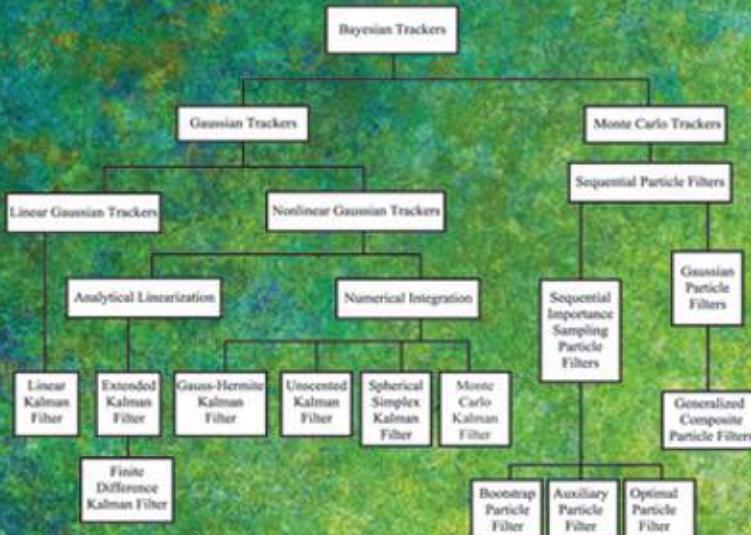


Bayesian Estimation and Tracking

A Practical Guide



Anton J. Haug

BAYESIAN ESTIMATION AND TRACKING

BAYESIAN ESTIMATION AND TRACKING

A Practical Guide

ANTON J. HAUG



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2012 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:

Haug, Anton J., 1941-

Bayesian estimation and tracking : a practical guide / Anton J. Haug.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-470-62170-7 (hardback)

1. Bayesian statistical decision theory. 2. Automatic tracking—Mathematics. 3. Estimation theory. I. Title.

QA279.5.H38 2012

519.5'42—dc23

2011044308

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

DEDICATION

To my wife, who inspires us all to achieve at the highest limit of our abilities.

To my children, whose achievements make me proud to be their father.

To my grandchildren, who will always keep me young.

And in memory of my parents, Rose and David, whose love was without bound.

CONTENTS

PREFACE	xv
ACKNOWLEDGMENTS	xvii
LIST OF FIGURES	xix
LIST OF TABLES	xxv

PART I PRELIMINARIES

1 Introduction	3
1.1 Bayesian Inference,	4
1.2 Bayesian Hierarchy of Estimation Methods,	5
1.3 Scope of This Text,	6
1.3.1 Objective,	6
1.3.2 Chapter Overview and Prerequisites,	6
1.4 Modeling and Simulation with MATLAB [®] ,	8
References,	9
2 Preliminary Mathematical Concepts	11
2.1 A Very Brief Overview of Matrix Linear Algebra,	11
2.1.1 Vector and Matrix Conventions and Notation,	11
2.1.2 Sums and Products,	12
2.1.3 Matrix Inversion,	13
2.1.4 Block Matrix Inversion,	14
2.1.5 Matrix Square Root,	15

2.2	Vector Point Generators,	16
2.3	Approximating Nonlinear Multidimensional Functions with Multidimensional Arguments,	19
2.3.1	Approximating Scalar Nonlinear Functions,	19
2.3.2	Approximating Multidimensional Nonlinear Functions,	23
2.4	Overview of Multivariate Statistics,	29
2.4.1	General Definitions,	29
2.4.2	The Gaussian Density,	32
	References,	40
3	General Concepts of Bayesian Estimation	42
3.1	Bayesian Estimation,	43
3.2	Point Estimators,	43
3.3	Introduction to Recursive Bayesian Filtering of Probability Density Functions,	46
3.4	Introduction to Recursive Bayesian Estimation of the State Mean and Covariance,	49
3.4.1	State Vector Prediction,	50
3.4.2	State Vector Update,	51
3.5	Discussion of General Estimation Methods,	55
	References,	55
4	Case Studies: Preliminary Discussions	56
4.1	The Overall Simulation/Estimation/Evaluation Process,	57
4.2	A Scenario Simulator for Tracking a Constant Velocity Target Through a DIFAR Buoy Field,	58
4.2.1	Ship Dynamics Model,	58
4.2.2	Multiple Buoy Observation Model,	59
4.2.3	Scenario Specifics,	59
4.3	DIFAR Buoy Signal Processing,	62
4.4	The DIFAR Likelihood Function,	67
	References,	69
PART II THE GAUSSIAN ASSUMPTION: A FAMILY OF KALMAN FILTER ESTIMATORS		
5	The Gaussian Noise Case: Multidimensional Integration of Gaussian-Weighted Distributions	73
5.1	Summary of Important Results From Chapter 3,	74
5.2	Derivation of the Kalman Filter Correction (Update) Equations <i>Revisited</i> ,	76
5.3	The General Bayesian Point Prediction Integrals for Gaussian Densities,	78

5.3.1	Refining the Process Through an Affine Transformation,	80
5.3.2	General Methodology for Solving Gaussian-Weighted Integrals,	82
References,	85	
6	The Linear Class of Kalman Filters	86
6.1	Linear Dynamic Models,	86
6.2	Linear Observation Models,	87
6.3	The Linear Kalman Filter,	88
6.4	Application of the LKF to DIFAR Buoy Bearing Estimation,	88
References,	92	
7	The Analytical Linearization Class of Kalman Filters: The Extended Kalman Filter	93
7.1	One-Dimensional Consideration,	93
7.1.1	One-Dimensional State Prediction,	94
7.1.2	One-Dimensional State Estimation Error Variance Prediction,	95
7.1.3	One-Dimensional Observation Prediction Equations,	96
7.1.4	Transformation of One-Dimensional Prediction Equations,	96
7.1.5	The One-Dimensional Linearized EKF Process,	98
7.2	Multidimensional Consideration,	98
7.2.1	The State Prediction Equation,	99
7.2.2	The State Covariance Prediction Equation,	100
7.2.3	Observation Prediction Equations,	102
7.2.4	Transformation of Multidimensional Prediction Equations,	103
7.2.5	The Linearized Multidimensional Extended Kalman Filter Process,	105
7.2.6	Second-Order Extended Kalman Filter,	105
7.3	An Alternate Derivation of the Multidimensional Covariance Prediction Equations,	107
7.4	Application of the EKF to the DIFAR Ship Tracking Case Study,	108
7.4.1	The Ship Motion Dynamics Model,	108
7.4.2	The DIFAR Buoy Field Observation Model,	109
7.4.3	Initialization for All Filters of the Kalman Filter Class,	111
7.4.4	Choosing a Value for the Acceleration Noise,	112
7.4.5	The EKF Tracking Filter Results,	112
References,	114	
8	The Sigma Point Class: The Finite Difference Kalman Filter	115
8.1	One-Dimensional Finite Difference Kalman Filter,	116
8.1.1	One-Dimensional Finite Difference State Prediction,	116

8.1.2	One-Dimensional Finite Difference State Variance Prediction,	117
8.1.3	One-Dimensional Finite Difference Observation Prediction Equations,	118
8.1.4	The One-Dimensional Finite Difference Kalman Filter Process,	118
8.1.5	Simplified One-Dimensional Finite Difference Prediction Equations,	118
8.2	Multidimensional Finite Difference Kalman Filters,	120
8.2.1	Multidimensional Finite Difference State Prediction,	120
8.2.2	Multidimensional Finite Difference State Covariance Prediction,	123
8.2.3	Multidimensional Finite Difference Observation Prediction Equations,	124
8.2.4	The Multidimensional Finite Difference Kalman Filter Process,	125
8.3	An Alternate Derivation of the Multidimensional Finite Difference Covariance Prediction Equations,	125
	References,	127
9	The Sigma Point Class: The Unscented Kalman Filter	128
9.1	Introduction to Monomial Cubature Integration Rules,	128
9.2	The Unscented Kalman Filter,	130
9.2.1	Background,	130
9.2.2	The UKF Developed,	131
9.2.3	The UKF State Vector Prediction Equation,	134
9.2.4	The UKF State Vector Covariance Prediction Equation,	134
9.2.5	The UKF Observation Prediction Equations,	135
9.2.6	The Unscented Kalman Filter Process,	135
9.2.7	An Alternate Version of the Unscented Kalman Filter,	135
9.3	Application of the UKF to the DIFAR Ship Tracking Case Study,	137
	References,	138
10	The Sigma Point Class: The Spherical Simplex Kalman Filter	140
10.1	One-Dimensional Spherical Simplex Sigma Points,	141
10.2	Two-Dimensional Spherical Simplex Sigma Points,	142
10.3	Higher Dimensional Spherical Simplex Sigma Points,	144
10.4	The Spherical Simplex Kalman Filter,	144
10.5	The Spherical Simplex Kalman Filter Process,	145
10.6	Application of the SSKF to the DIFAR Ship Tracking Case Study,	146
	Reference,	147

11 The Sigma Point Class: The Gauss–Hermite Kalman Filter	148
11.1 One-Dimensional Gauss–Hermite Quadrature, 149	
11.2 One-Dimensional Gauss–Hermite Kalman Filter, 153	
11.3 Multidimensional Gauss–Hermite Kalman Filter, 155	
11.4 Sparse Grid Approximation for High Dimension/High Polynomial Order, 160	
11.5 Application of the GHKF to the DIFAR Ship Tracking Case Study, 163	
References, 163	
12 The Monte Carlo Kalman Filter	164
12.1 The Monte Carlo Kalman Filter, 167	
Reference, 167	
13 Summary of Gaussian Kalman Filters	168
13.1 Analytical Kalman Filters, 168	
13.2 Sigma Point Kalman Filters, 170	
13.3 A More Practical Approach to Utilizing the Family of Kalman Filters, 174	
References, 175	
14 Performance Measures for the Family of Kalman Filters	176
14.1 Error Ellipses, 176	
14.1.1 The Canonical Ellipse, 177	
14.1.2 Determining the Eigenvalues of P, 178	
14.1.3 Determining the Error Ellipse Rotation Angle, 179	
14.1.4 Determination of the Containment Area, 180	
14.1.5 Parametric Plotting of Error Ellipse, 181	
14.1.6 Error Ellipse Example, 182	
14.2 Root Mean Squared Errors, 182	
14.3 Divergent Tracks, 183	
14.4 Cramer–Rao Lower Bound, 184	
14.4.1 The One-Dimensional Case, 184	
14.4.2 The Multidimensional Case, 186	
14.4.3 A Recursive Approach to the CRLB, 186	
14.4.4 The Cramer–Rao Lower Bound for Gaussian Additive Noise, 190	
14.4.5 The Gaussian Cramer–Rao Lower Bound with Zero Process Noise, 191	
14.4.6 The Gaussian Cramer–Rao Lower Bound with Linear Models, 191	

14.5 Performance of Kalman Class DIFAR Track Estimators,	192
References,	198

PART III MONTE CARLO METHODS

15 Introduction to Monte Carlo Methods	201
---	------------

15.1 Approximating a Density From a Set of Monte Carlo Samples,	202
15.1.1 Generating Samples from a Two-Dimensional Gaussian Mixture Density,	202
15.1.2 Approximating a Density by Its Multidimensional Histogram,	202
15.1.3 Kernel Density Approximation,	204
15.2 General Concepts Importance Sampling,	210
15.3 Summary,	215
References,	216

16 Sequential Importance Sampling Particle Filters	218
---	------------

16.1 General Concept of Sequential Importance Sampling,	218
16.2 Resampling and Regularization (Move) for SIS Particle Filters,	222
16.2.1 The Inverse Transform Method,	222
16.2.2 SIS Particle Filter with Resampling,	226
16.2.3 Regularization,	227
16.3 The Bootstrap Particle Filter,	230
16.3.1 Application of the BPF to DIFAR Buoy Tracking,	231
16.4 The Optimal SIS Particle Filter,	233
16.4.1 Gaussian Optimal SIS Particle Filter,	235
16.4.2 Locally Linearized Gaussian Optimal SIS Particle Filter,	236
16.5 The SIS Auxiliary Particle Filter,	238
16.5.1 Application of the APP to DIFAR Buoy Tracking,	242
16.6 Approximations to the SIS Auxiliary Particle Filter,	243
16.6.1 The Extended Kalman Particle Filter,	243
16.6.2 The Unscented Particle Filter,	243
16.7 Reducing the Computational Load Through Rao-Blackwellization,	245
References,	245

17 The Generalized Monte Carlo Particle Filter	247
---	------------

17.1 The Gaussian Particle Filter,	248
17.2 The Combination Particle Filter,	250
17.2.1 Application of the CPF–UKF to DIFAR Buoy Tracking,	252
17.3 Performance Comparison of All DIFAR Tracking Filters,	253
References,	255

PART IV ADDITIONAL CASE STUDIES

18 A Spherical Constant Velocity Model for Target Tracking in Three Dimensions	259
18.1 Tracking a Target in Cartesian Coordinates, 261	
18.1.1 Object Dynamic Motion Model, 262	
18.1.2 Sensor Data Model, 263	
18.1.3 Gaussian Tracking Algorithms for a Cartesian State Vector, 264	
18.2 Tracking a Target in Spherical Coordinates, 265	
18.2.1 State Vector Position and Velocity Components in Spherical Coordinates, 266	
18.2.2 Spherical State Vector Dynamic Equation, 267	
18.2.3 Observation Equations with a Spherical State Vector, 270	
18.2.4 Gaussian Tracking Algorithms for a Spherical State Vector, 270	
18.3 Implementation of Cartesian and Spherical Tracking Filters, 273	
18.3.1 Setting Values for q , 273	
18.3.2 Simulating Radar Observation Data, 274	
18.3.3 Filter Initialization, 276	
18.4 Performance Comparison for Various Estimation Methods, 278	
18.4.1 Characteristics of the Trajectories Used for Performance Analysis, 278	
18.4.2 Filter Performance Comparisons, 282	
18.5 Some Observations and Future Considerations, 293	
APPENDIX 18.A Three-Dimensional Constant Turn Rate Kinematics, 294	
18.A.1 General Velocity Components for Constant Turn Rate Motion, 294	
18.A.2 General Position Components for Constant Turn Rate Motion, 297	
18.A.3 Combined Trajectory Transition Equation, 299	
18.A.4 Turn Rate Setting Based on a Desired Turn Acceleration, 299	
APPENDIX 18.B Three-Dimensional Coordinate Transformations, 301	
18.B.1 Cartesian-to-Spherical Transformation, 302	
18.B.2 Spherical-to-Cartesian Transformation, 305	
References, 306	
19 Tracking a Falling Rigid Body Using Photogrammetry	308
19.1 Introduction, 308	
19.2 The Process (Dynamic) Model for Rigid Body Motion, 311	
19.2.1 Dynamic Transition of the Translational Motion of a Rigid Body, 311	
19.2.2 Dynamic Transition of the Rotational Motion of a Rigid Body, 313	
19.2.3 Combined Dynamic Process Model, 316	
19.2.4 The Dynamic Process Noise Models, 317	

19.3 Components of the Observation Model,	318
19.4 Estimation Methods,	321
19.4.1 A Nonlinear Least Squares Estimation Method,	321
19.4.2 An Unscented Kalman Filter Method,	323
19.4.3 Estimation Using the Unscented Combination Particle Filter,	325
19.4.4 Initializing the Estimator,	326
19.5 The Generation of Synthetic Data,	328
19.5.1 Synthetic Rigid Body Feature Points,	328
19.5.2 Synthetic Trajectory,	328
19.5.3 Synthetic Cameras,	333
19.5.4 Synthetic Measurements,	333
19.6 Performance Comparison Analysis,	334
19.6.1 Filter Performance Comparison Methodology,	335
19.6.2 Filter Comparison Results,	338
19.6.3 Conclusions and Future Considerations,	341
APPENDIX 19.A Quaternions, Axis-Angle Vectors, and Rotations,	342
19.A.1 Conversions Between Rotation Representations,	342
19.A.2 Representation of Orientation and Rotation,	343
19.A.3 Point Rotations and Frame Rotations,	344
References,	345
20 Sensor Fusion Using Photogrammetric and Inertial Measurements	346
20.1 Introduction,	346
20.2 The Process (Dynamic) Model for Rigid Body Motion,	347
20.3 The Sensor Fusion Observational Model,	348
20.3.1 The Inertial Measurement Unit Component of the Observation Model,	348
20.3.2 The Photogrammetric Component of the Observation Model,	350
20.3.3 The Combined Sensor Fusion Observation Model,	351
20.4 The Generation of Synthetic Data,	352
20.4.1 Synthetic Trajectory,	352
20.4.2 Synthetic Cameras,	352
20.4.3 Synthetic Measurements,	352
20.5 Estimation Methods,	354
20.5.1 Initial Value Problem Solver for IMU Data,	354
20.6 Performance Comparison Analysis,	357
20.6.1 Filter Performance Comparison Methodology,	359
20.6.2 Filter Comparison Results,	360
20.7 Conclusions,	361
20.8 Future Work,	362
References,	364
Index	367

PREFACE

This book presents a complete development of Bayesian estimation filters from first principles. We consider both linear and nonlinear dynamic systems driven by Gaussian or non-Gaussian noises. It is assumed that the dynamic systems are continuous because the observations related to those systems occur at discrete times only discrete filters are discussed. The primary goal is to present a comprehensive overview of most of the Bayesian estimation methods developed over the past 60 years in a unified approach that shows how each arises from the basic ideas underlying the Bayesian paradigms related to conditional densities.

The prerequisites for understanding the material presented in this book include a basic understanding of linear algebra, Bayesian probability theory, and numerical methods for finite differences and interpolation. Chapter 2 includes a review of all of these topics and is needed for an understanding of the remaining material in the book.

Many of the topics covered in this book grew out of a one semester course taught in the graduate mathematics department at the University of Maryland, College Park. The main goal of that course was to have the students develop their own Matlab® toolbox of target tracking methods. Several very specific tracking problems were presented to the students, and all homework problems consisted of coding each specific tracking (estimation) method into one or more Matlab® subroutines. In general, the subroutines the students developed were stand-alone and could be applied to any of a variety of tracking problems without much difficulty. Since the homework problems were dependent on our specific tracking problem (bearings-only tracking), I decided that this book would not contain any problem sets, allowing anyone using this book in a course to tailor their homework to a tracking problem of their choice. In addition, this book contains four fairly complicated case studies that contain enough material that any instructor can use one of them as the basis of their coding homework problems. The first case study is used as an example throughout most of Parts II and III of this

book while Part IV consists of the remaining three case studies with a separate chapter devoted to each.

This book has two emphases: First, detailed derivations from first principles of each estimation method (tracking filters) are emphasized through numerous tables and figures, very detailed step-by-step instructions for each method that will make coding of the tracking filter simple and easy to understand are also emphasized.

It is shown that recursive Bayesian estimation can be developed as the solution to a series of conditional density-weighted integrals of a transition or transformation function. The transition function is one that transitions a dynamic state vector from one time step to the next while the transformation function is one that transforms a state vector into an observation vector. There are a variety of numerical methods for solving these integrals, and each leads to a different estimation method. Each chapter in Parts II and III of this book considers one or more numerical approximations to solving these integrals leading to the class of Kalman filter methods for Gaussian-weighted integrals in Part II and the class of particle filters for density-weighted integrals with unknown densities in Part III.

This book is an outgrowth of many years of research in the field and it is hoped that it will be a significant contribution to the Bayesian estimation and tracking literature. I also hope that it will open up the field in new directions based on the many comments made about how these methods can be enhanced and applied in new ways and to new problems.

ANTON J. HAUG

ACKNOWLEDGMENTS

The author would like to acknowledge the many people who have contributed over the years to his knowledge base on Bayesian estimation and tracking through stimulating conversations.

Thanks are due to my colleagues from my previous position at MITRE, including G. Jacyna, D. Collella, and C. Christou.

Thanks are also due to my colleagues at the Johns Hopkins University Applied Physics Laboratory (JHUAPL), including L. Williams for her inspiring help on the case study in Chapter 18 and for general discussions on the material in the book; C. Davis for discussions on aspects of target tracking and for proofreading parts of the manuscript; and W. Martin for assessing the readability of the entire manuscript. Special thanks go to B. Beltran for the significant contribution of a Matlab® subroutine that computes the Gauss–Hermite sigma points and weights for any state vector dimension. I would also like to thank the JHUAPL Janney Publication Program committee for a one man-month work grant that allowed me to finish the manuscript in a timely fashion, and M. Whisnant for reviewing and commenting on the manuscript before granting a public release.

Finally, I would like to thank John Wiley & Sons for the opportunity to offer this book to the estimation and tracking community. I would also like to thank the anonymous reviewers of my initial book proposal submission for their many very helpful comments. I especially appreciate the role of S. Steitz-Filler, the Mathematics and Statistics Editor at John Wiley & Sons, for her patience over the four years that it took to write this book. She and her colleagues provided valuable advice, support, and technical assistance in transforming my initial manuscript submission into a book that I can be proud of.

LIST OF FIGURES

1.1	Hierarchy of Bayesian Estimation Tracker Filters.	5
2.1	Example of Two-Dimensional Cartesian Grid of Axial Vector Points.	17
2.2	Example of Three-Dimensional Cartesian Grid of Axial Vector Points.	17
2.3	Two-Dimensional Points for Multidimensional Sterling's Approximation.	28
2.4	Simplex Figures in up to Three Dimensions.	29
2.5	One-Dimensional Gaussian pdf Example.	33
2.6	Example of a Two-Dimensional Gaussian PDF.	34
2.7	An Example of a One-Dimensional Gaussian CDF.	35
2.8	An Example of a Two-Dimensional Gaussian CDF.	36
2.9	Effects of an Affine Transformation on a Gaussian Probability Density Function.	38
3.1	Depiction of One Step in the Recursive Bayesian Posterior Density Estimation Procedure.	49
3.2	General Block Diagram for Recursive Point Estimation Process.	54
4.1	Methodology for Development and Evaluation of Tracking Algorithms.	57

4.2	DIFAR Buoy Geometry.	60
4.3	Ships Track with DIFAR Buoy Field and the Associated Bearings for Two Buoys.	60
4.4	DIFAR Sensor Signal Processing.	62
4.5	The Spread of Bearing Observations from a DIFAR Buoy from 100 Monte Carlo Runs for Four SNRs.	67
4.6	DIFAR Likelihood Density for a BT of 10 and SNR from -10 to $+10$ dB.	69
5.1	Block Diagram of the Process Flow for the Bayesian Estimator with Gaussian Probability Densities.	80
6.1	Block Diagram of the LKF Process for a Single DIFAR Buoy.	91
6.2	Comparison of the LKF Bearing Estimates with the True Bearing for One DIFAR Buoy.	91
7.1	The Geometry Used for Initialization.	111
7.2	A Comparison of the Estimated Track of a Ship Transiting the Buoy Field with the True Track.	113
7.3	Comparison of the EKF Tracker Outputs for Six SNRs.	113
10.1	Depiction of a Two-Dimensional Simplex with Four Vector Integration Points.	142
13.1	Graphical Presentation of the Linear Kalman Filter Process Flow.	169
13.2	A Graphical Representation of the EKF Process Flow.	169
13.3	Process Flow for General Sigma Point Kalman Filter.	170
14.1	Error Ellipse Rotation Angle.	179
14.2	An Example of Error Ellipses Overplotted on the DIFAR UKF Track Output.	182
14.3	Comparison of the RMS Errors for Five Different Track Estimation Algorithms with the Signal SNR at 20 dB.	193
14.4	Comparison of the RMS Errors for Five Different Track Estimation Algorithms with the Signal SNR at 15 dB.	194
14.5	Comparison of the RMS Errors for Five Different Track Estimation Algorithms with the Signal SNR at 10 dB.	195
14.6	Comparison of the RMS Errors for Five Different Track Estimation Algorithms with the Signal SNR at 5 dB.	196

14.7	Comparison of the RMS Errors for Five Different Track Estimation Algorithms with the Signal SNR at 0 dB.	197
15.1	Samples Drawn from a Two-Dimensional Gaussian Mixture Density.	203
15.2	Two-Dimensional Histogram Based on Samples from a Gaussian Mixture Distribution.	204
15.3	Gaussian Kernel Density Estimate for Sparse Sample Data.	207
15.4	Visualization of the Gaussian Kernel Estimate of a Density Generated from Random Samples Drawn for that Density.	210
15.5	Example of the Creation of $w(x)$	212
16.1	Inverse of the Gaussian Cumulative Distribution.	223
16.2	Comparison of a Gaussian CDF with Its Inverse.	224
16.3	Principle of Resampling.	228
16.4	Target Track Generated from a DIFAR Buoy Field Using a Bootstrap Particle Filter.	232
16.5	A Comparison of Track Outputs at Six Different SNRs for the BPF tracker.	233
16.6	Track Estimation Results for the Auxiliary Particle Filter Applied to the DIFAR Tracking Case Study.	242
17.1	Process Flow Diagram for the Gaussian Particle Filter.	249
17.2	Combination Particle Filter that Uses an EKF as an Importance Density.	250
17.3	Combination Particle Filter that Uses a Sigma Point Kalman Filter as an Importance Density.	251
17.4	Monte Carlo Track Plots for the Sigma Point Gaussian Particle Filter for Six SNRs.	252
17.5	Comparison of the DIFAR Case-Study Root Mean Squared Position Errors for a Signal SNR of 20 dB.	253
17.6	Comparison of the DIFAR Case-Study Root Mean Squared Position Errors for a Signal SNR of 15 dB.	254
17.7	Comparison of the DIFAR Case-Study Root Mean Squared Position Errors for a Signal SNR of 10 dB.	255
18.1	A Simulated Radially Inbound Target Truth Track.	278
18.2	A Simulated Horizontally Looping Target Truth Track.	279

18.3	A Simulated Benchmark Target Truth Track.	279
18.4	Components of Both the Cartesian and Spherical Velocities for the Radially Inbound Trajectory.	280
18.5	Components of Both the Cartesian and Spherical Velocities for the Loop Trajectory.	281
18.6	Components of Both the Cartesian and Spherical Velocities for the Benchmark Trajectory.	281
18.7	Estimated Tracks for the Radially Inbound Target Trajectory for All Cartesian Tracking Methods.	283
18.8	Estimated Tracks for the Radially Inbound Target Trajectory for All Spherical Tracking Methods.	283
18.9	RMS Cartesian Position Errors of the Radially Inbound Target Trajectory for the Cartesian and Spherical Tracking Algorithms. . . .	284
18.10	RMS Spherical Position Errors of the Radially Inbound Target Trajectory for the Cartesian and Spherical Tracking Algorithms. . . .	284
18.11	Comparison of the Radially Inbound Trajectory RMS Cartesian Position Errors for Varying Values of q for Both Cartesian and Spherical EKF Filters.	285
18.12	Comparison of the Radially Inbound Trajectory RMS Spherical Position Errors for Varying Values of q for Both Cartesian and Spherical EKF Filters.	286
18.13	Estimated Tracks for the Looping Target Trajectory for All Cartesian Tracking Methods.	287
18.14	Estimated Tracks for the Looping Target Trajectory for All Spherical Tracking Methods.	287
18.15	RMS Cartesian and Spherical Position Errors of the Loop Target Trajectory for the Cartesian Tracking Algorithms.	288
18.16	RMS Cartesian and Spherical Position Errors of the Loop Target Trajectory for the Spherical Tracking Algorithms.	288
18.17	Comparison of RMS Cartesian Position Error Performance as a Function of q for the Loop Trajectory.	289
18.18	Comparison of RMS Spherical Position Error Performance as a Function of q for the Loop Trajectory.	289
18.19	Estimated Tracks for the Benchmark Target Trajectory for All Cartesian Tracking Methods.	290

18.20	Estimated Tracks for the Benchmark Target Trajectory for All Spherical Tracking Methods.	290
18.21	RMS Cartesian and Spherical Position Errors of the Benchmark Target Trajectory for the Cartesian Tracking Algorithms.	291
18.22	RMS Cartesian and Spherical Position Errors of the Benchmark Target Trajectory for the Spherical Tracking Algorithms.	291
18.23	Comparison of RMS Cartesian Position Error Performance as a Function of q for the Benchmark Trajectory.	292
18.24	Comparison of RMS Spherical Position Error Performance as a Function of q for the Benchmark Trajectory.	293
18.25	The East–North–Up Cartesian Coordinate System.	294
18.26	Depiction of a Constant Rate Turn in Both the Horizontal and Vertical Planes.	295
19.1	Photo of a US Navy F-18E Super Hornet Aircraft Preparing to Release Four Mk-62 Stores.	309
19.2	Close-up of a MK-62 Store Release as Viewed from a Camera Mounted Under the Aircraft's Tail.	310
19.3	Projection of a Feature Point onto a Camera's Image Plane.	319
19.4	Translational and Rotational Position Using a Second-Order Model with a UKF Filter.	334
19.5	Translational and Rotational Velocity Using a Second-Order Model with a UKF Filter.	335
19.6	Translational and Rotational Acceleration Using a Second-Order Model with a UKF Filter.	336
19.7	Lateral Position Estimates from Multiple Solvers Using Identical Inputs.	336
19.8	RMS Errors for Position and Orientation of All Tracking Filters Using Synthetic Data.	339
19.9	RMS Errors for Position and Orientation Velocities of All Tracking Filters Using Synthetic Data.	340
19.10	RMS Errors for Position and Orientation Accelerations of All Tracking Filters Using Synthetic Data.	341
20.1	A Free-Standing IMU with Transmitting Antenna Attached, and An IMU Mounted in the Forward Fuse Well of a 500 Pound Store.	348
20.2	Translational and Rotational Position Using the Sensor Fusion Estimator.	357

20.3	Translational and Rotational Velocity Using the Sensor Fusion Estimator.	358
20.4	Translational and Rotational Acceleration Using the Sensor Fusion Estimator.	359
20.5	Lateral Position Estimates from Multiple Estimation Filters Using Identical Measurement Inputs.	360
20.6	RMS Errors for Position and Orientation of All Tracking Filters Using Synthetic Data.	362
20.7	RMS Errors for Position and Orientation Velocities of All Tracking Filters Using Synthetic Data.	363
20.8	RMS Errors for Position and Orientation Accelerations of All Tracking Filters Using Synthetic Data.	364

LIST OF TABLES

4.1	Procedure for Generating a Vector of DIFAR Noisy Bearing Observations.	65
6.1	Linear Kalman Filter Process.	89
7.1	One-Dimensional Extended Kalman Filter Process.	98
7.2	Multidimensional Extended Kalman Filter Process.	106
8.1	One-Dimensional Finite Difference Kalman Filter Process.	119
8.2	Multidimensional Finite Difference Kalman Filter Process.	126
9.1	Multidimensional Unscented Kalman Filter Process.	136
9.2	Multidimensional Sigma Point Kalman Filter Process Applied to DIFAR Tracking.	137
10.1	Multidimensional Spherical Simplex Kalman Filter Process.	146
11.1	Multidimensional Gauss–Hermite Kalman Filter Process.	161
12.1	Multidimensional Spherical Simplex Kalman Filter Process.	166
13.1	Summary Data for Sigma Point Kalman Filters: Part 1—Form of c to Be Used for Sigma Points.	171
13.2	Summary Data for Sigma Point Kalman Filters: Part 2—Form of Sigma Point Weights.	171
13.3	Comparison of the Number of Integration Points Required for the Various Sigma Point Kalman Filters.	172

13.4	Hierarchy of Dynamic and Observation Models.	174
15.1	Common Univariate Kernel Functions of Order 2.	206
15.2	One-Dimensional Kernel Sample Generation.	208
15.3	Multidimensional Kernel Sample Generation.	210
16.1	General Sequential Importance Sampling Particle Filter.	221
16.2	Sequential Importance Sampling Particle Filter with Resampling.	227
16.3	Sequential Importance Sampling Particle Filter with Resampling and Regularization.	229
16.4	Bootstrap SIS Particle Filter with Resampling and Regularization.	231
16.5	Optimal SIS Particle Filter with Resampling and Regularization. . . .	237
16.6	Auxiliary Particle Filter Process Flow.	241
16.7	Unscented Particle Filter with Resampling and Regularization. . . .	244
18.1	Nominal Values for Initialization State Vector and Covariance Components.	276
18.2	Initial Characteristics of Three Simulated Scenarios.	280
19.1	Synthetic Data RMS Positional Error Summary.	338
19.2	Synthetic Data RMS Velocity Error Summary.	338
19.3	Synthetic Data RMS Acceleration Error Summary.	339
20.1	Synthetic Data RMS Positional Error Summary.	361
20.2	Synthetic Data RMS Velocity Error Summary.	361
20.3	Synthetic Data RMS Acceleration Error Summary.	361

PART I

PRELIMINARIES

1

INTRODUCTION

Estimation and tracking of dynamic systems has been the research focus of many a mathematician since the dawn of statistical mathematics. Many estimation methods have been developed over the past 50 years that allow statistical inference (estimation) for dynamic systems that are linear and Gaussian. In addition, at the cost of increased computational complexity, several methods have shown success in estimation when applied to nonlinear Gaussian systems. However, real-world dynamic systems, both linear and nonlinear, usually exhibit behavior that results in an excess of outliers, indicative of non-Gaussian behavior. The toolbox of standard Gaussian estimation methods have proven inadequate for these problems resulting in divergence of the estimation filters when applied to such real-world data.

With the advent of high-speed desktop computing, over the past decade the emphasis in mathematics has shifted to the study of dynamic systems that are non-Gaussian in nature. Much of the literature related to performing inference for non-Gaussian systems is highly mathematical in nature and is lacking in practical methodology that the average engineer can utilize without a lot of effort. In addition, several of the Gaussian methods related to estimation for nonlinear systems are presented ad hoc, without a cohesive derivation. Finally, there is a lack of continuity in the conceptual development to date between the Gaussian methods and their non-Gaussian counterparts.

In this book, we will endeavor to present a comprehensive study of the methods currently in use for statistical dynamic system estimation: linear and nonlinear, Gaussian

and non-Gaussian. Using a Bayesian framework, we will present a conceptually cohesive roadmap that starts at first principles and leads directly to derivations of many of the Gaussian estimation methods currently in use. We will then extend these concepts into the non-Gaussian estimation realm, where the theory leads directly to working Monte Carlo methods for estimation. Although the Bayesian approach leads to the estimation of statistical densities, in most cases we will develop point estimation methods that can be obtained through the evaluation of density-weighted integrals. Thus, this book is all about numerical methods for evaluating density-weighted integrals for both Gaussian and non-Gaussian densities.

For each estimation method that we discuss and derive, we present both pseudo-code and graphic block diagram that can be used as tools in developing a software-coded tracking toolbox. As an aid in understanding the methods presented, we also discuss what is required to develop simulations for several very specific real-world problems. These case-study problems will be addressed in great detail, with track estimation results presented for each. Since it is hard to compare tracking methods ad hoc, we also present multiple methods to evaluate the relative performance of the various tracking filters.

1.1 BAYESIAN INFERENCE

Inference methods consist of estimating the current values for a set of parameters based on a set of observations or measurements. The estimation procedure can follow one of two models. The first model assumes that the parameters to be estimated, usually unobservable, are nonrandom and constant during the observation window but the observations are noisy and thus have random components. The second model assumes that the parameters are random variables that have a prior probability and the observations are noisy as well. When the first model is used for parameter estimation, the procedure is called non-Bayesian or Fisher estimation [1]. Parameter estimation using the second model is called Bayesian estimation.

Bayesian estimation is conceptually very simple. It begins with some initial prior belief, such as the statement “See that ship. It is about 1000 yards from shore and is moving approximately Northeast at about 10 knots.” Notice that the initial belief statement includes an indication that our initial guess of the position and velocity of the ship are uncertain or random and based on some prior probability distribution. Based on one’s initial belief, one can then make the prediction “Since the ship appears to be moving at a constant velocity, it will be over there in about 10 minutes.” This statement includes a mental model of the ship motion dynamics as well as some additional uncertainty. Suppose now, that one has a small portable radar on hand. The radar can be used to measure (observe) the line-of-sight range and range rate of the ship to within some measure of uncertainty. Given the right mathematical model, one that links the observations to the Cartesian coordinates of the ships position and velocity, a current radar measurement can be used to update the predicted ships state (position and velocity).

The above paragraph contains the essence of recursive Bayesian estimation:

1. begin with some prior belief statement,
2. use the prior belief and a dynamic model to make a prediction,
3. update the prediction using a set of observations and an observation model to obtain a posterior belief, and
4. declare the posterior belief our new prior belief and return to 2.

This concept was first formalized in a paper by the Reverend Thomas Bayes, read to the Royal Statistical Society in 1763 by Richard Price several years after Bayes' death. An excellent review of the history and concepts associated with Bayesian statistical inference can be found in the paper by Stephen Brooks [2]. Brooks' paper also has some interesting examples that contrast the Bayesian method with the so-called "Frequentist" method for statistical inference. Since this book is devoted completely to Bayesian methods, we will not address the frequentist approach further and refer the interested reader to Brooks' paper.

1.2 BAYESIAN HIERARCHY OF ESTIMATION METHODS

As noted above, in this book we will present a cohesive derivation of a subset of modern tracking filters. Figure 1.1 shows the hierarchy of tracking filters that will

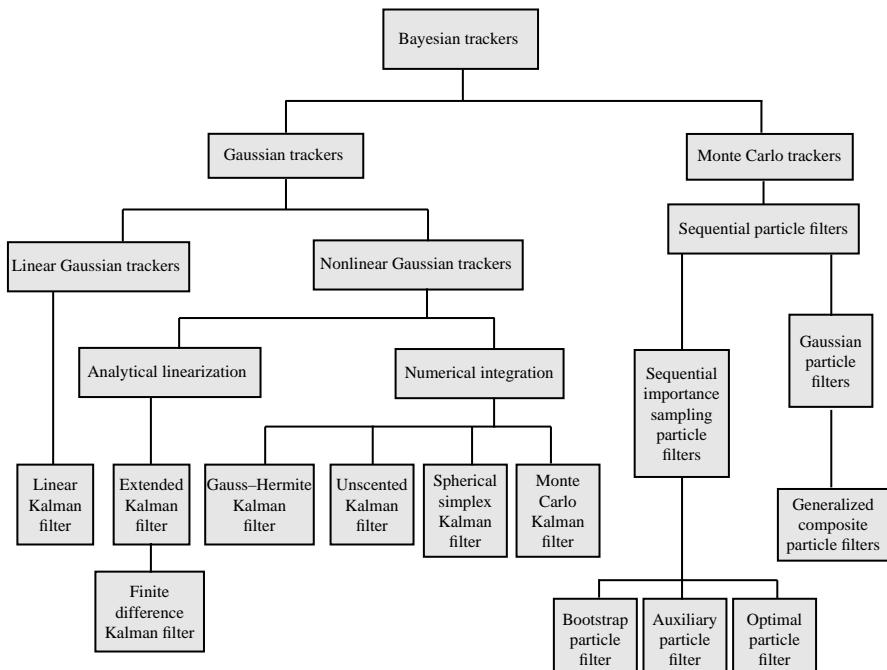


FIGURE 1.1 Hierarchy of Bayesian estimation tracker filters.

be addressed in this book. Along the left-hand side are all the Gaussian tracking filters and along the right-hand side are all of the Monte Carlo non-Gaussian filters. This figure will be our guide as we progress through our discussions on each tracking filter. We will use it to locate where we are in our developments. We may occasionally take a side trip into other interesting concepts, such as a discussion of performance measures, but for the most part we will stick to a systematic development from top to bottom and left to right. By the time we reach the bottom right, you the reader will have a comprehensive understanding of the interrelatedness of all of the Bayesian tracking filters.

1.3 SCOPE OF THIS TEXT

1.3.1 Objective

The objective of this book is to give the reader a firm understanding of Bayesian estimation methods and their interrelatedness. Starting with the first principles of Bayesian theory, we show how each tracking filter is derived from a slight modification to a previous filter. Such a development gives the reader a broader understanding of the hierarchy of Bayesian estimation and tracking. Following the discussions about each tracking filter, the filter is put into both pseudo-code and process flow block diagram form for ease in future recall and reference.

In his seminal book on filtering theory [3], originally published in 1970, Jazwinski stated that “The need for this book is twofold. First, although linear estimation theory is relatively well known, it is largely scattered in the journal literature and has not been collected in a single source. Second, available literature on the continuous nonlinear theory is quite esoteric and controversial, and thus inaccessible to engineers uninitiated in measure theory and stochastic differential equations.” A similar statement can be made about the current state of affairs in non-Gaussian Monte Carlo methods of estimation theory. Most of the published work is esoteric and inaccessible to engineers uninitiated in measure theory. The edited book of invited papers by Doucet et al. [4] is a prime example. This is an *excellent* book of invited papers, but is extremely esoteric in many of its stand-alone sections.

In this book, we will take Jazwinski’s approach and remove much of the esoteric measure theoretic-based mathematics that makes understanding difficult for the average engineer. Hopefully, we have not replaced it with equally esoteric alternative mathematics.

1.3.2 Chapter Overview and Prerequisites

This book is not an elementary book and is intended as a one semester graduate course or as a reference for anyone requiring or desiring a deeper understanding of estimation and tracking methods. Readers of this book should have a graduate level understanding of probability theory similar to that of the book by Papoulis [5]. The reader should also be familiar with matrix linear algebra and numerical methods

including finite differences. In an attempt to reduce the steep prerequisite requirements for the reader, we have included several review sections in the next chapter on some of these mathematical topics. Even though some readers may want to skip these sections, the material presented is integral to an understanding of what is developed in Parts II and III of this book.

Part I consists of this introduction followed by a chapter that presents an overview of some mathematical principles required for an understanding of the estimation methods that follow. The third chapter introduces the concepts of recursive Bayesian estimation for a dynamic system that can be modeled as a potentially unobservable discrete Markov process. The observations (measurements) are related to the system states through an observation model and the observations are considered to be discrete. Continuous estimation methods are generally not considered in this book. The last chapter of Part I is devoted to preliminary development of a case study that will be used as working examples throughout the book, the problem of tracking a ship through a distributed field of directional frequency analysis and recording (DIFAR) sonobuoys. Included for this case study will be demonstrations of methods for development of complete simulations of the system dynamics along with the generation of noisy observations.

Part II is devoted to the development and application of estimation methods for the Gaussian noise case. In Chapter 5, the general Bayesian estimation methods developed in Chapter 3 are rewritten in terms of Gaussian probability densities. Methods for specific Gaussian Kalman filters are derived and codified in Chapters 6 through 12, including the linear Kalman filter (LKF), extended Kalman filter (EKF), finite difference Kalman filter (FDKF), unscented Kalman filter (UKF), spherical simplex Kalman filter (SSKF), Gauss–Hermite Kalman filter (GHKF), and the Monte Carlo Kalman filter (MCKF). With the exception of the MCKF, four of latter five tracking filters can be lumped into the general category of sigma point Kalman filters where deterministic vector integration points are used in the evaluation of the Gaussian-weighted integrals needed to estimate the mean and covariance matrix of the state vector. In the MCKF, the continuous Gaussian distribution is replaced by a sampled distribution reducing the estimation integrals to sums leaving the nonlinear functions intact. It will be shown in Chapter 13 that the latter five Kalman filter methods can be summarized into a single estimation methodology requiring just a change in the number and location of the vector points used and their associated weights.

An important aspect of estimation, usually ignored in most books on estimation, is the quantification of performance measures associated with the estimation methods. In Chapter 14 this topic is addressed, with sections on methods for computing and plotting error ellipses based on the estimated covariance matrices for use in real-system environments, as well as methods for computing and plotting root mean squared (RMS) errors and their Cramer–Rao lower bounds (CRLB) for use in Monte Carlo simulation environments. The final section of this chapter is devoted to application of these estimation methods to the DIFAR buoy tracking case study and includes a comparison of performance results as a function of decreasing input signal-to-noise ratio (SNR).

Estimation methods for use primarily with non-Gaussian probability densities is the topic addressed in Part III. For the MCKF introduced in Chapter 12 of Part II, the Gaussian density is approximated by a set of discrete Monte Carlo samples, reducing the mean and covariance estimation integrals to weighted sums, usually referred to as sample mean and sample covariance, respectively. For Gaussian densities, the sample weight is always $1/N$, where N is the number of samples used. Non-Gaussian densities present two problems: first, it is usually very difficult to generate a set of Monte Carlo samples directly from the density. A second problem arises if the first or second moment does not exist for the density, with the Cauchy density as a prime example. To address the sampling problem, in Chapter 15 Monte Carlo methods are introduced and the concept of importance sampling developed that leads to estimation methods called particle filters, where the particles are the Monte Carlo sample points. Several problems arise when implementing these particle filters and potential enhancements are considered that correct for these problems. For importance sampling, weighting for each sample is calculated as the ratio of the non-Gaussian density to the importance density at the sample point. Under certain assumptions, the weights can be calculated recursively, giving rise to the sequential importance sampling (SIS) class of particle filters, the topic of Chapter 16. In Chapter 17, the case where the weights are recalculated every filter iteration step is addressed, leading to the Gaussian class of combination particle filters. Performance results for all of the particle filter track estimation methods applied to the DIFAR case study are presented as the conclusion of Chapter 17.

Several recently published books provide additional insight into the topics presented in this book. For Gaussian Kalman filters of Part II, books by Bar Shalom et al. [6] and Candy [7] are good companion books. For non-Gaussian filtering methods of Part III, books by Doucet et al. [4] and Ristic et al. [8] are excellent reference books.

1.4 MODELING AND SIMULATION WITH MATLAB[®]

It is important to the learning process that the reader be given concrete examples of application of estimation methods to a set of complex problems. This will be accomplished in this book through the use of simulations using MATLAB[®]. We present a set of four case studies that provide an increase in complexity from the first to the last. Each case study will include an outline of how to set up a simulation that models both the dynamics and observations of the system under study. We then show how to create a set of randomly generated observational data using a Monte Carlo methodology. This simulated observational data can then be used to exercise each tracking filter, producing sets of track data that can be compared across multiple track filters.

The first case study examines the problem of tracking a ship as it moves through a distributed field of DIFAR buoys. A DIFAR buoy uses the broadband noise signal radiated from the ship as an input and produces noisy observations of the bearing to the ship as an output. As we will show in Chapter 4, the probability density of the bearing estimates at the DIFAR buoy output is dependent on the SNR of the input

signal. The density will be Gaussian for high SNR but will transition to a uniform distribution as the SNR falls. The purpose of this case study will be to examine what happens to the filter tracking performance for each track estimation method as the observation noise transitions from Gaussian to non-Gaussian.

This DIFAR case study will be the primary tool used throughout this book to illustrate each track estimation filter in turn. In Chapter 4, we show how to set up a simulation of the DIFAR buoy processing so as to produce simulated SNR dependent observation sets. Using these sets of bearing observations, in subsequent chapters we exercise each tracking algorithm to produce Monte Carlo sets of track estimates, allowing us to see the impact of the Gaussian to non-Gaussian observation noise transition on each tracking method.

In Part IV of this book, we present three additional case studies that illustrate the use of many of the tracking filters developed in Parts II and III. In Chapter 18, we address the important problem of tracking a maneuvering object in three dimension space. In this chapter, we introduce a new approach that uses a constant spherical velocity model vice the more traditional constant Cartesian velocity model. We show how this spherical model shows improved performance for tracking a maneuvering object using most of the Gaussian tracking filters.

The third case study, found in Chapter 19, considers the rather complex problem of tracking the dynamics of a falling bomb through the use of video frames of multiple tracking points on both the plane dropping the bomb and the bomb itself. This is a particular example of a complex process called photogrammetry, in which the geometric and dynamic properties of an object are inferred from successive photographic image frames. Thus, this case study consists of a very complex nonlinear multidimensional observational process as well as a nonlinear multidimensional dynamic model. In addition, both the dynamic and observational models are of high dimension, a particularly taxing problem for tracking filters. This will illustrate the effects of the so-called “curse” of dimensionality, showing that it is computationally impractical to utilize all tracking filters.

The final case study, the topic of Chapter 20, improves on the use of photogrammetric methods in estimation by showing how a separate estimator can be used for fusing data from additional sensors, such as multiple cameras, translational accelerometers, and angular rate gyroscopes. When used independently, each data source has its unique strengths and weaknesses. When several different sensors are used jointly in an estimator, the resulting solution is usually more accurate and reliable. The resulting analysis shows that estimator aided sensor fusion can recover meaningful results from flight tests that would otherwise have been considered failures.

REFERENCES

1. Fisher R. *Statistical Methods and Scientific Inference*, Revised Edition. Macmillan Pub. Co.; 1973.
2. Brooks SP. Bayesian computation: a statistical revolution. *Phil. Trans. R. Soc. Lond. A* 2003;361:2681–2697.

3. Jazwinski AH. *Stochastic Processes and Filtering Theory*. Academic Press (1970), recently republished in paperback by Dover Publications; 2007.
4. Doucet A, de Freitas JFG, Gordon NJ, editors. *Sequential Monte Carlo Methods in Practice*. New York, NY: Springer-Verlag; 2001.
5. Papoulis A. *Probability, Random Variables, and Stochastic Processes*, 4th ed. McGraw-Hill; 2002.
6. Bar Shalom Y, Li XR, Kirubarajan T. *Estimation with Application to Tracking and Navigation: Theory, Algorithms and Software*. Wiley; 2001.
7. Candy JV. *Bayesian Signal Processing: Classical, Modern, and Particle Filtering Methods*. Hoboken, NJ: Wiley; 2009.
8. Ristic B, Arulampalam S, Gordon N. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Boston, MA: Artech House; 2004.

2

PRELIMINARY MATHEMATICAL CONCEPTS

In this chapter, we will present a host of mathematical concepts that are essential to an understanding of what follows in Parts II–IV. The mathematical developments presented here will be concise and without extensive or rigorous proofs. For most of the topics covered here, it will be assumed that the reader is familiar with the mathematical concepts from previous exposure. Although it is not recommended, this chapter can be skipped on first reading and used as a reference while reading the remainder of Part I and Parts II–IV. Additional mathematical concepts will be introduced as needed.

2.1 A VERY BRIEF OVERVIEW OF MATRIX LINEAR ALGEBRA

This section defines notational conventions and matrix operations used throughout the text and provides a brief summary of some matrix linear algebra concepts within the context of MATLAB®. An extremely good reference for the matrix operations and formulas presented in this section can be found online in Ref. [1].

2.1.1 Vector and Matrix Conventions and Notation

A scalar will usually be presented as lower case, a , a vector as lower case bold, \mathbf{a} , and a matrix as upper case bold, \mathbf{A} .

An n -vector is the $n \times 1$ column vector

$$\mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \quad (2.1)$$

By convention, all vectors will be column vectors. It follows immediately that one can write a column vector as the transpose of a row vector (especially useful to conserve space in a report or journal article)

$$\mathbf{a} = [a_1, \dots, a_n]^\top \quad (2.2)$$

with commas separating the elements of the vector. $[\cdot]^\top$ represent a vector or matrix transpose, which will be defined below.

An $n \times m$ matrix is a two-dimensional array of the form

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \quad (2.3)$$

The first dimension is the number of rows and the second is the number of columns. The ij th component of the matrix \mathbf{A} is a_{ij} .

The *transpose* of the matrix \mathbf{A} is designated as \mathbf{A}^\top and is defined by

$$\mathbf{A}^\top = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \quad (2.4)$$

The *trace* of an $n \times n$ square matrix \mathbf{A} is defined to be the sum of the elements on the main diagonal (the diagonal from the upper left to the lower right) of \mathbf{A} , that is,

$$\text{trace}\{\mathbf{A}\} = \sum_{i=1}^n a_{ii} \quad (2.5)$$

Equivalently, the trace of a matrix is the sum of its eigenvalues, making it an invariant with respect to a change of basis. This characterization can be used to define the trace for a linear operator in general.

We will occasionally use the notation $\mathbf{A} \in \mathbb{R}^{n \times m}$ or $\mathbf{a} \in \mathbb{R}^n$ to designate the dimension of a matrix or vector, respectively.

2.1.2 Sums and Products

Addition of matrices and multiplication by a scalar are defined as

$$\mathbf{C} = \alpha\mathbf{A} + \beta\mathbf{B} \quad (2.6)$$

where

$$c_{ij} = \alpha a_{ij} + \beta b_{ij}; \quad i = 1, \dots, n; \quad j = 1, \dots, m \quad (2.7)$$

The product of two matrices, for matrix $\mathbf{A} \in \mathbb{R}^{n \times k}$ and matrix $\mathbf{B} \in \mathbb{R}^{k \times m}$, is written as

$$\mathbf{C} = \mathbf{AB} \quad (2.8)$$

with the elements of the matrix $\mathbf{C} \in \mathbb{R}^{n \times m}$ given by

$$c_{ij} = \sum_{l=1}^k a_{il}b_{lj} \quad (2.9)$$

Note that for matrix multiplication, the number of columns of the matrix \mathbf{A} must match the number of rows of matrix \mathbf{B} .

The *Hadamard product* of two matrices of identical dimension is the element-by-element product matrix

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B} \quad (2.10)$$

or

$$c_{ij} = a_{ij}b_{ij} \quad (2.11)$$

Remark 2.1 Within MATLAB®, a full matrix multiplication is represented as $\mathbf{C} = \mathbf{A} * \mathbf{B}$, while the Hadamard product is represented as $\mathbf{C} = \mathbf{A}.*\mathbf{B}$.

2.1.3 Matrix Inversion

For a square matrix \mathbf{A} , its inverse is a matrix \mathbf{A}^{-1} that satisfies the equation

$$\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \quad (2.12)$$

where \mathbf{I} is an identity matrix of the same dimensions as \mathbf{A} . An *identity* matrix is a square matrix of zeros with ones along the diagonal. For some matrices an inverse does not exist.

Matrices without an inverse are called singular matrices. All elements of the inverse of a matrix contain a division by the determinant of the matrix. An easy way to determine if a matrix is singular is to calculate its determinant. If the determinant is zero, the matrix is singular.

There are many ways to compute the inverse of a matrix. For small matrices, one can use analytical methods using cofactors and matrix determinants. For larger matrices, numerical methods, such as Gauss–Jordon elimination or LU decomposition are used. If the matrix is symmetric and positive definite, LU decomposition reduces to Cholesky factorization. See a text on matrix linear algebra for the particulars of each technique.

Remark 2.2 Within MATLAB[®], there are two predominant methods to find the solution of the equation $\mathbf{A} * \mathbf{x} = \mathbf{b}$, one of which is preferred. The less favorable, but most straightforward method is to take $\mathbf{x} = \mathbf{A}^{-1} * \mathbf{b}$ literally: $\mathbf{x} = \text{inv}(\mathbf{A}) * \mathbf{b}$. Here, $\text{inv}(\mathbf{A})$ is a MATLAB[®] function that computes \mathbf{A}^{-1} analytically and the operator $*$ represents matrix multiplication. The preferable solution is found using the matrix left division operator or backward slash: $\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$. This later method uses LU decomposition or Cholesky factorization in a very efficient way depending on the properties of the matrix \mathbf{A} .

As a special note, occasionally just the inverse of a matrix is needed and can be efficiently obtained using $\mathbf{B} = \mathbf{A} \backslash \text{eye}(\text{size}(\mathbf{A}))$, where \mathbf{B} is the inverse of \mathbf{A} . Here, $\text{eye}(N)$ is a MATLAB[®] function that produce an identity matrix $\in \mathbb{R}^{N \times N}$ that has ones along the diagonal and zeros everywhere else. The `eye` MATLAB[®] function has other properties that can produce nonsquare matrices, but they will not be discussed here.

