pairs (z, h) into a two-dimensional histogram in which the contribution of each pair is distributed among neighboring bins as a fuzzy assignment:

$$P(c_z + i, c_h + j) = P(c_z + i, c_h + j) + K(z - b(c_z + i))K(h - b(c_h + j)) \quad (4.144)$$

where (c_z, c_h) is the bin to which (z, h) belongs, b is the square bin size, and K is a monodimensional kernel that weights the contribution of the pair to the cell and its neighbors. The integer indices i and j usually run within a small interval: for example, $[-2, 2]$.

If the kernel is differentiable, we have a differentiable function $P(c_z, c_h)$, with respect to each contributing pair (z, h) , influencing the main bin as well as its neighbors. Since each pair is, in turn, a function of the pose parameters, through the warp Jacobian and the image gradients, we can say that each cell

of the histogram has a dependence on \mathbf{p} accumulated over all sample pairs. Therefore, the Jacobian $\partial P / \partial p_d(c_z, c_h)$ is also a set of cells containing the derivatives of each histogram cell with respect to that pose parameter.

By considering this dependence while differentiating MI, we have

$$\frac{\partial \text{MI}}{\partial p_d} = \sum_{c_z} \sum_{c_h} \left. \frac{\partial P}{\partial p_d} \right|_{(c_z, c_h)} \log \frac{P(c_z, c_h)}{P(c_z)} \quad (4.145)$$

Since MI is not a sum of squares, the Gauss–Newton matrix cannot be computed directly; however, a similar approximation to the Hessian, containing only first-order derivatives, is given by

$$\frac{\partial^2 \text{MI}}{\partial p_{d_1} \partial p_{d_2}} = \sum_{c_z} \sum_{c_h} \left[\frac{1}{P} \frac{\partial P}{\partial p_{d_1}} \frac{\partial P}{\partial p_{d_2}} \right]_{(c_z, c_h)} - \sum_{c_z} \left[\frac{1}{P_z} \frac{\partial P_z}{\partial p_{d_1}} \frac{\partial P_z}{\partial p_{d_2}} \right]_{c_z} \quad (4.146)$$

where the Jacobian of the marginal distribution $\partial P_z / \partial p_d$ is obtained by marginalization of the joint distribution Jacobian (i.e., by summation over its columns).

Figure 4.75 shows some results for three-dimensional object tracking, where only one texture model has been used throughout the entire sequence, thus showing the robustness of MI with respect to light shading and partial occlusion. This approach can also be implemented in a multiresolution manner by building a Gaussian pyramid and operating Levenberg–Marquardt optimization as usual, from coarse to fine resolution. More details about the entire procedure, as well as experimental results, may be found in the literature (e.g., [134]).



Figure 4.75 Mutual information for three-dimensional object tracking is a robust index with respect to light and shading variations as well as partial occlusion. (From [134].)

CHAPTER 5

RECURSIVE STATE-SPACE ESTIMATION

The main “intelligence” within a tracking pipeline is provided by recursive state estimation, which predicts the current target distribution, or *belief*, based on the respective dynamics (Section 2.4), and using the respective processing tree, updates the posterior distribution on the basis of the associated measurements. In this chapter we provide examples of multitarget tracking schemes. All of them make use of target-associated and integrated measurements $Z = \langle z, h, H, e, R \rangle_X$.

Apart from the MLE and MAP estimators of Section 5.2, the general scheme behind this process is the *Bayesian tracking equation* [42,152], updating the multitarget distribution at time t_k according to

$$P(\mathbf{s}_k | \mathcal{Z}_k) \propto P(Z_k | \mathbf{s}_k) \int_{\mathbf{s}_{k-1}} P(\mathbf{s}_k | \mathbf{s}_{k-1}) P(\mathbf{s}_{k-1} | \mathcal{Z}_{k-1}) \quad (5.1)$$

where the ensemble state statistics \mathbf{s}_k are updated by integrating the current measurements $Z_k = (Z_k^1, \dots, Z_k^O)$ associated with all targets, together with the set \mathcal{Z}_{k-1} of all measurements up to $k-1$, all included into the last posterior distribution, \mathbf{s}_{k-1} , through state dynamics and measurement likelihood.

In most cases, this scheme is split into O independent single-target trackers,

$$P(s_k^o | Z_k^o) \propto P(Z_k^o | s_k^o) \int_{s_{k-1}^o} P(s_k^o | s_{k-1}^o) P(s_{k-1}^o | Z_{k-1}^o) \quad (5.2)$$

which also assumes independent dynamics for each target, as well as independent measurements Z_k^o , assigned with certainty to each target by the matching procedures.

The Bayesian equation is evaluated in two steps:

1. *Prediction* (Kolmogorov–Chapman equation):

$$P(s_k | Z_{k-1}) = \int_{s_{k-1}} P(s_k | s_{k-1}) P(s_{k-1} | Z_{k-1}) \quad (5.3)$$

2. *Correction* (Bayes' rule):

$$P(s_k | Z_k) \propto P(Z_k | s_k) P(s_k | Z_{k-1}) \quad (5.4)$$

which specialize to such well-known sequential estimators as Kalman or particle filters, when different assumptions are made concerning the form of dynamics and likelihood as well as the overall representation for $P(s_k | Z_k)$. These choices can, in turn, be made on the basis of available models, visual modalities employed, and measurement abstraction level.

5.1 TARGET-STATE DISTRIBUTION

To carry out object localization with Bayesian models, we first require a probabilistic representation for the state distribution. In Fig. 5.1 we show several ways to represent a single-target probability density, which can be extended to multiple targets by considering the *ensemble state*, which is initialized and propagated by object detection and tracking, respectively. In particular, an object detector provides the *initial prior* for new targets, whereas a Bayesian tracker *propagates* the posterior density by means prediction and correction, via the intermediate *prior* density.

From top to bottom and from left to right we have:

1. *Maximum a posteriori* (MAP). The globally or locally highest peak of the underlying density is estimated at each time point; this corresponds to a single-hypothesis discrete distribution (i.e., a Dirac). In this case, no Bayesian equation needs to be solved, since the MAP estimate is simply an object-level measurement, integrating the dynamical prior during the likelihood optimization.¹ If no prior is used, we have a

¹Notice, however, that the prior distribution is *not* a Dirac, in consequence of the motion covariance.

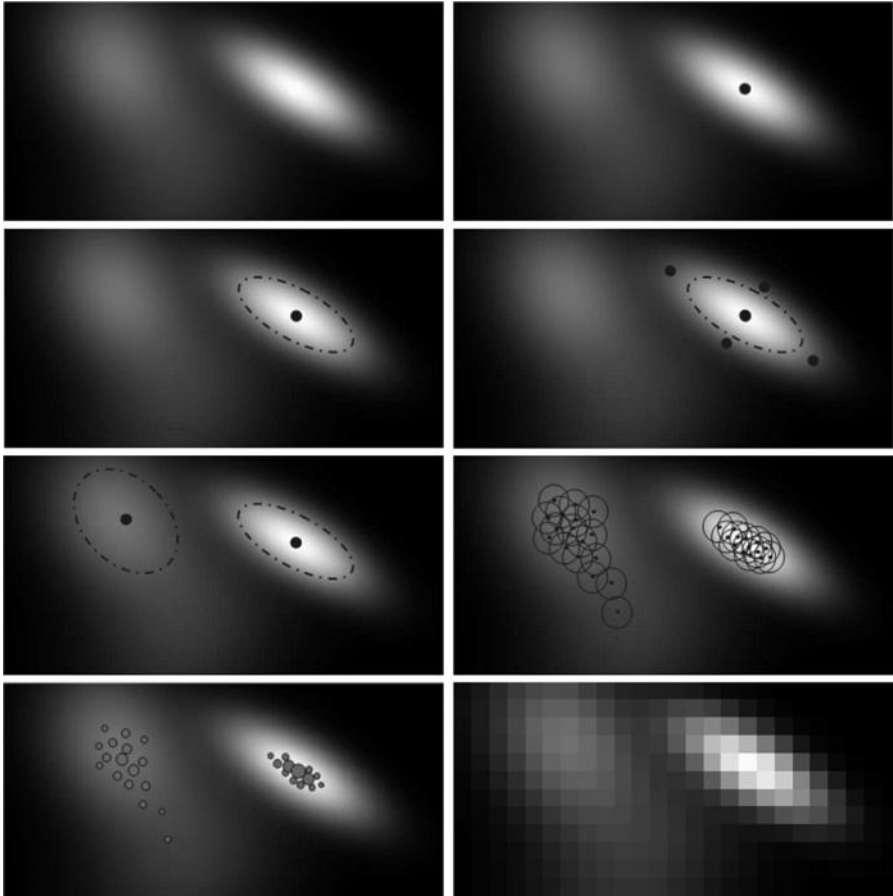


Figure 5.1 Modeling the target distribution in state space for Bayesian localization. *From first row, left to right:* underlying posterior density; MAP estimation; Gaussian density with mean and covariance; Gaussian density with sigma points; mixture of Gaussians; kernel density; weighted particle set; state-space grid.

maximum-likelihood estimation (MLE), which involves relying entirely on the current data while neglecting all of the history.

2. *Gaussian distribution.* Here a unimodal Gaussian approximates the target belief by covering the main peak while neglecting the remaining areas; this corresponds to a Kalman filter or, for nonlinear systems, an extended Kalman filter formulation.
3. *Sigma points.* For nonlinear systems, alternatively, a set of $(2n + 1)$ sigma points, where n is the state dimension, can be used to approximate a Gaussian via the *unscented transform* (Section 5.3.3). This corresponds to an unscented Kalman filter (UKF).

4. *Mixture of Gaussians*. A mixture model consists of a weighted sum of unimodal components placed onto the main peaks of the underlying distribution with the goal of providing more robustness, by not discarding possible areas of the state space where the target can be found. However, as can be seen from the literature (e.g., [79]), implementation of Bayesian filtering in this case is very complex, due to the necessary updating of all parameters, including the weights and the number of components, so that no unique solution has been proposed for general nonlinear models.
5. *Kernel density*. Using a large set of components, each with the same covariance, gives the *kernel* representation, which has been used elsewhere for Bayesian filtering (e.g., [73]). In this framework, the number of components as well as their covariance is fixed and only the mean values are updated; however, due to the large number of terms, the resulting probability density function can have an arbitrary number and shape of *modes*, according to the local density of the individual components. This is in contrast to Gaussian mixtures, where each component accounts explicitly for a single mode of the underlying distribution.
6. *Particle set*. Another flexible and popular representation is provided by Monte Carlo sampling in state space: A set of discrete weighted hypotheses, obtained through an *importance sampling* technique, represents the underlying probability density function as a large set of Diracs.
7. *State-space grids*. In some cases it is more convenient to work in a *quantized* state space, where the probability distribution is evaluated and updated globally for a given cell, and propagated to the neighbors during prediction. This corresponds to a Eulerian approach in state space, also known as *grid filters* [151] or *histogram filters* [153]; a related approach, called a *Bayesian occupancy filter*, was described by Mekhnacha et al. [117].

Consequently, we can define a *target* as a set consisting of:

- A vector of states, representing possibly multiple components (or *hypotheses*) of the target distribution, which are updated by the Bayesian filter during prediction and correction
- A vector of scalar *weights* and/or *covariance matrices*, representing the likelihood or the uncertainty of each hypothesis, respectively
- A pointer to its model information, including shape, appearance, degrees of freedom, and dynamics, which may be *shared* among similar targets: for example, a generic person model for tracking multiple people
- A unique identification number (*label*) that keeps its identity throughout the track lifetime (i.e., after detection and before a track loss occurs)
- The time stamp t_k of the most recent update, to be compared with the current measurement time stamp for prediction
- The overall output statistics, usually given by global average and covariance matrix of the distribution

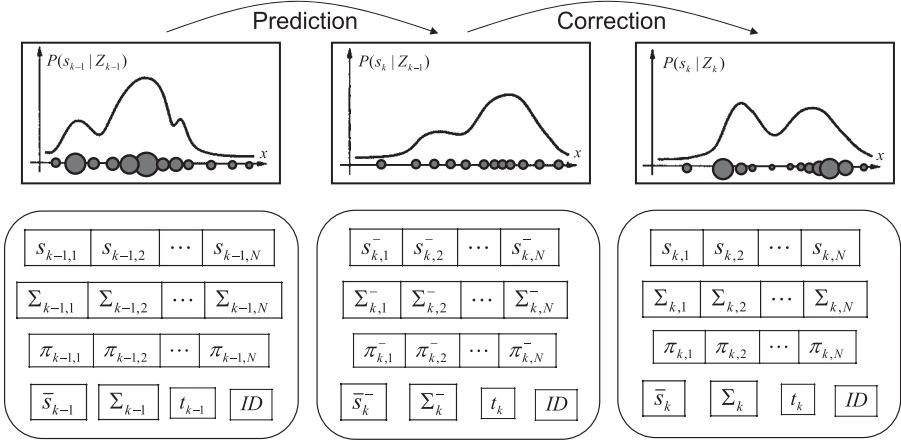


Figure 5.2 The target class contains all information needed to represent and propagate the state distribution in a Bayesian framework.

With this structure we can implement any type of Bayesian filter; for example, a Kalman filter requires a single-state hypothesis with covariance matrix, which is also its global output statistics, whereas a particle filter or an unscented Kalman filter requires multiple discrete hypotheses and the related weights (Fig. 5.2).

For parallel implementations, the internal state distribution may also be split among different processing *threads*, so that only a subset of them will be accessed and updated by the corresponding thread, while the full distribution will be available at the end, for computing the output statistics.

5.2 MLE AND MAP ESTIMATION

We first consider the case of a single-hypothesis estimation, obtained by maximizing the multitarget likelihood function. In particular, we restrict our attention to the case of independent likelihoods; that is, at the end of the processing pipeline we have $P(Z_k^o | s_k^o)$, where Z_k^o is the measurement associated with target o . Therefore, the goal of maximum-likelihood estimation (MLE) is to find

$$s_k^{o*} = \arg \max_{s_k^o} P(Z_k^o | s_k^o) \quad (5.5)$$

This maximization must be performed only for the observable part of the state; for example, if the state contains temporal derivatives of the pose \mathbf{p}_k but Z does not depend on this variable, the maximum is computed over the pose \mathbf{p}_k .

only, while the velocity is left undetermined. This is a consequence of our choice of ignoring the dynamics (i.e., the prior density).

If we also include the prior, we have a Bayesian approach, maximizing the posterior probability (MAP),

$$s_k^{o*} = \arg \max_{s_k^o} P(Z_k^o | s_k^o) P(s_k^o | s_{k-1}^o) \quad (5.6)$$

where the dynamics $P(s_k^o | s_{k-1}^o)$ acts as a *regularization* term, which ensures a well-conditioned optimization even in the presence of an ill-conditioned likelihood, at the same time keeping the estimation of the full state.

The estimation problem is formulated more conveniently with respect to the negative *log likelihood* (or log posterior):

$$s_k^{o*} = \arg \min_{s_k^o} [-\log P(Z_k^o | s_k^o) - \log P(s_k^o | s_{k-1}^o)] \quad (5.7)$$

which for additive Gaussian noise models corresponds to a nonlinear least-squares estimation (LSE).

If residual Jacobians are not available in closed form (e.g., because of a high computational demand or nondifferentiable pixel-level measurements), a derivative-free algorithm can be used, such as the Nelder–Mead *simplex* [128]. This type of heuristic search usually performs well in lower dimensions, at the price of a generally high number of iterations for convergence. When derivatives are available, the LSE problem can be solved instead via Gauss–Newton estimation, as explained next.

5.2.1 Least-Squares Estimation

After data fusion, the measurement for a single target Z , dropping the index o , may contain residuals $e_{m,c}$ from different modalities, sensors, and possibly at different processing levels. If all of these residuals are Gaussians, with covariances $R_{m,c}$, the MLE problem (5.5) is equivalent to a weighted nonlinear LSE:

$$s_k^* = \arg \min_{s_k} \sum_{m,c} e_{k,m,c}^T R_{k,m,c}^{-1} e_{k,m,c} = \arg \min_{s_k} e_k^T R_k^{-1} e_k \quad (5.8)$$

where (e_k, R_k) is obtained by stacking the individual terms together, as in eq. (3.12).

Let us consider the most common case, involving optimization in pose space only where $e_k = e(\mathbf{p}_k)$: To optimize eq. (5.8), we can perform a *Gauss–Newton update*:

$$\delta \mathbf{p}_k^* = -G_k^{-1} \mathbf{g}_k \quad (5.9)$$

where

$$\begin{aligned} G_k &= H_k^T R_k^{-1} H_k \\ \mathbf{g}_k &= H_k^T R_k^{-1} e_k \end{aligned} \quad (5.10)$$

are the Hessian matrix and gradient vector, respectively, weighted by the inverse covariance R^{-1} , and H contains the residual Jacobians $\partial e(\mathbf{p}, \delta\mathbf{p})/\partial(\delta\mathbf{p})$ evaluated at $\delta\mathbf{p}=0$ (see Section 2.2.1). The resulting increment $\delta\mathbf{p}_k$ is then used for updating the state with the given update rule.

If the predictive prior for the pose $P(\mathbf{p}_k | \mathbf{p}_{k-1})$ is a Gaussian, with mean \mathbf{p}_k^- and covariance matrix Σ_k^- , it can be included in the MAP estimation:

$$\delta\mathbf{p}_k^* = \arg \min_{\delta\mathbf{p}_k} \left[\delta\mathbf{p}_k^{-T} (\Sigma_k^-)^{-1} \delta\mathbf{p}_k^- + e_k^T R_k^{-1} e_k \right] \quad (5.11)$$

where $\delta\mathbf{p}^-(\mathbf{p}^-, \mathbf{p})$ is the pose increment from \mathbf{p}_k^- to \mathbf{p}_k , which also depends on the update rule. For the additive case, $\delta\mathbf{p}^- = \mathbf{p} - \mathbf{p}_k^-$, whereas for the compositional case, it is computed from the matrix increment $\delta\mathbf{p}^- \leftrightarrow \delta T^- = (T^-)^{-1} T$. Therefore, the Gauss–Newton matrices (5.10) are modified to

$$\begin{aligned} G_k &= H_k^T R_k^{-1} H_k + (\Sigma_k^-)^{-1} \\ \mathbf{g}_k &= H_k^T R_k^{-1} e_k + (\Sigma_k^-)^{-1} \delta\mathbf{p}_k^- \end{aligned} \quad (5.12)$$

In both cases, the optimization procedure starts from the predicted state \mathbf{p}_k^- , which is simply the previous MAP (or MLE) estimate.

5.2.2 Robust Least-Squares Estimation

The well-known work of Levenberg [102] and Marquardt [103] shows how to improve robustness and speed of convergence for nonlinear LSE problems by modifying the Gauss–Newton matrix G with an additive term λI , a multiple of the identity matrix

$$\delta\mathbf{p}^* = -(G + \lambda I)^{-1} \mathbf{g} \quad (5.13)$$

where λ is a positive regularization factor, adjusted during optimization in relation to the local behavior of the function. In particular, the adaptation rule at iteration i is

$$\begin{aligned} \text{if } \mathcal{L}(\mathbf{p}^{(i+1)}) > \mathcal{L}(\mathbf{p}^{(i)}) \quad &\text{then } \lambda \rightarrow 0.1\lambda; \text{ accept } \delta\mathbf{p}^* \\ \text{else } \quad &\lambda \rightarrow 10\lambda; \text{ reject } \delta\mathbf{p}^* \end{aligned} \quad (5.14)$$

where \mathcal{L} is the log-likelihood (or posterior) function to be maximized and i is the iteration number. This method can be employed when the underlying cost function is deterministic; that is, a given point, \mathbf{p} , yields a constant value of \mathcal{L} , but this may be violated by *subsampling* procedures that select a random subset of measurements at each iteration, as well as by multiresolution approaches, whenever the resolution is increased.

Another issue for many practical cases is that standard LSE cannot cope with *outliers* arising from wrong data association; outliers are basically residuals that lie outside the standard noise statistics given by the respective measurement covariance, and that may largely influence the result of LSE, possibly making the estimation unstable. We recall here two basic methods of coping with this problem: M-estimators and RANSAC.

M-estimators are a popular way of dealing with outliers by introducing a more robust cost function over the standard SSD. This function is defined such as to approximate the squared error when the residual is small, otherwise growing slower than the square, for high residuals (Fig. 5.3); for example, the *Huber M-estimator*

$$\rho_{\text{Hub}}(e) = \begin{cases} \frac{1}{2}e^2, & |e| \leq \bar{e} \\ \bar{e}|e| - \frac{1}{2}\bar{e}^2, & |e| > \bar{e} \end{cases} \quad (5.15)$$

exhibits *linear* behavior after the threshold \bar{e} , while the *Tukey function*

$$\rho_{\text{Tuk}}(e) = \begin{cases} \frac{\bar{e}^2}{6} \left(1 - \left(1 - \frac{e^2}{\bar{e}^2} \right)^3 \right), & |e| \leq \bar{e} \\ \frac{\bar{e}^2}{6}, & |e| > \bar{e} \end{cases} \quad (5.16)$$

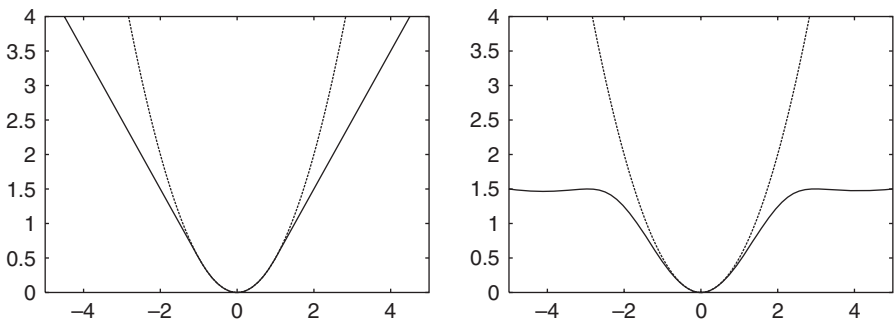


Figure 5.3 Two examples of an M-estimator for robust LSE: the Huber (*left*) and the Tukey (*right*) modified error functions.

becomes flat for $|e| > \bar{e}$, thus drastically removing outliers. However, the latter shows less robustness during Gauss–Newton optimization, since it is not convex and therefore reduces the convergence region. Therefore, ρ_{Hub} can be used for the first iterations, followed by the less outlier-sensitive estimator ρ_{Tuk} .

Another approach is to remove outliers *before* MLE (or MAP) optimization takes place by using the random sample consensus (RANSAC) method [65]. RANSAC explicitly seeks outliers by randomly sampling subsets of data, verifying the plausibility of the subset chosen by looking at the remaining inliers, and finally, retaining the subset that gives the best results.

Since this method performs multiple estimations with a minimal subset of points, its implementation depends strongly on the problem at hand: for example, the pose type; the feature type, such as points or lines; and the number of views available. However, when applicable it provides robust outlier rejection, as well as an initial estimate for subsequent MLE estimation using only the inliers.

The algorithm originally proposed by Fischler and Bolles [65] consists of the following steps. Given a set of n observations and a *model* to be fitted:

1. Select a random subset of m data, where m is the minimum number of data items necessary to estimate the pose parameters.
2. Estimate the model that fits this subset.
3. Compute the number of *inliers* for this estimate (i.e., data with residuals smaller than a threshold t).
4. If the number of inliers is higher than another threshold d , this is a good model; then, reestimate the pose using *all* the inliers and recompute the inlier residuals. If this error is the lowest error so far, keep the pose and the inliers as the “best case.”

After a given number of iterations k , the algorithm is stopped and the best model is retained. As we can see, this algorithm requires the choice of several parameters: the minimum amount of data for model fitting m , the number of iterations k , the threshold t for the error after fitting m points, and the minimum number of inliers d for a good model.

The thresholds t and d have to be determined from experimental evaluation, or from empirical considerations concerning the specific problem and data set.² Instead, the number of iterations k can be computed from a theoretical result. In fact, if P is the probability of selecting m inliers randomly from the data set, this is also the approximate probability of obtaining a good model. Moreover, if w is the expected percentage of inliers, $1 - w^m$ is the probability of selecting at least one outlier among the m points randomly, so that

$$1 - P = (1 - w^m)^k \quad (5.17)$$

²Although the threshold t may also be computed automatically, from the residual covariance at the given pose.

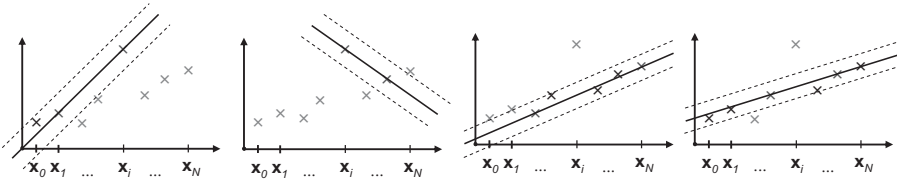


Figure 5.4 RANSAC algorithm applied to linear regression with outliers.

where both sides represent the probability of *never* selecting m inliers during all k iterations. If w is known roughly, we can compute the minimum number of iterations to achieve a desired P :

$$k = \frac{\log(1-P)}{\log(1-w^m)} \quad (5.18)$$

Figure 5.4 is a simple representation of the method applied to a linear regression problem; in this case, $m=2$ observations, each providing a constraint on the y value, are sufficient for estimating the model parameters, which are the slope and intercept of the line. The hypothesis on the right is the one with the maximum number of inliers and the minimum LSE error.

An example related to object tracking is also given by Fischler and Bolles [65] who fit a three-dimensional model with a Euclidean pose to correspondences between known object points and noisy image measurements, under a calibrated perspective camera. This problem is also known as *Perspective from n points* (PnP), and can be solved in closed form if we have a minimum number $m=3$ correspondences, each providing two constraints, known as the *P3P problem*.

In fact, as shown in Fig. 5.5, three correspondences define a triangle in space, so that the solution can be obtained by estimating the distances a, b, c of the three vertices A, B, C from the camera center given the angles between image

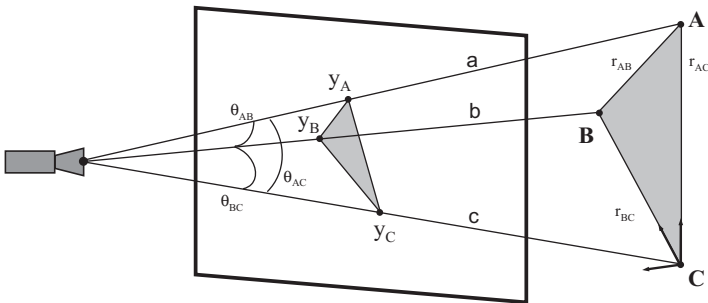


Figure 5.5 Perspective from the three points (P3P) problem.

rays $\theta_{AB}, \theta_{BC}, \theta_{AC}$ and the reciprocal distances in space r_{AB}, r_{BC}, r_{CA} . The rays can, in turn, be computed from the image positions $\mathbf{y}_A, \mathbf{y}_B, \mathbf{y}_C$ and the camera parameters. The solution, given in more detail by Fischler and Bolles consists basically of solving a system of three quadratic equations in three unknowns (a, b, c), which may have up to four solutions, to be evaluated separately.

After the initial pose estimation, the remaining points can be reprojected and tested against the corresponding data to look for inliers. In case of a good hypothesis, the refinement (step 4) is done by solving the general *PnP* by nonlinear LSE, starting from the initial estimate of the *P3P* algorithm and minimizing the overall re-projection error.

5.3 GAUSSIAN FILTERS

In many cases the state distribution at time k can be well approximated by a unimodal Gaussian $\mathcal{N}(s_k, \Sigma_k)$, represented by its mean vector and covariance matrix. The Bayesian prediction and correction steps propagate these two quantities through time, albeit with different implementations.

5.3.1 Kalman and Information Filters

We first consider the *linear + Gaussian* case, where both the dynamics and measurement models are given by

$$\begin{aligned} s_k &= F_k s_{k-1} + W_k w_k \\ z_k &= H_k s_k + V_k v_k \end{aligned} \quad (5.19)$$

and w_k and v_k are zero-mean white Gaussian noises with unit covariance, while W_k and V_k are the respective *gain* matrices; all of the quantities above can be time varying, which is especially true for predictions in the presence of asynchronous measurements, as we saw in Section 2.4. If the initial state is represented by a Gaussian $\mathcal{N}(s_0, \Sigma_0)$, the prior and posterior distributions will be Gaussian at all times, represented by $\mathcal{N}(s_k^-, \Sigma_k^-)$ and $\mathcal{N}(s_k, \Sigma_k)$, respectively.

The Kalman filter [89] is the optimal Bayesian tracker for this case [eq. (5.19)]. By defining the overall noise covariances

$$\begin{aligned} Q_k &= W_k W_k^T \\ R_k &= V_k V_k^T \end{aligned} \quad (5.20)$$

which may be rank deficient, if the dimension of the noise is less than the state (respectively, the measurement), the optimal predictor–corrector scheme [eqs. (5.3) and (5.4)] is given by

$$\begin{aligned} s_k^- &= F_k s_{k-1} \\ \Sigma_k^- &= F_k \Sigma_{k-1} F_k^T + Q_k \end{aligned} \quad (5.21)$$

and

$$\begin{aligned} s_k &= s_k^- + K_k (z_k - H_k s_k^-) \\ \Sigma_k &= (I - K_k H_k) \Sigma_k^- \end{aligned} \quad (5.22)$$

where

$$K_k = \Sigma_k^- H_k^T (H_k \Sigma_k^- H_k^T + R_k)^{-1} \quad (5.23)$$

is the *Kalman gain* and $S_k = H_k \Sigma_k^- H_k^T + R_k$ is the *innovation covariance*.

A dual formulation of the Kalman filter known as the *information filter* (IF) is especially useful for cases where the measurement dimension is very high compared to the state dimension: for example, in feature- and pixel-level measurements. This is accomplished by defining the dual variables

$$\begin{aligned} Y &\equiv \Sigma^{-1} \\ y &\equiv Y \cdot s \end{aligned} \quad (5.24)$$

called the *information matrix* and *information vector*, respectively. The corresponding Kalman filter has an updating equation where the gain is not computed explicitly, so there is no need for inversion of the innovation covariance.

In particular, the IF prediction is

$$\begin{aligned} Y_k^- &= (F_k Y_{k-1}^{-1} F_k^T + Q_k)^{-1} \\ y_k^- &= Y_k^- F_k Y_{k-1}^{-1} y_{k-1} \end{aligned} \quad (5.25)$$

which involves the inversion of the $n \times n$ state covariance, while the correction is

$$\begin{aligned} Y_k &= Y_k^- + H_k^T R_k^{-1} H_k \\ y_k &= y_k^- + H_k^T R_k^{-1} z_k \end{aligned} \quad (5.26)$$

which, instead, has an additive form for both the information matrix and vector, where inverting the measurement noise covariance R_t^{-1} is simpler because R is a diagonal matrix and in many cases is even constant in time. Moreover, this form allows *sequential* data fusion simply by adding the contribution of measurements one at a time, provided that they carry the same time stamp t_k .

5.3.2 Extended Kalman and Information Filters

If the state dynamics and/or the measurement model are nonlinear, the KF or IF schemes can no longer be employed; however, in many systems the Gaussian

assumption is approximately valid, and suboptimal Bayesian trackers can be implemented. We write the nonlinear system equations

$$\begin{aligned} s_k &= f_k(s_{k-1}, w_k, \tau_k) \\ z_k &= h_k(s_k, v_k) \end{aligned} \quad (5.27)$$

where one or both of f and h are nonlinear functions in one or both of their arguments. Here the state distributions will no longer be Gaussian, even if the distribution of process and measurement noises, w_k and v_k , are Gaussian, because of the nonlinearity.

In computer vision, nonlinearity is present especially in the measurement term $h(s)$, which, as we have seen, can be a very complex mapping from object to feature or image space. For this purpose, we consider two well-known filters, providing an approximate solution to the Bayesian tracking equations: the extended and unscented Kalman filter (EKF and UKF, respectively). The EKF is obtained by *linearizing* f and h with a first-order Taylor expansion around the respective mean states, which we indicate by \bar{s} :

$$\begin{aligned} \bar{s}_k^- &\equiv f_k(\bar{s}_{k-1}, 0, \tau_k) \\ s_k &\approx \bar{s}_k^- + F_k(s_{k-1} - \bar{s}_{k-1}) + W_k w_k \\ z_k &\approx h_k(\bar{s}_k^-, 0) + H_k(s_k - \bar{s}_k^-) + V_k v_k \end{aligned} \quad (5.28)$$

where

$$\begin{aligned} F_k &= \frac{\partial f_k}{\partial s}(\bar{s}_{k-1}, 0, \tau_k) \\ H_k &= \frac{\partial h_k}{\partial s}(\bar{s}_k^-, 0) \end{aligned} \quad (5.29)$$

are Jacobian matrices of f and h with respect to the state, and W_k and V_k are Jacobians with respect to the noise variables, computed for $w_k = 0$ and $v_k = 0$. If we also assume that w and v are additive noises, which is typical for most systems,

$$\begin{aligned} s_k &= f_k(s_{k-1}) + W_k w_k \\ z_k &= h_k(s_k) + V_k v_k \end{aligned} \quad (5.30)$$

then $Q_k = W_k W_k^T$ and $R_k = V_k V_k^T$ are the respective covariance matrices and, again dropping the \bar{s} , the EKF equations are given by prediction:

$$\begin{aligned} s_k^- &= f_k(s_{k-1}, 0, \tau_k) \\ \Sigma_k^- &= F_k \Sigma_{k-1} F_k^T + Q_k \end{aligned} \quad (5.31)$$

and correction:

$$\begin{aligned} s_k &= s_k^- + K_k(z_k - h_k(s_k^-, 0)) \\ \Sigma_k &= (I - K_k H_k) \Sigma_k^- \end{aligned} \quad (5.32)$$

where the Kalman gain is again given by eq. (5.23).

We notice how the EKF equations (5.31) and (5.32) reduce to (5.21) and (5.22) if f and h are linear, so that the KF can also be seen as a special case of the EKF. The dual formulation of the EKF is known as the extended information filter (EIF), and it is obtained in a similar way, by performing the same linearization and substituting the information vector and matrix for the state and covariance matrix, respectively (see also [153, Chap. 3.5]).

The EIF prediction becomes

$$\begin{aligned} s_{k-1} &= \Sigma_{k-1}^{-1} y_{k-1} \\ Y_k^- &= (F_k Y_{k-1}^{-1} F_k^T + Q_k)^{-1} \\ s_k^- &= f(s_{k-1}, 0, \tau_k) \\ y_k^- &= Y_k^- s_k^- \end{aligned} \quad (5.33)$$

and the correction is

$$\begin{aligned} Y_k &= Y_k^- + H_k^T R_k^{-1} H_k \\ y_k &= y_k^- + H_k^T R_k^{-1} (z_k - h(s_k^-, 0) - H_k s_k^-) \end{aligned} \quad (5.34)$$

Since the EKF and EIF perform a first-order linearization, the quality of the approximation is quite poor in many cases, possibly leading to unstable behavior. One way to cope with this difficulty is to *iterate* the prediction and/or correction, again performing linearization at the updated states.

In fact, under the Gaussian assumption, each Bayesian update is the solution of a sequential LSE problem where measurements are integrated frame by frame in order to update the posterior at k , as opposed to a *batch* LSE, where all of the measurement history \mathcal{Z}_k would be used at once, together with the initial prior $P(s_0)$, for the same estimation.

In a nonlinear context, the EKF therefore corresponds to a sequential NLSE, solved at each step k via a single Gauss–Newton step, obtained by the linearization formulas (5.31) and (5.32). Since Gauss–Newton is an iterative procedure, one can obtain a better approximation by repeating the two steps until a convergence criterion is met.

This idea leads to the iterated EKF (or IEKF), which can be implemented in different ways. One popular choice, in particular, is the algorithm of Fig. 5.6, where the iteration is done only in the mean-state update, while the posterior covariance is estimated at the end.

Prediction: Given the previous posterior (s_{k-1}, Σ_{k-1}) ,

$$\begin{aligned} s_k^- &= f_k(\bar{s}_{k-1}, 0, \tau_k) \\ \Sigma_k^- &= F_k \Sigma_{k-1} F_k^T + Q_k \end{aligned} \quad (5.35)$$

State update: Set $s_k^{(0)} = s_k^-$ and iterate:

$$\begin{aligned} H_k^{(i-1)} &= \left. \frac{\partial h}{\partial s} \right|_{s_k^{(i-1)}} \\ K_k^{(i)} &= \Sigma_k^- H_k^{(i-1)T} (H_k^{(i-1)} \Sigma_k^- H_k^{(i-1)T} + R_k)^{-1} \\ \bar{s}_k^{(i)} &= s_k^{(i-1)} + K_k^{(i)} (z_k - h_k(s_k^{(i-1)}, 0)) \end{aligned} \quad (5.36)$$

until $\|\Delta s\| = \|s_k^{(i)} - s_k^{(i-1)}\| < \varepsilon$ at iteration $i = N$.
Covariance update

$$\Sigma_k = (I - K_k^{(N)} H_k^{(N)}) \Sigma_k^- \quad (5.37)$$

Figure 5.6 Iterated extended Kalman filter.

5.3.3 Unscented Kalman and Information Filters

An alternative approximation to Gaussian state distributions in nonlinear systems is given by the unscented Kalman filter (UKF) [88]. The UKF achieves this goal not by linearizing the state dynamics and observation models, but rather, by representing the state statistics via the unscented transform (UT). Basically, if N is the state dimension, the unscented transform amounts to selecting a symmetric set of $2N+1$ weighted states, called *sigma points*, distributed around the mean, along the principal axes of the covariance matrix:

$$\begin{aligned} s_0 &= \bar{s} \\ s_n &= \bar{s} + \left(\sqrt{(N + \lambda) \Sigma^s} \right)_n \\ s_{N+n} &= \bar{s} - \left(\sqrt{(N + \lambda) \Sigma^s} \right)_n \\ \pi_0 &= \frac{\lambda}{N + \lambda} \\ \pi_n &= \frac{1}{2(N + \lambda)} \\ \pi_{N+n} &= \frac{1}{2(N + \lambda)} \end{aligned} \quad (5.38)$$

where s_n and π_n are the sigma points such that $\sum_n \pi_n = 1$, \bar{s} and Σ^s are the mean and covariance of the multivariate Gaussian distribution,³ and the parameter $\lambda = \alpha^2(N + \kappa) - N$, in turn defined by the two free parameters α and κ , which regulate the position and weight of each state. The term $(\sqrt{N\Sigma^s})_n$ is the n th column of the square-root matrix

$$\sqrt{\Sigma} = B : \Sigma = BB^T \quad (5.39)$$

which can be computed through the Cholesky factorization or, equivalently, via the singular value decomposition (since $N\Sigma^s$ is a symmetric positive-definite matrix).

The sigma points are an equivalent representation for the Gaussian, with the same mean and covariance matrix:

$$\begin{aligned} \bar{s} &= \sum_{n=0}^{2N} \pi_n s_n \\ \Sigma^s &= \sum_{n=0}^{2N} \pi_n^c (s_n - \bar{s})(s_n - \bar{s})^T \end{aligned} \quad (5.40)$$

where the covariance weights are slightly different: $\pi_0^c = \pi_0 + (1 - \alpha^2 + \beta)$ and $\pi_n^c = \pi_n$, $n = 1, \dots, 2N$, where β is an additional parameter. If we choose $\lambda = 0$, we obtain a reduced form of the UT, where the first sigma point is removed ($\pi_0 = 0$):

$$\begin{aligned} s_n &= \bar{s} + (\sqrt{N\Sigma^s})_n \\ \pi_n &= \frac{1}{2N} \\ s_{N+n} &= \bar{s} - (\sqrt{N\Sigma^s})_n \\ \pi_{N+n} &= \frac{1}{2N} \end{aligned} \quad (5.41)$$

As we have seen, when the state undergoes a nonlinear transform $z = h(s)$, the resulting distribution will no longer be Gaussian; however, in many cases, $P(z)$ is still unimodal, and its moments can be approximated by transforming the sigma points,

$$z_n = h(s_n) \quad (5.42)$$

³In the following we distinguish between covariance matrices of state and measurement, respectively, denoted by Σ^s and Σ^z , as well as the *cross-covariance* Σ^{sz} .

and computing the resulting mean and covariance:

$$\begin{aligned}\bar{z} &= \sum_n \pi_n z_n \\ \Sigma^z &= \sum_n \pi_n^c (z_n - \bar{z})(z_n - \bar{z})^T\end{aligned}\tag{5.43}$$

If applied to Bayesian tracking, this method leads to the unscented Kalman filter [88]. In a manner similar to the Kalman filter, we can also choose between two formulations, corresponding to the unscented Kalman and information filter, respectively.