If the matrix linear algebra equation is $\mathbf{x} * \mathbf{A} = \mathbf{b}$, then $\mathbf{x} = \mathbf{b} * \mathbf{A}^{-1}$, and the MATLAB[®] expression $\mathbf{x} = \mathbf{A} / \mathbf{b}$ is used. In a similar way, MATLAB picks the optimal algorithm for the computation based on the properties of \mathbf{A} and \mathbf{b} .

Finally, if only an element-by-element division of two matrices of identical dimension is desired, the MATLAB[®] expression $\mathbf{C} = \mathbf{A} ./ \mathbf{B}$ is used.

2.1.4 Block Matrix Inversion

For block matrices, there are several general formulas for inversion.

Lemma 2.1 Matrix Inversion Lemma: Let \mathbf{A} , \mathbf{C} and $(\mathbf{C}^{-1} + \mathbf{D}\mathbf{A}^{-1}\mathbf{B})$ be non-singular square matrices. Then

$$(\mathbf{A} + \mathbf{B}\mathbf{C}\mathbf{D})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B} \left(\mathbf{C}^{-1} + \mathbf{D}\mathbf{A}^{-1}\mathbf{B} \right)^{-1} \mathbf{D}\mathbf{A}^{-1} \quad (2.13)$$

Lemma 2.2 Matrix Inversion in Block Form: Let an $n \times m$ matrix \mathbf{M} be partitioned into a block form

$$\mathbf{M} = \underbrace{\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}}_{\substack{n \\ m}} \quad \} n \quad \} m \quad (2.14)$$

where the $n \times n$ matrix \mathbf{A} and the $m \times m$ matrix \mathbf{D} are invertible. Consider the product of two matrices such that

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{X} & \mathbf{Y} \\ \mathbf{Z} & \mathbf{U} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m \end{bmatrix} \quad (2.15)$$

Then, it follows immediately that

$$\begin{bmatrix} \mathbf{X} & \mathbf{Y} \\ \mathbf{Z} & \mathbf{U} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} \quad (2.16)$$

The block matrix inverse can be calculated from

$$\mathbf{M}^{-1} = \begin{bmatrix} (\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C})^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D} + \mathbf{CA}^{-1}\mathbf{B})^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C})^{-1} & (\mathbf{D} + \mathbf{CA}^{-1}\mathbf{B})^{-1} \end{bmatrix} \quad (2.17)$$

Or, one can calculate the block matrix inverse from

$$\begin{bmatrix} \mathbf{X} & \mathbf{Y} \\ \mathbf{Z} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m \end{bmatrix} \quad (2.18)$$

which results in the equivalent block matrix inverse

$$\mathbf{M}^{-1} = \begin{bmatrix} (\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C})^{-1} & -(\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C})^{-1}\mathbf{BD}^{-1} \\ -(\mathbf{D} + \mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1} & (\mathbf{D} + \mathbf{CA}^{-1}\mathbf{B})^{-1} \end{bmatrix} \quad (2.19)$$

2.1.5 Matrix Square Root

For any nonsingular square matrix \mathbf{A} , the square root of \mathbf{A} is defined as any matrix \mathbf{B} that satisfies the equation

$$\mathbf{A} = \mathbf{BB}^T \quad (2.20)$$

It is not actually a square root, but is rather a matrix decomposition.

Remark 2.3 In MATLAB®, there is a special function $\mathbf{B} = \text{sqrtm}(\mathbf{A})$ for computing the principal (positive) square root of the matrix \mathbf{A} . \mathbf{B} is the unique square root for which every eigenvalue has a nonnegative real part. If \mathbf{A} has any eigenvalues with negative real parts, then a complex result is produced. If \mathbf{A} is singular, then \mathbf{A} may not have a square root. MATLAB® displays an error message if an exact singularity is detected.

An alternate method uses a Cholesky decomposition to compute a lower triangular matrix \mathbf{B} that has strictly positive diagonal elements satisfying the properties of a matrix square root. $\mathbf{B} = \text{chol}(\mathbf{A})$ produces an upper triangular matrix \mathbf{B}^T from the matrix \mathbf{A} , satisfying the equation $\mathbf{A} = \mathbf{BB}^T$. The lower triangular matrix \mathbf{B} is assumed to be the (complex conjugate) transpose of the upper triangle matrix \mathbf{B}^T . Matrix \mathbf{A} must be positive definite; otherwise, MATLAB® displays an error message.

Although each MATLAB® method for computing the matrix square root results in a different matrix \mathbf{B} , they are both valid and differ only in the numerical technique used. Which method used will depend on the specific application.

2.2 VECTOR POINT GENERATORS

In the discussions on sigma point Kalman filters (Chapters 8–12), we will make use of a simplifying notation for the generation of the vector sigma points that are used for multidimensional numerical integration.

We define the n -dimensional vector *generator* function

$$\mathbf{u} = (u_1, \dots, u_r, 0, \dots, 0) \in \mathbb{R}^n \quad (2.21)$$

where $0 \leq u_i \leq u_j$ if $i \leq j$ [2,3]. In this notation, \mathbb{R}^n specifies the dimension of the vector points generated by \mathbf{u} and it is understood that u_i represents $\pm u_i$. Such a generator will be denoted as either

$$[\mathbf{u}] \in \mathbb{R}^n \quad (2.22)$$

or

$$[u_1, \dots, u_r] \in \mathbb{R}^n \quad (2.23)$$

where $\mathbf{u} \triangleq [u_1, \dots, u_r]$, with the zero coordinates suppressed for convenience. This notation represents the set of k multidimensional *vector points* that can be generated from \mathbf{u} by permutations and changing the sign of some coordinates.

In Part II, we will be considering rules for multidimensional integration of Gaussian-weighted integrals. Our numerical solutions will involve the evaluation of nonlinear functions at sets of vector points that consist of a constant times a set of vector points on unit hyperspheres or hypercubes. Hence, we will be using generators to create sets of vector points of the form $[1, 1, 1] \in \mathbb{R}^n$. Note that $[1, 1, 1] \in \mathbb{R}^n$ is equivalent to all possible permutations of $[\pm 1, \pm 1, \pm 1, 0, \dots, 0]$ where there are $n - 3$ zeros. Some examples are presented here. In the examples, we will use the notation [3]

$$n^{(k)} \triangleq n(n-1)\cdots(n-k+1) \quad (2.24)$$

$$n_{(k)} \triangleq \frac{n(n-1)\cdots(n-k+1)}{k!} \quad (2.25)$$

Consider the case where $u = 1$. Some examples of the vector points generated for a variety of cases are given below:

■ EXAMPLE 2.1

For $[1] \in \mathbb{R}^2$, the vector point set consists of the vectors

$$[1] \in \mathbb{R}^2 \triangleq \begin{cases} [1, 0]^T \\ [0, 1]^T \\ [-1, 0]^T \\ [0, -1]^T \end{cases} \quad (2.26)$$

For the space \mathbb{R}^2 , with $n = 2$ there will be $2n = 4$ vector points that lie along the axis of a two-dimensional Cartesian space, as shown in Figure 2.1.

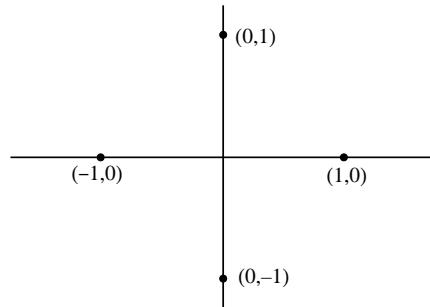


FIGURE 2.1 Example of two-dimensional Cartesian grid of axial vector points.

■ EXAMPLE 2.2

For $[1] \in \mathbb{R}^3$, the vector point set consists of

$$[1] \in \mathbb{R}^3 \triangleq \begin{cases} [1, 0, 0]^T \\ [0, 1, 0]^T \\ [0, 0, 1]^T \\ [-1, 0, 0]^T \\ [0, -1, 0]^T \\ [0, 0, -1]^T \end{cases} \quad (2.27)$$

Now, the vector points fall on the Cartesian axes of a three-dimensional hypersphere as shown in Figure 2.2.

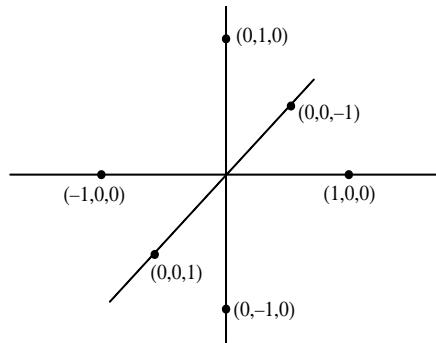


FIGURE 2.2 Example of three-dimensional Cartesian grid of axial vector points.

■ EXAMPLE 2.3

For $[1, 1] \in \mathbb{R}^2$, the vector point set consists of $2n^{(2)} = 2n(n - 1) = 4$ vector points given by

$$[1, 1] \in \mathbb{R}^2 \triangleq \begin{cases} [1, 1]^T \\ [1, -1]^T \\ [-1, 1]^T \\ [-1, -1]^T \end{cases} \quad (2.28)$$

■ EXAMPLE 2.4

For $[1, 1] \in \mathbb{R}^3$, we will have the $2n^{(2)} = 2n(n - 1)(n - 2) = 2 \cdot 3 \cdot 2 \cdot 1 = 12$ vector points given by the set

$$[1, 1] \in \mathbb{R}^3 \triangleq \begin{cases} [1, 1, 0]^T \\ [1, 0, 1]^T \\ [0, 1, 1]^T \\ [-1, 1, 0]^T \\ [-1, 0, 1]^T \\ [0, -1, 1]^T \\ [1, -1, 0]^T \\ [1, 0, -1]^T \\ [0, 1, -1]^T \\ [-1, -1, 0]^T \\ [-1, 0, -1]^T \\ [0, -1, -1]^T \end{cases} \quad (2.29)$$

Using (2.24) and (2.25), the number of vector points produced by a generator can be determined from the following formulas [3]

Generator	Number of Points
$[0] \in \mathbb{R}^n$	1
$[1] \in \mathbb{R}^n$	$2n$
$[1, 1] \in \mathbb{R}^n$	$2n^{(2)}$
$[1, 1, 1] \in \mathbb{R}^n$	$2^3 n_{(3)}$
$[1, 1, 1, 1] \in \mathbb{R}^n$	$2^4 n_{(4)}$
⋮	⋮

2.3 APPROXIMATING NONLINEAR MULTIDIMENSIONAL FUNCTIONS WITH MULTIDIMENSIONAL ARGUMENTS

For many tracking applications, estimation methods require the evaluation of integrals containing nonlinear multidimensional functions with multidimensional arguments weighted by a probability density. In this section, we will review methods for numerically approximating such nonlinear functions.

This section begins with a review of scalar methods that are then extended into multiple dimensions. All of the approximations to a nonlinear function are essentially expansions of the nonlinear function into a polynomial with arbitrary coefficients. Specification of the exact form of the coefficients depends on the application, the desired accuracy and other computational considerations. Issues related to the specification of these coefficients for estimation and tracking applications will be addressed in a later chapters on specific methods for the evaluation of density-weighted integrals.

In the first subsection we review methods of approximating scalar nonlinear functions with a scalar arguments. We begin with a discussion of a general polynomial expansion of a scalar function. This leads directly to the derivation of a scalar Taylor polynomial approximation for a nonlinear function. A numerical approximation of the Taylor polynomial is then derived by replacing the differentials of the Taylor polynomial by their finite difference equivalents, resulting in Stirling's interpolation formula (Stirling's polynomial).

This is followed by a subsection where the scalar approximations are generalized to approximating multidimensional nonlinear functions with multidimensional arguments through the introduction of multidimensional polynomial expansions. And, once again, this leads to the derivation of a general multidimensional Taylor polynomial. However, the generalization of Stirling's approximation to multiple dimensions can be accomplished in several ways, each of which leads to a different numerical approximation method. We derive a straightforward multidimensional Stirling's approximation that requires 3^{n_x} Cartesian interpolation points, where n_x is the dimension of the *argument* of the multidimensional function. Then we briefly discuss two simplifications. For the first simplification method, the number of interpolation points is arbitrarily reduced so as to include only those points that lie along the coordinate axis, reducing the required number of interpolation points to $2n_x + 1$. A second modification method is to use a simplex version of Stirling's approximation, which reduces the number of interpolation points to $n_x + 2$.

In this text, we will restrict ourselves to *methods* of approximation and will almost completely ignore the important subjects of approximation error estimation and convergence.

2.3.1 Approximating Scalar Nonlinear Functions

2.3.1.1 A General Polynomial Expansion. In general, any continuous scalar nonlinear function with a scalar argument, $f(x)$, can be approximated about an arbitrary

point x_0 to *any degree of accuracy* by a series approximation

$$f(x) = \sum_{i=0}^{\infty} a_i (x - x_0)^i = a_0 + a_1 (x - x_0) + a_2 (x - x_0)^2 + a_3 (x - x_0)^3 + \dots \quad (2.31)$$

Since it is usually difficult to find a closed form solution that takes the sum out to ∞ , the series is truncated into a *polynomial* approximation of order M

$$f(x) \simeq \sum_{i=0}^M a_i (x - x_0)^i = a_0 + a_1 (x - x_0) + a_2 (x - x_0)^2 + \dots + a_M (x - x_0)^M \quad (2.32)$$

Each of the terms of such a polynomial is a *monomial* of the form $a_i (x - x_0)^i$ that can be further expanded into monomial terms x^i with coefficients b_i . Therefore, any polynomial approximation of the function $f(x)$ can be written as an expansion of the form

$$f(x) \simeq \sum_{i=0}^M b_i x^i = b_0 + b_1 x + b_2 x^2 + \dots + b_M x^M \quad (2.33)$$

where

$$\begin{aligned} b_0 &\triangleq \sum_{i=0}^M (-1)^i a_i x_0^i \\ b_1 &\triangleq \sum_{i=1}^M (-2)^{i-1} a_i x_0^{i-1} \\ &\vdots \\ b_M &\triangleq a_M \end{aligned} \quad (2.34)$$

2.3.1.2 Taylor Polynomial Expansion. We now present a simple derivation of the Taylor and Maclaurin polynomials of order M . This requires that derivatives for $f(x)$ exist up to order M . Letting $x = x_0$ in (2.32) we obtain

$$a_0 = [f(x)]_{x=x_0} \quad (2.35)$$

To obtain a_1 , take the derivative of (2.32) with respect to x and set $x = x_0$, resulting in

$$a_1 = \left[\frac{d}{dx} f(x) \right]_{x=x_0} \quad (2.36)$$

Taking successive derivatives of (2.32) and setting $x = x_0$ for each case results in the evaluation of all coefficients in terms of higher order derivatives of $f(x)$ and leads to

the well-known scalar *Taylor polynomial* point approximation for $f(x_0)$,

$$\begin{aligned} f(x_0) &\simeq [f(x)]_{x=x_0} + \left[\frac{d}{dx} f(x) \right]_{x=x_0} (x - x_0) \\ &\quad + \cdots + \frac{1}{M!} \left[\left(\frac{d}{dx} \right)^M f(x) \right]_{x=x_0} (x - x_0)^M \end{aligned} \quad (2.37)$$

$$= \sum_{i=0}^M \frac{1}{i!} (x - x_0)^i \left[\left(\frac{d}{dx} \right)^i f(x) \right]_{x=x_0} \quad (2.38)$$

Now, defining the scalar operator D_x^i by its operation on $f(x)$

$$D_x^i f(x) \triangleq (x - x_0)^i \left[\left(\frac{d}{dx} \right)^i f(x) \right]_{x=x_0} \quad (2.39)$$

the M th order Taylor polynomial (2.38) becomes

$$f(x) = \sum_{i=0}^M \frac{1}{i!} D_x^i f(x) \quad (2.40)$$

Because it is sometimes difficult to calculate the higher order derivatives of $f(x)$, the Taylor polynomial approximation is usually terminated after no more than three terms ($M = 2$). A slightly different derivation of the Taylor polynomial can be found in Arfken [4], where he derives the error due to truncation of the Taylor polynomial. The error term in a Taylor polynomial of order M given by

$$R_M = \frac{(x - x_0)^M}{N!} \left(\frac{d}{d\xi} \right)^M f(\xi) \quad (2.41)$$

with $x_0 \leq \xi \leq x$.

2.3.1.3 Stirling's Polynomial Expansion. An alternate derivative-free polynomial approximation for $f(x)$ can be obtained by using central differences in place of the derivatives in the Taylor polynomial approximation. Finite difference approximations can be derived through the use of Taylor series expansions. Suppose we have a function $f(x)$, which is continuous and differentiable over the range of interest. Let's also assume we know the value $f(x_0)$ and all the derivatives at $x = x_0$. Using (2.37), examine the first-order Taylor polynomials for $f(x_0 + q)$ about x_0

$$f(x_0 + q) = f(x_0) + f'(x_0) q \quad (2.42)$$

where $q \triangleq x - x_0$ is the step size and $f'(x_0) \triangleq df(x_0) / dx_0$.

Following the same procedure, we can write

$$f(x_0 - q) = f(x_0) - f'(x_0) q \quad (2.43)$$

Subtracting the two equations leads to the first-order central difference approximation

$$f'(x) = \frac{1}{2q} [f(x+q) - f(x-q)] \quad (2.44)$$

where we have used x instead of x_0 .

Now following the same procedure for the second-order Taylor polynomial, we can write

$$f(x+q) = f(x) + f'(x)q + \frac{1}{2}f''(x)q^2 \quad (2.45)$$

and

$$f(x-q) = f(x) - f'(x)q + \frac{1}{2}f''(x)q^2 \quad (2.46)$$

where $f''(x) \triangleq d^2 f(x)/dx^2$. Adding the two equations leads to

$$f''(x) = \frac{1}{q^2} [f(x+q) - 2f(x) + f(x-q)] \quad (2.47)$$

Using these central difference approximations of the derivative terms, the second-order Taylor polynomial (2.37) reduces to the second-order *Stirling's interpolation formula* [5] around the point x_0

$$f(x) = f(x_0) + \tilde{D}_x f(x) + \frac{1}{2}\tilde{D}_x^2 f(x) \quad (2.48)$$

where

$$\tilde{D}_x f(x) \triangleq \frac{1}{2q} [f(x_0+q) - f(x_0-q)](x-x_0) \quad (2.49)$$

and

$$\tilde{D}_x^2 f(x) \triangleq \frac{1}{q^2} [f(x_0+q) - 2f(x_0) + f(x_0-q)](x-x_0)^2 \quad (2.50)$$

The approximation for $x_0 = 0$ follows in an obvious way.

This represents a derivative-free approximation to $f(x)$ with the free step-size parameter q . Examination of the above second-order approximation reveals that, for the one-dimensional case, $f(x)$ must be known at only the three points $\{x_0, x_0+q, x_0-q\}$. Higher order approximations can be obtained simply by approximating higher order derivatives, resulting in an increase in the number of evaluation points on a one-dimensional uniform grid with spacing q .

To summarize scalar methods, we have presented two polynomial approximations for the function $f(x)$, expansions in terms of the *Taylor polynomial* and *Stirling's interpolation formula*. In both cases, we have examined *point approximations*, that is, polynomial approximations of $f(x)$ at the point x_0 . Polynomial approximations of a nonlinear function over a larger support region is a much broader topic and is beyond the scope of this text.

2.3.2 Approximating Multidimensional Nonlinear Functions

2.3.2.1 A General Multidimensional Polynomial Expansion. For the multidimensional case, we can write the nonlinear function as a vector function with a vector argument, $\mathbf{f}(\mathbf{x})$, where $\mathbf{f} \in \mathbb{R}^{n_f}$ and $\mathbf{x} \in \mathbb{R}^{n_x}$

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_{n_f}(\mathbf{x})]^\top \quad (2.51)$$

and

$$\mathbf{x} = [x_1, x_2, \dots, x_{n_x}]^\top \quad (2.52)$$

Note that the vector function \mathbf{f} may not have the same number of dimensions as \mathbf{x} . For a *scalar* nonlinear function with a vector argument, a general *polynomial* expansion about the point $\mathbf{x}_0 \triangleq [x_{0,1}, x_{0,2}, \dots, x_{0,n_x}]^\top$ of order d can be written in terms of multidimensional monomials [2]

$$\begin{aligned} f(\mathbf{x}) &= f(x_1, x_2, \dots, x_{n_x}) \\ &\simeq \sum_{i_1=0}^d \cdots \sum_{i_{n_x}=0}^d a_j(i_1, i_2, \dots, i_{n_x}) \\ &\quad \times (x_1 - x_{0,1})^{i_1} (x_2 - x_{0,2})^{i_2} \cdots (x_{n_x} - x_{0,n_x})^{i_{n_x}} \end{aligned} \quad (2.53)$$

where $i_1 + i_2 + \cdots + i_{n_x} \leq d$ indicates that only those terms of the multiple sum that satisfy the specified condition can be used in the polynomial expansion. This is made clear with the following example.

■ EXAMPLE 2.5

For $\mathbf{x} \in \mathbb{R}^2$, a basis of multidimensional monomials spanned by monomials of degree $d = 2$ or less is the set of six monomials $\{1, (x_1 - x_{0,1}), (x_2 - x_{0,2}), (x_1 - x_{0,1})(x_2 - x_{0,2}), (x_1 - x_{0,1})^2, (x_2 - x_{0,2})^2\}$. The polynomial approximation of $f(x)$ about the point x_0 follows immediately

$$\begin{aligned} f(\mathbf{x}) &= f(x_1, x_2) \simeq a(0, 0) \\ &\quad + a(1, 0)(x_1 - x_{0,1}) + a(0, 1)(x_2 - x_{0,2}) \\ &\quad + a(1, 1)(x_1 - x_{0,1})(x_2 - x_{0,2}) \\ &\quad + a(2, 0)(x_1 - x_{0,1})^2 + a(0, 2)(x_2 - x_{0,2})^2 \end{aligned} \quad (2.54)$$

This can be written in the form (2.53)

$$f(\mathbf{x}) = \sum_{i_1=0}^d \sum_{i_2=0}^d a(i_1, i_2) (x_1 - x_{0,1})^{i_1} (x_2 - x_{0,2})^{i_2} \quad (2.55)$$

where the condition $i_1 + i_2 \leq d = 2$ must be met for *each* monomial term in the polynomial.

In general, for a vector $\mathbf{x} \in \mathbb{R}^{n_x}$ there are $\binom{n_x+d}{d}$ distinct monomials of precision d or less. The binomial coefficient $\binom{n}{m}$ is the number of ways of choosing m objects from a collection of n distinct objects without regard to order and is defined as

$$\binom{n}{m} \triangleq \frac{n!}{m!(n-m)!} \quad (2.56)$$

By inspection, the polynomial expansion for the *vector* function $\mathbf{f}(\mathbf{x})$ about the point \mathbf{x}_0 can be written as

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= \mathbf{f}(x_1, x_2, \dots, x_{n_x}) \\ &\simeq \sum_{i_1=0}^d \cdots \sum_{i_{n_x}=0}^d \mathbf{a}(i_1, i_2, \dots, i_{n_x}) \\ &\quad \times (x_1 - x_{0,1})^{i_1} (x_2 - x_{0,2})^{i_2} \cdots (x_{n_x} - x_{0,n_x})^{i_{n_x}} \end{aligned} \quad (2.57)$$

where

$$\mathbf{a} = [a_1, a_2, \dots, a_{n_f}]^\top \quad (2.58)$$

■ EXAMPLE 2.6

Again considering the case where $\mathbf{x} \in \mathbb{R}^2$ and keeping only terms up to second order, (2.57) becomes

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &\simeq \mathbf{a}(0, 0) + \mathbf{a}(1, 0)(x_1 - x_{0,1}) + \mathbf{a}(0, 1)(x_2 - x_{0,2}) \\ &\quad + \mathbf{a}(1, 1)(x_1 - x_{0,1})(x_2 - x_{0,2}) \\ &\quad + \mathbf{a}(2, 0)(x_1 - x_{0,1})^2 + \mathbf{a}(0, 2)(x_2 - x_{0,2})^2 \end{aligned} \quad (2.59)$$

In the next two sections we will first address the Taylor polynomial approximation for a multidimensional function, followed by a discussion of a multidimensional extension of Stirling's interpolation formula.

2.3.2.2 Multidimensional Taylor Polynomial Expansion. Beginning with the case where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^{n_f}$, the Taylor polynomial can be developed by first setting $\mathbf{x} = \mathbf{x}_0$ in (2.59) resulting in

$$\mathbf{a}(0, 0) = [\mathbf{f}(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0} \quad (2.60)$$

Now, taking the partial derivative of $\mathbf{f}(\mathbf{x})$ with respect to x_1 in (2.59) and then setting $\mathbf{x} = \mathbf{x}_0$ leads to

$$\mathbf{a}(1, 0) = \left[\frac{\partial}{\partial x_1} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \quad (2.61)$$

where the $\partial/\partial x_1$ operates on each component of $\mathbf{f}(\mathbf{x})$.

In a similar fashion, the remaining coefficients in (2.59) can be determined, resulting in the second-order polynomial expansion

$$\begin{aligned} \mathbf{f}(\mathbf{x}) \simeq & [\mathbf{f}(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0} + \left\{ (x_1 - x_{0,1}) \left[\frac{\partial}{\partial x_1} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \right. \\ & + (x_2 - x_{0,2}) \left[\frac{\partial}{\partial x_2} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \Big\} \\ & + \frac{1}{2!} \left\{ (x_1 - x_{0,1})^2 \left[\frac{\partial^2}{\partial x_1^2} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \right. \\ & + 2(x_1 - x_{0,1})(x_2 - x_{0,2}) \left[\frac{\partial^2}{\partial x_1 \partial x_2} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \\ & \left. + (x_2 - x_{0,2})^2 \left[\frac{\partial^2}{\partial x_2^2} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \right\} \end{aligned} \quad (2.62)$$

This can immediately be generalized to an M th order multidimensional Taylor polynomial approximation

$$\mathbf{f}(\mathbf{x}) \simeq \sum_{i=0}^M \frac{1}{i!} D_{\mathbf{x}-\mathbf{x}_0}^i \mathbf{f}(\mathbf{x}) \quad (2.63)$$

where the operator notation similar to that employed by Williamson et al. [6] has been adopted, with the scalar operator $D_{\mathbf{x}-\mathbf{x}_0}^i$ defined by

$$\begin{aligned} D_{\mathbf{x}-\mathbf{x}_0}^i \mathbf{f}(\mathbf{x}) \triangleq & \left[(x_1 - x_{0,1}) \frac{\partial}{\partial x_1} + (x_2 - x_{0,2}) \frac{\partial}{\partial x_2} \right. \\ & + \cdots + (x_{n_x} - x_{0,n_x}) \left. \frac{\partial}{\partial x_{n_x}} \right]^i \mathbf{f}(\mathbf{x}) \Big|_{\mathbf{x}=\mathbf{x}_0} \end{aligned} \quad (2.64)$$

The notation $[\cdot] |_{\mathbf{x}=\mathbf{x}_0}$ applies only to the differential terms.

When a polynomial $(x_1 + \cdots + x_n)^i$ is multiplied out, each term will consist of a constant times a factor of the form $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$ where the nonnegative integers i_k

satisfy $i_1 + \dots + i_n = i$. The multinomial expansion has the form

$$(x_1 + \dots + x_n)^i = \sum_{\substack{i_1=0 \\ i_1+\dots+i_n=i}}^i \dots \sum_{\substack{i_n=0 \\ i_1+\dots+i_n=i}}^i \binom{i}{i_1 \dots i_n} x_1^{i_1} \dots x_n^{i_n} \quad (2.65)$$

The multinomial coefficients can be calculated from

$$\binom{i}{i_1 \dots i_n} \triangleq \frac{i!}{i_1! \dots i_n!} \quad (2.66)$$

■ EXAMPLE 2.7

Consider the case

$$(x_1 + x_2 + x_3)^2 = \sum_{\substack{i_1=0 \\ i_1+i_2+i_3=2}}^2 \sum_{i_2=0}^2 \sum_{i_3=0}^2 \binom{2}{i_1 \dots i_3} x_1^{i_1} x_2^{i_2} x_3^{i_3} \quad (2.67)$$

$$= x_1^2 + x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3 + 2x_2x_3 \quad (2.68)$$

It follows immediately that (2.64) can be written as

$$\begin{aligned} D_{\mathbf{x}-\mathbf{x}_0}^i \mathbf{f}(\mathbf{x}) &= \sum_{\substack{i_1=0 \\ i_1+\dots+i_n=i}}^i \dots \sum_{\substack{i_n=0 \\ i_1+\dots+i_n=i}}^i \binom{i}{i_1 \dots i_n} (x_1 - x_{0,1})^{i_1} \dots (x_{n_x} - x_{0,n_x})^{i_n} \\ &\quad \times \left[\frac{\partial^i}{\partial x_1^{i_1} \dots \partial x_{n_x}^{i_n}} \mathbf{f}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}_0} \end{aligned} \quad (2.69)$$

Converting (2.63) into vector-matrix notation and keeping only terms to second-order results in the Taylor polynomial approximation for a multidimensional function

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= [\mathbf{f}(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0} + [\nabla_{\mathbf{x}} \mathbf{f}^T(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0}^\top (\mathbf{x} - \mathbf{x}_0) \\ &\quad + \frac{1}{2!} \sum_{p=1}^{n_x} \mathbf{e}_p(\mathbf{x} - \mathbf{x}_0)^\top [\nabla_{\mathbf{x}} \nabla_{\mathbf{x}}^T f_p(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) \end{aligned} \quad (2.70)$$

The first-order term contains the Jacobian matrix, $[\nabla_{\mathbf{x}} \mathbf{f}^T(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0}^\top \in \mathbb{R}^{n_x \times n_f}$, while the p th component of the second-order term contains the Hessian matrix $[\nabla_{\mathbf{x}} \nabla_{\mathbf{x}}^T f_p(\mathbf{x})]_{\mathbf{x}=\mathbf{x}_0} \in \mathbb{R}^{n_x \times n_x}$ and \mathbf{e}_p is an n_x -dimensional unit vector along the p th-dimensional Cartesian axis. This *Taylor polynomial* expansion for a multidimensional function is very difficult to evaluate beyond the first-order term, so the series is usually truncated without any of the higher order terms.

2.3.2.3 Multidimensional Stirling's Polynomial Expansion. A version of derivative-free Stirling's polynomial interpolation formula for multidimensional functions can be written as [2,7,8]

$$\mathbf{f}(\mathbf{x}) \simeq \sum_{i=0}^M \frac{1}{i!} \tilde{D}_{\mathbf{x}-\mathbf{x}_0}^i \mathbf{f}(\mathbf{x}) \quad (2.71)$$

or, keeping only terms to *second* order

$$\mathbf{f}(\mathbf{x}) \simeq \mathbf{f}(\mathbf{x}_0) + \tilde{D}_{\mathbf{x}-\mathbf{x}_0} \mathbf{f}(\mathbf{x}) + \frac{1}{2!} \tilde{D}_{\mathbf{x}-\mathbf{x}_0}^2 \mathbf{f}(\mathbf{x}) \quad (2.72)$$

The central difference operators $\tilde{D}_{\mathbf{x}-\mathbf{x}_0} \mathbf{f}(\mathbf{x})$ and $\tilde{D}_{\mathbf{x}-\mathbf{x}_0}^2 \mathbf{f}(\mathbf{x})$ are defined by

$$\tilde{D}_{\mathbf{x}-\mathbf{x}_0} \mathbf{f}(\mathbf{x}) \triangleq \frac{1}{2q} \sum_{p=1}^{n_x} (x_p - x_{0,p}) [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_p) - \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_p)] \quad (2.73)$$

$$\begin{aligned} \tilde{D}_{\mathbf{x}-\mathbf{x}_0}^2 \mathbf{f}(\mathbf{x}) &\triangleq \frac{1}{q^2} \left\{ \sum_{p=1}^{n_x} (x_p - x_{0,p})^2 [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_p) - 2\mathbf{f}(\mathbf{x}_0) + \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_p)] \right. \\ &+ 2 \sum_{p=1}^{n_x} \sum_{\substack{r=1 \\ r \neq p}}^{n_x} (x_p - x_{0,p})(x_r - x_{0,r}) \\ &\times [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_p + q\mathbf{e}_r) - \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_p + q\mathbf{e}_r) \\ &\left. - \mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_p - q\mathbf{e}_r) + \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_p - q\mathbf{e}_r)] \right\} \end{aligned} \quad (2.74)$$

where \mathbf{e}_p and \mathbf{e}_r are unit vectors along the p th-and r th-dimensional Cartesian axis, respectively.

■ EXAMPLE 2.8

For $\mathbf{x} \in \mathbb{R}^2$, these reduce to

$$\begin{aligned} \tilde{D}_{\mathbf{x}-\mathbf{x}_0} \mathbf{f}(\mathbf{x}) &\triangleq \frac{1}{2q} \left\{ (x_1 - x_{0,1}) [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_1) - \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_1)] \right. \\ &+ (x_2 - x_{0,2}) [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_2) - \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_2)] \left. \right\} \end{aligned} \quad (2.75)$$

and

$$\begin{aligned}\tilde{D}_{\mathbf{x}-\mathbf{x}_0}^2 \mathbf{f}(\mathbf{x}) &\triangleq \frac{1}{q^2} (x_1 - x_{0,1})^2 [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_1) - 2\mathbf{f}(\mathbf{x}_0) + \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_1)] \\ &\quad + \frac{1}{q^2} (x_2 - x_{0,2})^2 [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_2) - 2\mathbf{f}(\mathbf{x}_0) + \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_2)] \\ &\quad + \frac{2}{q^2} (x_1 - x_{0,1})(x_2 - x_{0,2}) \\ &\quad \times [\mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_1 + q\mathbf{e}_2) - \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_1 + q\mathbf{e}_2) \\ &\quad - \mathbf{f}(\mathbf{x}_0 + q\mathbf{e}_1 - q\mathbf{e}_2) + \mathbf{f}(\mathbf{x}_0 - q\mathbf{e}_1 - q\mathbf{e}_2)]\end{aligned}\quad (2.76)$$

It is obvious that the interpolation for $\mathbf{x} \in \mathbb{R}^2$ requires a set of nine points on a uniform two-dimensional grid with spacing q , that is, $J_2 \in \{(x_{0,1}, x_{0,2}), (x_{0,1} \pm q, x_{0,2}), (x_{0,1}, x_{0,2} \pm q), (x_{0,1} \pm q, x_{0,2} \pm q)\}$, as shown in Figure 2.3.

In general, the number of points required for the second-order multidimensional Stirling's polynomial approximation is 3^{n_x} on a uniform multidimensional grid (hypersquare) with spacing q .

An alternative method that uses a functional expansion in terms of Hermite orthogonal polynomials [5,9] in place of Equations (2.63) and (2.69) results in the same number of interpolation points as the second-order multidimensional Stirling's approximation (the interpolation points may not be identical). This alternative method does not require any derivatives or finite differences and in many ways is more satisfying than the finite difference approach. Of course, such a statement is purely a subjective personal view. The treatment of this method of interpolation will be postponed until the chapters on numerical integration of Gaussian-weighted integrals later

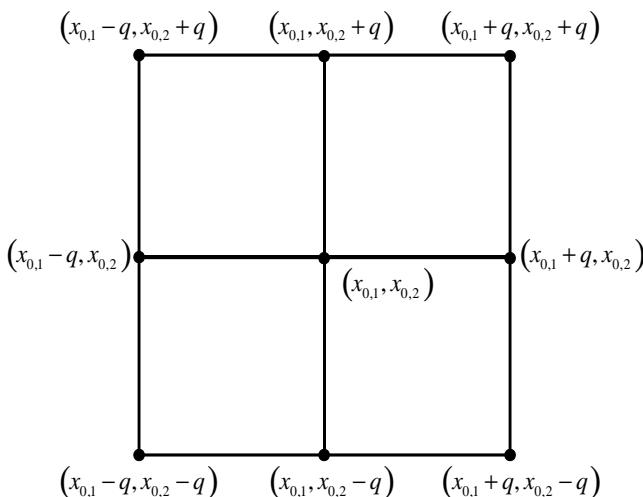


FIGURE 2.3 Two-dimensional points for multidimensional Sterlign's approximation.

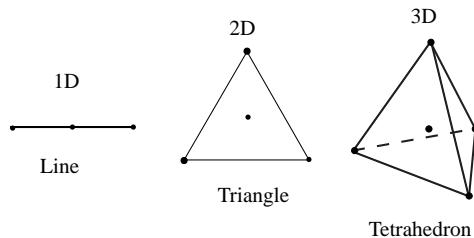


FIGURE 2.4 Simplex figures in up to three dimensions.

in this book. A discussion of Hermite interpolation is postponed because a Gaussian-weighted integral can be changed into the Hermite integral by a simple transform making understanding of the method simpler.

There are other derivative-free methods of interpolation that require fewer interpolation points than the multidimensional Stirling's polynomial approximation described above. One such interpolation method uses only those interpolation points in Figure 2.3 that lie along the orthogonal Cartesian axes and the origin. Examination of (2.73) and (2.74) reveals that only the cross-terms from (2.74) have been removed at the cost of a small amount of accuracy for the second-order term. With this simplification, the number of interpolation points is reduced to $2n_x + 1$.

Another multidimensional interpolation scheme uses one point at the origin and additional points equally spaced in angle around the circumference of a multidimensional hypersphere, and is referred to as spherical simplex interpolation. An example of regular simplex figures (ones with all sides and angles equal) that have vertices on a hypersphere are shown in Figure 2.4 for dimensions up to three. These methods can require as few as $n_x + 2$ points for second-order interpolation. Although simplex interpolation methods are the basis of numerical integration methods for multidimensional functions, only recently have simplex methods for multidimensional interpolation appeared in the literature in applications related to computer graphics [10–16]. Consequently, an exposition of purely interpolative methods on the simplex are beyond the scope of this text. In spite of this, we will present a simplex method for integration of Gaussian-weighted multidimensional functions in Chapter 10.

2.4 OVERVIEW OF MULTIVARIATE STATISTICS

2.4.1 General Definitions

From a Bayesian viewpoint, a stochastic variable is a variable whose value is not fixed deterministically, but can vary according to a probability distribution. The stochastic variable is often multidimensional so that the underlying probability distribution is multivariate. In addition, the expected value of the stochastic variable can be time varying, with the change of the underlying distribution in time governed by some dynamic or transition equation.

2.4.1.1 Probability Density Function. We begin this section with a discussion of the general definitions of a probability density function (pdf) and a cumulative distribution function (cdf) for multidimensional stochastic variables.

A *probability density function* of a stochastic (random) multidimensional variable \mathbf{x} at the point $\mathbf{x} = \xi$ is defined by [17]

$$p_{\mathbf{x}}(\xi) d\xi \triangleq \text{Prob}\{\xi \leq \mathbf{x} \leq \xi + d\xi\} \triangleq P\{\xi \leq \mathbf{x} \leq \xi + d\xi\} > 0 \quad (2.77)$$

where $d\xi$ is an infinitesimal interval. It follows immediately that the probability that \mathbf{x} lies in the interval $\eta \leq \mathbf{x} \leq \xi$ is given by

$$\text{Prob}\{\eta \leq \mathbf{x} \leq \xi\} \triangleq P\{\eta \leq \mathbf{x} \leq \xi\} = \int_{\eta}^{\xi} p_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} \quad (2.78)$$

where n_x is the dimension of \mathbf{x} and the multidimensional integral is defined by

$$\begin{aligned} \int_{\eta}^{\xi} p_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} &\triangleq \int_{\eta_1}^{\xi_1} dx_1 \cdots \int_{\eta_{n_x}}^{\xi_{n_x}} dx_{n_x} p_{\mathbf{x}}(x_1, \dots, x_{n_x}) \\ &= \prod_{i=1}^{n_x} \int_{\eta_i}^{\xi_i} dx_i p_{\mathbf{x}}(x_1, \dots, x_{n_x}) \end{aligned} \quad (2.79)$$

Thus, the probability that \mathbf{x} lies in the interval $\eta \leq \mathbf{x} \leq \xi$ is the multidimensional “volume” under the pdf over the interval. A more common notation for the pdf is to use $p(\mathbf{x})$ to mean $p_{\mathbf{x}}(\mathbf{x})$.

2.4.1.2 Cumulative Distribution Function. The *cumulative distribution function* is defined as

$$P_{\mathbf{x}}(\xi) \triangleq \text{Prob}\{\mathbf{x} \leq \xi\} = \int_{-\infty}^{\xi} p_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} \quad (2.80)$$

Note that as $\xi \rightarrow \infty$, $P_{\mathbf{x}}(\xi)$ goes to 1 because the pdf is always normalized to integrate to 1. The relationship between the pdf and cdf is given by

$$p(\mathbf{x}) = \left[\frac{d}{d\xi} P_{\mathbf{x}}(\xi) \right]_{\xi=\mathbf{x}} \quad (2.81)$$

2.4.1.3 Joint and Marginal Probability Density Functions. The *joint pdf* of two multidimensional random variables is the probability of the joint event $p(\mathbf{x}, \mathbf{z})$. The *marginal pdf* of \mathbf{x} is defined by

$$p(\mathbf{x}) = \int_{-\infty}^{\infty} p(\mathbf{x}, \mathbf{z}) d\mathbf{z} \quad (2.82)$$

The *conditional probability density* of \mathbf{x} given \mathbf{z} is

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{z})} \quad (2.83)$$

Thus, the pdf of \mathbf{x} can be obtained from

$$p(\mathbf{x}) = \int_{-\infty}^{\infty} p(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int_{-\infty}^{\infty} p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} \quad (2.84)$$

2.4.1.4 Bayes' Rule. Bayes' Rule follows immediately from the conditional probability

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) = p(\mathbf{z}|\mathbf{x}) p(\mathbf{x}) \quad (2.85)$$

and

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{z}|\mathbf{x}) p(\mathbf{x})}{p(\mathbf{z})} = \frac{p(\mathbf{z}|\mathbf{x}) p(\mathbf{x})}{\int_{-\infty}^{\infty} p(\mathbf{z}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}} \quad (2.86)$$

Thus, we have the proportionality (\propto indicates proportionality)

$$p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{z}|\mathbf{x}) p(\mathbf{x}) \quad (2.87)$$

2.4.1.5 Point Estimates (Moments) of a Probability Density Function. Consider the random vector $\mathbf{x} \triangleq [x_1, x_2, \dots, x_{n_x}]^T \in \mathbb{R}^{n_x}$ that is governed by the pdf $p(\mathbf{x})$. The *expected value* (first moment) of \mathbf{x} is defined by

$$\hat{\mathbf{x}} \triangleq \mathcal{E}\{\mathbf{x}\} = \int_{-\infty}^{\infty} \mathbf{x} p(\mathbf{x}) d\mathbf{x} \quad (2.88)$$

The *covariance* (second moment) is given by

$$\begin{aligned} \mathbf{P}^{\mathbf{xx}} &\triangleq \mathcal{E}\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T\} \\ &= \int_{-\infty}^{\infty} (\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T p(\mathbf{x}) d\mathbf{x} \\ &= \int_{-\infty}^{\infty} \mathbf{x}\mathbf{x}^T p(\mathbf{x}) d\mathbf{x} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T \end{aligned} \quad (2.89)$$

For two n_x -dimensional random vectors $\mathbf{x} \triangleq [x_1, x_2, \dots, x_{n_x}]^\top$ and $\mathbf{z} \triangleq [z_1, z_2, \dots, z_{n_x}]^\top$, the *cross-covariance* between the two variables is defined as

$$\begin{aligned}\mathbf{P}^{\mathbf{xz}} &\triangleq \mathcal{E}\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{z} - \hat{\mathbf{z}})^\top\} \\ &= \int_{-\infty}^{\infty} \int_{\mathbb{R}^{n_x}} (\mathbf{x} - \hat{\mathbf{x}})(\mathbf{z} - \hat{\mathbf{z}})^\top p(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z} \\ &= \int_{-\infty}^{\infty} \mathbf{xz}^\top p(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z} - \hat{\mathbf{x}}\hat{\mathbf{z}}^\top\end{aligned}\quad (2.90)$$

It must be noted that for some probability densities these moments do not exist because the moment integrals are infinite.

2.4.2 The Gaussian Density

The Gaussian probability density function has been important in estimation theory since it was first introduced by Abraham de Moivre in an article in the year 1733 [18]. Because simple analytical forms for this symmetric distribution are known for both scalar and vector stochastic variables, the Gaussian distribution is used to model noise processes in many fields. A very good summary of the properties of Gaussian distributions can be found online in Ref. [19]. In the paragraphs that follow, we will discuss those characteristics of Gaussian densities that are important to the recursive estimation methods developed in this book. We begin with a presentation of the Gaussian *probability density function*, followed by a derivation of the Gaussian *cumulative distribution function*.

We then introduce an affine transformation that transforms the origin of the coordinate system to the mean of the Gaussian density and rotates the coordinate axes to align them with the principle axis of the density function. This is done primarily to simplify the Gaussian-weighted integration methods developed in Part II. Finally, a set of identities for the transformed moment equations are presented.

2.4.2.1 The Gaussian pdf. In estimation of the moments of a stochastic variable, the most common assumption is that the underlying probability distribution is Gaussian. A one-dimensional stochastic variable x is governed by a *Gaussian pdf* if

$$p(x) = \mathcal{N}(x; \hat{x}, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2\sigma^2}[x - \hat{x}]^2\right\}\quad (2.91)$$

where \hat{x} is the first moment or mean value and σ^2 is the second moment or variance. σ is usually called the standard deviation. An example of a one-dimensional Gaussian pdf as a function of x for $\hat{x} = -5$ and $\sigma = 6$ is given in Figure 2.5.

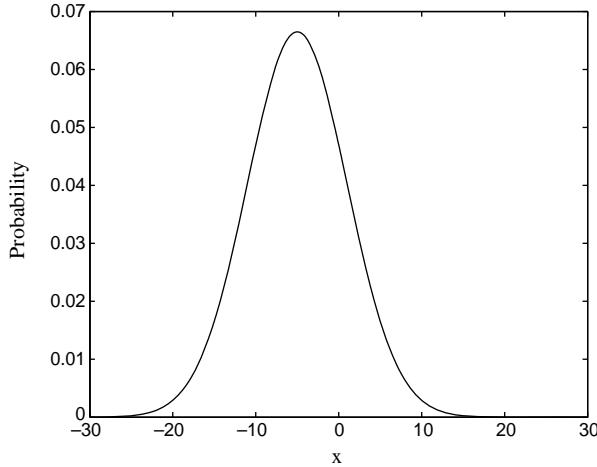


FIGURE 2.5 One-dimensional Gaussian pdf example.

The Gaussian pdf of an n_x -dimensional stochastic variable \mathbf{x} is defined by

$$\begin{aligned} p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\mathbf{xx}}) &\triangleq (2\pi)^{-n_x/2} |\mathbf{P}^{\mathbf{xx}}|^{-1/2} \\ &\times \exp \left\{ -\frac{1}{2} [\mathbf{x} - \hat{\mathbf{x}}]^\top (\mathbf{P}^{\mathbf{xx}})^{-1} [\mathbf{x} - \hat{\mathbf{x}}] \right\} \end{aligned} \quad (2.92)$$

where $\hat{\mathbf{x}}$ and $\mathbf{P}^{\mathbf{xx}}$ are the mean vector and the covariance matrix, respectively, and $|\mathbf{A}|$ represents the determinant of the matrix \mathbf{A} . Note that p is always a scalar.

Consider the two-dimensional Gaussian pdf with

$$\hat{\mathbf{x}} = [\hat{x}, \hat{y}]^\top = [-5, 2]^\top \quad (2.93)$$

and $\sigma_x = 6$, $\sigma_y = 8$, and $\rho = 0.5$. Now $\mathbf{P}^{\mathbf{xx}} \rightarrow \mathbf{P}^{xy}$ and we can write

$$\mathbf{P}^{xy} = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} = \begin{bmatrix} 6^2 & 24 \\ 24 & 8^2 \end{bmatrix} \quad (2.94)$$

The two-dimensional cross-covariance can also be written in the form

$$\mathbf{P}^{xy} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{bmatrix} \quad (2.95)$$

with $\sigma_{xy}^2 \triangleq \rho\sigma_x\sigma_y$. A graphical depiction of this density is presented in Figure 2.6.

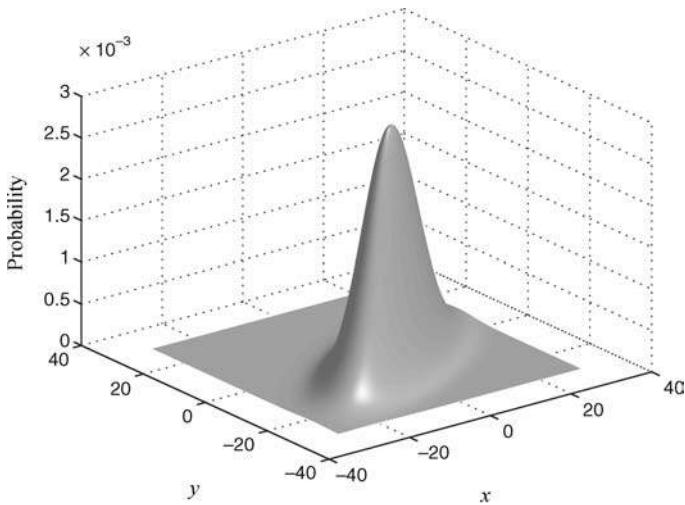


FIGURE 2.6 Example of a two-dimensional Gaussian pdf.

2.4.2.2 The Gaussian cdf. The multidimensional Gaussian *cumulative distribution function (cdf)* is defined by

$$\begin{aligned}
 P(\mathbf{x}) &\triangleq \int_{\mathbb{R}^{n_x}}^{\mathbf{x}} p(\mathbf{u}) d\mathbf{u} = \int_{\mathbb{R}^{n_x}}^{\mathbf{x}} \mathcal{N}(\mathbf{u}; \hat{\mathbf{x}}, \mathbf{P}^{\mathbf{x}\mathbf{x}}) d\mathbf{u} \\
 &= (2\pi)^{-n_x/2} |\mathbf{P}^{\mathbf{x}\mathbf{x}}|^{-1/2} \\
 &\times \int_{\mathbb{R}^{n_x}}^{\mathbf{x}} \exp \left\{ -\frac{1}{2} [\mathbf{u} - \hat{\mathbf{x}}]^\top (\mathbf{P}^{\mathbf{x}\mathbf{x}})^{-1} [\mathbf{u} - \hat{\mathbf{x}}] \right\} d\mathbf{u} \quad (2.96)
 \end{aligned}$$

Now, let

$$\mathbf{t}(\mathbf{u}) \triangleq \mathbf{D}^{-1} (\mathbf{u} - \hat{\mathbf{x}}) \quad (2.97)$$

with \mathbf{D} defined by

$$\mathbf{P}^{\mathbf{x}\mathbf{x}} = \mathbf{D}\mathbf{D}^\top$$

Now $P(\mathbf{x})$ becomes

$$P(\mathbf{x}) = \prod_{j=1}^{n_x} \left\{ \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t_j(\mathbf{x})} e^{-t_j^2/2} dt_j \right\} \quad (2.98)$$

where t_j is the j th element of \mathbf{t} . The *one-dimensional* equation inside the bracket can be rewritten as

$$\begin{aligned} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t_j(\mathbf{x})} e^{-t_j^2/2} dt_j(\mathbf{u}) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^0 e^{-t_j^2/2} dt_j \\ &\quad + \frac{1}{\sqrt{2\pi}} \int_0^{t_j} e^{-t_j^2/2} dt_j \\ &= \frac{1}{2} \\ &\quad + \frac{1}{2} \left(\frac{2}{\sqrt{\pi}} \int_0^{t_j} e^{-t_j^2/2} dt_j \right) \end{aligned} \quad (2.99)$$

Noting the MATLAB® functional definition

$$\text{erf}[q] = \frac{2}{\sqrt{\pi}} \int_0^q e^{-u^2/2} du \quad (2.100)$$

then the *multidimensional* Gaussian cdf becomes

$$P(\mathbf{x}) = \prod_{j=1}^{n_x} \frac{1}{2} \{1 + \text{erf}[t_j]\} \quad (2.101)$$

Figures 2.7 and 2.8 show examples of one-dimensional and two-dimensional cumulative distribution functions, respectively, for the same means and variances used for the one- and two-dimensional pdfs shown above.

2.4.2.3 Transformation of the General Gaussian Moment Equations. Notice that in the moment equations (2.88) and (2.89), the integrals are of the form $\int \mathbf{x} p(\mathbf{x}) d\mathbf{x}$.

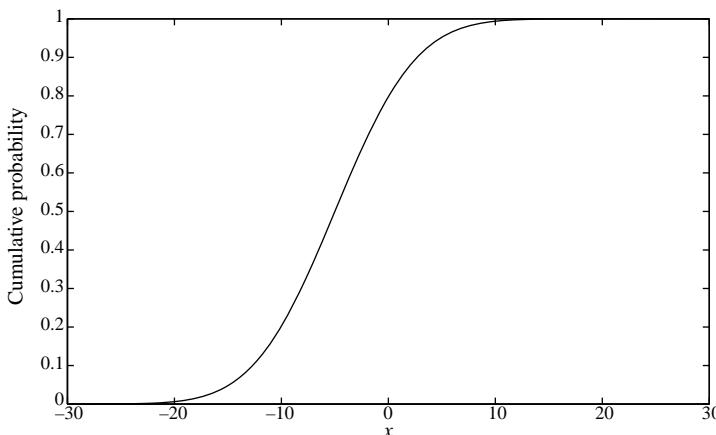


FIGURE 2.7 An example of a one-dimensional Gaussian cdf.

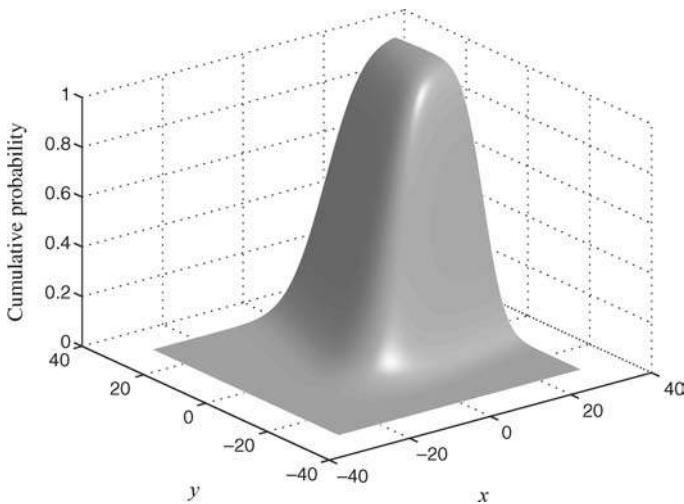


FIGURE 2.8 An example of a two-dimensional Gaussian cdf.

For the *one-dimensional* Gaussian case, consider the more general moment integral

$$\begin{aligned}\mathcal{E}\{f(x)\} &= \int_{-\infty}^{\infty} f(x) \mathcal{N}(x; \hat{x}, \sigma^2) dx \\ &= \int_{-\infty}^{\infty} f(x) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{1}{2\sigma^2} [x - \hat{x}]^2 \right\} \right) dx\end{aligned}\quad (2.102)$$

Now, apply the one-dimensional affine transformation

$$x = \hat{x} + \sigma c \quad (2.103)$$

This transformation moves the coordinate origin to \hat{x} and scales the standard deviation so that $\mathcal{N}(x; \hat{x}, \sigma^2) \rightarrow \mathcal{N}(c; 0, 1) / \sigma$. From this, it can be understood that c is a zero-mean, unit standard deviation random variable.

Letting

$$\tilde{f}(c) \triangleq f(\hat{x} + \sigma c) \quad (2.104)$$

and noting that $dx \rightarrow \sigma dc$, the general moment integral (2.102) becomes

$$\mathcal{E}\{f(x)\} = \int_{-\infty}^{\infty} \tilde{f}(c) \mathcal{N}(c; 0, 1) dc \quad (2.105)$$

Similarly, for the multidimensional case, consider the general moment integral

$$\begin{aligned}\mathcal{E}\{\mathbf{f}(\mathbf{x})\} &= \int_{-\infty}^{\infty} \mathbf{f}(\mathbf{x}) \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\mathbf{xx}}) d\mathbf{x} \\ &= \int_{-\infty}^{\infty} \mathbf{f}(\mathbf{x}) (2\pi)^{-n_x/2} |\mathbf{P}^{\mathbf{xx}}|^{-1/2} \\ &\quad \times \exp \left\{ -\frac{1}{2} [\mathbf{x} - \hat{\mathbf{x}}]^\top (\mathbf{P}^{\mathbf{xx}})^{-1} [\mathbf{x} - \hat{\mathbf{x}}] \right\} d\mathbf{x}\end{aligned}\quad (2.106)$$

Now apply the multidimensional affine transformation

$$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{D}\mathbf{c} \quad (2.107)$$

where \mathbf{D} is defined by the matrix square root equation

$$\mathbf{P}^{\mathbf{xx}} = \mathbf{D}\mathbf{D}^\top \quad (2.108)$$

leading to the identity

$$\mathbf{D} = [\mathbf{P}^{\mathbf{xx}}]^{1/2} \quad (2.109)$$

Here, \mathbf{D} can be the Cholesky factor of $\mathbf{P}^{\mathbf{xx}}$ or the principal (positive definite) matrix root of $\mathbf{P}^{\mathbf{xx}}$. Actually, any matrix \mathbf{D} that satisfies the definition (2.108) will suffice. In addition, \mathbf{c} can be identified as a zero-mean multidimensional random variable with a unit-diagonal covariance matrix, that is, $\mathbf{c} \sim \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$.

The affine transformation (2.107) has a geometric interpretation. The equiprobability contours of a multidimensional Gaussian distribution are ellipsoids centered at the mean. The direction of the principle axis are given by the eigenvectors of the covariance matrix, with eigenvalues equal to the squared relative lengths of the principal axes. The transformation (2.107) moves the coordinate origin to $\hat{\mathbf{x}}$ and rotates the coordinate axes so that they lie along the principal axes of the density function, with each axis scaled to a standard deviation of one, as shown in Figure 2.9. Note that the contours of the transformed density are circular rather than elliptical.

Noting that

$$\int_{-\infty}^{\infty} \mathbf{f}(\mathbf{x}) \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\mathbf{xx}}) d\mathbf{x} = \int_{-\infty}^{\infty} \mathbf{f}(\mathbf{x}(\mathbf{c})) \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\mathbf{xx}}) \left| \frac{d\mathbf{x}}{d\mathbf{c}} \right| d\mathbf{c} \quad (2.110)$$

and applying (2.110) and (2.107) to (2.106) we obtain

$$\mathcal{E}\{\mathbf{f}(\mathbf{x})\} = \int_{-\infty}^{\infty} \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (2.111)$$

where

$$\tilde{\mathbf{f}}(\mathbf{c}) \triangleq \mathbf{f}(\hat{\mathbf{x}} + \mathbf{D}\mathbf{c}) \quad (2.112)$$

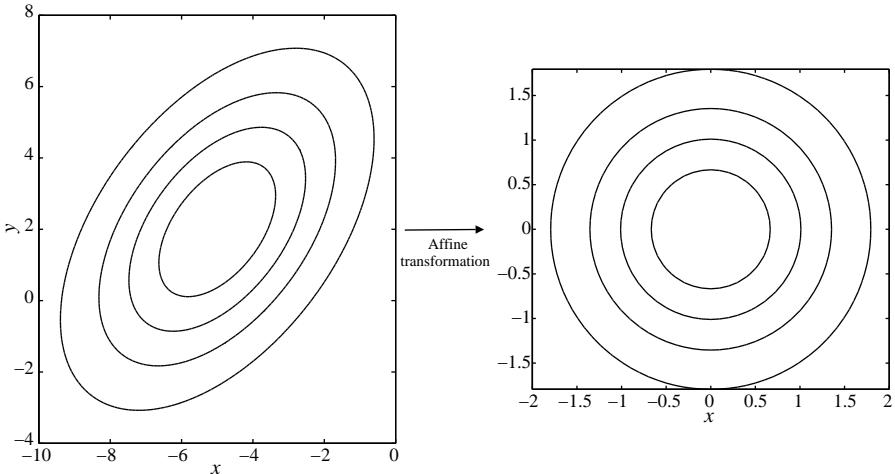


FIGURE 2.9 Effects of an affine transformation on a Gaussian probability density function.

Application of such an affine transformation is a most important concept that will be utilized to simplify the numerical evaluation of the Gaussian-weighted integrals that constitute the moment equations of Bayesian estimation. Use of such an affine transformation, in combination with the polynomial expansion of nonlinear functions discussed in Section 2.3, simplifies the numerical evaluation of these integrals.

2.4.2.4 The Gaussian Moment Equations. In this section, we will present, without derivation, some identities applicable to Gaussian moment equations.

For a random variable $\mathbf{x} \in \mathbb{R}^{n_x}$, and a symmetric matrix $\Lambda \in \mathbb{R}^{n_x \times n_x}$, the scalar quantity $\mathbf{x}^\top \Lambda \mathbf{x}$ is known as a quadratic form in \mathbf{x} . It can be shown that [20,21]

$$\mathcal{E}\{\mathbf{x}^\top \Lambda \mathbf{x}\} = \text{trace}\{\Lambda \mathbf{P}\} + \hat{\mathbf{x}}^\top \Lambda \hat{\mathbf{x}} \quad (2.113)$$

where $\hat{\mathbf{x}}$ and \mathbf{P} are the expected value and covariance matrix of \mathbf{x} , respectively, and $\text{trace}\{\bullet\}$ denotes the trace of the matrix contained within the bracket. This result only requires the existence of $\hat{\mathbf{x}}$ and \mathbf{P} , but is independent of the form of the probability density function. It can also be shown that the expected value of a quartic form in \mathbf{x} is given by [20]

$$\mathcal{E}\{\mathbf{x}^\top \Lambda_1 \mathbf{x} \mathbf{x}^\top \Lambda_2 \mathbf{x}\} = \text{trace}\{\Lambda_1 \mathbf{P}\} \text{trace}\{\Lambda_2 \mathbf{P}\} + 2\text{trace}\{\Lambda_1 \mathbf{P} \Lambda_2 \mathbf{P}\} \quad (2.114)$$

For the one-dimensional case, the Gaussian moment equations will contain integrals of the form $\int_{-\infty}^{\infty} c^n \mathcal{N}(c; 0, 1) dc$. Since $\hat{c} = 0$, and since $\mathcal{N}(c; 0, 1)$ is symmetric about zero, the moment equations for c^n are given by

$$\mathcal{E}\{c^n\} = \int_{-\infty}^{\infty} c^n \mathcal{N}(c; 0, 1) dc = \begin{cases} 1 & n = 0 \\ 1 \cdot 3 \cdot 5 \cdots (n-1) & n \text{ even} \\ 0 & n \text{ odd} \end{cases} \quad (2.115)$$

Using the one-dimensional moment equations (2.115) it follows immediately that the moment equations for the components of $\mathbf{c} = [c_1, c_2, \dots, c_{n_x}]^\top$, up to fourth order, are given by

$$\int_{\mathbb{R}^{n_x}} \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \int_{-\infty}^{\infty} \mathcal{N}(c_1; 0, 1) dc_1 \cdots \int_{-\infty}^{\infty} \mathcal{N}(c_{n_x}; 0, 1) dc_{n_x} = 1 \quad (2.116)$$

$$\int c_i \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \int c_i \mathcal{N}(c_i; 0, 1) dc_i \int \mathcal{N}(\mathbf{c}^*; \mathbf{0}, \mathbf{I}) d\mathbf{c}^* = 0 \quad (2.117)$$

$$\int c_i c_j \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \delta_{ij} \quad (2.118)$$

$$\begin{aligned} \int c_i c_j c_k \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} &= \delta_{ij} \int c_k \mathcal{N}(c_k; 0, 1) dc_k \\ &\quad + \delta_{ik} \int c_j \mathcal{N}(c_j; 0, 1) dc_j \\ &\quad + \delta_{jk} \int c_i \mathcal{N}(c_i; 0, 1) dc_i \\ &= 0 \end{aligned} \quad (2.119)$$

$$\int c_i^2 c_j^2 \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \begin{cases} 1, & i \neq j \\ 3, & i = j \end{cases} \quad (2.120)$$

$$\int c_i c_j c_k c_l \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \delta_{ij,kl} + \delta_{ik,jl} + \delta_{il,jk} + 3\delta_{ijkl} \quad (2.121)$$

where $\mathbf{c}^* \triangleq [c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_{n_x}]^\top$, the implied integration limits on all integrals have been dropped for convenience and we have defined

$$\delta_{ij} \triangleq \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \quad (2.122)$$

$$\delta_{ijkl} \triangleq \begin{cases} 1, & i = j = k = l \\ 0, & \text{elsewhere} \end{cases} \quad (2.123)$$

and

$$\delta_{ij,kl} \triangleq \begin{cases} 1, & i = j, k = l; i, j \neq k, l \\ 0, & \text{elsewhere} \end{cases} \quad (2.124)$$

with similar definitions for $\delta_{ik}\delta_{jl}$ and $\delta_{il}\delta_{jk}$.

From (2.113) and (2.114), it follows immediately that the quadratic moment equations for \mathbf{c} are given by

$$\int [\mathbf{c}^\top \mathbf{M} \mathbf{c}]^i \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \begin{cases} 1, & i = 0 \\ \text{trace}\{\mathbf{M}\}, & i = 1 \\ [\text{trace}\{\mathbf{M}\}]^2 + 2 \text{trace}\{\mathbf{M}\mathbf{M}\}, & i = 2 \end{cases} \quad (2.125)$$

where $\mathbf{M} \in \mathbb{R}^{n_x \times n_x}$ is a known symmetric square matrix.

From (2.116) to (2.121), it is easy to show that

$$\int \mathbf{c} \mathbf{c}^\top \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \mathbf{I} \quad (2.126)$$

2.4.2.5 Generating Samples from a Multidimensional Gaussian Distribution in Matlab®. Samples from a general multidimensional Gaussian distribution $\mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\mathbf{xx}})$ can be generated in Matlab in the following way:

1. Generate samples $\{\mathbf{c}^{(i)} \sim \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}), i = 1, \dots, n_x\}$. Here, the use of \sim indicates that the i th sample is drawn from the density $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$. In Matlab, this is accomplished with the command $\mathbf{c} = \text{randn}(N, M)$, where N is the dimension of \mathbf{c} and M is the number of samples desired.
2. Use (2.107) to transform the samples of \mathbf{c} into samples of \mathbf{x} . Thus,

$$\mathbf{x}^{(i)} = \hat{\mathbf{x}} + \mathbf{D}\mathbf{c}^{(i)} \quad (2.127)$$

with \mathbf{D} defined by $\mathbf{P}^{\mathbf{xx}} = \mathbf{D}\mathbf{D}^\top$.

REFERENCES

1. Brookes M. The Matrix Reference Manual. Available online at <http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/intro.html>. Accessed May 16, 2011.
2. Lerner UN. Hybrid Bayesian Networks for Reasoning About Complex Systems, Dissertation. Stanford, CA: Stanford University; 2002.
3. McNamee J, Stenger F. Construction of fully symmetric numerical integration formulas. *Numerische Math.* 1967;10:327–344.
4. Arfken G. *Mathematical Methods for Physicists*, 3rd ed. Academic Press; 1985.
5. Ralston A, Rabinowitz P. *A First Course in Numerical Analysis*, 2nd ed. Dover Press; 2001.
6. Williamson RE, Cromwell RH, Trotter, HF. *Calculus of Vector Functions*, 3rd ed. Prentice-Hall; 1972.
7. Julier S, Uhlmann, J. A General Method for Approximating Nonlinear Transformations of Probability Distributions. Department of Engineering Science, University of Oxford; 1996.
8. Nørgaard M, Poulsen NK, Ravn O. New developments in state estimation for nonlinear systems. *Automatica* 2000;36:1627–1638.
9. Burden, RL, Faires, JD. *Numerical Analysis*. Belmont: Brooks/Cole; 2000.

10. Silvester P, Ferrari R. *Finite Elements for Electrical Engineers*, 3rd ed. Cambridge University Press; 1996.
11. Waldron S. The error in linear interpolation at the vertices of a simplex. *SIAM J. Numer. Analy.* 1998;35(3):1191–1200.
12. Gasca M, Sauer T. Polynomial interpolation in several variables. *Adv. Comput. Math.* 2000;12(4):377–410.
13. Stämpfle M. Optimal estimates for the linear interpolation error on simplices. *J. Approx. Theory* 2000;103:78–90.
14. Lu Y, Wang RH. A multidimensional generalization of Simpson-type formulas over N-simplex. *J. Inform. Comput. Sci.* 2004;1(2):331–335.
15. Hemingway P. n-Simplex Interpolation. HPL-2002-320, Client and Media Systems Laboratory, HP laboratories, Bristol; November 2002.
16. Warburton T. An explicit construction for interpolation nodes on the simplex. *J. Eng. Math.* 2006;56(3):247–262.
17. Papoulis A. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill; 1965.
18. de Moivre A, Approximatio ad Summam Terminorum Binomii $(a + b)n$ in Seriem expansi. Printed on November 12, 1733.
19. [online] http://en.wikipedia.org/wiki/Normal_distribution.
20. Bar Shalom Y, Li XR, Kirubarajan T. *Estimation with Application to Tracking and Navigation: Theory, Algorithms and Software*. Wiley; 2001.
21. Koch KR. *Introduction to Bayesian Statistics*. New York: Springer; 2007.

3

GENERAL CONCEPTS OF BAYESIAN ESTIMATION

The process of estimation begins with an experiment that provides a set of observable outcomes, usually some form of data. Examples of observable data can include a time-sampled succession of bearings and/or ranges to a target or successive samples of a stock price for sales throughout a day of trading. Based on the observable data, one would like to estimate some characteristic parameters that may be unobservable directly. For example, with bearings-only observations one would like to estimate the target location and velocity as a function of time. In the case of stock price data, one would like to estimate the volatility of the stock.

This book concentrates on estimation methods that include models of the temporal dynamics of the variables to be estimated as well as models of the relationship between the observable data and the unobservable variables to be estimated. In particular, discussions will be limited to recursive estimation methods for discretely sampled data. Thus, it is always assumed that the parameters to be estimated follow a known recursive dynamic process and that there is a known analytical link between the observed data and the parameters to be estimated.

In addition, Bayesian estimation assumes that both the parameters to be estimated and the observed data are stochastic entities. The *analytical* link (a transformation) between the observed data and the parameters to be estimated provide a unifying framework for estimation where the recursive inference is characterized by a density function for the current state vector value conditioned on the current and all prior observations.

3.1 BAYESIAN ESTIMATION

Bayesian estimation has as its objective the estimation of successive values of a parameter vector \mathbf{x} given an observation vector \mathbf{z} . As noted above, it is customary to treat both \mathbf{x} and \mathbf{z} as random vectors. For the parameter vector, the stochastic assumption is inherent in the equations governing the dynamics of the parameter, where unmodeled effects are added as random noise. For the observation vector one can justify a stochastic nature by assuming that there is always some random measurement noise. The random vector \mathbf{x} is assumed to have a known prior density function $p(\mathbf{x})$. This prior distribution includes all that is known and unknown about the parameter vector prior to the availability of any observational data. If the true parameter value of \mathbf{x} were known, then the probability density of \mathbf{z} is given by the conditional density or likelihood function $p(\mathbf{z}|\mathbf{x})$ and the complete statistical properties of \mathbf{z} would be known.

Once an experiment has been conducted and a realization of the random variable \mathbf{z} is available, one can use Bayes' law to obtain the *posterior conditional density* of \mathbf{x} :

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{z}|\mathbf{x}) p(\mathbf{x})}{p(\mathbf{z})} \quad (3.1)$$

Thus, within the Bayesian framework, the posterior density $p(\mathbf{x}|\mathbf{z})$ contains everything there is to know about \mathbf{x} after taking into account the observational outcome of an experiment. Since the experimental outcome \mathbf{z} is now available, the denominator of (3.1) is just a scalar normalizing constant that can be found from

$$p(\mathbf{z}) = \int_{\mathbb{R}^{n_x}} p(\mathbf{z}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \quad (3.2)$$

For the full Bayesian estimation problem, the likelihood and the posterior, or alternately their joint density, define the statistical model for estimation, where the joint density of the parameter and observational vectors is defined by

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \quad (3.3)$$

The *solution* to the estimation problem is found in the posterior distribution given by (3.1). Consequently, the posterior distribution can be used to generate any point estimates of \mathbf{x} that are desired, if they exist. Note that the posterior density should be regarded as the most general solution to the estimation problem and in many cases the density function can be used to characterize \mathbf{x} .

3.2 POINT ESTIMATORS

Evaluating a posterior density can sometimes be a daunting prospect and is a complicated solution to the multidimensional inference problem. Consider a point estimate $\hat{\mathbf{x}}$,

which is an educated guess of the parameter value given the observations at hand. An analytical method to generate a point estimate $\hat{\mathbf{x}}$ based on all available observations can be designated as an estimator of \mathbf{x} . The actual value obtained for $\hat{\mathbf{x}}$ will vary based on the estimator used. In addition, it will vary from experiment to experiment using the same estimator. Thus, the point estimate should itself be viewed as a stochastic variable.

To find a suitable estimator we define a cost function, $L(\hat{\mathbf{x}}, \mathbf{x})$, which defines a penalty for an erroneous estimate $\hat{\mathbf{x}} \neq \mathbf{x}$. In general, we would prefer cost functions where the penalty increases based on the magnitude of the difference error $\mathbf{x} - \hat{\mathbf{x}}$. Without loss of generality, we assume that the cost function is positive and that $L(\mathbf{x}, \mathbf{x}) = 0$ is its unique minimum. For a point estimate $\hat{\mathbf{x}}$, the Bayesian risk R is defined as the expected value of the cost function

$$R \triangleq \mathcal{E}\{L(\hat{\mathbf{x}}, \mathbf{x})\} \quad (3.4)$$

where the expectation is over \mathbf{x} and \mathbf{z} . Here we note that since we have used the observations \mathbf{z} to generate $\hat{\mathbf{x}}$, we can write $\hat{\mathbf{x}}(\mathbf{z})$. The optimal choice of $\hat{\mathbf{x}}(\mathbf{z})$ is the one that minimizes the Bayesian risk

$$\hat{\mathbf{x}}(\mathbf{z}) = \arg \min_{\mathbf{x}^*(\mathbf{z})} \int_{\mathbb{R}^{n_z+n_x}} L(\mathbf{x}^*(\mathbf{z}), \mathbf{x}) p(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z} \quad (3.5)$$

where the minimization is over all values of $\mathbf{x}^*(\mathbf{z})$ and the dimensions of \mathbf{z} and \mathbf{x} are n_z and n_x , respectively.

Now, inserting (3.3) splits the integration into two parts

$$\hat{\mathbf{x}}(\mathbf{z}) = \arg \min_{\mathbf{x}^*(\mathbf{z})} \int_{\mathbb{R}^{n_z}} \left[\int_{\mathbb{R}^{n_x}} L(\mathbf{x}^*(\mathbf{z}), \mathbf{x}) p(\mathbf{x}|\mathbf{z}) d\mathbf{x} \right] p(\mathbf{z}) d\mathbf{z} \quad (3.6)$$

Since both $L(\mathbf{x}^*, \mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$ are positive, the inner integral is positive given any \mathbf{z} . Since $p(\mathbf{z})$ is also positive, the value of $\mathbf{x}^*(\mathbf{z})$ that minimizes the risk also minimizes the inner integral,

$$\hat{\mathbf{x}}(\mathbf{z}) = \arg \min_{\mathbf{x}^*(\mathbf{z})} \int_{\mathbb{R}^{n_x}} L(\mathbf{x}^*(\mathbf{z}), \mathbf{x}) p(\mathbf{x}|\mathbf{z}) d\mathbf{x} \quad (3.7)$$

for each \mathbf{z} . The Bayesian estimation problem is therefore defined by (3.7) where each choice for a cost function gives rise to a different estimator.

An alternative approach could be to choose as our estimator one that minimizes the maximal cost (min max estimator) such that

$$\hat{\mathbf{x}}(\mathbf{z}) = \arg \min_{\mathbf{x}^*(\mathbf{z})} \max \{L(\mathbf{x}^*(\mathbf{z}), \mathbf{x})\} \quad (3.8)$$

For most estimation problems, it is realistic to assume that the cost is based on the estimation error given by

$$\boldsymbol{\epsilon}^x \triangleq \mathbf{x} - \hat{\mathbf{x}} \quad (3.9)$$

The cost function $L(\boldsymbol{\epsilon}^x)$ is a function of a single variable.

Another typical cost function includes the weighted squared error cost function

$$L(\boldsymbol{\epsilon}^x) = \boldsymbol{\epsilon}^{x\top} \mathbf{M} \boldsymbol{\epsilon}^x \quad (3.10)$$

which accentuates the effects of large errors. Here, \mathbf{M} is some known square weighting matrix that does not depend on \mathbf{x} .

An additional method is to use the absolute error value cost function

$$L(\boldsymbol{\epsilon}^x) = |\boldsymbol{\epsilon}^x| \quad (3.11)$$

Another common choice is a cost function that is uniform across some region and zero outside that region. For example, we could choose $L(\boldsymbol{\epsilon}^x)$ such that

$$L(\boldsymbol{\epsilon}^x) = \begin{cases} 0, & |\boldsymbol{\epsilon}^x| \leq \Delta/2 \\ 1, & |\boldsymbol{\epsilon}^x| > \Delta/2 \end{cases} \quad (3.12)$$

The specific cost function chosen depends on what we wish to accomplish with the estimation method. Frequently, it is difficult to assign an analytic measure to what may be a subjective quantity. As noted above, our goal will be to find an estimate that minimizes the expected value of the cost.

First let us consider minimizing the expected value of the weighted squared error cost function, usually called the mean squared error criterion. Using \mathbf{x}^* instead of $\hat{\mathbf{x}}$ in (3.9), putting the results into (3.10), (3.7) becomes

$$\hat{\mathbf{x}}(\mathbf{z}) = \arg \min_{\mathbf{x}^*(\mathbf{z})} \int_{\mathbb{R}^{n_x}} (\mathbf{x} - \mathbf{x}^*)^\top \mathbf{M} (\mathbf{x} - \mathbf{x}^*) p(\mathbf{x}|\mathbf{z}) d\mathbf{x} \quad (3.13)$$

To obtain the minimum argument we take the derivative with respect to \mathbf{x}^* , set the results to zero and solve for \mathbf{x}^* . That is

$$\begin{aligned} & \frac{d}{d\mathbf{x}^*} \int_{\mathbb{R}^{n_x}} (\mathbf{x} - \mathbf{x}^*)^\top \mathbf{M} (\mathbf{x} - \mathbf{x}^*) p(\mathbf{x}|\mathbf{z}) d\mathbf{x} \\ &= -2\mathbf{M} \int_{\mathbb{R}^{n_x}} \mathbf{x} p(\mathbf{x}|\mathbf{z}) d\mathbf{x} + 2\mathbf{M}\mathbf{x}^* \int_{\mathbb{R}^{n_x}} p(\mathbf{x}|\mathbf{z}) d\mathbf{x} \end{aligned} \quad (3.14)$$

Setting this to zero and noting that the second integral equals 1, we have

$$\hat{\mathbf{x}}_{\text{ms}} = \int_{\mathbb{R}^{n_x}} \mathbf{x} p(\mathbf{x}|\mathbf{z}) d\mathbf{x} \triangleq \mathcal{E}\{\mathbf{x}|\mathbf{z}\} \quad (3.15)$$

Thus, minimizing the mean squared error cost function results in an estimate that is the mean relative to the posteriori density (or the conditional mean).

Since the estimates obtained from a mean-squared estimator are themselves random variables, one would like to quantify the uncertainty in these estimates. A convenient measure of uncertainty is given by the conditional covariance matrix

$$\begin{aligned}\mathbf{P}^{\mathbf{x}\mathbf{x}} &= \mathcal{E}\left\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T | \mathbf{z}\right\} \\ &= \int_{\mathbb{R}^{n_x}} (\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T p(\mathbf{x}|\mathbf{z}) d\mathbf{x}\end{aligned}\quad (3.16)$$

However, it must be noted that for some probability density functions, one or more of these point estimates do not exist. The Cauchy distribution is an example of a distribution that has no mean, covariance or higher order moments. However, its mode and median are well defined and the covariance can be replaced by a dispersion measure.

In this book, most of the attention is devoted to estimation methods that utilize the mean squared cost function. The absolute error and uniform cost functions lead to what is known as the maximum *a posteriori* (MAP) estimator and a special case of the MAP estimator called the maximum likelihood estimator. For a complete derivation and discussion of these estimators, see Ref. [1], pages 56–58.

3.3 INTRODUCTION TO RECURSIVE BAYESIAN FILTERING OF PROBABILITY DENSITY FUNCTIONS

A discrete dynamic process will be defined as a process where the current state of the system is dependent on one or more prior states. In continuous processes, the dependence of the current state on previous states is captured within a differential equation. When observations occur at discrete times, estimation conditioned on those observations can only occur at those time, so the differential equation is replaced by its finite difference equivalent that links the state at observation time t_n to states at observation times prior to t_n .

A first-order Markov process is one in which the current state is dependent only on the previous state. Thus, we can characterize a discrete random Markov dynamic process as

$$\mathbf{x}_n = \mathbf{f}_{n-1}(\mathbf{x}_{n-1}, \mathbf{u}_n, \boldsymbol{\eta}_{n-1}) = \mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n + \mathbf{v}_{n-1} \quad (3.17)$$

where \mathbf{x}_n is the state of the system (*state vector*) at time t_n , \mathbf{f}_{n-1} is a *deterministic* transition function (matrix) that moves the state \mathbf{x} from time t_{n-1} to time t_n and \mathbf{u}_n is a known (usually deterministic) control that constitutes some external input that drives the system dynamics.

Although the white noise $\boldsymbol{\eta}$ (not necessarily Gaussian) can start at the input and be transformed by the transition function, it is usually assumed that the noise is additive and represents those parts of the true transition function that are not modeled. Note

that the Markov process given above is only a model of the true transition function. For example, if the motion of a moving ship is modeled as constant velocity motion, the noise term will represent the accelerations that cause small deviations from a straight line path caused by wave action.

The specific problem of interest is estimating the usually unobservable state vector \mathbf{x}_n based on the set of all experimental observation vectors $\mathbf{z}_{1:n} \triangleq \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$. It will be assumed that an analytical link is known between the observation vector at time t_n and the state vector at time t_n that is represented by

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_n, \boldsymbol{\mu}_n) = \mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_n \quad (3.18)$$

Here, \mathbf{z}_n is designated as the *observation vector* and \mathbf{h}_n is a *deterministic* observation function that links the state vector with the observation. Once again, the white noise $\boldsymbol{\mu}_n$ (not necessarily Gaussian) can be transformed by the transition function, but it is usually assumed that the observation noise is additive.

For most problems of interest, both \mathbf{f}_{n-1} and \mathbf{h}_n are considered adiabatic, changing very slowly with time. They can therefore be considered time independent, that is, $\mathbf{f}_{n-1} \rightarrow \mathbf{f}$ and $\mathbf{h}_n \rightarrow \mathbf{h}$.

The specific analytical form that (3.17) and (3.18) take represent a complete *model* of the system for which estimated information about the state vector is desired. The estimation of \mathbf{x}_n based on the complete set of observations $\mathbf{z}_{1:n}$ can be understood in a Bayesian sense by turning the problem into an estimation of the conditional posterior density $p(\mathbf{x}_n | \mathbf{z}_{1:n})$. The goal of this book is to develop recursive Bayesian methods for estimation of this conditional distribution and the moments associated with \mathbf{x}_n under this posterior distribution.

In terms of the posterior distribution $p(\mathbf{x}_n | \mathbf{z}_{1:n})$, Bayes' law (3.1) can be rewritten as

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{p(\mathbf{z}_{1:n} | \mathbf{x}_n) p(\mathbf{x}_n)}{p(\mathbf{z}_{1:n})} \quad (3.19)$$

The posterior pdf, $p(\mathbf{x}_n | \mathbf{z}_{1:n})$, is the pdf of \mathbf{x}_n conditioned on all observations up to and including the current observation. Since the set $\{\mathbf{z}_{1:n}\}$ can be written as $\{\mathbf{z}_n, \mathbf{z}_{1:n-1}\}$, (3.19) can be expanded and factored into the following form

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{p(\mathbf{z}_n, \mathbf{z}_{1:n-1} | \mathbf{x}_n) p(\mathbf{x}_n)}{p(\mathbf{z}_n, \mathbf{z}_{1:n-1})} \quad (3.20)$$

Using (3.3) in both the numerator and denominator, this becomes

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{p(\mathbf{z}_n | \mathbf{z}_{1:n-1}, \mathbf{x}_n) p(\mathbf{z}_{1:n-1} | \mathbf{x}_n) p(\mathbf{x}_n)}{p(\mathbf{z}_n | \mathbf{z}_{1:n-1}) p(\mathbf{z}_{1:n-1})} \quad (3.21)$$

Applying Bayes law (3.1) to $p(\mathbf{z}_{1:n-1}|\mathbf{x}_n)$, and reducing the resulting equations, yields the following progression

$$\begin{aligned} p(\mathbf{x}_n|\mathbf{z}_{1:n}) &= \frac{p(\mathbf{z}_n|\mathbf{z}_{1:n-1}, \mathbf{x}_n) p(\mathbf{x}_n|\mathbf{z}_{1:n-1}) p(\mathbf{z}_{1:n-1}) p(\mathbf{x}_n)}{p(\mathbf{z}_n|\mathbf{z}_{1:n-1}) p(\mathbf{z}_{1:n-1}) p(\mathbf{x}_n)} \\ &= \frac{p(\mathbf{z}_n|\mathbf{z}_{1:n-1}, \mathbf{x}_n) p(\mathbf{x}_n|\mathbf{z}_{1:n-1})}{p(\mathbf{z}_n|\mathbf{z}_{1:n-1})} \end{aligned} \quad (3.22)$$

$$= \frac{p(\mathbf{z}_n|\mathbf{x}_n) p(\mathbf{x}_n|\mathbf{z}_{1:n-1})}{p(\mathbf{z}_n|\mathbf{z}_{1:n-1})} \quad (3.23)$$

where $p(\mathbf{z}_n|\mathbf{z}_{1:n-1}, \mathbf{x}_n) \rightarrow p(\mathbf{z}_n|\mathbf{x}_n)$ because, from (3.18), the observation at time t_n does not depend on the observations at times $t_{1:n-1}$.

One last step is needed to create a completely recursive form for the conditional probability density function equations. The Chapman–Kolmogorov equation provides a link between the *prior density*, defined as $p(\mathbf{x}_n|\mathbf{z}_{1:n-1})$, and the previous posterior density

$$\begin{aligned} p(\mathbf{x}_n|\mathbf{z}_{1:n-1}) &= \int p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{z}_{1:n-1}) p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \\ &= \int p(\mathbf{x}_n|\mathbf{x}_{n-1}) p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \end{aligned} \quad (3.24)$$

Now, from (3.24) and (3.23), a recursive link has been established between the previous posterior $p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1})$ and the current posterior $p(\mathbf{x}_n|\mathbf{z}_{1:n})$ that requires specification of the *predictive density* given by $p(\mathbf{x}_n|\mathbf{x}_{n-1})$ and the *likelihood function* $p(\mathbf{z}_n|\mathbf{x}_n)$.

In (3.24), $p(\mathbf{x}_n|\mathbf{x}_{n-1})$ has replaced $p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{z}_{1:n-1})$ because this predictive density is defined by the dynamic equation specified by (3.17) that does not depend on $\mathbf{z}_{1:n-1}$. Examination of (3.17) reveals that the density of \mathbf{x}_n is dependent of both \mathbf{x}_{n-1} and \mathbf{v}_{n-1} . If \mathbf{v}_{n-1} is considered to be zero-mean, then the mean value of \mathbf{x}_n will be the mean value of $\mathbf{f}(\mathbf{x}_{n-1}, \mathbf{u}_n)$.

In addition, from (3.18), \mathbf{z}_n can be seen as a random vector whose current value is dependent on \mathbf{x}_n . Therefore, (3.1) defines the normalized likelihood function $p(\mathbf{z}_n|\mathbf{x}_n)/p(\mathbf{z}_n)$. From (3.1) we see that since $p(\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$ are densities, $p(\mathbf{z}_n|\mathbf{x}_n)/p(\mathbf{z}_n)$ must also be a density. Therefore, $p(\mathbf{z}_n)$ is the normalizing function for $p(\mathbf{z}_n|\mathbf{x}_n)$.

One iteration in this Bayesian recursive procedure for developing successive posterior densities is shown in Figure 3.1. The recursive procedure is initiated by $p(\mathbf{x}_0)$, the pdf associated with \mathbf{x} prior to any observations.

As an aside, in many theoretical treatments of this subject, the Markov process (3.17) is considered as part of a Markov Chain $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, and the prediction and update equations, (3.24) and (3.23), respectively, are written in terms of the Markov Chain [2]. Since this book deals primarily with tracking applications, in which the Markov Chain reduces to a first-order Markov process, our developments will ignore the use of the Markov Chain notation in favor of a one-step recursion

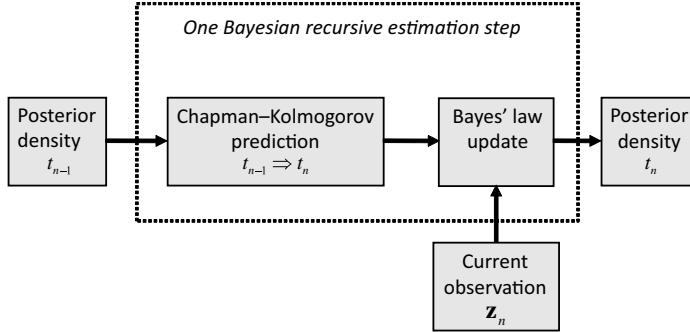


FIGURE 3.1 Depiction of one step in the recursive Bayesian posterior density estimation procedure.

formulation. For those cases where the dynamic model Markov process is of higher order, for example, analysis of time series data, one can usually create an augmented state vector that results in a first-order Markov process.

3.4 INTRODUCTION TO RECURSIVE BAYESIAN ESTIMATION OF THE STATE MEAN AND COVARIANCE

Introducing a notation that will be used throughout the remainder of this text, let an estimate of \mathbf{x}_n conditioned on all observations up to time t_p be written as $\hat{\mathbf{x}}_{n|p}$, with

$$\hat{\mathbf{x}}_{n|p} = \mathcal{E}\{\mathbf{x}_n | \mathbf{z}_{1:p}\} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{x}_n p(\mathbf{x}_n | \mathbf{z}_{1:p}) d\mathbf{x}_n \quad (3.25)$$

Thus, an estimate of \mathbf{x}_n that uses all observations, including the current one at time t_n , is written as

$$\hat{\mathbf{x}}_{n|n} = \mathcal{E}\{\mathbf{x}_n | \mathbf{z}_{1:n}\} = \int_{\mathbb{R}^{n_x}} \mathbf{x}_n p(\mathbf{x}_n | \mathbf{z}_{1:n}) d\mathbf{x}_n \quad (3.26)$$

while one that *predicts* \mathbf{x}_n based on all but the current observation is given by

$$\hat{\mathbf{x}}_{n|n-1} = \mathcal{E}\{\mathbf{x}_n | \mathbf{z}_{1:n-1}\} = \int_{\mathbb{R}^{n_x}} \mathbf{x}_n p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n \quad (3.27)$$

The same can be done for the covariance matrix, writing

$$\begin{aligned} \mathbf{P}_{n|n}^{\mathbf{xx}} &= \mathcal{E}\{(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top | \mathbf{z}_{1:n}\} \\ &= \int_{\mathbb{R}^{n_x}} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top p(\mathbf{x}_n | \mathbf{z}_{1:n}) d\mathbf{x}_n \end{aligned} \quad (3.28)$$

with *prediction* form of the covariance given by

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \mathcal{E} \left\{ (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^\top | \mathbf{z}_{1:n-1} \right\} \\ &= \int_{\mathbb{R}^{n_x}} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^\top \\ &\quad \times p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n\end{aligned}\tag{3.29}$$

3.4.1 State Vector Prediction

Using the Chapman–Kolmogorov equation (3.24) in (3.27) and (3.29) results in

$$\hat{\mathbf{x}}_{n|n-1} = \int_{\mathbb{R}^{n_x}} \int_{\mathbb{R}^{n_x}} \mathbf{x}_n p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} d\mathbf{x}_n\tag{3.30}$$

and

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int_{\mathbb{R}^{n_x}} \int_{\mathbb{R}^{n_x}} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^\top p(\mathbf{x}_n | \mathbf{x}_{n-1}) \\ &\quad \times p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} d\mathbf{x}_n\end{aligned}\tag{3.31}$$

From the system dynamic equation (3.17), (3.30) can be rewritten as

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \int_{\mathbb{R}^{n_x}} \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n + \mathbf{v}_{n-1}] p(\mathbf{x}_n | \mathbf{x}_{n-1}) \\ &\quad \times p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} d\mathbf{x}_n \\ &= \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n + \mathbf{v}_{n-1}] p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \\ &\quad \times \int_{\mathbb{R}^{n_x}} p(\mathbf{x}_n | \mathbf{x}_{n-1}) d\mathbf{x}_n\end{aligned}\tag{3.32}$$

$$\begin{aligned}&= \int_{\mathbb{R}^{n_x}} \mathbf{f}_{n-1}(\mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} + \mathbf{u}_n \\ &\quad + \int_{\mathbb{R}^{n_x}} \mathbf{v}_{n-1} p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1}\end{aligned}\tag{3.33}$$

In moving from (3.32) to (3.33), the integral $\int p(\mathbf{x}_n | \mathbf{x}_{n-1}) d\mathbf{x}_n = 1$, since the integral of a density function over the entire support region of that density is always 1. Equation (3.33) consists of two density-weighted integrals that usually cannot be

solved analytically, except in special cases. If the noise \mathbf{v}_{n-1} is zero-mean, a requirement for an unbiased estimator, the second integral reduces to zero.

Similarly, (3.31) reduces to

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \\ &\quad - \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}] \mathbf{v}_{n-1}^\top p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \\ &\quad - \int_{\mathbb{R}^{n_x}} \mathbf{v}_{n-1} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}]^\top p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \\ &\quad + \int_{\mathbb{R}^{n_x}} \mathbf{v}_{n-1} \mathbf{v}_{n-1}^\top p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \end{aligned} \quad (3.34)$$

Defining the noise covariance as

$$\mathbf{Q} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{v}_{n-1} \mathbf{v}_{n-1}^\top p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \quad (3.35)$$

and assuming that $\mathbf{f}_{n-1}(\mathbf{x}_{n-1}, \mathbf{u}_n)$ is uncorrelated with \mathbf{v}_{n-1} , the middle two integrals in (3.34) reduce to zero, and the density-weighted predictive covariance becomes

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} + \mathbf{Q} \end{aligned} \quad (3.36)$$

Thus, (3.33) and (3.36) are the predictive estimate of the first-and second-order moments of the state vector conditioned on all but the most current observation. What is needed now is a method for recursively updating these predictive moments, given the current observation of \mathbf{z}_n , so as to obtain corrected estimates of the first two moments of the posterior density without the need to evaluate (3.26) and (3.28) directly.

3.4.2 State Vector Update

For any general density, Kalman developed such a filter without using Bayes' Law [3]. He assumed that the system state \mathbf{x}_n could be estimated *sequentially* by updating only

the first and second order predictive moments. Specifically, he *assumed* that the updating estimate for the first moment be *linearly* dependent on the current *observation*, that is,

$$\hat{\mathbf{x}}_{n|n} = \mathbf{A}\mathbf{z}_n^0 + \mathbf{b} \quad (3.37)$$

where \mathbf{z}_n^0 is the *latest* noisy observation and \mathbf{A} and \mathbf{b} are unknowns that are to be determined.

The best linear estimate (in the minimum mean squared error (MMSE) sense) of the random variable \mathbf{x}_n in terms of a current observation \mathbf{z}_n^0 (which is also a random variable) is such that

1. the conditional estimate is unbiased, that is, each component of the *estimation error* (3.9) has zero mean, i.e., $\mathcal{E}\{\epsilon_n^x | \mathbf{z}_{1:n-1}\} = \mathbf{0} \in \mathbb{R}^{n_x}$, and
2. each component of the *estimation error* is uncorrelated with all components of the observation, that is, $\mathcal{E}\{\epsilon_n^x \mathbf{z}_n^{0\top} | \mathbf{z}_{1:n-1}\} = \mathbf{0} \in \mathbb{R}^{n_x \times n_z}$.

Note that both conditions are on the estimation error and not on the state or observation vectors themselves. Also, the latter condition requires that the estimation error be orthogonal to the observations.

Using the definition of ϵ_n^x from (3.9), and also using equation (3.37), then applying condition 1 leads to

$$\begin{aligned} \mathcal{E}\{\epsilon_n^x | \mathbf{z}_{1:n-1}\} &= \mathcal{E}\{\mathbf{x}_n - \hat{\mathbf{x}}_{n|n} | \mathbf{z}_{1:n-1}\} \\ &= \mathcal{E}\{\mathbf{x}_n | \mathbf{z}_{1:n-1}\} - (\mathbf{A}\mathcal{E}\{\mathbf{z}_n^0 | \mathbf{z}_{1:n-1}\} + \mathbf{b}) \\ &= \hat{\mathbf{x}}_{n|n-1} - \mathbf{A}\hat{\mathbf{z}}_{n|n-1} - \mathbf{b} = \mathbf{0} \end{aligned} \quad (3.38)$$

Thus

$$\mathbf{b} = \hat{\mathbf{x}}_{n|n-1} - \mathbf{A}\hat{\mathbf{z}}_{n|n-1} \quad (3.39)$$

From (3.37), follows immediately that

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{A}(\mathbf{z}_n^0 - \hat{\mathbf{z}}_{n|n-1}) \quad (3.40)$$

Condition 2, the orthogonality condition, requires that

$$\mathcal{E}\{\epsilon_n^x \mathbf{z}_n^{0\top} | \mathbf{z}_{1:n-1}\} = \mathcal{E}\{[\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} - \mathbf{A}(\mathbf{z}_n^0 - \hat{\mathbf{z}}_{n|n-1})] \mathbf{z}_n^{0\top}\} \quad (3.41)$$

Note that $\hat{\mathbf{x}}_{n|n-1}$ and $\hat{\mathbf{z}}_{n|n-1}$ are conditioned on $\mathbf{z}_{1:n-1}$. Since $\mathcal{E}\{\boldsymbol{\epsilon}_n^x\} = \mathbf{0}$, it follows that $\mathcal{E}\{\boldsymbol{\epsilon}_n^x \mathbf{z}_n^{o\top}\} = \mathcal{E}\{\boldsymbol{\epsilon}_n^x (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1})^\top\}$, and (3.41) can be written as

$$\begin{aligned}\mathcal{E}\{\boldsymbol{\epsilon}_n^x \mathbf{z}_n^{o\top} | \mathbf{z}_{1:n-1}\} &= \mathcal{E}\left\{\left[\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} - \mathbf{A}(\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1})\right] \times (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1})^\top | \mathbf{z}_{1:n-1}\right\} \\ &= \mathbf{P}_{n|n-1}^{xz} - \mathbf{A} \mathbf{P}_{n|n-1}^{zz} = \mathbf{0}\end{aligned}\quad (3.42)$$

Solving the last equation for \mathbf{A} yields

$$\mathbf{A} = \mathbf{K}_n \triangleq \mathbf{P}_{n|n-1}^{xz} [\mathbf{P}_{n|n-1}^{zz}]^{-1} \quad (3.43)$$

Combining (3.39) and (3.43) with (3.37) yields the linear MMSE estimator

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1}) \quad (3.44)$$

Here, the *innovations* (error in the observation prediction) is defined as

$$\boldsymbol{\epsilon}_n^z \triangleq \mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1} \quad (3.45)$$

The updated covariance can be obtained from

$$\begin{aligned}\mathbf{P}_{n|n}^{xx} &= \mathcal{E}\{\boldsymbol{\epsilon}_n^x \boldsymbol{\epsilon}_n^{x\top} | \mathbf{z}_{1:n-1}\} \\ &= \mathcal{E}\{(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top | \mathbf{z}_{1:n-1}\} \\ &= \mathcal{E}\left\{\left[\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} - \mathbf{K}_n(\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1})\right] \times \left[\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} - \mathbf{K}_n(\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1})\right]^\top | \mathbf{z}_{1:n-1}\right\} \quad (3.46)\end{aligned}$$

After some algebra, this becomes

$$\mathbf{P}_{n|n}^{xx} = \mathbf{P}_{n|n-1}^{xx} - \mathbf{K}_n \mathbf{P}_{n|n-1}^{zz} \mathbf{K}_n^\top \quad (3.47)$$

To summarize, (3.44) and (3.47), along with the Kalman gain definition (3.43), are the *Kalman filter update equations* that utilize the latest observation to update the predictive estimates given by (3.33) and (3.36). However, examination of these equations shows the need for the computation of additional predictive density-weighted integrals for $\hat{\mathbf{z}}_{n|n-1}$, $\mathbf{P}_{n|n-1}^{zz}$ and $\mathbf{P}_{n|n-1}^{xz}$. In general, these *observation-related* predictive estimates are defined by

$$\hat{\mathbf{z}}_{n|n-1} \triangleq \mathcal{E}\{\mathbf{z}_n | \mathbf{x}_n, \mathbf{z}_{1:n-1}\}, \quad (3.48)$$

$$\mathbf{P}_{n|n-1}^{zz} \triangleq \mathcal{E}\{(\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1})(\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1})^\top | \mathbf{x}_n, \mathbf{z}_{1:n-1}\} \quad (3.49)$$

$$\mathbf{P}_{n|n-1}^{xz} \triangleq \mathcal{E}\{(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})(\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1})^\top | \mathbf{z}_{1:n-1}\} \quad (3.50)$$

Following the arguments used to derive (3.33) and (3.36), it follows immediately that

$$\begin{aligned}\widehat{\mathbf{z}}_{n|n-1} &= \int_{\mathbb{R}^{n_x}} \mathbf{h}_n(\mathbf{x}_n) p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n \\ &\quad + \int_{\mathbb{R}^{n_x}} \mathbf{w}_n p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n\end{aligned}\quad (3.51)$$

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{h}_n(\mathbf{x}_n) - \widehat{\mathbf{z}}_{n|n-1}] [\mathbf{h}_n(\mathbf{x}_n) - \widehat{\mathbf{z}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n + \mathbf{R}\end{aligned}\quad (3.52)$$

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{x}_n - \widehat{\mathbf{x}}_{n|n-1}] [\mathbf{h}_n(\mathbf{x}_n) - \widehat{\mathbf{z}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n\end{aligned}\quad (3.53)$$

where \mathbf{R} is defined as

$$\mathbf{R} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{w}_n \mathbf{w}_n^\top p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n\quad (3.54)$$

For (3.52) it has been assumed that $\mathbf{h}_n(\mathbf{x}_n)$ is independent of \mathbf{w}_n , so that the cross-terms integrate to zero. It is usually assumed that \mathbf{w}_n is zero-mean so that the second integral in (3.51) integrates to zero.

A general block diagram for the recursive point estimation process for arbitrary distributions is shown in Figure 3.2.

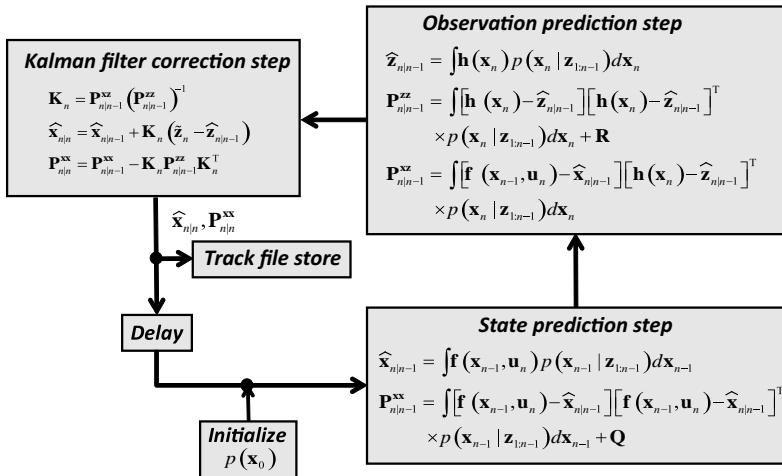


FIGURE 3.2 General block diagram for recursive point estimation process.

3.5 DISCUSSION OF GENERAL ESTIMATION METHODS

In this chapter, two main results have been developed for Bayesian estimation that have almost no restrictions on either the form of the probability densities or on the linearity of the dynamic or observation equations. The first, through the use of (3.23) and (3.24), is a method for a recursive link between the previous posterior $p(\mathbf{x}_{n-1}|\mathbf{z}_{1:n-1})$ and the current posterior $p(\mathbf{x}_n|\mathbf{z}_{1:n})$ that requires specification of the predictive density given by $p(\mathbf{x}_n|\mathbf{x}_{n-1})$ and the likelihood function $p(\mathbf{z}_n|\mathbf{x}_n)$.

In many problems of interest, recursive Monte Carlo evaluation of these densities provide visual insight into the probable location and spread of the state vector, conditioned on all observations. Part III of this book will expand on these Monte Carlo estimation methods applicable to nonlinear systems and non-Gaussian densities (linear and Gaussian cases are included as a subset). In addition, Monte Carlo methods will be presented that allow evaluation of the density-weighted integrals, where Monte Carlo samples of \mathbf{x}_n are used to create discrete density functions that reduce the integrals to weighted sums of the samples. These latter methods have come to be called particle filters.

The second estimation process developed in this chapter is a general method for recursively estimating the first two moments of a state vector conditioned on all observations. This method uses density-weighted integrals for prediction estimates $\hat{\mathbf{x}}_{n|n-1}$, $\mathbf{P}_{n|n-1}^{\mathbf{xx}}$, $\hat{\mathbf{z}}_{n|n-1}$, $\mathbf{P}_{n|n-1}^{\mathbf{zz}}$, and $\mathbf{P}_{n|n-1}^{\mathbf{xz}}$, given by (3.33), (3.36), (3.51), (3.52), and (3.53), respectively. This is followed by the Kalman filter update equations (3.43), (3.44), and (3.47).

Up to this point, developments has been kept completely general with no restrictions on the form of the dynamic or observation process or on the form of the probability density. Part II of this book will concentrate exclusively on the form this point estimation method takes when the densities are Gaussian. This Gaussian assumption leads to all of the Kalman filter variants found in the literature. The Kalman filter variants will be shown to fall into three main categories, the linear Kalman filter, extended Kalman filters, and sigma point Kalman filters where Gaussian-weighted integrals are evaluated numerically.

REFERENCES

1. Van Trees HL. *Detection, Estimation, and Modulation Theory, Part I: Detection, Estimation, and Linear Modulation Theory*. Wiley; 1968.
2. Doucet A, Johansen AM. A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later. Technical Report, Department of Statistics, University of British Columbia; 2008.
3. Kalman RE. A new approach to linear filtering and prediction problems. *ASME J. Basic Eng.* 1960:35–45.

4

CASE STUDIES: PRELIMINARY DISCUSSIONS

This book contains four case studies that illustrate the application and performance of many of the tracking filters developed in Parts II and III. The first case study, developed in this chapter, illustrates the problem of tracking a ship through a field of directional frequency and ranging (DIFAR) sonobuoys. This is a particularly interesting problem because the noise on the bearing observations obtained from each individual buoy is Gaussian for input signals with high signal-to-noise ratio (SNR) and non-Gaussian for signal with low SNR. This case study will allow us to examine the utility of the tracking filters in the low SNR environment where the *observation* noise becomes non-Gaussian. We use this case study in Parts II and III to show how many of the tracking filters can be utilized for this DIFAR problem, presenting implementation details for each estimation method along with the track outputs plotted against truth tracks. We also assess relative track performance measures for all of the tracking algorithms.

In Part IV of this book the remaining case studies will be discussed, including tracking of a object moving in three dimensions using radar observation data; tracking of a falling bomb from video feed observations (photogrammetry); and the use of tracking filters in data fusion for photogrammetry tracking systems. The case studies range from the relatively benign problem in DIFAR bearings-only tracking to a really difficult one that requires large state and observation vectors and high data rates. We show how, for some problems, all of the tracking filters give essentially the same results, while for others, some of the tracking filters cannot be used due to some inherent requirement that cannot be easily met by specific tracking filters.

4.1 THE OVERALL SIMULATION/ESTIMATION/EVALUATION PROCESS

Setting up software code for a specific tracking problem scenario can sometimes be a daunting task. In Figure 4.1, we present a diagram of the process blocks that must be addressed. First, one must build a scenario simulator that generates the truth data for the states to be estimated. This truth data set must then be transformed into a set of truth observations, prior to adding noise.

By way of example, for tracking a vehicle in three Cartesian dimensions, the vehicle truth position and velocity components in Cartesian coordinates (which we call the *truth track*) must be generated. If the observations are from a radar, the Cartesian truth track must undergo a Cartesian-to-spherical conversion, generating the noiseless spherical truth tracks that include range, bearing, elevation, and their respective rates. Then, the specific radar observations (e.g., range, bearing, elevation, and/or Doppler range rate) can be isolated from the spherical truth. In exercising a tracking filter using data obtained from a real radar, this last simulation step can be omitted. The simulation step is primarily used to test and debug the estimation algorithm implementation and for the conduct of special studies to evaluate comparative tracking algorithm performance.

In the second block, usually a Monte Carlo exercise of the tracking algorithm under test, noise is added to the observations and the noisy observations are used as input to

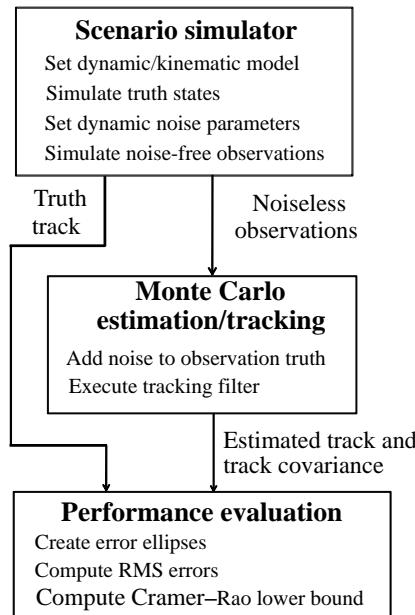


FIGURE 4.1 Methodology for development and evaluation of tracking algorithms.

the tracking filter. The desired tracking filter is then exercised to generate track state and track state error covariance estimates as a function of time.

The final block is where the performance of the tracking algorithm is assessed. For simulated observations, one can use truth track data along with estimated tracks in a series of Monte Carlos runs to generate root mean squared (RMS) errors and examine how these RMS errors compare with the Cramer–Rao Lower Bound (CRLB). When real system data is used, the truth tracks are usually not available, so one can use the estimated track error covariances to generate error ellipses for the estimated tracks.

Each process block of the overall simulation effort can be developed independently since, for the most part, they are invariant with respect to the specific tracking algorithm.