In particular, for the additive noise model (5.30) we introduce in Fig. 5.7 the UKF formulas given by Wan and van der Merwe [161, Table 7.3.2]. In this algorithm we notice that the Kalman gain K_k is computed from the measurement covariance Σ_k^z and cross-covariance Σ_k^{sz} matrices, obtained from the predicted distribution $s_{k,n}^-$ and its projection in measurement space $z_{k,n}^-$. When the measurement vector z consists of high-dimensional data, the inversion of Σ^z , obtained through eq. (5.43) for computing the Kalman gain, would be prohibitive. In that case we can use the unscented information filter (UIF) [95]

<p><i>Prediction</i></p> $s_{k,n}^- = f(s_{k-1,n}), \quad n = 0, \dots, 2N$ $\bar{s}_k^- = \sum_n \pi_n s_{k,n}^-$ $\Sigma_k^{s-} = \sum_n \pi_n^c (s_{k,n}^- - \bar{s}_k^-)(s_{k,n}^- - \bar{s}_k^-)^T + Q_k$ <p><i>Correction</i></p> $z_{k,n}^- = h(s_{k,n}^-), \quad n = 0, \dots, 2N$ $\bar{z}_k^- = \sum_n \pi_n z_{k,n}^-$ $\Sigma_k^z = \sum_n \pi_n^c (z_{k,n}^- - \bar{z}_k^-)(z_{k,n}^- - \bar{z}_k^-)^T + R_k$ $\Sigma_k^{sz} = \sum_n \pi_n^c (s_{k,n}^- - \bar{s}_k^-)(z_{k,n}^- - \bar{z}_k^-)^T$ $K_k = \Sigma_k^{sz} (\Sigma_k^z)^{-1}$ $\bar{s}_k = \bar{s}_k^- + K_k (z_k - \bar{z}_k^-)$ $\Sigma_k^s = \Sigma_k^{s-} - K_k \Sigma_k^z K_k^T$
--

Figure 5.7 Unscented Kalman filter, with additive noise.

<i>Prediction</i>	$s_{k,n}^- = f(s_{k-1,n}^-), \quad n = 1, \dots, 2N$ $\bar{s}_k^- = \sum_n \pi_n s_{k,n}^-$ $\Sigma_k^- = \sum_n \pi_n^c (s_{k,n}^- - \bar{s}_k^-)(s_{k,n}^- - \bar{s}_k^-)^T + Q_k$ $Y_k^- = (\Sigma_k^-)^{-1}$ $y_k^- = Y_k^- \frac{1}{2N} \sum_{n=1}^{2N} s_{k,n}^-$
<i>Correction</i>	$e_{k,n} = z_k - h(s_{k,n}^-), \quad n = 1, \dots, 2N$ $\bar{e}_k = \sum_n \pi_n e_{k,n}$ $\Sigma_k^{sz} = \sum_n \pi_n^c (s_{k,n}^- - \bar{s}_k^-)(\bar{e}_k - e_{k,n})^T$ $\mathcal{H}_k = Y_k^- \Sigma_k^{sz}$ $Y_k = Y_k^- + \mathcal{H}_k R_k^{-1} \mathcal{H}_k^T$ $y_k = y_k^- + \mathcal{H}_k R_k^{-1} [\bar{e}_k + \mathcal{H}_k^T \bar{s}_k^-]$

Figure 5.8 Unscented information filter.

shown in Fig. 5.8, where the update here is written in terms of the sigma point innovations, $e_{k,n}$. In particular, it can be shown that the *average innovation* \bar{e}_k can be removed from the cross-correlation Σ^{sz} (also notice the switched sign of these terms with respect to the expression in z used for the UKF). However, it is needed in the last formula in Fig. 5.8, to compute y_k .

The equivalence between this formulation and the original UKF has been demonstrated by Vercauteren and Wang [158], where

$$\mathcal{R}_k = \Sigma_k^z - \mathcal{H}_k \Sigma_k^{s-} \mathcal{H}_k^T \quad (5.44)$$

should more correctly be substituted for the matrix R_k . Here Σ_k^z represents the innovation covariance S_k estimated through the sigma points; this would better capture the nonlinearity of the representation, being $\mathcal{R}_k = R_k$ only in the linear case, but it would also defeat the computational advantages, since R_k is usually a diagonal matrix, trivially invertible into the related equations. Therefore, by taking $\mathcal{R}_k \approx R_k$, we introduce an approximation but retain the superior performance of the UIF with respect to the EKF.

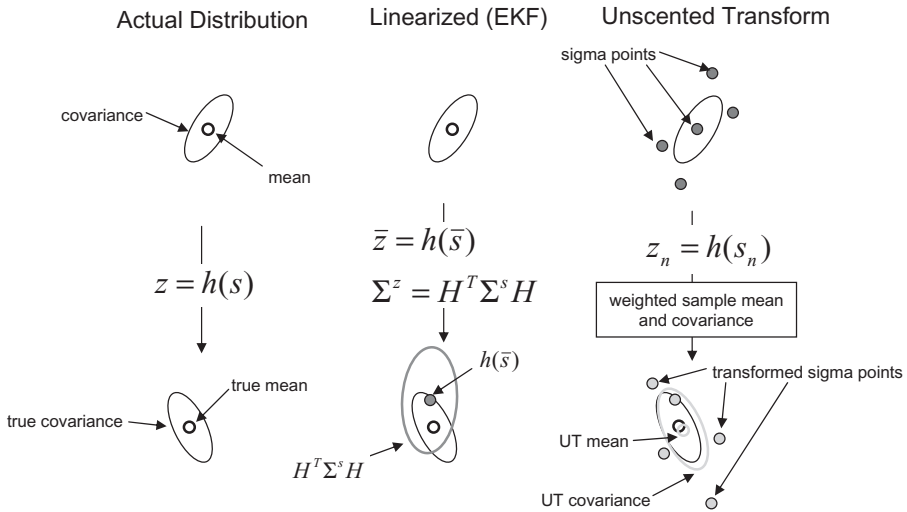


Figure 5.9 Comparison of an unscented transform with EKF linearization for the nonlinear mapping of a Gaussian. (From [161].)

Two important properties are the following (Fig. 5.9):

1. The effects of the nonlinear mapping equation (5.42) are shown approximated up to second order with respect to the first-order linearization of the EKF and EIF.
2. There is no need to compute Jacobian matrices $\partial f/\partial s$ and $\partial h/\partial s$; however, there are $2N$ predictions and measurements $f(s_n)$ and $h(s_n)$ to be evaluated instead of one.

These two properties may be crucial in computer vision when the mapping h is very complex and nonlinear, where often the derivatives cannot be computed analytically or may even not be defined. At the same time, the number of hypotheses to be evaluated is very small and increase linearly with the state dimension, unlike Monte Carlo filters.

5.4 MONTE CARLO FILTERS

For highly nonlinear models and non-Gaussian noise, the filters described above may not be able to represent the state statistics properly. In such cases, which occur often in computer vision, a more flexible representation is required that can eventually cope with multimodal distributions, corresponding to multiple data association hypotheses. Monte Carlo filters [29] realize the Bayesian tracking scheme by random sampling in state space, representing the prior and

posterior statistics through a large set of state hypotheses, also called *particles*.

In these filters, dropping the temporal index k , the state statistics $P(s)$ are given by the set

$$\{s_n, \pi_n\}, \quad n = 1, \dots, N \quad (5.45)$$

where the weights are normalized to $\sum_n \pi_n = 1$. This provides a nonparametric representation that in principle can approximate *any* distribution with the desired accuracy, for $N \rightarrow \infty$, by means of *Dirac* distributions centered at s_n . Therefore, the desired Monte Carlo approximation is given by

$$\lim_{N \rightarrow \infty} \sum_{n=1}^N \pi_n g(s_n) = \int_s g(s) P(s) ds \quad (5.46)$$

for any function $g(\cdot)$, where the limit is intended in a probabilistic sense. Any *moment* of the distribution is obtained by this formula, such as the mean and covariance matrix

$$\begin{aligned} \bar{s} &= \int_s s P(s) ds \approx \sum_{n=1}^N \pi_n s_n \\ \Sigma &= \int_s (s - \bar{s})(s - \bar{s})^T P(s) ds \approx \sum_{n=1}^N \pi_n s_n s_n^T - \bar{s} \bar{s}^T \end{aligned} \quad (5.47)$$

In the following we consider two types of strategies for particle filtering: the SIR (sampling–importance–resampling) scheme and its efficient variants for articulated objects, and the MCMC (Markov chain Monte Carlo) filter for multiple interacting targets. In particular, the SIR scheme achieves a *parallel* sampling procedure in which each particle can be handled independent of the others, while the MCMC scheme is a *serial* method in which particles are generated and evaluated one after another.

5.4.1 SIR Particle Filter

The most direct way to obtain a Monte Carlo approximation of a given distribution $P(s)$ is to sample N independent data from the distribution itself: $s_n \sim P(s)$, with equal weights $\pi_n = 1/N, \forall n$. Unfortunately, in many cases this is not possible, because a generative model for obtaining fairly distributed values from $P(s)$ would be too complex to define. However, if we know the functional, or implicit form of $P(s)$, we can at least *evaluate* the probability of a given state s , and in that case adopt the technique of *importance sampling*. This technique consists of choosing another distribution, $Q(s)$, from which it is easier to generate random samples, such as a Gaussian

or a mixture of Gaussians, and use the same sample to represent $P(s)$, in the following way:

$$\begin{aligned} s_n &\sim Q(s) \\ \pi_n &\propto \frac{P(s_n)}{Q(s_n)} \end{aligned} \quad (5.48)$$

where the weights π_n , subsequently normalized, account for the difference between P and Q . Q is called the *importance* or *proposal* distribution for P , and it can be shown that the efficiency of the Monte Carlo approximation [eq. (5.47)] depends on the actual distance between the two distributions; therefore, the choice of Q should be related to the knowledge we have about P .

For tracking purposes we are interested in approximating the posterior at time k , $s_{k,n} \sim P(s_n | \mathcal{Z}_n)$, in a recursive fashion: Given the previous sample set $\{s_{k-1,n}, \pi_{k-1,n}\}$, representing $P(s_{k-1} | \mathcal{Z}_{k-1})$, we can apply the Bayesian tracking equation (5.1) to the set of Diracs and obtain the functional form of the new posterior,

$$P(s_k | \mathcal{Z}_k) \propto P(Z_k | s_k) \sum_n \pi_{k-1,n} P(s_k | s_{k-1,n}) \quad (5.49)$$

from which a new Monte Carlo sample must be generated. However, in most cases a generative model for this distribution is not available. Therefore, if we sample from an importance distribution Q_k , possibly time varying, we have to compute the new weights:

$$\pi_{k,n} \propto \frac{P(Z_k | s_{k,n}) \sum_n \pi_{k-1,n} P(s_{k,n} | s_{k-1,n})}{Q_k(s_{k,n})} \quad (5.50)$$

Several choices can be used with Q_k , provided that it is not too different from the actual posterior. In particular, in the literature we find primarily two solutions, the most common being given by the *prior* at time k :

$$P(s_k | \mathcal{Z}_{k-1}) \propto \sum_n \pi_{k-1,n} P(s_k | s_{k-1,n}) \quad (5.51)$$

If the dynamic prior is a Gaussian, this is a mixture of Gaussians, which is most often the case: for example, when the process noise is additive [eq. (5.30)]. We can generate samples from it in two steps:

1. Select random particles $s'_{k,n}$ from the old set $s_{k-1,n}$, by choosing them according to the probabilities $\{\pi_{k-1,n}\}$.
2. Generate the next particles $s_{k,n}$ by sampling from $P(s_k | s'_{k,n})$, that is, by simulating the process noise in the dynamic model (2.73).

The first step, also called *resampling*, does not alter the distribution represented, provided that the weights are set back to uniform values $\pi_{k,n} = 1/N, \forall n$. In fact, it can be shown that it improves the statistical properties of the representation by removing particles of too low weight, thus “re-collecting” the sample set. With this choice for Q_k , the new weights for the posterior are simply given by the likelihoods

$$\pi_{k,n} \propto P(Z_k | s_{k,n}) \quad (5.52)$$

and therefore we have the SIR method shown in Fig. 5.10 and described in several places (e.g., [29,84,145]):

- *Resampling.* Select $s_{k,n} \leftarrow s_{k,m}$, with m chosen randomly according to the weights $\{\pi_{k,n}\}$; afterward, reset the weights: $\pi_{k,n} = 1/N$.
- *Sampling.* Drift particles from the previous set using the dynamic mode: $s_{k,n} \sim P(s_k | s_{k-1,n})$.
- *Importance.* Weight particles according to their likelihood $\pi_{k,n} \propto P(Z_k | s_{k,n})$, and normalize the weights $\sum_n \pi_{k,n} = 1$.

This scheme can also be represented symbolically as in Fig. 5.11; the resampling step is denoted by “ \sim ,” producing unweighted particles, followed by convolution with the prior distribution $*P(s_k | s_{k-1})$, and finally, multiplied by the likelihood $\times P(Z_k | s_k)$.

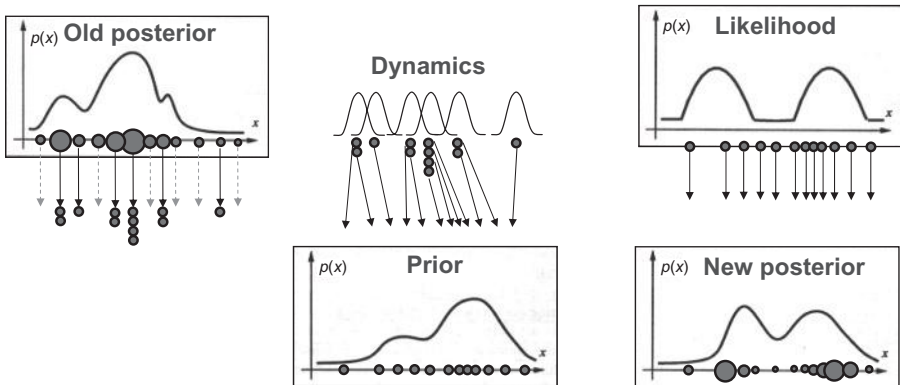


Figure 5.10 Sampling-importance-resampling (SIR) technique.

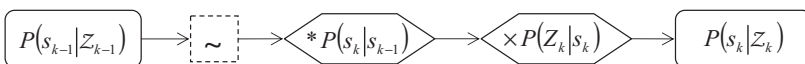


Figure 5.11 Symbolic illustration of a SIR particle filter: resampling, drift, weighting.

Another choice [85] consists of choosing an external importance function, which may be given by a different likelihood $Q \equiv P(Y_k | s_k)$, where Y is a measurement associated with s_k , obtained from another modality or processing tree. In that case, the new states are sampled directly from Q_k , which replaces the first two steps, and the weights are given by

$$\pi_{k,n} \propto \frac{P(Z_k | s_{k,n}) \sum_n \pi_{k-1,n} P(s_{k,n} | s_{k-1,n})}{P(Y_k | s_{k,n})} \quad (5.53)$$

The importance function should be simple enough to allow direct sampling; for example, this is the case for object-level measurements, $Y_k = s_k^*$, which are Gaussian in state space.

A combination of the foregoing methods may also be used, by defining $Q = w_{\text{prior}} Q_{\text{prior}} + w_{\text{ext}} Q_{\text{ext}}$, which is equivalent to sampling a random subset of particles from Q_{prior} and the others from Q_{ext} , where the relative percentages of particles are given by the respective weights. Obtaining Y through a global search in state space (i.e., a target detection) has the advantage of balancing the effects of a stable but coarse measurement Y_k with the accurate but local measurement Z_k .

An important issue in Monte Carlo filters is the quality of approximation to the real distribution. For this purpose we consider the *survival diagnostics*, defined for example, by MacCormick and Isard [110] as

$$\mathcal{D} = \left(\sum_{n=1}^N (\pi_{k,n})^2 \right)^{-1} \quad (5.54)$$

The meaning of \mathcal{D} is the average number of particles that will “survive” the next resampling because they have a meaningful weight within the distribution, and the weight should always be kept high. In the ideal case, $\mathcal{D} = N$ when all $\pi_i = 1/N$, whereas in the worst case, $\mathcal{D} = 1$ when $\pi_i = 1$ for one particle and 0 for the others.

A related quantity is the *survival rate*,

$$\alpha = \left(\int P(s)^2 / P^-(s) ds \right)^{-1} \quad (5.55)$$

where $P^-(s)$ and $P(s)$ are the prior and posterior distributions, respectively. It can be shown that it represents the ratio of the *volume* of the posterior to the volume of the prior in state space, and for large N it can be approximated by $\alpha \approx \mathcal{D}/N \in [1/N, 1]$.

Finally, the sample posterior covariance Σ_k is always a good indicator of tracking quality, since in the case of poor localization, the determinant, or some of the eigenvalues, will reach a high value.

5.4.2 Partitioned Sampling

For articulated shapes, the degrees of freedom can be very high, for example, a human hand is usually modeled with 26 dof, and in this case, the likelihood function $P(Z_k|s_k)$ is nonzero only in a very small percentage of the sampling volume in state space. Therefore, a standard implementation of the SIR particle filter would require a very high number of particles to have the possibility of populating the small volumes properly while preserving an acceptable survival diagnostics. More precisely, this number would increase *exponentially* with the space dimensions.

This can also be seen by looking at the definition of survival rate, or diagnostics, introduced in the preceding section. If we wish to keep a minimum number of meaningful particles \mathcal{D}_{\min} for successful tracking and the prior and posterior distributions have a roughly steady-state covariance $P^-(s)$, $P(s)$, we need at least $N = \mathcal{D}_{\min}/\alpha$ particles, where α is computed from the ratio of the volumes covered by the two distributions. If we have multiple targets with the same likelihood function, the ratio of volumes becomes α^2 since the dimension of the state space is doubled; therefore, the number of particles required to achieve the same performance is $N = \mathcal{D}_{\min}/\alpha^2$, and since usually $\alpha \ll 1$, N increases very quickly.

To overcome this difficulty, several schemes have been proposed in the literature. In particular, *partitioned sampling* [110] is a hierarchical procedure in which the state space is partitioned into the Cartesian product of lower-dimensional subspaces, and the search for the meaningful region is performed incrementally on each subspace. Referring to Fig. 5.12, instead of sampling particles directly from the entire area with a survival rate of α^2 , we first sample particles from the partition s_A , corresponding to the first target state, and subsequently try to populate the good region by sampling only the second

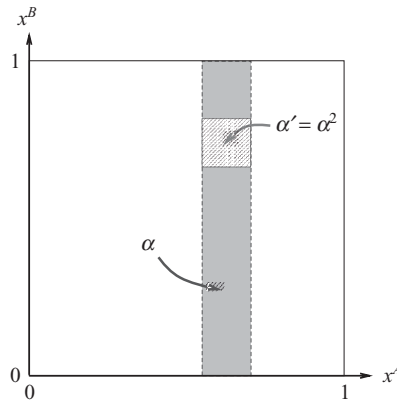


Figure 5.12 Partitioned sampling used to track two targets with a monodimensional state space. (From [110].)

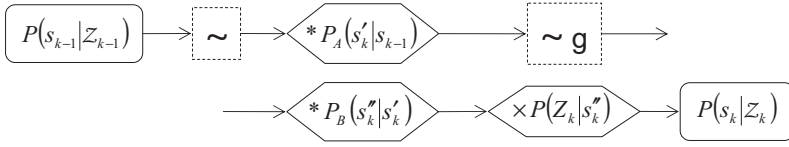


Figure 5.13 Partitioned sampling for a SIR particle filter.

variable, s_B . Both stages have a survival rate α , and therefore we can use the same number of particles of a single-target problem.

The idea of partitioned sampling can be implemented in the context of an SIR particle filter if we introduce the additional operation of *weighted resampling*, with respect to an *importance function* $g(s)$: Given a weighted particle set $\mathcal{S} = (s_n, \pi_n)_{n=1}^N$, we define the normalized importance weights as $\rho_n = g(s_n) / \sum_j g(s_j)$ and then re-sample the set according to these weights, thus producing a new set $\mathcal{S}' = (s'_n, \pi'_n)_{n=1}^N$, where $\pi'_n = \pi_n / \rho_n$, subsequently normalized to 1.

The effect of importance resampling is that of “populating” the peaks of g with particles without altering the represented distribution, as in standard resampling. Here, $g > 0$ can to some extent be an arbitrary function. However, its role is to focus particles in the area where the posterior, restricted to partition s_A , is high; therefore, an obvious choice for a multitarget problem would be the likelihood of target A only, $g = P(Z|s_A)$.

Then, if we indicate by “ $\sim g$ ” the weighted resampling operator, the partitioned sampling scheme is illustrated in Fig. 5.13. The first stage applies a drift within partition s_A only and then resamples the particles to keep them focused on the peaks of g ; the second stage applies the drift in the remaining partition s_B , and finally, re-weights particles according to their likelihood.

As we can see, the first basic requirement for applying partitioned sampling is the possibility of partitioning the system dynamics as the convolution of two subspace predictive priors:

$$P(s''|s) = \int_{s'} P_A(s''|s') P_B(s'|s) ds' \quad (5.56)$$

In the explicit dynamic model, this is equivalent to generating motion in two steps, with independent process noises. The above holds true for noninteracting targets, which may be violated by a general ensemble dynamics with interactions.

Moreover, to select g properly, a second requirement is the possibility of having a subspace likelihood that depends only on the first partition, s_A . If, furthermore, we can partition the measurement space into Z_A and Z_B such that

$$P(Z|s) = P(Z_A|s_A) P(Z_B|s_A, s_B) \quad (5.57)$$

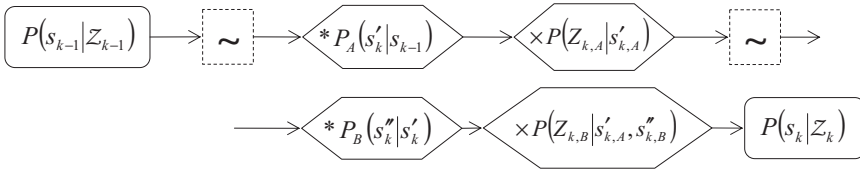


Figure 5.14 Partitioned sampling scheme, with measurement space partition.

where Z_A depends only on s_A while Z_B may depend on both, it can be shown that the scheme above simplifies to the scheme shown in Fig. 5.14, where weighted resampling has been replaced by standard resampling, and the intermediate likelihoods are simply multiplied, as in the standard scheme.

The foregoing schemes can easily be generalized to M partitions of state space and, eventually, of measurement space, by splitting the overall dynamics as the convolution of subspace predictive priors and choosing each importance function peaked in the restriction of the posterior density to the respective partition. One notable application is given by articulated objects. In fact, the pose vector is naturally partitioned according to the hierarchy of the skeleton links because of the dependencies between each link and its ancestors; at the same time, most of the feature- or pixel-level measurements can naturally be partitioned associated with different links, so that the more efficient scheme of Fig. 5.14 can be implemented.

5.4.3 Annealed Particle Filter

In the presence of articulated models, the likelihood function may also exhibit many local maxima in the high-dimensional space, so that even a large particle set may fail to represent the distribution properly because the particles will be dispersed over the peaks (Fig. 5.15). Another way to cope with the curse of dimensionality is to adopt a *multiresolution* approach; in fact, at higher scales the likelihood function is also smoother than at lower scales, and the

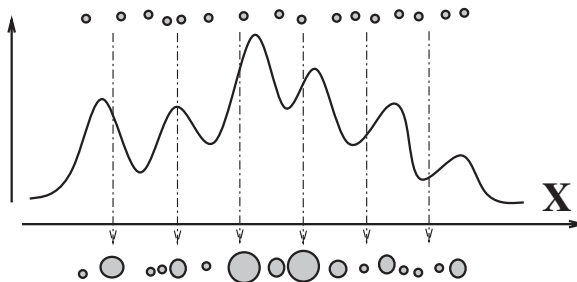


Figure 5.15 The particle set may be dispersed in the presence of many local maxima of the likelihood. (From [63]. Copyright © 2000 IEEE.)

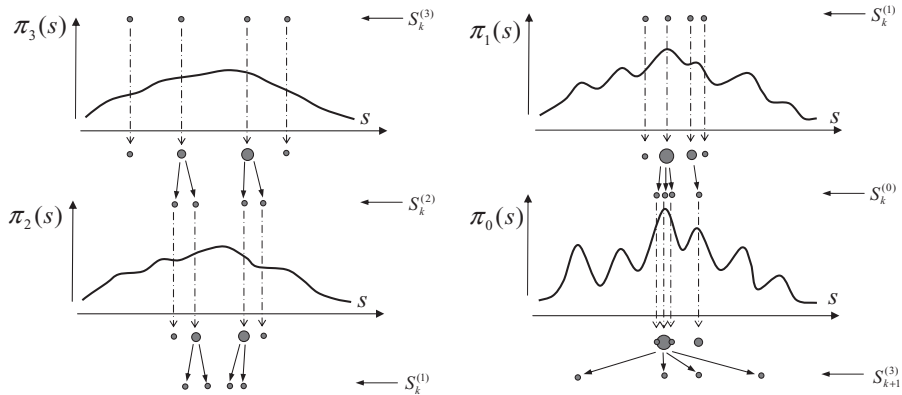


Figure 5.16 Annealed particle filtering. (From [63]. Copyright © 2000 IEEE.)

same set of particles has better survival diagnostics. After a first SIR step, particles get more dense in the correct volume, and the resolution can be increased by repeating the sampling and weighting procedure.

This procedure is reminiscent of the *simulated annealing strategy* [96] and has therefore been called an *annealed particle filter*. It has been proposed by Duetscher et al. [63] for use in tracking human body motion, where the number of degrees of freedom is about 30. In particular, if we denote by

$$\mathcal{S}_k^{(m)} = \left\{ \left(s_{k,1}^{(m)}, \pi_{k,1}^{(m)} \right), \dots, \left(s_{k,N}^{(m)}, \pi_{k,N}^{(m)} \right) \right\} \quad (5.58)$$

the m th annealing layer at time k with N hypotheses, the annealed particle filter consists of the following steps, performed from lowest resolution $m = M$ to $m = 0$ (Fig. 5.16):

1. Repeat for $m = M, M-1, \dots, 0$:
 - Start from the unweighted set $\mathcal{S}_k^{(m)}$, obtained from the previous layer or, if $m = M$, from the previous time stamp.
 - *Drift*. Drift the particles $s_{k,n}^{(m)}$ according to the dynamic model of layer m with covariance $W_k^{(m)}$ (see the discussion below).
 - *Weight*. Assign weights

$$\pi_{k,n}^{(m)} \propto P\left(Z_k | s_{k,n}^{(m)}\right)^{\beta_k^{(m)}} \quad (5.59)$$

where $\beta_k^{(m)}$ is the annealing coefficient of layer m , with $\beta^{(0)} > \beta^{(1)} > \dots > \beta^{(M)}$. The weights are subsequently normalized so that $\sum_n \pi_{k,n}^{(m)} = 1$.

- *Resampling.* Draw random particles from $\mathcal{S}_k^{(m)}$ with the probability given by the weights $\pi_{k,n}^{(m)}$ and possibly with replacement. This produces the unweighted particle set for the next layer, $\mathcal{S}_k^{(m-1)}$.

2. At the lowest layer, $m=0$, before resampling compute the average output state:

$$\bar{s}_k = \sum_n s_{k,n}^{(0)} \pi_{k,n}^{(0)} \quad (5.60)$$

3. Finally, the initial set $\mathcal{S}_{k+1}^{(M)}$ for the next time stamp is produced by drifting the unweighted particles $\mathcal{S}_k^{(0)}$ according to the dynamic model with the largest covariance matrix, W_{\max} .

We notice in the procedure described above that annealing is achieved in two ways (Fig. 5.16). During prediction we use a different dynamics for each layer, with decreasing noise covariance, as well as different weights during correction, with increasing resolution. All of the *annealing parameters* must be chosen carefully to prevent too fast or too slow annealing, in both cases with the risk of failing to locate the global optimum.

For this purpose let us consider the survival diagnostics $\mathcal{D}^{(m)}$ for layer m and the related survival rate α . We can see from eq. (5.59) that it depends on $\beta_k^{(m)}$; therefore, if we wish to impose a given rate α_m , we have to solve the equation $\alpha(\beta_k^{(m)}) = \alpha_m$, which only has a solution in $\beta_k^{(m)}$ that can be found by least-squares optimization:

$$\beta_k^{(m)} = \arg \min_{\beta} \left(\frac{1}{\left(\sum_{n=1}^N (\pi_{k,n}^{(m)})^{2\beta} \right)^{-1} N} - \alpha_m \right) \quad (5.61)$$

starting from the initial value $\beta_{k-1}^{(m)}$ of the previous time stamp.

Concerning the motion covariances $W^{(m)}$, the following rule is suggested [63]:

$$W^{(m)} = W_{\max} \cdot (\alpha_M \alpha_{M-1} \cdots \alpha_m) \quad (5.62)$$

and finally, the desired survival rates, α_m , can be chosen according to the number of layers; in particular, if a sufficient number M is used, then $\alpha_0 = \alpha_1 = \cdots = \alpha_M = 0.5$ gives an adequate resolution for the output estimate \bar{s}_k for most cases.

5.4.4 MCMC Particle Filter

In the Monte Carlo filters described so far, particles have been drawn independent of the importance distribution $Q(s)$, which also allows parallel

implementation of prediction and correction steps. A different class of methods is based, instead, on *sequential* sampling and evaluation of state hypotheses where no separate prediction and correction are given, or better, they are executed alternately for each particle. These methods are also named Markov chain Monte carlo (MCMC), because they generate candidate-state hypotheses in a sequence using a *conditional* proposal distribution $Q(s'|s)$, forming a Markov chain that in the limit of large numbers has the same statistics as those of the original probability density function (pdf).

In particular, given a $P(s)$ with known functional form and a conditional proposal $Q(s'|s)$ that to a large extent is arbitrary but also not too far from P , we can generate the chain via the *Metropolis–Hastings algorithm* [119]: Start from an initial state s_0 not too far from the main *mode* of $P(s)$, and repeat for $n = 1, \dots, N$:

1. Propose a new state s' from the previous one, s_{n-1} , by sampling from the proposal density $Q(s'|s_{n-1})$.
2. Compute the *acceptance ratio*:

$$a = \frac{P(s')}{P(s_{n-1})} \frac{Q(s_{n-1}|s')}{Q(s'|s_{n-1})} \quad (5.63)$$

3. If $a \geq 1$, accept the state $s_n = s'$; else, accept it with probability a . In case of rejection, the last value, $s_n = s_{n-1}$, is kept, producing duplicate particles.

It can be shown that after a short transient whose duration depends both on the initial state s_0 and on the choice of Q , the stationary chain follows the statistics of P . The initial transient, also called a *burn-in sample*, can be discarded from the set, providing useless state hypotheses and measurement likelihoods. This computational overhead is largely compensated for by the fact that the overall number of particles necessary to represent P is usually lower than that of a standard SIR scheme, in particular for high-dimensional spaces. This strategy can be used, for example, in a maximum-likelihood estimation $P(z|s)$; moreover, if Q is also symmetric, such as a Gaussian $Q(s_1|s_2) = \mathcal{N}(s_1 - s_2, \Sigma)$, the ratio of Q cancels out in the formula above.

When applied to Bayesian tracking, we have the approach presented by Khan et al. [92], where MCMC sampling is used in the presence of non-Gaussian likelihoods, target interactions, and uncertain data association arising from a cluttered background. In particular, while modeling target interactions we need the ensemble state s_k , which may be given by repulsive dynamics terms, to avoid overlapping targets, and also for modeling mutual occlusions during visual processing.

The dimensionality of the ensemble state would be far too large for the standard SIR filter, in particular for many targets, such as a cell-tracking

application. Instead, with the MCMC approach of [92], successful tracking of multiple targets can be achieved without an exponential increase in hypotheses, because of a clever exploration of high-dimensional space. The MCMC filter presented in this section has been described by Panin et al. [136], with some adaptations used to reduce computational costs for real-time problems.

Consider, for each target o , N unweighted particles with duplicates $s_{k,n}^o$, approximating the posterior distribution of the ensemble at time k , $P(\mathbf{s}_k | \mathcal{Z}_k)$, so that all weights can be set to $\pi_{k,n} = 1/N$. In this framework, the ensemble prior $P(\mathbf{s}_k | \mathcal{Z}_{k-1})$ is given by

$$P(\mathbf{s}_k | \mathcal{Z}_{k-1}) \approx \frac{1}{N} \sum_n \left[\prod_o P(s_{k,n}^o | s_{k-1,n}^o) \right] \quad (5.64)$$

where an independent predictive prior $P(s_k^o | s_{k-1}^o)$ has been applied to each target. If evaluated for every proposal state, this equation would require a very great computational effort. Thus, an approximation more suitable for real-time tracking is given by applying the predictive prior only to the last ensemble average $\bar{\mathbf{s}}_{k-1}$:

$$P(\mathbf{s}_k | \mathcal{Z}_{k-1}) \approx \prod_o P(s_k^o | \bar{s}_{k-1}^o) \quad (5.65)$$

where

$$\bar{s}_{k-1}^o = \frac{1}{N} \sum_n s_{k-1,n}^o, \quad o = 1, \dots, O \quad (5.66)$$

By combining the prior (5.65) with the ensemble likelihood, the posterior distribution is given by

$$P(\mathbf{s}_k | \mathcal{Z}_k) \propto P(\mathcal{Z}_k | \mathbf{s}_k) \prod_o P(s_k^o | \hat{s}_{k-1}^o) \quad (5.67)$$

where the ensemble likelihood of independent measurements is modeled by

$$P(\mathcal{Z}_k | \mathbf{s}_k) = \prod_o P(\mathcal{Z}_k^o | s_k^o) \quad (5.68)$$

This formulation can easily be extended to include dynamic interactions by introducing penalty terms $\psi(s^{o1}, s^{o2}) \leq 1$: for example, decreasing with the overlapping areas of projected bounding boxes $B(s^{o1})$ and $B(s^{o2})$:

$$\psi(s^{o1}, s^{o2}) \equiv \exp \left(-\beta \frac{|B(s^{o1}) \cap B(s^{o2})|}{\min(|B(s^{o1})|, |B(s^{o2})|)} \right) \quad (5.69)$$

where the intersection area is normalized to $[0, 1]$ and $\beta \geq 0$ is a coefficient regulating its behavior ($\beta = 0$ gives no penalty). This term is computed for all overlapping target pairs under the ensemble hypothesis \mathbf{s} , thus giving

$$\Psi(\mathbf{s}) = \prod_{o_1, o_2 > o_1} \psi(s^{o_1}, s^{o_2}) \quad (5.70)$$

which corresponds to a Markov random field (MRF) in state space. Then the posterior becomes

$$\underbrace{P(\mathbf{s}_k | \mathcal{Z}_k)}_{\text{total posterior}} \propto \underbrace{\left[\prod_o P(Z_k^o | s_k^o) \right]}_{\text{likelihood}} \underbrace{\left[\prod_{o_1, o_2 > o_1} \psi(s_k^{o_1}, s_k^{o_2}) \right]}_{\text{penalty}} \underbrace{\left[\prod_o P(s_k^o | \bar{s}_{k-1}^o) \right]}_{\text{prior}} \quad (5.71)$$

At each frame, the MCMC filter generates new particles from an initial seed \mathbf{s}_0 with the Metropolis–Hastings algorithm. In particular, the density Q proposal can be chosen to be the predictive prior $P(s_k^o | s_{k-1}^o)$, which is usually Gaussian.

For efficient sampling, a very convenient strategy is to pick up a random target o , apply the proposal distribution to this target only, generate the new state $s_k'^o$, and compute the acceptance ratio a , which simplifies to

$$a = \prod_{o_1 \neq o} \frac{\psi(s_k'^o, s_k'^{o_1})}{\psi(s_k^o, s_k^{o_1})} \cdot \frac{P(Z_k^o | s_k'^o)}{P(Z_k^o | s_k^o)} \cdot \frac{P(s_k'^o | \bar{s}_{k-1}^o)}{P(s_k^o | \bar{s}_{k-1}^o)} \quad (5.72)$$

where the three ratios favor, respectively, proposals with fewer overlapping states, a higher likelihood, and close to the previous average, which are obviously conflicting requirements. We notice how in this equation only single-target terms are present, and in particular, since the proposal likelihood concerns only the target selected, it simplifies the measurement process dramatically. Figure 5.17 shows a possible multitarget MCMC walk.

Finally, the initial seed for creating the chain should not be too far away from the posterior mode. Therefore, we can start from the previous average ensemble \bar{s}_{k-1} , leading quickly to a stationary distribution of the chain. Empirical results show that a burn-in sample of 15% of the overall hypotheses N can be used safely.

5.5 GRID FILTERS

An entirely different strategy for Bayesian tracking consists of using a *grid* representation of the target distribution. The idea behind this strategy is to discretize the state space, in advance, into a number of finite volumes, or *cells*, where a single prior or posterior probability is stored within the volume, pro-

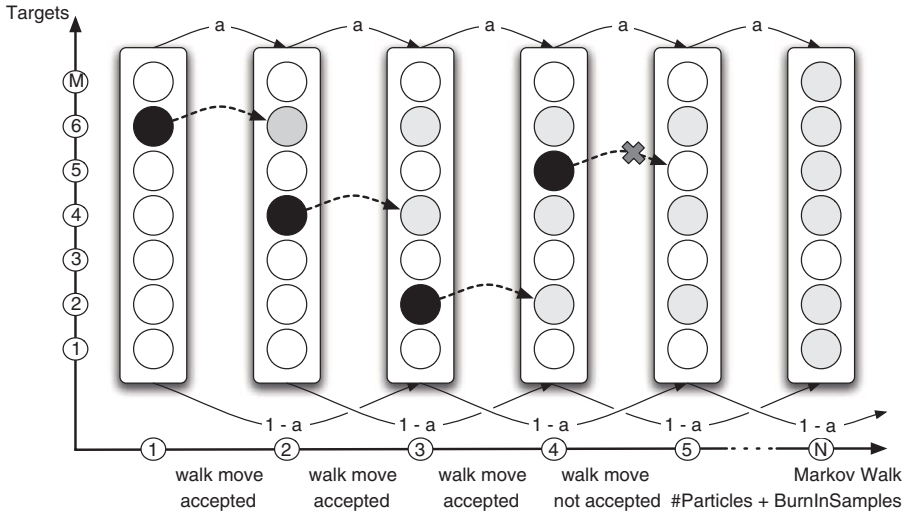


Figure 5.17 Snapshot of an MCMC walk.

viding a piecewise constant approximation of the pdf which, for an equispaced grid, is equivalent to a histogram. These values actually represent *integrals* of probability densities over finite volumes so that integrals in the Bayesian tracking equation (5.1) are replaced by *sums* over cells. Similar to particle filters, such a representation does not directly offer a representative output state for the target, such as the mean or MAP value, but this must subsequently be computed by averaging, or clustering, high-probability cells.

While being entirely nonparametric and allowing any type of dynamics or likelihood models, these filters have the disadvantage of a high number of hypotheses, exponentially increasing with the state dimension, due to the grid discretization along each dimension. However, this problem can be approached by coarsening the grid in regions with low probability, and refining the discretization only in high-probability cells; this gives rise to the *multiresolution* approach proposed by Stenger et al. [151] and described below.

Assume that the states of interest are located within a compact region \mathcal{R} in state space: for example, the view-volume intersection described in Section 6.4. Then we can partition this volume into a multiresolution grid \mathcal{S}_n^l , $n=1, \dots, N_l$, consisting of L different levels, each with N_l cells, covering \mathcal{R} entirely:

$$\bigcup_{n=1}^{N_l} \mathcal{S}_n^l = \mathcal{R}, \quad l=1, \dots, L \quad (5.73)$$

Cells across different levels are connected in a *tree* (Fig. 5.18), so that each cell at level l is subdivided in multiple cells at level $l+1$, until the maximum-resolution grid with N_L cells. At each level, we can define a discrete

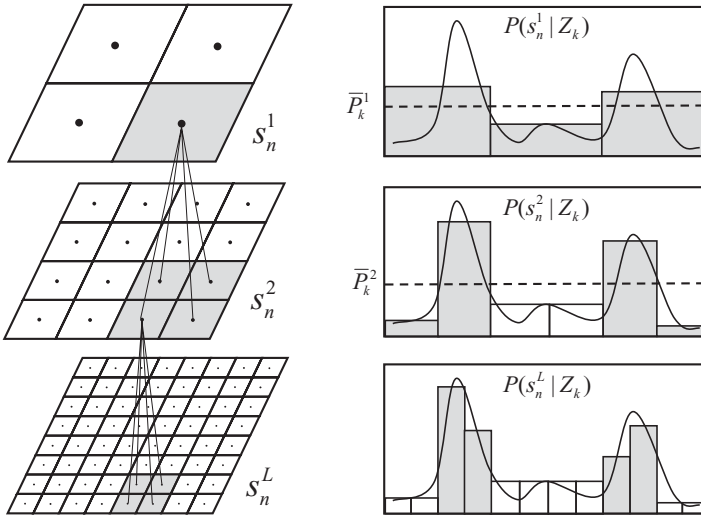


Figure 5.18 Hierarchical grid filter. *Left*: multiresolution grid with three levels of a tree structure; *right*: top-down representation of the posterior using the algorithm of Fig. 5.19. Values above the threshold are recomputed at the next level (gray boxes); otherwise, they are inherited from the previous level (white boxes). (From [151]. Copyright © 2006 IEEE.)

approximation of the posterior $P(s_n^l | \mathcal{Z}_k)$ for a single target by integrating over each cell:

$$P(s_n^l | \mathcal{Z}_k) = \int_{s \in S_n^l} P(s | \mathcal{Z}_k) ds \quad (5.74)$$

We notice how the time index k disappears from the state since the grid structure is constant. This is a big advantage for sampling model features, which can be done *off-line* for each state, stored in memory, and retrieved during matching.