In the section below, the algorithmic underpinnings of the DIFAR tracking scenario simulator will be discussed along with plots describing the particulars of the problem. For each of the track estimation methods presented in Parts II and III we will show track estimate results for the DIFAR problem as an example of a specific application of each algorithm. Performance comparisons of the various Gaussian estimation methods for the DIFAR problem will be presented in Section 14.5 while the performance for particle filter estimation methods for the non-Gaussian noise case will be delayed until the end of Part III of this book.

4.2 A SCENARIO SIMULATOR FOR TRACKING A CONSTANT VELOCITY TARGET THROUGH A DIFAR BUOY FIELD

Our DIFAR scenario consists of a ship traveling at constant velocity moving through a field of DIFAR buoys. The coordinate system we will use is a two-dimensional Cartesian system with an *origin at the center of the buoy field*, the x -axis pointing East, and the y -axis pointing North.

4.2.1 Ship Dynamics Model

Let the ship motion at time t be described by a *state vector* $\mathbf{x}(t)$ that consists of the position and speed components in the two dimensions, that is

$$\mathbf{x}(t) = [r^x(t) \quad v^x \quad r^y(t) \quad v^y]^\top \quad (4.1)$$

where $\{r^x(t), r^y(t)\}$ and $\{v^x, v^y\}$ are the Cartesian position and speed components, respectively. We have chosen a constant velocity model, so that the speed components are independent of time. Given that the ship's position at the start of the scenario is $\{r^x(t_0), r^y(t_0)\}$, for our scenario the position components of the ship at any later time t can be written as

$$r^x(t) = r^x(t_0) + v^x t \quad (4.2)$$

$$r^y(t) = r^y(t_0) + v^y t \quad (4.3)$$

In vector-matrix notation, this can be rewritten as

$$\mathbf{x}(t) = \mathbf{F}_t \mathbf{x}(t_0) \quad (4.4)$$

with

$$\mathbf{F}_t = \begin{bmatrix} \mathbf{F}_{t,1} & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{F}_{t,1} \end{bmatrix} \quad (4.5)$$

and

$$\mathbf{F}_{t,1} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \quad (4.6)$$

4.2.2 Multiple Buoy Observation Model

Assume that there are M buoys randomly deployed within a radius R_{\max} around the origin, with the m th buoy at a *fixed* position $\{x_m, y_m\}$. Of course, in a real buoy field tracking exercise, this assumption of fixed buoy positions over the time frame of the track estimation will not be true because of drift currents and wave action. But for algorithm demonstration, we will assume fixed buoy positions.

For a bearing defined as an angle clockwise relative to true North (y -axis), we can write the bearing from the m th buoy to the target at time t as

$$\theta_m(t) = \tan^{-1} \left(\frac{r^x(t) - x_m}{r^y(t) - y_m} \right) \quad (4.7)$$

with $-\pi \leq \theta_m(t) \leq \pi$.

4.2.3 Scenario Specifics

The first two steps in scenario generations therefore become:

1. Generate a truth track using (4.2) and (4.3) over a uniformly spaced set of times $\{t_n; n = 0, 1, \dots, N\}$.
2. Execute a Cartesian to polar conversion to generate the noise-free observations over the time frame of the scenario using (4.7).

An illustration of the geometry of the problem for the target ship track relative to the center of the buoy field at $(0, 0)$ is shown in Figure 4.2.

An example of the ship truth track and the truth bearings associated with two of the DIFAR sensors are shown in Figure 4.3. The panel on the left shows a small portion of the ships truth track moving from the lower left to the upper right. Also shown on the left panel is an example of a buoy field with 15 buoys randomly distributed about the origin. The right panel shows the bearing to the ship from the two buoys marked

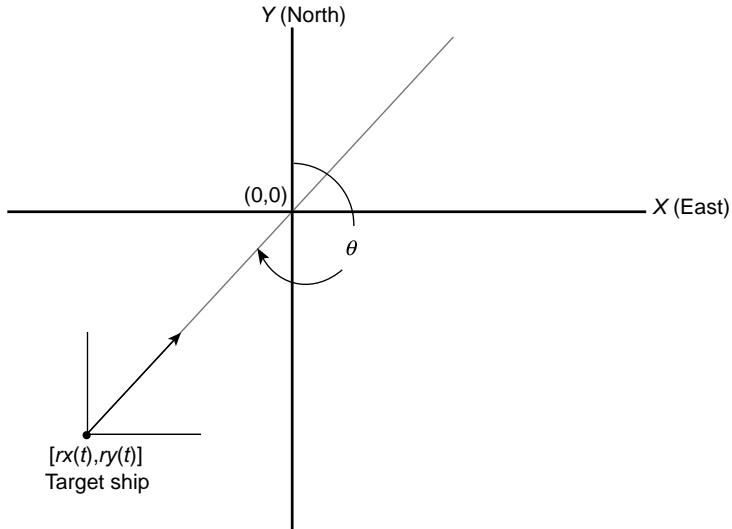


FIGURE 4.2 DIFAR buoy geometry.

in the left panel, one with a * and the other with an x. For this plot, the buoys were distributed in range and bearing about the origin according to

$$\rho_m = R_{\max} * u_m, m = 1, \dots, M$$

$$\vartheta_m \sim U(-\pi, \pi), m = 1, \dots, M$$

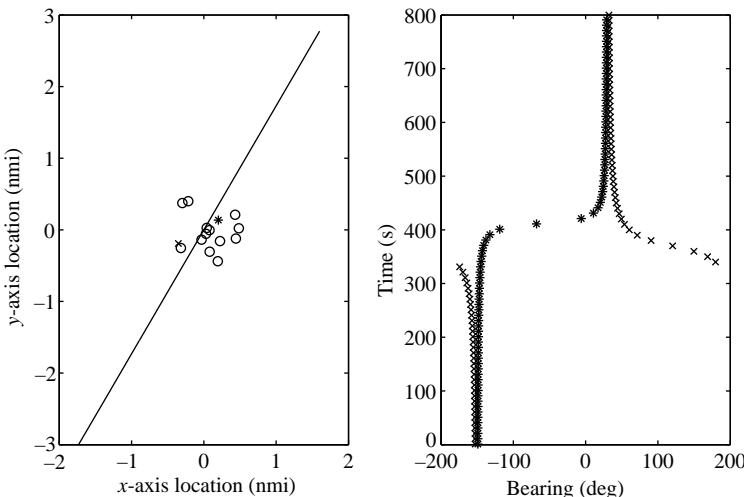


FIGURE 4.3 Ships track with DIFAR buoy field and the associated bearings for two buoys.

with R_{\max} set to 500 feet [$u_m \sim U(a, b)$ indicates that u_m is drawn from a uniform distribution over the interval (a, b)]. The set of polar buoy positions $\{\rho_m, \vartheta_m; m = 1, \dots, M\}$ are then transformed into the Cartesian coordinate set $\{x_m, y_m; m = 1, \dots, M\}$ using

$$x_m = \rho_m \sin \vartheta_m \quad (4.8)$$

$$y_m = \rho_m \cos \vartheta_m \quad (4.9)$$

In Figure 4.3, the right panel shows the bearings associated with the two marked buoys, one on either side of the ships track. From the figure we see that the bearings for the buoy to the East (right side) of the ships track go through zero deg, while the bearings from the buoy to the West go to -180 deg and then jump up to 180 deg.

The bearings for the West buoy undergo what is known as a phase wrap, where it makes a jump of 360 deg in bearing. We will show in Chapter 6 that this phase wrap presents a problem to all tracker algorithms that must be corrected to make the trackers work properly. Similar phase wrap difficulties occur in the application of estimation methods in many different fields and must be addressed in order to obtain valid results. A Matlab subroutine that can be used to generate a ships Cartesian and spherical truth track as well as a set of randomized buoy positions is presented in Listing 4.1.

Listing 4.1 Generation of Truth Track and Buoy Positions

```
function [x_true,r_true,f,s,SNR,BT,SensorLocations,
R,theta_sigma,t] = DifarScenarioGenerator(Stream, ...
N,M,Rmax,del_t,speed,heading,y_init)

global deg2rad hours2seconds
measurement_sigma = 3*deg2rad;
theta_sigma = 1;

t = del_t*(1:N);
speed = speed/hours2seconds;
heading = heading*deg2rad;
vx_init = speed*sin(heading);
vy_init = speed*cos(heading);

x_init = y_init*(vx_init/vy_init);
x_true_init = [x_init;vx_init;y_init;vy_init];

r_m = Rmax*rand(Stream,M,1);
theta_m = 2*pi*(rand(Stream,M,1) - 0.5);
xx = r_m.*sin(theta_m) + Sensor_Offset(1);
yy = r_m.*cos(theta_m) + Sensor_Offset(2);
Sensor_Locations = [xx yy];

R = diag(measurement_sigma^2*ones(M,1),0);

[x_true,f,s,SNR,BT] = ...
Constant_Velocity_Kinematics_Difar(Stream, ...
```

```
N,t, x_true_init);

[r_true] = Cart2Polar(x_true,Sensor_Locations,N,M);
```

In Parts II and III of this book, many methods are presented that enable sequential (in time) estimation of the ships track based on a set of noisy buoy observations. For each estimation method, we use this DIFAR observation data to illustrate filter track estimation outputs.

4.3 DIFAR BUOY SIGNAL PROCESSING

The DIFAR sensor (one of a class of vector sensors) is a passive receiver that operates on the broadband signal radiated from a distant ship and converts the signal into a noisy bearing to the ship using the process illustrated in Figure 4.4. The DIFAR buoy consists of three independent sensors: an omnidirectional hydrophone, an East–West dipole hydrophone, and a North–South dipole hydrophone. The dipole hydrophones may not line up in the proper orientation but their outputs are electronically mixed to produce two dipole outputs of the proper orientation. We will not discuss this reorientation process further because it is irrelevant to the tracking problem and we will always assume the output of each channel to be oriented with the Cartesian axes.

For simulation purposes, one needs to generate noisy bearing observations at the output of each simulated DIFAR sensor. From Figure 4.4 we see that for the m^{th} sensor, the real signals output by each channel are given by

$$z_{\text{NS}}(t) = s(t) \cos \theta + n_{\text{NS}}(t) \quad (4.10)$$

$$z_{\text{EW}}(t) = s(t) \sin \theta + n_{\text{EW}}(t) \quad (4.11)$$

$$z_{\text{O}}(t) = s(t) + n_{\text{O}}(t) \quad (4.12)$$

where $s(t)$ represents the signal at the input to the DIFAR buoy.

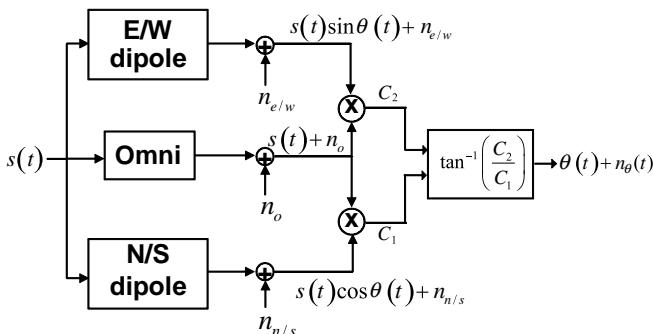


FIGURE 4.4 DIFAR sensor signal processing.

The cross-correlations $C_1(t)$ and $C_2(t)$ are defined by

$$\begin{aligned} C_1(t) &= z_{\text{NS}}(t) \times z_{\text{O}}(t) \\ &= s^2(t) \cos \theta + s(t) n_{\text{O}}(t) \cos \theta \\ &\quad + s(t) n_{\text{NS}}(t) + n_{\text{NS}}(t) n_{\text{O}}(t) \end{aligned} \quad (4.13)$$

and

$$\begin{aligned} C_2(t) &= z_{\text{EW}}(t) \times z_{\text{O}}(t) \\ &= s^2(t) \sin \theta + s(t) n_{\text{O}}(t) \sin \theta \\ &\quad + s(t) n_{\text{EW}}(t) + n_{\text{EW}}(t) n_{\text{O}}(t) \end{aligned} \quad (4.14)$$

Assuming that θ is adiabatic (slowly changing) in time, we can write the expected values of $C_1(t)$ and $C_2(t)$ as

$$\mathcal{E}\{C_1(t)\} = \mathcal{E}\left\{s^2(t)\right\} \cos \theta \quad (4.15)$$

$$\mathcal{E}\{C_2(t)\} = \mathcal{E}\left\{s^2(t)\right\} \sin \theta \quad (4.16)$$

where we have assumed that the signal and noise components are zero-mean Gaussian processes that are all uncorrelated across sensors and time. Thus, the bearing can be obtained from

$$\theta(t) = \tan^{-1} \frac{\mathcal{E}\{C_2(t)\}}{\mathcal{E}\{C_1(t)\}} \quad (4.17)$$

In order to estimate $\mathcal{E}\{C_1(t)\}$ and $\mathcal{E}\{C_2(t)\}$, sample $z_{\text{NS}}(t)$, $z_{\text{EW}}(t)$, and $z_{\text{O}}(t)$, and subdivide time into blocks of length T . Now, let

$$\tau_q = t_n + (i - 1) \delta_t - \frac{N}{2} \delta_t, \quad i = 1, \dots, N; \quad n = 1, 2, \dots, \infty; \quad q = 1, 2, \dots, \infty \quad (4.18)$$

with $\delta = T/N$. For example, we would write

$$\begin{aligned} \tau_1 &= t_1 - \frac{N}{2} \delta_t \\ \tau_2 &= t_1 + \delta_t - \frac{N}{2} \delta_t \\ &\vdots \\ \tau_N &= t_1 + (N - 1) \delta_t - \frac{N}{2} \delta_t = t_1 + \left(\frac{N}{2} - 1\right) \delta_t \\ \tau_{N+1} &= t_2 - \frac{N}{2} \delta_t \end{aligned} \quad (4.19)$$

At time t_n , $\mathcal{E}\{C_1(t_n)\}$ and $\mathcal{E}\{C_2(t_n)\}$ can now be estimated from

$$\mathcal{E}\{C_1(t_n)\} \simeq \int_{t_n - \frac{N}{2}\delta_t}^{t_n + \frac{N}{2}\delta_t} z_{\text{NS}}(t) z_{\text{O}}(t) dt \quad (4.20)$$

$$\mathcal{E}\{C_2(t_n)\} \simeq \int_{t_n - \frac{N}{2}\delta_t}^{t_n + \frac{N}{2}\delta_t} z_{\text{EW}}(t) z_{\text{O}}(t) dt \quad (4.21)$$

From the Weiner–Khintchine theorem [1], (4.20) and (4.21) can be written as

$$\mathcal{E}\{C_1(t_n)\} = \int_{-\infty}^{\infty} Y_{\text{NS}}(t_n, f_k) Y_{\text{O}}^*(t_n, f_k) df \quad (4.22)$$

$$\mathcal{E}\{C_2(t_n)\} = \int_{-\infty}^{\infty} Y_{\text{EW}}(t_n, f_k) Y_{\text{O}}^*(t_n, f_k) df \quad (4.23)$$

where the $*$ represents a complex conjugate and

$$Y_{\text{NS}}(t_n, f_k) = S(t_n, f_k) \cos \theta + N_{\text{NS}}(t_n, f_k) \quad (4.24)$$

$$Y_{\text{EW}}(t_n, f_k) = S(t_n, f_k) \sin \theta + N_{\text{EW}}(t_n, f_k) \quad (4.25)$$

$$Y_{\text{O}}(t_n, f_k) = S(t_n, f_k) + N_{\text{O}}(t_n, f_k) \quad (4.26)$$

For sampled data, the time-domain equations (4.20) and (4.21) become

$$\mathcal{E}\{C_1(t_n)\} \simeq \frac{1}{T} \sum_{i=1}^N z_{\text{NS}}(a_i \delta_t) z_{\text{O}}(a_i \delta_t) \quad (4.27)$$

$$\mathcal{E}\{C_2(t_n)\} \simeq \frac{1}{T} \sum_{i=1}^N z_{\text{EW}}(a_i \delta_t) z_{\text{O}}(a_i \delta_t) \quad (4.28)$$

where

$$a_i \triangleq i - \frac{N+2}{2} \quad (4.29)$$

In the frequency domain, taking the Fast-Fourier transform (FFT) of N samples of each of the set $\{z_{\text{NS}}(t), z_{\text{EW}}(t), z_{\text{O}}(t)\}$ yields the frequency sample set $\{Y_{\text{NS}}(t_n, f_k), Y_{\text{EW}}(t_n, f_k), Y_{\text{O}}(t_n, f_k)\}$ that leads to

$$\mathcal{E}\{C_1(t_n)\} \simeq \frac{\delta_f}{N/2} \sum_{k=1}^{N/2} Y_{\text{NS}}(t_n, f_k) Y_{\text{O}}^*(t_n, f_k) \quad (4.30)$$

$$\mathcal{E}\{C_2(t_n)\} \simeq \frac{\delta_f}{N/2} \sum_{k=1}^{N/2} Y_{\text{EW}}(t_n, f_k) Y_{\text{O}}^*(t_n, f_k) \quad (4.31)$$

TABLE 4.1 Procedure for Generating a Vector of DIFAR Noisy Bearing Observations

Step 1.	Generate truth bearings.	$\theta_m(t) = \tan^{-1} \left(\frac{r^x(t)-x_m}{r^y(t)-y_m} \right)$
Step 2.	Generate COMPLEX random noises for each channel.	$N_{\text{NS}}(t_n, f_k), N_{\text{EW}}(t_n, f_k), N_{\text{O}}(t_n, f_k)$
Step 3.	Generate COMPLEX random signal.	$S(t_n, f_k)$
Step 4.	Calculate DIFAR channel outputs.	$Y_{\text{NS,m}}(t_n, f_k) = S(t_n, f_k) \cos \theta_m + N_{\text{NS}}(t_n, f_k)$ $Y_{\text{EW,m}}(t_n, f_k) = S(t_n, f_k) \sin \theta_m + N_{\text{EW}}(t_n, f_k)$ $Y_{\text{O}}(t_n, f_k) = S(t_n, f_k) + N_{\text{O}}(t_n, f_k)$
Step 5.	Calculate convolution products.	$C_{1,m}(t_n) = Y_{\text{NS}}(t_n, f_k) Y_{\text{O}}^*(t_n, f_k)$ $C_{2,m}(t_n) = Y_{\text{EW}}(t_n, f_k) Y_{\text{O}}^*(t_n, f_k)$
Step 6.	Sum over frequency band	$\mathcal{E}\{C_{1,m}(t_n)\} = \frac{\delta_f}{N/2} \sum_{k=1}^{N/2} C_{1,m}(t_n)$ $\mathcal{E}\{C_{2,m}(t_n)\} = \frac{\delta_f}{N/2} \sum_{k=1}^{N/2} C_{2,m}(t_n)$
Step 7.	Compute noisy bearings	$\theta_{n,m} = \tan^{-1} \left[\frac{\mathcal{E}\{C_{2,m}(t_n)\}}{\mathcal{E}\{C_{1,m}(t_n)\}} \right]$

The procedure for generating noisy bearing observations from a ship truth track for each DIFAR sensor is presented in Table 4.1. In the Listing 4.2 we present a Matlab snippet that produces a noisy wideband signal. A Matlab subroutine that executes the procedure given in Table 4.1 is shown in Listing 4.3.

Listing 4.2 A Snippet of Matlab Code that Produces N Noisy Signal Samples for K Frequency Bins

```

f = 300:15:400;
f_target = [10 20 50 100 200 400 1000 4000]';
SPL_target = [20;20;20;20;20;20;20;20];

SPL = interp1q(log10(f_target), SPL_target, log10(f));
spl = repmat(10.^((SPL/10), 1, N);
SNR = spl(1);
K = length(f);
BT = (f(K) - f(1))*(t(2)-t(1));

SPL = interp1q(log10(f_target), SPL_target, log10(f));
spl = repmat(10.^((SPL/10), 1, N);
SNR = spl(1);
K = length(f);

```

```

BT = (f(K) - f(1))*(t(2)-t(1));

s = sqrt(spl/2).*randn(Stream,K,N) + ...
1i*randn(Stream,K,N);

K = length(f);
BT = (f(K) - f(1))*(t(2)-t(1));

```

Listing 4.3 A Matlab Subroutine that Generates the Noisy Bearing Measurements Using the Steps in Table 4.1

```

function [bearing_measurement] = ...
First_Sensor_Measurement_Generator_Difar(f,s, ...
theta_sigma,theta_true,N,M,Stream)

K = length(f);
del_f = f(2) - f(1);

NS_ms = sqrt(theta_sigma)*(randn(Stream,K,N,M) + ...
1i*randn(Stream,K,N,M));
EW_ms = sqrt(theta_sigma)*(randn(Stream,K,N,M) + ...
1i*randn(Stream,K,N,M));
O_ms = sqrt(theta_sigma)*(randn(Stream,K,N,M) + ...
1i*randn(Stream,K,N,M));

Y_NS = repmat(s,[1,1,M]).*...
repmat(cos(theta_true),[K,1,1]) + NS_ms;
Y_EW = repmat(s,[1,1,M]).*...
repmat(sin(theta_true),[K,1,1]) + EW_ms;
Y_O = repmat(s,[1,1,M]) + O_ms;

C1 = real(Y_NS.*conj(Y_O));
C2 = real(Y_EW.*conj(Y_O));

E_C1 = (2*del_f/N)*permute(sum(C1),[2 3 1]);
E_C2 = (2*del_f/N)*permute(sum(C2),[2 3 1]);

bearing_measurement = permute(atan2(E_C2,E_C1),[2 1]);

```

One must note that the ratio inside the inverse tangent operation contained in Step 7 of Table 4.1 is a ratio of random numbers. It is well known that if both variables are independent with zero mean, the distribution of the ratio is a Cauchy distribution. However, when the two variables do not have a zero mean, the distribution of the ratio is much more complicated, as we will see in the next section. In addition, for the DIFAR bearings, the distribution is further complicated by the inverse tangent operation. So, we cannot expect the DIFAR bearing observations to have a Gaussian distribution! However, we can expect the distribution to be a function of the SNR

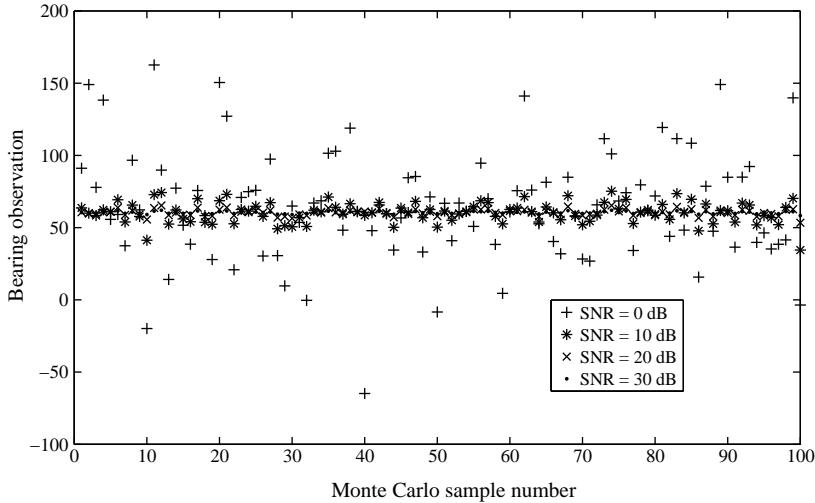


FIGURE 4.5 The spread of bearing observations from a DIFAR buoy from 100 Monte Carlo runs for four SNRs.

of the signal at the input to the DIFAR buoy. To illustrate this, Figure 4.5 show the bearing observations of one buoy at one specific time from 100 Monte Carlo runs for four different signal SNRs (at the input to the DIFAR sensor). It is clear that as SNR increases, the distribution becomes more peaked and approximates a Gaussian, but as the SNR decreases, the bearing distribution approaches that of a uniform distribution over $[-180^\circ, 180^\circ]$.

4.4 THE DIFAR LIKELIHOOD FUNCTION

If we assume that the DIFAR sensor likelihood functions are independent from sensor to sensor, then the likelihood function for the observation vector conditioned on the target ship position is given by the joint density

$$p(\mathbf{z}_n | \mathbf{x}_n) = p(\boldsymbol{\theta}_n | \mathbf{x}_n) = \prod_{m=1}^M p(\theta_{n,m} | \mathbf{x}_n) \quad (4.32)$$

where $\boldsymbol{\theta}_n = [\theta_{n,1}, \dots, \theta_{n,M}]^\top$ with

$$\theta_{n,m} = \tan^{-1} \left(\frac{r_n^x - x_m}{r_n^y - y_m} \right) \quad (4.33)$$

Although it is an interesting problem in itself, the full derivation of the likelihood function for a DIFAR buoy is not the main emphasis of this book, so any interested reader can find the derivation in the appendix of Ref. [2], available online. Here,

because it is needed in Part III of this book, we present just the resulting likelihood density and its properties.

If we ignore the time index for clarity and let $z_m \triangleq \tan \theta_m$, then from Appendix A in Ref. [2] we can write

$$p(\theta_m | \mathbf{x}_n) = \left[1 + (z_m)^2 \right] p(z_m | \theta_{n,m}) \quad (4.34)$$

with

$$\begin{aligned} p(z_m | \theta_m) = A & \left\{ \frac{\sqrt{(1-r^2)}}{\pi} e^{-\alpha} + \frac{\delta \beta^{1/2}}{\sqrt{2\pi}} \right. \\ & \times \left. \left[\Phi \left(\frac{-\delta \beta^{1/2}}{\sqrt{(1-r^2)}} \right) - \Phi \left(\frac{\delta \beta^{1/2}}{\sqrt{(1-r^2)}} \right) \right] e^{-\gamma} \right\} \end{aligned} \quad (4.35)$$

Here

$$A \triangleq \frac{\tilde{\sigma}_1 \tilde{\sigma}_2}{\tilde{\sigma}_2^2 - 2r\tilde{\sigma}_1 \tilde{\sigma}_2 z_m + \tilde{\sigma}_1^2 z_m^2} \quad (4.36)$$

$$\alpha \triangleq \frac{\tilde{\sigma}_2^2 \tilde{\eta}_1^2 - 2r\tilde{\sigma}_1 \tilde{\sigma}_2 \tilde{\eta}_1 \tilde{\eta}_2 + \tilde{\sigma}_1^2 \tilde{\eta}_2^2}{2\tilde{\sigma}_1^2 \tilde{\sigma}_2^2 (1-r^2)} \quad (4.37)$$

$$\beta \triangleq \frac{\left[(\tilde{\sigma}_2^2 \tilde{\eta}_1 - r\tilde{\sigma}_1 \tilde{\sigma}_2 \tilde{\eta}_2) + (\tilde{\sigma}_1^2 \tilde{\eta}_2 - r\tilde{\sigma}_1 \tilde{\sigma}_2 \tilde{\eta}_1) z_m \right]^2}{\tilde{\sigma}_1^2 \tilde{\sigma}_2^2 (\tilde{\sigma}_2^2 - 2r\tilde{\sigma}_1 \tilde{\sigma}_2 z_m + \tilde{\sigma}_1^2 z_m^2)} \quad (4.38)$$

$$\gamma \triangleq \frac{(\tilde{\eta}_2 - \tilde{\eta}_1 z_m)^2}{2 (\tilde{\sigma}_2^2 - 2r\tilde{\sigma}_1 \tilde{\sigma}_2 z_m + \tilde{\sigma}_1^2 z_m^2)} \quad (4.39)$$

$$\delta \triangleq \text{sign} \left[\left(\tilde{\sigma}_2^2 \tilde{\eta}_1 - r\tilde{\sigma}_1 \tilde{\sigma}_2 \tilde{\eta}_2 \right) + \left(\tilde{\sigma}_1^2 \tilde{\eta}_2 - r\tilde{\sigma}_1 \tilde{\sigma}_2 \tilde{\eta}_1 \right) z_m \right] \quad (4.40)$$

with

$$\tilde{\eta}_1 \triangleq \sqrt{2BT} \text{SNR} \cos \theta_m^{(i)} \quad (4.41)$$

$$\tilde{\eta}_2 \triangleq \sqrt{2BT} \text{SNR} \sin \theta_m^{(i)} \quad (4.42)$$

$$\tilde{\sigma}_1^2 \triangleq (2\text{SNR} + 1) \text{SNR} \cos^2 \theta_m^{(i)} + \rho (\text{SNR} + 1) \quad (4.43)$$

$$\tilde{\sigma}_2^2 \triangleq (2\text{SNR} + 1) \text{SNR} \sin^2 \theta_m^{(i)} + \rho (\text{SNR} + 1) \quad (4.44)$$

$$r \triangleq \frac{(2\text{SNR} + 1) \text{SNR} \cos \theta_m^{(i)} \sin \theta_m^{(i)}}{\tilde{\sigma}_1 \tilde{\sigma}_2} \quad (4.45)$$

$$\Phi(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-u^2/2} du \quad (4.46)$$

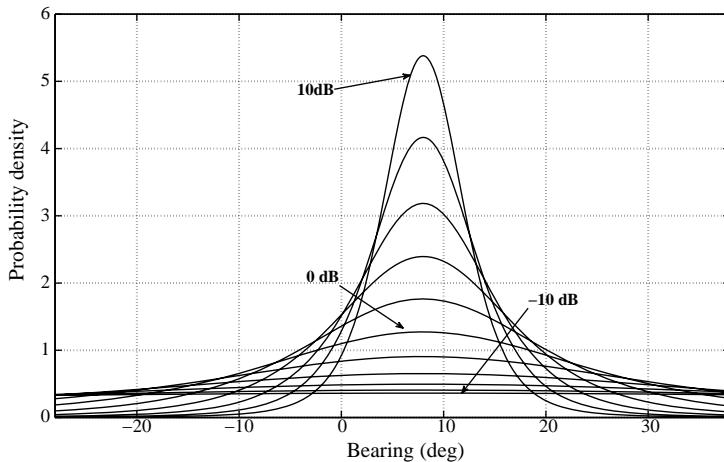


FIGURE 4.6 DIFAR likelihood density for a BT of 10 and SNR from -10 to $+10$ dB.

SNR is the signal-to-noise ratio at the monopole channel, BT is the timebandwidth product of the vehicle frequency signature and ρ is the noise gain (loss) factor applied to either dipole channel and $\Phi(x)$ is related to the error function. The value of $\rho = 1/2$ or $1/3$ for 2D-isotropic or 3D-isotropic noise, respectively [3].

Note that the likelihood function given by (4.34) is actually the conditional likelihood $p(\theta|\theta_0)$, where θ_0 is the true bearing to the target ship based on (4.7) and θ represents the set of bearings at which the likelihood is calculated. A plot of $p(\theta|\theta_0)$ is shown in Figure 4.6 for a bandwidth-time product (BT) of 10 and a range of SNRs ($-10 \leq \text{SNR} \leq 10$). One can readily see that for low SNRs the likelihood is far from Gaussian because of the excessively high tails and the fact that it is only defined over the range $-\pi \leq \theta \leq \pi$.

REFERENCES

1. Marple SL. *Digital Spectral Analysis with Applications*. Prentice-Hall: Englewood Cliffs, NJ; 1987.
2. Haug AJ. A Tutorial on Bayesian Estimation and Tracking Techniques Applicable to Non-linear and Non-Gaussian Processes. MITRE Technical Report MTR 05W0000004; 2005.
3. Maranda BH. The statistical accuracy of an arctangent bearing estimator. *OCEANS 2003*; 4:2127–2132.

PART II

THE GAUSSIAN ASSUMPTION: A FAMILY OF KALMAN FILTER ESTIMATORS

5

THE GAUSSIAN NOISE CASE: MULTIDIMENSIONAL INTEGRATION OF GAUSSIAN-WEIGHTED DISTRIBUTIONS

At the end of Chapter 3, the Bayesian point estimation equations were developed using general probability density functions. In this part of the book, it will be assumed that the dynamic and observation equations constitute Gaussian processes. Under this assumption all of the distribution functions contained in the point estimator equations become Gaussian. It is well known that the first two moments of a Gaussian density characterize the density completely. Therefore, a recursive propagation of estimates of the first two moments produces an optimal estimation method for Gaussian processes. The subject of this part of the book includes the derivation of numerous methods for solving the density-weighted predictive point estimates developed in Chapter 3 for the special case of Gaussian densities.

In Section 3.4, a set of Kalman filter update equations were developed that levied no requirements on the linearity of the processes or the form of the densities associated with those processes. The main assumption made was that the posterior point estimate of \mathbf{x}_n , written as $\hat{\mathbf{x}}_{n|n}$, be a linear function of the latest observation \mathbf{z}_n^o . Additional requirements on the point estimator included that the estimation error be zero-mean and be uncorrelated with the observation vector. It is shown in the sections that follow how these Kalman filter update equations arise naturally from the Gaussian distribution because it automatically fulfills all of these original assumptions and requirements.

Armed with the general form of the Bayesian point estimation equations from Chapter 3, Section 5.3 is devoted to showing that the predictive estimation equations reduce to Gaussian-weighted integrals. In the chapters that follow, the Kalman filter predictive Gaussian-weighted integrals are solved numerically for a variety of

assumptions and approximations concerning the form of the dynamic and observation functions $\mathbf{f}_n(\mathbf{x}_n)$ and $\mathbf{h}_n(\mathbf{x}_n)$, respectively. These solutions lead to four classes of Kalman filters:

- *The Linear Class:* When the dynamic and observation equations are both linear and all densities are Gaussian, the integrals can be solved directly leading to the linear Kalman filter (LKF).
- *The Analytical Linearization Class:* When all nonlinear functions are expanded in Taylor polynomials and only the linear terms are maintained, the integrals can again be solved directly leading to a Kalman filter form almost identical to that of the LKF. But an additional step requires the computation of the Jacobian of each nonlinear function. These filters consist of the extended Kalman filter (EKF) and all of its variants.
- *The Sigma Point Class:* For this class, the nonlinear functions are expanded in more general polynomials such that the integrals reduce to weighted summations over a set of deterministic vector points, called sigma points. Specific filters of this type include the finite difference Kalman filter (FDKF), the unscented Kalman filter (UKF), the spherical simplex Kalman filter (SSKF), and the Gauss–Hermite Kalman filter (GHKF).
- *The Monte Carlo Class:* If a set of Monte Carlo samples are drawn from the Gaussian density, creating a discrete density, then the integrals reduce to a sum over discrete random sample points. This method leads to the Monte Carlo Kalman filter (MCKF).

Much of the material presented in this part of the book is based on ideas whose origin lie in a paper by Wu et al. [1]. This paper outlines and compares most of the solutions developed to date for Gaussian-weighted integration. It formulated the thread that can be used to link all of the Kalman filter algorithms into a single framework. Through this paper, the author discovered the vast literature related to numerical integration methods that can be used to understand all of the recursive Bayesian point-estimation methods for solving Gaussian-weighted integrals. Some of the related references used to formulate this part of the book include the paper by McNamee and Stenger [2] and the PhD dissertation of Lerner [3], both of which inspired the work of Wu et al. Equally important to the development of this entire book were the numerical integration books by Davis and Rabinowitz [4] and Evans and Swartz [5]. Finally, the paper by Ito and Xiong [6] was seminal in providing initial insight into the general concepts of Gaussian filters.

5.1 SUMMARY OF IMPORTANT RESULTS FROM CHAPTER 3

The important results of Chapter 3 are summarized. Consider a dynamic process of the form

$$\mathbf{x}_n = \mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n + \mathbf{v}_{n-1} \quad (5.1)$$

with observations generated from the observation process

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_n \quad (5.2)$$

A point *prediction estimate* of the state vector at time t_n , based on all observations up to and including time t_{n-1} , is given by the density weighted integral

$$\hat{\mathbf{x}}_{n|n-1} = \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n + \mathbf{v}_{n-1}] p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \quad (5.3)$$

Similarly, a point *prediction estimate* of the state covariance at time t_n , based on all observations up to and including time t_{n-1} , is given by the density weighted integral

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{f}_{n-1}(\mathbf{x}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} + \mathbf{Q} \end{aligned} \quad (5.4)$$

In addition, point *prediction estimates* of the observation vector \mathbf{z}_n , the covariance of \mathbf{z}_n , and its cross-covariance with \mathbf{x}_n are given by

$$\hat{\mathbf{z}}_{n|n-1} = \int_{\mathbb{R}^{n_x}} [\mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_n] p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n \quad (5.5)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{h}_n(\mathbf{x}_n) - \hat{\mathbf{z}}_{n|n-1}] [\mathbf{h}_n(\mathbf{x}_n) - \hat{\mathbf{z}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n + \mathbf{R} \end{aligned} \quad (5.6)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \int_{\mathbb{R}^{n_x}} [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{h}_n(\mathbf{x}_n) - \hat{\mathbf{z}}_{n|n-1}]^\top \\ &\quad \times p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n \end{aligned} \quad (5.7)$$

where \mathbf{Q} and \mathbf{R} are defined by

$$\mathbf{Q} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{v}_{n-1} \mathbf{v}_{n-1}^\top p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \quad (5.8)$$

and

$$\mathbf{R} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{w}_n \mathbf{w}_n^\top p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) d\mathbf{x}_n \quad (5.9)$$

5.2 DERIVATION OF THE KALMAN FILTER CORRECTION (UPDATE) EQUATIONS REVISITED

In this section, we provide an alternate derivation of the Kalman filter correction equations (3.43), (3.44), and (3.47), based on the assumption that all conditional densities are Gaussian. Bayes' law provides a link between the posterior density and the joint density of \mathbf{x}_n with \mathbf{z}_n resulting in

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{p(\mathbf{x}_n, \mathbf{z}_n | \mathbf{z}_{1:n-1})}{p(\mathbf{z}_n)} \quad (5.10)$$

Now defining the joint vector

$$\mathbf{q} \triangleq \begin{bmatrix} \mathbf{x}_n \\ \mathbf{z}_n \end{bmatrix} \quad (5.11)$$

(5.10) can be written as

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{p(\mathbf{q}_n | \mathbf{z}_{1:n-1})}{p(\mathbf{z}_n)} \quad (5.12)$$

The general form for a multivariate Gaussian distribution can be written as

$$\mathcal{N}(\mathbf{t}; \mathbf{s}, \Sigma) \triangleq \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp \left\{ -\frac{1}{2} (\mathbf{t} - \mathbf{s})^\top \Sigma^{-1} (\mathbf{t} - \mathbf{s}) \right\} \quad (5.13)$$

Let all densities in (5.12) be Gaussian so that

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n}, \mathbf{P}_{n|n}^{\mathbf{xx}}) \quad (5.14)$$

$$p(\mathbf{q}_n | \mathbf{z}_{1:n-1}) = \mathcal{N}(\mathbf{q}_n; \hat{\mathbf{q}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\mathbf{qq}}) \quad (5.15)$$

$$p(\mathbf{z}_n) = \mathcal{N}(\mathbf{z}_n; \hat{\mathbf{z}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\mathbf{zz}}) \quad (5.16)$$

Ignoring the factor of $(-1/2)$, the exponent of the exponential in the analytical expression for the posterior Gaussian density (5.14) can now be written out explicitly as

$$\begin{aligned} d &\triangleq (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top [\mathbf{P}_{n|n}^{\mathbf{xx}}]^{-1} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}) \\ &= \mathbf{x}_n^\top [\mathbf{P}_{n|n}^{\mathbf{xx}}]^{-1} \mathbf{x}_n - \mathbf{x}_n^\top [\mathbf{P}_{n|n}^{\mathbf{xx}}]^{-1} \hat{\mathbf{x}}_{n|n} \\ &\quad - \mathbf{x}_{n|n}^\top [\mathbf{P}_{n|n}^{\mathbf{xx}}]^{-1} \mathbf{x}_n + \mathbf{x}_{n|n}^\top [\mathbf{P}_{n|n}^{\mathbf{xx}}]^{-1} \hat{\mathbf{x}}_{n|n} \end{aligned} \quad (5.17)$$

Similarly, the exponent in the analytical expression for the ratio of the Gaussian densities on the right-hand side of (5.15) is given by

$$\begin{aligned} g &= (\mathbf{q}_n - \hat{\mathbf{q}}_{n|n-1})^\top [\mathbf{P}_{n|n-1}^{\mathbf{qq}}]^{-1} (\mathbf{q}_n - \hat{\mathbf{q}}_{n|n-1}) \\ &\quad - (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1})^\top [\mathbf{P}_{n|n-1}^{\mathbf{zz}}]^{-1} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \end{aligned} \quad (5.18)$$

Using (5.11), (5.18) can be written more explicitly as

$$\begin{aligned} g &= \begin{bmatrix} \mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} \\ \mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1} \end{bmatrix}^\top \begin{bmatrix} \mathbf{P}_{n|n-1}^{\text{xx}} & \mathbf{P}_{n|n-1}^{\text{xz}} \\ \mathbf{P}_{n|n-1}^{\text{zx}} & \mathbf{P}_{n|n-1}^{\text{zz}} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} \\ \mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1} \end{bmatrix} \\ &\quad - (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1})^\top [\mathbf{P}_{n|n-1}^{\text{zz}}]^{-1} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \end{aligned} \quad (5.19)$$

Define the inverse of the block matrix term in (5.19) as

$$\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{P}_{n|n-1}^{\text{xx}} & \mathbf{P}_{n|n-1}^{\text{xz}} \\ \mathbf{P}_{n|n-1}^{\text{zx}} & \mathbf{P}_{n|n-1}^{\text{zz}} \end{bmatrix}^{-1} \quad (5.20)$$

and using the block matrix inverse presented in Section 2.1.4 [7], we obtain

$$\mathbf{C}_{11} = \left[\mathbf{P}_{n|n-1}^{\text{xx}} - \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} \mathbf{P}_{n|n-1}^{\text{zx}} \right]^{-1} \quad (5.21)$$

$$\mathbf{C}_{12} = -\mathbf{C}_{11} \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} \quad (5.22)$$

$$\mathbf{C}_{21} = -\mathbf{C}_{22} \mathbf{P}_{n|n-1}^{\text{zx}} (\mathbf{P}_{n|n-1}^{\text{xx}})^{-1} \quad (5.23)$$

$$\mathbf{C}_{22} = \mathbf{P}_{n|n-1}^{\text{zz}} - \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{xx}})^{-1} \mathbf{P}_{n|n-1}^{\text{xz}} \quad (5.24)$$

Now, using (5.20)–(5.24) in (5.19), expanding and rearranging terms, we obtain

$$g = \mathbf{x}_n^\top \mathbf{C}_{11} \mathbf{x}_n + \mathbf{x}_n^\top [-\mathbf{C}_{11} \hat{\mathbf{x}}_{n|n-1} + \mathbf{C}_{12} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1})] + \dots \quad (5.25)$$

Comparing the first term in (5.17) with the first term in (5.25), it follows that

$$\mathbf{P}_{n|n}^{\text{xx}} = \mathbf{P}_{n|n-1}^{\text{xx}} - \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} \mathbf{P}_{n|n-1}^{\text{zx}} \quad (5.26)$$

Comparing the second term in (5.17) with the second term in (5.25) results in

$$\begin{aligned} [\mathbf{P}_{n|n}^{\text{xx}}]_{n|n}^{-1} \hat{\mathbf{x}}_{n|n} &= \mathbf{C}_{11} \hat{\mathbf{x}}_{n|n-1} - \mathbf{C}_{12} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \\ &= \left[\mathbf{P}_{n|n-1}^{\text{xx}} - \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} \mathbf{P}_{n|n-1}^{\text{zx}} \right]_{n|n-1}^{-1} \hat{\mathbf{x}}_{n|n-1} \\ &\quad + \left[\mathbf{P}_{n|n-1}^{\text{xx}} - \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} \mathbf{P}_{n|n-1}^{\text{zx}} \right]^{-1} \\ &\quad \times \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \end{aligned} \quad (5.27)$$

Solving for $\hat{\mathbf{x}}_{n|n}$ and using (5.26), this becomes

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \quad (5.28)$$

Let the Kalman gain be defined as

$$\mathbf{K}_n \triangleq \mathbf{P}_{n|n-1}^{\text{xz}} (\mathbf{P}_{n|n-1}^{\text{zz}})^{-1} \quad (5.29)$$

Now, (5.28) and (5.26) become the *Kalman filter correction equations*

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1}) \quad (5.30)$$

$$\mathbf{P}_{n|n}^{xx} = \mathbf{P}_{n|n-1}^{xx} - \mathbf{K}_n \mathbf{P}_{n|n-1}^{zz} \mathbf{K}_n^\top \quad (5.31)$$

where \mathbf{z}_n has been replaced by an actual observation \mathbf{z}_n^o and we have used the fact that $\mathbf{P}_{n|n-1}^{xz} \left(\mathbf{P}_{n|n-1}^{zz} \right)^{-1} \mathbf{P}_{n|n-1}^{zx} = \mathbf{K}_n \mathbf{P}_{n|n-1}^{zz} \mathbf{K}_n^\top$. Note that in the derivation of these Kalman filter correction equations, no assumptions have been made regarding the linearity of either the dynamic or observation equations (5.1) and (5.2), respectively.

Thus, the Kalman filter correction equations have been derived directly from the Gaussian distributions, without the need for any assumptions or conditions. However, it is useful to remember that these correction equations are more general and can be applied to the first two moments of ANY distribution, as long as the appropriate moments exist. It must be noted that, to date, there have been no journal articles using these correction equations for non-Gaussian densities (to our knowledge).

5.3 THE GENERAL BAYESIAN POINT PREDICTION INTEGRALS FOR GAUSSIAN DENSITIES

Remembering the dynamic equation (5.1), and assuming that \mathbf{v}_n is a zero-mean Gaussian random variable independent of both $\mathbf{x}_n \in \mathbb{R}^{n_x}$ and $\mathbf{z}_n \in \mathbb{R}^{n_z}$, it follows that

$$\mathbf{v}_{n-1} \sim p(\mathbf{x}_n | \mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n - [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n]; \mathbf{0}, \mathbf{Q}) = \mathcal{N}(\mathbf{v}_{n-1}; \mathbf{0}, \mathbf{Q}) \quad (5.32)$$

Similarly, in the observation equation (5.2), assume that \mathbf{w}_n is a zero-mean Gaussian random variables independent of both \mathbf{x}_n and \mathbf{z}_n , that is

$$\mathbf{w}_n \sim p(\mathbf{z}_n | \mathbf{x}_n) = \mathcal{N}(\mathbf{z}_n - \mathbf{h}_n(\mathbf{x}_n); \mathbf{0}, \mathbf{R}) = \mathcal{N}(\mathbf{w}_n; \mathbf{0}, \mathbf{R}) \quad (5.33)$$

Now, from Bayes' law, the prior density can be obtained from

$$p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) \propto p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) \quad (5.34)$$

making $p(\mathbf{x}_n | \mathbf{z}_{1:n-1})$ Gaussian, since the product of two Gaussian densities is also Gaussian. Assuming that all density functions are Gaussian, we can identify

$$p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) = \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{xx}) \quad (5.35)$$

$$p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) = \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{xx}) \quad (5.36)$$

Assuming that \mathbf{v}_n and \mathbf{w}_n are zero-mean Gaussian processes, and substituting (5.36) into (5.3) and (5.4) and (5.35) into (5.5)–(5.7), the prediction estimation process

can be summarized by the Gaussian-weighted integrals

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \int \mathbf{f}_{n-1}(\mathbf{x}_{n-1}) \\ &\quad \times \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\text{xx}}) d\mathbf{x}_{n-1} + \mathbf{u}_n\end{aligned}\quad (5.37)$$

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\text{xx}} &= \int [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}] \\ &\quad \times [\mathbf{f}_{n-1}(\mathbf{x}_{n-1}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}]^\top \\ &\quad \times \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\text{xx}}) d\mathbf{x}_{n-1} + \mathbf{Q}\end{aligned}\quad (5.38)$$

$$\hat{\mathbf{z}}_{n|n-1} = \int \mathbf{h}_n(\mathbf{x}_n) \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\text{xx}}) d\mathbf{x}_n \quad (5.39)$$

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\text{zz}} &= \int (\mathbf{h}_n(\mathbf{x}_n) - \hat{\mathbf{z}}_{n|n-1}) (\mathbf{h}_n(\mathbf{x}_n) - \hat{\mathbf{z}}_{n|n-1})^\top \\ &\quad \times \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\text{xx}}) d\mathbf{x}_n + \mathbf{R}\end{aligned}\quad (5.40)$$

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\text{xz}} &= \int (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) (\mathbf{h}_n(\mathbf{x}_n) - \hat{\mathbf{z}}_{n|n-1})^\top \\ &\quad \times \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\text{xx}}) d\mathbf{x}_n\end{aligned}\quad (5.41)$$

with

$$\mathbf{Q} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{v}_{n-1} \mathbf{v}_{n-1}^\top \mathcal{N}(\mathbf{v}_{n-1}; \mathbf{0}, \mathbf{Q}) d\mathbf{v}_{n-1} \quad (5.42)$$

and

$$\mathbf{R} \triangleq \int_{\mathbb{R}^{n_x}} \mathbf{w}_n \mathbf{w}_n^\top \mathcal{N}(\mathbf{w}_n; \mathbf{0}, \mathbf{R}) d\mathbf{w}_n \quad (5.43)$$

A block diagram of the general Bayesian estimation process flow with Gaussian probability densities is shown in Figure 5.1. The process contains the following steps:

1. Initialize the tracking filter by generating best guess values for $\hat{\mathbf{x}}_0$ and \mathbf{P}_0^{xx} . These can usually be based on some initial observations.
2. Evaluate the state vector prediction integrals in some manner.
3. Evaluate the observation prediction integrals in some manner.
4. Perform the Kalman filter correction equations to produce the posterior point estimates $\hat{\mathbf{x}}_{n|n}$ and $\mathbf{P}_{n|n}^{\text{xx}}$.
5. Store the results in a track file.
6. Step forward in time to the next filter iteration using $\hat{\mathbf{x}}_{n-1|n-1}$ and $\mathbf{P}_{n-1|n-1}^{\text{xx}}$ as the new input values.
7. Return to step 2.

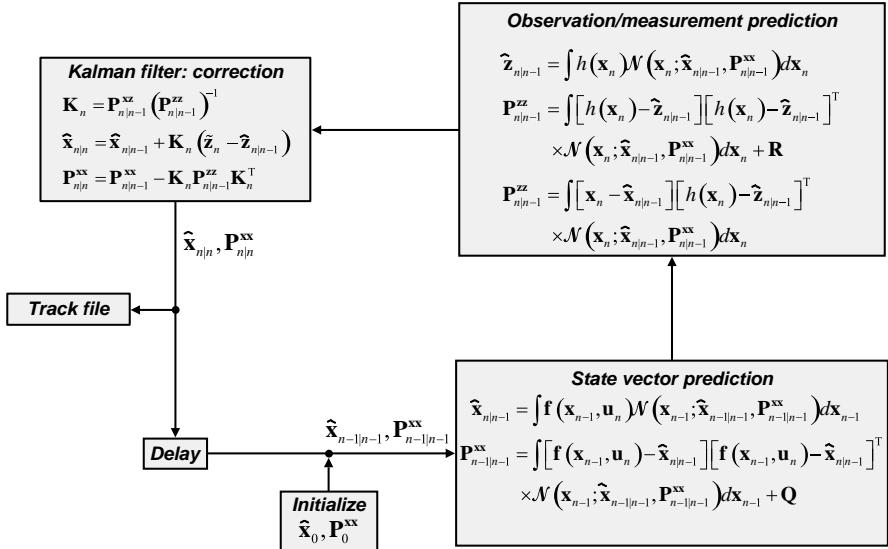


FIGURE 5.1 Block diagram of the process flow for the Bayesian estimator with Gaussian probability densities.

All of the Bayesian estimation methods based on the Gaussian assumption will follow this same basic process flow, with either the functional approximations or the method of integration changing.

5.3.1 Refining the Process Through an Affine Transformation

In Chapter 2 (Section 2.4.2.3) we discussed the application of a vector affine transformation to a general Gaussian integral of the form (2.106), repeated here for clarity

$$\begin{aligned} \mathcal{E}\{\mathbf{f}(\mathbf{x})\} &= \int \mathbf{f}(\mathbf{x}) \frac{1}{(2\pi)^{n_x/2} |\mathbf{P}^{\text{xx}}|^{1/2}} \\ &\quad \times \exp \left\{ -\frac{1}{2} (\mathbf{x} - \hat{\mathbf{x}})^T [\mathbf{P}^{\text{xx}}]^{-1} (\mathbf{x} - \hat{\mathbf{x}}) \right\} d\mathbf{x} \end{aligned} \quad (5.44)$$

$$= \int \mathbf{f}(\mathbf{x}) \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\text{xx}}) d\mathbf{x} \quad (5.45)$$

As shown in Chapter 2, the vector affine transformation

$$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{D}\mathbf{c} \quad (5.46)$$

results in

$$\int \mathbf{f}(\mathbf{x}) \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{P}^{\text{xx}}) d\mathbf{x} = \int \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.47)$$

where \mathbf{D} is defined by

$$\mathbf{P}^{\mathbf{xx}} = \mathbf{DD}^\top \quad (5.48)$$

Or, we can write

$$\mathbf{D} = [\mathbf{P}^{\mathbf{xx}}]^{1/2} \quad (5.49)$$

In addition, we have defined $\tilde{\mathbf{f}}(\mathbf{c})$ as

$$\tilde{\mathbf{f}}(\mathbf{c}) \triangleq \mathbf{f}(\hat{\mathbf{x}} + \mathbf{D}\mathbf{c}) \quad (5.50)$$

Applying the vector affine transformation to equations (5.37) and (5.38) results in

$$\hat{\mathbf{x}}_{n|n-1} = \int \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \mathbf{u}_n \quad (5.51)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int [\tilde{\mathbf{f}}(\mathbf{c}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}] [\tilde{\mathbf{f}}(\mathbf{c}) + \mathbf{u}_n - \hat{\mathbf{x}}_{n|n-1}]^\top \\ &\quad \times \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \mathbf{Q} \end{aligned} \quad (5.52)$$

where, for this case

$$\tilde{\mathbf{f}}(\mathbf{c}) = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}\mathbf{c}) \quad (5.53)$$

with

$$\mathbf{D}_{n-1|n-1} \triangleq [\mathbf{P}_{n-1|n-1}^{\mathbf{xx}}]^{1/2} \quad (5.54)$$

In addition, (5.39)–(5.41) become

$$\hat{\mathbf{z}}_{n|n-1} = \int \tilde{\mathbf{h}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c}, \quad (5.55)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \int [\tilde{\mathbf{h}}(\mathbf{c}) - \hat{\mathbf{z}}_{n|n-1}] [\tilde{\mathbf{h}}(\mathbf{c}) - \hat{\mathbf{z}}_{n|n-1}]^\top \\ &\quad \times \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \mathbf{R} \end{aligned} \quad (5.56)$$

$$= \int [\tilde{\mathbf{f}}(\mathbf{c}) + \mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}] [\tilde{\mathbf{h}}(\mathbf{c}) - \hat{\mathbf{z}}_{n|n-1}] \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.57)$$

with

$$\tilde{\mathbf{h}}(\mathbf{c}) = \mathbf{h}(\hat{\mathbf{x}}_{n|n-1} + \mathbf{D}_{n|n-1}\mathbf{c}) \quad (5.58)$$

and

$$\mathbf{D}_{n|n-1} = [\mathbf{P}_{n|n-1}^{\mathbf{xx}}]^{1/2} \quad (5.59)$$

The set of equations (5.51)–(5.52) and (5.55)–(5.57) require the solutions for three general moment integrals of the form

$$\mathbf{I}(\tilde{\mathbf{f}}) \triangleq \int \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.60)$$

$$\mathbf{I}(\tilde{\mathbf{f}}\tilde{\mathbf{f}}^\top) \triangleq \int [\tilde{\mathbf{f}}(\mathbf{c}) - \mathbf{I}(\tilde{\mathbf{f}})] [\tilde{\mathbf{f}}(\mathbf{c}) - \mathbf{I}(\tilde{\mathbf{f}})]^\top \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.61)$$

and

$$\mathbf{I}(\mathbf{c}\tilde{\mathbf{f}}) \triangleq \int \mathbf{c}\tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.62)$$

5.3.2 General Methodology for Solving Gaussian-Weighted Integrals

For a general nonlinear function $\tilde{f}(\mathbf{c})$, the integrals (5.60)–(5.62) cannot be solved analytically. Generally speaking, numerical methods to solve these integrals are called *multiple integration rules* and each rule is designed to integrate a specific class of multidimensional polynomial approximations of $\tilde{f}(\mathbf{c})$. Similar approaches were taken in Ref. [8] and the references contained therein, where the method was described as a statistical linearization of $\mathbf{f}(\mathbf{x})$ through a series of transformations on the random variable \mathbf{x} and subsequent expansion of the transformed $\mathbf{f}(\mathbf{x})$ in a polynomial to facilitate evaluation of the expectation integrals.