Assuming a uniform initial prior, the first posterior corresponds to the likelihood function $P(\mathcal{Z}_k | s_n^l)$, while in subsequent steps it will also incorporate the prior distribution through the dynamical model. In particular, to propagate the previous pdf at time $k-1$ to the current time k , the highest-resolution grid is used $P(s_n^L | \mathcal{Z}_{k-1})$, and the prior distribution is computed at any level l , by discretizing the Chapman–Kolmogorov equation:

$$P(s_n^l | \mathcal{Z}_{k-1}) = \sum_{m=1}^{N_L} P(s_n^l | s_m^L) P(s_m^L | \mathcal{Z}_{k-1}) \quad (5.75)$$

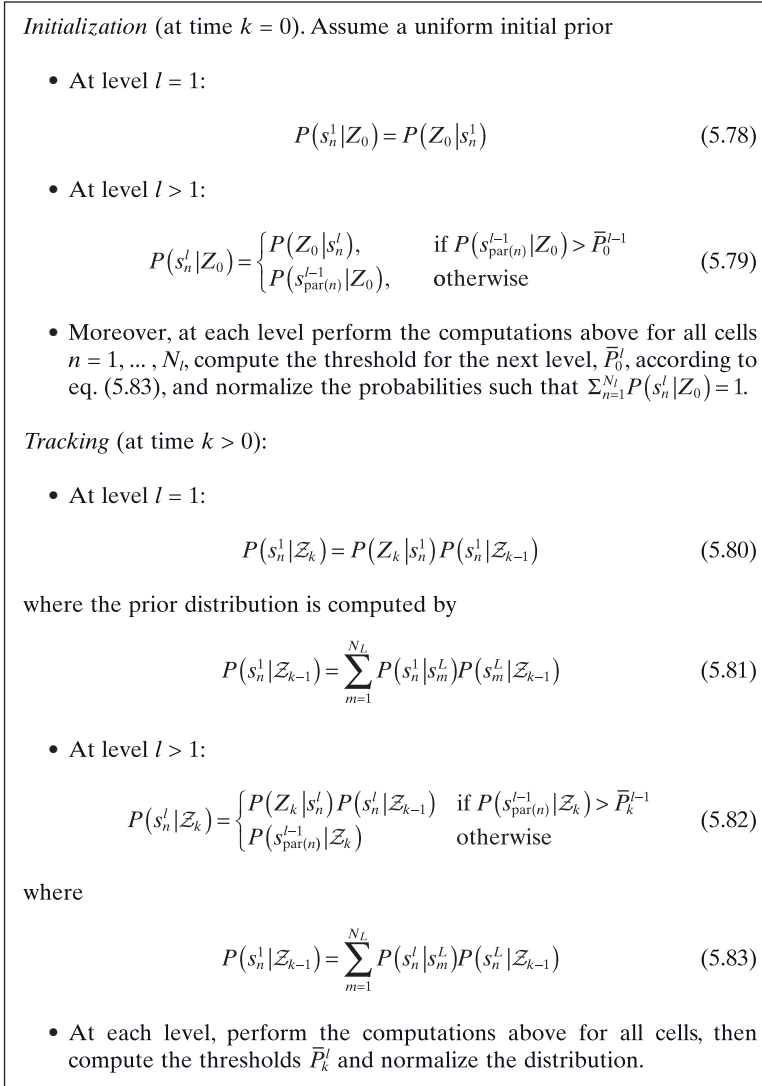


Figure 5.19 Hierarchical grid-based Bayesian filter.

where a time-invariant predictive prior, $P(s|s')$, was used to approximate *region-to-region* transition probabilities.

Furthermore, the Bayesian update uses the likelihood within each cell, which is also assumed to be piecewise constant and to be represented by the central state,

$$P(Z_k | s_n^l) \propto P(Z_k | c(s_n^l)) \quad (5.76)$$

where $c(\mathcal{S}_n^l)$ can also be considered as a cluster *prototype*, representing all similar shapes within the cell.

The reasoning behind a multilevel grid is that the first levels do not represent the posterior pdf with sufficient accuracy; nevertheless, they can be used to *prune* the tree and avoid evaluating the posterior in cells with low probability. Thus, low-probability cells at level l are *inherited* by their parent cell at $l+1$, and so on. This approximation leads to a dramatic efficiency improvement, while not significantly affecting the precision of the final estimate at level L .

This criterion also requires the choice of a probability *threshold* \bar{P}_k^l at each level, which is selected empirically according to the maximum and minimum values of the pdf in the grid:

$$\bar{P}_k^l = P_k^{l,\min} + c(P_k^{l,\max} - P_k^{l,\min}) \quad (5.77)$$

where $P_k^{l,\min}$ and $P_k^{l,\max}$ are the extreme values at level l , and c is a constant, usually set to the midvalue $c = 0.5$. The right side of Fig. 5.18 shows an example of tree pruning where we can see that many cells at each level are inherited from the preceding level. If we denote by $\text{par}(n)$ the parent of cell n in the tree structure, the hierarchical grid-based filter is as summarized in Fig. 5.19.

EXAMPLES OF TARGET DETECTORS

In this chapter we consider the problem of object detection with the goal of finding new targets entering the scene, and of labeling them uniquely until they get lost because of occlusions, unexpected maneuvers, or motion outside the field of view. Target detection is a localization task very similar to tracking, however with more severe restrictions and difficulties:

- *Global vs. local search.* A detector must search for targets in the *full* state space, or at least, the *visible-from-camera* state volume, and provide roughly the initial distribution of targets for tracking. This means that it must solve a global optimization problem [162] of a multiobjective likelihood function in state space, labeling all its relevant peaks as new targets. Instead, the tracking algorithms presented so far update each target distribution only by looking in a *neighborhood* of its predicted position.
- *Static vs. dynamic data association.* For target detection, no prediction is available apart from a possible *absolute* prior, restricting the search to the most likely areas. Therefore, any visual modality employed cannot make use of dynamic data association, such as the nearest neighbor to the predicted feature, but can only rely on static methods, matching features by using an invariant descriptor. For example, SIFT or SURF key-points can be used because their descriptor is to some extent invariant to geometric and photometric variations from different views of the object.

These limitations result in an increased computational complexity, which may often become intractable, as well as a reduced precision of localization, which is tackled basically in three ways:

1. *State-space reduction.* To perform the global search efficiently, the search manifold is usually reduced to a lower dimension, with a simpler and possibly linear parametrization, so that global optimization can be carried out more easily, from a set of hypothetical correspondences. For example, instead of a three-dimensional Euclidean pose with 6 dof, one can localize a target by considering planar roto-translation and scale similarity with 4 dof, and ignore the two out-of-plane rotations, if we assume that we detect only targets that initially “face” the camera in a frontal position.
2. *Use of trained classifiers.* Most visual modalities described so far involve a *generative*, model-based matching strategy in which model features are re-projected under an arbitrary state hypothesis and matched to the image data at different levels. Instead, a *discriminative method* for localization solves the localization problem as a classification task (“questions and answers”) by exploring a discrete set of configurations and discarding less likely hypotheses. This implies a *training* phase for the classifier, such as the feature-based *AdaBoost method* [66].
3. *Bottom-up search.* Sometimes, instead of a *top-down search* in state space, which consists of formulating hypotheses, verifying them on the data, and clustering their likelihoods, it is more convenient to cluster image features directly and generate model association hypotheses in a *bottom-up* (or data-driven) fashion. For example, after detecting line segments (Fig. 4.39) they can be *grouped* pairwise according to their relative location, such as parallel or collinear lines, that *vote* for corresponding model features, thus generating state-space hypotheses that can be verified on the remaining data. To this class of methods belong, for example, the geometric hashing technique (Section 6.3) and the blob clustering and triangulation method described in Section 6.1.

In this chapter we briefly describe a few examples of “specialized” detectors that meet the requirements specified above for localizing new targets, that use predefined object models in the absence of state prediction.

6.1 BLOB CLUSTERING

The blob modality described in Section 4.3 can also be used for object detection, because it provides a set of candidate target locations over the entire field of view. This information is especially useful in a multicamera setup, since redundant views reduce the number of ambiguous data associations that may arise from a rather coarse shape descriptor, and provide a three-dimensional

localization via triangulation algorithms. These blobs can be obtained from any pixel-level map, such as color, foreground, and motion segmentation, as well as from any pixel-level data fusion.

However, some difficulties arise in this procedure; in fact, most often a segmented image produces multiple connected components in place of a single component because of misclassified areas (missing detections) that may arise because the model does not adapt to the real imaging conditions. For example, in foreground segmentation it often happens that the target shares some color properties with the local background and thus is also classified as background; and motion segmentation may fail where texture is absent. For this reason, individual blobs belonging to the same targets must be *clustered* in some way, by paying attention to the resulting image shape and size of the resulting group. An example is provided by the minimum spanning tree, mentioned earlier in Section 4.3.

6.1.1 Localization with Three-Dimensional Triangulation

When multiple views are available, a single target should produce a set of blobs that can effectively be matched and used for three-dimensional localization, by means of a *triangulation method* (Fig. 6.1). In particular, the main steps involved in this *bottom-up* approach are the following:

1. *Segmentation*. Images are segmented according to known models: for example, a reference histogram or a background model.
2. *Blob detection and clustering*. Connected components are identified, possibly after image processing using morphological operators, and clustered into the major blobs.
3. *Data association hypotheses*. Each blob from each view must be associated with a given target, and a combinatorial number of hypotheses can be generated.
4. *Validation and three-dimensional triangulation*. To find the correct association hypothesis while localizing targets in three-dimensional space, triangulation is carried out, and the residual error is evaluated. The maximum-likelihood hypothesis will produce the lowest image residual.

In practice, this technique may present several difficulties:

- It requires preprocessing and clustering as well as morphology operator thresholds.
- Occlusions cannot be solved easily because, in at least one view, two blobs will merge into a single blob.
- The technique produces combinatorial association hypotheses, possibly including false alarms and missing detection.

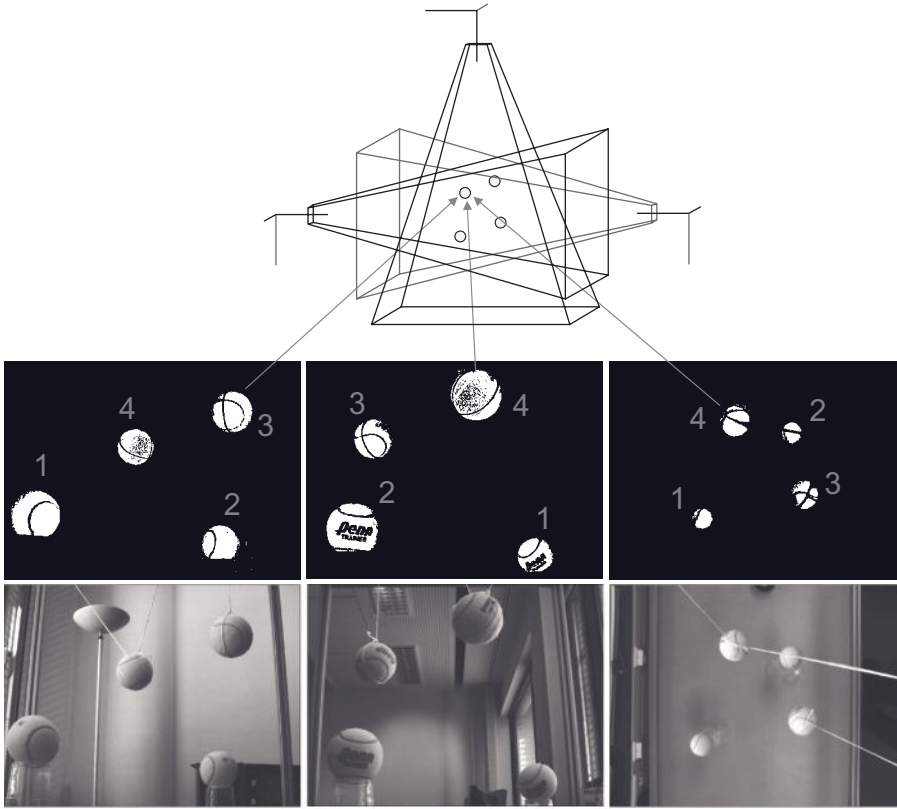


Figure 6.1 Multitarget detection from detected blobs.

- The three-dimensional localization may be quite poor, since the blob location is known with coarse precision, and this has a significant influence over the triangulation procedure.

However, in many cases this method may be preferred because of relatively fast computation and sufficient precision for initializing the tracking pipeline.

Starting with two calibrated views (Fig. 6.2), the triangulation problem can be formulated as follows: Given the internal and external parameters and two corresponding image measurements \mathbf{y}_1 and \mathbf{y}_2 , find the point $\hat{\mathbf{x}}$ that best matches the measurements given. The optimal (maximum-likelihood) solution in the presence of Gaussian noise on the measurements is given by minimizing the re-projection error, that is, the sum of squared differences $\|\mathbf{y}_1 - \hat{\mathbf{y}}_1\| + \|\mathbf{y}_2 - \hat{\mathbf{y}}_2\|$, where $\hat{\mathbf{y}}_c$ is the projection of $\hat{\mathbf{x}}$ onto camera c . However, this results in a nonlinear estimation problem (see [77, Chap. 12]); a well-known approximation, computationally cheaper and easier to implement,

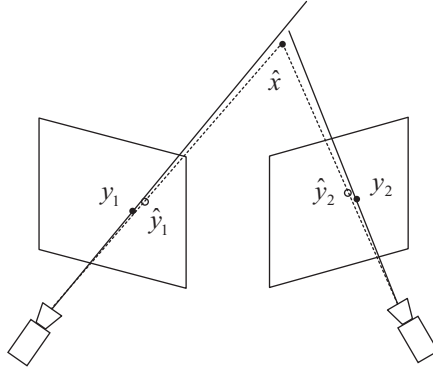


Figure 6.2 Triangulation from two views.

which usually produces an acceptable precision, is given by a linear triangulation using the direct linear transform (DLT).

If we write the camera projection matrices $P_c = K_c[R_c|t_c]$ for camera c , we can formulate the problem in homogeneous coordinates: Find \mathbf{x} such that $\mathbf{y}_c = P_c \mathbf{x}$ for all c . The algebraic triangulation error can be expressed by the cross-products $\mathbf{y}_c \times (P_c \mathbf{x})$, resulting in three equations, of which only two are independent

$$\begin{aligned} y_c^1 (\mathbf{p}_c^{3T} \mathbf{x}) - (\mathbf{p}_c^{1T} \mathbf{x}) &= 0 \\ y_c^2 (\mathbf{p}_c^{3T} \mathbf{x}) - (\mathbf{p}_c^{2T} \mathbf{x}) &= 0 \\ y_c^1 (\mathbf{p}_c^{2T} \mathbf{x}) - y_c^2 (\mathbf{p}_c^{1T} \mathbf{x}) &= 0 \end{aligned} \quad (6.1)$$

where \mathbf{p}_c^i is the i th row of the projection matrix P_c . By taking only the first two equations for each camera, and organizing them into a matrix,

$$A = \begin{bmatrix} y_1^1 \mathbf{p}_1^{3T} - \mathbf{p}_1^{1T} \\ y_1^2 \mathbf{p}_1^{3T} - \mathbf{p}_1^{2T} \\ \vdots \\ y_c^1 \mathbf{p}_c^{3T} - \mathbf{p}_c^{1T} \\ y_c^2 \mathbf{p}_c^{3T} - \mathbf{p}_c^{2T} \end{bmatrix} \quad (6.2)$$

we have a homogeneous problem, $A\mathbf{x} = 0$, which in the absence of noise would have infinite solutions in \mathbf{x} up to a scale factor, and can be solved by imposing a constraint such as $\|\mathbf{x}\| = 1$; in the presence of noise, we look instead for a minimum-norm solution $\|A\mathbf{x}\|$ with the constraint on \mathbf{x} given above, given by the SVD decomposition

$$A = U\Sigma V^T \quad (6.3)$$

where $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_n)$ are the singular values of A (in our case, $n = 4$), and U and V are two orthogonal matrices. In particular, the last column of V , namely V_4 , corresponds to the smallest value σ_4 , which should be 0 in the noise-free case and therefore is the value sought for \mathbf{x} . The solution to the DLT triangulation can be used as a starting point \mathbf{x}_0 for a nonlinear LSE refinement, through the Gauss–Newton (or Levenberg–Marquardt) equations.

6.2 ADABOOST CLASSIFIERS

AdaBoost, short for *adaptive boosting*, is a machine learning algorithm formulated by Freund and Schapire [66]. It is a *meta-algorithm*, so can be used in conjunction with many other learning algorithms (classifiers) to improve their performance. AdaBoost is adaptive, in the sense that updated classifiers are “tweaked” in favor of those instances that have been misclassified by previously. AdaBoost is sensitive to noisy data and outliers, but it is also less susceptible to the *overfitting* problem than are most learning algorithms.

AdaBoost calls a *weak classifier* repeatedly, in a series of rounds $t = 1, \dots, T$. For each call, a distribution of *weights* w_t is updated, indicating the importance of examples in the data set, for the classification task. On each round, the weights of each incorrectly classified example are increased, or alternatively, the weights of each correctly classified example are decreased, so that the new classifier focuses more on misclassified examples.

6.2.1 AdaBoost Algorithm for Object Detection

The algorithm for binary classification tasks in AdaBoost, which can be used for object detection, is the following [160]. Given a training set $(x_1, y_1), \dots, (x_m, y_m)$, where $x_i \in X$ are negative and positive example images (e.g., nonfaces vs. faces), respectively, indicated by $y_i \in Y = \{0, 1\}$, the AdaBoost weights are initialized as $w_{1,i} = 1/2m$ and $1/2l$ for $y_i = 0, 1$, where m and l are the number of negative and positive examples. Then for $t = 1, \dots, T$, carry out the following procedure:

1. Normalize the weights

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}} \quad (6.4)$$

so that w_t is a probability distribution.

2. For each *feature* j , train a weak classifier h_j over all examples, which is restricted to use only that feature. Its weighted error is evaluated: $\varepsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.

3. Find the classifier h_t with minimum error, ε_t .
4. Update the weights

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \quad (6.5)$$

where $e_i = 0$ if the example x_i is classified correctly by h_t , $e_i = 1$ otherwise, and $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.

The output of the final classifier is

$$h(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0, & \text{otherwise} \end{cases} \quad (6.6)$$

where $\alpha_t = \log(1/\beta_t)$. Thus, after selecting an optimal classifier h_t for the distribution w_t , the examples x_i that the classifier h_t identified correctly are weighted less, and those that it identified incorrectly are weighted more. Therefore, when the algorithm is testing the classifiers on the distribution w_{t+1} , it will select a classifier that better identifies those examples that the previous classifier missed.

6.2.2 Example: Face Detection

Viola and Jones [160] used a set of features to classify faces vs. nonfaces, over small 24×24 image windows (Fig. 6.3). In particular, each feature is a Haar-like

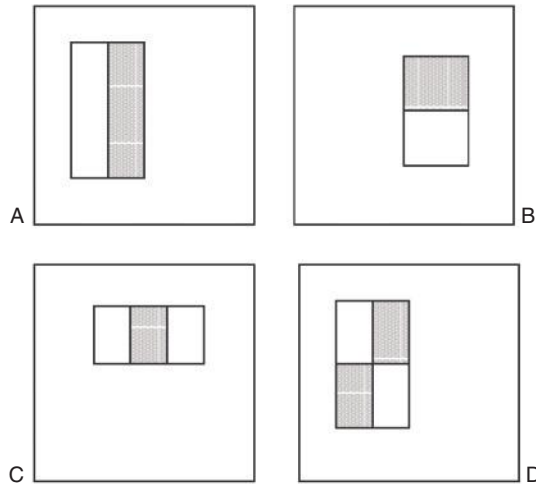


Figure 6.3 Some of the Haar-like features used for AdaBoost face detection. (From [160].)

rectangular function [138]. Each of these features (f_j) is used to train a weak classifier h_j , which performs the operation

$$h_j(x) = \begin{cases} 1, & \text{if } p_j f_j(x) < p_j \theta_j \\ 0, & \text{otherwise} \end{cases} \quad (6.7)$$

where f_j is the operator over the image window x , p_j a parity sign, changing the inequality direction, and θ_j a classification threshold. Such a classifier, even after training over the entire sample set x_i , will not have a good classification performance because more or less, half of the sample is misclassified; however, a cascade of two classifiers with AdaBoost training produces an acceptable detection rate.

Open-source libraries for computer vision, such as OpenCV [15], contain efficient implementations of both AdaBoost training and detection functionalities. After performing face detection on a stereo camera setup, image locations can be used as a feature-level measurement so that the state-space localization can be done by triangulation.

Figure 6.4 shows an example of face detection and three-dimensional localization, with stereo triangulation based on the DLT [77]. This method can easily be extended to multiple targets by additionally matching face pairs (data association) of different people across stereo images with a simple template matching.

6.3 GEOMETRIC HASHING

When no texture is given and the object shape contains many straight edges (e.g., a polyhedral surface), object detection can proceed by matching contour segments (Section 4.4.2) detected, for example, via the Hough transform over an edge map, or by any other method. However, contour segments provide little information for a data association problem, which in the static case becomes a large combinatorial problem. In other words, we must look for the best match between m image features and n model features that yields the pose with the smallest re-projection error (Section A.2). The problem is made more complex by the fact that each view of the object offers a different subset of visible lines, and that from the image side, not all lines are detected successfully, while background items provide false alarms as well.

To cope with this complexity, a bottom-up approach may accelerate the search procedure significantly. In particular, we consider the well-known *geometric hashing technique*, proposed by Wolfson and Lamdan [166] and reviewed more recently by Wolfson and Rigontsos [165]. This is a general method for model-based object recognition in occluded scenes, based on a *voting* strategy that can also be parallelized for further efficiency. It is based on the observation that a few model features (often a single feature) can be chosen as a

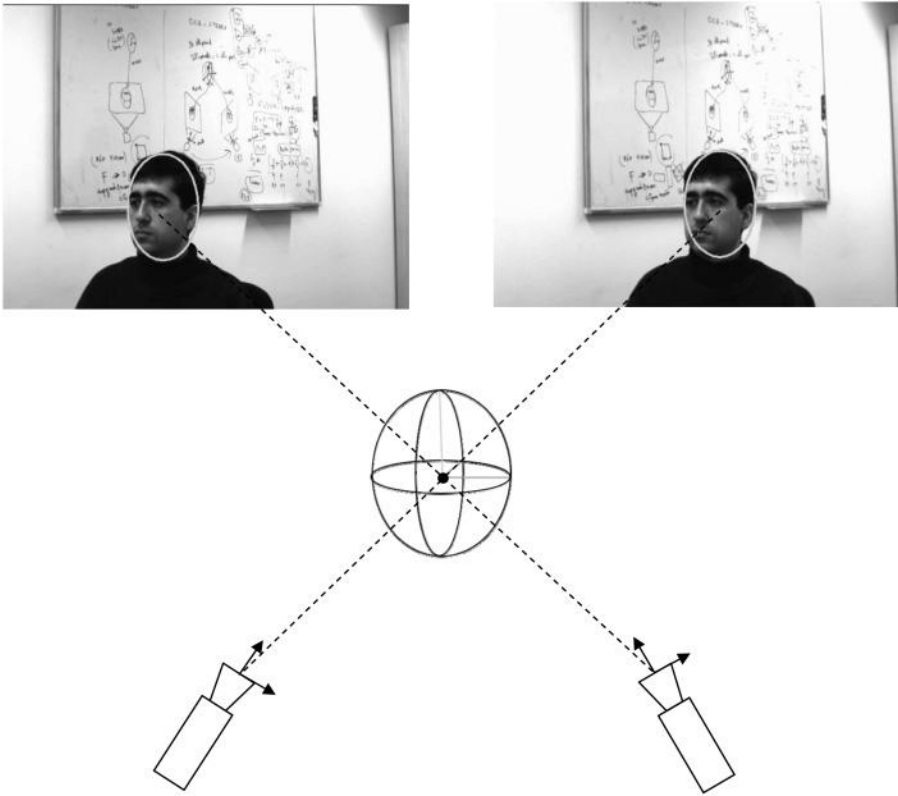


Figure 6.4 Face detection with AdaBoost and three-dimensional triangulation.

geometric *basis* onto which a geometrically invariant description of all segments is done, with respect to a given transformation class (e.g., similarities). This description can therefore be matched to a similarity-invariant description on the image side, provided by a corresponding group of detected segments.

For this purpose, in the off-line phase a *hash table* for the object shape is built, consisting of an entry for each possible model basis, used to describe all the other segments. In the online recognition phase, randomly selected segments on the image provide a basis for describing all the other segments, so that each can *vote* for one or more matching segments in the hash table, and thus for the related basis. If a given model basis receives a large number of votes, there is evidence for the presence of an object instance, to which the matching segments must belong. This also provides a geometric transformation between model and image bases, so that the localization problem is also solved. We consider first the case of two-dimensional similarities, and later, an extension of the three-dimensional case.

For two-dimensional similarities, with roto-translation and uniform scale, it is possible to describe model segments in an invariant manner, by selecting

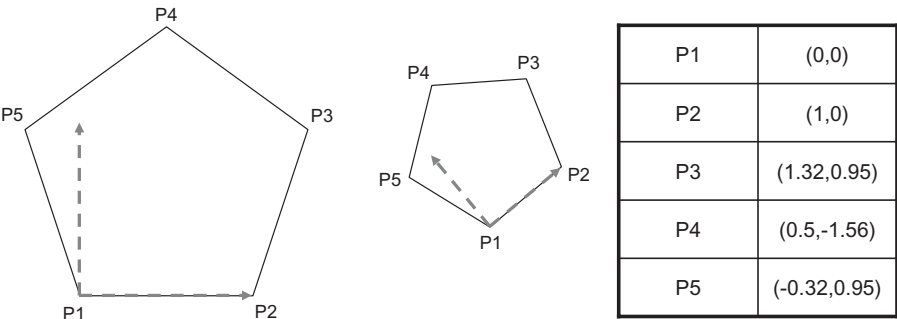


Figure 6.5 Similarity-invariant description of model vertices.

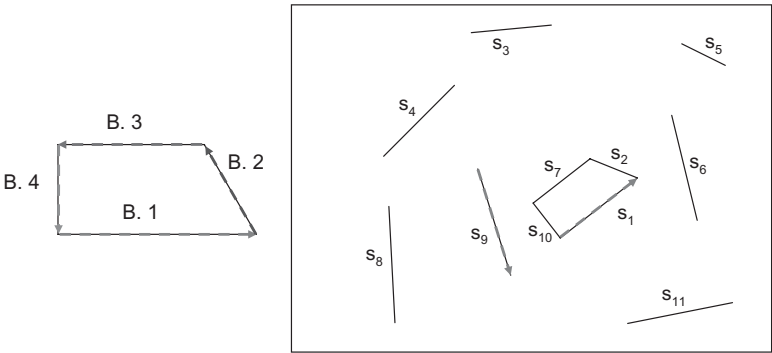


Figure 6.6 Matching the model to detected image features with a similarity-invariant description.

one of them as the x axis of a Cartesian frame, with the y axis orthogonal to it, and then describe all of the other vertices and segments in terms of this basis. Figure 6.5 shows an example of invariant description; all vertices are described by the same coordinates in both views.

By considering all possible bases, we build a hash table by storing for each entry the invariant description of the model segments (i.e., pairs of vertices). The *key* of each entry will be given by the basis, described by integer numbers obtained by discretizing the (x, y) coordinates. On the image data (Fig. 6.6), we can select a candidate feature as a basis, such as s_1 , and describe all of the other segments in invariant coordinates as well. Afterward, every other segment may look in the model hash table for a matching segment, and if found, vote for the corresponding basis (i.e., one of B_1, \dots, B_4). Referring to Table 6.1, selecting s_1 produces 4 votes for B_1 , given by the segments s_1, s_2, s_7 , and s_{10} ; the same happens for s_7 , which produces 4 votes for B_3 . Instead, by selecting s_9 we would collect only 1 vote for any model basis, which is given by the trivial correspondence $((0,0),(1,0))$ in invariant coordinates.

TABLE 6.1 Votes Collected for the Model Based on the Features of Fig. 6.6

Model Basis	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
B_1	4	1	1	1	1	1	1	1	1	1	1
B_2	1	4	1	1	1	1	1	1	1	1	1
B_3	1	1	1	1	1	1	4	1	1	1	1
B_4	1	1	1	1	1	1	1	1	1	4	1

Of course, since image segments are detected in random order, we must rely on random sampling to select a correct segment: in the worst case, after $11 - 4 = 7$ attempts.

One more tricky detail, not considered so far, is the *direction* of the segments detected, which may sometimes be reversed with respect to the model. To cope with this possibility, we consider each model segment *twice* as a possible basis, therefore doubling the size of the hash table in *both* dimensions (rows and columns). In this case, each image segment will have a chance to vote for the corresponding model, independent of its direction.

To extend the algorithm to three-dimensional models, several approaches are possible. One of them consists in casting the three-dimensional problem into a set of two-dimensional, similarity-invariant recognition tasks, by taking a discretized set of views, building a hash table for each view, and further indexing each entry by the view. Since the model will be matched up to a similarity, which already includes a planar rotation, we need to consider only the two in-depth rotations, which greatly reduces the number of views.

A further approximation that is required by this approach is to consider scaled orthographic projections in place of full perspective; this keeps our representation independent of the depth of the object, up to a similarity transform. These two approximations lead to imperfect model matching, even in the absence of measurement noise (localization of segments):

- The orthographic approximation, for which parallel model lines are kept parallel in the image, does not hold for a perspective camera; however, as long as the depth of the object is high with respect to its size, the error will not be too significant.
- The discretization among possible three-dimensional views, none of which will in general match the real data; this error could be reduced by incrementing the number of views, limited however by the computational complexity of the overall hash table.

Because of these errors, three-dimensional localization will be poor; therefore, we can refine it by means of nonlinear pose estimation, using all of the

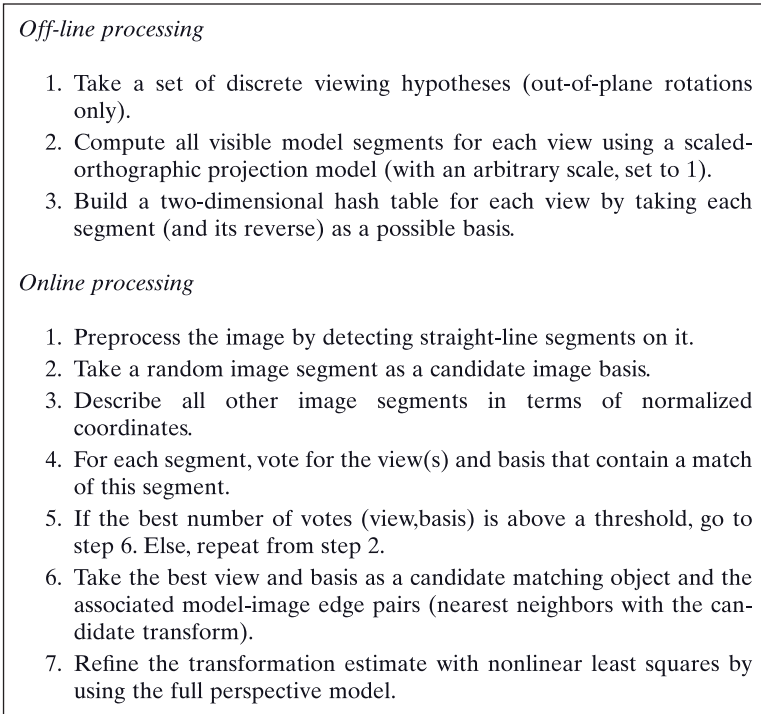


Figure 6.7 Geometric hashing procedure for three-dimensional object detection.

matches under a full perspective camera, as described in Section A.2. We finally note how each discretized view shows only a subset of model segments, which can be computed off-line by using GPU-based contour sampling facilities (Section 3.2).

The full three-dimensional object recognition algorithm is summarized in Fig. 6.7, and an illustrative example is given in Fig. 6.8. A given image segment (arrow) votes for the corresponding basis in a given view only (α_2, β_2) , where the model-image transform is up to a two-dimensional similarity. In reality, we will have a *cluster* of views, close to the correct one, that will receive many votes; in this case, we can take the local maximum within the cluster.

6.4 MONTE CARLO SAMPLING

In this section we describe a general-purpose method for detecting multiple objects with known visual models from multiple camera views [131]. This method is based on Monte Carlo sampling and weighted mean-shift clustering, and can make use of any combination of visual modalities with an arbitrary camera setup.

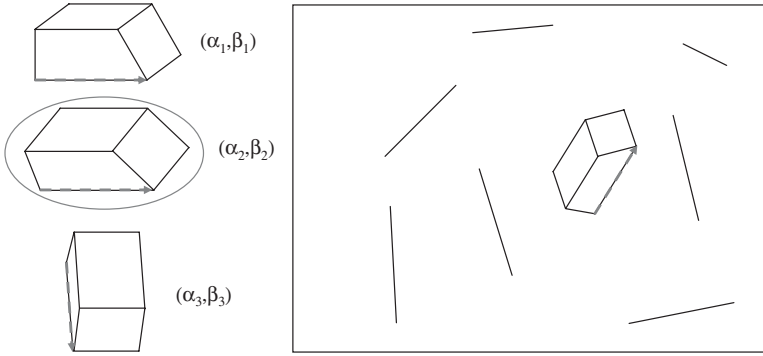


Figure 6.8 Three-dimensional geometric hashing: An image segment votes for the corresponding basis in the second view only (α_2, β_2) .

To detect targets, we look basically for local maxima of the overall likelihood. In the absence of any prior information about their state, we can initialize a particle with uniform distribution of states \mathbf{s}_i , covering the feasible state-space volume, where targets can be viewed through a single- or multi-camera setup. Each peak of the likelihood will be surrounded by a cluster of high-weighted particles, and a weighted *state-space clustering* algorithm can therefore be run.

To identify these modes we need a smooth representation of the underlying density, such as the *kernel-based* representation (Section 4.1). By restricting attention to isotropic kernels ($H = hI$) and a pure translational model $s = \mathbf{x}$, where \mathbf{x} is a point in world coordinates, the density can be optimized locally with the weighted mean-shift algorithm

$$\mathbf{m}_h(\mathbf{x}) = \frac{\sum_{i=1}^n w_i g\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \mathbf{x}_i}{\sum_{i=1}^n w_i g\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x} \quad (6.8)$$

where $g = -k'_p$, its first derivative of the kernel profile, and the position is updated by $\mathbf{x} \rightarrow \mathbf{x} + \mathbf{m}_h$. The iteration is stopped when the update becomes smaller than a threshold: $\|\mathbf{m}_h\| < \varepsilon$.

Initialization of the particle set requires computing the joint *viewing volume* of C cameras (Fig. 6.9), where each camera provides six *clipping planes* that define a truncated pyramid. If we denote by $\pi_{c,i}$, $i = 1, \dots, 6$ the world-related planes of camera c , the overall intersection polyhedron is defined by the homogeneous inequalities

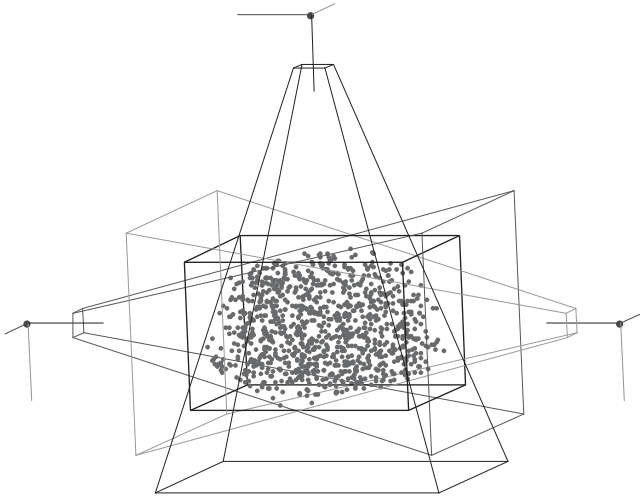


Figure 6.9 Uniform sample from the joint viewing volume of a three-camera configuration. (From [131].)

$$\begin{bmatrix} A_1 \\ \vdots \\ A_C \end{bmatrix} \mathbf{x} \leq \mathbf{0}, \quad A_c \equiv \begin{bmatrix} \pi_{c,1}^T \\ \vdots \\ \pi_{c,6}^T \end{bmatrix} \quad (6.9)$$

To sample uniformly distributed points from this polyhedron, we first sample from its *bounding box*, discarding all points that do not satisfy (6.9). For computing the box, the vertices of the polyhedron must be obtained explicitly; this can be accomplished by solving the *vertex enumeration* problem via the *primal–dual method* [47]. In Fig. 6.9 we show the result of the sampling procedure described above applied to a three-camera configuration. The three viewing volumes intersect in the central polyhedron, which is filled by uniformly distributed points. Its bounding box is also shown in black.

A multitarget detection example is given in Fig. 6.10. The object model is given by a yellow sphere of radius 65 mm, and as image likelihood we compute the Bhattacharyya distance between color histograms as described in Section 4.1.3:

$$P(z|s) \propto \prod_c \exp\left(-\frac{B^2(q, p_c(s))}{2\sigma^2}\right) \quad (6.10)$$

where q is the reference histogram, $p_c(s)$ the image histogram at pose s projected on camera c , B the Bhattacharyya distance, and σ^2 the measurement noise covariance.

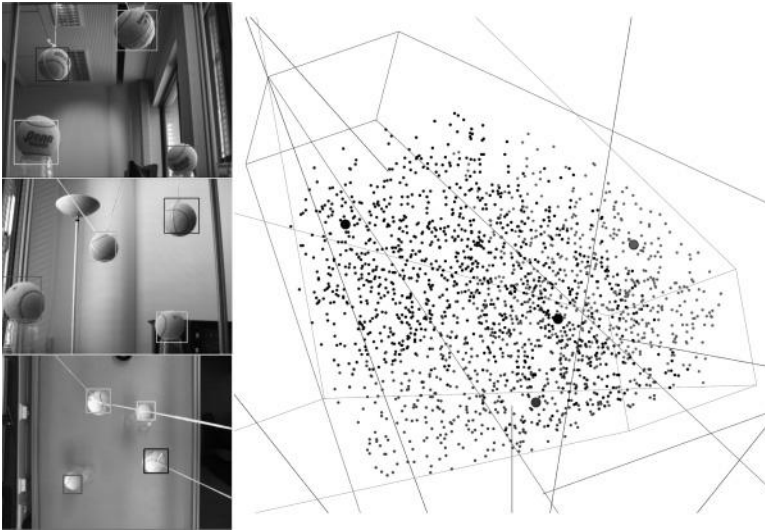


Figure 6.10 Detection results with a color-based likelihood. (From [131].)



Figure 6.11 Model-based object detection with SIFT keypoints. Multiple instances of the train model are found even in the presence of occlusions. (From [106].)

6.5 INVARIANT KEYPOINTS

The invariant keypoints described in Section 4.5.3, such as SIFT [106] or SURF [36], provide a powerful way of detecting and estimating the pose of planar objects with a significant texture. In fact, after detecting keypoints from a given viewpoint, relevant features can be back-projected in space and used for subsequent detection from different views of the same object.

An example of model-based object detection with SIFT keypoints is shown in Fig. 6.11, where multiple instances of an object can also be detected, by matching and clustering features and by performing pose estimation (in this example, a planar affine transform is used). It can also be seen how the method is robust to partial occlusions.



Figure 6.12 Three-dimensional pose estimation with invariant keypoints: GPU-based implementation.

The algorithm also works in the presence of multiple targets, provided that each image feature is matched to the respective model database. In this context, special care is required for targets sharing the same model, since a model feature may have multiple matchings in the image, and the system has to detect the presence of multiple instances. For this purpose, a clustering algorithm can be used before pose estimation (e.g., a Hough transform in pose space) [32].

After matching the SIFT descriptors, outliers may still be present and negatively influence the pose estimation task. For this purpose, robust methods such as RANSAC [65] can be used with the proper choice for the transformation to be estimated (e.g., three-dimensional Euclidean, two-dimensional affine). Finally, any pose estimation algorithm (Appendix A) can be employed in order to localize the object.

Figure 6.12 shows an example of planar object detection through invariant keypoint matching and RANSAC. In this example, invariant keypoint detection and matching [106] has been implemented for speed purposes on graphics hardware, following the paper by Ziegler et al. [172]. Invariant keypoints can also be used for nonplanar object detection [68,105]. In this approach, the object is rendered off-line into one or more key frames, depending on the shape, and SIFT keypoints are collected and back-projected onto its surface, creating multiple databases. However, similar views may show overlapping subsets of keypoints, and therefore a more sophisticated matching scheme has to be adopted, as described by Lowe [105].

The choice of the number and position of key frames may also be different for each shape. Planar surfaces require only one or two views, which has some

limitations, however, since the SIFT descriptor is not invariant to in-depth rotations. On the contrary, polyhedral surfaces need to be rendered and sampled from multiple views in order to capture a larger number of texture variations. Rounded surfaces offer the most difficult challenge, since a keypoint is only a planar approximation of the object curvature and therefore is hardly recognizable in different views.

Finally, a multicamera extension of this detector can be implemented directly as long as simultaneous views of the object are available for keypoint matching in a calibrated setup. As emphasized in Section 3.4.2, this may be the key for robust recognition and localization of complex models.

CHAPTER 7

BUILDING APPLICATIONS WITH OpenTL

In this chapter we describe briefly how the visual tracking framework proposed so far has been realized in the OpenTL software architecture. We first present the multilayer structure of OpenTL, and then develop an incremental tutorial for building an object-tracking application. Finally, we provide examples and results of more complex applications, realized by specifying the related modules of the tracking pipeline and the visual processing tree. The OpenTL library can be downloaded at the OpenTL Web page [20].

7.1 FUNCTIONAL ARCHITECTURE OF OpenTL

The main classes that OpenTL provides as building blocks for model-based tracking have been organized in *functional layers* (Fig. 7.1), representing an increasing level of abstraction, from base utility and data structures up to application-related functionalities. In particular, from top to bottom we have:

1. *Tracking data*: raw sensory data; any preprocessed data related to a given visual modality; measurement variables and residuals associated with a given target; multitarget state distribution

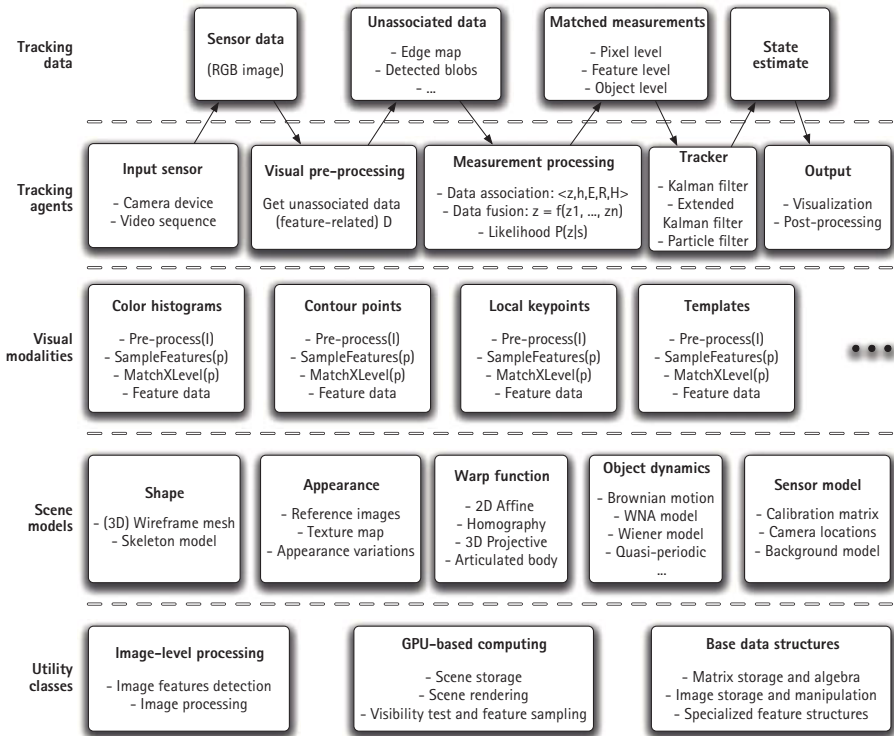


Figure 7.1 Layered class architecture of OpenTL.

2. *Tracking agents*: image acquisition, preprocessing, measurement processing (sampling, matching, data fusion) and likelihood computation; Bayesian tracking; output visualization and postprocessing
3. *Visual modalities*: the common abstraction for visual modalities: preprocessing, sampling, and updating model features, matching by re-projection, and data fusion; within each class, storage for off-line and online model features, and intermediate processing results
4. *Scene models*: object shape and appearance, pose parameters, dynamics, sensor models, and related object-to-sensor warp function, with Jacobian computation and back-projection
5. *Utility classes*: low-level, model-free image processing and computer vision algorithms; data storage for matrices, vectors, and images; basic algebra and image manipulation utilities; GPU-assisted scene rendering and visibility testing of geometric primitives under a given pose and camera view

This architecture has been implemented in C++ code, currently consisting of about 200 classes, in a platform-independent way for Windows and

Linux operating systems (OSs). In particular, it depends on a few open-source libraries, OpenCV [15] and OpenGL [17], respectively, for image- and model-based processing as well as for GPU programming, and the Boost C++ library [3].

7.1.1 Multithreading Capabilities

The tracking pipeline described above offers several possibilities for a parallel implementation, which on a multicore architecture can be obtained by means of independent processing *threads*:

- When tracking independent targets without considering mutual interaction or occlusion, the Bayesian filter becomes a bank of independent single-target filters which can therefore run in parallel and possibly use a different processing tree per target.
- A multihypothesis Bayesian tracker (Chapter 5) such as a particle filter or unscented Kalman filter requires, for the correction step, evaluating measurements related to independent state predictions, which may be ensemble or single-state hypotheses, by performing local data association over the same image data; therefore, individual hypotheses can be evaluated in parallel. For this purpose, the entire processing tree can be *cloned* into each thread in order to avoid any access conflicts.
- The pipeline itself for single-state measurement processing consists of several visual modalities, connected in cascade as well as in parallel (see also the examples in Section 7.3); therefore, multiple modalities on the same level of the tree can be processed in parallel, and independently deliver their outputs to the next module, which will combine them as soon as they are all available.

The *Boost* C++ library [3] offers convenient and easy-to-use multithreading facilities which are used in OpenTL for the three types of parallel processing mentioned above. However, at any level of the processing hierarchy, there are potentially many more possibilities for parallelization that we did not consider here and that may also be exploited by means of GPU programming.

7.2 BUILDING A TUTORIAL APPLICATION WITH OpenTL

To introduce the OpenTL application programming interface (API), we build a step-by-step application: a monocular planar object tracker, based on color and edge features, using a particle filter. This tutorial describes how to achieve the goal in seven main steps while introducing the relevant classes and methods.

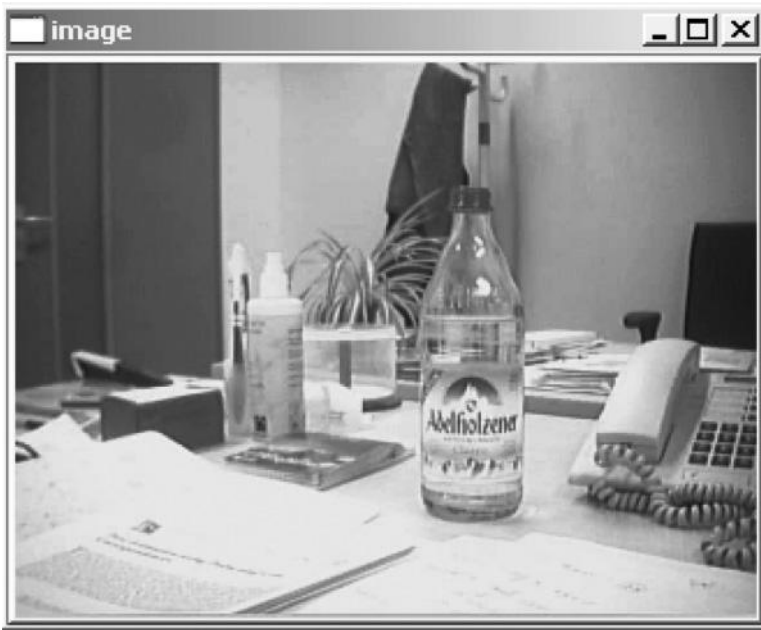


Figure 7.2 Tutorial 1: Capture and display the camera stream online.

7.2.1 Setting the Camera Input and Video Output

The first step in a visual tracking application consists of grabbing frames from the camera and displaying the video signal online (Fig. 7.2).

OpenTL currently supports IEEE-1394 FireWire camera inputs as well as standard Webcams, from one or multiple devices, for any OS supported. In particular, the main abstraction `opentl::input::ImageSensor` is provided, from which the following classes are derived:

- `opentl::input::FirewireCamera`: IEEE-1394 FireWire cameras for both OSs
 - `opentl::input::LinuxDC1394Camera`
 - `opentl::input::WindowsCMU1394Camera`
- `opentl::input::OpenCVCamera`: standard Webcams, handled through the OpenCV capture functions

To describe both camera types, for this tutorial we provide two versions. In the code they can be selected at startup by specifying a command-line parameter. In particular, a command-line option makes it possible to switch the type of camera, so that the user may specify it at startup by calling

```
./tutorial# -camType <GUPPY|SONY|OPENCV>
```

Command line options:			
Option	Default	Current	Description
-help	(<*>)false	true	display a help message
-canType	GUPPY	GUPPY	camera type to use <OPENCV ! GUPPY ! SONY>
(<*>): mandatory argument			
(<*>): argument was set on commandline			

Figure 7.3 Output of tutorial 1, called with the help option.

To display all available command-line options, the user may call the application with

```
./tutorial# -help
```

The output help for tutorial 1 is shown in Fig. 7.3.

FireWire Camera Input The abstract class `FirewireCamera` from the `opentl::input` name space provides platform independency, so that we can write the same application code for both OSs. At compile time, it is decided automatically which inherited class, `LinuxDC1394Camera` or `WindowsCMU1394Camera`, to instantiate and link to the code.

In particular, under Windows this class uses the *CMU library* [7] drivers, whereas on Linux the standard `libdc1394` is used. The first instruction,

```
opentl::input::FirewireCamera::getNumberOfFirewire  
Cameras();
```

retrieves the number of FireWire cameras currently connected to our machine. Afterward, we select one device (e.g., number 0) and create the camera class

```
FirewireCamera::createFirewireCamera(0);
```

We notice that both instructions are static function calls; after creating the camera, we can use the internal class methods. The next call,

```
camera->open();
```

initializes the device, and

```
camera->captureStart();
```

starts the acquisition engine. So far we have assumed use of a camera with `FORMAT_7` pixel coding, which requires a *de-bayering* mechanism to pack input pixels into the standard RGB format. This is the default choice, but of course there are several other possibilities; however, in this first tutorial we cover only the basic functionalities and refer the user to the `FireWireCamera.h` header file for more details.

If we desire to set an automatic color adjustment, we can use

```
camera->enableWhiteBalanceAuto(true);
```

as well as automatic gain adjustment, to the environment light

```
camera->enableGainAuto(true);
```

Next, we declare an image that will contain the camera input. In OpenTL, images are handled through the `Image` class of the `core::cvdata` namespace, built on top of the OpenCV `IplImage` data structure, and contains the necessary facilities for allocation, accessing, and copying.

```
Image( int initWidth, int initHeight,  
       ColorChannels initChannels = RGB,  
       BitsPerChannel bpc = BPC8U);
```

The first two arguments specify the image resolution, the third the number of channels, and the last the number of bits per channel, which reflect the OpenCV image types. For example, RGB are three-channel color images, and BPC8U specifies an 8-bit unsigned char storage per channel. For a complete list of types available, refer to the `Image.h` header file.

The camera input class already contains an image with the corresponding resolution and pixel format, whose pointer we can obtain using the function `image()`. Moreover, to record the input to a video file, we declare a video writer using the OpenCV structure `CvVideoWriter`:

```
cvCreateVideoWriter( "video.avi",  
                    codec, 30.0,  
                    cvSize(camera->width(),  
                             camera->height(), 1)
```

where `codec` is a suitable video format; in our example we use the XViD codec, given by the OpenCV macro `CV_FOURCC('X','V','I','D')`.

Now we can start the online loop, which performs the following operations:

1. Acquire a new frame:

```
camera->captureNext();
```

2. Get the camera image as a pointer:

```
srcImage = camera->image();
```

3. Display it onto an OpenCV-HighGUI window:

```
cvShowImage("image",srcImage->iplImage());
```

4. Add it to the output video:

```
cvWriteFrame(video, srcImage->iplImage());
```

In this call, we use the method `iplImage()` to retrieve the internal pointer to the OpenCV image structure:

```
IplImage* iplImage();
```

Similarly, matrices and vectors can also access the internal `CvMat` pointer by calling `Matrix::cvMat()`, `Vector::cvMat()`. The loop can be exited at any time by pressing the `esc` key.

USB Camera Input In a second version, we consider a simple Webcam device that can be handled by the OpenCV-HighGUI functionalities, through the `CvCapture` data structure. For this purpose, the class `opentl::input::OpenCVCamera` contains most of the same methods and data of the `ImageSensor` abstraction; we do not repeat the previous description here, only point out the principal differences.

- The constructor for `OpenCVCamera` is direct, unlike the virtual constructor `opentl::input::OpenCVCamera(0)`.
- We do not yet provide automatic gain and color balance adjustments for USB cameras.
- At the end, we call `delete camera;`, which in turn calls the virtual destructor `deleteFirewireCamera()`.

No other modifications are necessary.

7.2.2 Pose Representation and Model Projection

Now we learn how to specify and use the degrees of freedom of the object model. For this purpose we start with a simple task: to project a few points from object space to camera screen under a given pose representation (Fig. 7.4). The first step consists of declaring the pose class. We choose a simple two-dimensional pose consisting of a uniform scale s and planar translations t_x and t_y ; the corresponding transformation matrix is given by

$$T = \begin{bmatrix} s & 0 & 0 & t_x \\ 0 & s & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.1)$$

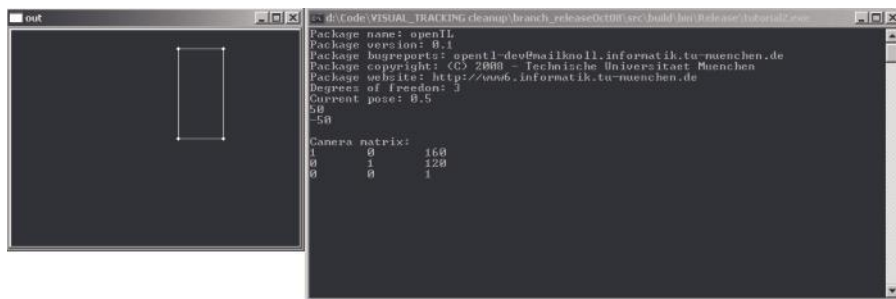


Figure 7.4 Tutorial 2: Draw a simple shape model at a given pose (two-dimensional translation and scale).

This is accomplished by declaring a variable of the class `Pose2d1ScaleTranslation`:

```
opentl::core::cvdata::Pose2d1ScaleTranslation pose(
    opentl::core::cvdata::Pose::COMPOSITIONAL);
```

Notice that this class belongs to the `opentl::core::cvdata` name space, and in particular to the `POSE_2D` subset of types, which is the set of planar, uncalibrated transformations. Furthermore, the *update mode* has to be specified in the constructor. Here one may choose between `Pose::COMPOSITIONAL` and `Pose::ADDITIVE`, where the former corresponds to an exponential representation for the incremental parameters, in the respective Lie algebra.

Afterward, insert this pose representation into a *state*, which may include kinematic parameters such as velocity and acceleration. For this purpose we need the class `State`, which is part of the `opentl::core` name space. Moreover, to access states in a thread-safe way, each state is created through a shared pointer from the *Boost* library as

```
boost::shared_ptr<opentl::core::State> state;
```

The state must be initialized with the pose type and data, and we accomplish this by using the class defined above as a template:

```
state.reset(new opentl::core::State(pose));
```

where two nonspecified flags, by default false, add velocity and acceleration variables to the state. We can read the overall number of state parameters (which is three) with

```
state->stateDimension()
```

To set the parameters, we declare a vector with three components, from the `math` name space; for example,

```
math::Vector pose_data(dof);
pose_data[0] = -50;
pose_data[1] = 50;
pose_data[2] = -50;
```

and set the state with

```
state->setPoseData(pose_data);
```

Notice that scale factors are represented in a logarithmic way, so that a scale parameter -50 corresponds to the actual scale s :

$$s = 1.01^{-50}$$

where the base 1.01 is used to achieve a numerically convenient representation.

Next we declare the camera model. This is done by using the same `ImageSensor` class from which the input cameras have been derived:

```
opentl::input::ImageSensor sensor;
```

A calibrated camera setup requires two sets of information for each camera: internal parameters (acquisition model) and external parameters (relative world location). However, for two-dimensional transforms, an affine model can be provided where no external parameters are given so that world coordinates coincide with image coordinates in pixels and the only parameters are the horizontal and vertical image resolution.

The latter is accomplished by setting

```
sensor.setK(focus, xres, yres);
```

where `xres`, `yres` is the resolution; `focus` is not used and is therefore set to 1. When the warp function is called for a two-dimensional pose (see below), the internal matrix K will be set to

$$K = \begin{bmatrix} 1 & 0 & 0 & r_{y_1}/2 \\ 0 & 1 & 0 & r_{y_2}/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.2)$$

whereas for a three-dimensional pose it assumes a perspective K matrix by swapping the last two columns. The world origin then resides at the center of

the image, with y pointing down and x pointing to the right. The overall mapping, from object to screen homogeneous coordinates, is given by

$$\bar{\mathbf{y}} = K \cdot T \cdot \bar{\mathbf{x}}$$

$$\mathbf{y} = \begin{bmatrix} \bar{y}_1 & \bar{y}_2 \\ \bar{y}_3 & \bar{y}_3 \end{bmatrix}^T \quad (7.3)$$

In our framework, multiple cameras can be present, so we need to put them into a `std::vector` of pointers; here no shared pointers are required, since cameras are read-only devices:

```
std::vector<input::ImageSensor *> sensVector;
```

Now we can declare the main class for image projection: the `Warp`. This class is part of the `modelprojection` name space, and it has a unique instance throughout any application, also called a *singleton* class. Its semantics consist of managing the relationship between all object and camera spaces. In other words, it acts as a “dispatcher,” projecting points between the spaces according to the current pose, and if required, computing screen Jacobians.

```
opentl::modelprojection::Warp warp(sensVector);
```

The `Warp` class keeps a pointer to the sensor models and each time gets a pose class, both for updating its internal transform matrices and for computing individual point projections and derivatives.

In particular, whenever the pose changes, the `Warp` class requires updating of the internal projection matrices and Jacobians before projecting individual points. This is obtained by calling the `warpUpdate` function

```
warp.warpUpdate(*state);
```

Now we take a few points in homogeneous three-dimensional coordinates (`math::Vector4`), corresponding to the four corners of a square:

```
math::Vector4 objP1(-50, -100, 0, 1);
math::Vector4 objP2(50, -100, 0, 1);
math::Vector4 objP3(50, 100, 0, 1);
math::Vector4 objP4(-50, 100, 0, 1);
```

where the z coordinate is set to 0; this is an arbitrary choice for two-dimensional poses, since any nonzero z is ignored. The corresponding nonhomogeneous screen coordinates are of type `math::Vector2`.

```
math::Vector2 screenP1, screenP2, screenP3, screenP4;
```

Screen projection is accomplished by the `warpPoint` function:

```
void warpPoint( cvdata::Pose& objPose,
               const math::Vector4& objPoint,
               math::Vector2& screenPoint,
               std::vector<math::Vector2>* jacobian = NULL,
               int cameraIdx = 0, int linkIdx = 0);
```

In this function, the first argument is a `Pose`, the second argument comprises the homogeneous object coordinates, and the output is given by the screen coordinates and, optionally, the Jacobian $2 \times \text{dof}$ matrix

$$J \equiv \frac{\partial \mathbf{p}}{\partial \mathbf{p}}$$

Finally, `cameraIdx` specifies the viewing camera index and `linkIdx` the part to which this point belongs for articulated or deformable shapes; by default, both are set to 0.

We call this function four times, for each point:

```
warp.warpPoint(state->pose(), objP1, screenP1);
warp.warpPoint(state->pose(), objP2, screenP2);
warp.warpPoint(state->pose(), objP3, screenP3);
warp.warpPoint(state->pose(), objP4, screenP4);
```

To plot the result we can declare a color image with the same resolution of the sensor device:

```
opentl::core::cvdata::Image img(xres,yres)
```

and afterward cast the screen points into a `CvPoint` structure:

```
CvPoint pt1 = cvPoint(cvRound(screenP1[0]),
                      cvRound(screenP1[1]));
```

and similarly for the others. This makes it possible to use the OpenCV drawing functions `cvCircle` and `cvLine`, which we use to draw the shape: for example,

```
cvLine(img.iplImage(), pt1, pt2, CV_RGB(255,255,255), 1);
```

At the end we should get the output shown in Fig. 7.4.

7.2.3 Shape and Appearance Model

In the previous step we learned how to represent an object pose through the `Pose` class and to project individual points in camera space through the `Warp`

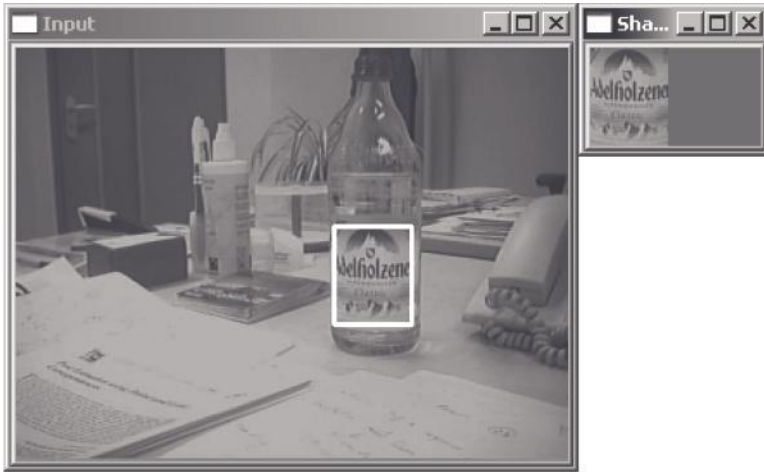


Figure 7.5 Tutorial 3: From the online camera input, take a reference shape and appearance model.

class. Now we consider the full shape model, represented by a set of connected primitives (i.e., polygons, edges, and vertices) of a CAD model. This provides an OpenGL-style representation, which, however, contains no information about the surface appearance, which is also needed for color-based tracking. Therefore, in this step we take from the online camera input a simple but complete shape and appearance model, consisting of a rectangular shape and a single key frame; this model can be initialized manually (Fig. 7.5).

The main class introduced here is called `ShapeAppearance`, which is part of the `opentl::core::cvdata` name space:

```
opentl::core::cvdata::ShapeAppearance shapeApp;
```

We begin by declaring the OpenCV display window, on which we set up a mouse callback function, through the HighGUI functionalities

```
cvNamedWindow( "Input", 1 );
cvSetMouseCallback( "Input", mouseCallback, 0 );
```

where the `mouseCallback` function takes care of mouse events happening over the window: in particular, left button press and release, and dragging the pointer. As a result, this callback provides a rectangular region of interest (ROI) from which get the shape and appearance model.

In order to interface the callback with the main application, we need to declare some global variables: the camera image (`cvdata::Image` *

image), the output ROI (`CvRect selection`), and a few static variables for the callback itself, such as button press/release flags.

Afterward we set up the camera and run the input as in step 1. To store the appearance model, we declare an additional color image,

```
core::cvdata::Image * appearance = NULL;
```

which will be allocated at the right time with the ROI size. In the main loop, the camera image is updated and the mouse callback handles the user interaction until a rectangle has been dragged.

This event sets the flag `ready_shape` to true so that the model can be taken. At this point we declare the shape a rectangle with the size of the ROI but centered about the origin (0,0,0) in object coordinates. This is done by the `initRectangle` function,

```
shapeApp.initRectangle(selection.width, selection.height);
```

Afterward, we allocate the appearance image, and copy the ROI contents using the OpenCV `cvSetImageROI` and `cvCopy` functions.

This image must now be inserted into the `shapeApp` class. Since an object appearance can be specified by multiple key frames that can also be used as texture maps, we put the image into a `std::vector`:

```
std::vector<boost::shared_ptr<core::cvdata::Image> >
    imageList(1);
```

Moreover, each image is identified and retrieved through its name, to avoid the confusion arising from numerical indexes. Here we give it the name `MyAppearance`:

```
std::vector<std::string> namesVec;
namesVec.push_back("MyAppearance");
```

and finally insert it into the model:

```
shapeApp.initAppearance(imageList, namesVec);
```

We can verify the internal appearance model by retrieving this image by name:

```
shapeApp.getMaterialByName("MyAppearance")
```

Notice that for the sake of generality, we call the appearance images *material*, since they may also represent textures in the presence of a texture map.

7.2.4 Setting the Color-Based Likelihood

So far, we have obtained a model of the object to be tracked and have learned how to map it to camera space. The next task is how to compare the model projected with the image data under a given visual modality. This is the *likelihood* function

$$P(z|s)$$

where s is the object state and z is the *measurement* associated with the object. This function expresses the probability of the data observed if the true state were s ; in other words, it models our sensory processing system in a probabilistic form, where the uncertainty may arise from many sources, including signal noise and modeling errors.

To understand this step, we first recall our `Modality` abstract class from the `opentl::modalities` name space, defined by the following abstract methods:

1. `preProcess`. Processing of model- and state-independent images, producing data suitable for subsequent matching.
2. `sampleModelFeatures`. Collect visible features from the shape and appearance model at predicted pose s (also called *off-line* features).
3. `match{Pix/Feat/Obj}Level`. Match model and image data and produce output at one of three levels: pixel maps, associated features, or a maximum-likelihood state estimate.
4. `updateModelFeatures`. Update model functions from image data after updating the state s update; also the the *online* version of the reference features.

Here we consider a color-based modality using histograms in hue–saturation space (Section 4.1). In OpenTL, this modality, called `ColourHist2D`, inherits from the `Modality` abstraction:

1. `ColourHist2D::preProcess`. Convert the camera image from RGB to HSV color space.
2. `ColourHist2D::sampleModelFeatures`. Collect the HSV histogram from the reference appearance model (off-line).
3. `ColourHist2D::matchPixelLevel`. Perform a pixel-wise color segmentation using the reference histogram, obtaining a binary (or gray-value) map.
4. `ColourHist2D::matchFeatureLevel`. Collect the underlying image histogram under the pose predicted and compute the Bhattacharyya coefficient.
5. `ColourHist2D::updateModelFeatures`. From image data, after updating the state s , collect and update the online reference histogram.

Here matching is done at the feature level; therefore, our measurement data comprise a color histogram, to be compared to the corresponding model histogram via a suitable metric e , such as the Bhattacharyya distance.

After the metric has been defined, for the probabilistic model $P(z|s)$ we also need to know the *covariance* of the residual noise, R , so that we can use a Gaussian model

$$P(z|s) \propto \exp\left(-\frac{e^2}{2R^2}\right)$$

The first part of the code for this tutorial is the same as step 3. Set up the camera input and mouse callback, and obtain the shape and appearance model. Afterward, we need to encapsulate all model information in a unique container class, called `ObjModel`:

```
ObjModel obj(&shapeApp,
             core::cvdata::CvData::POSE_2D_1SCALE_TRANSLATION);
```

which is part of the `opentl::core::models` name space. In the constructor we pass the shape and appearance information and the degrees of freedom, this time given directly as a data type, `POSE_2D_1SCALE_TRANSLATION`. The third argument, which is optional, specifies predefined limits for the pose parameters, which can be enforced during tracking. As we will see in the next tutorial step, this constructor, as a fourth argument, accepts a motion model of the class `models::Motion::MotionType`, which we leave to the default value `BROWNIAN` here, since we do not need it.

To provide a convenient way of handling informations about objects being tracked, with possibly different Bayesian filters, the entire state distribution is stored within a `Target` class, taking care to represent single instances of any object (see Section 5.1). Therefore, the `Target` class is part of the `opentl::models` name space; due to thread-safety issues, it is declared a Boost shared pointer. Since multiple targets can possibly be present in the system, we declare a vector of targets,

```
std::vector<boost::shared_ptr<models::Target> > targetVec;
```

and since targets belong to a respective object model, instances are created directly from the `TargetFactory` static class:

```
models::TargetFactory::addNewTargetsFromModel(objectModels.back(), targetVec, 1);
```

which inserts one new target in the vector, containing the corresponding state distribution, initially with a single state, and a pointer to the parent model. As

we learned in step 2, to map from object to camera space, we also need the `Warp` class and a `SensorModel`; this is done in the usual way.

Next we declare the color-based modality

```
ColourHist2D * pColHisto =
    new modalities::ColourHist2D(warp, camIdx)
```

where the camera number `camIdx` must be specified, since for multicamera systems we have a modality instance for each camera. Before using this class, we need to set some parameters; since each modality may have many parameters for its processing functions, of very different type and semantics, we introduce an abstract class `core::util::ParameterContainer`, from which any modality also inherits. In general, we also distinguish between *online* and *off-line* parameters, where the former are allowed to change during processing, while modifying the latter forces the program to do a reinitialization of the modality.

For color histograms, we have the following relevant feature-matching parameters:

- `ColourFormat preProcess_DestColorFormat`: output color space for the preprocessing step, in our case `cvdata::Image::HSV`
- `int sampleModelFeatures_Bins1/2/3`: number of bins for each histogram channel
- `double matchFeatVariance`: the variance R^2 for the feature-level likelihood
- `bool matchFeatWithMdetect`: a flag, specifying whether to match the off-line reference histogram (after features sampling)
- `bool matchFeatWithMtrack`: whether also to match the online reference histogram (after features update)
- `double matchFeatMtrackWeight`: weight for the online histogram distance
- `double matchFeatMdetectWeight`: weight for the off-line histogram distance
- `Modality::MeasComputeFlags`: common flags to all modalities, defining which data are going to be stored in the measurement structure

The last parameter specifies which of the five fields of measurement data [eq. 3.1] are needed by the subsequent module in the pipeline. In this case, there is no further data fusion and the next module is the `Likelihood` class, which needs only the measurement residual e and covariance matrix R ; therefore, the flags will be `COMPUTE_E` and `COMPUTE_R`. Other possible flags, which are not used here, are `COMPUTE_Z` and `COMPUTE_H`, respectively, returning

the observed and expected histograms, and `COMPUTE_JAC` for computing the Jacobian of h with respect to the state.

Each parameter may be set to the desired value by

```
pColHisto->setParameter<paramT>
(opentl::modalities::ColourHist2D::param_name, value);
```

with *paramT* the parameter type and *param_name* the name, as specified in the list above. Notice that if the user does not set a value for a parameter at this stage, the modality will be initialized with default values. Finally, to fix the parameters and perform the initialization, the user must call

```
pColHisto->init();
```

The next class of the processing pipeline is

```
modalities::Likelihood likelihood;
```

This class is the root of the visual processing tree (Fig. 3.15) and handles all of the connected visual modalities and static fusion classes, to collect the target-related measurements for the Bayesian tracker. In particular, it has two main methods:

1. `implicitModel`: $P(z|s)$, with P the overall likelihood
2. `explicitModel`: $z = h(s) + v$, with v the explicit measurement noise with covariance R , and h the expected measurement

For a particle filter, we require the `implicitModel`, as described in the next step; instead, for an extended Kalman filter the `explicitModel` stacks together the measurement residuals $e = z - h$ (also called *innovations*) and the noise covariance R . This class also provides *dynamic* data fusion, as described in Section 3.4, either by stacking together multimodal and multicamera data (explicit model), or by computing the joint likelihood (implicit model), whereas static fusion is handled by dedicated classes of the `Modality` abstraction.

To build the pipeline, we need to connect the likelihood class to the visual modalities by building the *tree* structure. For the present tutorial we use only one modality, which we pass as a template at construction time,

```
likelihood.addChild(pColHisto,
modalities::Modality::FEATURE_LEVEL);
```

which internally clones and stores the `Modality` template, through the method `Modality::clone()`. The specification `Modality::FEATURE_LEVEL` states that the likelihood class internally calls a feature-level matching: `matchFeatureLevel`.

In the case of multiple threads, the entire tree is cloned recursively by calling the `Likelihood::clone()` method, so that multiple measurements for different state hypotheses may be called in parallel. This is decided internally by the Bayesian tracker, which stores the processing tree at construction time.

Some modalities require an off-line sampling of visual features from the shape and appearance model. This is the case for `ColorHist2D`, where `sampleModelFeatures` collects a reference histogram from a key frame; in particular, we call this function from the `likelihood` class, which calls the same method recursively for all children:

```
likelihood.sampleModelFeatures(targetVec);
```

For multithreading, the number of particles is divided into *partitions*. In the present case, threading is out of scope, and we therefore set the number of active partitions to 1 and set the pointer to the only active state to be processed:

```
targetVec[0]->resizeActivePartitions(1);
targetVec[0]->setActiveState(0, targetVec[0]->state(0));
```

Before starting the camera loop, we fix the object pose to the center of the image:

```
poseVec[0] = 0;
poseVec[1] = 0;
poseVec[2] = 0;
targetVec[0]->state(0)->setPoseData(poseVec);
```

and update the internal transformation matrices:

```
warp->warpUpdate(*targetVec[0]->state(0));
```

Now we can compare the incoming images with the reference model, under the given pose, to get the likelihood values. For this purpose, we first do color conversion:

```
pColHisto->preProcess(srcImage, preProcessROIs);
```

where `srcImage` is the input image and `preProcessROIs` is a vector of ROIs, declared as 4-tuples of integers: upper-left corner (x,y) , width, and height. They define the only regions where preprocessing should take place, to save computations, according to the target distribution predicted. If this information is not given, all of the image will be preprocessed.

Finally, the likelihood is computed by calling

```
likelihood.implicitModel(targetVec, 0);
```

where 0 is the partition number. The likelihood value will be stored as the *weight* of the active state; therefore, we can retrieve it by

```
targetVec[0]->activeWeight(0)
```

To show the projected object shape, we also introduce a static function, which is part of the output name space:

```
output::Output::drawPose( cvdata::Image& img,
                           cvdata::CvData& visFeat,
                           modelprojection::Warp& warp,
                           cvdata::Pose& pose,
                           int thickness = 1,
                           CvScalar color=cvScalarAll(255.0),
                           bool drawEdgeNormals=false,
                           bool drawEdges=true
                           int camIdx = 0);
```

The first argument is the output image, the second specifies the type of data we wish to visualize, normally its `object::ShapeAppearance`. Then we need the `Warp` class and the specific `Pose`, and finally some data related to drawing: the line style and color, whether to draw screen edges and normals (the latter are useful for debugging purposes), and the camera view. In our case we have

```
output::Output::drawPose( outImage,
                           obj->getShapeApp(),
                           *warp,
                           targetVec[0]->state(0)->pose());
```

Figure 7.6 displays the likelihood values on the output console. In this application the maximum likelihood should be observed when the interior of the projected rectangle coincides with the reference image.

7.2.5 Setting the Particle Filter and Tracking the Object

In this part of the tutorial we “close the loop.” Based on visual data, we wish to update our knowledge of the object state in a Bayesian prediction–correction scheme. For this purpose, OpenTL offers several filters that can be used in different situations, such as single or multiple targets, linear or nonlinear measurement or dynamics, Gaussian or non-Gaussian noise distributions. These filters are provided inside the `tracker::` name space, and all are derived from the common abstraction `Tracker`, which has three virtual methods:

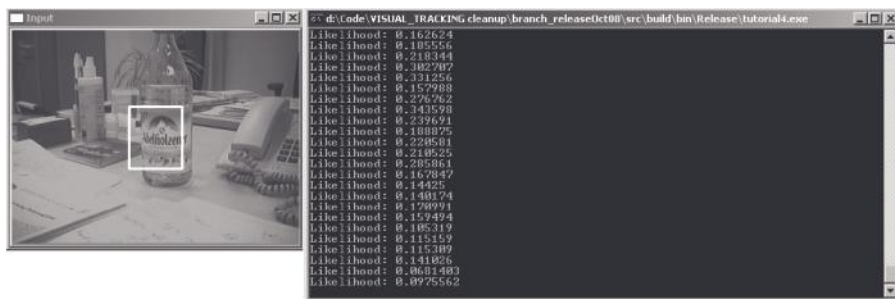


Figure 7.6 Tutorial 4: Color-based likelihood evaluation between the reference model (Fig. 7.5) and the current image under a fixed rectangular region. The likelihood is computed in realtime while moving the camera around the object.

1. `init()`. Initialize the filter with a prior distribution, usually uniform or Gaussian.
2. `predict()`. From the previous posterior, generate the prior distribution using the dynamic model.
3. `correct()`. For each hypothesis predicted, perform a measurement by calling one of the likelihood methods `explicitModel` or `implicitModel` and update the new posterior. Following this, compute the average state and covariance for output visualization and features update.

The prediction step always requires a motion model, which can be passed at construction time to the class `ObjModel` as a fourth argument. In particular, motion models are defined in the class `models::Motion::MotionType`, which describes several autoregressive models of first and second order, including constrained and unconstrained Brownian motion, constant velocity, and damped oscillatory dynamics. By default, the object model sets the dynamics to `BROWNIAN`, which is an unconstrained Brownian model in which only the pose is part of the state, while velocity is a white Gaussian noise. After specifying the model, we also need to set its process noise covariance:

```
math::SquareMatrix motionNoiseCovarianceMatrix(dof);
```

which for AR models always has the dimensions of the pose. For Brownian motion, the diagonal of this matrix contains the squares of the expected maximum frame-to-frame velocity. For example, a value of 10,000 means that we expect the object to move about 100 pixels in either direction and have a scale of the same amount (with respect to the logarithmic parameter). Then we set this matrix into the object model:

```
objModel->motion->setNoiseCovariance(motionNoiseCovarianceMatrix);
```

For initialization, we also need to specify the absolute prior, which is usually set to a uniform distribution. This is accomplished by the instruction

```
objModel->motion->setUniformNoiseRange(uniformNoise
                                         Range);
```

where `uniformNoiseRange` is a `math::Vector` of the same dimensions as the pose parameters, containing the sizes of the bounding box (symmetric) around the initial pose of the target, where a uniform sample will be drawn by the `Tracker::init()` function. This parameter is used only by Monte Carlo filters, whereas Kalman-based filters require an initial Gaussian distribution. We also notice that the motion noise covariance concerns *incremental* pose parameters (for prediction), whereas the initial distribution refers to *absolute* parameters, which may be different in case of a compositional update, which however is not this case.

For the present color-based tracker, we have a highly nonlinear measurement model, given by the Bhattacharyya histogram distance, and we wish to avoid Jacobian matrices. This case is suitable for a sample-importance-resampling (SIR) particle filter, that we can instantiate right after the likelihood declaration with

```
SIRParticle particle(likelihood, *warp);
```

The constructor accepts a likelihood class as the root of the processing tree, which internally instantiates a clone. From that moment on, if we need to access the Modality methods, we have to do it from the Tracker class. In particular, to sample the off-line reference histogram, we call

```
particle.sampleModelFeatures(targetVec);
```

which calls the internal likelihood method, in turn calling the `ColourHist2D` sampler using the given target model. To initialize the particle filter, the number of hypotheses must be specified, by specifying a vector containing the number of particles per target:

```
std::vector<int> nOfParticles(1, 200)
```

where in our case, with 3 dof, a value of 200 is sufficient for an acceptable tracking quality. For initialization, we specify the initial average state as the central position of the rectangle selected, and pass this parameter to the `init()` function:

```
particle.init(targetVec, initState, NULL, nOfParticles);
```

with `initStates` a vector of *Boost* shared pointers of states:

```
std::vector<boost::shared_ptr<core::State> > initState;
```

so that the uniform sample bounding box, specified above, will be centered around this state.