The application of the affine transformation (5.46) is very important because it converts the general Gaussian-weighted integral into one that is *fully symmetric*. A fully symmetric density-weighted integral is one in which the density is symmetric about zero in all dimensions with the equal contours forming hyperspheres. In addition, the limits of the integral must also be symmetric about zero in all dimensions. This will be shown to greatly simplify numerical methods for approximating Gaussian-weighted integrals of nonlinear functions.

From the analytical expression for $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$, one can see that at specific vector point $\mathbf{c}^{(i)}$ in a multidimensional coordinate system, $\mathcal{N}(\mathbf{c}^{(i)}; \mathbf{0}, \mathbf{I})$ will have a specific numerical value. Now if we consider *any* vector point $\mathbf{c}^{(i)}$ of dimension n_x that lies on an n_x -dimensional hypersphere of *unit radius*, it is easy to show that $\mathcal{N}(\mathbf{c}^{(i)}; \mathbf{0}, \mathbf{I}) = e^{-1/2} / (2\pi)^{-n_x/2}$. This can be generalized to the statement that for a hypersphere of any given radius, all vector points $\mathbf{c}^{(i)}$ on the surface of that hypersphere will produce the same value for $\mathcal{N}(\mathbf{c}^{(i)}; \mathbf{0}, \mathbf{I})$. For the two-dimensional case, this can be seen in Figure 2.9 where each equal-value contours of $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$ form circles of constant radius.

If $\tilde{f}(\mathbf{c})$ is linear, that is, $\tilde{f}(\mathbf{c}) \rightarrow \mathbf{Fc}$, where \mathbf{F} is deterministic transition matrix that does not depend on \mathbf{c} , the solutions to (5.60)–(5.62) are straightforward and (5.51)–(5.57) reduce to the well-known linear Kalman filter, as will be shown in Chapter 6.

For an $\tilde{f}(\mathbf{c})$ that is nonlinear, the standard approach is to expand $\tilde{f}(\mathbf{c})$ analytically in a multidimensional polynomial formed from product monomials in each dimension. For example, in two dimensions, the polynomial expansion for $\tilde{f}(\mathbf{c})$, where $\mathbf{c} = [c_1, c_2]^\top$, is of the form

$$\begin{aligned}\tilde{\mathbf{f}}(c_1, c_2) &= \sum_{i_1=0}^{m_1} \sum_{i_2=0}^{m_2} a(i_1, i_2) c_1^{i_1} c_2^{i_2} \\ &= a(0, 0) + a(1, 0)c_1 + a(0, 1)c_2 + a(1, 1)c_1c_2 + a(2, 0)c_1^2 \\ &\quad + \cdots + a(m_1, m_2)c_1^{m_1}c_2^{m_2}\end{aligned}\tag{5.63}$$

Here, each term in the polynomial is a monomial of the form $a(i_1, i_2)c_1^{i_1}c_2^{i_2}$. For a multidimensional polynomial of degree d , each term contains a monomial $\prod_{j=1}^d c_j^{i_j}$ with $i_j \geq 0$ and $\sum_{j=1}^d i_j \leq d$. For the two-dimensional case cited above, the degree of the polynomial is $d = m_1 + m_2 \geq \sum_{j=1}^d i_j$.

For nonlinear functions $\tilde{\mathbf{f}}(\mathbf{c})$ we will be using polynomial expansions of $\tilde{\mathbf{f}}(\mathbf{c})$ in integrals of the form (5.60)–(5.62) to generate general multiple integration rules. We now introduce the definition of the *precision* p of an integration rule [1,9]. For integrals of the form (5.60) and (5.62), “a rule is said to have precision p if it integrates monomials up to degree p exactly, that is, monomials $\prod_{j=1}^p c_j^{i_j}$ with $i_j \geq 0$ and $\sum_{j=1}^p i_j \leq p$, but not exactly for some monomials of degree $\sum_{j=1}^p i_j \leq p + 1$ ” [1].

The classical approach for polynomial expansion is to expand $\tilde{f}(\mathbf{c})$ in a Taylor polynomial. In Chapter 7, it will be shown that keeping only the linear terms in the Taylor polynomial leads to the EKF. However, one of the main drawbacks of the EKF is the requirement to evaluate Jacobian matrices. For highly nonlinear functions, using a Taylor polynomial that includes second-order terms yields a more accurate solution, but it is one that adds the additional requirement to evaluate Hessian matrices. This leads to a solution formulation that results in a second-order EKF that is also addressed in Chapter 7.

An alternative is to replace the Jacobian and Hessian differential matrices by their multidimensional central finite difference approximations. This leads to the FDKF, the subject of Chapter 8. The FDKF is the first example of a *sigma point Kalman filter*. A sigma point Kalman filter is one that requires evaluation of $\tilde{f}(\mathbf{c})$ at deterministic sigma points that form a multidimensional grid. In addition, for sigma point Kalman filters all solutions to equations of the form (5.60)–(5.62) require evaluation of a set of moment equations, as will be shown in the chapters following. In the literature on methods for numerical integration, this conceptual methodology has been generalized to approximations for (5.60) of the form [10]

$$\mathbf{I}(\tilde{\mathbf{f}}) = \sum_{j=1}^{n_s} w_j \tilde{\mathbf{f}}\left(\mathbf{c}^{(j)}\right) \quad \text{with } \tilde{\mathbf{f}}(\mathbf{c}) \in \mathbb{R}^{n_x}\tag{5.64}$$

where the weights w_j , the sigma points $\mathbf{c}^{(j)}$, and the number of points n_s , are determined by the integration method chosen. The choices for w_j and $\mathbf{c}^{(j)}$ are independent of the function $\tilde{\mathbf{f}}$. If the dimension of the integration region is one, that is, $n_x = 1$, the approximation is called a quadrature formula. If $n_x \geq 2$, the approximation is called a cubature formula. There are two classes of cubature formulas: number theoretic methods and polynomial-based methods. The first class requires that the integration (sigma) points be distributed uniformly on a multidimensional grid. Methods of the second class are designed to be *exact* for some set of orthogonal polynomials. The FDKF is an example of the first class.

In what follows, two sets of vector points will be used that lie on the same hypersphere. For the first set, consider a vector point $\mathbf{c}^{(i)} = q [1, 0, \dots, 0]^\top = q\mathbf{r}^{(i)}$, where q is the radius of a hypersphere and $\mathbf{r}^{(i)}$ is a unit vector *along one of the dimensional axes*. Additional points that lie on the hypersphere along different coordinate axes can be generated from this first point by a change of sign and/or a permutation of dimension. For example, the points $\mathbf{c}^{(j)} = q [-1, 0, \dots, 0]^\top$ or $\mathbf{c}^{(j)} = q [0, \dots, 1, \dots, 0]^\top$ would also lie on the same hypersphere as the original point. In fact, it follows immediately that there will be two points along each axis, one positive and one negative, such that there are a total of $2n_x$ such vector points. We can define this set of vector points as $q\mathbf{r}$, with \mathbf{r} being the set of positive and negative unit vector along the n_x Cartesian axes. This approach leads to the UKF that will be addressed in Chapter 9.

The symmetry can also be expressed in sets of vector points that lie *equally spaced in angle* (instead of along the axes) on a hypersphere. These vector points will produce a symmetric set radiating from the origin to the vertex points of a multidimensional simplex, all of which will lie on the same hypersphere. Once again, if one point and the rotation angle between points is specified, all points of this set can be generated from the original point. We will show that this set contains only $n_x + 2$ points. Such an approach is taken in Chapter 10 where the SSKF will be developed.

There are, in fact, many such sets of points that can act as generators. For example, on a square grid (not on a hypersphere), the two vector points $\mathbf{c}^{(i)} = q [1, 1, 0, \dots, 0]^\top$ and $\mathbf{c}^{(i)} = q [1, 1, 1, \dots, 1]^\top$ will each act as a generator for a different set of symmetric points. But for these cases, not all values of $\mathcal{N}(\mathbf{c}^{(i)}; \mathbf{0}, \mathbf{I})$ will be identical. More will be discussed about these generator vector points in Chapter 11 where the orthogonal Hermite polynomials are used to develop the GHKF.

The restrictions imposed by the degree of the polynomial used to approximate $\tilde{\mathbf{f}}(\mathbf{c})$ lead to a requirement to satisfy a set of moment equations of the form

$$\sum_{j=1}^{n_s} w_j \tilde{\mathbf{f}}_i \left(\mathbf{c}^{(j)} \right) = \int \tilde{\mathbf{f}}_i (\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.65)$$

where $\tilde{\mathbf{f}}_i(\mathbf{c})$ represents all terms in the polynomial expansion of $\tilde{\mathbf{f}}(\mathbf{c})$ that are of order i . This set of moment equations lead to a system of nonlinear equations that can be solved for the unknowns $\{w_j, \mathbf{c}^{(j)}\}$. For matrix equations, as in (5.61). This becomes

$$\sum_{j=1}^{n_s} w_j \tilde{\mathbf{f}}_i \left(\mathbf{c}^{(j)} \right) \tilde{\mathbf{f}}_l \left(\mathbf{c}^{(j)} \right) = \int \tilde{\mathbf{f}}_i (\mathbf{c}) \tilde{\mathbf{f}}_l (\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (5.66)$$

For a more detailed understanding of these numerical methods, see books by Davis and Rabinowitz [4] and Evans and Swartz [5] and any of the other referenced material.

REFERENCES

1. Wu Y, Hu D, Wu M, Hu X. A numerical-integration perspective on Gaussian filters. *IEEE Trans. Sig. Proc.* 2006;54(8):2910–2921.
2. McNamee J, Stenger F. Construction of fully symmetric numerical integration formulas. *Numerische Math.* 1967;10:327–344.
3. Lerner UN. Hybrid Bayesian Networks for Reasoning About Complex Systems, Dissertation. Department of Computer Science, Stanford University; October 2002.
4. Davis PJ, Rabinowitz P. *Methods of Numerical Integration*, Reprint of Second Edition. Minneola, NY: Dover Publications; 1984.
5. Evans M, Swartz T. *Approximating Integrals via Monte Carlo and Deterministic Methods*. Oxford Statistical Science Series: 20, Oxford University Press; 2000, Reprinted 2005.
6. Ito K, Xiong K. Gaussian filters for nonlinear filtering problems. *IEEE Trans. Automatic Control*. 2000;45(5):910–927.
7. Marple SL. *Digital Spectral Analysis with Applications*. Prentice-Hall; 1987.
8. Gelb A. *Applied Optimal Estimation*. The MIT Press; 1974.
9. Stroud AK. *Approximate Calculation of Multiple Integrals*. Englewood cliffs, NJ: Prentice Hall; 1971.
10. Cool R. Advances in multidimensional integration. *J. Comput. and Appl. Math.* 2002; 149:1–12.

6

THE LINEAR CLASS OF KALMAN FILTERS

The recursive linear Kalman filter (LKF) was first introduced by Kalman [1,2], with the books [3–5] offering a more in depth discussion of the method with application examples. Newer editions or reprints of the original books are still available. For many estimation problems, either the dynamic or observation equations are linear and in many situations both are linear. Therefore, we will treat the linear dynamic or observation equation cases separately and then combine them into the linear Kalman filter.

6.1 LINEAR DYNAMIC MODELS

For a linear dynamic model, (5.1) can be written as

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{u}_n + \mathbf{v}_{n-1} \quad (6.1)$$

where \mathbf{F} is a deterministic transition matrix that is independent of time. For most cases, the control factor \mathbf{u}_n is not needed but we will include it in what follows for the sake of completion.

Putting (6.1) into (5.37) we obtain

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F} \int \mathbf{x}_{n-1} \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\mathbf{xx}}) d\mathbf{x}_{n-1} + \mathbf{u}_n \quad (6.2)$$

which reduces to

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}\hat{\mathbf{x}}_{n-1|n-1} + \mathbf{u}_n \quad (6.3)$$

Similarly, putting (6.1) into (5.38), yields

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int \mathbf{F} [\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n|n-1}]^\top \mathbf{F}^\top \\ &\quad \times \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\mathbf{xx}}) d\mathbf{x}_{n-1} + \mathbf{Q} \\ &= \mathbf{F} \left\{ \int [\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n|n-1}]^\top \right. \\ &\quad \left. \times \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\mathbf{xx}}) d\mathbf{x}_{n-1} \right\} \mathbf{F}^\top + \mathbf{Q} \end{aligned} \quad (6.4)$$

But, since

$$\begin{aligned} \mathbf{P}_{n-1|n-1}^{\mathbf{xx}} &= \int [\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n|n-1}]^\top \\ &\quad \times \mathcal{N}(\mathbf{x}_{n-1}; \hat{\mathbf{x}}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\mathbf{xx}}) d\mathbf{x}_{n-1} \end{aligned} \quad (6.5)$$

for a linear dynamic model it follows immediately that

$$\mathbf{P}_{n|n-1}^{\mathbf{xx}} = \mathbf{F}\mathbf{P}_{n-1|n-1}^{\mathbf{xx}}\mathbf{F}^\top + \mathbf{Q} \quad (6.6)$$

where $\mathbf{Q} = \mathcal{E}(\mathbf{v}_{n-1}\mathbf{v}_{n-1}^\top)$ is the dynamic noise covariance matrix.

6.2 LINEAR OBSERVATION MODELS

If the observation equation is linear, assume that the observation equation can be written as

$$\mathbf{z}_n = \mathbf{H}\mathbf{x}_n + \mathbf{w}_n \quad (6.7)$$

where \mathbf{H} is an $n_z \times n_x$ deterministic matrix that is independent of \mathbf{x}_n and of time. Therefore, we can write

$$\mathbf{h}(\mathbf{x}_n) = \mathbf{H}\mathbf{x}_n \quad (6.8)$$

Assuming that the observation noise \mathbf{w}_n is a zero mean process, it follows immediately that (5.39) and (5.40) reduce to

$$\hat{\mathbf{z}}_{n|n-1} = \mathbf{H}\hat{\mathbf{x}}_{n|n-1} \quad (6.9)$$

and

$$\mathbf{P}_{n|n-1}^{\mathbf{zz}} = \mathbf{H}\mathbf{P}_{n|n-1}^{\mathbf{xx}}\mathbf{H}^\top + \mathbf{R} \quad (6.10)$$

where $\mathbf{R} = \mathcal{E}(\mathbf{w}_n\mathbf{w}_n^\top)$ is the observation noise covariance matrix.

Finally, using (6.7) and (6.9) in (5.57) we obtain

$$\begin{aligned}
 \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \int [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}]^\top \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\mathbf{xx}}) d\mathbf{x}_n \\
 &= \int [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{Hx}_n + \mathbf{w}_n - \mathbf{H}\hat{\mathbf{x}}_{n|n-1}]^\top \\
 &\quad \times \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\mathbf{xx}}) d\mathbf{x}_n \\
 &= \left\{ \int [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}]^\top \right. \\
 &\quad \left. \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\mathbf{xx}}) d\mathbf{x}_n \right\} \mathbf{H}^\top \\
 &\quad + \int [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}] \mathbf{w}_n^\top \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}^{\mathbf{xx}}) d\mathbf{x}_n \mathbf{H}^\top \quad (6.11)
 \end{aligned}$$

This reduces to

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \mathbf{P}_{n|n-1}^{\mathbf{xx}} \mathbf{H}^\top \quad (6.12)$$

6.3 THE LINEAR KALMAN FILTER

When both the dynamic and observation processes are linear, Bayesian estimation reduces to the LKF. Note that it can be shown that the prediction equations can be derived directly from (5.3) to (5.7) using general distributions for the posterior and prior densities, making the linear Kalman filter applicable for *any* density. Only the requirements of linearity of the dynamic and observation equations need to be applied. However, this requires the density function to be of known analytical form so that the moment integrals can be evaluated in some manner. In many cases this has proven to be an insurmountable requirement.

The complete LKF process is presented in Table 6.1.

6.4 APPLICATION OF THE LKF TO DIFAR BUOY BEARING ESTIMATION

The LKF can be applied to a single DIFAR buoy using its bearing observations to estimate a state vector that includes bearing and bearing rate. That is, let the state vector be defined as

$$\mathbf{x}_n \triangleq [\theta_n, \dot{\theta}_n]^\top \quad (6.13)$$

where the bearing θ_n is the angle between the y-axis, which points to true North, and the line drawn from an origin at the buoy to the target ship, with the convention

TABLE 6.1 Linear Kalman Filter Process

Step 1.	Filter initialization:	Initialize $\hat{\mathbf{x}}_0$ and $\mathbf{P}_{0 n}^{\text{xx}}$
Step 2.	State vector prediction:	$\hat{\mathbf{x}}_{n n-1} = \mathbf{F}\hat{\mathbf{x}}_{n-1 n-1} + \mathbf{u}_n,$ $\mathbf{P}_{n n-1}^{\text{xx}} = \mathbf{F}\mathbf{P}_{n-1 n-1}^{\text{xx}}\mathbf{F}^\top + \mathbf{Q}$
Step 3.	Observation-related prediction:	$\hat{\mathbf{z}}_{n n-1} = \mathbf{H}\hat{\mathbf{x}}_{n n-1},$ $\mathbf{P}_{n n-1}^{\text{zz}} = \mathbf{H}\mathbf{P}_{n n-1}^{\text{xx}}\mathbf{H}^\top + \mathbf{R}$ $\mathbf{P}_{n n-1}^{\text{xz}} = \mathbf{P}_{n n-1}^{\text{xx}}\mathbf{H}^\top$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\text{xz}} (\mathbf{P}_{n n-1}^{\text{zz}})^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1})$ $\mathbf{P}_{n n}^{\text{xx}} = \mathbf{P}_{n n-1}^{\text{xx}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\text{zz}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\text{xx}}$ to a Track File
Step 6.	Return to Step 2.	

that $-180 \text{ deg} < \theta \leq 180 \text{ deg}$. $\dot{\theta}_n$ is the bearing rate of change. For this simple problem, the control variable \mathbf{u}_n is not needed so the dynamic transition equation is given by

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{v}_{n-1} \quad (6.14)$$

with

$$\mathbf{F} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \quad (6.15)$$

and \mathbf{v}_{n-1} a zero-mean Gaussian random dynamic acceleration noise process defined by $\mathbf{v}_{n-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, where

$$\mathbf{Q} = q \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix} \quad (6.16)$$

The \mathbf{Q} used here is the dynamic noise covariance of a continuous noise process with q , the variance of the bearing acceleration noise, set at 0.1 for this example. A complete derivation of \mathbf{Q} for the continuous noise model is beyond the scope of this book, but the interested reader can find it in Ref. [6]. The units of q can be obtained by

examining the state covariance prediction equation (6.6), noting that

$$\begin{aligned} \begin{bmatrix} \sigma_\theta^2 & \sigma_{\theta\dot{\theta}}^2 \\ \sigma_{\theta\dot{\theta}}^2 & \sigma_{\dot{\theta}}^2 \end{bmatrix}_{n|n-1} &= \mathbf{P}_{n|n-1}^{\mathbf{xx}} = \mathbf{F} \mathbf{P}_{n-1|n-1}^{\mathbf{xx}} \mathbf{F}^\top + \mathbf{Q} \\ &= \mathbf{F} \begin{bmatrix} \sigma_\theta^2 & \sigma_{\theta\dot{\theta}}^2 \\ \sigma_{\theta\dot{\theta}}^2 & \sigma_{\dot{\theta}}^2 \end{bmatrix}_{\mathbf{n-1}|\mathbf{n-1}} \mathbf{F}^\top \\ &\quad + q \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix} \end{aligned} \quad (6.17)$$

Now, the units of q can be determined by noting that qT must have the same units as $\sigma_{\dot{\theta}}^2$, or units of speed². Thus, q has units of speed²/time or acceleration² × time. Also, we observe that q is a standard deviation about a zero mean, so if we want the dynamic noise to represent a deviation from the constant velocity path, the standard deviation will take into account both positive and negative deviations and will therefore be twice the allowed deviation. So for our case, we are allowing a deviation in speed of $\sqrt{0.1}/2 = 0.16$ rad/s.

For a single DIFAR buoy, the output is a noisy bearing observation at each time t_n . The linear observation equation can therefore be written as

$$z_n = \theta_n = \mathbf{H} \mathbf{x}_n + w_n \quad (6.18)$$

with $w_n \sim \mathcal{N}(0, \sigma_\theta^2)$ and

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad (6.19)$$

Here, σ_θ is the standard deviation of the bearing measurements and is taken as a characteristic of all DIFAR buoys that is constant and independent of time.

The LKF can be initialized by using

$$\hat{\mathbf{x}}_0 = \begin{bmatrix} \tilde{\theta}_0 \\ \dot{\theta}_0 \end{bmatrix}$$

where $\tilde{\theta}_0$ is the first bearing observation from the DIFAR and arbitrarily setting $\dot{\theta}_0 = -0.09$ deg/s. We also let the initial state covariance matrix be given by

$$\mathbf{P}_0^{\mathbf{xx}} = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_{\dot{\theta}}^2 \end{bmatrix} \quad (6.20)$$

where we set $\sigma_\theta = -0.09$ deg/s and $\sigma_{\dot{\theta}}$ is set to the observation uncertainty, in degrees. Note that in our implementation of the LKF, all bearing and bearing rates are converted to radians and radians per second, respectively, but that transformation is not really needed for this linear example. The signal-to-noise ratio at the input to the DIFAR buoy was set to 20 dB with a time-bandwidth product of 90. This will ensure that the bearing observations are Gaussian.

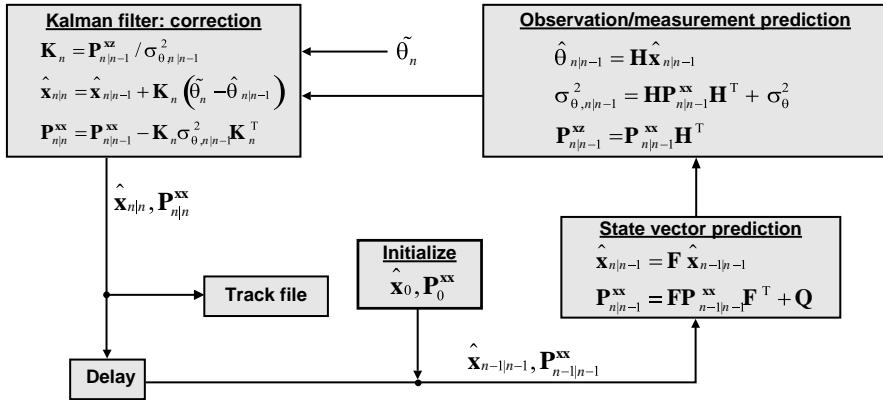


FIGURE 6.1 Block diagram of the LKF process for a single DIFAR buoy.

A block diagram of the LKF process for bearing tracking for a single DIFAR buoy is presented in Figure 6.1.

Note that in the Kalman filter correction step, we calculate the innovations (or sometimes called the measurement residual)

$$\eta_n = \tilde{\theta}_n - \hat{\theta}_{n|n-1} \quad (6.21)$$

Occasionally, the observation and prediction fall on opposite sides of the discontinuity at $-180/180$. This is referred to in the literature as the phase wrap problem and must

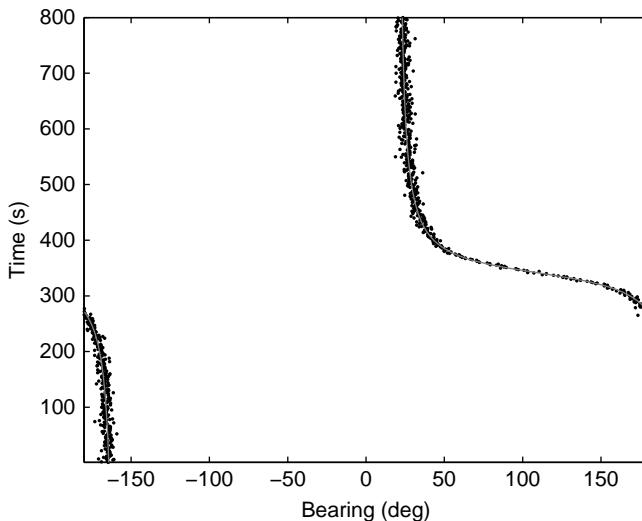


FIGURE 6.2 Comparison of the LKF bearing estimates with the true bearing for one DIFAR buoy.

be dealt with or the tracking filter will fail. The simplest way to deal with this is to calculate η_n and if it is greater than 180 deg, subtract 360 deg from $\hat{\theta}_{n|n-1}$ or if η_n is less than -180 deg, add 360 deg to $\hat{\theta}_{n|n-1}$. If neither of these conditions is true, do nothing.

A plot of the bearing estimated in this manner for one of the DIFAR buoys that exhibits this phase wrap problem is shown in Figure 6.2. In this figure, the estimated bearing outputs from the filter are represented by black dots. The superimposed gray line is the true bearings from the buoy to the target ship. This form of a plot is called a bearing-time record (BTR). It is obvious from the figure that the bearings are estimated quite well using the LKF for the chosen filter update rate (1 s) and bearing acceleration noise (0.1).

REFERENCES

1. Kalman RE. A new approach to linear filtering and prediction problems. *ASME* 1960;82 (Series D):35-40.
2. Kalman RE, Bucy RS. New results in linear filtering and prediction theory. *ASME* 1961;83 (Series D):95–107.
3. Papoulis A. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill; 1965.
4. Jazwinski AH. *Stochastic Processes and Filtering Theory*. Academic Press; 1970.
5. Gelb A. *Applied Optimal Estimation*. The MIT Press; 1974.
6. Bar Shalom Y, Li RX, Kirubarajan T. *Estimation with Application to Tracking and Navigation: Theory, Algorithms and Software*. Wiley; 2001.

7

THE ANALYTICAL LINEARIZATION CLASS OF KALMAN FILTERS: THE EXTENDED KALMAN FILTER

For mildly nonlinear and smooth (differentiable) functions, analytical methods can be used to linearize the nonlinear equations, making the linear Kalman filter structure available for use with nonlinear dynamic and/or observation equations. To accomplish this linearization, the nonlinear function $\tilde{f}(c)$ in (5.51)–(5.57) can be expanded in a multidimensional Taylor polynomial about the mean value $\hat{\mathbf{c}} = \mathbf{0}$ leading to the extended Kalman filter [1,2]. To make understanding of the method easier, and for future reference, we will develop the scalar extended Kalman filter (EKF) first and then move to the multidimensional case.

7.1 ONE-DIMENSIONAL CONSIDERATION

First, for the one-dimensional case, the state prediction equations (5.51) and (5.52) become

$$\hat{x}_{n|n-1} = \int \tilde{f}(c) \mathcal{N}(c; 0, 1) dc + u_n \quad (7.1)$$

$$\sigma_{xx,n|n-1}^2 = \int [\tilde{f}(c) + u_n - \hat{x}_{n|n-1}]^2 \mathcal{N}(c; 0, 1) dc + \sigma_{v,n}^2 \quad (7.2)$$

with

$$\tilde{f}(c) = f(\hat{x}_{n-1|n-1} + \sigma_{xx,n-1|n-1} c) \quad (7.3)$$

From (5.55) to (5.57), for the one-dimensional case, the observation prediction equations become

$$\hat{z}_{n|n-1} = \int \tilde{h}(c) \mathcal{N}(c; 0, 1) dc \quad (7.4)$$

$$\sigma_{zz,n|n-1}^2 = \int \tilde{h}^2(c) \mathcal{N}(c; 0, 1) dc - \hat{z}_{n|n-1}^2 + \sigma_{w,n}^2 \quad (7.5)$$

$$\sigma_{xz,n|n-1}^2 = \sigma_{xx,n|n-1} \int c \tilde{h}(c) \mathcal{N}(c; 0, 1) dc \quad (7.6)$$

with

$$\tilde{h}(c) = h(\hat{x}_{n|n-1} + \sigma_{xx,n|n-1} c) \quad (7.7)$$

7.1.1 One-Dimensional State Prediction

A general one-dimensional M th-order Taylor polynomial was presented in equation (2.38). For a third-order one-dimensional Taylor polynomial for $\tilde{f}(c)$ expanded about the zero-mean value of \hat{c} , (2.38) becomes

$$\begin{aligned} \tilde{f}(c) &= \tilde{f}(0) + \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0} c + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0} c^2 \\ &\quad + \frac{1}{6} \left[\frac{d^3}{dc^3} \tilde{f}(c) \right]_{c=0} c^3 \end{aligned} \quad (7.8)$$

Inserting this into (7.1) yields

$$\begin{aligned} \hat{x}_{n|n-1} &= \tilde{f}(0) \int \mathcal{N}(c; 0, 1) dc \\ &\quad + \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0} \int c \mathcal{N}(c; 0, 1) dc \\ &\quad + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0} \int c^2 \mathcal{N}(c; 0, 1) dc \\ &\quad + \frac{1}{6} \left[\frac{d^3}{dc^3} \tilde{f}(c) \right]_{c=0} \int c^3 \mathcal{N}(c; 0, 1) dc \\ &\quad + u_n \end{aligned} \quad (7.9)$$

Evaluating the integrals using the one-dimensional moment equations (2.115) yields the one-dimensional prediction estimate of the state vector

$$\hat{x}_{n|n-1} = \tilde{f}(0) + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0} + u_n \quad (7.10)$$

Note that this estimate is accurate to a third-order polynomial in c , because the third-order term in (7.8) integrates to zero.

7.1.2 One-Dimensional State Estimation Error Variance Prediction

From (7.2), consider only the first term

$$I(c) = \int \left[\tilde{f}(c) + u_n - \hat{x}_{n|n-1} \right]^2 \mathcal{N}(c; 0, 1) dc \quad (7.11)$$

But from (7.1) we can write the progression

$$\begin{aligned} \hat{x}_{n|n-1} &= \int \tilde{f}(c) \mathcal{N}(c; 0, 1) dc + u_n \\ &= \int \tilde{f}(c) \mathcal{N}(c; 0, 1) dc + \tilde{f}(0) - \tilde{f}(0) + u_n \\ &= \tilde{f}(0) + \int \left[\tilde{f}(c) - \tilde{f}(0) \right] \mathcal{N}(c; 0, 1) dc + u_n \end{aligned} \quad (7.12)$$

Thus (7.11) becomes

$$\begin{aligned} I(c) &= \int \left\{ \left[\tilde{f}(c) - \tilde{f}(0) \right] - \int \left[\tilde{f}(\tau) - \tilde{f}(0) \right] \mathcal{N}(\tau; 0, 1) d\tau \right\}^2 \mathcal{N}(c; 0, 1) dc \\ &= \int \left[\tilde{f}(c) - \tilde{f}(0) \right]^2 \mathcal{N}(c; 0, 1) dc \\ &\quad - \left\{ \int \left[\tilde{f}(c) - \tilde{f}(0) \right] \mathcal{N}(c; 0, 1) dc \right\}^2 \end{aligned} \quad (7.13)$$

From the Taylor polynomial (7.8), keeping terms to second order, we can write

$$\left[\tilde{f}(c) - \tilde{f}(0) \right] = \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0} c + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0} c^2 \quad (7.14)$$

and

$$\begin{aligned} \left[\tilde{f}(c) - \tilde{f}(0) \right]^2 &= \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0}^2 c^2 + \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0} c^3 \\ &\quad + \frac{1}{4} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0}^2 c^4 \end{aligned} \quad (7.15)$$

Now, using (7.14) and (7.15) in (7.13) and evaluating the resultant integrals using the one-dimensional moment equations (2.115) yields

$$I(c) = \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0}^2 + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0}^2 \quad (7.16)$$

Thus, the state error variance becomes

$$\sigma_{xx,n|n-1}^2 = \left[\frac{d}{dc} \tilde{f}(c) \right]_{c=0}^2 + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0}^2 + \sigma_{v,n}^2 \quad (7.17)$$

From (7.15), we can see that this approximation is valid up to fifth order in c .

7.1.3 One-Dimensional Observation Prediction Equations

Derivation of $\hat{z}_{n|n-1}$ and $\sigma_{zz,n|n-1}^2$ follow the same procedure as that of the last two sections, resulting in

$$\hat{z}_{n|n-1} = \tilde{h}(0) + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{h}(c) \right]_{c=0} \quad (7.18)$$

$$\sigma_{zz,n|n-1}^2 = \left[\frac{d}{dc} \tilde{h}(c) \right]_{c=0}^2 + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{h}(c) \right]_{c=0}^2 + \sigma_{w,n}^2 \quad (7.19)$$

For the cross-correlation, we have

$$\begin{aligned} \sigma_{xz,n|n-1}^2 &= \sigma_{xx,n|n-1} \int c \tilde{h}(c) \mathcal{N}(c; 0, 1) dc \\ &= \sigma_{xx,n|n-1} \int c \left\{ \tilde{h}(0) + \left[\frac{d}{dc} \tilde{h}(c) \right]_{c=0} c \right. \\ &\quad \left. + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{h}(c) \right]_{c=0} c^2 \right\} \mathcal{N}(c; 0, 1) dc \\ &= \sigma_{xx,n|n-1} \left[\frac{d}{dc} \tilde{h}(c) \right]_{c=0} \end{aligned} \quad (7.20)$$

This latter estimate of the cross-correlation can be seen to be of third order in c .

7.1.4 Transformation of One-Dimensional Prediction Equations

Remembering the definition of the affine transformation (5.46), and using the chain rule for differentiation, we can write

$$\frac{d}{dc} = \frac{dx_{n-1}}{dc} \frac{d}{dx_{n-1}} = \sigma_{xx,n-1|n-1} \frac{d}{dx_{n-1}} \quad (7.21)$$

$$\begin{aligned} \frac{d^2}{dc^2} &= \left(\frac{dx_{n-1}}{dc} \right)^2 \frac{d^2}{dx_{n-1}^2} \\ &= \sigma_{xx,n-1|n-1}^2 \frac{d^2}{dx_{n-1}^2} \end{aligned} \quad (7.22)$$

Also from (7.3), it follows that $\tilde{f}(0) = f(\hat{x}_{n-1|n-1})$ and from (5.46) $c = 0 \rightarrow x_{n-1} = \hat{x}_{n-1|n-1}$. Now (7.10) becomes

$$\hat{x}_{n|n-1} = f(\hat{x}_{n-1|n-1}) + \frac{1}{2}\sigma_{xx,n-1|n-1}^2 \left[\frac{d^2}{dx_{n-1}^2} f(x_{n-1}) \right]_{x_{n-1}=\hat{x}_{n-1|n-1}} + u_n \quad (7.23)$$

In a similar manner, (7.17) becomes

$$\begin{aligned} \sigma_{xx,n|n-1}^2 &= \sigma_{xx,n-1|n-1}^2 \left[\frac{d}{dx_{n-1}} f(x_{n-1}) \right]_{x_{n-1}=\hat{x}_{n-1|n-1}}^2 \\ &+ \frac{1}{2}\sigma_{xx,n-1|n-1}^4 \left[\frac{d^2}{dx_{n-1}^2} f(x_{n-1}) \right]_{x_{n-1}=\hat{x}_{n-1|n-1}}^2 \\ &+ \sigma_{v,n}^2 \end{aligned} \quad (7.24)$$

For the observation prediction equations, instead of (7.21) and (7.22), we use the affine transformation (5.46), and the chain rule becomes

$$\frac{d}{dc} = \frac{dx_n}{dc} \frac{d}{dx_n} = \sigma_{xx,n|n-1} \frac{d}{dx_n} \quad (7.25)$$

and

$$\frac{d^2}{dc^2} = \sigma_{xx,n|n-1}^2 \frac{d^2}{dx_n^2} \quad (7.26)$$

Now, (7.18)–(7.20) transform into

$$\hat{z}_{n|n-1} = h(\hat{x}_{n|n-1}) + \frac{1}{2}\sigma_{xx,n|n-1}^2 \left[\frac{d^2}{dx_n^2} h(x_n) \right]_{x_n=\hat{x}_{n|n-1}} \quad (7.27)$$

$$\begin{aligned} \sigma_{zz,n|n-1}^2 &= \sigma_{xx,n|n-1}^2 \left[\frac{d}{dx_n} h(x_n) \right]_{x_n=\hat{x}_{n|n-1}}^2 \\ &+ \frac{1}{2}\sigma_{xx,n|n-1}^4 \left[\frac{d^2}{dx_n^2} f(x_n) \right]_{x_n=\hat{x}_{n|n-1}}^2 \\ &+ \sigma_{w,n}^2 \end{aligned} \quad (7.28)$$

and

$$\sigma_{xz,n|n-1}^2 = \sigma_{xx,n|n-1}^2 \left[\frac{d}{dx_n} h(x_n) \right]_{x_n=\hat{x}_{n|n-1}} \quad (7.29)$$

TABLE 7.1 One-Dimensional Extended Kalman Filter Process

Step 1.	Filter initialization:	Initialize \hat{x}_0 and $\sigma_{xx,0}^2$,
Step 2.	State vector prediction:	$\hat{x}_{n n-1} = f(\hat{x}_{n-1 n-1}) + u_n$ $\sigma_{xx,n n-1}^2 = \sigma_{xx,n-1 n-1}^2$ $\times \left[\frac{d}{dx_{n-1}} f(x_{n-1}) \right]_{x_{n-1}=\hat{x}_{n-1 n-1}}^2$ $+ \sigma_{v,n}^2$
Step 3.	Observation-related prediction:	$\hat{z}_{n n-1} = h(\hat{x}_{n n-1})$ $\sigma_{zz,n n-1}^2 = \sigma_{xx,n n-1}^2 \left[\frac{d}{dx_n} h(x_n) \right]_{x_n=\hat{x}_{n n-1}}^2 + \sigma_{w,n}^2$ $\sigma_{xz,n n-1}^2 = \sigma_{xx,n n-1}^2 \left[\frac{d}{dx_n} h(x_n) \right]_{x_n=\hat{x}_{n n-1}}$
Step 4.	Kalman filter update:	$K_n = \sigma_{xz,n n-1}^2 / \sigma_{zz,n n-1}^2$ $\hat{x}_{n n} = \hat{x}_{n n-1} + K_n (z_n^o - \hat{z}_{n n-1})$ $\sigma_{xx,n n}^2 = \sigma_{xx,n n-1}^2 - K_n^2 \sigma_{zz,n n-1}^2$
Step 5.	Store results	Store $\hat{x}_{n n}$ and $\sigma_{xx,n n}^2$ to a Track File
Step 6.	Return to Step 2.	

7.1.5 The One-Dimensional Linearized EKF Process

The EKF developed above contained terms that are of higher order than linear. The standard linear EKF can be obtained by truncating so that only the linear terms are maintained. This leads to an EKF that has the same form as the Linear Kalman filter developed in Chapter 6. Thus, in summary, the process for the one-dimensional linearized EKF is shown in Table 7.1. Note that for this one-dimensional case $\rho_{xz,n|n-1} = \rho_{zx,n|n-1}$. In the update step, z_n^o is the observation at time t_n .

7.2 MULTIDIMENSIONAL CONSIDERATION

For multidimensional functions with multidimensional arguments, the state prediction integrals for $\tilde{f}(\mathbf{c})$ are given by (5.51)–(5.57). In both the linear Kalman filter and the one-dimensional EKF it is obvious from (6.3) and (7.23) that the deterministic control function \mathbf{u}_n is additive for the state prediction and does not appear in any of the remaining filter equations. So, for what follows, we will ignore \mathbf{u}_n and add it back in if it is needed for the case studies.

The general multidimensional Taylor polynomial was presented in (2.63). For a *second-order* Taylor polynomial for $\tilde{f}(\mathbf{c})$ expanded about the zero-mean value of \mathbf{c} ,

using (2.69) in (2.63) results in

$$\begin{aligned}
 \tilde{\mathbf{f}}(\mathbf{c}) &= \sum_{i=0}^2 \frac{1}{i!} D_{\mathbf{c}}^i \tilde{\mathbf{f}}(\mathbf{c}) \\
 &= \tilde{\mathbf{f}}(\mathbf{0}) + \sum_{i=1}^{n_x} c_i \left[\frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\
 &\quad + \frac{1}{2} \sum_{i_1=0}^2 \cdots \sum_{i_{n_x}=0}^2 \binom{2}{i_1 \cdots i_{n_x}} c_1^{i_1} \cdots c_{n_x}^{i_{n_x}} \\
 &\quad \times \left[\frac{\partial^2}{\partial c_1^{i_1} \cdots \partial c_{n_x}^{i_{n_x}}} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}}
 \end{aligned} \tag{7.30}$$

It follows immediately that (7.30) can be rewritten as

$$\begin{aligned}
 \tilde{\mathbf{f}}(\mathbf{c}) &= \tilde{\mathbf{f}}(\mathbf{0}) + \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i^2 \\
 &\quad + \left[\sum_{i=1}^{n_x} \sum_{\substack{j=1 \\ j \neq i}}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i c_j
 \end{aligned} \tag{7.31}$$

7.2.1 The State Prediction Equation

From (5.51), using the Taylor polynomial (7.31), we can express the predictive first moment as

$$\begin{aligned}
 \hat{\mathbf{x}}_{n|n-1} &= \tilde{\mathbf{f}}(\mathbf{0}) \int \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \int c_i \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\
 &\quad + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \int c_i^2 \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\
 &\quad + \left[\sum_{i=1}^{n_x} \sum_{\substack{j=1 \\ j \neq i}}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \int c_i c_j \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c}
 \end{aligned} \tag{7.32}$$

Using the moment equations (2.116)–(2.121) this reduces to

$$\hat{\mathbf{x}}_{n|n-1} = \tilde{\mathbf{f}}(\mathbf{0}) + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (7.33)$$

If the third-order terms had been included in (7.31), they would have integrated to zero in (7.32), so (7.33) is valid to third order.

7.2.2 The State Covariance Prediction Equation

From (5.52), the covariance prediction equation is

$$\mathbf{P}_{n|n-1}^{\text{xx}} = \int [\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1}] [\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1}]^\top \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \mathbf{Q} \quad (7.34)$$

Noting that we can add and subtract $\tilde{\mathbf{f}}(\mathbf{0})$, (5.51) can be rewritten as

$$\hat{\mathbf{x}}_{n|n-1} = \tilde{\mathbf{f}}(\mathbf{0}) + \int [\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0})] \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (7.35)$$

Substituting (7.35) for $\hat{\mathbf{x}}_{n|n-1}$, (7.34) becomes

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\text{xx}} &= \int [\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0})] [\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0})]^\top \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &\quad - \left\{ \int [\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0})] \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \right\} \\ &\quad \times \left\{ \int [\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0})]^\top \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \right\} \\ &\quad + \mathbf{Q} \end{aligned} \quad (7.36)$$

Rewriting the second-order Taylor polynomial (7.31) leads to

$$\begin{aligned} \tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0}) &= \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i^2 \\ &\quad + \left[\sum_{i=1}^{n_x} \sum_{j=1, j \neq i}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i c_j \end{aligned} \quad (7.37)$$

Utilizing the moment equations (2.116)–(2.121) the second integral in (7.36) yields

$$\int \left[\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0}) \right] \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (7.38)$$

Similarly, from the second-order Taylor polynomial (7.31), we can write

$$\begin{aligned} & \left[\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0}) \right] \left[\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0}) \right]^T \\ &= \left\{ \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i^2 \right. \\ & \quad \left. + \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i c_j \right\} \\ & \quad \times \left\{ \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i^2 \right. \\ & \quad \left. + \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} c_i c_j \right\} \end{aligned} \quad (7.39)$$

Carrying out the outer product term by term, putting the results into the first integral in (7.36), and using the moment equations (2.116)–(2.121), results in

$$\begin{aligned} & \int \left[\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0}) \right] \left[\tilde{\mathbf{f}}(\mathbf{c}) - \tilde{\mathbf{f}}(\mathbf{0}) \right]^T \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &= \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ & \quad + \frac{3}{4} \left[\sum_{i=1}^{n_c} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_c} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ & \quad + \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \end{aligned} \quad (7.40)$$

It follows immediately that (7.36) becomes

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} &= \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad + \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad + \mathbf{Q} \end{aligned} \tag{7.41}$$

If all terms are kept, the state error covariance matrix will be valid to fifth order, since the fifth-order terms will all integrate to zero.

7.2.3 Observation Prediction Equations

In a similar fashion, the prediction estimates $\hat{\mathbf{z}}_{n|n-1}$ and $\mathbf{P}_{n|n-1}^{\mathbf{z}\mathbf{z}}$ are given by

$$\hat{\mathbf{z}}_{n|n-1} = \tilde{\mathbf{h}}(\mathbf{0}) + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{h}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \tag{7.42}$$

and

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{z}\mathbf{z}} &= \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{h}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{h}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{h}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{h}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad + \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{h}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{h}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad + \mathbf{R} \end{aligned} \tag{7.43}$$

Finally, $\mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{z}}$ is given by (5.57), which can be written as

$$\mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{z}} = \mathbf{D}_{n|n-1} \int \mathbf{c} \tilde{\mathbf{h}}_\mathbf{n}^\top(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \tag{7.44}$$

Using (7.31) to expand $\tilde{\mathbf{h}}_n(\mathbf{c})$ in a Taylor polynomial, this becomes

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \mathbf{D}_{n|n-1} \tilde{\mathbf{h}}^\top(\mathbf{0}) \int \mathbf{c} \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &\quad + \mathbf{D}_{n|n-1} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{h}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \int \mathbf{c} c_i \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &\quad + \mathbf{D}_{n|n-1} \left[\sum_{i=1}^{n_x} \sum_{\substack{j=1 \\ j \neq i}}^{n_x} \frac{\partial}{\partial c_i} \frac{\partial}{\partial c_j} \tilde{\mathbf{h}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ &\quad \times \int \mathbf{c} c_i c_j \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \end{aligned} \quad (7.45)$$

Evaluating the integrals results in

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \mathbf{D}_{n|n-1} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{h}}_i(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (7.46)$$

Note that this result is accurate to third order because the third-order term in \mathbf{c} is zero.

7.2.4 Transformation of Multidimensional Prediction Equations

From the definition of the trace of a matrix (2.5), we can write

$$\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \triangleq \text{trace} \{ \nabla_{\mathbf{c}} \nabla_{\mathbf{c}}^\top \} \quad (7.47)$$

Using the vector chain rule,

$$\nabla_{\mathbf{c}} = [\nabla_{\mathbf{c}} \mathbf{x}^\top] \nabla_{\mathbf{x}} \quad (7.48)$$

and from (2.107) it follows that

$$\nabla_{\mathbf{c}} \mathbf{x}^\top = \nabla_{\mathbf{c}} [\hat{\mathbf{x}} + \mathbf{D}\mathbf{c}]^\top = \mathbf{D}^\top \quad (7.49)$$

Thus (7.47) becomes

$$\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} = \text{trace} \{ \mathbf{D}^\top \nabla_{\mathbf{x}} \nabla_{\mathbf{x}}^\top \mathbf{D} \} \quad (7.50)$$

Since the trace is invariant under circular permutations, this can be rewritten as

$$\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} = \text{trace} \{ \mathbf{D} \mathbf{D}^\top \nabla_{\mathbf{x}} \nabla_{\mathbf{x}}^\top \} = \text{trace} \{ \mathbf{P}^{\mathbf{xx}} \hat{\mathbf{G}}_{n-1|n-1} \} \quad (7.51)$$

where $\hat{\mathbf{G}}_{n-1|n-1}$ is define as the Hessian matrix

$$\begin{aligned}\hat{\mathbf{G}}_{n-1|n-1} &\triangleq \nabla_{\mathbf{x}_{n-1}} \nabla_{\mathbf{x}_{n-1}}^T \\ &= \left[\begin{array}{ccc} \frac{\partial^2}{\partial x_{n-1,1}^2} & \cdots & \frac{\partial}{\partial x_{n-1,1}} \frac{\partial}{\partial x_{n-1,n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_{n-1,n_x}} \frac{\partial}{\partial x_{n-1,1}} & \cdots & \frac{\partial^2}{\partial x_{n-1,n_x}^2} \end{array} \right]_{\mathbf{x}_{n-1}=\hat{\mathbf{x}}_{n-1|n-1}}\end{aligned}\quad (7.52)$$

Here, $\partial/\partial x_{n-1,i}$ represents the partial with respect to the i th component of the vector \mathbf{x}_{n-1} .

For state vector prediction, from (5.46) it follows that when $\mathbf{c} = \mathbf{0}$, $\mathbf{x} = \hat{\mathbf{x}}_{n-1|n-1}$, and from (5.60), $\tilde{f}(\mathbf{0}) = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1})$. Now (7.33) can be written as

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) \\ &+ \frac{1}{2} \text{trace} \left\{ \mathbf{P}_{n-1|n-1}^{\mathbf{xx}} \hat{\mathbf{G}}_{n-1|n-1} \right\} \mathbf{f}(\mathbf{x}_{n-1}) \Big|_{\mathbf{x}_{n-1}=\hat{\mathbf{x}}_{n-1|n-1}}\end{aligned}\quad (7.53)$$

where $\text{trace}\{\mathbf{P}_{n-1|n-1}^{\mathbf{xx}} \hat{\mathbf{G}}_{n-1|n-1}\}$ operates on every element of $\mathbf{f}(\mathbf{x}_{n-1})$.

Following the same logic, (7.41) can be transformed into

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \mathbf{Q} + \hat{\mathbf{F}}_{n-1|n-1} \mathbf{P}_{n-1|n-1}^{\mathbf{xx}} \hat{\mathbf{F}}_{n-1|n-1}^T \\ &+ \frac{1}{2} \left[\text{trace} \left\{ \mathbf{P}_{n-1|n-1}^{\mathbf{xx}} \hat{\mathbf{G}}_{n-1|n-1} \right\} \mathbf{f}(\mathbf{x}_{n-1}) \right] \\ &\times \left[\text{trace} \left\{ \mathbf{P}_{n-1|n-1}^{\mathbf{xx}} \hat{\mathbf{G}}_{n-1|n-1} \right\} \mathbf{f}^T(\mathbf{x}_{n-1}) \right] \\ &+ \text{HOT}\end{aligned}\quad (7.54)$$

where the Jacobian $\hat{\mathbf{F}}_{n-1|n-1}$ is defined as

$$\begin{aligned}\hat{\mathbf{F}}_{n-1|n-1} &\triangleq [\nabla_{\mathbf{x}_{n-1}} \mathbf{f}^T(\mathbf{x}_{n-1})]^T \Big|_{\mathbf{x}_{n-1}=\hat{\mathbf{x}}_{n-1|n-1}} \\ &= \left[\begin{array}{ccc} \frac{\partial f_1}{\partial x_{n-1,1}} & \cdots & \frac{\partial f_1}{\partial x_{n-1,n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_x}}{\partial x_{n-1,1}} & \cdots & \frac{\partial f_{n_x}}{\partial x_{n-1,n_x}} \end{array} \right]_{\mathbf{x}_{n-1}=\hat{\mathbf{x}}_{n-1|n-1}}\end{aligned}\quad (7.55)$$

and HOT stands for the higher order cross-terms.

In a similar fashion, the observation prediction estimates $\hat{\mathbf{z}}_{n|n-1}$, $\mathbf{P}_{n|n-1}^{\mathbf{zz}}$ and $\mathbf{P}_{n|n-1}^{\mathbf{xz}}$ transform into

$$\hat{\mathbf{z}}_{n|n-1} = \mathbf{h}(\hat{\mathbf{x}}_{n|n-1}) + \frac{1}{2} \text{trace} \left\{ \mathbf{P}_{n|n-1}^{\mathbf{xx}} \hat{\mathbf{H}}_{n|n-1} \right\} \mathbf{h}(\mathbf{x}_n) \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}}\quad (7.56)$$

$$\begin{aligned}
\mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \mathbf{R} + \hat{\mathbf{H}}_{n|n-1} \mathbf{P}_{n|n-1}^{\mathbf{xx}} \hat{\mathbf{H}}_{n|n-1}^T \\
&+ \frac{1}{2} \left[\text{trace} \left\{ \mathbf{P}_{n|n-1}^{\mathbf{xx}} \hat{\mathbf{G}}_{n|n-1} \right\} \mathbf{h}(\mathbf{x}_n) \right] \\
&\times \left[\text{trace} \left\{ \mathbf{P}_{n|n-1}^{\mathbf{xx}} \hat{\mathbf{G}}_{n|n-1} \right\} \mathbf{h}^T(\mathbf{x}_n) \right] \\
&+ \text{HOT}
\end{aligned} \tag{7.57}$$

and

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \mathbf{P}_{n|n-1}^{\mathbf{xx}} \hat{\mathbf{H}}_{n|n-1}^T \tag{7.58}$$

where

$$\hat{\mathbf{H}}_{n|n-1} = [\nabla_{\mathbf{x}_n} \mathbf{h}^T(\mathbf{x}_n)]^T \Big|_{\mathbf{x}_n = \hat{\mathbf{x}}_{n|n-1}} = \begin{bmatrix} \frac{\partial h_1}{\partial x_{n,1}} & \cdots & \frac{\partial h_1}{\partial x_{n,n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{n_x}}{\partial x_{n,1}} & \cdots & \frac{\partial h_{n_x}}{\partial x_{n,n_x}} \end{bmatrix} \Big|_{\mathbf{x}_n = \hat{\mathbf{x}}_{n|n-1}} \tag{7.59}$$

and

$$\begin{aligned}
\hat{\mathbf{G}}_{n|n-1} &\triangleq \nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_n}^T \\
&= \begin{bmatrix} \frac{\partial^2}{\partial x_{n,1}^2} & \cdots & \frac{\partial}{\partial x_{n,1}} \frac{\partial}{\partial x_{n,n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_{n,n_x}} \frac{\partial}{\partial x_{n,1}} & \cdots & \frac{\partial^2}{\partial x_{n,n_x}^2} \end{bmatrix} \Big|_{\mathbf{x}_n = \hat{\mathbf{x}}_{n|n-1}}
\end{aligned} \tag{7.60}$$

7.2.5 The Linearized Multidimensional Extended Kalman Filter Process

The EKF equations developed above contained terms that are of higher order than linear. The standard *linear* EKF can be obtained by truncating so that only the linear terms are maintained. This leads to an EKF that has the same form as the Linear Kalman filter developed in Chapter 6. Thus, in summary, the process for the *linearized* EKF is shown in Table 7.2.

Comparison of the LKF prediction equation from Table 6.1 with the EKF prediction equations in Table 7.2 reveals that the only difference, in form, between the two is the added requirement for the EKF to compute the Jacobians $\hat{\mathbf{F}}_{n-1|n-1}$ and $\hat{\mathbf{H}}_{n|n-1}$. In the update step, \mathbf{z}_n^0 is the observation at time t_n .