Now, to track the object, we simply call in the main camera loop the methods

```
particle.predict(targetVec);
pColHisto->preProcess(srcImage, preProcessROIs);
particle.correct(targetVec);
```

where the `preProcess` method should be called after prediction if the ROIs predicted are going to be used. The `correct()` step will compute the particle weights by calling the `Likelihood` function `implicitModel` for each hypothesis, then estimate the output average and covariance matrix. The resampling step will be done instead by the `predict()` function before drifting the particles.

We notice here that `preProcess` is the only method that should be called from the original Modality template `pColHisto`, and not from the `Tracker` or `Likelihood` classes. This is meant for efficiency reasons, especially in multithreading, since multiple modality trees may share the same preprocessed data, which should be obtained only once. Therefore, the preprocessed data are always stored into static variables and computed by calling the method from the base modality, which has been cloned in the tree instances.

Afterward, we can display the output of our tracker by taking the internal output state of the respective target:

```
targetVec[0]->outputState()
```

that corresponds to the weighted average of the particle states for target 0. The function that provides visualization of our model is `output::Output::drawPose`, as described in Section 7.2.4.

Figure 7.7 is a snapshot of the object defined previously, tracked in real time with the color-based particle filter. For debugging purposes we can also draw the entire particle set by calling the same function on each particle p state `particle.getParticles(p)`.

7.2.6 Tracking Multiple Targets

Now we extend the tracker of step 5 to handle multiple, noninteracting targets. This extension is straightforward in OpenTL. As in step 3, for each new target the user selects a rectangular selection to obtain the shape and appearance model. This selection can be done online while the previous targets are being tracked.

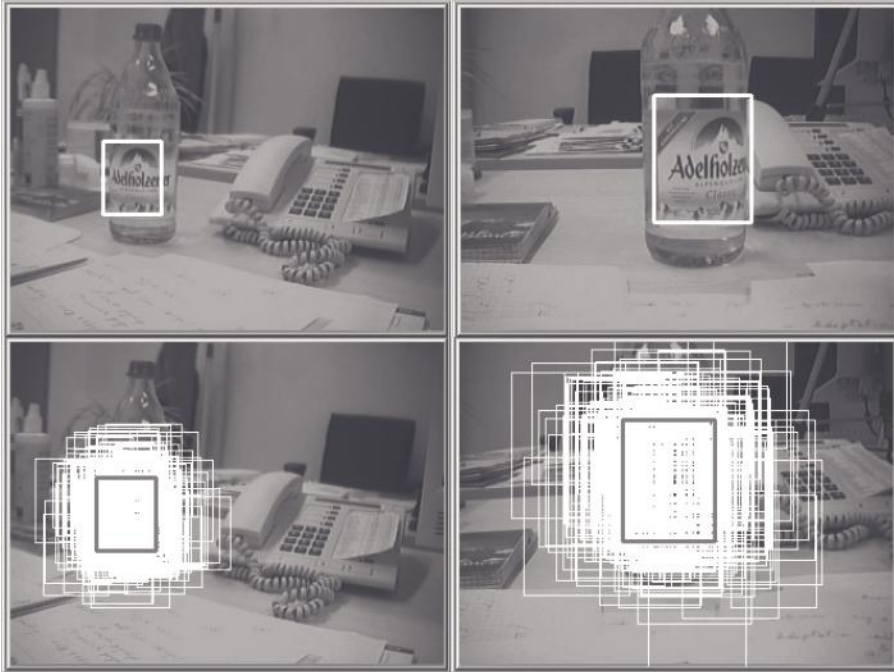


Figure 7.7 Tutorial 5: Color-based particle filter. *Top row:* The object with the shape and appearance model defined in the previous steps is tracked online from different viewpoints. *Bottom row:* The same frames with particles superimposed, representing the estimated posterior distribution (the thick rectangle represents the *weighted average* of the particle set).

For each selection a new object model, with a new shape and appearance and the same motion and pose parameters, is created. Then a single target is instantiated from this model by using the `TargetFactory` class, and the particle filter initializes its distribution by calling the `init()` function with the new target only (put into a temporary vector). Afterward, multiple objects in the scene can be selected and tracked with a single call to the `predict()` and `correct()` methods by giving the complete target vector.

Figure 7.8 shows some example results. In this filter, targets are noninteracting because their dynamics is independent and can also be of a different type, and their measurement data are collected independently without taking mutual occlusions into account. The latter makes sense for a two-dimensional pose, where depth information is not available for computing occlusions, although one could use multihypothesis occlusion handling, as described in Section 3.3.4.

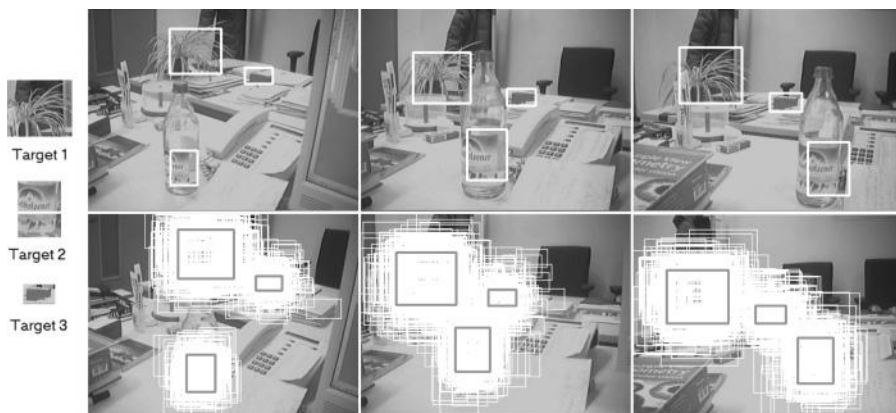


Figure 7.8 Tutorial 6: Tracking multiple targets using color statistics and particle filtering. *Left column*: the three object models; *upper row*: result of the tracking in different frames of a video sequence; *lower row*: respective hypothesis of the particle filter.

7.2.7 Multimodal Measurement Fusion

Another extension of step 5 consists of adding a second modality to the processing tree, to obtain dynamic data fusion. Here we use the `IntensityEdges` modality, described in Section 4.4.1, which for feature-level matching is implemented in the following way:

1. `IntensityEdges::preProcess`. Perform edge detection of the image. We will use a combination of the *Sobel* filter and the *Canny* edge map, so that edges are localized with *both* position and orientation information.
2. `IntensityEdges::sampleModelFeatures`. Sample uniformly distributed contour points from the visible model contours, retaining their object coordinates for subsequent re-projection. This is executed only once per frame, at the average pose predicted.
3. `IntensityEdges::matchFeatureLevel`. Re-project and match model points to the preprocessed edges by searching along the normals of the respective edges, up to a distance given by the *validation gate*. Here, we can also choose whether to restrict to search to the nearest neighbor only, yielding a single association hypothesis, or to keep multiple hypotheses within the validation gate and use a non-Gaussian likelihood, for more robustness.

In OpenTL, contour sampling is done on the GPU and implemented into the lower-level class `opentl::modelprojection::ContourSampler`, which may be shared by different contour-based modalities such as CCD.

This class requires an OpenGL representation of the object model, provided by the class `GLScene` of the `modelprojection` name space, responsible for handling the virtual OpenGL representation of the real scene, for both GPU programming and visualization purposes. The `GLScene` class is instantiated by

```
GLScene scene(objectModels);
```

with its own parameters; here we only need the camera resolution:

```
construct_initWindowSizeX/Y
```

To add this modality to the processing tree, we can instantiate it by

```
IntensityEdges * pContourPoints =  
new ContourPointsGPU(objectModels, warp, &scene, camIdx);
```

and afterward, as for the *ColourHistogram* modality, set the specific parameters for feature-level matching:

- `double matchFeatLevel_fixedCov`: value for the fixed monodimensional measurement variance along each normal (in pixels)
- `FeatDetectFilterType preProcess_featureFilterType`: CANNY, SOBEL, or CANNY_AND_SOBEL
- `double matchF_angleThreshold`: maximum angle between the edge for matching expected and that observed only possible using CANNY_AND_SOBEL
- `double matchF_missingAssocRate`: rate of missing detections expected (i.e., unassociated contour points)
- `bool matchF_robustFlag`: whether to compute the standard LSE residual or a robust version, corresponding to an M-estimator
- `std::vector<unsigned int> matchF_edgeSearchlengthAlongNormalinPixel`: for each link, the size of validation gate in pixels
- `bool matchF_nearestNeighbor`: whether to use only the nearest-neighbor edges or to collect multiple hypotheses
- `bool matchF_enableDebugOutput`: enabling the generation of debug output images, which also slows down the computation
- `bool generate_visCheck`: whether to generate the internal contour sampler at construction time or to pass it from outside

Like for the *ColourHist2D* modality, we finalize the initialization of the modality by calling

```
pContourPoints->init();
```

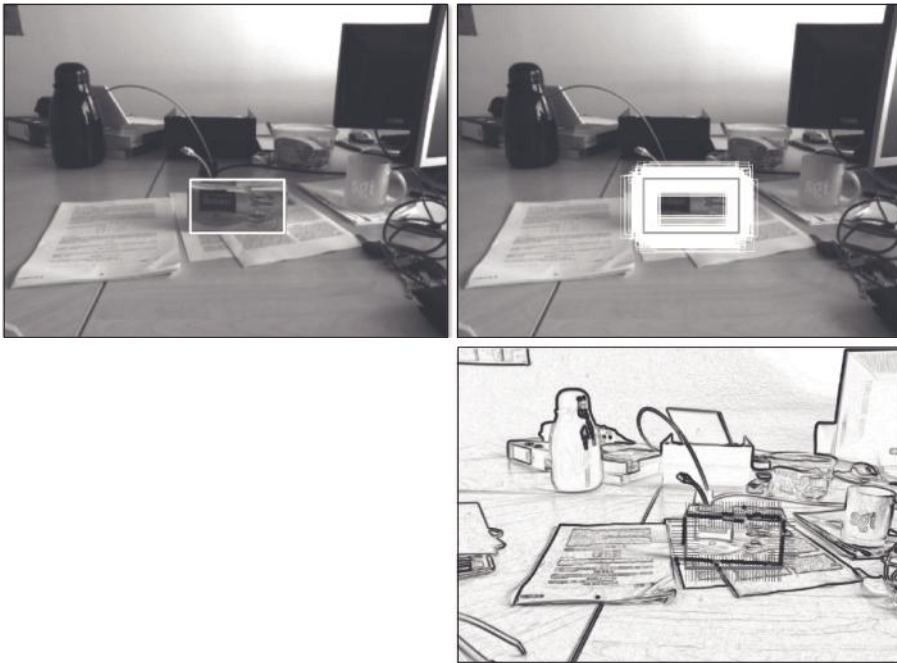


Figure 7.9 Tutorial 7: Tracking a single target using dynamic fusion of color statistics and intensity edges, with particle filtering. *Upper row*: estimated output state on the left side and all particles on the right side; *bottom row*: appearance model selected and the edge map with matched points along the normals for one particle hypothesis.

and then we can connect it to the likelihood class by

```
likelihood. addChild(pContourPoints,
                    core::cvdata::FEATURE-LEVEL);
```

Internally, the joint likelihood is computed as a product of the individual likelihoods, resulting in a dynamic fusion. Here we track only a single target, as in step 5. Figure 7.9 shows the output of this application.

The advantage of this type of fusion is that a color-based modality may fail to detect the scale of the object, since any distribution that contains the right amount of color matches the model, so the shape could shrink indefinitely. On the other hand, edges locate both scale and translation with precision but may fail in the presence of a cluttered background. The product of these likelihoods has, instead, a well-defined peak on the right color region at the correct scale.

7.3 OTHER APPLICATION EXAMPLES

Following our framework, we are able to formulate most state-of-the-art algorithms and systems, as well as novel ones, in terms of a self-contained description and parameter specification. In this section, we present several applications of different complexity implemented with OpenTL.

As a first case, consider the color-based particle filter [140] described in the tutorial step 5 (Section 7.2.5) using a single reference image as the appearance model, and a simple rectangular shape with planar translation and scale. The full specification is summarized in Fig. 7.10. In Fig. 7.11 we can see the main processing blocks and a face-tracking example.

<p><i>Task description</i> Planar object tracking with color histograms using a SIR particle filter and a feature-level likelihood</p> <p><i>Prior models</i></p> <ul style="list-style-type: none"> • <i>Shape and appearance</i>: rectangular model, with a key frame containing the object of interest (without texture map) • <i>Degrees of freedom</i>: two-dimensional translation and uniform scale (3 dof) $\mathbf{y} = \mathbf{a}\mathbf{x} + \mathbf{t}$ • <i>Dynamics</i>: Brownian motion in the pose parameters $s = [a; t_x, t_y]$ • <i>Camera model</i>: single camera, uncalibrated • <i>Visual modalities</i>: color histograms • <i>Absolute prior</i>: initialized target with a large, uniform distribution $P(s_0)$ around the central pose <p><i>Tracking pipeline</i></p> <ol style="list-style-type: none"> 1. <i>Bayesian prediction</i>. Resample and drift the old particle set according to the dynamics. 2. <i>Preprocessing</i>. Carry out color conversion to the HSV space. 3. <i>Features sampling</i>. Off-line; convert the reference image to HSV and a sample color histogram (h). 4. <i>Matching (feature level)</i>. For each hypothesis $s_{k,n}$, project the shape and compute the histogram (z), and the residual $e = B(h, z)$, given by the Bhattacharyya distance, with R^2 the measurement covariance. 5. <i>Likelihood</i>. This is a Gaussian in the residual value $L \propto \exp(-e^2/2R^2)$. 6. <i>Bayesian update</i>. Update weights $w_{k,n}$ for all particles with the likelihood values, and compute the average state \bar{s}_k. 7. <i>Features update</i>. Update online reference histograms from image data at pose \bar{s}_k. 8. <i>Loss detection</i>. Compute the sample covariance \bar{P} and check against a threshold value $\det(\bar{P}) < d$; in case of loss, reinitialize the filter.
--

Figure 7.10 Specification for a complete color-based tracker. (From [140].)

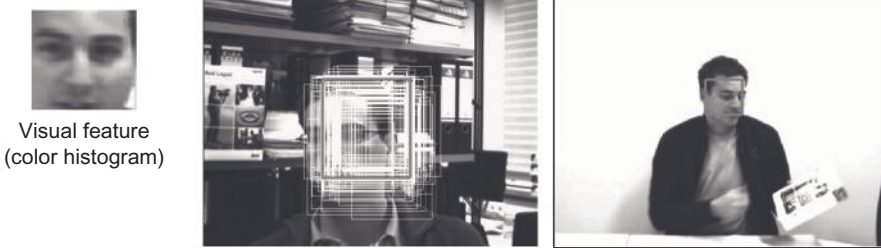
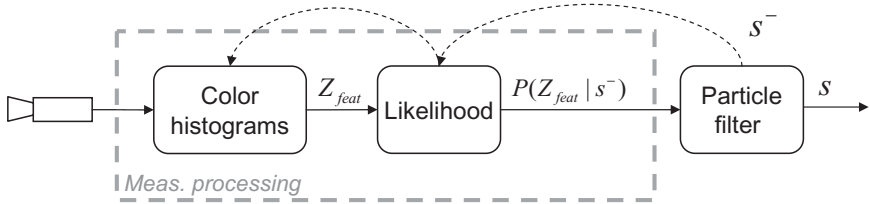


Figure 7.11 Color-based particle filter.

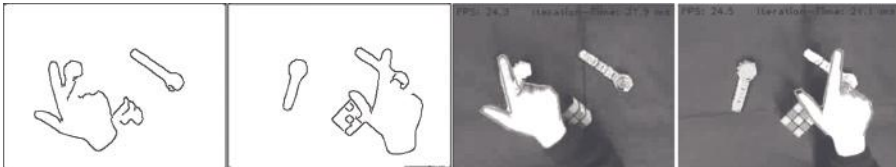
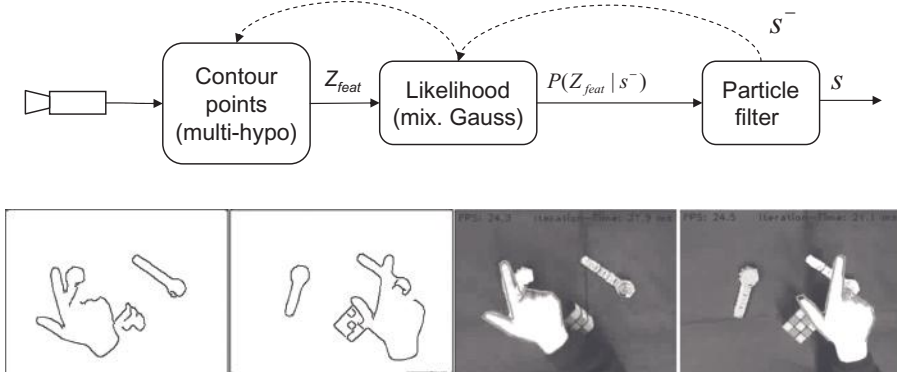


Figure 7.12 Contour-based particle filter for planar hand tracking.

In Fig. 7.12 we show the processing blocks for a contour-based particle filter [84] with multihypotheses data association, applied to planar hand tracking. In this application, a set of 200 particles has been used; in particular, contour sampling is done off-line, since the shape is planar and the transformation is a similarity, so that its visibility conditions do not change with the pose.

Next, suppose that we want to track an object by statically combining information from color and foreground segmentation at the pixel level using a simple *AND* operation, equivalent to multiple binary likelihoods, and afterward detect the nearest component (blob) connected to the position predicted, finally computing the implicit measurement model of Section 3.3 at the feature level. The top row of Fig. 7.13 shows the resulting scheme.

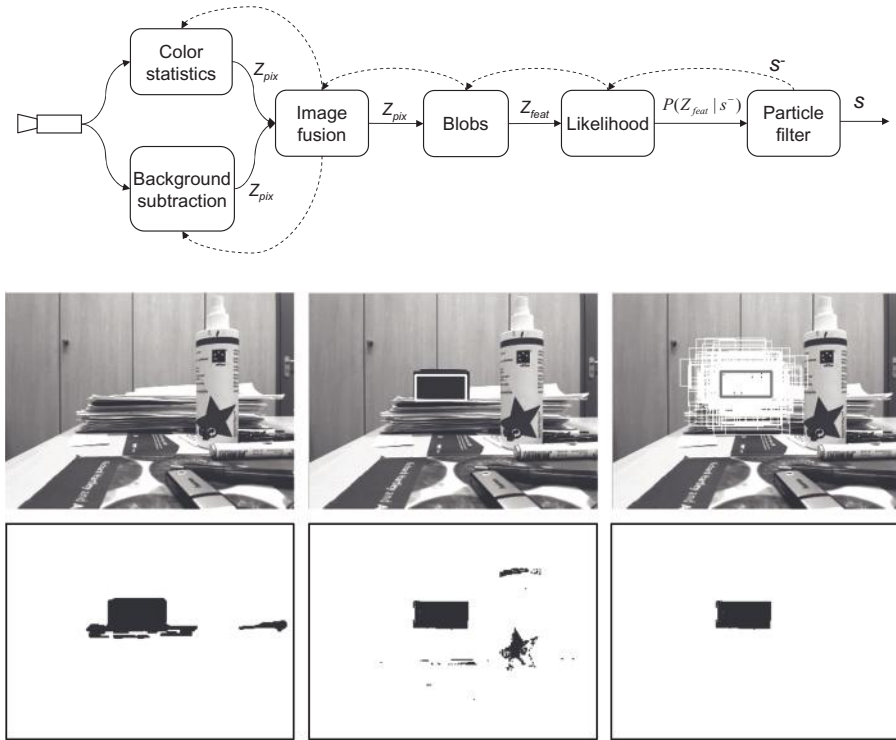


Figure 7.13 Integrating background and object color models, with pixel-level fusion.

In particular, here we have three visual modalities: color statistics and foreground segmentation, followed by static pixel-level fusion, then blob detection and matching. We notice here that both *parallel* and *cascade* connections are present. In this scheme, color and motion segmentation provide binary images in the z field of the respective Z_{pix} , but no expected images h or residuals e are needed, since the images are used only for the subsequent blob detection.

Instead, the blob modality will compute the expected blob for each state hypothesis and match it to the closest one on the fused image, with a single association hypothesis, thus providing measurement residuals and covariances into Z_{feat} for computing the likelihood. As we have seen in the tutorial, OpenTL provides for this purpose a set of flags, indicating which fields of the measurement Z are required. These flags have to be specified while building the processing tree. Furthermore, some of these data need to be computed only once per frame: for example, blob detection z_{feat} or the segmented images before fusion z_{pix} . Also, this information has to be specified in advance.

In this pipeline, we also note that preprocessing needs to be called only for the lower-level modalities, whereas the blob modality does no preprocessing,

since blob detection depends on the result of image fusion, which is model dependent in a multitarget scenario.

Some results are shown on the bottom of Fig. 7.13 for a single planar object. In particular, from left to right in the middle row we can see the background model, the object model, and blob detection and tracking on the fused image. The latter, as shown in the bottom row, is obtained through foreground segmentation with the method [69] (left), color segmentation by histogram back-projection [45] (middle), and pixel-level fusion via the *AND* operator (right). Pose parameters are again two-dimensional (t_x, t_y, s) with uniform scale and Brownian dynamics.

Extending this scheme to multiple, calibrated cameras is straightforward, with dynamic fusion of the respective blobs, performed by multiplying the two likelihoods. Such a scheme makes it possible to better localize three-dimensional objects by matching blobs across different views. An alternative version of this filter using *pixel-level* matching [100] and GMM color statistics is shown in Fig. 7.14. Here, color segmentations expected and observed are computed and compared directly on the GPU, providing global image residuals for the likelihood function. Due to the generic formulation, it is suitable for any type of object, including three-dimensional shapes, for which the projected shadow provides enough information to localize their pose, and can easily be extended to a multicamera setup.

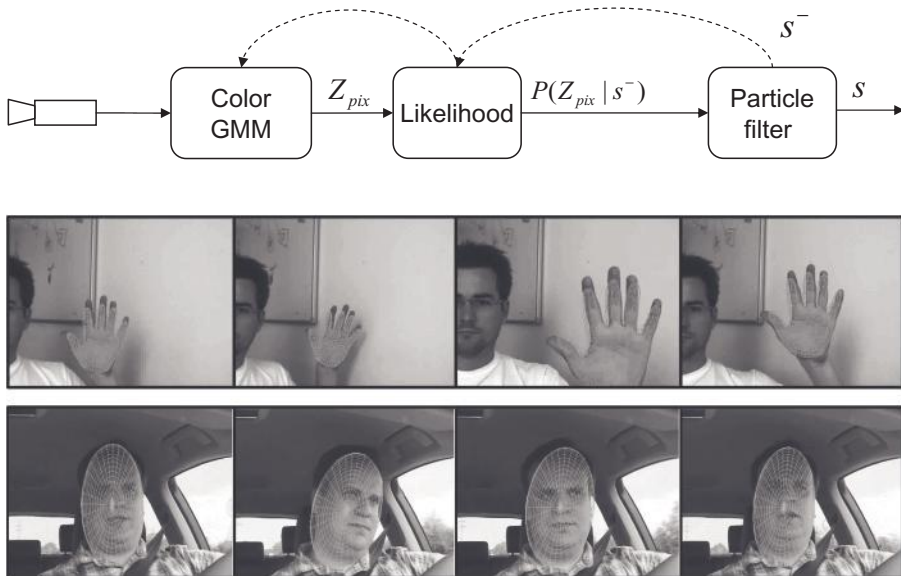


Figure 7.14 Hand and face tracking sequences using different mesh-models and a GPU-accelerated foreground segmentation and particle filtering. (From [100].)

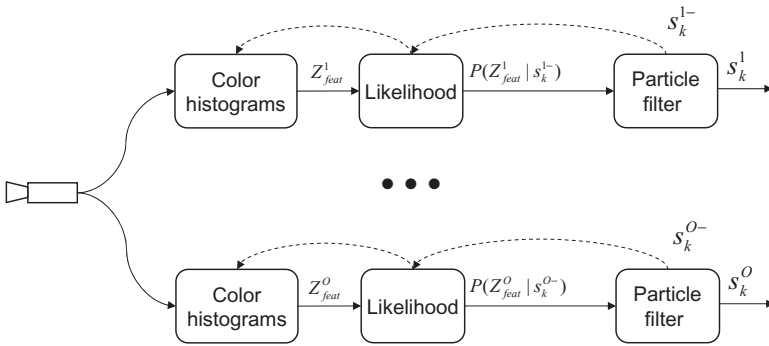


Figure 7.15 Multitarget tracking with independent color-based particle filters. Target models are show on the left. Results can be seen in Fig. 7.8.

Moving to the multitarget case, Fig. 7.15 shows a scheme for color-based tracking without occlusion handling, as described in the tutorial step of Section 7.2.6. Here, the filter of Fig. 7.11 is instantiated with three targets of a different shape and appearance model. In this case particle filters are independent of one another, so no *simultaneity* of state hypotheses is considered (independent particle drift and resampling), and the likelihoods are kept separate for each target.

On the other hand, Fig. 7.16 shows an interacting multitarget tracker using an MCMC particle filter [92,130], with the same color histogram modality. Unlike the preceding example, this system also handles target interactions at both the level of dynamics, by enforcing a penalty term for overlapping targets, and of measurement likelihoods, by considering mutual occlusions.

This is possible because this Bayesian filter makes ensemble-state hypotheses during prediction and correction (Section 5.4.4).

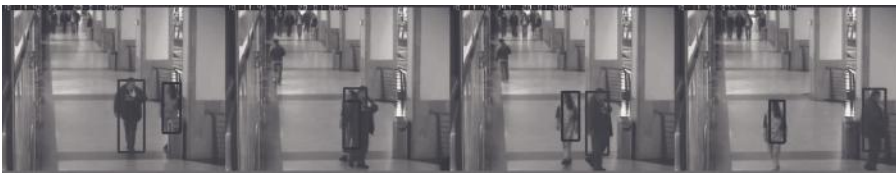
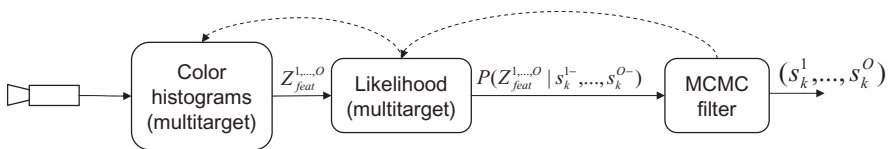


Figure 7.16 Tracking interacting targets using the MCMC algorithm, with the same color-based likelihood.

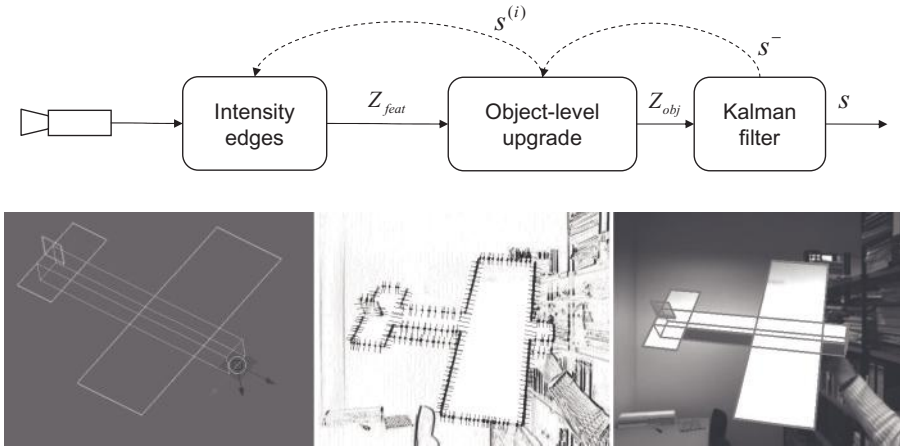


Figure 7.17 Object tracking with edge-based least-squares pose estimation.

Going to more complex shapes, in Fig. 7.17 we show an LSE edge-based tracker using model contour points obtained through the GPU-assisted algorithm of Section 3.2. This system corresponds to the well-known estimator described in [61,76], based on local least-squares optimization of re-projected contour points. An *object-level* measurement, $z_k = \mathbf{p}_k^*$, is performed by maximizing the single-hypothesis contour likelihood starting from the pose hypothesis \mathbf{p}_k^- predicted. For this purpose, at each hypothesis $\mathbf{p}^{(i)}$ only nearest edges along the normals are matched, in order to evaluate the overall error and its derivatives. Pose parameters are Euclidean with 6dof and compositional updates through Lie algebras for the estimation process; on top of it, a Kalman filter provides Bayesian tracking in state space.

Figure 7.18 shows an example of an articulated model used for contour-based planar hand tracking [132]. This example combines CCD and intensity edges in a static data fusion scheme consisting of joint LSE estimation [which we call an *object-level upgrade*, eq. (3.17)], with two-dimensional articulated pose parameters. In particular, a total of 9 dof is given by a skeleton composed of five links: the palm with 4-dof similarity, and five fingers each with a rotational joint. Model contour points, common to both modalities, are sampled and re-projected with respect to each link. The maximum-likelihood pose estimate measurement is fed to a Kalman filter. This scheme provides robustness for situations such as the one depicted in Fig. 3.13, where the integration is more robust than each modality alone.

We can show an example of hierarchical data fusion in Fig. 7.19, also described by Panin and Knoll [133]. This example shows a complementary data fusion where two visual modalities, contour and template matching, provide two subsets of the full pose space: respectively, three-dimensional translation and rotation. In particular, the head contour is first estimated through the

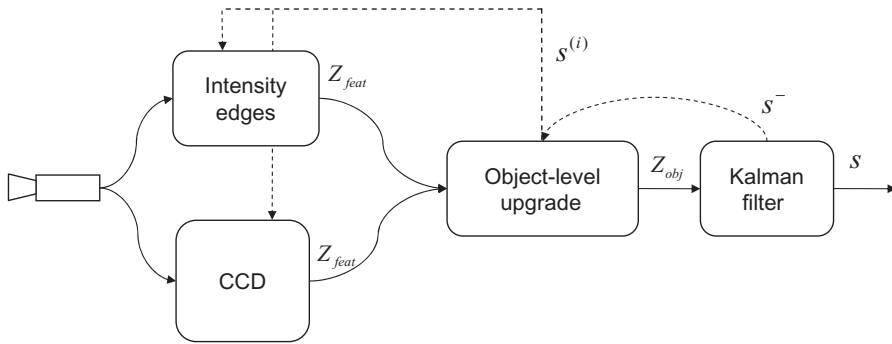


Figure 7.18 Monocular, articulated hand tracking with contour points, combining CCD and intensity edges. Sampled contour points and normals are shown. (From [132].)

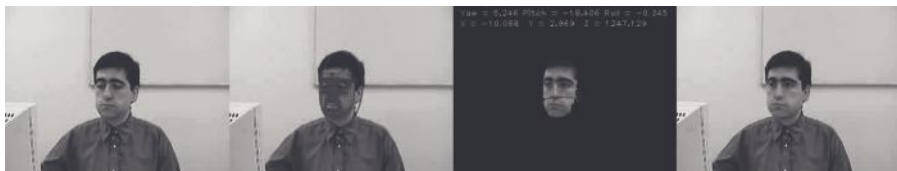
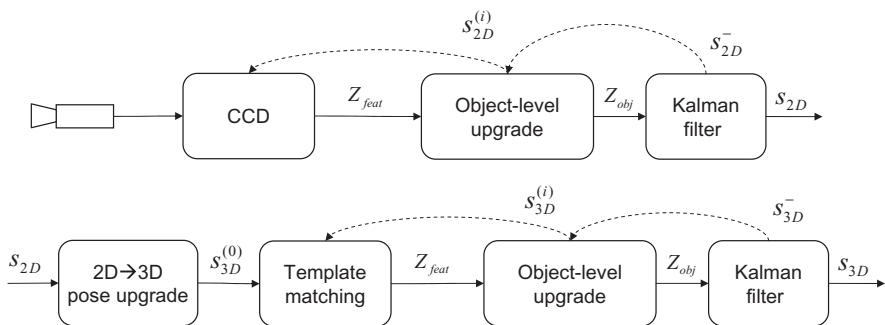


Figure 7.19 Three-dimensional face tracking, integrating contour and template matching in cascade. (From [133].)



Figure 7.20 People tracking with a multipatch color histogram model. *Top row*: frontal camera frames; *bottom row*: overhead camera. (From [126,127]. Copyright © 2008 IEEE.)

CCD algorithm using a simple elliptical model; subsequently, the two-dimensional parameters s_{2D} are upgraded to provide an initial estimate of three-dimensional position $s_{3D}^{(0)}$ for estimating the head rotation based on template matching, using mutual information.

In this scheme we have a cascade of two pipelines, with different object models and different pose parameters. In between, an LSE pose can be upgraded if the number of corresponding points between the three- and two-dimensional shape is given. Both pipelines are closed by respective Kalman filters (object level).

Next, we consider applications involving multiple cameras. Figure 7.20 shows a complementary setup involving person tracking for virtual-reality TV studios, where the two orthogonal views are attached to independent tracking pipelines and their two-dimensional estimation results are fused later for the purpose of image-based visual servoing.

Both pipelines track using the usual color-based particle filter. However, here the model consists of multiple *patches* (rectangular regions) where different color histograms are collected and independently matched to the image data. This results in a more robust localization, in the presence of different statistics: for example, of head, upper body, and legs.

In OpenTL, a multipatch model is a special case of an articulated body but without any degrees of freedom between the links. Only the root of the skeleton moves and scales, which results in a single scale and translation of all parts. A robot-cameraman application using this system has been presented [126,127].

An example of redundant multicamera data fusion is given by the tree shown in Fig. 7.21. Here we connect the two contour-based modalities to both cameras, and evaluate a joint LSE pose out of the four feature-level measurements. Figure 7.22 shows this system integrated in a control loop for stabilizing the flight of a *quadrotor* device, with visual position and

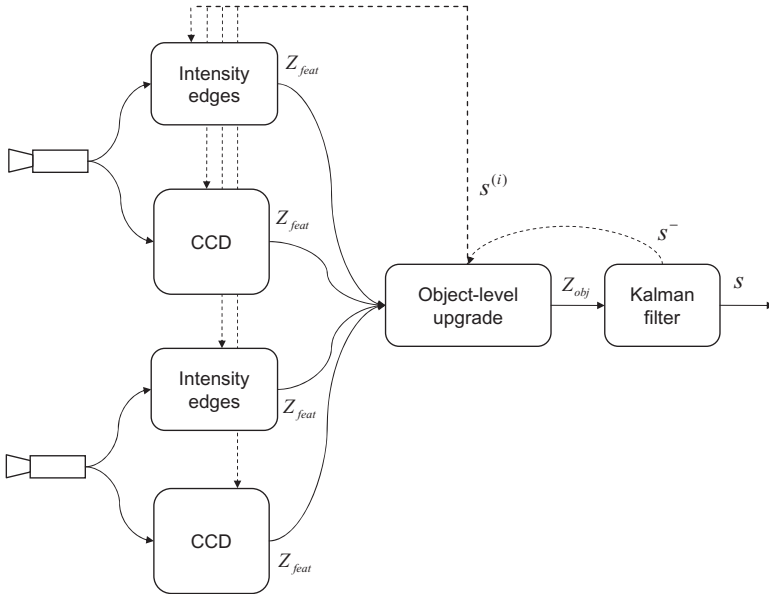


Figure 7.21 Stereo camera setup integrating two contour-based modalities.



Figure 7.22 Tracking a quadrotor device using the scheme of Fig. 7.21.

orientation feedback. This model is a complex mesh with self-occlusions and concavities without texture. The picture shows how the robot is able to maintain and recover its “home” position after perturbation.

Next we consider a final application of this scheme, involving three cameras for tracking the full articulated model of a human hand. In this setup, three cameras are located as shown in Fig. 7.23, and calibrated with respect to a common world frame. The hand model is a three-dimensional skeleton, with 26dof and 16 links (Fig. 7.24). The shape of each link is built from a simple model with seven structural parameters given by the overall length, the two

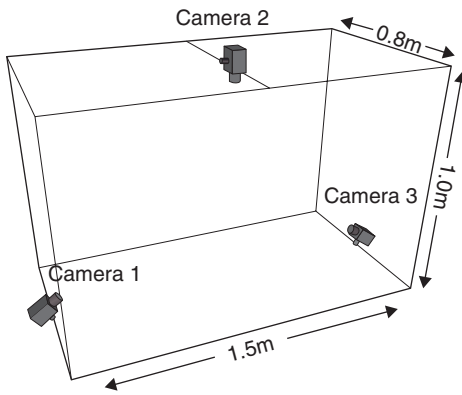
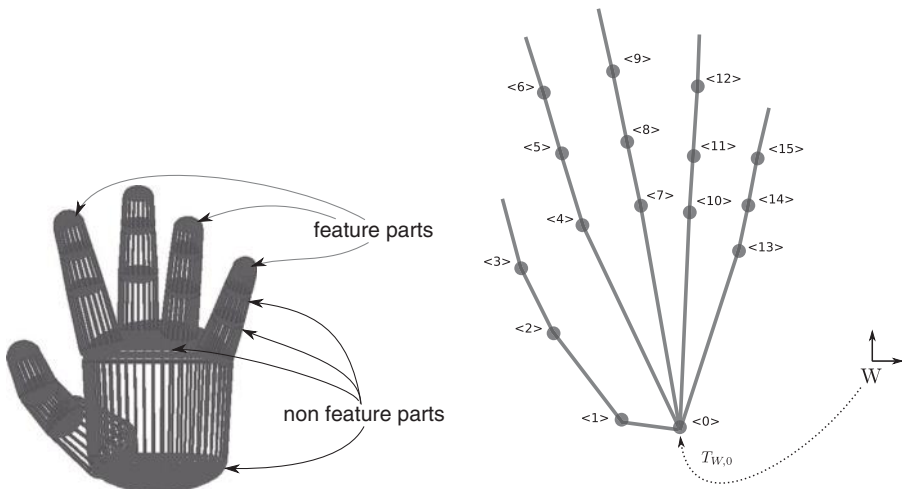


Figure 7.23 Multicamera setup for three-dimensional object tracking.



(a) Shape model, with marked non-feature parts.

(b) Skeleton of the shape model.

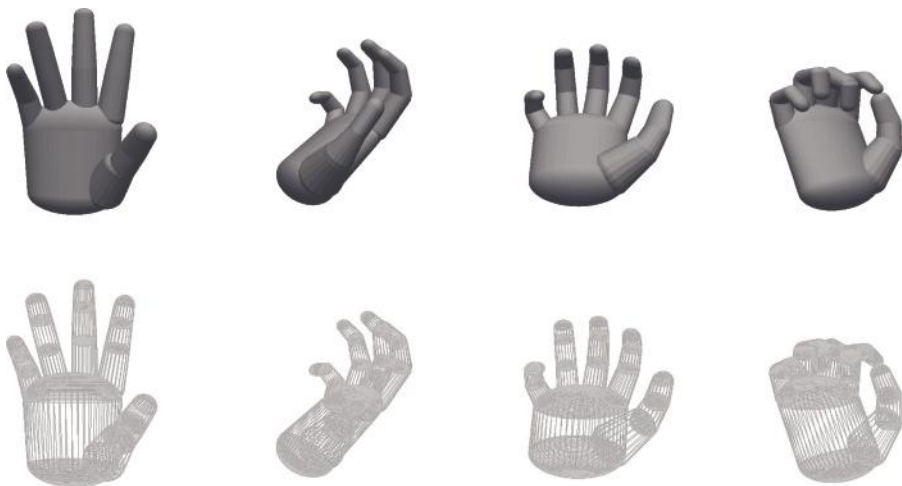


Figure 7.24 *Top*: Articulated hand model; *bottom*: rendered views of three-dimensional hand model.

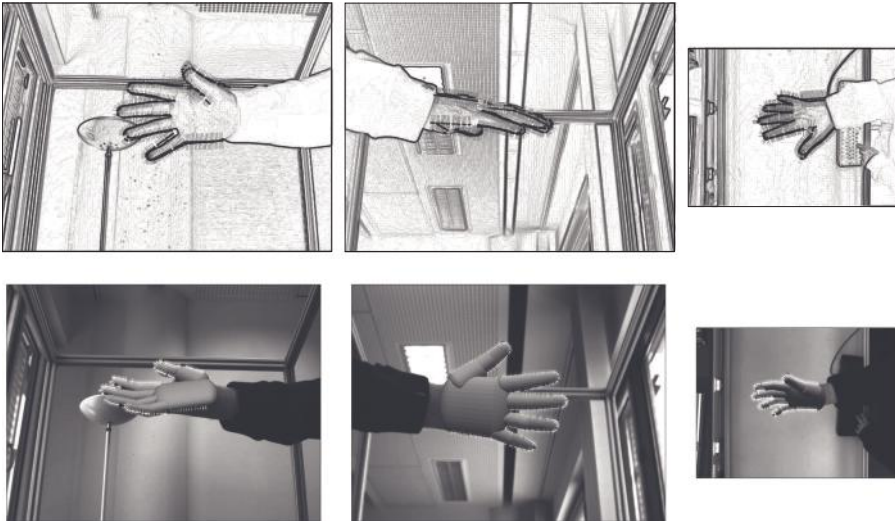


Figure 7.25 *Top: edge detection and model matching on the three camera views; bottom: results of three-dimensional hand tracking.*

axes of the elliptical section at each extreme, and the length of the two terminating caps. These parameters, along with the reference matrices \bar{T} defined in Section 2.2.1, are constant and determined off-line by a *model adaptation* procedure, while the 26 motion parameters consist of the palm Euclidean pose of the palm plus 4 rotational dof for each finger, modeling the *proximal* and *distal joints*.

One important issue is that some parts of the model are never used for tracking, although they could be visible from a given viewpoint, such as closure of the palm on the wrist. These parts of the mesh (polygons and edges) are marked in advance as *nonfeatures*, meaning that no visible features such as contour points can be sampled on them. We notice that unlike the two-dimensional scheme of Fig. 7.18, here we need to resample contour points at each view.

An example of edge detection and matching is shown at the top of Fig. 7.25; some tracking results are shown in the bottom rows.

APPENDIX A

POSE ESTIMATION

As we have seen in Section 2.2.1, pose parameters define the motion and deformation of visible surfaces with respect to the *world frame* W , and may range from the simplest single-body transforms up to articulated and deformable shapes such as a human hand or face. In particular, model-based tracking concentrates on the task of estimating the pose parameters from a set of corresponding *features*, observed in the image data and matched to the model counterpart. This problem is complex in that it may involve different types of geometric primitives, such as points, lines, curves, planes, and quadrics, for any of the representations mentioned above. In this and the following appendix, we restrict our attention to methods for single-body pose estimation using points and lines, which are the most common image features, and provide different solutions involving affine or perspective cameras.

A.1 POINT CORRESPONDENCES

Let us first consider object points in three- or two-dimensional space, \mathbf{x}_i , and their projections on the image, \mathbf{y}_i (Fig. A.1). In a maximum-likelihood sense, the goal is to estimate the world transform matrix T from N correspondences between exact model primitives and their noisy observation in the two-dimensional image, supposed to have Gaussian noise statistics with uniform covariance (also called *homoscedastic* measurements). Its solution therefore

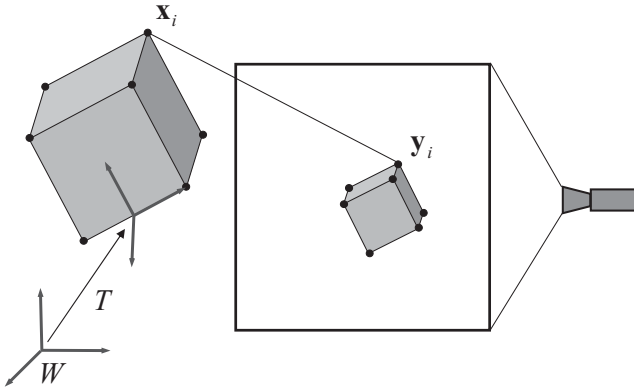


Figure A.1 Pose estimation from point correspondences.

consists of minimizing an equally weighted least-squares error in terms of the image *residuals*.

The object pose is represented locally by a 3×3 or 4×4 homogeneous transform T per link, belonging to a given *submanifold* \mathcal{T} of all homographies, the latter also called $SL(3), SL(4)$ (for *special linear* group), where the undetermined scale factor is removed by imposing $\det(T)=1$. Moreover, a reference (constant) interlink transformation \bar{T} may be present, which premultiplies the local T : for example, representing fixed displacement of Cartesian frames in articulated skeletons not necessarily belonging to the same manifold of \mathcal{T} . The camera projection matrix, $P_{c,w}$ (3×4), defined in Section 2.1, is also supposed to be known. Since both are constant terms, we can incorporate them into a single projection matrix, $P \equiv P_{c,w} \cdot \bar{T}$. Overall, the estimation problem can be formulated in two ways, using either homogeneous or nonhomogeneous coordinates. The two formulations raise two different least-squares errors to be optimized: algebraic and geometric error, respectively.

Geometric errors, also called *re-projection errors*, correspond directly to Euclidean distances in image space. Therefore, minimizing geometric errors yields the maximum-likelihood estimate in the case of independent Gaussian noise. However, the cost function is often nonlinear and requires a reasonably good initial guess, in order to converge to the global minimum.

Algebraic errors are formulated using homogeneous equations and provide algebraic residuals that usually do not possess an intuitive geometric interpretation. Therefore, minimizing an algebraic error may not correspond to a maximum-likelihood estimate. On the other hand, the algebraic LSE is often linear and can be solved in one step [e.g., via the *singular value decomposition* (SVD)], providing an initial value for the geometric error optimization.

A.1.1 Geometric Error

For a single link, consider the following problem: Given N exact model points¹ $\mathbf{x}_i = (x_i^1, x_i^2, x_i^3, 1)^T$ and corresponding noisy measurements $\mathbf{y}_i = (y_i^1, y_i^2)$, find the optimal transformation T_{geo}^* , belonging to \mathcal{T} , such that

$$T_{\text{geo}}^* = \underset{T \in \mathcal{T}}{\operatorname{argmin}} \sum_{i=1}^N \|\pi(P \cdot T \cdot \mathbf{x}_i) - \mathbf{y}_i\|^2 \quad (\text{A.1})$$

where P is the world-to-camera projection matrix, and π (defined in Section 2.1) operates the conversion from homogeneous to nonhomogeneous coordinates:

$$\pi(z^1, z^2, z^3) = [z^1/z^3 \quad z^2/z^3]^T \quad (\text{A.2})$$

The solution can be obtained by nonlinear LSE optimization, starting from an initial guess $T_0 \in \mathcal{T}$, close enough to T_{geo}^* to ensure convergence.

In particular, we note that (A.1) is a *constrained* optimization problem that can be cast as an unconstrained problem if we define a suitable *parametrization* of the manifold $T(\mathbf{p})$, where \mathbf{p} is a vector of pose parameters, with the dimension of the manifold.

As we have seen in Section 2.2.1, most of the the transformations possess a *Lie group* structure, and the local tangent space at T_0 is a *Lie algebra*, covering the entire manifold through the *exponential mapping*

$$T = T_0 \exp \sum_d G_d \delta p_d, \quad \forall T \in \mathcal{T} \quad (\text{A.3})$$

with G_d the Lie algebra generators and $\delta \mathbf{p}$ the local, incremental pose parameters. In this case, Gauss–Newton optimization is implemented straightforwardly by computing the singularity-free Jacobians in $\delta \mathbf{p} = 0$ and carrying out the compositional update defined above.

A.1.2 Algebraic Error

Using homogeneous coordinates, in the absence of noise we look for T^* in \mathcal{T} satisfying the condition

$$T^* \in \mathcal{T} : (P \cdot T^* \cdot \mathbf{x}_i) \propto \mathbf{y}_i, \quad \forall i \quad (\text{A.4})$$

¹In the following we do not distinguish explicitly between homogeneous and standard coordinates, letting the same symbol \mathbf{x} represent both whenever this does not lead to confusion.

As we can see, the homogeneous formulation carries the projective ambiguity as a scale factor, accounting for the augmented number of coordinates. We can cast these conditions into equalities by writing (A.4) as a cross-product,

$$\mathbf{y}_i \times (P \cdot T^* \cdot \mathbf{x}_i) = 0 \quad (\text{A.5})$$

which is a redundant set of homogeneous equations in T . This means that one component of each cross product (e.g., the third) can be discarded to keep $2N$ nonredundant equations.

For noisy data, (A.5) can be cast into an LSE:

$$T_{\text{alg}}^* = \underset{T \in \mathcal{T}}{\operatorname{argmin}} \sum_{i=1}^N \|\mathbf{y}_i \times (P \cdot T \cdot \mathbf{x}_i)\|^2 \quad (\text{A.6})$$

and if T is parametrized in a linear way, the entire problem is linear: $\operatorname{argmin}_{\mathbf{p}} \|A\mathbf{p}\|$ or $\operatorname{argmin}_{\mathbf{p}} \|A\mathbf{p} - \mathbf{b}\|$, where A is a column-rank-deficient matrix. This problem has ∞^1 solutions in \mathbf{p} and can be solved by imposing an additional constraint, such as a unit-norm condition $\|\mathbf{p}\| = 1$. The globally optimal solution is found in one step, via the SVD algorithm, and the resulting method is called the *direct linear transform* (DLT) [77].

Finally, maximum-likelihood pose estimation can be achieved in two steps:

1. Estimate the T_{alg}^* matrix, reasonably close to T_{geo}^* , by minimizing the algebraic error (DLT).
2. Starting from T_{alg}^* , minimize the geometric error with nonlinear LSE in order to obtain T_{geo}^* . For this purpose, whenever possible we use compositional updates δT and Lie algebra derivatives.

In some cases of two-dimensional/two-dimensional (2D-2D) and three-dimensional/three-dimensional (3D-3D) problems, the geometric error is also linear and can be solved at once, without need for T_{alg}^* . Also, some nonlinear problems (e.g., the *absolute orientation*) may still be solved in one step. However, when a 3D-2D perspective projection P is involved, we have an inherent nonlinearity due to $\pi(\cdot)$, and therefore the two-step procedure cannot be avoided. We also notice that although step 2 can be implemented in a unified way through Lie algebras, step 1 must be solved differently for each specific manifold parametrization (see also Appendix B).

A.1.3 2D-2D and 3D-3D Transforms

If the model and measurement space have the same dimension n , \mathbf{x} and \mathbf{y} have the same size, and the projection matrix P reduces to a $n \times n$ *homography*. Therefore, all image data \mathbf{y}_i can be preprocessed to remove P :

$$\bar{\mathbf{y}}_i = P^{-1} \mathbf{y}_i \quad (\text{A.7})$$

and the algebraic LSE is reformulated as

$$T_{\text{alg}}^* = \underset{T \in \mathcal{T}}{\operatorname{argmin}} \sum_{i=1}^N \|\bar{\mathbf{y}}_i \times (T \cdot \mathbf{x}_i)\|^2 \quad (\text{A.8})$$

which can be solved in one step for a linear parametrization $T(\mathbf{p})$. Moreover, if the last row of T , as well as P , is $[0, \dots, 0, 1]$, which is true in many cases, the geometric error reduces to

$$T^* = \underset{T \in \mathcal{T}}{\operatorname{argmin}} \sum_{i=1}^N \|T \cdot \mathbf{x}_i - \bar{\mathbf{y}}_i\|^2 \quad (\text{A.9})$$

which is also linear, for a linear parametrization. Appendix B shows several cases of poses that admit a linear parametrization. Here we just mention the two-dimensional affine transform

$$T = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (\text{A.10})$$

parametrized by

$$\mathbf{p} = [A_{11} \ A_{12} \ A_{21} \ A_{22} \ t_1 \ t_2]^T \quad (\text{A.11})$$

so that

$$T\mathbf{x} = \begin{bmatrix} x^1 & x^2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x^1 & x^2 & 0 & 1 \end{bmatrix} \mathbf{p} \equiv \mathcal{X}\mathbf{p} \quad (\text{A.12})$$

where the matrix \mathcal{X} is defined in terms of \mathbf{x} . A similar result can be obtained for the two-dimensional similarity (uniform scale, rotation, and translation) with four parameters, parametrized by

$$T = \begin{bmatrix} c & -s & t_1 \\ s & c & t_2 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.13})$$

with

$$\mathbf{p} = [c \ s \ t_1 \ t_2]^T \quad (\text{A.14})$$

so that

$$\mathcal{X} = \begin{bmatrix} x^1 & -x^2 & 1 & 0 \\ x^2 & x^1 & 0 & 1 \end{bmatrix} \quad (\text{A.15})$$

However, the Euclidean group where $c = \cos \theta$, $s = \sin \theta$, involves a nonlinearity. In this case we can first estimate the pose as a similarity, and afterward remove the scale factor simply by dividing (c, s) by $\sqrt{c^2 + s^2}$. We notice that this type of upgrade, also called parameter *clamping*, in general does not solve the geometric LSE; therefore, it should be followed by nonlinear optimization. A direct solution for the Euclidean group is given instead by the Umeyama approach, described in Appendix B.

A.1.4 DLT Approach to 3D-2D Projections

In the 3D-2D case, T is a 4×4 matrix and P is a 3×4 projection. In particular, we consider here *pinhole* cameras without distortion, so that $P_{c,w} = K \cdot T_{c,w}$, where

$$K = \begin{bmatrix} K' & \mathbf{0} \end{bmatrix} \quad (\text{A.16})$$

is the intrinsic projection matrix. If we restrict ourselves to transforms within the affine group, thus avoiding full three-dimensional homographies, we have

$$T = \begin{bmatrix} A_{3 \times 3} & \mathbf{t}_3 \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathcal{T} \quad (\text{A.17})$$

$$T_{c,w} = \begin{bmatrix} \bar{A}_{3 \times 3} & \bar{\mathbf{t}}_3 \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathcal{T}' \quad (\text{A.18})$$

belonging to possibly different groups $\mathcal{T}, \mathcal{T}'$, where \mathcal{T}' is usually Euclidean.

As already mentioned, because of the dimensionality loss from three to two dimensions, the nonlinear operator $\pi(\bullet)$ must be used no matter the group (A, \mathbf{t}) . Therefore, the initialization provided by the algebraic error (DLT) has to be used as an initial guess for the geometric error optimization. The algebraic estimation problem now is to find (A^*, \mathbf{t}^*) such that

$$[A^* \quad \mathbf{t}^*] \in \mathcal{T} : (K' [\bar{A} A^* \quad \bar{A} \mathbf{t}^* + \bar{\mathbf{t}}] \mathbf{x}_i) \propto \mathbf{y}_i, \quad \forall i \quad (\text{A.19})$$

We can again preprocess the image data \mathbf{y}_i :

$$\bar{\mathbf{y}}_i = (K')^{-1} \mathbf{y}_i \quad (\text{A.20})$$

and rewrite

$$[\bar{A}A^* \quad \bar{A}\mathbf{t}^* + \bar{\mathbf{t}}] \mathbf{x}_i \propto \bar{\mathbf{y}}_i \quad (\text{A.21})$$

However, here in general we cannot remove either \bar{A} or $\bar{\mathbf{t}}$, because the matrix on the left side is not square.²

In order to apply the DLT to any submanifold with less than the maximum number of parameters, in this case 12, we also assume that \mathcal{T} is parametrized or at least approximated by

$$[A(\mathbf{q}) \quad \mathbf{t}(\mathbf{q})] \quad (\text{A.22})$$

where $\dim(\mathbf{q}) = D \leq 12$ is obtained by employing the linear constraint $\mathbf{p} = Q\mathbf{q} + \mathbf{p}_0$, and \mathbf{p} represents the column-wise entries of $[A \quad \mathbf{t}]$.

This condition seems to be quite restrictive, since many transformation groups have only nonlinear parametrizations, particularly in the presence of rotations; nevertheless, we can always find a linearly parametrized manifold \mathcal{T}_q that includes \mathcal{T} , with $\dim(\mathcal{T}_q)$ higher than $\dim(\mathcal{T})$ but as close to it as possible.

Afterward, the estimated transform T_q^* can be upgraded to the correct manifold by means of a *Procrustes analysis*,

$$T_0 = \underset{T \in \mathcal{T}}{\operatorname{argmin}} \|T - T_q^*\|_F \quad (\text{A.23})$$

where F is, for example, the *Frobenius norm* (i.e., the element-wise sum of squared differences). As we have seen for the Euclidean case, the same upgrade may be made using simpler methods such as clamping the scale parameters by normalizing the matrix entries. In any case, the result of this procedure needs to be accurate enough as an initial value for a full geometric optimization.

We mention here a few examples of linear constraints that we can impose on the affine group (A, \mathbf{t}) . A pure translation in three dimensions is given by $A = I_3$, which corresponds to

$$Q = \begin{bmatrix} 0_{3 \times 3} \\ 0_{3 \times 3} \\ 0_{3 \times 3} \\ I_{3 \times 3} \end{bmatrix} \quad (\text{A.24})$$

$$\mathbf{p}_0 = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]^T \quad (\text{A.25})$$

so that

²The only exception is given by $\bar{\mathbf{t}} = \mathbf{0}$, in which case \bar{A} can also be removed from the data points $\bar{\mathbf{y}}_i = (K'\bar{A})^{-1} \mathbf{y}_i$ and the problem becomes a purely projective one: $[A^* \quad \mathbf{t}^*] \mathbf{x}_i \propto \bar{\mathbf{y}}_i$.

$$\mathbf{q} = [t_x \quad t_y \quad t_z]^T \quad (\text{A.26})$$

while a rotation around z , without the nonlinear constraint $c^2 + s^2 = 1$ (which can subsequently be enforced), is given by

$$A = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{t} = \mathbf{0}_3 \quad (\text{A.27})$$

which corresponds to

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix} \quad (\text{A.28})$$

$$\mathbf{p}_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]^T \quad (\text{A.29})$$

and therefore

$$\mathbf{q} = [c \quad s]^T \quad (\text{A.30})$$

and so on.

The DLT solution is now obtained by rewriting the algebraic error in terms of the reduced parameters \mathbf{q} :

$$\mathbf{q}^* = \operatorname{argmin}_{\mathbf{q} \in \mathbb{R}^D} \sum_{i=1}^N \|\mathbf{y}_i \times ([\bar{A}A(\mathbf{q}) \quad \bar{A}\mathbf{t}(\mathbf{q}) + \bar{\mathbf{t}}] \cdot \mathbf{x}_i)\|^2 \quad (\text{A.31})$$

and since A and \mathbf{t} are linear in \mathbf{q} , we show in Section A.4 that

$$\mathbf{q}^* = \operatorname{argmin}_{\mathbf{q} \in \mathbb{R}^D} \sum_{i=1}^N \|F_i \cdot \mathbf{q} - \mathbf{f}_i\|^2 \quad (\text{A.32})$$

with F_i and \mathbf{f}_i two data matrices, not dependent on \mathbf{q} . In what follows we call this method *generalized, projective DLT* (GP-DLT).

A.2 LINE CORRESPONDENCES

In some cases, image measurements consist of line *segments*—for example, those detected through the Hough transform (Section 4.4)—which can be matched to the corresponding model segments. Each segment provides two point correspondences (Fig. A.2), that is, four equations. However, as we can see from the example in Fig. 4.39, the endpoints of segments that have been detected often are not as well localized as the *line* itself, in terms of the direction and distance to the origin; therefore, a more reliable matching can be obtained by using only the line information, again providing two equations for each correspondence.³

A simple way to describe line-to-line correspondences consists of replacing them by two *point-to-line* correspondences, where the two model points \mathbf{L}^1 and \mathbf{L}^2 can be chosen arbitrarily from the respective line. Therefore, we can formulate the problem as follows: Given a model line $(\mathbf{L}^1, \mathbf{L}^2)$ and a corresponding image line $\mathbf{l} = (l_1, l_2, l_3)^T$ in homogeneous coordinates, find a transformation T such that both projections $(PT \cdot \mathbf{L}^1)$ and $(PT \cdot \mathbf{L}^2)$ lie on l .

For noisy measurements, the error can again be formulated in two ways, by using homogeneous or standard coordinates. Concerning the latter, we also assume that the orthogonal line direction $\mathbf{n} = (l_1, l_2)$ is normalized $\|\mathbf{n}\| = 1$, so that the third component $l_3 = d$ represents the actual distance to the origin. Therefore, we have

- Algebraic error:

$$\mathbf{l}^T (PT \cdot \mathbf{L}^1) = \mathbf{l}^T (PT \cdot \mathbf{L}^2) = 0 \quad (\text{A.33})$$

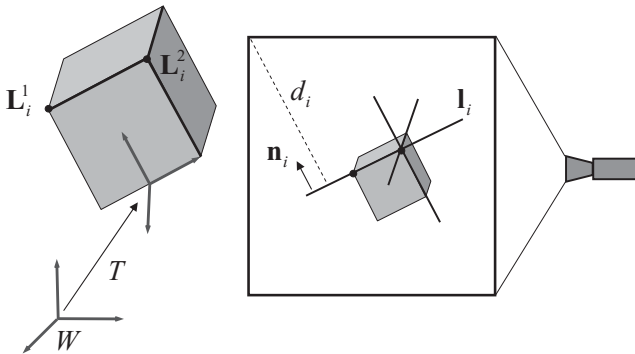


Figure A.2 Pose estimation from line correspondences.

³Alternatively, the endpoint information can still be included, but with a higher covariance (i.e., a lower weight) in the LSE procedure.

- Geometric error:

$$\mathbf{n}^T \pi(PT \cdot \mathbf{L}^1) + d = \mathbf{n}^T \pi(PT \cdot \mathbf{L}^2) + d = 0 \quad (\text{A.34})$$

which in a least-squares setting become:

- Algebraic LSE:

$$T^* = \arg \min_{T \in \mathcal{T}} \sum_{i=1}^N \left[(\mathbf{l}_i^T \cdot PT \cdot \mathbf{L}_i^1)^2 + (\mathbf{l}_i^T \cdot PT \cdot \mathbf{L}_i^2)^2 \right] \quad (\text{A.35})$$

- Geometric LSE:

$$T^* = \arg \min_{T \in \mathcal{T}} \sum_{i=1}^N \left[(\mathbf{n}_i^T \pi(PT \cdot \mathbf{L}_i^1) + d_i)^2 + (\mathbf{n}_i^T \pi(PT \cdot \mathbf{L}_i^2) + d_i)^2 \right] \quad (\text{A.36})$$

and all of the minimization procedures defined previously for point correspondences can be applied here as well.

A.2.1 2D-2D Line Correspondences

In the case of line matching, for many 2D-2D transforms apart from the general homography, the geometric LSE is fully equivalent to the algebraic LSE. In fact, if the transformation is a subgroup of the affine,

$$T = \begin{bmatrix} A & \mathbf{t} \\ 0 & 1 \end{bmatrix} \quad (\text{A.37})$$

we have

$$\mathbf{n}^T \pi(PT \cdot \mathbf{L}) + d = \mathbf{l}^T (PT \cdot \mathbf{L}) = \mathbf{l}^T P \cdot \begin{pmatrix} A\tilde{\mathbf{L}} + \mathbf{t} \\ 1 \end{pmatrix} \quad (\text{A.38})$$

where $\tilde{\mathbf{L}}$ are the first two (nonhomogeneous) coordinates of \mathbf{L} .

The constant 3×3 homography P can be removed by preprocessing lines \mathbf{l} :

$$\bar{\mathbf{l}}^T = \mathbf{l}^T P = (\bar{\mathbf{n}}^T, \bar{d}) \quad (\text{A.39})$$

which we can see as the “dual” version of the point preprocessing step [eq. (A.7)]. Furthermore, if the parametrization of T is linear in \mathbf{p} , the problem becomes linear:

$$\mathbf{p}^* = \operatorname{argmin}_{\mathbf{p} \in \mathbb{R}^D} \sum_{i=1}^N \left[(\bar{\mathbf{n}}_i^T \mathcal{L}_i^1 \cdot \mathbf{p} + \bar{d}_i)^2 + (\bar{\mathbf{n}}_i^T \mathcal{L}_i^2 \cdot \mathbf{p} + \bar{d}_i)^2 \right] \quad (\text{A.40})$$

where the matrix \mathcal{L} is defined in the same way as \mathcal{X} , according to the transformation group. This equation can be written in the more compact form

$$\mathbf{p}^* = \operatorname{argmin}_{\mathbf{p} \in \mathbb{R}^D} \sum_{i=1}^N \|\mathcal{N}\mathcal{L}_i \cdot \mathbf{p} + \bar{\mathbf{d}}_i\|^2 \quad (\text{A.41})$$

with

$$\mathcal{N}\mathcal{L}_i \equiv \begin{bmatrix} \bar{\mathbf{n}}_i^T \mathcal{L}_i^1 \\ \bar{\mathbf{n}}_i^T \mathcal{L}_i^2 \end{bmatrix}, \quad \bar{\mathbf{d}}_i = \begin{bmatrix} \bar{d}_i \\ \bar{d}_i \end{bmatrix} \quad (\text{A.42})$$

A.3 POINT AND LINE CORRESPONDENCES

When both point and line correspondences are available, pose estimation is obtained simply by minimizing the sum of the respective error terms (which we do not repeat here). In particular, the algebraic error is obtained by minimizing the sum of eqs. (A.6) and (A.35), while the geometric error is the sum of eqs. (A.1) and (A.36). However, special care should be taken concerning the covariances, since point and line residuals, both algebraic and geometric, may not have the same amount of noise. In that case, each error term should be *weighted* by the respective inverse covariance, respectively w_p and w_l .

In the 2D-2D case, for a linear parametrization $T(\mathbf{p})$ we can work directly with geometric errors and solve the problem in one step. In particular, we recall the data preprocessing steps

$$\bar{\mathbf{y}} = P^{-1}\mathbf{y}, \quad \bar{\mathbf{I}} = \mathbf{I}^T P \quad (\text{A.43})$$

and therefore, for N_l line and N_p point correspondences, we have⁴

$$\mathbf{p}^* = \operatorname{argmin}_{\mathbf{p} \in \mathbb{R}^D} \left(w_l \sum_{i=1}^{N_l} \|\mathcal{N}\mathcal{L}_i \cdot \mathbf{p} + \bar{\mathbf{d}}_i\|^2 + w_p \sum_{j=1}^{N_p} \|\mathcal{X}_j \cdot \mathbf{p} - \bar{\mathbf{y}}_j\|^2 \right) \quad (\text{A.44})$$

with $\mathcal{N}\mathcal{L}_i$ and $\bar{\mathbf{d}}_i$ defined by eq. (A.42).

⁴Notice the different signs in the two error terms.

A.4 COMPUTATION OF THE PROJECTIVE DLT MATRICES

To compute the DLT matrices F_i and \mathbf{f}_i defined in eq. (A.32) for estimating the linear parametrization \mathbf{q} , we split the algebraic error [eq. (A.31)] into two parts:

$$[\mathbf{y}_i]_{\times} [\bar{A}A(\mathbf{q}) \quad \bar{A}\mathbf{t}(\mathbf{q})] \cdot \mathbf{x}_i + [\mathbf{y}_i]_{\times} [0_{3 \times 3} \quad \bar{\mathbf{t}}] \cdot \mathbf{x}_i \quad (\text{A.45})$$

where

$$[\mathbf{y}_i]_{\times} \equiv \begin{bmatrix} 0 & -y_i^3 & y_i^2 \\ y_i^3 & 0 & -y_i^1 \\ -y_i^2 & y_i^1 & 0 \end{bmatrix} \quad (\text{A.46})$$

is the cross-product matrix. Then consider the product

$$\bar{A}[A \quad \mathbf{t}]\mathbf{x}_i = W\mathbf{x}_i \quad (\text{A.47})$$

with $W = \bar{A}[A \quad \mathbf{t}]$ a 3×4 projection matrix. If we express it row-wise,

$$W = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \end{bmatrix} \quad (\text{A.48})$$

where \mathbf{w}_j is the j th row, we can write

$$W\mathbf{x}_i = \begin{bmatrix} \mathbf{x}_i^T & \mathbf{0}^T & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{x}_i^T & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{0}^T & \mathbf{x}_i^T \end{bmatrix} \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \end{bmatrix} \quad (\text{A.49})$$

so that the error term $[\mathbf{y}_i]_{\times} W\mathbf{x}_i$ becomes

$$[\mathbf{y}_i]_{\times} W\mathbf{x}_i = \hat{X}_i \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \end{bmatrix} \quad (\text{A.50})$$

where

$$\hat{X}_i \equiv \begin{bmatrix} \mathbf{0}^T & -y_i^3 \mathbf{x}_i^T & y_i^2 \mathbf{x}_i^T \\ y_i^3 \mathbf{x}_i^T & \mathbf{0}^T & -y_i^1 \mathbf{x}_i^T \\ -y_i^2 \mathbf{x}_i^T & y_i^1 \mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \quad (\text{A.51})$$

Next we express each element of W as a scalar product:

$$W_{hk} = \bar{\mathbf{a}}_h^T \mathbf{p}_k$$

$$\bar{A} = \begin{bmatrix} \bar{\mathbf{a}}_1^T \\ \bar{\mathbf{a}}_2^T \\ \bar{\mathbf{a}}_3^T \end{bmatrix}, \quad [A \quad \mathbf{t}] = [\mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3 \quad \mathbf{p}_4] \quad (\text{A.52})$$

where \bar{A} is expressed row-wise and T is expressed column-wise. Equivalently, we can write

$$W = \hat{A} \mathbf{p} \quad (\text{A.53})$$

where \mathbf{p} contains the 12 entries of $[A \quad \mathbf{t}]$ and

$$\hat{A} \equiv \begin{bmatrix} \bar{\mathbf{a}}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{a}}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{a}}_3 \end{bmatrix}^T \quad (\text{A.54})$$

Our assumption about the parametrization \mathbf{q} is that

$$\mathbf{p} = Q \cdot \mathbf{q} + \mathbf{p}_0 \quad (\text{A.55})$$

with a suitable $12 \times D$ matrix Q and vector \mathbf{p}_0 . Therefore, we have

$$[\mathbf{y}_i]_{\times} \bar{A} [A \quad \mathbf{t}] \mathbf{x}_i = \hat{X}_i \hat{A} (Q \cdot \mathbf{q} + \mathbf{p}_0) \quad (\text{A.56})$$

The first term of the sum is $F_i \cdot \mathbf{q}$, with

$$F_i = \hat{X}_i \hat{A} Q \quad (\text{A.57})$$

while the second term does not depend on \mathbf{q} and therefore can be moved to the right-hand side of the LSE. Furthermore, the other term on the right-hand side can be developed similarly:

$$[\mathbf{y}_i]_{\times} [0_{3 \times 3} \quad \bar{\mathbf{t}}] \mathbf{x}_i = \hat{X}_i \hat{\mathbf{t}} \quad (\text{A.58})$$

with

$$\hat{\mathbf{t}} = [0 \quad 0 \quad 0 \quad \bar{t}_1 \quad 0 \quad 0 \quad 0 \quad \bar{t}_2 \quad 0 \quad 0 \quad 0 \quad \bar{t}_3]^T \quad (\text{A.59})$$

so that, finally, the LSE problem becomes

$$\mathbf{q}^* = \underset{\mathbf{q} \in \mathcal{R}^D}{\operatorname{argmin}} \sum_{i=1}^N \|F_i \cdot \mathbf{q} - \mathbf{f}_i\|^2 \quad (\text{A.60})$$

where

$$F_i = \hat{X}_i \hat{A} Q, \quad \mathbf{f}_i = \hat{X}_i (\hat{A} \mathbf{p}_0 - \hat{\mathbf{t}}) \quad (\text{A.61})$$

are the data matrices.

APPENDIX B

POSE REPRESENTATION

As a complement to Appendix A in this appendix we provide detailed examples of two-dimensional pose representations and, if available, a direct estimation procedure from point correspondences. In the following we consider only problems where correspondences are between spaces of the *same* dimension (i.e., 2D-2D or 3D-3D).

Additional symbols that are used:

n	Space dimension (usually $n = 2$ or $n = 3$)
I_n	$n \times n$ Identity matrix
\mathbf{t}_n	$n \times 1$ Translation vector
R_n	$n \times n$ Rotation matrix: $R_n^T R_n = I_n$
s	Uniform scale factor
$D_n = \text{diag}(s_1, \dots, s_n)$	Nonuniform scale matrix
A_n	$n \times n$ Linear transformation
\mathbf{v}_n	$n \times 1$ Perspective distortion vector

B.1 POSES WITHOUT ROTATION

We first examine transforms that do not involve rotations because of the linearity and relative simplicity of the parametrization.

B.1.1 Pure Translation

The simplest case is a pure translation (Fig. B.1):

$$T = \begin{bmatrix} I_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.1})$$

which has n dof and can be estimated from at least one point correspondence. The geometric error is

$$T^* = \underset{\mathbf{t}}{\operatorname{argmin}} \sum_{i=1}^N \|A_i \mathbf{t} - \mathbf{b}_i\|^2 \quad (\text{B.2})$$

with

$$\begin{aligned} A_i &= I_n \\ \mathbf{b}_i &= \mathbf{y}_i - \mathbf{x}_i \end{aligned} \quad (\text{B.3})$$

that is, $T^* = \operatorname{argmin}_{\mathbf{t}} \|A\mathbf{t} - \mathbf{b}\|^2$ with

$$A = \begin{bmatrix} I_n \\ \vdots \\ I_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{bmatrix}$$

This is a linear problem, solved by $\mathbf{t}^* = A^+ \mathbf{b}$; in this case, the solution corresponds to the displacement of point centroids μ :

$$\mathbf{t}^* = \frac{1}{N} \sum_{i=1}^N (\mathbf{y}_i - \mathbf{x}_i) \equiv \mu_y - \mu_x \quad (\text{B.4})$$

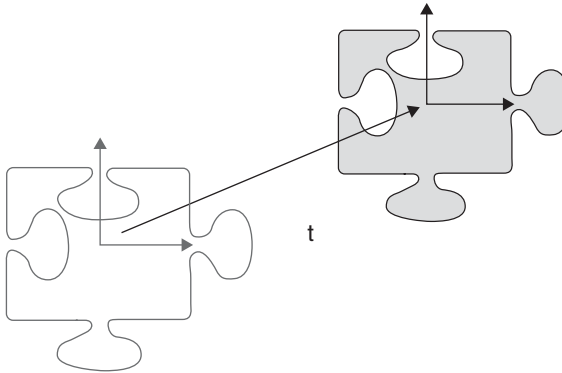


Figure B.1 Pure translation.

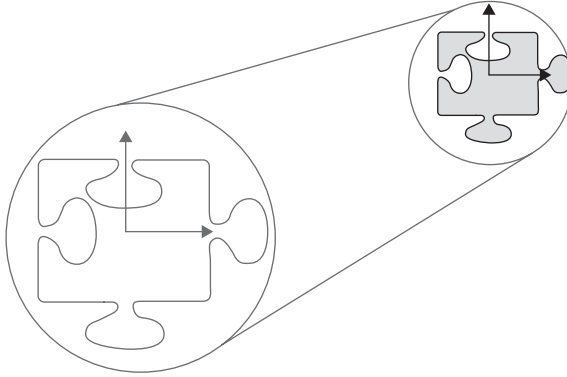


Figure B.2 Translation and uniform scale.

B.1.2 Translation and Uniform Scale

Next, we add a uniform scale (Fig. B.2):

$$T = \begin{bmatrix} sI_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.5})$$

A geometric error results:

$$\sum_{i=1}^N \|T\mathbf{x}_i - \mathbf{y}_i\|^2 = \sum_{i=1}^N \left\| \begin{bmatrix} sx_i^1 + t^1 - y_i^1 \\ \vdots \\ sx_i^n + t^n - y_i^n \end{bmatrix} \right\|^2 = \left\| A \cdot \begin{bmatrix} s \\ t^1 \\ \vdots \\ t^n \end{bmatrix} - \mathbf{b} \right\|^2 \quad (\text{B.6})$$

with

$$A = \begin{bmatrix} \mathbf{x}_1 & I_n \\ \vdots & \vdots \\ \mathbf{x}_N & I_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \quad (\text{B.7})$$

This is again a linear problem in $\mathbf{p} = (s, t^1, \dots, t^n)$.

B.1.3 Translation and Nonuniform Scale

A nonuniform scale with translation (Fig. B.3) is given by

$$T = \begin{bmatrix} D_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.8})$$

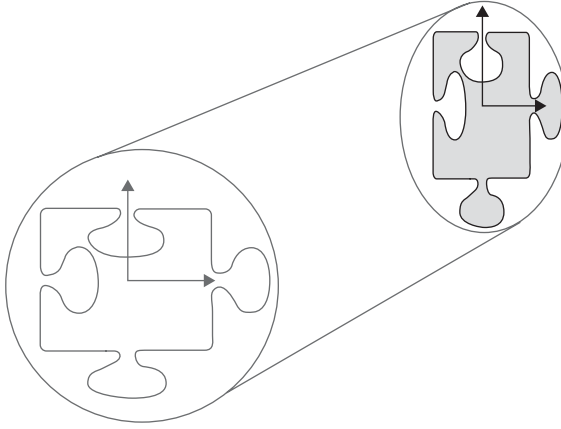


Figure B.3 Translation and nonuniform scale.

with $D_n = \text{diag}(s_1, \dots, s_n)$. This is similar to the previous problem, but with a different scale for each dimension. By minimizing the geometric error, we have

$$T^* = \underset{(s_1, \dots, s_n, \mathbf{t})}{\text{argmin}} \left\| \begin{bmatrix} \text{diag}(x_1^1, \dots, x_1^n) & I_n \\ \vdots & \vdots \\ \text{diag}(x_N^1, \dots, x_N^n) & I_n \end{bmatrix} \begin{bmatrix} s_1 \\ \vdots \\ s_n \\ \mathbf{t} \end{bmatrix} - \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_n \end{bmatrix} \right\|^2 \quad (\text{B.9})$$

B.2 PARAMETERIZING ROTATIONS

Before introducing transformations involving rotations, we review briefly some well-known parametrizations for the *special-orthogonal group* $\text{SO}(n)$ of pure rotations in \mathbb{R}^n . In general, the $\text{SO}(n)$ group consists of orthogonal matrices R_n , with the property $R_n^T R_n = I_n$ and the special condition $\det(R) = +1$. The latter ensures that *mirroring* the model is avoided, which in \mathbb{R}^3 corresponds to a switch between *right-* and *left-handed* coordinate frames.

The orthogonality constraint reduces the degrees of freedom of a rotation matrix with respect to the total number of elements n^2 . In particular, in $\text{SO}(2)$ we have only $4 - 3 = 1$ free parameter, which is the rotation angle θ , while in $\text{SO}(3)$ we are left with $9 - 6 = 3$. For many applications, it is necessary to have an explicit parametrization of the degrees of freedom in place of the implicit constraints given above. In $\text{SO}(2)$, the only degree of freedom can be used directly to express the matrix

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (\text{B.10})$$

which corresponds to the nonlinear transformation $\mathbf{y} = R\mathbf{x}$:

$$y_1 = \cos \theta x_1 - \sin \theta x_2 \quad (\text{B.11})$$

$$y_2 = \sin \theta x_1 + \cos \theta x_2 \quad (\text{B.12})$$

However, in $\text{SO}(3)$ the parametrization problem is far more complex. In fact, there are several possible solutions, with different properties from the geometric and computational point of view. The most intuitive solution is given by *Euler angles*, expressing any rotation as a cascade of three elementary rotations around each axis of the respective Cartesian frame (Fig. B.4). For example, the *yaw-pitch-roll* angles, or *x-y-z*, are defined by

$$R(\alpha, \beta, \gamma) = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma) \quad (\text{B.13})$$

where

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}, \quad R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix} \quad (\text{B.14})$$

$$R_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

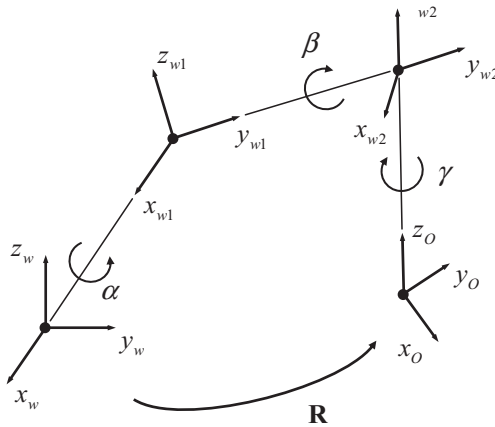


Figure B.4 Euler angles for parametrizing $\text{SO}(3)$.

However, this representation suffers from the presence of singular configurations, also called *gimbal lock*, because for $\beta = \pi/2$ any pair $\alpha = \gamma$ gives the same rotation matrix, and therefore there are infinite solutions to the *inverse problem* (i.e., computing the pose parameters from R). This problem is present for any sequence of axes chosen, such as the standard z - x - z used to model the wrist of industrial manipulators.