7.2.6 Second-Order Extended Kalman Filter

Occasionally, the nonlinearities are severe and the linearized EKF fails to converge. For these cases, one approach is to use a second-order extended Kalman filter

TABLE 7.2 Multidimensional Extended Kalman Filter Process

Step 1.	Filter initialization:	Initialize $\hat{\mathbf{x}}_0$ and \mathbf{P}_0^{xx}
Step 2.	State vector prediction:	$\hat{\mathbf{x}}_{n n-1} = \mathbf{f}(\hat{\mathbf{x}}_{n-1 n-1})$ $\hat{\mathbf{F}}_{n-1 n-1} \triangleq [\nabla_{\mathbf{x}_{n-1}} \mathbf{f}^\top(\mathbf{x}_{n-1})]^\top \Big _{\mathbf{x}_{n-1}=\hat{\mathbf{x}}_{n-1 n-1}}$ $\mathbf{P}_{n n-1}^{\text{xx}} = \hat{\mathbf{F}}_{n-1 n-1} \mathbf{P}_{n-1 n-1}^{\text{xx}} \hat{\mathbf{F}}_{n-1 n-1}^\top + \mathbf{Q}$
Step 3.	Observation-related prediction:	$\hat{\mathbf{z}}_{n n-1} = \mathbf{h}(\hat{\mathbf{x}}_{n n-1})$ $\hat{\mathbf{H}}_{n n-1} = [\nabla_{\mathbf{x}_n} \mathbf{h}^\top(\mathbf{x}_n)]^\top \Big _{\mathbf{x}_n=\hat{\mathbf{x}}_{n n-1}}$ $\mathbf{P}_{n n-1}^{\text{zz}} = \hat{\mathbf{H}}_{n n-1} \mathbf{P}_{n n-1}^{\text{xx}} \hat{\mathbf{H}}_{n n-1}^\top + \mathbf{R}$ $\mathbf{P}_{n n-1}^{\text{xz}} = \mathbf{P}_{n n-1}^{\text{xx}} \hat{\mathbf{H}}_{n n-1}^\top$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\text{xz}} (\mathbf{P}_{n n-1}^{\text{zz}})^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1})$ $\mathbf{P}_{n n}^{\text{xx}} = \mathbf{P}_{n n-1}^{\text{xx}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\text{zz}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\text{xx}}$ to a Track File
Step 6.	Return to Step 2.	

(SOEKF) [2,3]. The SOEKF includes the second-order terms in the prediction equations. Thus, the linearized EKF prediction equations of Table 7.2 are replaced by the second-order prediction equations

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) \quad (7.61)$$

$$+ \frac{1}{2} \text{trace} \left\{ \mathbf{P}_{n-1|n-1}^{\text{xx}} \hat{\mathbf{G}}_{n-1|n-1} \right\} \mathbf{f}(\mathbf{x}_{n-1}) \Big|_{\mathbf{x}_{n-1}=\hat{\mathbf{x}}_{n-1|n-1}} \quad (7.62)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\text{xx}} &= \mathbf{Q} + \hat{\mathbf{F}}_{n-1|n-1} \mathbf{P}_{n-1|n-1}^{\text{xx}} \hat{\mathbf{F}}_{n-1|n-1}^\top \\ &+ \frac{1}{2} \left[\text{trace} \left\{ \mathbf{P}_{n-1|n-1}^{\text{xx}} \hat{\mathbf{G}}_{n-1|n-1} \right\} \mathbf{f}(\mathbf{x}_{n-1}) \right] \\ &\times \left[\text{trace} \left\{ \mathbf{P}_{n-1|n-1}^{\text{xx}} \hat{\mathbf{G}}_{n-1|n-1} \right\} \mathbf{f}^\top(\mathbf{x}_{n-1}) \right] \end{aligned} \quad (7.63)$$

$$\hat{\mathbf{z}}_{n|n-1} = \mathbf{h}(\hat{\mathbf{x}}_{n|n-1}) + \frac{1}{2} \text{trace} \left\{ \mathbf{P}_{n|n-1}^{\text{xx}} \hat{\mathbf{H}}_{n|n-1} \right\} \mathbf{h}(\mathbf{x}_n) \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}} \quad (7.64)$$

$$\begin{aligned}\mathbf{P}_{n|n-1}^{zz} &= \mathbf{R} + \hat{\mathbf{H}}_{n|n-1} \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} \hat{\mathbf{H}}_{n|n-1}^T \\ &\quad + \frac{1}{2} \left[\text{trace} \left\{ \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} \hat{\mathbf{G}}_{n|n-1} \right\} \mathbf{h}(\mathbf{x}_n) \right] \\ &\quad \times \left[\text{trace} \left\{ \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} \hat{\mathbf{G}}_{n|n-1} \right\} \mathbf{h}^T(\mathbf{x}_n) \right]\end{aligned}\quad (7.65)$$

$$\mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{z}} = \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} \hat{\mathbf{H}}_{n|n-1}^T \quad (7.66)$$

These are usually very difficult to implement because they require the computation of both the Jacobian and Hessian matrices.

7.3 AN ALTERNATE DERIVATION OF THE MULTIDIMENSIONAL COVARIANCE PREDICTION EQUATIONS

For future reference, an alternate set of EKF state *covariance* prediction equations will be derived here. In the next chapter, these will be turned into finite difference solutions that are used to show the relationship between the finite difference method and the unscented Kalman filter. For numerical integration methods *other than* the finite difference methods, this is the preferred method for evaluating covariance integrals.

Define the matrix

$$\tilde{\mathbf{g}}(\mathbf{c}) \triangleq \left[\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1} \right] \left[\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1} \right]^T \quad (7.67)$$

Now (7.34) can be rewritten as

$$\bar{\mathbf{P}}_{n|n-1}^{\mathbf{x}\mathbf{x}} = \int \tilde{\mathbf{g}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \mathbf{Q} \quad (7.68)$$

This has the exact same form as the state prediction equation (5.51), resulting in

$$\bar{\mathbf{P}}_{n|n-1}^{\mathbf{x}\mathbf{x}} = \tilde{\mathbf{g}}(\mathbf{0}) + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{g}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} + \mathbf{Q} \quad (7.69)$$

where $\bar{\mathbf{P}}$ has been used to distinguish this form from (7.41).

Similarly, the observation covariance prediction becomes

$$\bar{\mathbf{P}}_{n|n-1}^{zz} = \tilde{\mathbf{p}}(\mathbf{0}) + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{p}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} + \mathbf{R} \quad (7.70)$$

where

$$\tilde{\mathbf{p}}(\mathbf{c}) \triangleq \left[\tilde{\mathbf{h}}(\mathbf{c}) - \hat{\mathbf{z}}_{n|n-1} \right] \left[\tilde{\mathbf{h}}(\mathbf{c}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \quad (7.71)$$

The cross-covariance matrix prediction equation (5.41) can be transformed into

$$\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xz}} = \int \tilde{\mathbf{r}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (7.72)$$

where

$$\tilde{\mathbf{r}}(\mathbf{c}) \triangleq \left[\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1} \right] \left[\tilde{\mathbf{h}}(\mathbf{c}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \quad (7.73)$$

Thus

$$\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xz}} = \tilde{\mathbf{r}}(\mathbf{0}) + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{r}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (7.74)$$

By substituting (7.67) into (7.69) and carrying out the differentiation and comparing the results to the $\mathbf{P}_{n|n-1}^{\mathbf{xx}}$ found in Table 7.2, we find the relationship between $\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xx}}$ and the linearized EKF covariance prediction estimate $\mathbf{P}_{n|n-1}^{\mathbf{xx}}$

$$\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xx}} = \mathbf{P}_{n|n-1}^{\mathbf{xx}} - \frac{1}{4} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}^T(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (7.75)$$

This reveals that $\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xx}}$ will be more accurate than the *linearized* EKF version of $\mathbf{P}_{n|n-1}^{\mathbf{xx}}$ because $\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xx}}$ includes part of the higher order terms from the SOEKF. Similar results can be obtained for the observational covariance prediction estimates.

These forms of the covariance matrices will be used in the next chapter to develop a finite difference filter that is identical to the unscented Kalman filter to be presented in Chapter 9. Since we will not be using this form for the EKF itself, the derivation will be carried no further.

7.4 APPLICATION OF THE EKF TO THE DIFAR SHIP TRACKING CASE STUDY

In applying the EKF to the problem of tracking the Cartesian position and velocity of a ship as it transits a buoy field, a dynamic *model* of the ships kinematics over time is required along with a model linking the bearing observations to the ships position. Two additional things need to be discussed: how to initialize $(\hat{\mathbf{x}}_0, \mathbf{P}_0^{\mathbf{xx}})$ and how to choose a proper acceleration noise q .

7.4.1 The Ship Motion Dynamics Model

The dynamic model can be obtained in vector–matrix form by using the ships state vector defined by (4.1) and combining (4.2) and (4.3) into

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{v}_{n-1} \quad (7.76)$$

where we have added a Gaussian white noise acceleration noise term \mathbf{v}_{n-1} to allow for small variations and maneuvers in the ship's track. Here, we define the time-invariant transition matrix \mathbf{F} as

$$\mathbf{F} \triangleq \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.77)$$

and note that

$$\mathbf{Q} \triangleq \mathcal{E}\{\mathbf{v}_n \mathbf{v}_n^\top\} = q \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} & 0 & 0 \\ \frac{T^2}{2} & T & 0 & 0 \\ 0 & 0 & \frac{T^3}{3} & \frac{T^2}{2} \\ 0 & 0 & \frac{T^2}{2} & T \end{bmatrix} \quad (7.78)$$

7.4.2 The DIFAR Buoy Field Observation Model

Define the buoy field observation vector at time t_n as

$$\mathbf{z}_n = [\theta_{1,n}, \dots, \theta_{M,n}]^\top \quad (7.79)$$

The observation model for each buoy can be defined by (4.7). We then combine all buoys into the vector observation equation

$$\mathbf{z}_n = \mathbf{h}(\mathbf{x}_n) + \mathbf{w}_n \quad (7.80)$$

with

$$\theta_{m,n} = h_m(\mathbf{x}_n) + w_{m,n} \quad (7.81)$$

and

$$h_m(\mathbf{x}_n) = \tan^{-1} \left(\frac{r_n^x - x_m}{r_n^y - y_m} \right) \quad (7.82)$$

The covariance of the Gaussian white observation noise process \mathbf{w}_n is defined as

$$\mathbf{R} \triangleq \mathcal{E}\{\mathbf{w}_n \mathbf{w}_n^\top\} = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2^2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma_{M-1}^2 & 0 \\ 0 & 0 & \cdots & 0 & \sigma_M^2 \end{bmatrix} \quad (7.83)$$

with σ_m^2 the variance associated with bearing observations from the m th buoy. For our simulations, we will assume that $\sigma_m = \sigma = 3^\circ, \forall m$.

Since the dynamic equation is linear, the LKF state prediction equations from Table 6.1 can be used for the tracking filter. Thus the state prediction equation that will be used are, from Table 6.1

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}\hat{\mathbf{x}}_{n-1|n-1} \quad (7.84)$$

$$\mathbf{P}_{n|n-1}^{\text{xx}} = \mathbf{F}\mathbf{P}_{n-1|n-1}^{\text{xx}}\mathbf{F}^T + \mathbf{Q} \quad (7.85)$$

However, since the observation equations (7.82) for each buoy are obviously non-linear, the EKF observation prediction equations from Table 7.2 must be used, requiring the computation of the Jacobian $\hat{\mathbf{H}}_{n|n-1}$.

The Jacobian is defined by

$$\begin{aligned} \hat{\mathbf{H}}_{n|n-1} &= [\nabla_{\mathbf{x}_n} \mathbf{h}^T(\mathbf{x}_n)]^T \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}} \\ &= \left[\begin{bmatrix} \frac{\partial}{\partial r_n^x} \\ \frac{\partial}{\partial v_n^x} \\ \frac{\partial}{\partial r_n^y} \\ \frac{\partial}{\partial v_n^y} \end{bmatrix} \begin{bmatrix} h_1(\mathbf{x}_n) & \cdots & h_M(\mathbf{x}_n) \end{bmatrix} \right]^T \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}} \\ &= \begin{bmatrix} \frac{\partial h_1}{\partial r_n^x} & \cdots & \frac{\partial h_1}{\partial v_n^y} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_M}{\partial r_n^x} & \cdots & \frac{\partial h_M}{\partial v_n^y} \end{bmatrix} \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}} \end{aligned} \quad (7.86)$$

Carrying out the derivatives yields

$$\hat{\mathbf{H}}_{n|n-1} = \begin{bmatrix} \frac{\hat{r}_{n|n-1}^y - y_1}{R_{n|n-1}^2} & \cdots & \frac{\hat{r}_{n|n-1}^y - y_M}{R_{n|n-1}^2} \\ 0 & \cdots & 0 \\ -\frac{\hat{r}_{n|n-1}^x - x_1}{R_{n|n-1}^2} & \cdots & -\frac{\hat{r}_{n|n-1}^x - x_M}{R_{n|n-1}^2} \\ 0 & \cdots & 0 \end{bmatrix} \quad (7.87)$$

with

$$R_{n|n-1} = \sqrt{\left(\hat{r}_{n|n-1}^x\right)^2 + \left(\hat{r}_{n|n-1}^y\right)^2} \quad (7.88)$$

The observation prediction equations we use for the application of the EKF to the DIFAR case study will therefore be (from Table 7.2)

$$\hat{\mathbf{z}}_{n|n-1} = \mathbf{h}(\hat{\mathbf{x}}_{n|n-1}) \quad (7.89)$$

$$\mathbf{P}_{n|n-1}^{\text{zz}} = \hat{\mathbf{H}}_{n|n-1} \mathbf{P}_{n|n-1}^{\text{xx}} \hat{\mathbf{H}}_{n|n-1}^T + \mathbf{R} \quad (7.90)$$

$$\mathbf{P}_{n|n-1}^{\text{xz}} = \mathbf{P}_{n|n-1}^{\text{xx}} \hat{\mathbf{H}}_{n|n-1}^T \quad (7.91)$$

And, of course, we then use the standard Kalman filter update equations from Step 4 in Table 7.2.

7.4.3 Initialization for All Filters of the Kalman Filter Class

In a later chapter, we will be comparing the performance of all of the tracking filters of the Kalman filter class. In order to make sure that we are comparing apples to apples, we must use a common initialization for all of the tracking filters. First, we assume that the target ship is radially inbound toward the origin from the Southwest, as shown in Figure 7.1, starting from an initial range R of 4 nautical miles (nmi).

Taking an initial set of buoy measurements, $\{\theta_m, m = 1, \dots, M\}$, we initialize the state vector position components using

$$r_0^x = \text{mean}(r_m^x - R \sin \theta_m) \quad (7.92)$$

$$r_0^y = \text{mean}(r_m^y - R \cos \theta_m) \quad (7.93)$$

where (r_m^x, r_m^y) is the position of the m th buoy and the mean is taken over all buoys.

For the speed components we generate an estimate of the bearing to the target relative to the origin by averaging the first set of bearing measurements from all buoys $\theta_0 = [\sum \theta_m]/M$. We arbitrarily set the target ship speed at $v = 30$ knots and compute a target heading by noting that $\theta_0 \simeq \vartheta - \pi$ where ϑ is the target heading.

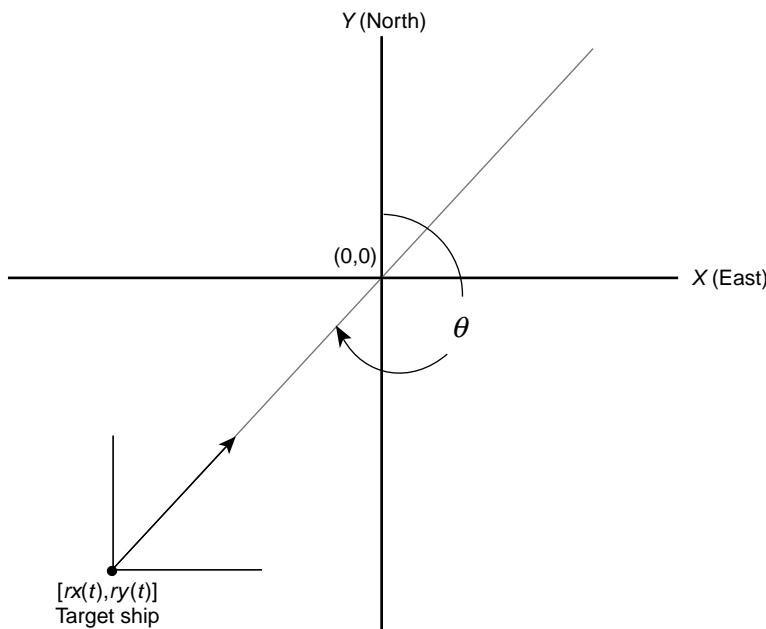


FIGURE 7.1 The geometry used for initialization.

The initial target speed components can now be computed from

$$v_0^x = v \sin \vartheta \quad (7.94)$$

$$v_0^y = v \cos \vartheta \quad (7.95)$$

Based on this initialization procedure, we see that there are two variables that must be guessed at, namely the range R and the speed v , with the initial bearing estimated from the first set of DIFAR element bearing measurements.

Since we don't have enough information to generate an initial covariance matrix from the measurement covariance, we will arbitrarily initialize the state error covariance with the diagonal matrix

$$\mathbf{P}_0^{\mathbf{x}\mathbf{x}} = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_{vx}^2 & 0 & 0 \\ 0 & 0 & \sigma_y^2 & 0 \\ 0 & 0 & 0 & \sigma_{vy}^2 \end{bmatrix} \quad (7.96)$$

with $\sigma_x^2 = \sigma_y^2 = 0.2 \text{ nmi}^2$ and $\sigma_{vx}^2 = \sigma_{vy}^2 = 3 \text{ knot}^2$.

7.4.4 Choosing a Value for the Acceleration Noise

As noted previously, q has units of either speed $^2/\text{time}$ or acceleration $^2 \times \text{time}$. For slow moving targets like ships, it is more convenient to use a variation in speed, while for fast moving targets like fighter planes or missiles, the use of acceleration variations (e.g., a 3 g turn) is more appropriate. For our slow moving ship target, we choose $q = 10^{-6}$ to allow for very small deviations in the ships velocity due to wave action and wind. This small number is a result of the assumption that the ship will not maneuver during the tracking interval. For a maneuvering vessel, a higher value of q would be more appropriate.

7.4.5 The EKF Tracking Filter Results

Running the tracking EKF filter for a filter update rate of one second, with the incoming ship's signal set with an SNR of 20 dB and a time-bandwidth product of 90, resulted in the track displayed in Figure 7.2. Although this track plot gives an intuitive feel for how the tracker algorithm is performing, the true performance measures will be discussed and compared in a later chapter. From this plot we notice that it takes the tracker some time before it converges on the true track, due to the inaccurate initialization. But it appears to converge rapidly on the true track (shown in gray). This tracker forgiveness for choosing an inaccurate initialization is important. The EKF (and all of the following nonlinear Gaussian Kalman filters) are almost independent of the initialization values chosen for R and v because the filter uses the observations to quickly correct for the initialization errors. We have run this filter for an SNR of 20 dB with range initializations that go from 20 nmi down to 3 nmi and with speed initializations from 10 to 30 knots with the filter quickly converging on the true track.

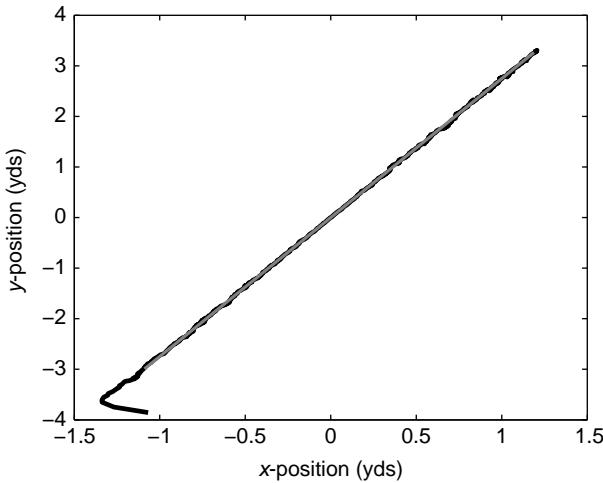


FIGURE 7.2 A comparison of the estimated track of a ship transiting the buoy field with the true track.

We noted in Chapter 4 that the bearing noise on the observations can become non-Gaussian when the SNR is low. To see what affect this has on the tracks estimated using a Gaussian EKF, in Figure 7.3 we present a plot of the EKF tracker output for SNRs from 20 to -5 dB in increments of 5 dB. In Figure 7.3, the black lines are the

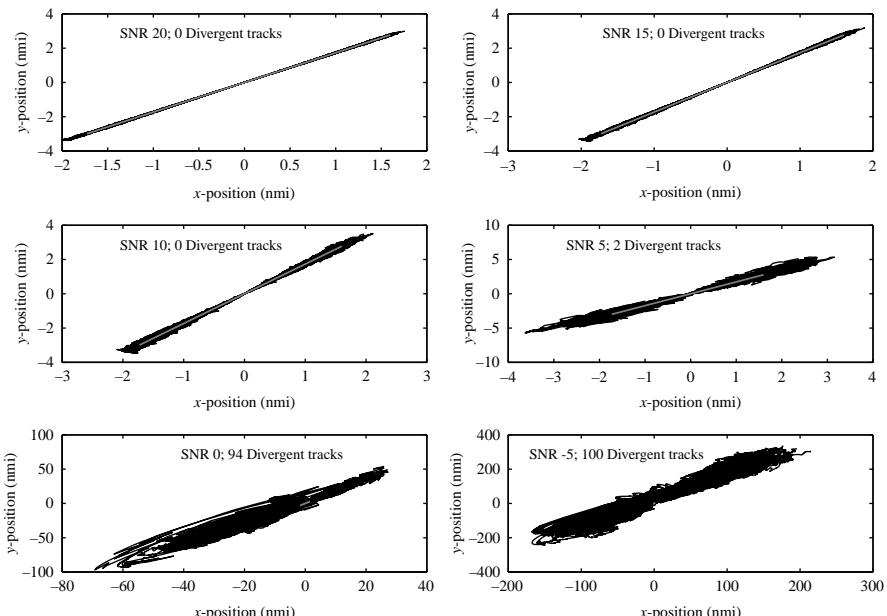


FIGURE 7.3 Comparison of the EKF tracker outputs for six SNRs.

filter track estimates for 100 Monte Carlo runs (all using identical initializations) and the gray line is the true track. One of the performance metrics that will be discussed in Chapter 14 is the percentage of divergent tracks in a set of Monte Carlo runs. In Figure 7.3, we have listed the number of divergent tracks for each SNR. Since there are 100 Monte Carlo runs, the number of divergent tracks is identical to the percentage of divergent tracks. It is obvious from the plots that the EKF has trouble tracking when the SNR is at 5 dB or below due to the excessive number of bearing observation outliers produced by the non-Gaussian noise.

REFERENCES

1. Jazwinski AH. *Stochastic Processes and Filtering Theory*. Academic Press; 1970.
2. Gelb A. *Applied Optimal Estimation*. The MIT Press; 1974.
3. Mahakababis A, Farooq M. A Second-order method for state estimation of non-linear dynamical systems. *Int. J. Control* 1971;14(4) 631–639.

8

THE SIGMA POINT CLASS: THE FINITE DIFFERENCE KALMAN FILTER

The linearized EKF yields reasonable estimation results if the nonlinearities are not very severe. For problems where the dynamic or observation transition functions are highly nonlinear, second-order terms can be included in the EKF but at the expense of adding the requirement of calculating the Hessian matrices. This added complication is sometimes very difficult to accomplish. One method to alleviate this difficulty is to replace the differentials of the EKF with their finite difference equivalents.

Schei was the first to propose using a central finite difference approach to linearization in nonlinear estimation algorithms [1,2]. While the EKF is a linearization of $\mathbf{f}(\mathbf{x})$ about the point $\hat{\mathbf{x}}$, Schei's method linearizes about the central difference points $\hat{\mathbf{x}} + \mathbf{D}\mathbf{q}$. Because Schei replaces only the Jacobian in the prediction equations for both the state (observation) vector and its covariance, the method is more accurate than the EKF only in the state (observation) vector prediction step.

A more accurate derivative-free estimation method for nonlinear systems was developed by Nørgaard et al. [3,4] by replacing the Taylor polynomial approximation of the nonlinear function by a multidimensional version of Stirling's interpolation formula. This allowed one to keep higher order terms in both the state (observation) vector and covariance prediction steps. The material presented below follow from the developments in Ref. [3].

The papers by Ito and Xiong [5] and Wu et al. [6] expand on these finite difference approximation methods and show how they fit into a framework of prediction techniques that implement the approximation of a nonlinear function using more general polynomial expansions.

In this chapter, a complete derivation of the finite difference Kalman filter will be presented, beginning with the scalar case and then extending to the multidimensional version.

8.1 ONE-DIMENSIONAL FINITE DIFFERENCE KALMAN FILTER

8.1.1 One-Dimensional Finite Difference State Prediction

The scalar derivation for the EKF state prediction equation in terms of $\tilde{f}(c)$ was given in (7.10), repeated here for clarity

$$\hat{x}_{n|n-1} = \tilde{f}(0) + \frac{1}{2} \left[\frac{d^2}{dc^2} \tilde{f}(c) \right]_{c=0} \quad (8.1)$$

As noted previously, this is accurate to third order in c .

Replacing the second-order derivative with its finite difference approximation from (2.47), (8.1) becomes

$$\hat{x}_{n|n-1} = \tilde{f}(0) + \frac{1}{2q^2} \left[\tilde{f}(q) - 2\tilde{f}(0) + \tilde{f}(-q) \right] \quad (8.2)$$

Gathering terms, this can be rewritten as

$$\hat{x}_{n|n-1} = \left[\frac{q^2 - 1}{q^2} \right] \tilde{f}(0) + \frac{1}{2q^2} \left[\tilde{f}(q) + \tilde{f}(-q) \right] \quad (8.3)$$

Let

$$w_0 \triangleq \frac{q^2 - 1}{q^2} \quad (8.4)$$

so that

$$q = \frac{1}{\sqrt{1 - w_0}} \quad (8.5)$$

and define the finite difference evaluation points as

$$c^{(j)} = qr^{(j)} \quad (8.6)$$

where

$$r^{(j)} \triangleq \begin{cases} 0, & j = 0 \\ [1] \in \mathbb{R}^1, & j = 1, 2 \end{cases} \quad (8.7)$$

See Section 2.2 for an explanation of the notation $[1] \in \mathbb{R}^1$. Using (8.4) and (8.6), (8.3) becomes

$$\hat{x}_{n|n-1} = w_0 \tilde{f}(c^{(0)}) + \frac{1 - w_0}{2} \left[\tilde{f}(c^{(1)}) + \tilde{f}(c^{(2)}) \right] \quad (8.8)$$

Replacing the continuous c defined in (7.3) with a discrete version, we can write

$$\tilde{f}(c^{(j)}) \triangleq f(\hat{x}_{n-1|n-1} + \sigma_{xx,n-1|n-1} c^{(j)}) \quad (8.9)$$

Defining one-dimensional finite difference sigma points, $\chi_{n-1|n-1}^{(j)}$, in terms of the finite-difference evaluation points, we obtain

$$\begin{aligned} \chi_{n-1|n-1}^{(j)} &\triangleq \hat{x}_{n-1|n-1} + \sigma_{xx,n-1|n-1} c^{(j)} \\ &= \begin{cases} \hat{x}_{n-1|n-1}, & j = 0 \\ \hat{x}_{n-1|n-1} + \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n-1|n-1}, & j = 1 \\ \hat{x}_{n-1|n-1} - \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n-1|n-1}, & j = 2 \end{cases} \end{aligned} \quad (8.10)$$

Now, the one-dimensional finite difference state prediction equation (8.8) becomes

$$\hat{x}_{n|n-1} = w_0 f(\chi_{n-1|n-1}^{(0)}) + \frac{1-w_0}{2} [f(\chi_{n-1|n-1}^{(1)}) + f(\chi_{n-1|n-1}^{(2)})] \quad (8.11)$$

Letting $w_1 = w_2 \triangleq (1-w_0)/2$, this can be written as the sigma point sum

$$\hat{x}_{n|n-1} = \sum_{j=0}^2 w_j f(\chi_{n-1|n-1}^{(j)}) \quad (8.12)$$

8.1.2 One-Dimensional Finite Difference State Variance Prediction

Replacing the first- and second-order derivatives in (7.17) with their finite difference equivalent from (2.44) and (2.47), the state error variance prediction equation becomes

$$\sigma_{xx,n|n-1}^2 = \frac{1}{4q^2} [\tilde{f}(q) - \tilde{f}(-q)]^2 + \frac{1}{2q^4} [\tilde{f}(q) - 2\tilde{f}(0) + \tilde{f}(-q)]^2 + \sigma_{v,n}^2 \quad (8.13)$$

Using (8.4), (8.6), and (8.10), this reduces to

$$\begin{aligned} \sigma_{xx,n|n-1}^2 &= \frac{1-w_0}{4} [f(\chi_{n-1|n-1}^{(1)}) - f(\chi_{n-1|n-1}^{(2)})]^2 \\ &\quad + \frac{(1-w_0)^2}{2} [f(\chi_{n-1|n-1}^{(1)}) - 2f(\chi_{n-1|n-1}^{(0)}) \\ &\quad + f(\chi_{n-1|n-1}^{(2)})]^2 + \sigma_{v,n}^2 \end{aligned} \quad (8.14)$$

Note that this is accurate to fifth order, as shown in Chapter 7.

8.1.3 One-Dimensional Finite Difference Observation Prediction Equations

Following the same procedure as in the previous two sections, it follows immediately that the one-dimensional observation prediction and cross-variance prediction equations are given by

$$\hat{z}_{n|n-1} = \sum_{j=0}^2 w_i h \left(\chi_{n|n-1}^{(j)} \right) \quad (8.15)$$

$$\begin{aligned} \sigma_{zz,n|n-1}^2 &= \frac{1-w_0}{4} \left[h \left(\chi_{n|n-1}^{(1)} \right) - h \left(\chi_{n|n-1}^{(2)} \right) \right]^2 \\ &\quad + \frac{(1-w_0)^2}{2} \left[h \left(\chi_{n|n-1}^{(1)} \right) - 2h \left(\chi_{n|n-1}^{(0)} \right) + h \left(\chi_{n|n-1}^{(2)} \right) \right]^2 \\ &\quad + \sigma_{w,n}^2 \end{aligned} \quad (8.16)$$

$$\sigma_{xz,n|n-1}^2 = \sigma_{xx,n|n-1}^2 \left\{ w_1 \left[h \left(\chi_{n|n-1}^{(1)} \right) - h \left(\chi_{n|n-1}^{(2)} \right) \right] \right\} \quad (8.17)$$

where the predictive one-dimensional finite difference sigma points $\{\chi_{n|n-1}^{(j)}, j = 0, 1, 2\}$ are given by

$$\chi_{n|n-1}^{(j)} = \begin{cases} \hat{x}_{n|n-1}, & j = 0 \\ \hat{x}_{n|n-1} + \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n|n-1}^2, & j = 1 \\ \hat{x}_{n|n-1} - \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n|n-1}^2, & j = 2 \end{cases} \quad (8.18)$$

8.1.4 The One-Dimensional Finite Difference Kalman Filter Process

(8.12) and (8.14)–(8.17) constitute the prediction portion of a one-dimensional finite difference Kalman filter that is accurate to third order for the state and observation predictions and fifth order for the covariance predictions. The procedure for the complete one-dimensional finite difference Kalman filter is presented in Table 8.1.

8.1.5 Simplified One-Dimensional Finite Difference Prediction Equations

Several simplifications can be made for these one-dimensional prediction equations. The easiest simplification is to take $w_0 = 0$. This removes the point at the origin and results in prediction equations with no free parameters. The prediction equations now

TABLE 8.1 One-Dimensional Finite Difference Kalman Filter Process

Step 1.	Filter initialization:	Set the parameter $0 \leq w_0 < 1$ Initialize \hat{x}_0 and $\sigma_{xx,0}^2$
Step 2.	State vector prediction:	$\chi_{n-1 n-1}^{(j)} = \begin{cases} \hat{x}_{n-1 n-1}, & j = 0 \\ \hat{x}_{n-1 n-1} + \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n-1 n-1}, & j = 1 \\ \hat{x}_{n-1 n-1} - \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n-1 n-1}, & j = 2 \end{cases}$
		$\hat{x}_{n n-1} = \sum_{j=0}^2 w_j f(\chi_{n-1 n-1}^{(j)})$
		$\sigma_{xx,n n-1}^2 = \frac{1-w_0}{4} \times [f(\chi_{n-1 n-1}^{(1)}) - f(\chi_{n-1 n-1}^{(2)})]^2 + \frac{(1-w_0)^2}{2} [f(\chi_{n-1 n-1}^{(1)}) - 2f(\chi_{n-1 n-1}^{(0)}) + f(\chi_{n-1 n-1}^{(2)})]^2 + \sigma_{v,n}^2$
Step 3.	Observation-related prediction:	$\chi_{n n-1}^{(j)} = \begin{cases} \hat{x}_{n n-1}, & j = 0 \\ \hat{x}_{n n-1} + \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n n-1}^2, & j = 1 \\ \hat{x}_{n n-1} - \frac{1}{\sqrt{1-w_0}} \sigma_{xx,n n-1}^2, & j = 2 \end{cases}$
		$\hat{z}_{n n-1} = \sum_{j=0}^2 w_j h(\chi_{n n-1}^{(j)})$
		$\sigma_{zz,n n-1}^2 = \frac{1-w_0}{4} \times [h(\chi_{n n-1}^{(1)}) - h(\chi_{n n-1}^{(2)})]^2 + \frac{(1-w_0)^2}{2} \times [h(\chi_{n n-1}^{(1)}) - 2h(\chi_{n n-1}^{(0)}) + h(\chi_{n n-1}^{(2)})]^2 + \sigma_{w,n}^2$
		$\sigma_{xz,n n-1}^2 = \sigma_{xx,n n-1}^2 \times \{w_1 [h(\chi_{n n-1}^{(1)}) - h(\chi_{n n-1}^{(2)})]\}$
Step 4.	Kalman filter update:	$K_n = \sigma_{xz,n n-1}^2 / \sigma_{zz,n n-1}^2$ $\hat{x}_{n n} = \hat{x}_{n n-1} + K_n (z_n^o - \hat{z}_{n n-1})$ $\sigma_{xx,n n}^2 = \sigma_{xx,n n-1}^2 - K_n^2 \sigma_{zz,n n-1}^2$
Step 5.	Store results	Store $\hat{x}_{n n}$ and $\mathbf{P}_{n n}^{\text{xx}}$ to a Track File
Step 6.	Return to Step 2.	

become

$$\hat{x}_{n|n-1} = \frac{1}{2} \left[f\left(\chi_{n-1|n-1}^{(1)}\right) + f\left(\chi_{n-1|n-1}^{(2)}\right) \right] \quad (8.19)$$

$$\begin{aligned} \sigma_{xx,n|n-1}^2 &= \frac{1}{4} \left[f\left(\chi_{n-1|n-1}^{(1)}\right) - f\left(\chi_{n-1|n-1}^{(2)}\right) \right]^2 \\ &\quad + \frac{1}{2} \left[f\left(\chi_{n-1|n-1}^{(1)}\right) - 2f\left(\chi_{n-1|n-1}^{(0)}\right) \right. \\ &\quad \left. + f\left(\chi_{n-1|n-1}^{(2)}\right) \right]^2 + \sigma_{v,n}^2 \end{aligned} \quad (8.20)$$

$$\hat{z}_{n|n-1} = \frac{1}{2} \left[h\left(\chi_{n|n-1}^{(1)}\right) + h\left(\chi_{n|n-1}^{(2)}\right) \right] \quad (8.21)$$

$$\begin{aligned} \sigma_{zz,n|n-1}^2 &= \frac{1}{4} \left[h\left(\chi_{n|n-1}^{(1)}\right) - h\left(\chi_{n|n-1}^{(2)}\right) \right]^2 \\ &\quad + \frac{1}{2} \left[h\left(\chi_{n|n-1}^{(1)}\right) - 2h\left(\chi_{n|n-1}^{(0)}\right) \right. \\ &\quad \left. + h\left(\chi_{n|n-1}^{(2)}\right) \right]^2 + \sigma_{w,n}^2 \end{aligned} \quad (8.22)$$

$$\sigma_{xz,n|n-1}^2 = \frac{1}{2} \sigma_{xx,n|n-1} \left[h\left(\chi_{n|n-1}^{(1)}\right) - h\left(\chi_{n|n-1}^{(2)}\right) \right] \quad (8.23)$$

with

$$\chi_{n-1|n-1}^{(j)} = \begin{cases} \hat{x}_{n-1|n-1} + \sigma_{xx,n-1|n-1}, & j = 1 \\ \hat{x}_{n-1|n-1} - \sigma_{xx,n-1|n-1}, & j = 2 \end{cases} \quad (8.24)$$

and

$$\chi_{n|n-1}^{(j)} = \begin{cases} \hat{x}_{n|n-1} + \sigma_{xx,n|n-1}, & j = 1 \\ \hat{x}_{n|n-1} - \sigma_{xx,n|n-1}, & j = 2 \end{cases} \quad (8.25)$$

Another simplification includes the truncation of some of the higher order terms. These methods will be discussed more detail when we treat the multidimensional case in the next section.

8.2 MULTIDIMENSIONAL FINITE DIFFERENCE KALMAN FILTERS

8.2.1 Multidimensional Finite Difference State Prediction

In manner similar to the approach taken in the one-dimensional case, we first repeat the multidimensional state prediction equation developed for the EKF given by (7.33)

$$\hat{\mathbf{x}}_{n|n-1} = \tilde{\mathbf{f}}(\mathbf{0}) + \sum_{j=1}^{n_x} \left[\frac{\partial^2}{\partial c_j^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (8.26)$$

Now, using the second-order finite difference term of multidimensional Stirling's polynomial given by equation (2.74) and letting $\mathbf{x} \rightarrow \mathbf{c}$ and $\mathbf{x}_0 = \mathbf{0}$, (8.26) becomes

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \tilde{\mathbf{f}}(\mathbf{0}) + \frac{1}{2q^2} \sum_{j=1}^{n_x} [\tilde{\mathbf{f}}(q\mathbf{e}_j) - 2\tilde{\mathbf{f}}(\mathbf{0}) + \tilde{\mathbf{f}}(-q\mathbf{e}_j)] \\ &= \left[\frac{q^2 - n_x}{q^2} \right] \tilde{\mathbf{f}}(\mathbf{0}) + \frac{1}{2q^2} \sum_{j=1}^{n_x} [\tilde{\mathbf{f}}(q\mathbf{e}_j) + \tilde{\mathbf{f}}(-q\mathbf{e}_j)]\end{aligned}\quad (8.27)$$

where \mathbf{e}_j is a unit vector along the Cartesian axis of the j th dimension and q is a step size that must be finite, real, and greater than zero.

Defining

$$w_0 \triangleq \frac{q^2 - n_x}{q^2} \quad (8.28)$$

leads to the identities

$$\frac{1}{q^2} = \frac{1 - w_0}{n_x} \quad (8.29)$$

and

$$q = \sqrt{\frac{n_x}{1 - w_0}} \quad (8.30)$$

Here, w_0 is a free parameter that determines the value of q . To maintain q as finite, real, and greater than zero, we must restrict w_0 to the range $0 \leq w_0 < 1$.

Equation (8.27) can now be written as

$$\hat{\mathbf{x}}_{n|n-1} = w_0 \tilde{\mathbf{f}}(\mathbf{0}) + \frac{1 - w_0}{2n_x} \sum_{j=1}^{n_x} \left[\tilde{\mathbf{f}}\left(\sqrt{\frac{n_x}{1 - w_0}} \mathbf{e}_j\right) + \tilde{\mathbf{f}}\left(-\sqrt{\frac{n_x}{1 - w_0}} \mathbf{e}_j\right) \right] \quad (8.31)$$

To simplify this even further, we utilize the multidimensional vector *generator* function [7,8] defined in Section 2.2. For example, in the four-dimensional case we have

$$\mathbf{r} = \begin{cases} [0] \in \mathbb{R}^4 \\ [1] \in \mathbb{R}^4 \end{cases} = \begin{cases} \mathbf{r}^{(0)} = [0, 0, 0, 0]^T \\ \mathbf{r}^{(1)} = [1, 0, 0, 0]^T = \mathbf{e}_1 \\ \mathbf{r}^{(2)} = [0, 1, 0, 0]^T = \mathbf{e}_2 \\ \mathbf{r}^{(3)} = [0, 0, 1, 0]^T = \mathbf{e}_3 \\ \mathbf{r}^{(4)} = [0, 0, 0, 1]^T = \mathbf{e}_4 \\ \mathbf{r}^{(5)} = [-1, 0, 0, 0]^T = -\mathbf{e}_1 \\ \mathbf{r}^{(6)} = [0, -1, 0, 0]^T = -\mathbf{e}_2 \\ \mathbf{r}^{(7)} = [0, 0, -1, 0]^T = -\mathbf{e}_3 \\ \mathbf{r}^{(8)} = [0, 0, 0, -1]^T = -\mathbf{e}_4 \end{cases} \quad (8.32)$$

These vector integration points form a four-dimensional grid of unit vectors along the axis of a Cartesian coordinate system and one additional point at the origin. If the point at the origin is excluded, these unit vectors will all fall on the contour of $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$ at the four-dimensional hypersphere radius of one. See Figure 2.9 for a graphical example in two dimensions.

For the general n_x -dimensional case we have

$$\mathbf{r} = \begin{cases} [0] \in \mathbb{R}^{n_x}, j = 0 \\ [1] \in \mathbb{R}^{n_x}, j = 1, \dots, 2n_x \end{cases} \quad (8.33)$$

where the number of vector points for $[1] \in \mathbb{R}^{n_x}$ is given by (2.30). A discrete \mathbf{c} can be now be defined as

$$\mathbf{c}^{(j)} = q\mathbf{r}^{(j)} = \sqrt{\frac{n_x}{1 - w_0}} \mathbf{r}^{(j)}, \quad j = 0, \dots, 2n_x \quad (8.34)$$

Now we can write equation (5.53) as

$$\tilde{\mathbf{f}}(\mathbf{c}^{(j)}) = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}\mathbf{c}^{(j)}) \quad (8.35)$$

where $\mathbf{D}_{n-1|n-1}$ was defined by (5.48). This leads to a definition of general *multidimensional finite difference sigma points*

$$\begin{aligned} \chi_{n-1|n-1}^{(j)} &\triangleq \hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}\mathbf{c}^{(j)} \\ &= \hat{\mathbf{x}}_{n-1|n-1} + \sqrt{\frac{n_x}{1 - w_0}} \mathbf{D}_{n-1|n-1}\mathbf{r}^{(j)}, \quad j = 0, \dots, 2n_x \end{aligned} \quad (8.36)$$

It follows immediately that (8.31) can be rewritten as

$$\hat{\mathbf{x}}_{n|n-1} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\chi_{n-1|n-1}^{(j)}) \quad (8.37)$$

with

$$w_j = \begin{cases} w_0, & j = 0 \\ \frac{1-w_0}{2n_x}, & j = 1, \dots, 2n_x \end{cases} \quad (8.38)$$

where $\sum_{j=0}^{2n_x} w_j = 1$. As noted previously, this is a third order approximation.

8.2.2 Multidimensional Finite Difference State Covariance Prediction

The expression for the multidimensional EKF predictive state error covariance matrix is given by (7.41), repeated here for clarity

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} = & \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{f}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ & + \frac{1}{2} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \frac{\partial^2}{\partial c_i^2} \tilde{\mathbf{f}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ & + \left[\sum_{i=1}^{n_x} \sum_{\substack{j=1 \\ j \neq i}}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \left[\sum_{i=1}^{n_x} \sum_{\substack{j=1 \\ j \neq i}}^{n_x} \frac{\partial^2}{\partial c_i \partial c_j} \tilde{\mathbf{f}}^\top(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \\ & + \mathbf{Q} \end{aligned} \quad (8.39)$$

Using the first- and second-order finite difference terms of multidimensional Stirling's polynomial for each dimension of $\tilde{\mathbf{f}}(\mathbf{c})$, given by equations (2.73) and (2.74), this becomes

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} = & \frac{1}{4q^2} \sum_{i=1}^{n_x} \sum_{j=1}^{n_x} [\tilde{\mathbf{f}}(q\mathbf{e}_i) - \tilde{\mathbf{f}}(-q\mathbf{e}_i)] [\tilde{\mathbf{f}}(q\mathbf{e}_j) - \tilde{\mathbf{f}}(-q\mathbf{e}_j)]^\top \\ & + \frac{1}{2q^4} \sum_{i=1}^{n_x} \sum_{j=1}^{n_x} [\tilde{\mathbf{f}}(q\mathbf{e}_i) - 2\tilde{\mathbf{f}}(\mathbf{0}) + \tilde{\mathbf{f}}(-q\mathbf{e}_i)] \\ & \times [\tilde{\mathbf{f}}(q\mathbf{e}_j) - 2\tilde{\mathbf{f}}(\mathbf{0}) + \tilde{\mathbf{f}}(-q\mathbf{e}_j)]^\top \\ & + \frac{1}{4q^4} \sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \sum_{k=1}^{n_x} \sum_{l=1}^{n_x} [\tilde{\mathbf{f}}(q\mathbf{e}_i + q\mathbf{e}_j) - \tilde{\mathbf{f}}(q\mathbf{e}_i - q\mathbf{e}_j) \\ & - \tilde{\mathbf{f}}(-q\mathbf{e}_i + q\mathbf{e}_j) + \tilde{\mathbf{f}}(-q\mathbf{e}_i - q\mathbf{e}_j)] \\ & \times [\tilde{\mathbf{f}}(q\mathbf{e}_k + q\mathbf{e}_l) - \tilde{\mathbf{f}}(q\mathbf{e}_k - q\mathbf{e}_l) \\ & - \tilde{\mathbf{f}}(-q\mathbf{e}_k + q\mathbf{e}_l) + \tilde{\mathbf{f}}(-q\mathbf{e}_k - q\mathbf{e}_l)]^\top \\ & + \mathbf{Q} \end{aligned} \quad (8.40)$$

With the aid of (8.28) and (8.36), this transforms into

$$\begin{aligned}
 \mathbf{P}_{n|n-1}^{\mathbf{xx}} = & \frac{1-w_0}{4n_x} \sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(i)}) - \mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(n_x+i)}) \right] \\
 & \times \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(j)}) - \mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(n_x+j)}) \right]^T \\
 & + \frac{1}{2} \left(\frac{1-w_0}{n_x} \right)^2 \\
 & \times \sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(i)}) - 2\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(0)}) + \mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(n_x+i)}) \right] \\
 & \times \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(j)}) - 2\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(0)}) + \mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(n_x+j)}) \right]^T \\
 & + \mathbf{Q} + \text{HOT}
 \end{aligned} \tag{8.41}$$

where some of the higher order cross-terms have been lumped into the word HOT.

8.2.3 Multidimensional Finite Difference Observation Prediction Equations

The observation and observation covariance were given in (7.42) and (7.43), respectively. Following the method used in the last section, replacing the differentials by their finite difference approximations, these become

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{2n_x} w_j \tilde{\mathbf{h}}(\boldsymbol{\chi}_{n|n-1}^{(j)}) \tag{8.42}$$

and

$$\begin{aligned}
 \mathbf{P}_{n|n-1}^{\mathbf{zz}} = & \frac{1-w_0}{4n_x} \sum_{j=1}^{n_x} \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(n_x+j)}) \right] \\
 & \times \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(n_x+j)}) \right]^T \\
 & + \frac{1}{2} \left(\frac{1-w_0}{n_x} \right)^2 \sum_{j=1}^{n_x} \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - 2\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(0)}) + \mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(n_x+j)}) \right] \\
 & \times \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - 2\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(0)}) + \mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(n_x+j)}) \right]^T \\
 & + \mathbf{R}
 \end{aligned} \tag{8.43}$$

where the higher order cross-terms have been ignored and

$$\boldsymbol{\chi}_{n|n-1}^{(j)} = \hat{\mathbf{x}}_{n|n-1} + \sqrt{\frac{n_x}{1-w_0}} \mathbf{D}_{n|n-1} \mathbf{r}^{(j)} \tag{8.44}$$

From (7.46), the state-observation cross-covariance prediction can be written as

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \mathbf{D}_{n|n-1} \left[\sum_{i=1}^{n_x} \frac{\partial}{\partial c_i} \tilde{\mathbf{h}}_i(\mathbf{c}) \right]_{\mathbf{c}=\mathbf{0}} \quad (8.45)$$

Substituting the finite difference equivalent of the partial derivatives, (8.45) reduces to

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \mathbf{D}_{n|n-1} \frac{1}{2q} \sum_{i=1}^{n_x} [\tilde{\mathbf{h}}(q\mathbf{e}_i) - \tilde{\mathbf{h}}(-q\mathbf{e}_i)] \quad (8.46)$$

Using (8.36) and (8.38), this becomes

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \frac{1}{2} \sqrt{\frac{1-w_0}{n_x}} \mathbf{D}_{n|n-1} \sum_{j=1}^{n_x} \left[\mathbf{h}\left(\chi_{n|n-1}^{(j)}\right) - \mathbf{h}\left(\chi_{n|n-1}^{(n_x+j)}\right) \right] \quad (8.47)$$

8.2.4 The Multidimensional Finite Difference Kalman Filter Process

The FDKF equations developed above contained terms that are of higher order than linear and will be more accurate than either the LKF or the linearized EKF. If the HOT terms are ignored, the FDKF process can be summarized as shown in Table 8.2.

As we did for the one-dimensional case, these prediction equations can be simplified by setting $w_0 = 0$. This removes the point at the origin and results in prediction equations with no free parameters.

8.3 AN ALTERNATE DERIVATION OF THE MULTIDIMENSIONAL FINITE DIFFERENCE COVARIANCE PREDICTION EQUATIONS

Applying the finite difference equation (2.73), the alternate EKF state error covariance equation (7.69) becomes

$$\begin{aligned} \tilde{\mathbf{P}}_{n|n-1}^{\mathbf{xx}} &= \tilde{\mathbf{g}}(\mathbf{0}) + \frac{1}{2q^2} \sum_{j=1}^{n_x} [\tilde{\mathbf{g}}(q\mathbf{e}_j) - 2\tilde{\mathbf{g}}(\mathbf{0}) + \tilde{\mathbf{g}}(-q\mathbf{e}_j)] + \mathbf{Q} \\ &= \left[\frac{q^2 - n_x}{q^2} \right] \tilde{\mathbf{g}}(\mathbf{0}) + \frac{1}{2q^2} \sum_{j=1}^{n_x} [\tilde{\mathbf{g}}(q\mathbf{e}_j) + \tilde{\mathbf{g}}(-q\mathbf{e}_j)] + \mathbf{Q} \end{aligned} \quad (8.48)$$

From (7.67) we have $\tilde{\mathbf{g}}(\mathbf{c}) \triangleq [\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1}] [\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1}]^\top$. Using the same method as above, (8.48) transforms into

$$\tilde{\mathbf{P}}_{n|n-1}^{\mathbf{xx}} = \mathbf{Q} + \sum_{i=0}^{2n_x} w_j \left[\mathbf{f}\left(\chi_{n-1|n-1}^{(j)}\right) - \hat{\mathbf{x}}_{n|n-1} \right] \left[\mathbf{f}\left(\chi_{n-1|n-1}^{(j)}\right) - \hat{\mathbf{x}}_{n|n-1} \right]^\top \quad (8.49)$$

TABLE 8.2 Multidimensional Finite Difference Kalman Filter Process

Step 1.	Filter initialization:	Set $0 \leq w_0 < 1$ Initialize $\hat{\mathbf{x}}_0$ and \mathbf{P}_0^{xx}
Step 2.	State vector prediction:	$\boldsymbol{\chi}_{n-1 n-1}^{(j)} = \hat{\mathbf{x}}_{n-1 n-1}$ $+ \sqrt{\frac{n_x}{1-w_0}} \mathbf{D}_{n-1 n-1} \mathbf{r}^{(j)}, \quad j = 0, \dots, 2n_x$ $\hat{\mathbf{x}}_{n n-1} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\boldsymbol{\chi}_{n-1 n-1}^{(j)})$ $\mathbf{P}_{n n-1}^{\text{xx}} = \frac{1-w_0}{4n_x} \times \sum_{i=1}^{n_x} \sum_{j=1}^{n_x} \left[\mathbf{f}(\boldsymbol{\chi}_{n-1 n-1}^{(i)}) - \mathbf{f}(\boldsymbol{\chi}_{n-1 n-1}^{(n_x+i)}) \right] \times \left[\mathbf{f}(\boldsymbol{\chi}_{n-1 n-1}^{(j)}) - \mathbf{f}(\boldsymbol{\chi}_{n-1 n-1}^{(n_x+j)}) \right]^T + \mathbf{Q}$
Step 3.	Observation-related prediction:	$\boldsymbol{\chi}_{n n-1}^{(j)} = \hat{\mathbf{x}}_{n n-1}$ $+ \sqrt{\frac{n_x}{1-w_0}} \mathbf{D}_{n n-1} \mathbf{r}^{(j)}, \quad j = 0, \dots, 2n_x$ $\hat{\mathbf{z}}_{n n-1} = \sum_{j=0}^{2n_x} w_j \tilde{\mathbf{h}}(\boldsymbol{\chi}_{n n-1}^{(j)})$ $\mathbf{P}_{n n-1}^{\text{zz}} = \frac{1-w_0}{4n_x} \sum_{j=1}^{n_x} \left[\mathbf{h}(\boldsymbol{\chi}_{n n-1}^{(j)}) - \mathbf{h}(\boldsymbol{\chi}_{n n-1}^{(n_x+j)}) \right] \times \left[\mathbf{h}(\boldsymbol{\chi}_{n n-1}^{(j)}) - \mathbf{h}(\boldsymbol{\chi}_{n n-1}^{(n_x+j)}) \right]^T + \mathbf{R}$ $\mathbf{P}_{n n-1}^{\text{xz}} = \frac{1}{2} \sqrt{\frac{1-w_0}{n_x}} \mathbf{D}_{n n-1} \times \sum_{j=1}^{n_x} \left[\mathbf{h}(\boldsymbol{\chi}_{n n-1}^{(j)}) - \mathbf{h}(\boldsymbol{\chi}_{n n-1}^{(n_x+j)}) \right]$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\text{xz}} \left(\mathbf{P}_{n n-1}^{\text{zz}} \right)^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n \left(\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1} \right)$ $\mathbf{P}_{n n}^{\text{xx}} = \mathbf{P}_{n n-1}^{\text{xx}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\text{zz}} \mathbf{K}_n^T$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\text{xx}}$ to a Track File
Step 6.	Return to Step 2.	

As in the one-dimensional case, this latter form for the predictive state covariance matrix is only partially of second order. It will be shown below that (8.37) and (8.49) are identical to the unscented Kalman filter.

Note that (8.43) can be written in the alternate form

$$\bar{\mathbf{P}}_{n|n-1}^{\text{zz}} = \mathbf{R} + \sum_{j=0}^{2n_x} w_j \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right] \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \quad (8.50)$$

In addition, (8.47) can be rewritten as

$$\bar{\mathbf{P}}_{n|n-1}^{\mathbf{xz}} = \frac{1}{2} \sqrt{\frac{1-w_0}{n_x}} \sum_{j=0}^{2n_x} [\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1}] [\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1}]^\top \quad (8.51)$$

Because the finite-difference filter is almost identical to the unscented Kalman filter, discussed in the next chapter, we will only present the results for the DIFAR case study for the unscented Kalman filter.

REFERENCES

1. Schei TS. A finite-difference approach to linearization in nonlinear estimation algorithms. *Proc. Am. Control Conf.* 1995;114–118.
2. Schei TS. A finite-difference method for linearization in nonlinear estimation algorithms. *Automatica* 1997;33(11):2053–2058.
3. Nørgaard M, Poulsen NK, Ravn O. Advances in Derivative-Free State Estimation for Nonlinear Systems. Technical University of Denmark, IMM-REP-1998-15; 2000.
4. Nørgaard M, Poulsen NK, Ravn O. New developments state estimation for nonlinear systems. *Automatica* 2000;36:1627–1638.
5. Ito K, Xiong K. Gaussian filters for nonlinear filtering problems. *IEEE Trans. Automatic Control* 2000;45(5):910–927.
6. Wu Y, Hu D, Wu M, Hu X. A numerical-integration perspective on Gaussian filters. *IEEE Trans. Sig. Proc.* 2006;54(8):2910–2921.
7. Lerner UN. Hybrid Bayesian Networks for Reasoning About Complex Systems, PhD Dissertation. Department of Computer Science, Stanford University; October 2002.
8. McNamee J, Stenger F. Construction of fully symmetric numerical integration formulas. *Numerische Math.* 1967;10:327–344.

9

THE SIGMA POINT CLASS: THE UNSCENTED KALMAN FILTER

9.1 INTRODUCTION TO MONOMIAL CUBATURE INTEGRATION RULES

In the derivations for the EKF in Chapter 7, all nonlinear functions were expanded in a Taylor polynomial about a single point and the moment equations were then used to evaluate the integrals for each monomial term of the expansion. This constitutes an *analytical* method for evaluation of the integrals over the Gaussian-weighted nonlinear functions. This analytical approach was continued for the FDKF, where the Taylor polynomial is replaced by a multidimensional Stirling's interpolation formula. Stirling's interpolation approximation required the evaluation of the nonlinear functions at vector points equally spaced in all dimensions about the original Taylor polynomial expansion point. Although the FDKF has the same form as a sigma point Kalman filter, it was derived in a completely analytical way and can therefore also be thought of as part of the analytical linearization class of Kalman filters.

An alternative approach to evaluation of the Gaussian-weighted integrals is through the use of multidimensional numerical integration methods. In one dimension, these methods are classified as the well-known (to those who know them well) quadrature integration methods. For the multidimensional case, they have been labeled as "multidimensional quadrature" or "cubature" integration methods [1,2]. In this book, we will use both terms interchangeably.

As noted in Chapter 5, the Gaussian prediction equations shown in (5.51) through (5.57) are of the general integration form shown in (5.60) through (5.62). In this

chapter, we will present several cubature methods for the numerical integration of these integrals.

Consider the general multidimensional Gaussian-weighted integral

$$\hat{\mathbf{g}}(\mathbf{x}) = \int \tilde{\mathbf{g}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (9.1)$$

We have seen in Chapter 5 that this integral can be approximated by a discrete sum of the form (5.64)

$$\hat{\mathbf{g}}(\mathbf{x}) \simeq \sum_{j=1}^{n_s} w_j \tilde{\mathbf{g}}(\mathbf{c}^{(j)}) \quad (9.2)$$

The exact form of w_j , $\mathbf{c}^{(j)}$ and n_s depend on the multidimensional cubature integration rule chosen. In general, an n_s point rule that evaluates the integral is expressed as

$$R_{n_s} = \left\{ \left(w_j, \mathbf{c}^{(j)} \right), j = 1, 2, \dots, 1, \dots, n_s \right\} \quad (9.3)$$

where n_s is the number of integration points. n_s is usually related in some way to n_x . In this chapter and the two that follow we will consider only monomial rules, which will be defined below.

If we consider the vector $\mathbf{c} = [c_1, c_2, \dots, c_k]^T$, then a multidimensional monomial of degree d takes the form (note discussion at the end of Chapter 5)

$$a(j_1, j_2, \dots, j_k) c^{j_1} c^{j_2} \cdots c^{j_k} \quad (9.4)$$

where $j_1 + j_2 + \cdots + j_k \leq d$. For example, following the two-dimensional example shown at the end of Chapter 5, for $\mathbf{c} \in \mathbb{R}^2$, a basis of polynomials spanned by the monomials of degree d less than or equal to 2 is the set $\{1, c_1, c_2, c_1 c_2, c_1^2, c_2^2\}$. For $d = 2$, the condition $j_1 + j_2 \leq d$ must be met for *each* monomial term in the polynomial. In general, for a vector $\mathbf{c} \in \mathbb{R}^k$ there are $\binom{k+d}{d}$ distinct monomials of precision d or less. The binomial coefficient $\binom{n}{m}$ is the number of ways of choosing m objects from a collection of n distinct objects without regard to order. It is defined as

$$\binom{n}{m} \triangleq \frac{n!}{m!(n-m)!} \quad (9.5)$$

■ EXAMPLE 9.1

For $k = 2$ and $d = 2$, we define

$$\dim(k, d) \triangleq \binom{k+d}{d} = \binom{4}{2} = \frac{4!}{2!(4-2)!} = 6 \quad (9.6)$$

An n_s -point monomial rule of degree d is a rule (9.3) that evaluates $\int_{\mathbb{R}^k} \tilde{\mathbf{g}}(\mathbf{c}) w(\mathbf{c}) d\mathbf{c}$ exactly via $\sum_{j=1}^{n_s} w_j \tilde{\mathbf{g}}(\mathbf{c}^{(j)})$ whenever $\tilde{\mathbf{g}}(\mathbf{c})$ is a polynomial of degree less than or equal to d . Label an arbitrary basis set of monomials as $\{b_1, b_2, \dots, b_{\dim(k,d)}\}$. If a rule $R_{n_s} = \{(w_j, \mathbf{c}^{(j)}, j = 1, \dots, n_s\}$ integrates every polynomial of degree d or less, then it must satisfy the monomial system of density-weighted moment equations [3]

$$\sum_{j=1}^{n_s} w_j b_i(\mathbf{c}^{(j)}) = \int_{\mathbb{R}^k} b_i(\mathbf{c}) w(\mathbf{c}) d\mathbf{c}, \quad i = 1, 2, \dots, 1, \dots, \dim(k, d) \quad (9.7)$$

For a Gaussian weight density, this becomes

$$\sum_{j=1}^{n_s} w_j b_i(\mathbf{c}^{(j)}) = \int_{\mathbb{R}^k} b_i(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c}, \quad i = 1, 2, \dots, 1, \dots, \dim(k, d) \quad (9.8)$$

Now, assume that we want a monomial rule that satisfies the Gaussian density-weighted moment equations just up to the second moment. For this case, $\{b_1, b_2, \dots, b_{\dim(k,d)}\} \rightarrow \{1, \mathbf{c}, \mathbf{c}\mathbf{c}^\top\}$ since $\dim(k, d) = \dim(k, 2)$, and the system of moment equations become

$$\sum_{j=1}^{n_s} w_j = \int_{\mathbb{R}^k} \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = 1 \quad (9.9)$$

$$\sum_{j=1}^{n_s} w_j \mathbf{c}^{(j)} = \int_{\mathbb{R}^k} \mathbf{c} \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \mathbf{0} \quad (9.10)$$

$$\sum_{j=1}^{n_s} w_j \mathbf{c}^{(j)} \mathbf{c}^{(j)\top} = \int_{\mathbb{R}^k} \mathbf{c} \mathbf{c}^\top \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \mathbf{I} \quad (9.11)$$

The final step in the formation of the integration rule is to specify a set of vector points $\{\mathbf{c}^{(j)}, j = 1, 2, \dots, 1, \dots, n_s\}$ that lie on a k -dimensional hypersphere and use them to solve for the weights w_j using (9.9)–(9.11).

9.2 THE UNSCENTED KALMAN FILTER

9.2.1 Background

The unscented Kalman filter (UKF) has its origins in a pair of papers presented at the 1995 American Control Conference in Seattle, Washington [4,5]. In both papers, a new method for implementing a linearized approximation to nonlinear state estimation was presented that, unlike the EKF, did not require the explicit calculation of Jacobians. Their method of “approximating a Gaussian distribution efficiently by a discrete distribution [of vector points] allows a nonlinear transformation to be applied

to each of the points independently.” Their method was ad hoc in that they specified a set of vector points that are symmetric about the mean of a multidimensional Gaussian distribution. The points were selected so that when the set of points are used as the input to a nonlinear transformation, a weighted sum of the resulting output points produced a better estimate of the transformed mean than that produced by an EKF.

In Refs [6–8], Julier presents his method in much more detail where, in Ref. [7], he called his method a “distribution approximation filter” and demonstrated its superiority to the EKF for highly nonlinear functional transformations. In Ref. [8] Julier claimed “that it is easier to approximate a probability distribution than it is to approximate an arbitrary nonlinear function or transformation.” However, in Ref. [9], Lefebvre, et al. “derives the exact same estimator by linearizing the process and measurement functions by a statistical linear regression through some regressions points.” Their method has some similarities with the method we develop below, which is completely analytical and involves the use of monomial integration rules with polynomial approximations of the nonlinear functions. The method presented below does not involve in any way the approximation of a density function, but rather the evaluation of a set of Gaussian moment equations for the selected integration points. The sigma points developed in an ad hoc way by Julier and Uhlmann are shown to be a natural result of the affine transformation of the state vector leading to estimation solutions that utilize simple moment integrals of a “normalized” Gaussian density.

Subsequent extension of the unscented Kalman filter quickly followed. In Ref. [10], Wan and van der Merwe extend the UKF to include parameter estimation in addition to state estimation. A square root version of the UKF for state and parameter estimation was published in Ref. [11]. Additional test points were added to the UKF in Ref. [12] allowing the UKF to be applied to discontinuous nonlinear functions. Julier generalized his UKF algorithms in Ref. [13] to allow the sigma points to be scaled to an arbitrary number of dimensions. The UKF was extended to higher order by Tenne and Singh in Ref. [14]. Finally, for those cases where the state (or observation) vector can be parsed into linear and nonlinear components, a Rao-Blackwell UKF was developed by Briers et al. in Ref. [15]. Most recently, an iterated version of the UKF was published by Banani and Masnadi-Shiraz [16].

Excellent summaries of the original UKF along with many of the extensions were published almost simultaneously by van der Merwe [17] and Julier and Uhlmann [18]. These provide an excellent review of the methods in addition to some practical applications and examples.

9.2.2 The UKF Developed

To develop the unscented Kalman filter, consider the case where the vector points $\{\mathbf{c}^{(j)}, j = 1, 2, \dots, 1, \dots, n_s\}$, of the same dimension as the state vector, form symmetric pairs at a radius of $\pm q$ along each Cartesian axes and a single point at the origin. This latter condition ensures that, with the exception of the point at the origin, all vector points will lie on the same n_x -dimensional hypersphere. The symmetry of having two vector points for each dimension also ensures that the total number of integration points will be given by $n_s = 2n_x + 1$. This set of vector points can be described

by $\mathbf{c}^{(j)} = q\mathbf{r}^{(j)}$, with the unit vector set $\mathbf{r} = [1] \in \mathbb{R}^{n_x}$ all lying on the axes of a unit hypersphere [obtained from (2.30)] and a point at the origin. For $n_x = 4$, \mathbf{r} is given by (8.32).

For $\mathbf{c} \in \mathbb{R}^4$, there are $n_s = 2n_x + 1 = 9$ vector evaluation points, since $n_x = 4$. Using these points in (9.9)–(9.11) results in the following system of moment equations

$$\sum_{j=0}^8 w_j = 1 \quad (9.12)$$

$$\sum_{j=0}^8 w_j \mathbf{c}^{(j)} = q \left[w_1 \mathbf{r}^{(1)} + w_2 \mathbf{r}^{(2)} + \cdots + w_8 \mathbf{r}^{(8)} \right] = \mathbf{0} \quad (9.13)$$

$$\sum_{j=0}^8 w_j \mathbf{c}^{(j)} \mathbf{c}^{(j)\top} = q^2 \left[w_1 \mathbf{r}^{(1)} \mathbf{r}^{(1)\top} + \cdots + w_8 \mathbf{r}^{(8)} \mathbf{r}^{(8)\top} \right] = \mathbf{I} \quad (9.14)$$

Equation (9.13) can be rewritten as

$$(w_1 - w_5) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + (w_2 - w_6) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + (w_3 - w_7) \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + (w_4 - w_8) \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This can be satisfied by setting $w_1 = w_5$, $w_2 = w_6$, $w_3 = w_7$, and $w_4 = w_8$.

Equation (9.14) can be written explicitly as

$$(w_1 + w_5) q^2 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + (w_2 + w_6) q^2 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + (w_3 + w_7) q^2 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + (w_4 + w_8) q^2 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9.15)$$

This leads to the set of equations

$$(w_1 + w_5)q^2 = 2w_1q^2 = 1 \rightarrow q = \sqrt{\frac{1}{2w_1}} \quad (9.16)$$

and

$$(w_2 + w_6)q^2 = 2w_2q^2 = 1 \rightarrow q = \sqrt{\frac{1}{2w_2}} \quad (9.17)$$

Thus, $w_1 = w_2$. In a similar manner, it follows that

$$w_j = w_1, \quad j = 2, 3, \dots, 8 \quad (9.18)$$

Now, from (9.12) it follows that

$$w_0 + 8w_1 = 1 \quad (9.19)$$

or

$$w_1 = \frac{1 - w_0}{8} \quad (9.20)$$

Using (9.20) in (9.16) results in

$$q = \sqrt{\frac{4}{1 - w_0}} \quad (9.21)$$

Now

$$\mathbf{c}^{(j)} = qr^{(j)} = \sqrt{\frac{4}{1 - w_0}} \mathbf{r}^{(j)}, \quad j = 0, 1, \dots, 8 \quad (9.22)$$

with

$$w_j = \begin{cases} w_0, & j = 0 \\ \frac{1-w_0}{8}, & j = 1, 2, \dots, 8 \end{cases} \quad (9.23)$$

Here, w_0 is a free parameter, but in order to satisfy (9.12), and to ensure that all weights are positive, we must impose the restriction $0 \leq w_0 \leq 1$.

For the general case where $\mathbf{c} \in \mathbb{R}^{n_x}$, by induction it follows immediately that

$$\mathbf{c}^{(j)} = qr^{(j)} = \sqrt{\frac{n_x}{1 - w_0}} \mathbf{r}^{(j)}, \quad j = 0, 1, \dots, 2n_x \quad (9.24)$$

$$w_j = \begin{cases} w_0, & j = 0 \\ \frac{1-w_0}{2n_x}, & j = 1, \dots, 2n_x \end{cases} \quad (9.25)$$

9.2.3 The UKF State Vector Prediction Equation

From (5.51), (9.1), and (9.2), we can now write the state vector prediction equation as

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \int \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &= \sum_{j=0}^{2n_x} w_j \tilde{\mathbf{f}}(\mathbf{c}^{(j)})\end{aligned}\quad (9.26)$$

From (5.53), for each vector integration point $\mathbf{c}^{(j)}$ we can write

$$\tilde{\mathbf{f}}(\mathbf{c}^{(j)}) = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}^{(j)}), \quad j = 0, 1, \dots, 2n_x \quad (9.27)$$

Defining the *UKF sigma points* as

$$\chi_{n-1|n-1}^{(j)} \triangleq \hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}^{(j)}, \quad j = 0, 1, \dots, 2n_x \quad (9.28)$$

the state vector prediction equation (9.26) becomes

$$\hat{\mathbf{x}}_{n|n-1} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\chi_{n-1|n-1}^{(j)}) \quad (9.29)$$

where $\mathbf{c}^{(j)}$ and w_j are given by (9.24) and (9.25), respectively, and $\mathbf{D}_{n-1|n-1} = [\mathbf{P}_{n-1|n-1}^{\mathbf{xx}}]^{1/2}$.

9.2.4 The UKF State Vector Covariance Prediction Equation

If we let

$$\tilde{\mathbf{g}}(\mathbf{c}) \triangleq [\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1}] [\tilde{\mathbf{f}}(\mathbf{c}) - \hat{\mathbf{x}}_{n|n-1}]^\top \quad (9.30)$$

and use it in (9.26) in place of $\tilde{\mathbf{f}}(\mathbf{c})$, then (9.29) becomes the state covariance prediction equation

$$\begin{aligned}\mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int \tilde{\mathbf{g}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} + \mathbf{Q} \\ &= \sum_{j=0}^{2n_x} w_j \tilde{\mathbf{g}}(\mathbf{c}^{(j)}) + \mathbf{Q} \\ &= \sum_{j=0}^{2n_x} w_j \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\quad \times \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right]^\top + \mathbf{Q}\end{aligned}\quad (9.31)$$

9.2.5 The UKF Observation Prediction Equations

In a similar manner, the observation prediction equations are given by

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{2n_x} w_j \mathbf{h}(\chi_{n|n-1}^{(j)}) \quad (9.32)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \sum_{j=0}^{2n_x} w_j \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right] \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \\ &\quad + \mathbf{R} \end{aligned} \quad (9.33)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \sum_{j=0}^{2n_x} w_j \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\quad \times \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \end{aligned} \quad (9.34)$$

with

$$\chi_{n|n-1}^{(j)} \triangleq \hat{\mathbf{x}}_{n|n-1} + \mathbf{D}_{n|n-1} \mathbf{c}^{(j)}, \quad j = 0, 1, \dots, 2n_x$$

Note that the UKF state and observation vector prediction equations (9.29 and (9.32) are identical to the form of the finite difference prediction equations (8.37) and (8.42), respectively, while the UKF covariance prediction equations (9.31), (9.33), and (9.34) are the same as those of the second form of the finite difference equations (8.51)–(8.49), respectively.

9.2.6 The Unscented Kalman Filter Process

The UKF equations developed above contained terms that are of higher order than linear and will be more accurate than either the LKF or the linearized EKF. The UKF process is summarized in Table 9.1.

9.2.7 An Alternate Version of the Unscented Kalman Filter

In the discussion contained in the paragraph prior to (9.12) it was noted that all point *except* the point at the origin lie on a hypersphere at a radius q . Since the point at the origin was added in an arbitrary way, there is no loss of generality if we take $w_0 = 0$, removing the sigma point at the origin from the integration sum. Note that the sigma point at the origin (at $\mathbf{r}^{(0)}$) coincides with $\hat{\mathbf{x}}_{n-1|n-1}$ or $\hat{\mathbf{x}}_{n|n-1}$. This simplifies the UKF removing the arbitrary nature of picking a value for $0 \leq w_0 \leq 1$. Now all sums have the limits $j = 1, \dots, 2n_x$, with $w_j = 1/2n_x, \forall j$, and $\chi^{(j)} = \hat{\mathbf{x}} + \sqrt{n_x} \mathbf{D}\mathbf{r}^{(j)}$.

The identical results were obtained by Arasaratnam and Haykin [19] following the methods first presented in Refs [2,20]. They considered a spherical–radial cubature rule in which the Cartesian Gaussian-weighted integrals are transformed into polar

TABLE 9.1 Multidimensional Unscented Kalman Filter Process

Step 1.	Filter initialization:	Set $0 \leq w_0 < 1$ Initialize $\hat{\mathbf{x}}_0$ and \mathbf{P}_0^{xx}
Step 2.	State vector prediction:	$\begin{aligned} \chi_{n-1 n-1}^{(j)} &= \hat{\mathbf{x}}_{n-1 n-1} \\ &+ \mathbf{D}_{n-1 n-1} \mathbf{c}^{(j)}, \quad j = 0, \dots, 2n_x \end{aligned}$ $\hat{\mathbf{x}}_{n n-1} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\chi_{n-1 n-1}^{(j)})$ $\begin{aligned} \mathbf{P}_{n n-1}^{\text{xx}} &= \sum_{j=0}^{2n_x} w_j [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}] \\ &\times [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}]^\top + \mathbf{Q} \end{aligned}$
Step 3.	Observation-related prediction:	$\begin{aligned} \chi_{n n-1}^{(j)} &= \hat{\mathbf{x}}_{n n-1} \\ &+ \mathbf{D}_{n n-1} \mathbf{c}^{(j)}, \quad j = 0, \dots, 2n_x \end{aligned}$ $\hat{\mathbf{z}}_{n n-1} = \sum_{j=0}^{2n_x} w_j \tilde{\mathbf{h}}(\chi_{n n-1}^{(j)})$ $\begin{aligned} \mathbf{P}_{n n-1}^{\text{zz}} &= \sum_{j=0}^{2n_x} w_j [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}] \\ &\times [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]^\top + \mathbf{R} \end{aligned}$ $\begin{aligned} \mathbf{P}_{n n-1}^{\text{xz}} &= \sum_{j=0}^{2n_x} w_j [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}] \\ &\times [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]^\top \end{aligned}$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\text{xz}} (\mathbf{P}_{n n-1}^{\text{zz}})^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1})$ $\mathbf{P}_{n n}^{\text{xx}} = \mathbf{P}_{n n-1}^{\text{xx}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\text{zz}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\text{xx}}$ to a Track File
Step 6.	Return to Step 2.	

coordinates. In the development of the integration rule used for the UKF, (9.12)–(9.14) required the solution for $2n_x + 1$ simultaneous equations. In the spherical–radial method, the number of simultaneous equations is reduced to two by taking advantage of the fully symmetric nature of the Gaussian-weighted integrals. But in the end, their method results in a nonlinear Kalman filter that is identical to the UKF developed above with $w_0 = 0$.

9.3 APPLICATION OF THE UKF TO THE DIFAR SHIP TRACKING CASE STUDY

Since the dynamic equation is linear for the DIFAR tracking problem, the UKF process used for this problem is shown in Table 9.2. Note that this process is generic for all sigma point Kalman filters, which will differ only in the values assigned to $\{(w_j, \mathbf{c}^{(j)}), j = 0, 1, \dots, n_s\}$. For the DIFAR case, in Table 9.2 we take $\mathbf{r} = [1] \in \mathbb{R}^4$ for $n_x = 4$. Thus, there are nine sigma points for this UKF since we included the sigma point at the origin. For the sigma points, the values used for $\{(w_j, \mathbf{c}^{(j)})\}$, $j = 0, 1, \dots, n_s$ are given in (9.22) and (9.23), where we let $w_0 = 0.25$.

TABLE 9.2 Multidimensional Sigma Point Kalman Filter Process Applied to DIFAR Tracking

Step 1.	Filter initialization:	Set $0 \leq w_0 < 1$ Initialize $\hat{\mathbf{x}}_0$ and $\mathbf{P}_{0 0}^{\mathbf{xx}}$
Step 2.	State vector prediction:	$\hat{\mathbf{x}}_{n n-1} = \mathbf{F}\hat{\mathbf{x}}_{n-1 n-1}$ $\mathbf{P}_{n n-1}^{\mathbf{xx}} = \mathbf{F}\mathbf{P}_{n-1 n-1}^{\mathbf{xx}}\mathbf{F}^\top + \mathbf{Q}$ $\mathbf{D}_{n n-1} = [\mathbf{P}_{n n-1}^{\mathbf{xx}}]^{1/2}$
Step 3.	Observation-related prediction:	$\chi_{n n-1}^{(j)} = \hat{\mathbf{x}}_{n n-1}$ $+ \mathbf{D}_{n n-1}\mathbf{c}^{(j)}, j = 0, \dots, 2n_x$ $\hat{\mathbf{z}}_{n n-1} = \sum_{j=0}^{2n_x} w_j \mathbf{h}(\chi_{n n-1}^{(j)})$ $\mathbf{P}_{n n-1}^{\mathbf{zz}} = \sum_{j=0}^{2n_x} w_j \left[\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1} \right] \left[\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1} \right]^\top + \mathbf{R}$ $\mathbf{P}_{n n-1}^{\mathbf{xz}} = \sum_{j=0}^{2n_x} w_j \left[\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1} \right] \times \left[\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1} \right]^\top$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\mathbf{xz}} (\mathbf{P}_{n n-1}^{\mathbf{zz}})^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1})$ $\mathbf{P}_{n n}^{\mathbf{xx}} = \mathbf{P}_{n n-1}^{\mathbf{xx}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\mathbf{zz}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\mathbf{xx}}$ to a Track File
Step 6.	Return to Step 2.	

This UKF filter used the same initialization and dynamic noise q values used for the EKF, given in Sections 7.4.3 and 7.4.4, respectively.

A comparison of 100 Monte Carlo output tracks for each of six different signal SNRs from a UKF tracker for the DIFAR case study with the plot for the EKF shown in Figure 7.3 reveals that the outputs at all signal dB levels is almost identical for the two tracking algorithms as are the number of divergent tracks so such plots will not be presented here.

REFERENCES

1. Lu J, Darmofal DL. Higher-dimensional integration with Gaussian weights for applications in probabilistic design. *SIAM J. Sci. Comput.* 2004; 26(2):613–624.
2. Dunavant D A. Efficient symmetrical cubature rules for complete polynomials of higher degree over the unit cube. *Int. J. Numer. Methods Eng.* 1986;23:397–407.
3. Haber S. Numerical evaluation of multiple integrals. *SIAM Rev.* 1970;12(4):481–526.
4. Quine B, Uhlmann J, Durrant-Whyte H. Implicit Jacobians for linearized state estimation in nonlinear systems. *Proc. Am. Control Conf.* 1995:1645–1646.
5. Julier SJ, Uhlmann JK, Durrant-Whyte HF. A new approach for filtering nonlinear systems. *Proc. Am. Control Conf.* 1995:1628–1632.
6. Julier S, Uhlmann, JK. A General Method for Approximating Nonlinear Transformations of Probability Distributions. Robotics Research Group, Department of Engineering Science, University of Oxford, UK; 1996.
7. Julier SJ. Process Models for the Navigation of High Speed Land Vehicles, Dissertation. Oxford, UK: Wadham College; 1997.
8. Julier S, Uhlmann J, Durrant-Whyte HF. A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Trans. Automatic Control* 2000;45(3):477–482.
9. Lefebvre T, Bruyninckx H, De Schutter J. Comments on “A new method for the nonlinear transformation of means and covariances in filters and estimators.” *IEEE Trans. Automatic Control* 2002;47(8):1406–1408.
10. Wan EA, van der Merwe R. The Unscented Kalman Filter for Nonlinear Estimation. *Adaptive Systems for Signal Processing, Communications, and Control Symposium, AS-SPCC*, 2000, pp.153–158.
11. van der Merwe R, Wan EA. The Square-Root Unscented Kalman Filter for State and Parameter Estimation. ICASSP 01, Vol. 6, 2001, pp.3461–3464.
12. Van Zandt J. A more robust unscented transform. *Proc. SPIE* 2001;4473:371–379.
13. Julier SJ. The scaled unscented transform. *Proc. Am. Control Conf.* 2002;4555–4559.
14. Tenne D, Singh T. The higher order unscented filter. *Proc. Am. Control Conf.* 2003;3:2441–2446.
15. Briers M, Maskell SR, Wright R. A Rao-Blackwellised unscented Kalman filter. *Inform. Fusion* 2003;1:55–61.
16. Banani SA, Masnadi-Shirazi MA. A new version of unscented Kalman filter. *Proc. World Acad. Sci. Eng. Technol.* 2007;20:192–197.

17. van der Merwe R. Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models, Dissertation. Portland OR: Oregon Health & Science University; 2004.
18. Julier SJ, Uhlmann JK. Unscented filtering and nonlinear estimation. *Proc. IEEE* 2004;92(3):401–422.
19. Arasaratnam I, Haykin S. Cubature Kalman filters. *IEEE Trans. Automatic Control* 2009;54(6):1254–1269.
20. Monahan J, Genx A. Spherical-radial integration rules for Bayesian computation. *J. Am. Stat. Assoc.* 1997;92(438):664–674.

10

THE SIGMA POINT CLASS: THE SPHERICAL SIMPLEX KALMAN FILTER

For the UKF, with the exception of the integration point at the origin, the vector integration points used for the moment integrations (9.9)–(9.11) are symmetrical about $\mathbf{0}$, equidistant from the origin, and fall on the axes of an n_x -dimensional Cartesian coordinate system. For the spherical simplex Kalman filter, we replace the requirement for symmetry about the origin with a requirement that all vector points be equidistant from the origin and from each other, thus lying at the vertices of an n_x -dimensional simplex (see Figure 2.4). This integration rule was first proposed by Julier [1].

Let the integration (sigma) points be redefined as $\mathbf{c}^{(j)} \rightarrow \mathbf{c}_j$. All of these simplex vector points \mathbf{c}_j must satisfy the moment equations (9.9)–(9.11), repeated here for clarity

$$\sum_{j=0}^{n_x} w_j = 1 \quad (10.1)$$

$$\sum_{j=0}^{n_x} w_j \mathbf{c}_j = \mathbf{0} \quad (10.2)$$

$$\sum_{j=0}^{n_x} w_j \mathbf{c}_j \mathbf{c}_j^T = \mathbf{I} \quad (10.3)$$

10.1 ONE-DIMENSIONAL SPHERICAL SIMPLEX SIGMA POINTS

Consider the one-dimensional case ($n_x = 1$) with three arbitrary points along a line at points $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2) = (0, q_1, -q_2)$, as shown in Figure 2.4. Inserting these points into the moment equations yields the set of equations

$$w_0 + w_1 + w_2 = 1 \quad (10.4)$$

$$w_1 q_1 - w_2 q_2 = 0 \quad (10.5)$$

$$w_1 q_1^2 + w_2 q_2^2 = 1 \quad (10.6)$$

Noting that the vector points must be equidistant from the origin, making $q_2 = q_1$, from (10.5) it follows immediately that

$$w_2 = w_1 \quad (10.7)$$

From (10.4) we obtain

$$w_1 = w_2 = \frac{1 - w_0}{2} \quad (10.8)$$

Now from (10.6)

$$q_1 = \frac{1}{\sqrt{2w_1}} \quad (10.9)$$

Thus, for the 1D case, the sigma points are defined by

$$c_j^{1D} = \begin{cases} 0, & j = 0 \\ 1/\sqrt{2w_1}, & j = 1 \\ -1/\sqrt{2w_1}, & j = 2 \end{cases} \quad (10.10)$$

with the weights

$$0 \leq w_0 \leq 1 \quad (10.11)$$

$$w_1 = \frac{1 - w_0}{2} \quad (10.12)$$

$$w_2 = \frac{1 - w_0}{2} \quad (10.13)$$

Remembering that for one dimension, $x = \hat{x} + \sigma c$, then the 1D sigma points are given by

$$\chi_0 = \hat{x} \quad (10.14)$$

$$\chi_1 = \hat{x} + \sigma c_1^{1D} \quad (10.15)$$

$$\chi_2 = \hat{x} + \sigma c_2^{1D} \quad (10.16)$$

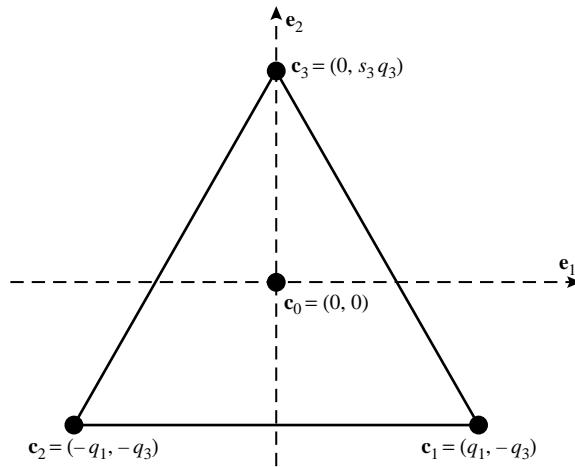


FIGURE 10.1 Depiction of a two-dimensional simplex with four vector integration points.

10.2 TWO-DIMENSIONAL SPHERICAL SIMPLEX SIGMA POINTS

For the two-dimensional case, consider the vector points at the vertices of the simplex shown in Figure 10.1. In this equilateral triangle, all sides and angles are equal and all integration points are equidistant from the point at the origin. For this 2D case, the *vector* integration points are the set $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3) = \{(0, 0)^T, (q_1, -q_3)^T, (-q_1, -q_3)^T, (0, s_3 q_3)^T\}$. Here, s_3 is a scale factor that will be determined below.

Once again, the moment equations (10.1)–(10.3) must be satisfied. Thus, from (10.1) it follows that

$$w_0 + w_1 + w_2 + w_3 = 1 \quad (10.17)$$

and (10.2) results in

$$w_1 q_1 - w_2 q_1 = 0 \quad (10.18)$$

$$-w_1 q_3 - w_2 q_3 + w_3 s_3 q_3 = 0 \quad (10.19)$$

Finally, (10.3) leads to

$$w_1 \begin{bmatrix} q_1^2 & 0 \\ 0 & q_3^2 \end{bmatrix} + w_2 \begin{bmatrix} q_1^2 & 0 \\ 0 & q_3^2 \end{bmatrix} + w_3 \begin{bmatrix} 0 & 0 \\ 0 & s_3^2 q_3^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (10.20)$$

which reduces to the set of equations

$$w_1 q_1^2 + w_2 q_1^2 = 1 \quad (10.21)$$

$$w_1 q_3^2 + w_2 q_3^2 + w_3 s_3^2 q_3^2 = 1 \quad (10.22)$$

From (10.18) it follows that

$$w_2 = w_1 \quad (10.23)$$

and (10.22) becomes

$$2w_1q_3^2 + w_3s_3^2q_3^2 = 1 \quad (10.24)$$

Since each point is equidistant from the origin, it can be assumed that the weight on each point is the same, and therefore $w_3 = w_1$. From (10.17) it follows that

$$w_1 = w_2 = w_3 = \frac{1 - w_0}{3} \quad (10.25)$$

From (10.19) we find that $s_3 = 2$ and (10.24) now leads to

$$q_3 = \frac{1}{\sqrt{6w_1}} = \frac{1}{\sqrt{2(2+1)w_1}} \quad (10.26)$$

To summarize, for the 2D case, the integration points are given by

$$\mathbf{c}_0^{2D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_0^{1D} \\ 0 \end{bmatrix} \quad (10.27)$$

$$\mathbf{c}_1^{2D} = \begin{bmatrix} c_1^{1D} \\ -\frac{1}{\sqrt{2(2+1)w_1}} \end{bmatrix} \quad (10.28)$$

$$\mathbf{c}_2^{2D} = \begin{bmatrix} c_2^{1D} \\ -\frac{1}{\sqrt{2(2+1)w_1}} \end{bmatrix} \quad (10.29)$$

$$\mathbf{c}_3^{2D} = \begin{bmatrix} c_3^{1D} \\ \frac{2}{\sqrt{2(2+1)w_1}} \end{bmatrix} \quad (10.30)$$

with the weights given by

$$0 \leq w_0 \leq 1 \quad (10.31)$$

$$w_j = \frac{1 - w_0}{3}, \quad j = 1, 2, 3 \quad (10.32)$$

Note that (10.28) and (10.29) can be combined into one equation

$$\mathbf{c}_j^{2D} = \begin{bmatrix} c_j^{1D} \\ -\frac{1}{\sqrt{2(2+1)w_1}} \end{bmatrix}, \quad j = 1, 2 \quad (10.33)$$

10.3 HIGHER DIMENSIONAL SPHERICAL SIMPLEX SIGMA POINTS

For the general n_x -dimensional case, we obtain, by inspection

$$\mathbf{c}_0^{n_x D} = \mathbf{0}_{n_x} = \begin{bmatrix} \mathbf{c}_0^{(n_x-1)D} \\ 0 \end{bmatrix}, \quad j = 0 \quad (10.34)$$

$$\mathbf{c}_j^{n_x D} = \begin{bmatrix} \mathbf{c}_j^{(n_x-1)D} \\ -\frac{1}{\sqrt{n_x(n_x+1)w_1}} \end{bmatrix}, \quad j = 1, \dots, n_x \quad (10.35)$$

$$\mathbf{c}_{n_x+1}^{n_x D} = \begin{bmatrix} \mathbf{c}_0^{(n_x-1)D} \\ \frac{n_x}{\sqrt{n_x(n_x+1)w_1}} \end{bmatrix} \quad (10.36)$$

with the weights given by

$$0 \leq w_0 \leq 1 \\ w_j = \frac{1 - w_0}{n_x + 1}, \quad j = 1, \dots, n_x + 1 \quad (10.37)$$

Remembering that $\mathbf{x} = \hat{\mathbf{x}} + \mathbf{D}\mathbf{c}$, the integration points become the spherical simplex sigma points

$$\boldsymbol{\chi}^{(j)} = \hat{\mathbf{x}} + \mathbf{D}\mathbf{c}_j^{n_x D}, \quad j = 0, \dots, n_x + 1 \quad (10.38)$$

10.4 THE SPHERICAL SIMPLEX KALMAN FILTER

Returning to (5.51), we can now write

$$\begin{aligned} \hat{\mathbf{x}}_{n|n-1} &= \int \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &= \sum_{j=0}^{n_x+1} w_j \mathbf{f} \left(\boldsymbol{\chi}_{n-1|n-1}^{(j)} \right) \end{aligned} \quad (10.39)$$

where from (10.38) we can identify the sigma points as

$$\boldsymbol{\chi}_{n-1|n-1}^{(j)} = \hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}_j^{n_x D}, \quad j = 0, \dots, n_x + 1 \quad (10.40)$$

with $\mathbf{c}_j^{n_x \text{D}}$ defined by (10.34)–(10.36). In a manner similar to that of the UKF, we obtain the remaining predictive equations, with the state covariance given by

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \int \tilde{\mathbf{g}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} = \sum_{j=0}^{n_x+1} w_j \tilde{\mathbf{g}}(\mathbf{c}^{(j)}) \\ &= \sum_{j=0}^{n_x+1} w_j \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\quad \times \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right]^T + \mathbf{Q} \end{aligned} \quad (10.41)$$

and the observation prediction equations

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{n_x+1} w_j \mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) \quad (10.42)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \sum_{j=0}^{n_x+1} w_j \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right] \\ &\quad \times \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T + \mathbf{R} \end{aligned} \quad (10.43)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \sum_{j=0}^{n_x+1} w_j \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\quad \times \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \end{aligned} \quad (10.44)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \sum_{j=0}^{n_x+1} w_j \left[\mathbf{f}(\boldsymbol{\chi}_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\quad \times \left[\mathbf{h}(\boldsymbol{\chi}_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \end{aligned} \quad (10.45)$$

where

$$\boldsymbol{\chi}_{n|n-1}^{(j)} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{D}_{n|n-1} \mathbf{c}_j^{n_x \text{D}}, \quad j = 0, \dots, n_x + 1 \quad (10.46)$$

10.5 THE SPHERICAL SIMPLEX KALMAN FILTER PROCESS

The SSKF process is summarized in Table 10.1. Note that the only difference between Tables 10.1, 8.2 and 9.1 is the upper summation limit. Because of this similarity in the process for all sigma point Kalman filters, in a later chapter we will combine all sigma point Kalman filter processes into a single, more general, process.

TABLE 10.1 Multidimensional Spherical Simplex Kalman Filter Process

Step 1.	Filter initialization:	Set $0 \leq w_0 < 1$ Initialize $\hat{\mathbf{x}}_0$ and \mathbf{P}_0^{xx}
Step 2.	State vector prediction:	$\chi_{n-1 n-1}^{(j)} = \mathbf{D}_{n-1 n-1} \mathbf{c}_j^{n_x \text{D}}, j = 0, \dots, n_x + 1$
		$\hat{\mathbf{x}}_{n n-1} = \sum_{j=0}^{n_x+1} w_j \mathbf{f}(\chi_{n-1 n-1}^{(j)})$
		$\mathbf{P}_{n n-1}^{\text{xx}} = \sum_{j=0}^{n_x+1} w_j [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}] \times [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}]^\top + \mathbf{Q}$
Step 3.	Observation-related prediction:	$\chi_{n n-1}^{(j)} = \hat{\mathbf{x}}_{n n-1} + \mathbf{D}_{n n-1} \mathbf{c}_j^{n_x \text{D}}, j = 0, \dots, 2n_x$
		$\hat{\mathbf{z}}_{n n-1} = \sum_{j=0}^{n_x+1} w_j \mathbf{h}(\chi_{n n-1}^{(j)})$
		$\mathbf{P}_{n n-1}^{\text{zz}} = \sum_{j=0}^{n_x+1} w_j [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}] \times [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]^\top + \mathbf{R}$
		$\mathbf{P}_{n n-1}^{\text{xz}} = \sum_{j=0}^{n_x+1} w_j [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}] \times [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]^\top$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\text{xz}} (\mathbf{P}_{n n-1}^{\text{zz}})^{-1}$
		$\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1})$
		$\mathbf{P}_{n n}^{\text{xx}} = \mathbf{P}_{n n-1}^{\text{xx}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\text{zz}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\text{xx}}$ to a Track File
Step 6.	Return to Step 2.	

10.6 APPLICATION OF THE SSKF TO THE DIFAR SHIP TRACKING CASE STUDY

Since the dynamic equation is linear for the DIFAR tracking problem, the SSKF process used for this problem is identical to that shown in Table 9.2, except for the upper limit of the summation.

For this DIFAR case, $n_x = 4$ and we let $\mathbf{c}^{(j)} \rightarrow \mathbf{c}_j^{4\text{D}}$. Since we include the sigma point at the origin, for the SSKF there are six sigma points. For the sigma points, the

values used for $\{(w_j, \mathbf{c}_j^{4D}), j = 0, 1, \dots, 5\}$ are given in (10.34)–(10.37), where we let $w_0 = 0.25$. We used the same initialization and dynamic noise q values used for the EKF, given in Sections 7.4.3 and 7.4.4, respectively. The 100 Monte Carlo track plots obtained from the SSKF for each of the signal SNR values used previously are similar to those shown in Figure 7.3. With the exception of the number of divergent tracks for an SNR of 5 dB (3 for the SSKF vice 2 for the EKF and UKF), there is little discernible difference in the output of the SSKF from those of the EKF and SSKF, as will be demonstrated when we discuss RMS errors later in this book.

REFERENCE

1. Julier SJ. The spherical simplex unscented transformation. *Proc. Am. Control Conf.* 2003:2430–2434.

11

THE SIGMA POINT CLASS: THE GAUSS–HERMITE KALMAN FILTER

Practical methods for one-dimensional Gauss–Hermite integration has its origins in the work of Wilf [1], as noted in the Numerical Recipes book [2], and the short but excellent paper by Ball [3]. Unlike the methods addressed in the preceding chapters, Gauss–Hermite integration is a quadrature method that uses orthogonal polynomials instead of simple polynomials.

Consider the general multidimensional integral (5.60), rewritten here in more explicit form

$$\mathbf{I}(\tilde{\mathbf{f}}) = \frac{1}{(2\pi)^{n_x/2}} \int \tilde{\mathbf{f}}(\mathbf{c}) e^{-\mathbf{c}^\top \mathbf{c}/2} d\mathbf{c} \quad (11.1)$$

Changing variables to remove the factor of 1/2 from the exponent, let

$$\mathbf{q} \triangleq \frac{\mathbf{c}}{\sqrt{2}} = \frac{1}{\sqrt{2}} \mathbf{D}^{-1} (\mathbf{x} - \hat{\mathbf{x}}) \quad (11.2)$$

or

$$\mathbf{x} = \hat{\mathbf{x}} + \sqrt{2} \mathbf{D} \mathbf{q} \quad (11.3)$$

Thus, (11.1) becomes

$$\mathbf{I}(\tilde{\mathbf{f}}) = \frac{\sqrt{2}}{(2\pi)^{n_x/2}} \int \tilde{\mathbf{f}}(\mathbf{q}) e^{-\mathbf{q}^\top \mathbf{q}} d\mathbf{q} \quad (11.4)$$

with

$$\tilde{\mathbf{f}}(\mathbf{q}) \triangleq \mathbf{f} \left(\hat{\mathbf{x}} + \sqrt{2} \mathbf{D} \mathbf{q} \right) \quad (11.5)$$

11.1 ONE-DIMENSIONAL GAUSS–HERMITE QUADRATURE

The objective now is to expand $\tilde{f}(\mathbf{q})$ in terms of orthogonal polynomials in some way. First, consider the one-dimensional case integral of the form

$$I = \int \tilde{f}(q) e^{-q^2} dq \quad (11.6)$$

Expanding $\tilde{f}(q)$ in a general set of orthogonal polynomials, $\tilde{f}(q)$ can be approximated by

$$\tilde{f}(q) \simeq \sum_{l=0}^m p_l \phi_l(q) \quad (11.7)$$

where $\{\phi_l(q), l = 0, 1, \dots, m\}$ represent a set of orthogonal polynomials, with the polynomials $\phi_l(q)$ and $\phi_k(q)$ orthogonal with respect to a weight (or kernel) function $p(q)$ over the interval $[a, b]$ such that

$$\int_a^b p(q) \phi_l(q) \phi_k(q) dq = c_l \delta_{lk} \quad (11.8)$$

Here, c_l is a normalization constant and

$$\delta_{lk} = \begin{cases} 1, & k = l \\ 0, & k \neq l \end{cases} \quad (11.9)$$

For integrals of the form (11.6), the Hermite polynomials $H_l(q)$ can be used, with the weight function given by

$$p(q) = e^{-q^2} \quad (11.10)$$

with the integration interval becoming $[a, b] \rightarrow [-\infty, \infty]$. The orthogonality condition (11.8) becomes

$$\int_{-\infty}^{\infty} e^{-q^2} H_l(q) H_k(q) dq = h_l \delta_{lk} \quad (11.11)$$

where the normalization constant is given by

$$h_l = \sqrt{\pi} 2^l l! \quad (11.12)$$

The Hermite polynomials can be generated using the recursion relationships [3]

$$H_{-1}(q) = 0 \quad (11.13)$$

$$H_0(q) = 1 \quad (11.14)$$

$$H_{l+1}(q) = 2qH_l(q) - 2lH_{l-1}(q), \quad l \geq 0 \quad (11.15)$$

Defining the normalized Hermite polynomial as

$$\tilde{H}_l(q) \triangleq \frac{H_l(q)}{\sqrt{h_l}} \quad (11.16)$$

leads to the orthonormality condition

$$\int_{-\infty}^{\infty} e^{-q^2} \tilde{H}_l(q) \tilde{H}_k(q) dq = \delta_{lk} \quad (11.17)$$

The recursion relationship for the orthonormal Hermite polynomials follows immediately

$$\tilde{H}_{-1}(q) = 0 \quad (11.18)$$

$$\tilde{H}_0(q) = h_0^{-1/2} = \pi^{-1/4} \quad (11.19)$$

$$\tilde{H}_{l+1}(q) = q\sqrt{\frac{2}{l+1}}\tilde{H}_l(q) - \sqrt{\frac{l}{l+1}}\tilde{H}_{l-1}(q), \quad l \geq 0 \quad (11.20)$$

If we define

$$\beta_l \triangleq \sqrt{l/2} \quad (11.21)$$

(11.20) becomes

$$q\tilde{H}_l(q) = \beta_l\tilde{H}_{l-1}(q) + \beta_{l+1}\tilde{H}_{l+1}(q) \quad (11.22)$$

Defining the vectors $\mathbf{h}(q)$ and \mathbf{e}_m and the matrix \mathbf{J}_m as

$$\mathbf{h}(q) \triangleq [\tilde{H}_0(q), \tilde{H}_1(q), \dots, \tilde{H}_{m-1}(q)]^\top \quad (11.23)$$

$$\mathbf{e}_m \triangleq [0, 0, \dots, 0, 1]^\top \quad (11.24)$$

$$\mathbf{J}_m \triangleq \begin{bmatrix} 0 & \beta_1 & \cdots & 0 \\ \beta_1 & 0 & & \vdots \\ \vdots & & \ddots & \beta_{m-1} \\ 0 & \cdots & \beta_{m-1} & 0 \end{bmatrix} \quad (11.25)$$

the orthogonal Hermite recursion relationship (11.22) can be rewritten as the vector-matrix equation

$$q\mathbf{h}(q) = \mathbf{J}_m \mathbf{h}(q) + \beta_m \tilde{H}_m(q) \mathbf{e}_l \quad (11.26)$$

Setting

$$\tilde{H}_m(q) = 0 \quad (11.27)$$

(11.26) becomes the eigenvalue equation

$$[\mathbf{J}_m - q\mathbf{I}] \mathbf{h}(q) = \mathbf{0} \quad (11.28)$$

The m distinct eigenvalues of \mathbf{J}_m , given by $\{q_l, l = 0, 1, \dots, m-1\}$, are the interpolation points for the m th-order expansion of $f(q)$. It can be shown that the error in interpolation of $\tilde{f}(q)$ at the distinct eigenvalue points $\{q_l, l = 0, 1, \dots, m-1\}$ is zero [2]. Because the orthonormality condition (11.17) includes a weight function, the eigenvalues can be written as the set given by the normalized solution to (11.27)

$$q_l = \frac{\tilde{H}_m(q_l)}{p(q_l)}, \quad l = 0, 1, \dots, m-1 \quad (11.29)$$

where $p(q_l)$ is given by (11.10).

Consider a third-order polynomial with $m = 3$. Although higher order polynomials have been developed for the one-dimensional case (see, for example, [4,5]), higher order methods quickly become unruly when applied to multidimensional cases. For $m = 3$, (11.25) becomes

$$\mathbf{J}_3 = \begin{bmatrix} 0 & \beta_1 & 0 \\ \beta_1 & 0 & \beta_2 \\ 0 & \beta_2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (11.30)$$

The eigenvalues of (11.30) consist of the set

$$\{q_0, q_1, q_2\} = \frac{\sqrt{3}}{2} \{-1, 0, 1\} \quad (11.31)$$

Letting $\mathbf{r} \triangleq \{0, 1, -1\}$, (11.31) can be rearranged into the set

$$\{q_0, q_1, q_2\} = \frac{\sqrt{3}}{2} \mathbf{r} \quad (11.32)$$

The function $\tilde{f}(q)$ can now be approximated by a third-order expansion in terms of the normalized Hermite polynomials

$$\tilde{f}(q) \simeq \sum_{l=0}^2 a_l \tilde{H}_l(q) \quad (11.33)$$

Since the expansion of $\tilde{f}(q)$ is exact at the eigenvalue points, writing (11.33) for each point results in the system of moment equations

$$\tilde{f}(q_0) \simeq \sum_{l=0}^2 a_l \tilde{H}_l(q_0) \quad (11.34)$$

$$\tilde{f}(q_1) \simeq \sum_{l=0}^2 a_l \tilde{H}_l(q_1) \quad (11.35)$$

$$\tilde{f}(q_2) \simeq \sum_{l=0}^2 a_l \tilde{H}_l(q_2) \quad (11.36)$$

These three equations in the three unknowns $\{a_0, a_1, a_2\}$ can now be solved for the unknown coefficients. Defining the vectors $\tilde{\mathbf{f}}$ and \mathbf{a} and the matrix $\tilde{\mathbf{H}}$ as

$$\tilde{\mathbf{f}} \triangleq [\tilde{f}(q_0), \tilde{f}(q_1), \tilde{f}(q_2)]^\top \quad (11.37)$$

$$\mathbf{a} \triangleq [a_0, a_1, a_2]^\top \quad (11.38)$$

$$\tilde{\mathbf{H}} \triangleq \begin{bmatrix} \tilde{H}_0(q_0) & \tilde{H}_1(q_0) & \tilde{H}_2(q_0) \\ \tilde{H}_0(q_1) & \tilde{H}_1(q_1) & \tilde{H}_2(q_1) \\ \tilde{H}_0(q_2) & \tilde{H}_1(q_2) & \tilde{H}_2(q_2) \end{bmatrix} \quad (11.39)$$

the system of equations (11.34)–(11.36) can be rewritten as the vector–matrix equation

$$\tilde{\mathbf{f}} = \tilde{\mathbf{H}} \mathbf{a} \rightarrow \mathbf{a} = \tilde{\mathbf{H}}^{-1} \tilde{\mathbf{f}} \quad (11.40)$$

Using (11.12)–(11.16) along with (11.32), we find that

$$\tilde{\mathbf{H}} = \frac{1}{\pi^{1/4}} \begin{bmatrix} 1 & 0 & -1/\sqrt{2} \\ 1 & \sqrt{3} & \sqrt{2} \\ 1 & -\sqrt{3} & \sqrt{2} \end{bmatrix} \quad (11.41)$$

The inverse of $\tilde{\mathbf{H}}$ is given by

$$\tilde{\mathbf{H}}^{-1} = \pi^{1/4} \begin{bmatrix} 2/3 & 1/6 & 1/6 \\ 0 & 1/2\sqrt{3} & -1/2\sqrt{3} \\ -\sqrt{2}/3 & 1/3\sqrt{2} & 1/3\sqrt{2} \end{bmatrix} \quad (11.42)$$

Using (11.42) in (11.40) results in

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \pi^{1/4} \begin{bmatrix} 2/3 & 1/6 & 1/6 \\ 0 & 1/2\sqrt{3} & -1/2\sqrt{3} \\ -\sqrt{2}/3 & 1/3\sqrt{2} & 1/3\sqrt{2} \end{bmatrix} \begin{bmatrix} \tilde{f}(0) \\ \tilde{f}\left(\frac{\sqrt{3}}{2}\right) \\ \tilde{f}\left(-\frac{\sqrt{3}}{2}\right) \end{bmatrix} \quad (11.43)$$

Solving (11.43) for $\{a_0, a_1, a_2\}$ yields

$$a_0 = \pi^{1/4} \left[\frac{2}{3} \tilde{f}(0) + \frac{1}{6} \tilde{f}\left(\sqrt{\frac{3}{2}}\right) + \frac{1}{6} \tilde{f}\left(-\sqrt{\frac{3}{2}}\right) \right] \quad (11.44)$$

$$a_1 = \frac{\pi^{1/4}}{2\sqrt{3}} \left[\tilde{f}\left(\sqrt{\frac{3}{2}}\right) - \tilde{f}\left(-\sqrt{\frac{3}{2}}\right) \right] \quad (11.45)$$

$$a_2 = \frac{\pi^{1/4}}{3\sqrt{2}} \left[\tilde{f}\left(\sqrt{\frac{3}{2}}\right) - 2\tilde{f}(0) + \tilde{f}\left(-\sqrt{\frac{3}{2}}\right) \right] \quad (11.46)$$

Keeping only the first term in the Hermite expansion of $\tilde{f}(q)$, (11.33) becomes

$$\tilde{f}(q) = a_0 \tilde{H}_0(q) \quad (11.47)$$

or

$$\tilde{f}(q) = \frac{2}{3} \tilde{f}(0) + \frac{1}{6} \tilde{f}\left(\sqrt{\frac{3}{2}}\right) + \frac{1}{6} \tilde{f}\left(-\sqrt{\frac{3}{2}}\right) \quad (11.48)$$

To avoid confusion when this is applied to multiple dimensions, define the one-dimensional set of interpolation points and their weights as

$$\{q^{(0)}, q^{(1)}, q^{(2)}\} = \left\{0, \sqrt{\frac{3}{2}}, -\sqrt{\frac{3}{2}}\right\} \quad (11.49)$$

and

$$\{p_0, p_1, p_2\} = \left\{\frac{2}{3}, \frac{1}{6}, \frac{1}{6}\right\} \quad (11.50)$$

Now, the Hermite polynomial expansion (11.48) becomes

$$\tilde{f}(q) \simeq \sum_{i=0}^2 p_i \tilde{f}\left(q^{(i)}\right) \quad (11.51)$$

with $q^{(i)}$ and p_i given by (11.49) and (11.50), respectively.