An alternative representation is given by the *axis-angle* pair (Fig. B.5). In fact, it can be shown that any rotation matrix has a unit eigenvalue, corresponding to a given eigenvector \mathbf{u} such that $R\mathbf{u} = \mathbf{u}$. By choosing $\|\mathbf{u}\| = 1$, this vector provides a direction coincident with the unique rotation *axis*, left unmodified by the transformation. Therefore, the rotation can be fully parametrized by a pair (\mathbf{u}, θ) , where θ is the rotation *angle*, which can be computed as the angle between \mathbf{v} and $R\mathbf{v}$, where \mathbf{v} is any vector orthogonal to \mathbf{u} .

In order to have a three-parameter representation, we can multiply together the axis and angle, obtaining the *rotation vector* $\rho \equiv \mathbf{u}\theta$. In this case one can show that the rotation matrix can be computed through the *matrix exponential* formula

$$R(\rho) = \exp([\rho]_{\times}) \equiv \sum_{k=0}^{\infty} \frac{1}{k!} [\rho]_{\times}^k = I + [\rho]_{\times} + \frac{1}{2} [\rho]_{\times}^2 + \frac{1}{6} [\rho]_{\times}^3 + \dots \quad (\text{B.15})$$

where $[\rho]_{\times}$ is the skew-symmetric *cross-product* matrix

$$[\rho]_{\times} \equiv \begin{bmatrix} 0 & -\rho_3 & \rho_2 \\ \rho_3 & 0 & -\rho_1 \\ -\rho_2 & \rho_1 & 0 \end{bmatrix} \quad (\text{B.16})$$

A closed-form expression for R is given by the *Rodrigues formula*:

$$R(\rho) = I + [\mathbf{u}]_{\times} \sin \theta + [\mathbf{u}]_{\times}^2 (1 - \cos \theta) \quad (\text{B.17})$$

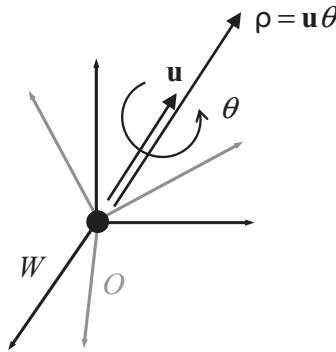


Figure B.5 Axis-angle representation.

which for small angles $\theta \rightarrow 0$ is approximated by

$$R(\rho) \approx I + [\rho]_{\times} \quad (\text{B.18})$$

corresponding to the Lie algebra representation of the tangent space to $\text{SO}(3)$.

Unlike Euler angles, the inverse problem $R \rightarrow \rho$ always provides only one solution, which can be computed with

$$\theta = \cos^{-1} \left(\frac{\text{trace}(R) - 1}{2} \right), \quad \mathbf{u} = \frac{1}{2 \sin \theta} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \quad (\text{B.19})$$

except for the cases $\theta \rightarrow k\pi$ with k integer. In particular, for $\theta = 0$ the solution is trivially $\rho = 0$, whereas for $\theta = \pi$ there are *two* solutions, $\pm \rho$, which is the only exception. Alternatively, one can use the eigenvalue decomposition of R since the axis is given by the unit eigenvector whereas the angle is given by one of the complex eigenvalues, $\cos \theta \pm i \sin \theta$.

Another common parametrization for three-dimensional rotations is given by *unit quaternions*. Quaternions are hypercomplex numbers, which can be seen as generalizations of complex numbers, given by $(a + b\mathbf{i})$, where $\mathbf{i}^2 = -1$. In fact, complex numbers are ordered pairs (a, b) with, in addition the operations of sum and product by a scalar, already defined in \mathfrak{R}^2 , the product operation

$$(a, b) \otimes (c, d) \equiv (a + b\mathbf{i})(c + d\mathbf{i}) = (ac - bd) + (bc - ad)\mathbf{i} \quad (\text{B.20})$$

If we generalize this definition, we can define a quaternion as a *4-tuple* $(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k})$, or simply (a, b, c, d) , where $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\mathbf{j}\mathbf{k} = -1$. In this case, a and $\mathbf{w} = (b, c, d)$ are called the *scalar* and the *vector* part of the quaternion, respectively.

From the axiomatic definition one can compute the internal product \otimes , in this case called the *Hamilton product*, from the name of their inventor, the mathematician Sir William Rowan Hamilton [72]:

$$q_1 \otimes q_2 = (a_1, \mathbf{w}_1) \otimes (a_2, \mathbf{w}_2) = (a_1 a_2 - \mathbf{w}_1 \cdot \mathbf{w}_2, a_1 \mathbf{w}_2 + a_2 \mathbf{w}_1 + \mathbf{w}_1 \times \mathbf{w}_2) \quad (\text{B.21})$$

which involves the scalar and cross products of the vector parts. Still in analogy with complex numbers, we can also define the *conjugate* of a quaternion $q = (a, \mathbf{w})$ as $\bar{q} = (a, -\mathbf{w})$. Finally, by introducing the Euclidean norm $\|q\|^2 = a^2 + b^2 + c^2 + d^2$, we can define a *unit* quaternion if $\|q\| = 1$.

Unit quaternions can be related to rotations in $\text{SO}(3)$ in the following way: Given a vector $\mathbf{v} \in \mathfrak{R}^3$ and a unit quaternion q , we can write $q = (\cos(\theta/2), \mathbf{w} \sin(\theta/2))$ for some values of (θ, \mathbf{w}) , with $\|\mathbf{w}\| = 1$. Then, if we

define by $(0, \mathbf{v})$ the imaginary quaternion associated with \mathbf{v} , we can compute the product

$$\mathbf{v}' = q \otimes \mathbf{v} \otimes \bar{q} \quad (\text{B.22})$$

where, with some abuse of notation, we denote by \mathbf{v} the associated quaternion. It can be shown that this operation is geometrically equivalent to *rotating* \mathbf{v} into \mathbf{v}' around the axis-angle pair (θ, \mathbf{w}) . Therefore, the unit quaternion is fully equivalent to a rigid rotation, given by the matrix

$$R(q) = \begin{bmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2ac + 2bd \\ 2ad + 2bc & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2ab + 2cd & a^2 - b^2 - c^2 + d^2 \end{bmatrix} \quad (\text{B.23})$$

The advantage of quaternions is that they provide a fully nonsingular representation, where there is always a *one-to-two* correspondence $R \leftrightarrow \pm q$ between a rotation and a unit quaternion, so that every rotation can be represented by two antipodal points on a hypersphere of radius 1 in \mathfrak{R}^4 , without exceptions. Moreover, the polynomial expression in $R(q)$, as well as the inverse formula, is very simple and numerically stable for estimation purposes.

However, the price that must be paid is the higher dimension, removed by the constraint $\|q\| = 1$, which must be enforced to prevent infinite solutions. In fact, in the formula above, $R(kq) = R(q), \forall k \neq 0$. This can be imposed by means of penalty functions or, better, as a hard *quadratic* constraint in q , which then requires constrained optimization techniques such as Lagrange multipliers.

B.3 POSES WITH ROTATION AND UNIFORM SCALE

Next we consider transforms involving rotations and the related estimation procedures. The latter are in principle nonlinear problems because of the rotation matrix R_n , but fortunately, due to the nature of the problem, the geometric error can be globally optimized in one step, using the SVD decomposition.

B.3.1 Similarity (Roto-translation and Uniform Scale)

We start by giving the solution for the general *similarity* transform (Fig. B.6):

$$T = \begin{bmatrix} sR_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.24})$$

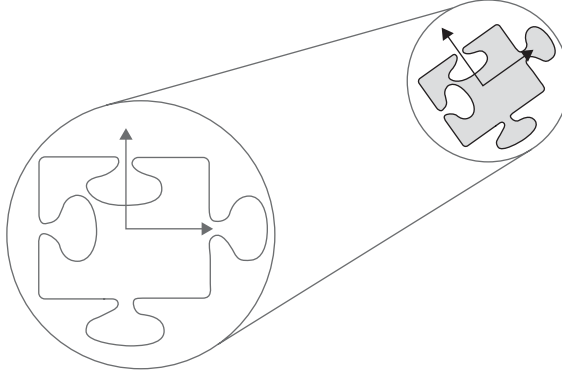


Figure B.6 Similarity (rigid roto-translation and uniform scale).

with uniform scale, rotation, and translation, and deduce its subcases later. Omitting the n index, the LSE solution in terms of (s, R, \mathbf{t}) has been given by Umeyama [156] and consists of the following computations:

- Mean vectors: $\mu_y = (1/N) \sum_{i=1}^N \mathbf{y}_i, \mu_x = (1/N) \sum_{i=1}^N \mathbf{x}_i$
- Variance of the norms: $\sigma_y^2 = (1/N) \sum_{i=1}^N \|\mathbf{y}_i - \mu_y\|^2, \sigma_x^2 = (1/N) \sum_{i=1}^N \|\mathbf{x}_i - \mu_x\|^2$
- Cross-covariance matrix ($n \times n$): $\Sigma_{yx} = (1/N) \sum_{i=1}^N (\mathbf{y}_i - \mu_y)(\mathbf{x}_i - \mu_x)^T$
- SVD of the cross-covariance: $\Sigma_{yx} = UDV^T$
- Sign correction for $\det(R)$: $S = \begin{cases} I, & \text{if } \det(\Sigma_{yx}) \geq 0 \\ \text{diag}(1, 1, 1, \dots, -1), & \text{if } \det(\Sigma_{yx}) < 0 \end{cases}$
- Rotation reconstruction: $R = USV^T$
- Scale reconstruction: $s = (1/\sigma_x^2) \text{tr}(DS)$
- Translation vector: $\mathbf{t} = \mu_y - sR\mu_x$

B.3.2 Rotation and Uniform Scale

Similar to our previous description, but without translation (Fig. B.7),

$$T = \begin{bmatrix} sR_n & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.25})$$

This implies that the mean vectors are $\mu_y = \mu_x = \mathbf{0}$. Therefore, we have:

- Variance of the norms: $\sigma_y^2 = (1/N) \sum_{i=1}^N \|\mathbf{y}_i\|^2, \sigma_x^2 = (1/N) \sum_{i=1}^N \|\mathbf{x}_i\|^2$
- Cross-covariance matrix ($n \times n$): $\Sigma_{yx} = (1/N) \sum_{i=1}^N \mathbf{y}_i \mathbf{x}_i^T$

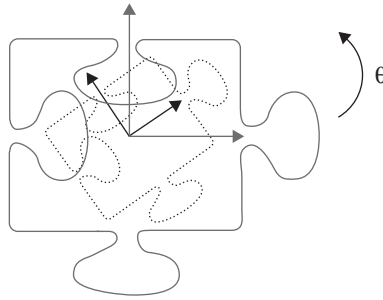


Figure B.7 Rotation and uniform scale.

and the remaining steps are the same, except for the translation vector, which is not computed.

B.3.3 EUCLIDEAN (RIGID BODY) TRANSFORM

By removing the scale ($s = 1$), we get the Euclidean transform (Fig. B.8), corresponding to a rigid roto-translation that preserves distances between points:

$$T = \begin{bmatrix} R_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.26})$$

where $\sigma_y = \sigma_x = 1$. The algorithm is the same as for the similarity, except that the variance of the norms, as well as the scale factor, are not computed.

B.3.4 Pure Rotation

Finally, if both scale and translation are removed, we have the *absolute orientation* problem (Fig. B.9):

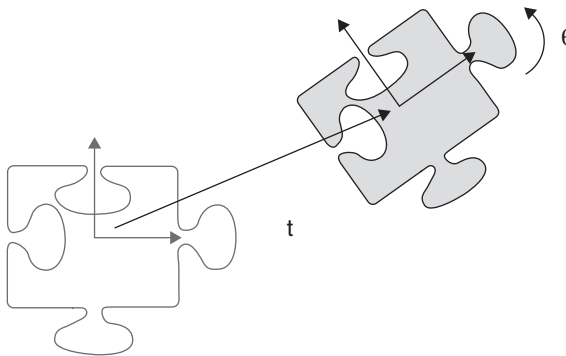


Figure B.8 Euclidean transform (rigid roto-translation).

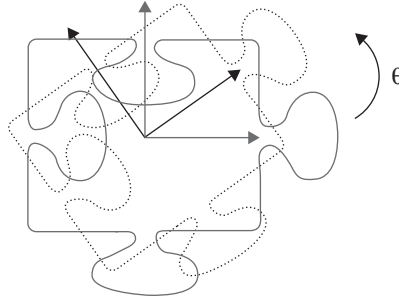


Figure B.9 Pure rotation.

$$T = \begin{bmatrix} R_n & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.27})$$

which is solved by:

- Cross-covariance matrix ($n \times n$): $\Sigma_{yx} = (1/N) \sum_{i=1}^N \mathbf{y}_i \mathbf{x}_i^T$
- SVD of the cross-covariance: $\Sigma_{yx} = U D V^T$
- Sign correction for $\det(R)$: $S = \begin{cases} I, & \text{if } \det(\Sigma_{yx}) \geq 0 \\ \text{diag}(1, 1, 1, \dots, -1), & \text{if } \det(\Sigma_{yx}) < 0 \end{cases}$
- Rotation reconstruction: $R = U S V^T$

B.4 AFFINITY

The affine transform is the most general case of transforms not involving the nonlinear scaling from homogeneous to standard coordinates, and it has the property of preserving parallel lines and planes but, in general, not the angles between them (Fig. B.10). It is represented by the matrix

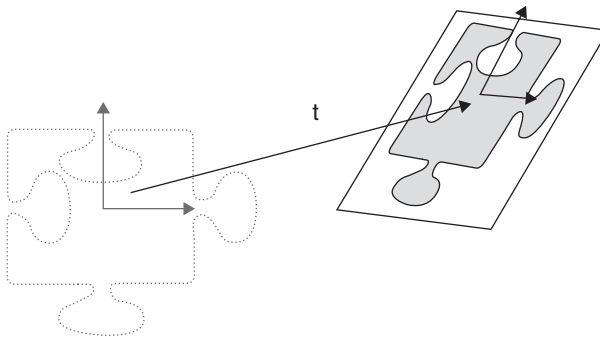


Figure B.10 Affine transform (linear + constant).

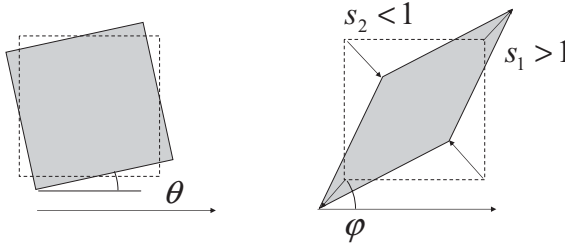


Figure B.11 Decomposition of a two-dimensional affine (or linear) transform, where $R_{\text{rot}}(\theta)$ represents a pure rotation and $R_{\text{def}}(\phi)$ is responsible for a change of axes before scaling. (From [77].)

$$T = \begin{bmatrix} A_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.28})$$

Using the SVD, we can express $A_n = R_{\text{rot}} R_{\text{def}}^T D_n R_{\text{def}}$ with $D_n = \text{diag}(s_1, \dots, s_n)$ and $s_1 \geq \dots \geq s_n$. The orthogonal matrix R_{rot} accounts for a rigid rotation, while R_{def} provides a change of axes for the deformation. For example, in the case $n=2$ we have $A_2 = R_{\text{rot}}(\theta) R_{\text{def}}(-\phi) D_2 R_{\text{def}}(\phi)$, where θ is a pure planar rotation while ϕ changes the two axes along which the nonuniform scaling D_2 takes place (Fig. B.11).

Estimation of affine parameters is a linear problem in the geometric error

$$[A^*, \mathbf{t}^*] = \underset{(A, \mathbf{t})}{\text{argmin}} \sum_{i=1}^N \|A \mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 = \underset{\mathbf{p}}{\text{argmin}} \|\mathcal{X}_i \mathbf{p} - \mathbf{y}_i\|^2 \quad (\text{B.29})$$

with \mathbf{p} the $n(n+1)$ -vector

$$\mathbf{p} = [A_{11} \quad \dots \quad A_{1n} \quad \dots \quad A_{n1} \quad \dots \quad A_{nn} \quad t^1 \quad \dots \quad t^n]^T \quad (\text{B.30})$$

and \mathcal{X}_i a coefficient matrix, a function of \mathbf{x}_i :

$$\mathcal{X}_i = \begin{bmatrix} \mathbf{x}_i^T & \dots & \mathbf{0}^T \\ \dots & \dots & \dots \\ \mathbf{0}^T & \dots & \mathbf{x}_i^T \end{bmatrix} \quad (\text{B.31})$$

The solution is obtained by stacking the \mathcal{X}_i matrices and \mathbf{y}_i vectors together and solving for \mathbf{p}^* .

Similar equations can be written for the purely linear case ($\mathbf{t} = \mathbf{0}$):

$$T = \begin{bmatrix} A_n & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.32})$$

with n^2 parameters

$$\mathbf{p} = [A_{11} \quad \cdots \quad A_{1n} \quad \cdots \quad A_{n1} \quad \cdots \quad A_{nn}]^T \quad (\text{B.33})$$

and

$$\mathcal{X}_i = \begin{bmatrix} \mathbf{x}_i^T & \cdots & \mathbf{0}^T \\ \cdots & \cdots & \cdots \\ \mathbf{0}^T & \cdots & \mathbf{x}_i^T \end{bmatrix} \quad (\text{B.34})$$

B.5 POSES WITH ROTATION AND NONUNIFORM SCALE

Unlike the similarities of Section B.1, estimating nonuniform scale *and* rotation at the same time (Fig. B.12) is difficult to do in closed form. However, since the number of degrees of freedom is close to full affinity (6 vs. 5 in two dimensions or 12 vs. 9 in three dimensions), we can solve first for an affinity and then upgrade to the nonuniform scale by removing the additional degrees of freedom. This procedure is suboptimal and therefore should be refined with a nonlinear LSE.

Now we reconsider the cases with and without translation. The first is given by

$$T = \begin{bmatrix} R_n D_n & \mathbf{t}_n \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.35})$$

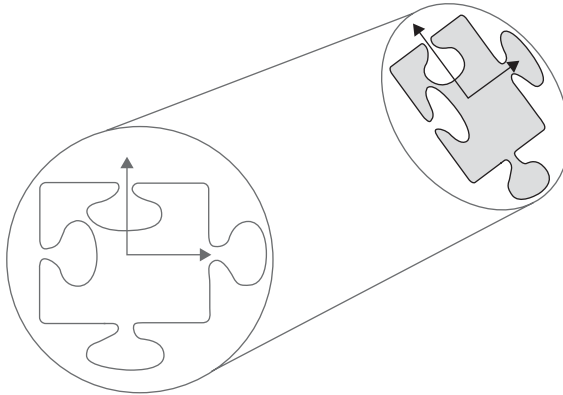


Figure B.12 Roto-translation with nonuniform scale.

and the estimation procedure is as follows:

1. Solve for an affine transform as in the preceding section, and find A_n and \mathbf{t}_n .
2. Using the SVD, express $A = USV^T = (UV^T)(VSV^T)$, with $S = \text{diag}(s_1, \dots, s_n)$ and $s_1 \geq \dots \geq s_n$. Set $R_{\text{rot}} = UV^T$ and $D^* = VSV^T$. Then remove the off-diagonal elements of D^* , in fact, in the absence of affine deformations, $R_{\text{def}} = V^T$ should be either the identity or a pure permutation of axes, since unlike the D^* , the elements of s_i may not be in decreasing order.
3. Finally, since this solution is generally *not* the optimal LSE, it is recommended that a Gauss–Newton optimization be run on the geometric error.

Next, if we remove the translation vector,

$$T = \begin{bmatrix} R_n D_n & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{B.36})$$

This is a special case of the preceding one, where $\mathbf{t} = \mathbf{0}$. Therefore, we can solve it as a purely linear transform, with n^2 degrees of freedom, then upgrade A_n to the desired manifold by removing the R_{def} matrix as in the preceding section.

B.6 GENERAL HOMOGRAPHY: THE DLT ALGORITHM

The most general transformation in homogeneous coordinates is a *homography*,

$$T = \begin{bmatrix} A_n & \mathbf{t}_n \\ \mathbf{v}_n^T & 1 \end{bmatrix} \quad (\text{B.37})$$

which is defined up to a scale factor that we can remove, for example, by setting $T(n, n) = 1$. Here, the additional vector \mathbf{v}_n accounts for transforms that do not preserve parallelism (Fig. B.13). By applying the projection operator $\pi(\bullet)$ to nonhomogeneous coordinates, we have a nonlinear geometric LSE; therefore, the pose estimation problem must be formulated in two steps: algebraic and geometric error minimization.

In particular, let us consider the case $n = 2$, which is the most common in the literature: for example, when modeling transformations of planes under perspective cameras. Concerning the algebraic error, after preprocessing the data, that is, removing the constant P from the data points \mathbf{y}_i (Section A.1.3), we formulate the DLT in terms of the nine redundant entries of T :

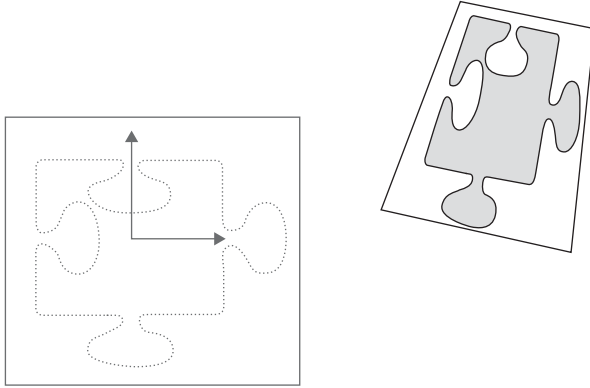


Figure B.13 General two-dimensional homography.

$$T^* = \operatorname{argmin}_T \sum_{i=1}^N \|\mathbf{y}_i \times (T\mathbf{x}_i)\|^2 \quad (\text{B.38})$$

using homogeneous coordinates. If we take the first two terms of each cross product, we have for each point two homogeneous equations,

$$\begin{bmatrix} 0^T & -y_i^3 \mathbf{x}_i^T & y_i^2 \mathbf{x}_i^T \\ y_i^3 \mathbf{x}_i^T & \mathbf{0}^T & -y_i^1 \mathbf{x}_i^T \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = 0 \quad (\text{B.39})$$

where the \mathbf{h}_i are the three rows of $T = \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \end{bmatrix}$.

If A is the $2N \times 9$ stacked matrix of all the terms on the left-hand side above, we get the DLT equation

$$\mathbf{h}^* = \operatorname{argmin}_h \|A\mathbf{h}\| \quad (\text{B.40})$$

where A is rank deficient, providing ∞^1 solutions in \mathbf{h} . By further imposing a scaling condition such as $\|\mathbf{h}\|=1$, the solution is obtained by the SVD decomposition

$$A = USV^T \quad (\text{B.41})$$

as the last column of $\mathbf{h}^* = V_3$, corresponding to the smallest singular value $S_{3,3}$.

In addition, the following *normalization* technique [77] ensures better numerical stability:

1. Remove the centroids. Compute the mean values $\mu_y = (1/N) \sum_{i=1}^N \mathbf{y}_i$ and $\mu_x = (1/N) \sum_{i=1}^N \mathbf{x}_i$ and remove them from the respective point sets.
2. When carrying out isotropic scaling, make sure that the average distance from the mass center (which is 0) is equal to $\sqrt{2}$; that is, that the “average” point of both sets is $(1,1,1)^T$.
3. Estimate the homography \tilde{T}^* from the normalized point sets.
4. Remove the two normalizations by

$$T^* = T_y^{-1} \tilde{T}^* T_x \quad (\text{B.42})$$

where T_x and T_y are the equivalent transformations induced by steps 1 and 2.

NOMENCLATURE

Indices and Operators

$k = 0, 1, \dots$	Temporal index of current sensory data (integer)
$c = 1, \dots, C$	Camera (or sensor) index
$m = 1, \dots, M$	Modality index
$o = 1, \dots, O$	Object index
$n, i = 1, \dots, N$	State hypothesis index
$d = 1, \dots, D$	Pose parameter index (D = degrees of freedom)
$l = 1, \dots, L$	Link index for articulated (or multipart) models
X -level	Processing level of data: pixel, feature, or object level
\equiv	Equal by definition to
\approx	Approximately equal to
\propto	Proportional to
\cdot	Vector scalar product or matrix product
\times	Vector cross product
\otimes	Quaternion product
$[\mathbf{v}]_{\times}$	3×3 Cross-product matrix of \mathbf{v}

Object State and Dynamics

t_k	Time stamp of sensory data
τ_k	Interval between time stamps: $t_k - t_{k-1}$
s_k^o	State of object o at time t_k , containing pose and kinematic parameters, and possibly appearance or structural parameters
$\mathbf{s}_k \equiv (s_k^1, \dots, s_k^O)$	Ensemble state, of all objects at time t_k
$s_k^o = f_k^o(s_{k-1}^o, w_k^o, \tau_k)$	Single-target dynamics for object o
w_k^o	Motion (or <i>process</i>) noise
Q_k^o	Process noise covariance
W_k^o	Process noise gain, for additive models
$s_k^{o-} = f_k^o(s_{k-1}^o, 0, \tau_k)$	Predicted state of object o at time t_k
$F_k^o(s_{k-1}^o)$	Jacobian of dynamical model with respect to the state
$P(s_k^o s_{k-1}^o, \tau_k)$	Single-target predictive prior
$(\bar{s}_k^o, \Sigma_k^o)$	Mean and covariance matrix of a single-target state distribution
$\mathcal{N}(\bar{s}, \Sigma)$	Multivariate Gaussian state distribution
$\mathbf{s}_k = \mathbf{f}_k(\mathbf{s}_{k-1}, \mathbf{w}_k, \tau_k)$	Generative, multitarget dynamics
$\mathbf{s}_k^- \equiv (s_k^{1-}, \dots, s_k^{O-})$	Predicted ensemble state at time t_k
$(\bar{\mathbf{s}}_k, \Sigma_k)$	Mean and covariance matrix of an ensemble-state distribution
$P(\mathbf{s}_k \mathbf{s}_{k-1})$	Multitarget predictive prior
$\psi(s_k^{o1}, s_k^{o2})$	Penalty term for two interacting targets (Markov random field)
$\Psi(\mathbf{s}_k)$	Penalty function for multiple, interacting targets

Measurement Models

$I_{k,c}$	Raw data from camera c , captured at time t_k
$Z \equiv \langle z, h, H, e, R \rangle_X$	Measurement data at processing level X : pixel maps, feature sets, or state-space estimates
z	Data observed
h	Data predicted
v	Measurement noise
R	Measurement noise covariance
V	Measurement noise gain for additive models
H	Jacobian of h with respect to the state predicted
e	Residual (or <i>innovation</i>) between data expected and data observed

S	Innovation covariance
$Z_{k,m,c}^o$	Target-associated measurement for object o from modality m and sensor c
$z_{k,m,c}^o = h_{k,m,c}^o(s_k^o, v_k^o)$	Measurement model for noninteracting targets (without occlusion handling)
$h_{k,m,c}^{o-} = h_{k,m,c}^o(s_k^o, 0)$	Data predicted for object o
$z_{k,m,c}^o = h_{k,m,c}^o(\mathbf{s}_k, \mathbf{v}_k)$	Single-target measurement over the ensemble state (handling occlusions)
$P(Z_{k,m,c}^o s_k^o), P(Z_{k,m,c}^o \mathbf{s}_k)$	Likelihood model for a single modality and sensor, respectively, without and with occlusion handling
$Z_k^o = \mathcal{F}\{Z_{k,1..M,1..C}^o\}$	Data fusion for object o from all cameras and modalities
$P(Z_k^o s_k^o), P(Z_k^o \mathbf{s}_k)$	Likelihood of integrated measurements for object o
$Z_k^o \equiv (Z_0^o, \dots, Z_k^o)$	History of integrated measurements for object o , up to time t_k
$P(s_k^o Z_{k-1}^o)$	Prior state distribution of object o , before Z_k^o
$P(s_k^o Z_k^o)$	Posterior state distribution of object o , including Z_k^o

Pose Representation and Sensor Space Mapping

W	World reference frame
$\mathbf{x} = (x_1, x_2, x_3)^T$	Space point
$\mathbf{x}_c = (x_{c,1}, x_{c,2}, x_{c,3})^T$	Space point expressed in camera coordinates (similar notation for object and world coordinates)
$\mathbf{y} = (y_1, y_2)^T$	Image point
$\bar{\mathbf{x}} = (\mathbf{x}^T, 1)^T, \bar{\mathbf{y}} = (\mathbf{y}^T, 1)^T$	Homogeneous coordinates of space and image points
H	Planar homography (3×3)
$\mathbf{p}^o, \delta \mathbf{p}^o$	Absolute and incremental pose of object o
$p_d^o, \delta p_d^o; d = 1, \dots, D$	Individual pose parameters
$T_{i,j}(\mathbf{p}), \delta T_{i,j}(\delta \mathbf{p})$	Homogeneous (4×4) transformation matrix, respectively absolute and incremental, from frame i to frame j
$R_{i,j}, \mathbf{t}_{i,j}$	3×3 Rotation matrix and translation vector for Euclidean transforms
$J_{i,j}^d, \delta J_{i,j}^d$	Jacobian of the homogeneous transformation matrix $T_{i,j}$ with respect to the d -th pose parameter (absolute or incremental)

G_d^o	Lie algebra generator (4×4) for the d – th degree of freedom of object o
$T_{c,w}$	Extrinsic camera parameters (from world-to-camera coordinates)
K_c	Intrinsic calibration matrix
r_{y1}, r_{y2}	Horizontal and vertical screen resolution (in pixels)
\mathbf{C}	Camera center
f	Camera focal length
\mathbf{c}	Principal point
k_{r1}, k_{r2}	Radial distortion coefficients
$P_{c,w}$	World-to-camera screen projection matrix
$T_{w,l}$	Absolute world-referred pose of link l (for articulated models)
$T_{l_1,l_2}(\mathbf{p}), \delta T_{l_1,l_2}(\delta \mathbf{p})$	Local interlink transforms
\bar{T}_{l_1,l_2}	Structural (constant) interlink transform
$(\mathbf{p}, \delta \mathbf{p}) \rightarrow (\mathbf{p} + \delta \mathbf{p}, \mathbf{0})$	Additive pose update
$(T_{i,j}, \delta T_{i,j}) \rightarrow (T_{i,j} \cdot \delta T_{i,j}, I)$	Compositional pose update
$\mathbf{y} = \mathcal{W}_{o,c}(\mathbf{x}, \mathbf{p}^o, P_{c,w})$	Warp function, mapping from object o to camera screen c at pose p^o

BIBLIOGRAPHY

- [1] Active shape model training. [Online] http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/Papers/asm_overview.pdf.
- [2] Bidirectional reflectance distribution function (bdrf). [Online] http://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function.
- [3] The boost C++ library. [Online] <http://www.boost.org>.
- [4] Canny edge detector. [Online] <http://robotics.eecs.berkeley.edu/~sastry/ee20/cademo.html>.
- [5] CIE-LUV chromaticity diagrams. [Online] http://en.wikipedia.org/wiki/CIELUV_color_space.
- [6] CIE-XYZ tristimulus curves and chromaticity diagrams. [Online] http://en.wikipedia.org/wiki/CIE_1931_color_space.
- [7] The CMU digital camera drivers. [Online] <http://www.cs.cmu.edu/~iwan/1394/>.
- [8] Cone responses. [Online] <http://en.wikipedia.org/wiki/Color>.
- [9] Conversion between RGB and HSV color spaces. [Online] http://en.wikipedia.org/wiki/HSL_and_HSV.
- [10] The COTESYS excellence research cluster. [Online] <http://www.cotesys.org>.
- [11] Dense optical flow. [Online] <http://of-eval.sourceforge.net/>.
- [12] K-d tree. [Online] <http://en.wikipedia.org/wiki/Kd-tree>.
- [13] Level sets method. [Online] http://en.wikipedia.org/wiki/Level_set_method.
- [14] Level sets segmentation example. [Online] http://www.ceremade.dauphine.fr/~peyre/numerical-tour/tours/variational_segmentation/.

- [15] The OpenCV library. [Online] <http://opencv.willowgarage.com>.
- [16] OpenGL integer texture extension. [Online] http://www.opengl.org/registry/specs/EXT/texture_integer.txt.
- [17] The OpenGL library. [Online] <http://www.opengl.org>.
- [18] OpenGL occlusion query extension. [Online] http://www.opengl.org/registry/specs/ARB/occlusion_query.txt.
- [19] The OpenGL shader language. [Online] <http://www.opengl.org/documentation/glsl/>.
- [20] The OpenTL library. [Online] <http://www.opentl.org>.
- [21] Optical flow vector. [Online] http://en.wikipedia.org/wiki/Optical_flow.
- [22] Radial distortion. [Online] http://www.uni-koeln.de/~al001/radcor_files/hs100.htm.
- [23] Rendering equation. [Online] http://en.wikipedia.org/wiki/Rendering_equation.
- [24] RGB color space decomposition. [Online] http://en.wikipedia.org/wiki/RGB_color_model.
- [25] Silhouette detection software. [Online] <http://www.dgp.toronto.edu/~hertzman/sil/>.
- [26] David Adalsteinsson and James A. Sethian. A fast level set method for propagating interfaces. *J. Comput. Phys.*, 118(2):269–277, 1995.
- [27] Hamid Aghajan and Andrea Cavallaro. *Multi-camera Networks: Principles and Applications*. Academic Press, San Diego, CA, 2009.
- [28] A. A. Amini, T. E. Weymouth, and R. C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(9):855–867, 1990.
- [29] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Trans. Signal Process.*, 50(2):174–188, Feb. 2002.
- [30] Simon Baker and Iain Matthews. Equivalence and efficiency of image alignment algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, Vol. 1, p. 1090.
- [31] Simon Baker and Iain Matthews. Lucas–Kanade 20 years on: a unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, 2004.
- [32] Dana H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recogn.*, 13(2):111–122, 1981.
- [33] Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. Wiley, Hoboken, NJ, 2002.
- [34] Yaakov Bar-Shalom and Xiao-Rong Li. *Multitarget–Multisensor Tracking: Principles and Techniques*. YBS Publishing, 1995.
- [35] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, Feb. 2000.
- [36] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Comput. Vision Image Underst.*, 110(3):346–359, 2008.
- [37] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *IEEE Computer Society Conference on Computer Vision and Patter Recognition*, 1997, pp. 1000–1006.

- [38] Y. Benezeth, P. M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Review and evaluation of commonly-implemented background subtraction algorithms. In *Proceedings of the 19th International Conference on Pattern Recognition (ICPR)*, Dec. 8–11, 2008, Tampa, FL, pp. 1–4.
- [39] S. Benhimane and E. Malis. Homography-based 2D visual tracking and servoing. *Int. J. Robot. Res.*, 26(7):661–676, 2007.
- [40] Jeff Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *Technical Report TR-97-021*. International Computer Science Institute (ICSI), Berkeley, CA, 1998.
- [41] Stanley T. Birchfield and Shrinivas J. Pundlik. Joint tracking of features and edges. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–6.
- [42] Samuel S. Blackman and Robert Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, Nowood, MA, 1999.
- [43] Andrew Blake and M. Isard. *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer-Verlag, New York, 1998.
- [44] P. Bouthemy. A maximum likelihood framework for determining moving edges. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(5):499–511, 1989.
- [45] Gary R. Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technol. J.*, Q2, 1998.
- [46] Gary R. Bradski and James W. Davis. Motion segmentation and pose recognition with motion history gradients. *Mach. Vision Appl.*, 13(3):174–184, 2002.
- [47] David Bremner, Komei Fukuda, and Ambros Marzetta. Primal–dual methods for vertex and facet enumeration (preliminary version). In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pp. 49–56. ACM, New York, 1997.
- [48] Matthew Brown and David G. Lowe. Invariant features from interest point groups. In Paul L. Rosin and A. David Marshall, ed., *Proceedings of the British Machine Vision Conference 2002*, Cardiff, UK, Sep. pp. 2–5, 2002.
- [49] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *Int. J. Comput. Vision*, 22(1):61–79, 1997.
- [50] Tony F. Chan and Luminita A. Vese. Active contours without edges. *IEEE Trans. Image Process.*, 10(2):266–277, 2001.
- [51] James Joseph Clark and Alan L. Yuille. *Data Fusion for Sensory Information Processing Systems*. Kluwer Academic, Norwell, MA, 1990.
- [52] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.
- [53] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):564–575, 2003.
- [54] T. F. Cootes, G. J. Edwards, and C. J. Taylor. *Active Appearance Models*. Lecture Notes in Computer Science, Vol. 1407, pp. 484–498. Springer-Verlay, New York, 1998.
- [55] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models: their training and application. *Comput. Vision Image Underst.*, 61(1):38–59, 1995.

- [56] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.
- [57] Daniel Cremers, Mikaël Rousson, and Rachid Deriche. A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape. *Int. J. Comput. Vision*, 72(2):195–215, 2007.
- [58] James W. Davis and Aaron F. Bobick. The representation and recognition of human movement using temporal templates. In *1997 Conference on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, June 17–19, 1997, pp. 928–934. IEEE Computer Society, Washington, DC, 1997.
- [59] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Statist. Soc. B*, 39(1):1–38, 1977.
- [60] Tom Drummond and Roberto Cipolla. Visual tracking and control using Lie algebras. In *1999 Conference on Computer Vision and Pattern Recognition*, Los Alamitos, CA, Vol. 2, pp. 2652–2659. IEEE Computer Society, Washington, DC, 1999.
- [61] Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):932–946, 2002.
- [62] Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972.
- [63] J. Duetscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2000, Vol. 2, p. 2126.
- [64] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. *Technical Report*. Cornell Computing and Information Science, Ithaca, NY, Sept. 2004.
- [65] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.
- [66] Y. Freund and R. Schapire. A short introduction to boosting. *J. Jpn. Soc. Artif. Intel.*, 14(5):771–780, 1999.
- [67] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 124–133. ACM, New York, 1980.
- [68] Iryna Gordon and David G. Lowe. What and where: 3D object recognition with accurate pose. In *Toward Category-Level Object Recognition*. Lecture Notes in Computer Science, Vol. 4170, pp. 67–82. Springer-Verlag, New York, 2006.
- [69] Andreas Griesser, Stefaan De Roeck, Alexander Neubeck, and Luc Van Gool. GPU-based foreground–background segmentation using an extended collinearity criterion. In G. Greiner, J. Hornegger, H. Niemann, and M. Stamminger, eds., *Proceedings of Vision, Modeling, and Visualization*, 2005, pp. 319–326. IOS Press, Erlangen, Germany, Nov. 2005.
- [70] Gregory D. Hager and Peter N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(10):1025–1039, 1998.
- [71] David L. Hall and James Llinas. *Handbook of Multisensor Data Fusion*. CRC Press, Boca Raton, FL, Jun. 2001.

- [72] William Rowan Hamilton. On quaternions, or on a new system of imaginaries in algebra. *Philos. Mag.*, 25:489–495, Nov. 1844.
- [73] Bohyung Han, Ying Zhu, Dorin Comaniciu, and Larry S. Davis. Kernel-based Bayesian filtering for object tracking. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, CA, June 20–26, 2005, pp. 227–234. IEEE Computer Society, Washington, DC, 2005.
- [74] Robert Hanek and Michael Beetz. The contracting curve density algorithm: fitting parametric curve models to images using local self-adapting separation criteria. *Int. J. Comput. Vision*, 59(3):233–258, 2004.
- [75] C.J. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, Manchester, UK, 1988, pp. 147–151.
- [76] Chris Harris. Tracking with rigid models. In *Active Vision*, pp. 59–73. MIT Press, Cambridge, MA, 1993.
- [77] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, New York, 2004.
- [78] Aaron Hertzmann and Denis Zorin. *Illustrating smooth surfaces*. In *Proceedings of the 27th Annual Conference on Computer Graphics and Introactive Techniques*, pp. 517–526. ACM, New York, 2000.
- [79] Tom Heskes and Onno Zoeter. Expectation propogation for approximate inference in dynamic Bayesian networks. In Adnan Darwiche and Nir Friedman, eds., *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence*, University of Alberta, Edmonton, Alberta, Canada, Aug. 1–4, 2002, pp. 216–223. Morgan Kaufmann, San Francisco, 2002.
- [80] Berthold K. P. Horn and Brian G. Schunk. Determining optical flow. *Artif. Intell.*, 17:185–203, 1981.
- [81] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IEEE Trans. Inform. Theory*, 8(2):179–187, 1962.
- [82] Hunter Associates Laboratories. Hunter lab color scale. *Insight on Color*, 8(9), Aug 1996.
- [83] Piotr Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O’Rourke, eds., *Handbook of Discrete and Computational Geometry*, 2nd ed., pp. 877–892. Chapman & Hall/CRC Press, New York, 2004.
- [84] M. Isard and A. Blake. Condensation: conditional density propagation for visual tracking. *Int. J. Comput. Vision*, 29(1):5–28, 1998.
- [85] Michael Isard and Andrew Blake. Icondensation: Unifying low-level and high-level tracking in a stochastic framework. In *Proceedings of the 5th European Conference on Computer Vision*, London, Vol. I, pp. 893–908. Springer-Verlag, New York, 1998.
- [86] Deane B. Judd. The 1931 I.C.I. standard observer and coordinate system for colorimetry. *J. Opt. Soc. Am.*, 23(10):359–373, 1933.
- [87] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, Orlando, FL, 1997.
- [88] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proc. IEEE*, 92(3):401–422, 2004.

- [89] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME J. Basic Eng.*, 82(Seri. D):35–45, 1960.
- [90] Sing Bing Kang, Yin Li, Xin Tong, and Heung-Yeung Shum. Image-based rendering. *Found. Trends Comput. Graphics Vision*, 2(3), 2006.
- [91] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: active contour models. *Int. J. Comput. Vision*, V1(4):321–331, Jan. 1988.
- [92] Zia Khan, Tucker Balch, and Frank Dellaert. MCMC-based particle filtering for tracking a variable number of interacting targets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(11):1805–1918, 2005.
- [93] A. Khotanzad and Y. H. Hong. Invariant image recognition by Zernike moments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(5):489–497, 1990.
- [94] Whoi-Yul Kim and Yong-Sung Kim. A region-based shape descriptor using Zernike moments. *Signal Process. Image Commun.*, 16(1–2):95–102, 2000.
- [95] Y. S. Kim, J. H. Lee, H. M. Do, B. K. Kim, T. Tanikawa, K. Ohba, G. Lee, and S. H. Yun. Unscented information filtering method for reducing multiple sensor registration error. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Aug. 2008, pp. 326–331.
- [96] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [97] D. Koller, K. Daniilidis, and H. H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *Int. J. Comput. Vision*, 10(3):257–281, 1993.
- [98] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Am. Math. Soc.*, 7(1):48–50, Feb. 1956.
- [99] Jeff Lander. Shades of Disney: Opaquing a 3D world. *Game Dev. Mag.*, 7(3):15–20, Mar. 2000.
- [100] Claus Lenz, Giorgio Panin, and Alois Knoll. A GPU-accelerated particle filter with pixel-level likelihood. In *International Workshop on Vision, Modeling and Visualization*, Konstanz, Germany, Oct. 2008.
- [101] Vincent Lepetit and Pascal Fua. *Monocular Model-Based 3D Tracking of Rigid Objects: Foundations and Trends in Computer Graphics and Vision*. Now Publishers, 2005.
- [102] K. Levenberg. A method for the solution of certain problems in least squares. *Q. Appl. Math.*, 2:164–168, 1944.
- [103] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer, Dordrecht, The Netherlands, 1994.
- [104] David G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artif. Intell.*, 31(3):355–395, 1987.
- [105] David G. Lowe. Local feature view clustering for 3D object recognition. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, with CD-ROM*, Kauai, HI, 8–14 Dec. 2001, pp. 682–688. IEEE Computer Society, Washington, DC, 2001.
- [106] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.
- [107] D. G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer, Dordrecht, The Netherlands, Jun. 1985.

- [108] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (DARPA). In *Proceedings of the 1981 DARPA Image Understanding Workshop*, Apr. 1981, pp. 121–130.
- [109] Q.-T. Luong and O. D. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices. *Int. J. Comput. Vision*, 22(3):261–289, 1997.
- [110] John MacCormick and Michael Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *Proceedings of the 6th European Conference on Computer Vision*, London, 2000, Part II, pp. 3–19. Springer-Verlag, New York, 2000.
- [111] Frederik Maes, André Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens. Multimodality image registration by maximization of mutual information. *IEEE Trans. Med. Imag.*, 16(2):187–198, 1997.
- [112] Ravikanth Malladi, James A. Sethian, and Baba C. Vemuri. Shape modeling with front propagation: a level set approach. *IEEE Trans. Pattern Anal. and Mach. Intell.*, 17:158–175, 1995.
- [113] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11(2):431–441, 1963.
- [114] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman, San Francisco, 1982.
- [115] D. Marr and E. C. Hildreth. Theory of edge detection. *Proc. R. Soc. London*, 207:187–217, 1980.
- [116] Iain Matthews and Simon Baker. Active appearance models revisited. *Int. J. Comput. Vision*, 60(2):135–164, 2004.
- [117] Kamel Mekhnacha, Yong Mao, David Raulo, and Christian Laugier. Bayesian occupancy filter based “fast clustering-tracking” algorithm. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, 2008.
- [118] Rudolf Mester, Til Aach, and Lutz Dümbgen. Illumination-invariant change detection using a statistical collinearity criterion. In *DAGM-Symposium*, pp. 170–177, 2001.
- [119] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.
- [120] Krystian Mikolajczyk and Cordelia Schmid. Scale and affine invariant interest point detectors. *Int. J. Comput. Vision*, 60(1):63–86, 2004.
- [121] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.
- [122] Farahnaz Mohanna and Farzin Mokhtarian. An efficient active contour model through curvature scale space filtering. *Multimedia Tools Appl.*, 23(3):225–242, 2003.
- [123] Farzin Mokhtarian, Sadegh Abbasi, and Josef Kittler. Robust and efficient shape indexing through curvature scale space. In *Proceedings of the British Machine Vision Conference 1996*, University of Edinburgh, Edinburgh, UK, 1996. British Machine Vision Association, 1996.

- [124] Farzin Mokhtarian and Alan K. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(8):789–805, 1992.
- [125] Farzin Mokhtarian, Yoke Khim Ung, and Zhitao Wang. Automatic fitting of digitised contours at multiple scales through the curvature scale space technique. *Comput. Graph.*, 29(6):961–971, 2005.
- [126] Suraj Nair, Giorgio Panin, Thorsten Röder, Thomas Friedelhuber, and Alois Knoll. A distributed and scalable person tracking system for robotic visual servoing with 8 dof in virtual reality TV studio automation. In *Proceedings of the 6th IEEE International Symposium on Mechatronics and Its Applications*, Mar. 2009.
- [127] Suraj Nair, Giorgio Panin, Martin Wojtczyk, Claus Lenz, Thomas Friedelhuber, and Alois Knoll. A multi-camera person tracking system for robotic applications in virtual reality TV studio. In *Proceedings of the 17th IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 2008.
- [128] J. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7:308–313, 1965.
- [129] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, Nov. 1988.
- [130] G. Panin, T. Röder, and A. Knoll. Integrating robust likelihoods with Monte-Carlo filters for multi-target tracking. In *International Workshop on Vision, Modeling and Visualization*, Konstanz, Germany, Oct. 2008.
- [131] Giorgio Panin, Sebastian Klose, and Alois Knoll. Multi-target and multi-camera object detection with Monte-Carlo sampling. In *International Symposium on Visual Computing*, Las Vegas, NV, Dec. 2009.
- [132] Giorgio Panin, Sebastian Klose, and Alois Knoll. Real-time articulated hand detection and pose estimation. In *International Symposium on Visual Computing*, Las Vegas, NV, Dec. 2009, pp. 1131–1140.
- [133] Giorgio Panin and Alois Knoll. Real-time 3D face tracking with mutual information and active contours. In *International Symposium on Visual Computing*, Lake Tahoe, NV, 2007.
- [134] Giorgio Panin and Alois Knoll. Mutual information-based 3D object tracking. *Int. J. Comput. Vision*, 78(1):107–118, 2008.
- [135] Giorgio Panin, Alexander Ladikos, and Alois Knoll. An efficient and robust real-time contour tracking system. In *IEEE International Conference on Computer Vision Systems*, New York, 2006, pp. 44.
- [136] Giorgio Panin, Thorsten Röder, and Alois Knoll. Dynamic multi-level fusion with MCMC particle filters. In *International Workshop on Vision, Modeling and Visualization*, Konstanz, Germany, Oct. 2008.
- [137] Giorgio Panin, Erwin Roth, and Alois Knoll. Robust contour-based object tracking integrating color and edge likelihoods. In *International Workshop on Vision, Modeling and Visualization*, Konstanz, Germany, Oct. 2008.
- [138] Constantine Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *International Conference on Computer Vision*, 1998, pp. 555–562.

- [139] D. H. Parks and S. S. Fels. Evaluation of background subtraction algorithms with post-processing. In *IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*, Sept. 2008, pp. 192–199.
- [140] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *Proceedings of the 7th European Conference on Computer Vision*, Part I, London, 2002, pp. 661–675. Springer-Verlag, New York, 2002.
- [141] Markus Peura and Jukka Iivarinen. Efficiency of simple shape descriptors. In C. Arcelli, L. P. Cordella, and G. Sanniti di Baja, eds., *Advances in Visual Form Analysis*, pp. 443–451. World Scientific, Singapore, 1997.
- [142] Massimo Piccardi. Background subtraction techniques: a review. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, The Hague, The Netherlands, 2004, pp. 3099–3104.
- [143] Richard J. Radke, Srinivas Andra, Omar Al-Kofahi, and Badrinath Roysam. Image change detection algorithms: a systematic survey. *IEEE Trans. Image Process.*, 14(3):294–307, 2005.
- [144] David Reynard, Andrew Wildenberg, Andrew Blake, and John A. Marchant. Learning dynamics of complex motions from image sequences. In *Proceedings of the 4th European Conference on Computer Vision*, London, 1996, Vol. I, pp. 357–368. Springer-Verlag, New York, 1996.
- [145] Branko Ristic, Sanjeev Arulampalam, and Neil Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, Norwood, MA, 2004.
- [146] Paul L. Rosin. Measuring shape: ellipticity, rectangularity, and triangularity. *Mach. Vision Appl.*, 14(3):172–184, 2003.
- [147] Randi J. Rost. *OpenGL(R) Shading Language*. Addison Wesley Longman, Redwood City, CA, 2004.
- [148] Erwin Roth, Giorgio Panin, and Alois Knoll. Sampling feature points for contour tracking with graphics hardware. In *International Workshop on Vision, Modeling and Visualization*, Konstanz, Germany, Oct. 2008.
- [149] Janos Schanda. *Colorimetry: Understanding the CIE System*. Wiley, Hoboken, NJ, 2007.
- [150] G. Silveira and E. Malis. Real-time visual tracking under arbitrary illumination changes. In *IEEE Computer Vision and Pattern Recognition*, Minneapolis, MN, Jun. 2007.
- [151] Bjoern Stenger, Arasanathan Thayananthan, Philip H. S. Torr, and Roberto Cipolla. Model-based hand tracking using a hierarchical Bayesian filter. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28:1372–1384, 2006.
- [152] L. D. Stone, T. L. Corwin, and C. A. Barlow. *Bayesian Multiple Target Tracking*. Artech House, Norwood, MA, 1999.
- [153] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics: Intelligent Robotics and Autonomous Agents*. MIT Press, Cambridge, MA, Sept. 2005.
- [154] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Found. Trends Comput. Graph. Vision*, 3(3):177–280, 2007.
- [155] M. Uchinoumi, J. K. Tan, S. Ishikawa, T. Naito, and M. Yokota. High-speed human motion recovery employing back projection. *Artif. Life Robot.*, 10(2):112–115, Nov. 2006.

- [156] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, 1991.
- [157] Luca Vacchetti and Vincent Lepetit. Stable real-time 3D tracking using online and offline information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1385–1391, 2004.
- [158] Tom Vercauteren and Xiaodong Wang. Decentralized sigma-point information filters for target tracking in collaborative sensor networks. *IEEE Trans. Signal Process.*, 53(8–2):2997–3009, 2005.
- [159] Paul Viola and William M. Wells III. Alignment by maximization of mutual information. *Int. J. Comput. Vision*, 24(2):137–154, 1997.
- [160] Paul A. Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, 2004.
- [161] E. Wan and R. van der Merwe. *The Unscented Kalman Filter*. Wiley, Hoboken, NJ, 2001.
- [162] Thomas Weise. *Global Optimization Algorithms: Theory and Application*. Thomas Weise, July 16, 2007.
- [163] Greg Welch and Gary Bishop. An introduction to the Kalman filter. *Technical report*. University of North Carolina, Chapel Hill, NC, 1995.
- [164] Greg Welch and Gary Bishop. SCAAT: incremental tracking with incomplete information. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, New York, 1997, pp. 333–344.
- [165] Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: an overview. *Comput. Sci. Eng.*, 4(4):10–21, 1997.
- [166] H. J. Wolfson and Y. Lamdan. Geometric hashing: a general and efficient model-based recognition scheme. In *International Conference on Computer Vision*, 1988, pp. 238–249.
- [167] Robert J. Woodham. Photometric method for determining surface orientation from multiple images. *Opt. Eng.*, 19(1):139–144, 1980.
- [168] Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-time combined 2D + 3D active appearance models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004, Vol. 2, pp. 535–542.
- [169] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: a survey. *ACM Comput. Surv.*, 38(4):13, 2006.
- [170] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *IEEE International Conference on Computer Vision*, 1999, Vol. 1, p. 666.
- [171] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22:1330–1334, 2000.
- [172] Gernot Ziegler, Art Tevs, Christian Theobalt, and Hans-Peter Seidel. GPU point list generation through histogram pyramids. *Research Report MPI-I-2006-4-002*. Max-Planck-Institut für Informatik, Saarbrücken, Germany, Jun. 2006.
- [173] Z. Zivkovic and F. van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recogn. Lett.*, 27(7):773–780.

INDEX

- Absolute
 - conic, 23
 - orientation, 254, 274
 - prior, 240
- Abstraction level, 56, 163
- Acceptance ratio, 190, 192
- Active
 - appearance model (AAM), 36, 37, 151, 152, 157
 - contour, 104–107, 112
 - shape model (ASM), 31, 32, 37, 152
- Affine
 - piecewise, 31, 33, 151, 153
 - transform, 129, 131, 211, 212, 255, 257, 275, 278
- Algebraic error, 20
- Annealing, 188, 189
- Aperture problem, 144
- Appearance parameters, 36
- Autocorrelation, 53
- Autoregressive model (AR), 45, 46, 51–53
- Background
 - color, 57, 95, 199, 243
 - edges, 96
 - keypoints, 96
 - model, 13, 78, 94, 96
- Back-projection
 - features, 10, 13, 57
 - histogram, 89, 90, 243
 - points, 39, 41, 42, 147, 148
 - silhouette, 97
- Bayesian
 - prediction, 8, 43, 240
 - tracking, 6, 7, 13, 51, 60, 73, 74, 77, 162–167, 172, 174, 175, 178, 180, 182, 190, 192, 193, 215, 216, 228, 232, 244, 245
 - update, 8–10, 57, 70, 73, 151, 163, 195, 240
- Best-bin-first algorithm (BBF), 138, 140
- Bhattacharyya
 - coefficient, 89, 92, 94, 227
 - distance, 90, 210, 228, 240

- Bidirectional reflectance distribution function (BDRF), 35
- Binary space partitioning (BSP), 139
- Blobs
 - description, 64, 97, 101
 - detection, 56, 70, 74, 75, 92, 95, 97, 199
 - matching, 57, 61, 97
 - tracking, 67, 92, 97
- Bottom-up localization, 199
- Bounding box, 80, 191, 210
- Brightness constancy, 143, 144, 145
- Burn-in sample, 190, 192
- Camera
 - affine, 3, 18, 19, 27, 40, 43, 240
 - calibration, 15, 18, 20–22, 24, 25
 - center, 14, 18, 19, 41, 171
 - extrinsic parameters, 12, 13, 18, 21, 23–26, 42, 43, 128
 - focal length, 14–16, 19, 20
 - intrinsic parameters, 12, 13, 18
 - pinhole, 3, 13, 14
 - principal point, 14, 15
 - radial distortion, 16
 - skew distortion, 15
 - stereo, 3, 71, 132, 204, 247, 248
- Charge-coupled device (CCD), 14, 15
- Clustering
 - blobs, 78, 97, 199
 - color space, 90
 - histogram, 193, 196
 - keypoints, 211
 - mean shift, 90, 92, 208
 - minimum spanning tree (MST), 98
 - segments, 121
 - state space, 209
- Color
 - chromaticity, 82–85
 - CIE-Lab, 83
 - CIE-Luv, 84
 - CIE-RGB, 80, 81, 94, 125, 219
 - CIE-XYZ, 82
 - gamut, 80, 82
 - Gaussian mixture model (GMM), 85, 243
 - Grassman's law, 81
 - histogram, 5, 34, 58, 69, 85, 90, 94, 210, 227, 229, 240, 244, 247
 - hue–saturation–value (HSV), 34, 84, 85, 94, 227, 240
 - kernel density, 87, 88
 - matching, 89, 93, 123, 210, 233
 - modality, 5, 12, 216, 227, 229, 235, 240, 244
 - perception, 80, 82–84
 - segmentation, 56, 68, 77, 84, 89, 91, 199, 241, 243
 - spaces, 56, 80–86, 89, 94, 124
 - statistics, 6, 34, 57, 64, 71, 78, 79, 85, 94, 122–124, 127, 237, 242
 - tristimulus, 81–83
- Contour
 - active, 104–107, 112
 - curvature scale space (CSS), 102, 103
 - descriptor, 70, 71, 97, 98
 - energy, 104–109
 - geodesic, active, 109
 - lines, 37, 58, 114, 119, 121, 122, 204
 - model, 5, 6, 26, 112, 120
 - points, 58, 59, 64, 66, 99, 106, 117, 119, 123, 124, 238, 245, 246
- Contracting curve density (CCD), 58, 70, 122–127, 245, 247
- Cornerness, 130, 131
- Cross-covariance, 177, 178, 273, 275
- Damping coefficient, 50, 52
- Data association, 5, 7, 10, 55, 58, 59, 66–68, 169, 198, 204, 216
 - dynamic, 5, 8, 55, 112, 242
 - feature level, 94, 99, 112, 114, 117, 120, 122, 127, 133, 148, 151
 - multihypothesis, 9, 59, 60, 64, 65, 68, 97, 117, 180, 190, 199
 - object-level, 94, 120, 122, 127, 133, 142, 151
 - pixel-level, 94, 95, 104, 116, 120, 142, 148, 151
 - short baseline, 128
 - static, 5, 55, 138, 197, 204
 - validation gate, 5, 65, 66, 117, 122, 237, 238
 - wide baseline, 128, 133
- Data fusion, 5, 7, 9–11, 60, 62, 70, 77, 215, 229, 230, 245
 - dynamic, 44, 55, 71, 73–75, 167, 230, 239, 243

- feature level, 75
- multimodal, 70, 71, 76, 124, 237
- multisensor, 70–72, 76
- object-level, 75
- off-line and online, 57
- pixel level, 74, 242, 243
- static, 55, 71, 72–75, 199, 230, 247
- De-bayering, 218
- Deformation modes, 12, 32, 33, 106, 151, 251
- Depth, 14, 18–20, 27, 42, 43, 55, 68, 69, 71, 128, 236
 - map, 5, 42
 - of k-d tree, 139
- Difference of Gaussians (DoG), 134, 135
- Direct linear transform (DLT), 18, 20, 201, 202, 204, 254–258, 278, 279
- Discriminative model, 44, 198
- Distance transform (DT), 108, 109, 116, 117, 120
- Dynamics
 - ensemble, 43, 44
 - learning, 52
 - model, 3, 5, 6, 8, 12, 13, 43, 44, 48, 50, 51, 52, 71, 73, 74, 95, 152, 162, 163, 167, 172, 173, 176, 182, 183, 186–190, 193, 194, 215, 232, 233, 236, 240, 243
 - steady state, 46–48, 185
- Entropy, 87, 159
- Eulerian, 61, 107, 165
- Expectation maximization (EM), 87, 123
- Exponential
 - decay, 88, 123, 125, 126
 - growth, 185, 191
 - map, 29, 221, 253
 - matrix, 45
- False
 - alarms, 5, 59, 62–64, 67, 118, 119, 199
 - features, 113
- Feature
 - descriptor, 34, 58, 61, 64, 92, 97, 102, 121, 129
 - geometric descriptor, 58
 - global descriptor, 64, 80, 96, 99
 - level, 61, 62, 64, 68, 69, 75, 89, 94, 95, 104, 112, 117, 119, 120, 122–124, 126, 127, 133, 142, 147, 148, 151, 204, 228–230, 240, 241
 - off-line sampling, 10, 57
 - online update, 57, 151
 - photometric descriptor, 58
- Filter
 - annealed particle, 188
 - extended information (EIF), 175, 180
 - extended Kalman (EKF), 60, 166, 174, 175, 180
 - grid-based, 192, 193
 - information (IF), 173
 - iterated, extended Kalman (IEKF), 175
 - Kalman gain, 173
 - Markov chain Monte Carlo (MCMC), 181, 190, 192, 244
 - sampling–importance–resampling (SIR), 6, 74, 126, 163, 166, 180–183, 185, 186, 216, 230, 234–237, 239, 240, 241, 244, 247
 - unscented information (UIF), 178
 - unscented Kalman (UKF), 51, 164, 166, 174, 176, 178, 216
 - unscented transform (UT), 164, 176, 180
- Fragment shader, 56
- Fuzzy
 - histogram binning, 93, 160
 - pixel classification, 89, 125, 126
- Gauss–Newton, 97, 119, 120, 123, 125, 126, 145, 147, 148, 153–158, 161, 167, 168, 170, 175, 202, 253, 278
- Generative model, 44, 181, 182, 198
- Geometric
 - hashing, 204, 208, 209
 - invariance, 27, 28, 99, 100, 101, 103, 127, 129, 131, 132, 133, 138, 205, 206
 - invariants, 27
 - transformation manifold, 25, 27–29, 128, 155, 156, 198, 205, 252–254, 257, 278
- Global
 - feature, 34, 64, 80, 92, 96, 99, 104, 147, 149, 153

Global (*con'd*)

- measurement, 9, 61, 70, 72, 73, 166, 243
- optimization, 145, 158, 163, 189, 197, 198, 254, 272
- search, 6

GPU

- general-purpose programming (GP-GPU), 4, 55, 56, 58, 59, 95, 208, 212, 215, 237, 238, 243, 245
- OpenGL shader language (GLSL), 56, 58, 62, 147, 151

Gradient constraint equation, 143

Ground

- appearance, 12, 36, 39
- shape, 12, 39

Hamilton–Jacobi, 108

Hash table, 205–208

Histogram

- back-projection, 89, 90, 243
- clustering, 193, 196
- color, 5, 34, 58, 68, 69, 85, 86, 89, 90, 93, 94, 199, 210, 227–229, 240, 244, 247
- co-occurrence matrix, 150, 160, 161
- distance, 90, 210, 228, 234, 240
- fuzzy binning, 93, 137, 160
- oriented gradients (HOG), 34, 64, 80, 136, 138, 142
- state space, 165, 193

Homography, 22, 23, 27, 128, 129, 254, 260, 278, 280

Hough transform, 212, 259

Huber M-estimator, 169

Hu moments, 99, 100

Hyperplane, 139, 140

Importance

- distribution, 74, 182, 184–187, 189
- sampling, 51, 165, 181, 183, 186
- weighted resampling, 186

Incremental

- pose, 28, 29, 40, 50, 160
- state, 50
- transform, 30, 32, 156

Inliers/outliers, 169–172, 202

Innovation, 5, 179, 230

- covariance, 65, 66

Intensity edges

- Canny detector, 114–117, 119, 237
- crease, 113
- depth discontinuity, 113
- detection, 70, 71, 96, 114, 204, 237
- Marr–Hildreth detector, 70, 114, 115, 117, 134
- match, 116
- matching, 58, 74, 113, 117, 119, 238, 245, 250
- modality, 5, 34, 56, 58, 78, 114, 123, 216, 237, 239, 241, 245, 246
- silhouette, 113
- texture, 113

K-d tree, 138–140

Kernel

- bandwidth, 87, 88, 116, 133, 209
- basis function, 87–89, 93, 114, 116, 130, 133, 134, 160
- density estimation (KDE), 87, 90, 93, 94, 165, 209
- density maximization, 90, 92, 209
- profile, 88, 89, 91
- radially symmetric, 87, 88, 130

Key frame, 34, 133, 240

Keypoint

- descriptor, 129, 136, 137–140, 148, 197, 212, 213
- detection, 77, 127, 133, 212
- matching, 132, 212, 213
- scale invariant, 70, 135, 211, 212

Kullback–Leibler divergence, 90, 160

Lagrangian, 61, 62, 107

Laplacian of Gaussian (LoG), 105, 114–116, 134

Learning

- active shape/appearance model, 36

Least-squares estimation (LSE), 60, 75, 150, 152, 153, 158, 169, 171, 175, 238, 254, 266

- algebraic error, 252, 254, 255, 259, 260, 262–264, 278

- geometric error, 252–254, 259, 260, 266, 267, 272, 273, 276–278

- linear, 75, 158

- nonlinear, 152, 167, 168, 172, 202, 245, 253, 254

- normal equations, 144
- robust, 153, 169
- sequential vs. batch, 175
- weighted, 122, 127, 259
- Level sets, 106–109, 111, 112
 - function, 106, 108, 109, 111
 - geodesic, 109
 - geometric, 108
 - narrowband, 111
- Lie
 - algebra, 29, 30, 41, 60, 253, 254
 - generators, 29, 253
 - group, 28, 253
- Likelihood
 - color-based, 85, 210, 231, 244
 - contour-based, 58, 117, 239
 - feature-level, 65, 89, 94, 120, 229, 240
 - function, 5, 6, 58, 60, 79, 97, 126, 162, 163, 165, 167, 169, 183–186, 190, 192–195, 197, 215, 227, 230, 231, 234, 240
 - image segmentation, 110, 123, 124
 - log-, 52, 53, 167
 - maximum (ML), 18, 21, 22, 24–26, 52, 53, 60, 61, 67, 70, 77, 86, 87, 90, 92, 94, 119, 120, 123, 126, 127, 133, 151, 162, 164, 166–168, 170, 200, 209, 227, 251, 252
 - multihypotheses, 5, 64, 120, 190, 237, 241
 - multimodal, 9, 62, 70, 73–75, 118, 119, 230, 239, 243
 - multiresolution, 70, 123, 126, 187
 - object level, 67
 - of AR sequence, 52
 - of ensemble state, 68, 191, 244
 - of kernel density, 91, 92
 - of partitioned model, 186, 187
 - pixel-level, 63, 68, 74, 89, 94, 95, 241, 243
 - ratio, 96, 192
 - shape-based, 242
 - single hypothesis, 65, 118, 126, 245
- Local
 - coordinates, 12, 26, 35, 36
 - features, 6, 9, 10, 26, 34, 58, 64, 66, 77, 92, 94, 105, 123, 124, 126, 127, 145, 148, 149
 - optima, 67, 77, 91, 102, 106, 135, 136, 158, 187, 208, 209
 - search, 61, 67, 97, 104, 147, 197, 245
 - transform, 30, 32, 252, 253
- Lucas–Kanade (LK)
 - inverse-compositional, 156, 157
 - optical flow, 143, 144, 147
 - template matching, 153–155, 158
- Luminance, 71, 80, 82, 83, 114
- Mapping
 - between color spaces, 80, 82, 83
 - camera space to image, 12, 13, 153
 - exponential, 29, 221, 253
 - image to space, 41
 - object to image (*see also* warp), 40, 143, 174, 223, 227
 - object to measurement space, 174
 - texture, 34
- Markov
 - chain Monte Carlo (MCMC), 181, 190, 192, 244
 - process, 45
 - random field (MRF), 192
- Matching (*see also* data association), 5
- Maximum a posteriori (MAP), 9, 110, 162–164, 167–170, 175, 193
- Measurement
 - expected, 59, 117, 230
 - feature-level, 62, 64, 75, 123, 126, 148, 204
 - noise, 60, 65, 174, 207, 230, 252
 - noise covariance, 59, 173, 210
 - object-level, 67, 68, 75, 119, 163, 184, 245
 - pixel-level, 61, 62, 63, 142, 147, 152, 167, 173, 187
 - space, 178, 186, 187
 - synchronous/asynchronous, 8, 44, 46, 72, 73, 77, 172
- M-estimator, 152, 169, 238
 - Huber, 169
 - Tukey, 169
- Metric, 4, 14, 15, 18, 19, 109, 121, 122, 228
- Metropolis–Hastings (MH), 190, 192
- Minimum spanning tree (MST), 98
- Missing detection, 59, 60, 61, 62, 63, 64, 118, 199, 204

- Moments, 64, 78, 99, 181
 - central, 100
 - Hu, 99, 100
 - raw, 99, 100
 - scale-invariant, 100
 - Zernike, 100–102
- Morphological operator, 97, 199
- Motion field, 78, 97, 140, 143
- Moving edges, 67
- Multiresolution pyramid, 70, 134, 135, 148, 151, 161
- Mutual information (MI), 150, 151, 159, 161, 247
- Navigation, 1, 40
- Nearest neighbor (NN), 10, 61, 65, 66, 97, 118, 138, 139, 140, 197, 208, 238, 242
- Nonphotorealistic rendering (NPR), 117, 120
- Normalized cross-correlation, 149, 150, 158, 160, 179
- Object
 - detection, 2, 4, 6, 7, 58, 70, 95, 99, 129, 165, 197–199, 202, 204, 210–212
 - level, 55, 56, 59, 61, 67, 68, 75, 119, 120, 122, 123, 127, 133, 142, 148, 151, 163, 184, 245, 247
 - shadow, 61–63, 68, 69, 94, 96, 97, 113, 128, 143, 243
- Occlusion
 - external, 4, 6, 63, 148, 152, 161, 197, 211
 - feature-level, 69
 - GPU query, 58
 - mutual (between targets), 4, 5, 34, 55, 58, 68, 69, 190, 199, 216, 236, 244
 - pixel-level, 68, 152
 - self-, 4, 58, 113
- Octave, 134, 135
- Optical
 - axis, 14
 - flow, 62, 67, 104, 132, 140, 142–145, 147, 153
- Orthographic projection, 19, 20, 207, 208
- Parallel processing, 2, 11, 56, 77, 166, 181, 189, 204, 216, 242
- Parametrization
 - of color distribution, 86
 - of contour, 106, 107, 121
 - of contour energy, 111
 - of dynamical model, 45
 - of geometric transforms, 25, 28, 40, 155, 198, 254, 255, 257, 260, 261
 - of photometric transforms, 35, 36, 157
- Partial differential equation (PDE), 108
- Partition
 - of an image, 109–111
 - of measurement space, 186, 187
 - of state space, 138, 139, 185–187, 193
- Perspective from n points (PnP), 171, 172
- Piecewise
 - affine transform, 31–33, 151, 153
 - linear surface approximation, 31
- Pinhole camera model, 3, 13, 14
- Pixel-level, 56, 61–63, 68, 69, 74, 89, 94, 95, 104, 112, 116, 120, 122, 127, 133, 142, 143, 147, 148, 151, 152, 167, 173, 187, 199, 242, 243
- Polygonal mesh, 3, 12, 26, 34, 35, 147
- Pose
 - additive update, 28, 50, 51, 155, 168, 221
 - compositional update, 28, 34, 50, 51, 155, 168, 221, 245, 253, 254
 - incremental parameters, 28, 29, 39, 40, 50, 160
 - inverse-compositional update, 28, 148, 151, 155, 156
 - parameters, 3, 13, 18, 28, 34, 40, 50, 52, 92, 97, 126, 160, 171, 222, 223, 228, 240, 243, 245, 247, 251, 276
 - representation singularities, 28, 50
- Posterior distribution, 6, 162–164, 174, 182, 184, 185, 187, 191, 194, 236
- Prediction–correction, 6, 175, 180, 232
- Predictive prior, 44, 51, 168, 182, 186, 187, 191, 192, 195
- Preprocessing, 5, 8, 10, 55, 56, 94, 95, 104, 112, 114, 116, 120, 122, 123, 127, 133, 142, 148, 151, 199, 215, 229, 231, 240, 242, 250, 260, 261, 278
- Principal component analysis (PCA), 32, 36, 37, 39, 151
- Prior distribution, 8, 65, 66, 163, 183, 191, 194, 195, 233

- Process noise, 44, 50–52, 186, 233
 - covariance, 45, 46, 51, 172
 - gain, 54
- Procrustes analysis, 257
- Radial distortion, 16, 17, 21, 25, 26
- Random sample consensus (RANSAC), 169, 170, 171, 212
- Regularization, 105, 106, 145, 167, 168
- Rendering, 58, 94, 133, 151, 215
 - equation, 35
 - image-based, 34
 - off-screen, 57, 58
 - pipeline, 56
 - silhouette, 117, 118, 120
- Re-projection, 31, 66, 67, 97, 132, 133, 148, 151, 154, 160, 172, 200, 204, 215, 252
- Resectioning, 20
- Retinal
 - cone cells, 80, 81
 - plane, 13, 14
- Riccati equation, 46
- Sampling
 - importance–resampling (SIR), 6, 74, 126, 163, 166, 180–183, 185, 186, 190, 216, 230, 234–237, 239–241, 244, 247
 - model features, 5, 8, 10, 13, 41, 94, 95, 104, 112, 120, 122, 124, 127, 133, 142, 148, 151, 215
 - time interval, 44–46, 48, 51
- Scale-invariant
 - features descriptor, 138, 140, 212, 213
 - features detection, 70, 132, 135, 212
 - features matching, 212
 - features transform (SIFT), 133, 211, 212
 - moments, 100
- Scale space
 - image representation, 133, 134, 136
 - of curvature (CSS), 102, 103
- Segmentation
 - color, 56, 57, 68, 69, 75, 77, 80, 84, 89, 91, 94, 97, 199, 227, 241–243
 - foreground, 62, 75, 94, 95, 97, 141, 199, 241–243
 - level sets, 109, 112
 - motion, 56, 75, 77, 97, 141, 199, 242
- Shading, 35, 36, 129, 143, 148, 149, 150, 151, 158, 161
- Shape
 - circularity, 98
 - compactness, 98
 - convexity, 99
 - descriptor, 98, 99, 102, 198
 - eccentricity, 98
 - ellipticity, 99
 - model, 31, 33, 42, 221, 226
 - orientation, 98
 - perimeter, 98
 - rectangularity, 99
 - skeleton model, 30, 187, 224, 245, 247, 248, 250
- Sigma points, 164, 176, 177, 179
- Silhouette
 - back-projection, 97
 - descriptor, 98
 - detection, 96, 102
 - model, 58, 78, 97, 122
 - motion, 141
 - rendering, 117
 - sampling, 58
- Similarity
 - geometric, 99
 - photometric, 150, 151, 158
 - transform, 19, 100, 103, 129, 198, 205–208, 241, 245, 255, 256, 272, 273, 277
- Simplex optimization (Nelder–Mead), 167
- Singular value decomposition (SVD), 21, 24, 38, 201, 252, 254, 272, 273, 275, 276, 278, 279
- State
 - incremental, 50
 - space, 68, 71, 116, 148, 163, 164, 165, 173, 180, 184, 185, 187, 192, 193, 197, 198, 204, 209, 245
 - transition matrix, 44, 45, 47
- Steepest-descent images, 154, 156–158
- Sum of squared differences (SSD), 257, 130, 150, 158, 160, 169, 200
- Survival
 - diagnostics, 6, 184, 185, 188, 189
 - rate, 184, 185, 189

- Tangent space, 28, 29, 253
- Target-oriented tracking, 6, 7, 11
- Temporal motion history image (tMHI), 140–142
- Texture
 - edges, 113
 - matching, 5, 92, 97, 109, 113, 147, 151
 - memory, 58
 - model (or template), 3, 6, 10, 12, 34, 36, 58, 62, 78, 80, 126, 133, 147, 148, 151, 158, 213, 226
- Track
 - initiation, 7
 - loss detection, 165, 240
 - maintainance, 7, 8
- Tracking pipeline, 8, 12, 13, 55, 162, 166, 214, 216, 229, 230, 232, 240
- Training
 - active shape/appearance model, 37–39, 151
 - AdaBoost classifier, 202, 204
 - autoregressive dynamics, 51, 52
 - background model, 94, 95
- Triangulation, 61, 199–202, 204
- Tukey M-estimator, 169
- Twist vector, 29
- Two-phase segmentation, 109, 111
- Unscented
 - information filter, 178
 - Kalman filter, 51, 164, 166, 174, 176, 178, 216
 - transform, 164, 176, 180
- Update
 - online features, 9, 94, 95, 104, 112, 120, 122, 127, 133, 142, 148, 151
 - pose (*see also* Pose), 28, 29, 30, 41, 50, 51, 168, 245, 253
 - state estimate, 2, 6, 9, 44, 175, 176
- Upgrade
 - geometric transform, 152, 247, 257, 277, 278
 - measurement level, 75
- Validation gate, 5, 65, 66, 117, 122, 237, 238
- Variational
 - image segmentation, 104, 107, 108, 111
 - optical flow, 146
- Viewing volume, 209, 210
- Visual modality, 2, 5, 9, 34, 55, 56, 70, 71, 78, 79, 95, 114, 163, 197, 198, 214, 215, 230, 240, 242, 245
- Visual processing tree, 55, 68, 77, 162, 184, 216, 234, 237, 238, 242
- Voting, 74, 120, 204, 206
- Warp, 17, 39, 40, 58–61, 84, 129, 133, 153–157, 215, 223, 229, 232
- Wavelength, 35, 36, 80–82
- Weak classifier, 202, 204
- Weighted
 - average, 9, 74, 77, 146, 165, 235, 236
 - histogram, 93, 137
 - least-squares estimation, 122, 127, 153, 167, 168, 259
 - mean shift, 93, 208, 209
 - mixture of Gaussians, 88
 - resampling, 186, 187
 - state hypotheses, 93, 164, 165, 176, 183, 186, 188, 189, 191, 209, 235
 - weak classifier, 203
- World reference system, 13, 18, 22, 27, 31, 33, 40, 128, 222, 248
- Zooming, 18, 19



Figure 4.2 RGB color space decomposition.

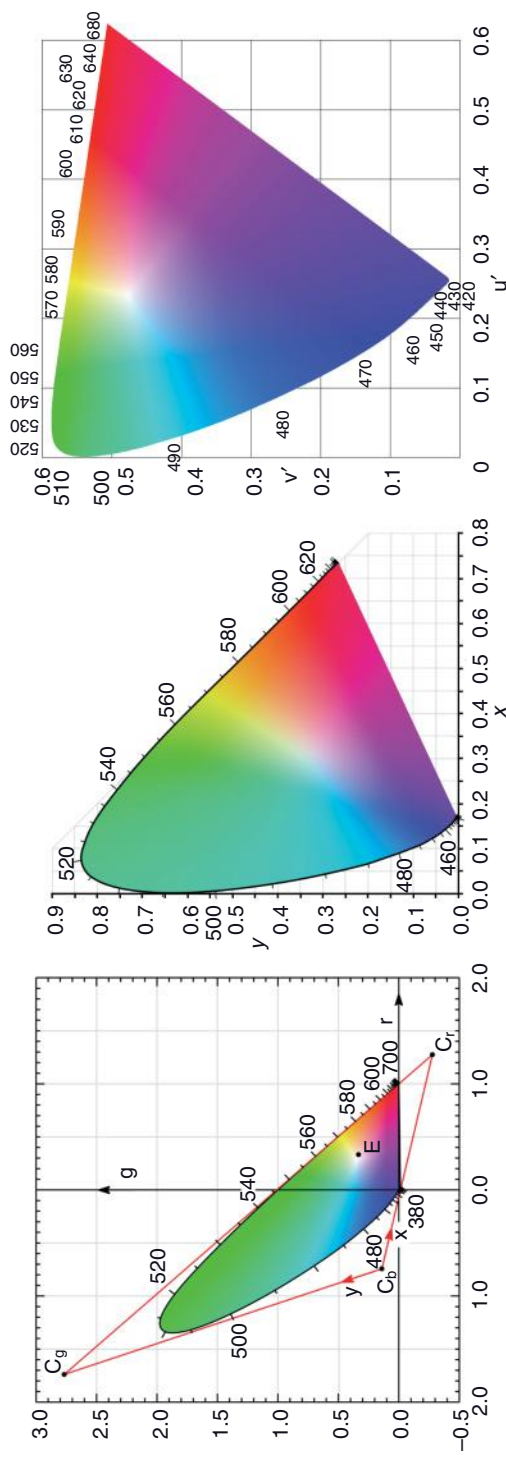


Figure 4.4 *Left:* CIE- rg chromaticity diagram; *middle:* CIE- xy diagram, obtained after normalizing the *gamut* (left triangle) with a linear mapping (from [6]); *right:* the $(u'v')$ chromatic space attempts to obtain a perceptually uniform color distribution.