11.2 ONE-DIMENSIONAL GAUSS–HERMITE KALMAN FILTER

For one dimension, (11.4) becomes

$$I(\tilde{f}) = \frac{1}{\sqrt{\pi}} \int \tilde{f}(q) e^{-q^2} dq \quad (11.52)$$

where

$$q = \frac{c}{\sqrt{2}} = \frac{1}{\sqrt{2}}\sigma^{-1}(x - \hat{x}) \quad (11.53)$$

or

$$x = \hat{x} + \sqrt{2}\sigma q \quad (11.54)$$

Using (11.49) in (11.54) yields the one-dimensional Gauss–Hermite sigma points

$$\chi^{(i)} = \hat{x} + \sqrt{2}\sigma q^{(i)} \quad (11.55)$$

In addition, we can write

$$\tilde{f}\left(q^{(i)}\right) = f\left(\hat{x} + \sqrt{2}\sigma q^{(i)}\right) = f\left(\chi^{(i)}\right) \quad (11.56)$$

and (11.52) now becomes

$$\begin{aligned} I\left(\tilde{f}\right) &= \sum_{i=0}^2 p_i f\left(\chi^{(i)}\right) \frac{1}{\sqrt{\pi}} \int e^{-q^2} dq \\ &= \sum_{i=0}^2 p_i f\left(\chi^{(i)}\right) \frac{1}{\sqrt{2\pi}} \int e^{-c^2/2} dc \\ &= \sum_{i=0}^2 p_i f\left(\chi^{(i)}\right) \int \mathcal{N}(c; 0, 1) dc \end{aligned} \quad (11.57)$$

and finally, because the integral over the density function equals one, (11.57) reduces to

$$I\left(\tilde{f}\right) = \sum_{i=0}^2 p_i f\left(\chi^{(i)}\right) \quad (11.58)$$

Using (11.58), the one-dimensional state prediction equation (5.51) now becomes

$$\hat{x}_{n|n-1} = \sum_{i=0}^2 p_i f\left(\chi_{n-1|n-1}^{(i)}\right) \quad (11.59)$$

where

$$\chi_{n-1|n-1}^{(i)} = \hat{x}_{n-1|n-1} + \sqrt{3}\sigma_{xx,n-1|n-1}\mathbf{r}^{(i)} \quad (11.60)$$

with

$$\mathbf{r} \triangleq [1] \in \mathbb{R}^1 = \begin{cases} 0, & i = 0 \\ 1, & i = 1 \\ -1, & i = 2 \end{cases} \quad (11.61)$$

For the one-dimensional state variance prediction, if we let

$$\tilde{g}(c) \triangleq \left[\tilde{f}(c) - \hat{x}_{n|n-1} \right]^2 \quad (11.62)$$

the one-dimensional state variance prediction (7.2) becomes

$$\sigma_{xx,n|n-1}^2 = \int \tilde{g}(c) \mathcal{N}(c; 0, 1) dc + \sigma_{v,n}^2 \quad (11.63)$$

Noting that the integral in (11.63) is of the same form as (11.6), it follows immediately that

$$\sigma_{xx,n|n-1}^2 = \sum_{i=0}^2 p_i \left[f\left(\chi_{n-1|n-1}^{(i)}\right) - \hat{x}_{n|n-1} \right]^2 + \sigma_{v,n}^2 \quad (11.64)$$

In a similar fashion, (5.51)–(5.52) reduce to the prediction equations

$$\hat{z}_{n|n-1} = \sum_{i=0}^2 p_i h\left(\chi_{n|n-1}^{(i)}\right) \quad (11.65)$$

$$\sigma_{zz,n|n-1}^2 = \sum_{i=0}^2 p_i \left[f\left(\chi_{n|n-1}^{(i)}\right) - \hat{x}_{n|n-1} \right]^2 + \sigma_{w,n}^2 \quad (11.66)$$

and

$$\sigma_{xz,n|n-1}^2 = \sum_{i=0}^2 p_i \left[f\left(\chi_{n|n-1}^{(i)}\right) - \hat{x}_{n|n-1} \right] \left[h\left(\chi_{n|n-1}^{(i)}\right) - \hat{z}_{n|n-1} \right] \quad (11.67)$$

with

$$\chi_{n|n-1}^{(i)} = \hat{x}_{n|n-1} + \sqrt{3} \sigma_{xx,n|n-1} \mathbf{r}^{(i)} \quad (11.68)$$

(11.59) and (11.64)–(11.67) make up the prediction equations for the one-dimensional Gauss–Hermite Kalman filter. The process flow for the one-dimensional case will not be given here but can be treated as a special case of the multidimensional case presented in the next section.

11.3 MULTIDIMENSIONAL GAUSS–HERMITE KALMAN FILTER

Constructing multidimensional orthogonal polynomial integration rules can be a difficult task if one first attempts to construct interpolative multidimensional polynomials.

The simplest method of constructing multidimensional cubature rules is to form product rules from one-dimensional integration rules. Consider first a two-dimensional case where $\mathbf{q} = [q_1, q_2]^\top$ and we expand $\tilde{\mathbf{f}}(\mathbf{q})$ as the double sum

$$\tilde{\mathbf{f}}(\mathbf{q}) = \sum_{i_1=0}^2 \sum_{i_2=0}^2 p_{i_1} p_{i_2} \tilde{\mathbf{f}}(q_1^{(i_1)}, q_2^{(i_2)}) = \sum_{j=0}^{3^2-1} w_j \tilde{\mathbf{f}}(\mathbf{q}^{(j)}) \quad (11.69)$$

where p_{i_1} and p_{i_2} are defined by (11.50) and $q_1^{(i_1)}$ and $q_2^{(i_2)}$ are defined by (11.49). The double sum in (11.69) can be converted into the single sum indexed over $\{j = 0, 1, \dots, 8\}$ by forming the weights

$$w_0 = \left(\frac{2}{3}\right)^2 = p_0^2 \quad (11.70)$$

$$w_1 = w_2 = w_3 = w_4 = \left(\frac{2}{3}\right) \left(\frac{1}{6}\right) = p_0 p_i = p_i p_0, \quad i = 1, 2 \quad (11.71)$$

$$w_5 = w_6 = w_7 = w_8 = \left(\frac{1}{6}\right)^2 = p_i p_l = p_l p_i, \quad i, l = 1, 2 \quad (11.72)$$

Putting this into a more compact form, w_j is defined by

$$w_j \triangleq \begin{cases} \left(\frac{2}{3}\right)^2, & j = 0 \\ \left(\frac{2}{3}\right) \left(\frac{1}{6}\right), & j = 1, \dots, 4 \\ \left(\frac{1}{6}\right)^2, & j = 5, \dots, 8 \end{cases} \quad (11.73)$$

From (11.50) it follows that

$$\mathbf{q}^{(j)} = \sqrt{\frac{3}{2}} \mathbf{r}^{(j)} \quad (11.74)$$

where

$$\mathbf{r}^{(j)} = \begin{cases} [0] \in \mathbb{R}^2 = [0, 0]^T, & j = 0 \\ [1] \in \mathbb{R}^2 = \begin{cases} [1, 0]^T, & j = 1 \\ [0, 1]^T, & j = 2 \\ [-1, 0]^T, & j = 3 \\ [0, -1]^T, & j = 4 \end{cases} \\ [1, 1] \in \mathbb{R}^2 = \begin{cases} [1, 1]^T, & j = 5 \\ [-1, 1]^T, & j = 6 \\ [1, -1]^T, & j = 7 \\ [-1, -1]^T, & j = 8 \end{cases} \end{cases} \quad (11.75)$$

In a similar manner, for a state vector of four dimensions, the expansion of $\tilde{f}(\mathbf{q})$ is given by

$$\begin{aligned} \tilde{\mathbf{f}}(\mathbf{q}) &= \sum_{i_1=0}^2 \sum_{i_2=0}^2 \sum_{i_3=0}^2 \sum_{i_4=0}^2 p_{i_1} p_{i_2} p_{i_3} p_{i_4} \tilde{\mathbf{f}}\left(q_1^{(i_1)}, q_2^{(i_2)}, q_1^{(i_3)}, q_2^{(i_4)}\right) \\ &= \sum_{j=0}^{3^4-1} w_j \tilde{\mathbf{f}}\left(\mathbf{q}^{(j)}\right) \end{aligned} \quad (11.76)$$

Again, combining the products of p in the same way in which we obtained (11.73) results in a set of weights

$$w_j = \begin{cases} \left(\frac{2}{3}\right)^4, & j = 0 \\ \left(\frac{2}{3}\right)^3 \left(\frac{1}{6}\right), & j = 1, \dots, 8 \\ \left(\frac{2}{3}\right)^2 \left(\frac{1}{6}\right)^2, & j = 9, \dots, 32 \\ \left(\frac{2}{3}\right) \left(\frac{1}{6}\right)^3, & j = 33, \dots, 64 \\ \left(\frac{1}{6}\right)^4, & j = 65, \dots, 80 \end{cases} \quad (11.77)$$

From (11.50) it follows that

$$\mathbf{q}^{(j)} = \sqrt{\frac{3}{2}} \mathbf{r}^{(j)} \quad (11.78)$$

where $\mathbf{r}^{(j)}$ is given by the generators

$$\mathbf{r}^{(j)} = \begin{cases} [0] \in \mathbb{R}^4, & j = 0 \\ [1] \in \mathbb{R}^4, & j = 1, \dots, 8 \\ [1, 1] \in \mathbb{R}^4, & j = 9, \dots, 32 \\ [1, 1, 1] \in \mathbb{R}^4, & j = 33, \dots, 64 \\ [1, 1, 1, 1] \in \mathbb{R}^4, & j = 65, \dots, 80 \end{cases} \quad (11.79)$$

This can be generalized to the general n_x -dimensional case with

$$\tilde{\mathbf{f}}(\mathbf{q}) = \sum_{j=0}^{3^{n_x}-1} w_j \tilde{\mathbf{f}}\left(\mathbf{q}^{(j)}\right) \quad (11.80)$$

where $\mathbf{q}^{(j)}$ is given by (11.78) with

$$\mathbf{r}^{(j)} = \begin{cases} [0] \in \mathbb{R}^n, & j = 0 \\ [1] \in \mathbb{R}^n, & j = 1, \dots, 2n \\ [1, 1] \in \mathbb{R}^n, & j = 2n + 1, \dots, 2(n + n^{(2)}) \\ [1, 1, 1] \in \mathbb{R}^n, & j = [2(n + n^{(2)}) + 1], \dots, \\ & [2(n + n^{(2)}) + 2^3 n_{(3)}] \\ \vdots & \\ [1, 1, \dots, 1] \in \mathbb{R}^n, & j = \left[2(n + n^{(2)}) + \sum_{i=3}^{n-2} 2^i n_{(i)}\right] \\ & + [1, \dots, 2^{n-1} n_{(n-1)}] \\ [1, 1, 1, \dots, 1] \in \mathbb{R}^n, & j = \left[2(n + n^{(2)}) + \sum_{i=3}^{n-1} 2^i n_{(i)}\right] \\ & + [1, \dots, 2^n n_{(n)}] \end{cases} \quad (11.81)$$

Here, n_x has been replaced by n for convenience and we have used the definitions (2.24) and (2.25) for $n^{(i)}$ and $n_{(i)}$, respectively. Also, for the last generator, there are

n copies of ± 1 . The weights follow immediately as

$$w_j = \begin{cases} \left(\frac{2}{3}\right)^n, & j = 0 \\ \left(\frac{2}{3}\right)^{n-1} \left(\frac{1}{6}\right), & j = 1, \dots, 2n \\ \vdots \\ \left(\frac{2}{3}\right) \left(\frac{1}{6}\right)^{n-1}, & j = [2(n+n^{(2)}) + \sum_{i=3}^{n-2} 2^i n_{(i)}] \\ & + [1, \dots, 2^{n-1} n_{(n-1)}] \\ \left(\frac{1}{6}\right)^n, & j = [2(n+n^{(2)}) + \sum_{i=3}^{n-1} 2^i n_{(i)}] \\ & + [1, \dots, 2^n n_{(n)}] \end{cases} \quad (11.82)$$

Now, inserting the general Hermite polynomial expansion of $\tilde{f}(\mathbf{q})$ (11.80) into the moment integral (11.4) yields

$$\mathbf{I}(\tilde{\mathbf{f}}) = \frac{\sqrt{2}}{(2\pi)^{n_x/2}} \int \sum_{j=0}^{3^{n_x}-1} w_j \tilde{\mathbf{f}}(\mathbf{q}^{(j)}) e^{-\mathbf{q}^\top \mathbf{q}} d\mathbf{q} \quad (11.83)$$

Changing variables back to an integral over \mathbf{c} , (11.83) becomes

$$\begin{aligned} \mathbf{I}(\tilde{\mathbf{f}}) &= \sum_{j=0}^{3^{n_x}-1} w_j \tilde{\mathbf{f}}(\mathbf{q}^{(j)}) \int \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &= \sum_{j=0}^{3^{n_x}-1} w_j \tilde{\mathbf{f}}(\mathbf{q}^{(j)}) \end{aligned} \quad (11.84)$$

Converting $\tilde{f}(\mathbf{q}^{(j)})$ using (11.5), (11.84) becomes

$$\mathbf{I} = \sum_{j=0}^{3^{n_x}-1} w_j \mathbf{f}(\hat{\mathbf{x}} + \sqrt{3} \mathbf{D} \mathbf{r}^{(j)}) \quad (11.85)$$

Letting

$$\mathbf{X}_{n-1|n-1}^{(j)} \triangleq \hat{\mathbf{x}}_{n-1|n-1} + \sqrt{3} \mathbf{D}_{n-1|n-1} \mathbf{r}^{(j)} \quad (11.86)$$

and using (11.85) to evaluate (5.51)–(5.52) results in the general Gauss–Hermite state prediction equations

$$\hat{\mathbf{x}}_{n|n-1} = \sum_{j=0}^{3^{n_x}-1} w_j \mathbf{f}(\chi_{n-1|n-1}^{(j)}) \quad (11.87)$$

and

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xx}} &= \sum_{j=0}^{3^{n_x}-1} w_j \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\times \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right]^T + \mathbf{Q} \end{aligned} \quad (11.88)$$

Similarly, defining

$$\chi_{n|n-1}^{(j)} \triangleq \hat{\mathbf{x}}_{n|n-1} + \sqrt{3} \mathbf{D}_{n|n-1} \mathbf{r}^{(j)} \quad (11.89)$$

the observation prediction equations (5.55)–(5.57) can be written as

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{3^{n_x}-1} w_j \mathbf{h}(\chi_{n|n-1}^{(j)}) \quad (11.90)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{zz}} &= \sum_{j=0}^{3^{n_x}-1} w_j \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right] \\ &\times \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T + \mathbf{R} \end{aligned} \quad (11.91)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{xz}} &= \sum_{j=0}^{3^{n_x}-1} w_j \left[\mathbf{f}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \\ &\times \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^T \end{aligned} \quad (11.92) \quad (11.93)$$

Note that the Gauss–Hermite prediction equations are identical to those of the UKF and SSKF, as well as the second form of the FDKF, except for the limits of the summation. We will summarize these sigma point Kalman filters and present their process diagram in Chapter 13.

The GHKF process is summarized in Table 11.1.

11.4 SPARSE GRID APPROXIMATION FOR HIGH DIMENSION/HIGH POLYNOMIAL ORDER

As noted above, the number of grid points needed for the Gauss–Hermite filter is given by

$$N_{gh} = m^{n_x} \quad (11.95)$$

TABLE 11.1 Multidimensional Gauss–Hermite Kalman Filter Process

Step 1.	Filter initialization:	Initialize $\hat{\mathbf{x}}_0$ and $\mathbf{P}_0^{\mathbf{x}}$
Step 2.	State vector prediction:	$\chi_{n-1 n-1}^{(j)} = \hat{\mathbf{x}}_{n n-1}$ $+ \sqrt{3} \mathbf{D}_{n-1 n-1} \mathbf{r}^{(j)}, \quad j = 0, 1, \dots, 3^{n_x} - 1$ $\hat{\mathbf{x}}_{n n-1} = \sum_{j=0}^{3^{n_x}-1} w_j \mathbf{f}(\chi_{n-1 n-1}^{(j)})$ $\mathbf{P}_{n n-1}^{\mathbf{x}\mathbf{x}} = \sum_{j=0}^{3^{n_x}-1} w_j [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}]$ $\times [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}]^\top + \mathbf{Q}$
Step 3.	Observation related prediction:	$\chi_{n n-1}^{(j)} = \hat{\mathbf{x}}_{n n-1}$ $+ \sqrt{3} \mathbf{D}_{n n-1} \mathbf{r}^{(j)}, \quad j = 0, 1, \dots, 3^{n_x} - 1$ $\hat{\mathbf{z}}_{n n-1} = \sum_{j=0}^{3^{n_x}-1} w_j \tilde{\mathbf{h}}(\chi_{n n-1}^{(j)})$ $\mathbf{P}_{n n-1}^{\mathbf{z}\mathbf{z}} = \sum_{j=0}^{3^{n_x}-1} w_j [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]$ $\times [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]^\top + \mathbf{R}$ $\mathbf{P}_{n n-1}^{\mathbf{x}\mathbf{z}} = \sum_{j=0}^{3^{n_x}-1} w_j [\mathbf{f}(\chi_{n-1 n-1}^{(j)}) - \hat{\mathbf{x}}_{n n-1}]$ $\times [\mathbf{h}(\chi_{n n-1}^{(j)}) - \hat{\mathbf{z}}_{n n-1}]^\top$
Step 4.	Kalman Filter Update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\mathbf{x}\mathbf{z}} (\mathbf{P}_{n n-1}^{\mathbf{z}\mathbf{z}})^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1})$ $\mathbf{P}_{n n}^{\mathbf{x}\mathbf{x}} = \mathbf{P}_{n n-1}^{\mathbf{x}\mathbf{x}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\mathbf{z}\mathbf{z}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\mathbf{x}\mathbf{x}}$ to a Track File
Step 6.	Return to Step 2.	

where m is the order of the Hermite polynomial used in the approximation and n_x is the dimension of the state vector. As the dimension of the state vector increases, the number of Sigma points required grows exponentially resulting in the “curse of dimensionality.” The fully populated square grid of Sigma points becomes so large at dimensions above 4 that the GHKF is rarely used. In the previous section of this chapter, we considered Hermite expansions of second order ($m = 3$) that required 3^{n_x} sigma points. The only way to reduce the number of points for this case is to use the UKF instead of the GHKF at a small cost in accuracy.

As noted just before (11.30), the Hermite interpolation method can be expanded to higher order. Of course, higher order interpolation requires more sigma points making the curse of dimensionality even worse. Because higher order Hermite interpolation requires the solution to polynomial equations of the same order as the Hermite polynomial, numerical methods are generally used to generate the eigenvalues, resulting in weights and sigma points that are irrational numbers vice the rational numbers shown in (11.48). It is these irrational numbers that are tabulated in such books as Ref. [7]. However, care must be taken when using the tabulated data since the weights are generated for the unnormalized Hermite polynomials.

Recent advances in multidimensional quadrature integration, based on the little known work of Smolyak [8], has made higher order Gauss–Hermite integration more manageable. Smolyak formulated a numerical method for multidimensional quadrature integration in a way that leads to a reduction of the number of grid points at the expense of accuracy. For a complete introduction to the mathematics of this grid point reduction method, the reader is referred to Refs [9–13]. Our presentation of the generation of the Hermite sigma points presented above conforms to the Smolyak method where the Sigma points are grouped hierarchically according to their distance from the origin, as can be seen for the general dimension case in (11.81). For sparse-grid numerical integration, one merely has to take a subset of the full grid and renormalize the reduced weight set. Unfortunately, this sparse-grid integration method does not reduce the number of required sigma points for Hermite polynomials of second order.

For example, one could take just the two lowest level of grid points, for example, $\mathbf{r}^{(0)} = [0] \in \mathbb{R}^{n_x}$, $j = 0$ and $\mathbf{r}^{(j)} = [1] \in \mathbb{R}^{n_x}$, $j = 1, \dots, n_x$. The original weights are therefore $w_0 = (2/3)^{n_x}$ and $w_j = (2/3)^{n_x-1}(1/6)$, $j = 1, \dots, n_x$. The weights must be renormalized by dividing all the weights by the sum of the weights, which is given by $(2/3)^{n_x-1}([4 + n_x]/6)$. It follows immediately that the renormalized weights are $w_0 = 4/(4 + n_n)$ and $w_j = 1/(4 + n_n)$, $j = 1, \dots, n_x$. This results in a sparse Gauss–Hermite filter that is identical in form to the UKF, (9.24) and (9.25), but with

$$\mathbf{c}^{(j)} = q\mathbf{r}^{(j)} = \sqrt{\frac{3}{2}}\mathbf{r}^{(j)}, \quad j = 0, 1, \dots, 2n_x \quad (11.96)$$

$$w_j = \begin{cases} 4/(4 + n), & j = 0 \\ \frac{1}{4}w_0, & j = 1, \dots, 2n_x \end{cases} \quad (11.97)$$

In a similar manner, one can take as many of the Sigma grid point subsets desired from (11.81) and after renormalizing the weights, create alternate sparse-grid GHKF filters. This method will allow the sparse-grid GHKF to be used for higher dimensional cases. In addition, for highly nonlinear cases, higher order Hermite polynomials can be used to develop higher order sparse-grid GHKF filters that give greater accuracy at a reasonable computational cost. Jia et al. [14] are the first (to my knowledge) to utilize this method in an application for spacecraft attitude estimation. In their paper, they have tabulated the number of points required for different levels of sparseness for GHKF filters up to seventh order for the Hermite polynomial and state vector dimensions up to 6.

11.5 APPLICATION OF THE GHKF TO THE DIFAR SHIP TRACKING CASE STUDY

Since the dynamic equation is linear for the DIFAR tracking problem, the GHKF process used for this problem is identical to that shown in Table 9.2. For the weights and sigma points, the values used for $\{(w_j, \mathbf{c}^{(j)})\}$, $j = 0, 1, \dots, n_x\}$ are given in (11.77)–(11.79), where we let $\mathbf{c}^{(j)} = \sqrt{3}\mathbf{r}^{(j)}$. We used the same initialization and dynamic noise q values used for the EKF, given in Sections 7.4.3 and 7.4.4, respectively. The 100 Monte Carlo track plots obtained for the GHKF for each of the signal SNR values used previously showed little discernible difference relative to those from the EKF and will therefore not be presented. Even the number of divergent tracks were identical to those of the EKF Monte Carlo tracks.

REFERENCES

1. Wilf HS. *Mathematics for the Physical Sciences*. New York: Wiley; 1962, p. 54.
2. Press WH, et. al. *Numerical Recipes in C*. Cambridge, UK: Cambridge University Press; 1992.
3. Ball JS. Orthogonal polynomials, Gaussian quadratures and PDEs. *Comput. Sci. & Eng.* 1999;92–95.
4. Liu Q, Jäckel P. A Note on multivariate Gauss–Hermite quadrature. *Biometrika* 1994;**81**(3):624–629.
5. Nia VP. Gauss–Hermite quadrature: *Numer. Stat. Method* 2006; Online:vahid.probstat.ch/paper/ghq.pdf.
6. Arasaratnam I, Haykin S, Elliot RJ. Discrete-time nonlinear filtering algorithms using Gauss–Hermite quadrature. *Proc. IEEE*, 2007;**95**(5):953–977.
7. Abramowitz M, Stegun IA. *Handbook of Mathematical Functions*. Dover Publications 1972.
8. Smolyak SA. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSSR* 1963;4:240–243.
9. Wasilkowski GW, Woźniakowski H. Explicit cost bounds of algorithms for multivariate tensor product problems. *J. Complex.* 1994;**11**(1):1–56.
10. Gerstner T, Griebel M. Numerical integration using sparse grids. *Numer. Algorithms*, 1998;**18**(4):209–232.
11. Gerstner T, Griebel M. Dimension-adaptive tensor-product quadrature. *Computing* 2003;**71**:65–87.
12. Heiss F, Winschel V. Estimation with Numerical Integration on Sparse Grids. Discussion Papers in Economics 916; 2006.
13. Heiss F, Winschel V. Likelihood approximation by numerical integration on sparse grids. *J. Econ.* 2008;**144**:62–80.
14. Jia B, Xin M, Cheng Y. Sparse Gauss–Hermite quadrature filter for spacecraft attitude estimation. *Am. Control Conf.*, 2010;ThA15.4:2873–2878

12

THE MONTE CARLO KALMAN FILTER

In the sigma point Kalman filters, after an affine transformation of the state vector, the nonlinear functions were expanded in a polynomial reducing the Gaussian-weighted integrals to a set of moment equations that were solved for a set of weights and sigma points. Although these sigma point methods have been referred to in the literature as an approximation to the density function, no approximations for the Gaussian density have been applied in our derivations.

However, a method is presented here where the Gaussian density is approximated by a set of Monte Carlo samples from that density. Remember from (5.51) that

$$\hat{\mathbf{x}}_{n|n-1} = \int \tilde{\mathbf{f}}(\mathbf{c}) \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \quad (12.1)$$

Suppose that we can generate a set of Monte Carlos sample $\{\mathbf{c}^{(j)}, j = 1, \dots, N_s\}$ from $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$. Now we express $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$ as the discrete approximation to the density given by

$$\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) \simeq \frac{1}{N_s} \sum_{j=1}^{N_s} \delta(\mathbf{c} - \mathbf{c}^{(j)}) \quad (12.2)$$

where $\delta(\mathbf{c} - \mathbf{c}^{(j)})$ represents the delta-Dirac mass located at $\mathbf{c}^{(j)}$.

Substituting (12.2) into the state prediction Equation (5.51) yields

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \frac{1}{N_s} \sum_{j=1}^{N_s} \int \tilde{\mathbf{f}}(\mathbf{c}) \delta(\mathbf{c} - \mathbf{c}^{(j)}) d\mathbf{c} \\ &= \frac{1}{N_s} \sum_{j=1}^{N_s} \tilde{\mathbf{f}}(\mathbf{c}^{(j)})\end{aligned}\quad (12.3)$$

It can be shown that as $N_s \rightarrow \infty$ this approximation of the integral, given by the sample mean, becomes exact (for a proof, see the first chapter in Ref. [1]). We must note that the factor $1/N_s$ is the importance weight of each sample and *not a probability*. The probability is given by the normalized density of samples in a given region represented by a normalized histogram.

Using (5.53), (12.3) becomes

$$\hat{\mathbf{x}}_{n|n-1} = \frac{1}{N_s} \sum_{j=1}^{N_s} \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}^{(j)}) \quad (12.4)$$

Using our standard definition for $\chi_{n-1|n-1}$ given by

$$\chi_{n-1|n-1}^{(j)} = \hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}^{(j)} \quad (12.5)$$

and defining $w_j \triangleq 1/N_s$, $\forall j$, we can rewrite (12.4) as

$$\hat{\mathbf{x}}_{n|n-1} = \sum_{j=1}^{N_s} w_j \mathbf{f}(\chi_{n-1|n-1}^{(j)}) \quad (12.6)$$

which has the same form as the state prediction equation for any of the sigma point Kalman filters.

It follows immediately that the remaining prediction equations for the Monte Carlo Kalman filter (MCKF) are given by the sigma point prediction equations

$$\mathbf{P}_{n|n-1}^{\mathbf{xx}} = \sum_{j=0}^{N_s} w_j \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right] \quad (12.7)$$

$$\times \left[\mathbf{f}(\chi_{n-1|n-1}^{(j)}) - \hat{\mathbf{x}}_{n|n-1} \right]^T + \mathbf{Q} \quad (12.8)$$

and

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{N_s} w_j \mathbf{h}(\chi_{n|n-1}^{(j)}) \quad (12.9)$$

$$\mathbf{P}_{n|n-1}^{\mathbf{zz}} = \sum_{j=0}^{N_s} w_j \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right] \quad (12.10)$$

$$\times \left[\mathbf{h} \left(\boldsymbol{\chi}_{n|n-1}^{(j)} \right) - \hat{\mathbf{x}}_{n|n-1} \right]^\top + \mathbf{R} \quad (12.11)$$

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}} = \sum_{j=0}^{N_s} w_j \left[\mathbf{f} \left(\boldsymbol{\chi}_{n-1|n-1}^{(j)} \right) - \hat{\mathbf{x}}_{n|n-1} \right] \quad (12.12)$$

$$\times \left[\mathbf{h} \left(\boldsymbol{\chi}_{n|n-1}^{(j)} \right) - \hat{\mathbf{x}}_{n|n-1} \right]^\top \quad (12.13)$$

TABLE 12.1 Multidimensional Spherical Simplex Kalman Filter Process

Step 1.	Filter initialization:	Set $0 \leq w_0 < 1$ Initialize $\hat{\mathbf{x}}_0$ and $\mathbf{P}_0^{\mathbf{x}}$ $w_j = 1/N_s, j = 1, \dots, N_s$ $\mathbf{c}^{(j)} \sim \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}), j = 1, \dots, N_s$
Step 2.	Generate samples State vector prediction:	$\boldsymbol{\chi}_{n-1 n-1}^{(j)} = \hat{\mathbf{x}}_{n-1 n-1} + \mathbf{D}_{n-1 n-1} \mathbf{c}^{(j)}, j = 1, \dots, N_s$ $\hat{\mathbf{x}}_{n n-1} = \sum_{j=1}^{N_s} w_j \mathbf{f} \left(\boldsymbol{\chi}_{n-1 n-1}^{(j)} \right)$ $\mathbf{P}_{n n-1}^{\mathbf{xz}} = \sum_{j=0}^{N_s} w_j \left[\mathbf{f} \left(\boldsymbol{\chi}_{n-1 n-1}^{(j)} \right) - \hat{\mathbf{x}}_{n n-1} \right]$ $\times \left[\mathbf{f} \left(\boldsymbol{\chi}_{n-1 n-1}^{(j)} \right) - \hat{\mathbf{x}}_{n n-1} \right]^\top + \mathbf{Q}$
Step 3.	Generate samples Observation-related prediction:	$\mathbf{c}^{(j)} \sim \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}), j = 1, \dots, N_s$ $\boldsymbol{\chi}_{n n-1}^{(j)} = \hat{\mathbf{x}}_{n n-1} + \mathbf{D}_{n n-1} \mathbf{c}^{(j)}, j = 1, \dots, N_s$ $\hat{\mathbf{z}}_{n n-1} = \sum_{j=0}^{N_s} w_j \tilde{\mathbf{h}} \left(\boldsymbol{\chi}_{n n-1}^{(j)} \right)$ $\mathbf{P}_{n n-1}^{\mathbf{zz}} = \sum_{j=0}^{N_s} w_j \left[\mathbf{h} \left(\boldsymbol{\chi}_{n n-1}^{(j)} \right) - \hat{\mathbf{z}}_{n n-1} \right]$ $\left[\mathbf{h} \left(\boldsymbol{\chi}_{n n-1}^{(j)} \right) - \hat{\mathbf{z}}_{n n-1} \right]^\top + \mathbf{R}$ $\mathbf{P}_{n n-1}^{\mathbf{xz}} = \sum_{j=0}^{N_s} w_j \left[\mathbf{f} \left(\boldsymbol{\chi}_{n-1 n-1}^{(j)} \right) - \hat{\mathbf{x}}_{n n-1} \right]$ $\times \left[\mathbf{f} \left(\boldsymbol{\chi}_{n n-1}^{(j)} \right) - \hat{\mathbf{z}}_{n n-1} \right]^\top$
Step 4.	Kalman filter update:	$\mathbf{K}_n \triangleq \mathbf{P}_{n n-1}^{\mathbf{xz}} \left(\mathbf{P}_{n n-1}^{\mathbf{zz}} \right)^{-1}$ $\hat{\mathbf{x}}_{n n} = \hat{\mathbf{x}}_{n n-1} + \mathbf{K}_n \left(\mathbf{z}_n^o - \hat{\mathbf{z}}_{n n-1} \right)$ $\mathbf{P}_{n n}^{\mathbf{xz}} = \mathbf{P}_{n n-1}^{\mathbf{xz}} - \mathbf{K}_n \mathbf{P}_{n n-1}^{\mathbf{zz}} \mathbf{K}_n^\top$
Step 5.	Store results	Store $\hat{\mathbf{x}}_{n n}$ and $\mathbf{P}_{n n}^{\mathbf{xz}}$ to a Track File
	Return to Step 2.	

In the later three prediction equations, $\chi_{n|n-1}^{(j)}$ is given by

$$\chi_{n|n-1}^{(j)} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{D}_{n|n-1} \mathbf{c}^{(j)} \quad (12.14)$$

where $\mathbf{c}^{(j)}$ represents a new set of Monte Carlo samples drawn from $\mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$. All of these prediction equations are in fact nothing more than the sample means and covariances of the nonlinear functions.

It must be noted that in this MCKF, the Monte Carlo samples are never reused. As we will show in Part III, there are some estimation methods for non-Gaussian densities where the Monte Carlo samples are reused over and over again, with just their weights updated in each iteration. This is where the term *particle* filter originated, because each Monte Carlo sample is considered a particle that propagates from one iteration to the next.

12.1 THE MONTE CARLO KALMAN FILTER

The MCKF process is summarized in Table 12.1.

REFERENCE

1. Doucet A, deFreitas N, Gordon N, editors *Sequential Monte Carlo Methods in Practice*. Springer; 2001.

13

SUMMARY OF GAUSSIAN KALMAN FILTERS

In this chapter, we will summarize some of the more important features of the Gaussian-based Kalman filters developed in Chapters 6–12. We first examine the LKF and EKF and present process flow diagrams for each. Then we address the sigma point class of Kalman filters, presenting a process flow diagram that is common to all of them. Two tables are then presented that define the sigma points and weights specific to each filter. This is followed by an assessment of the performance characteristics of each sigma point Kalman filter from the processing efficiency point of view.

13.1 ANALYTICAL KALMAN FILTERS

There are many dynamic systems where either the dynamic or observation equation is linear. This is true for most of the case studies considered in this book.

For some cases, the dynamic equation is linear while the observation equation is highly nonlinear. For these, the linear Kalman filter prediction Equations (6.3) and (6.6) can always be used for the state and state covariance predictions in lieu of any of the other prediction methods. For other tracking problems, the dynamic equation may be nonlinear while the observation equation is linear. For these problems, the reverse is true: one must use one of the nonlinear methods for state prediction, but the linear Kalman filter predictions (6.9)–(6.12) can be used for observation prediction. Finally, if both the dynamic and observation equations are linear, all of the LKF prediction equations are used.

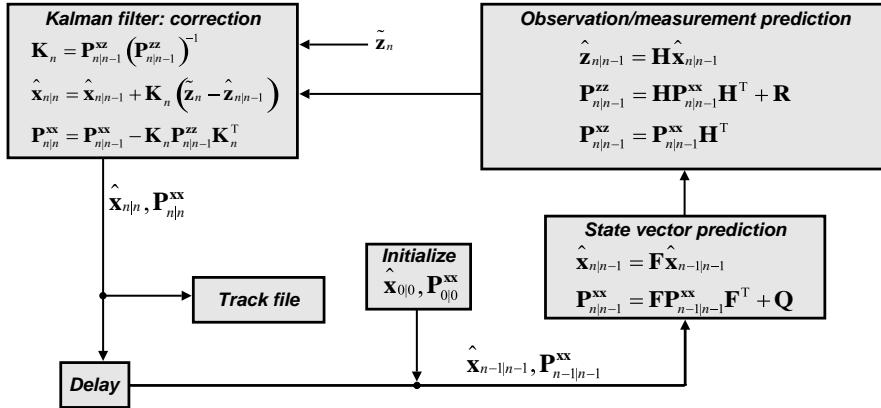


FIGURE 13.1 Graphical presentation of the linear Kalman filter process flow.

A process diagram for the LKF is presented in Figure 13.1

When the dynamic and/or observation equation is nonlinear, one of the nonlinear methods developed in Chapters 7–12 must be used. The method that has seen the most use over the past 50 years is the EKF. As we have shown in Chapter 7, the EKF involves an expansion of the nonlinear functions in a first-order Taylor polynomial resulting in a Kalman filter with the same form as the LKF, but with the added burden of Jacobian computations. The process diagram for this method is presented in Figure 13.2. In Chapter 7 we also developed a second-order extended Kalman filter, but it required the additional computation of the Hessian matrix, usually a difficult task.

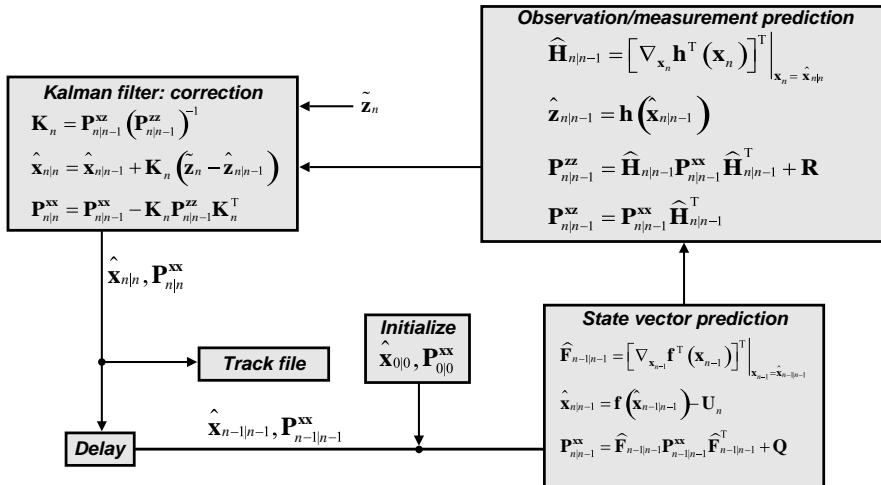


FIGURE 13.2 A graphical representation of the EKF process flow.

13.2 SIGMA POINT KALMAN FILTERS

In Chapter 8, we presented a method for replacing the Jacobian and Hessian matrices with their central finite difference approximations resulting in several versions of the FDKF. The covariance prediction equations for the standard FDKF, given by (8.41), (8.43), and (8.47), differed in form from those of the remaining sigma point methods. In addition, the modified version of the FDKF covariance prediction equations, (8.49), (8.50), and (8.51), were shown to be identically to those of the UKF. For these reasons, the FDKF will not be included in our summary of the sigma point Kalman filters.

The sigma point Kalman filters, UKF, SSKF, and GHKF, as well as the MCKF, share a similar structure so that all follow the common process flow shown in Figure 13.3, but differ in $c^{(j)}$ and w_j . The values for $c^{(j)}$ and w_j for each of the filters are presented in Tables 13.1 and 13.2, respectively. It is relatively easy to write a general subroutine for the sigma point Kalman filter, based on the process flow shown in Figure 13.3, that calls a second subroutine which selects the appropriate sigma points and weights from Tables 13.1 and 13.2.

Some of the sigma point Kalman filters suffer from what is known as the curse of dimensionality, where the number of integration points increase exponentially with the dimension of the state vector. The number of integration points required by the various filters is shown in Table 13.3. One can see that for state vectors with a large

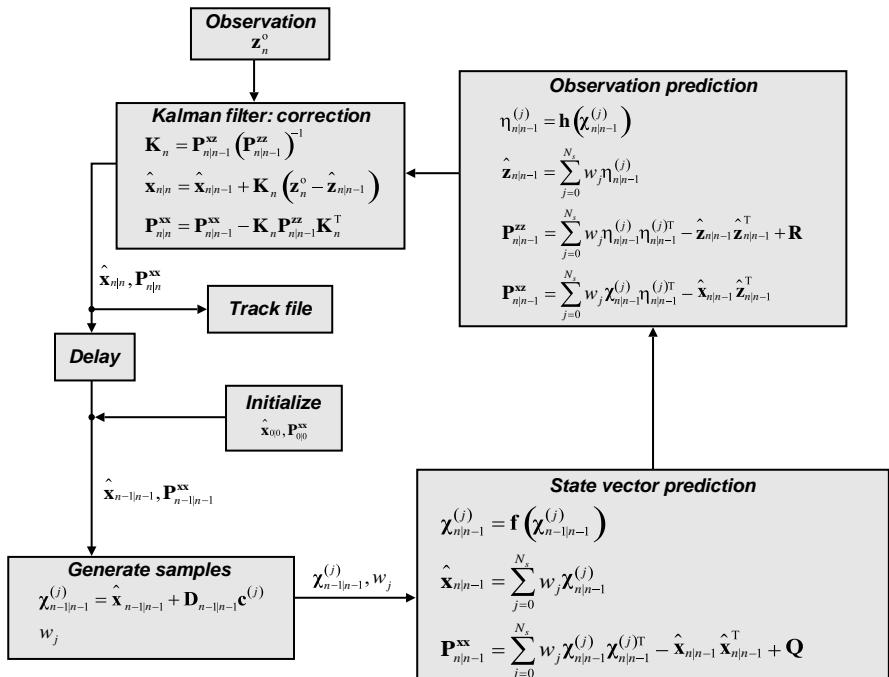


FIGURE 13.3 Process flow for general sigma point Kalman filter.

TABLE 13.1 Summary Data for Sigma Point Kalman Filters: Part 1—Form of c to Be Used for Sigma Points

Kalman Filter Method	Form of $c^{(j)}$
MCKF	$\mathbf{c}^{(j)} \sim N(\mathbf{c}; \mathbf{0}, \mathbf{I}), j = 1, \dots, N_s$
UKF	$\mathbf{c}^{(j)} = \sqrt{\frac{n_x}{1-w_0}} \mathbf{r}^{(j)}, j = 0, 1, \dots, 2n_x$
SSKF	$\mathbf{c}_{\mathbb{R}^{n_x}}^{(j)} = \begin{cases} [\mathbf{0}] \in \mathbb{R}^{n_x}, j = 0 \\ [\mathbf{1}] \in \mathbb{R}^{n_x}, j = 1, 2, \dots, 2n_x \end{cases}$
GHKF	$\mathbf{c}^{(j)} = \sqrt{3} \mathbf{r}^{(j)}; \text{ see (11.81) for } \mathbf{r}^{(j)}$

number of dimensions, the best filter from a computational point of view would be the SSKF, followed by the UKF. One would not want to use the GHKF for any problem in which the dimension exceeded four. For highly nonlinear dynamics, the accuracy of each filter is usually determined by the order of the polynomial expansion used within the integration. Thus, the MCKF will usually give the most accurate results since no approximation was used for the nonlinear function, as long as the number of random samples is high enough. This is followed closely by the GHKF that uses the highest polynomial order. The UKF will be ranked third due to its use of a third-order polynomial. And finally, the least accurate will be the SSKF. Thus, it is sometimes best

TABLE 13.2 Summary Data for Sigma Point Kalman Filters: Part 2—Form of Sigma Point Weights

Kalman Filter Method	Form of w_j
MCKF	$1/N_s$
UKF	$\begin{cases} 0 \leq w_0 \leq 1 \\ w_j = \frac{1-w_0}{2n_x}, j = 1, \dots, 2n_x \end{cases}$
SSKF	$\begin{cases} 0 \leq w_0 \leq 1 \\ w_j = \frac{1-w_0}{n_x+1}, j = 1, \dots, n_x + 1 \end{cases}$
GHKF	See (11.82) for w_j

TABLE 13.3 Comparison of the Number of Integration Points Required for the Various Sigma Point Kalman Filters

Number of Sample Points		Dimension of State Vector				
		1	2	4	8	16
UKF	$2n_x + 1$	3	5	9	17	33
SSKF	$n_x + 2$	3	4	6	10	18
GHKF	3^{n_x}	3	9	81	6561	43,046,721
MCKF	N_s	N_s	N_s	N_s	N_s	N_s

to try all of these filters on a given estimation problem and choose the one that yields the lowest RMS error within the problems computational constraints. An interesting discussion of accuracy, efficiency, and stability factors for these and related filters can be found in the paper by Wu et al. [1].

A simple set of Matlab subroutines that generate the set $\{w_i, c^{(i)}, i = 1, \dots, N_s\}$ for all of the sigma point Kalman filters, and for any state vector dimension, are presented in Listings 13.1 and 13.2. The first subroutine computes the weights and sigma points based on the formulas shown in Tables 13.1 and 13.2, respectively. It requires four inputs, a flag (`t_flag`) indicating which of the four sigma point Kalman filters requires the sigma points and weights, the state vector dimension ($L = n_x$), the number of sigma (or sample) points, LL , computed from Table 13.3, and the value `Stream`, which is the stream for the random number generator. The second subroutine, written by Neill Bryant Beltran of The Johns Hopkins University Applied Physics Laboratory, generates the Gauss–Hermite vector points. This function outputs sigma points with length u and dimensionality n . To do this, a combination of binary numbers and combinatorics is employed. First, all possible binary numbers with the same number of digits as dimensions is created. Next, this list of binary numbers is pruned such that the only remaining binary numbers have u “1”s. Next, a negative mask is created that has all possible combinations of 1 and –1. Finally, the “shape mask” and “negative mask” are combined to find all possible combinations of Gauss–Hermite vector points.

Listing 13.1 Generation of Weights and Sigma Points

```
function [w,c] = WeightssigmaPoints(t_flag,L,LL,Stream)
w = zeros(1,LL);
switch tflag
    case{1}      % UKF
        w(1) = 0.25;
        w(2:LL) = (1 - w(1))/(2*L);
        r = zeros(2*L+1,L);
        r(2:L+1,:) = eye(L);
        r(L+2:2*L+1,:) = -eye(L);
        c = sqrt(L/(1-w(1)))*r;
    case{2}      % SSKF
        w(1) = 0.25;
```

```
w(2:LL) = (1 - w(1))/(L+1);
c = zeros(LL,L);
c(2,1) = 1/sqrt(2*w(2));
c(3,1) = -1/sqrt(2*w(2));
for n = 2:L
    for q = 1:n
        c(q+1,1:n) = [c(q+1,1:n-1) 1/...
                        sqrt(n*(n+1)*w(2))];
    end
    c(n+2,1:n) = [zeros(1,n-1) -n/...
                    sqrt(n*(n+1)*w(2))];
end
case{3}      % GHKF
    fact1 = 2/3;
    fact2 = 1/6;
    r = zeros(LL,L);
    w(1) = fact1^L;
    q = 1;
    for u = 1:L
        rr = vectorPointGenerator(u,L);
        k = size(rr,1);
        r(q+1:q+k,:) = rr;
        w(q+1:q+k) = (fact1^(L-u))*(fact2^u);
        q = q+k;
    end
    c = sqrt(3)*r;
case{4}      % MCKF
    w = ones(1,LL)/LL;
    c = randn(Stream,LL,L);
end
```

Listing 13.2 Vector Point Generator

```
function out = vectorPointGenerator(u, n)
% Written by
% Neill Bryant Beltran
% 8 November 2010
if u == 0
    out = zeros(1, n);
    return
end
% Find all binary numbers with same number
% of digits as there are dimensions, n
shape_mask = dec2bin(2^n-1:-1:1) == '1';
shape_size = size(shape_mask);
% Remove all vectors that do not have u '1's
for ii = shape_size(1):-1:1
    if nnz(shape_mask(ii, :)) ~= u
        shape_mask(ii, :) = [];
    end
end
shape_size = size(shape_mask);
% Create the negative mask
```

```

neg_mask = (dec2bin(0:2^u-1) == '1')*1.0;
neg_mask(neg_mask == 0) = -1;
neg_mask = -1*neg_mask;
neg_size = size(neg_mask);
out = zeros(shape_size(1)*neg_size(1), n);
% Combine the shape mask and negative mask
for ii = 1:neg_size(1)
    for jj = 1:shape_size(1)
        kk = 1;
        for ll = 1:n
            if ~shape_mask(jj, ll)
                continue
            end
            out((ii-1)*shape_size(1)+jj, ll) = ...
                shape_mask(jj, ll)*neg_mask(ii, kk);
            kk = kk + 1;
        end
    end
end

```

Additional methods for numerical evaluation of Gaussian-weighted integrals can be found in Refs [2–7]. Many of the methods discussed in the referenced papers can be applied in a manner similar to the sigma point Kalman filters developed above. Because most of them are of higher order than the ones presented in Chapters 8–12, they will require many more additional integration points than the UKF, with the number of required points falling between those of the UKF and GHKF. A more traditional approach to the development of sigma point Kalman filters can be found in the excellent book by Candy [8], where regression methods are used that are similar to those used in the original development of the Kalman filter.

13.3 A MORE PRACTICAL APPROACH TO UTILIZING THE FAMILY OF KALMAN FILTERS

A close examination of the dynamic and observation models given by (3.17) and (3.18) for the various case studies reveal that a hierarchy of models exists. This hierarchy is presented in Table 13.4.

Based on this model hierarchy, it is almost obvious that the *dynamic prediction* process can be considered to be completely independent of the *observation prediction*

TABLE 13.4 Hierarchy of Dynamic and Observation Models

Combination	Dynamic Model	Observation Model
Linear/linear	$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{v}_{n-1}$	$\mathbf{z}_n = \mathbf{H}\mathbf{x}_n + \mathbf{w}_n$
Linear/nonlinear	$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{v}_{n-1}$	$\mathbf{z}_n = \mathbf{f}(\mathbf{x}_n) + \mathbf{w}_n$
Nonlinear/linear	$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{v}_{n-1}$	$\mathbf{z}_n = \mathbf{H}\mathbf{x}_n + \mathbf{w}_n$
Nonlinear/nonlinear	$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{v}_{n-1}$	$\mathbf{z}_n = \mathbf{f}(\mathbf{x}_n) + \mathbf{w}_n$

process. This enables one to use the Kalman filter method that is most suited for each model. For example, for the linear/nonlinear case, one can use the linear Kalman filter equations for dynamic prediction and one of the nonlinear Kalman filter methods for observation prediction. Similarly, for the nonlinear/nonlinear case, when the state vector \mathbf{x}_n is of high dimension and the observation vector \mathbf{z}_n is of low dimension, then one could use the UKF for dynamic prediction (since the GHKF is impractical) but use the GHKF for observation prediction (since it is usually the most accurate). This consideration reveals a flexibility within the Kalman filter structure that can be put to practical use in setting up the most efficient estimation procedure for the problem at hand. Of course, from a software implementation point of view, this may not be the most efficient approach.

REFERENCES

1. Wu Y, Hu D, Wu M, Hu X. A numerical-integration perspective on Gaussian filters. *IEEE Trans. Sig. Proc.* 2006;54(8):2910–2921.
2. McNamee J, Stenger F. Construction of fully symmetric numerical integration formulas. *Numerische Math.* 1967;10:327–344.
3. Haber S. Numerical evaluation of multiple integrals. *SIAM Rev.* 1970;12(4):481–526.
4. Dunavant D A. Efficient symmetrical cubature rules for complete polynomials of higher degree over the unit cube. *Int. J. Numer. Methods Eng.* 1986;23:397–407.
5. Genz A, Monahan J. A stochastic algorithm for higher-dimensional integrals over unbounded regions with Gaussian weights. *J. Comput. Appl. Math.* 1999;112:71–81.
6. Cools R. Advances in multidimensional integration. *J. Comput. Appl. Math.* 2002;149:1–12.
7. Lu J, Darmofal D L. Higher-dimensional integration with Gaussian weight for applications in probabilistic design. *SIAM J. Sci. Comput.* 2004;26(2):613–624.
8. Candy JV. *Bayesian Signal Processing: Classical, Modern, and Particle Filtering Methods*, Hoboken NJ: Wiley; 2009.

14

PERFORMANCE MEASURES FOR THE FAMILY OF KALMAN FILTERS

In this chapter, we will address several general methods to measure the performance of any of the Kalman filters developed in Chapters 6–12. The performance measures fall into two classes. The first class is applicable when ground truth tracks are not available or where the number of ground truth tracks available for a given scenario are not statistically significant, such as in the analysis of a small number of field tests. For this case, only the filters estimated track covariance matrix can be used as an input to generate successive track error ellipses as a performance measure. On the other hand, a larger class of performance measures are available when a simulation is used to generate a completely known target track. This is true because a large number of Monte Carlo observation data sets can be generated that allow for the use of a variety of specific tracker performance measures, such as the root mean squared (RMS) error, the Cramer–Rao lower bound (CRLB), and the percentage of divergent tracks. In this chapter, all of these performance measures will be discussed and specific implementation issues associated with each will be addressed.

14.1 ERROR ELLIPSES

In applications of track estimation filters when the true state is unknown, such as for a fielded system tracking a real ship, or when only a small number of field tests are conducted and the truth state is only approximately known, the best way to gauge the

performance of a tracking filter is to use the estimated covariance matrix output by the filter to generate and plot error ellipses around the filtered track state. For clarity of presentation only the two-dimensional Cartesian coordinate system positions are used due to the inherent two-dimensional nature of displays.

At any discrete time t_n , the probability density function of the track state \mathbf{x}_n is given by the Gaussian

$$\mathbf{x}_n \sim \mathcal{N}(\mathbf{x}_n; \hat{\mathbf{x}}_{n|n}, \mathbf{P}_{n|n}^{\text{xx}}) \quad (14.1)$$

where $(\hat{\mathbf{x}}_{n|n}, \mathbf{P}_{n|n}^{\text{xx}})$ are the output of the Kalman filter update equations in Cartesian coordinates. Now, dropping the time subscript, examine *only* the two-dimensional position components given by $\mathbf{x} = [r^x, r^y]^\top$. Examination of Figure 2.9 shows that the contour ellipses of constant probability (before the affine transformation) are generally elongated along the major axis and may also be skewed so that the axes of the ellipse are not aligned with the axes associated with the state position vector. The question answered by the procedures developed below is, given a *containment* probability, how does one plot the associated error ellipses in the same coordinate system as that of the state vector. The containment probabilities correspond to the contours of constant probability.

14.1.1 The Canonical Ellipse

The exponent of the probability distribution for the position components can be represented by

$$s = (\mathbf{x} - \hat{\mathbf{x}})^\top \mathbf{P}^{-1} (\mathbf{x} - \hat{\mathbf{x}}) \quad (14.2)$$

where \mathbf{P} is the error covariance matrix of the position components. Error ellipses are defined by

$$(\mathbf{x} - \hat{\mathbf{x}})^\top \mathbf{P}^{-1} (\mathbf{x} - \hat{\mathbf{x}}) \leq C^2 \quad (14.3)$$

where C^2 is the containment area, which depends on the desired containment probability, P_c , as will be shown below.

First, we need to rewrite (14.3) in the standard form of the equation for an ellipse. Performing an eigenvalue decomposition of \mathbf{P} , we can write

$$\mathbf{P} = \mathbf{V} \mathbf{D} \mathbf{V}^\top \quad (14.4)$$

where \mathbf{D} is a 2×2 diagonal matrix containing the eigenvalues of \mathbf{P}

$$\mathbf{D} \triangleq \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad (14.5)$$

and \mathbf{V} is a 2×2 matrix constructed from the two eigenvectors of \mathbf{P} , and is given by

$$\mathbf{V} \triangleq [\mathbf{v}_1, \mathbf{v}_2] \quad (14.6)$$

Here, \mathbf{v}_1 and \mathbf{v}_2 are 2×1 column vectors. It is easy to show that for a symmetric, nonsingular matrix \mathbf{P}

$$\mathbf{P}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^\top$$

Now (14.3) can be rewritten as

$$(\mathbf{x} - \hat{\mathbf{x}})^\top \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^\top (\mathbf{x} - \hat{\mathbf{x}}) \leq C^2 \quad (14.7)$$

Letting

$$\mathbf{y} \triangleq \mathbf{V}^\top (\mathbf{x} - \hat{\mathbf{x}}) \quad (14.8)$$

we can rewrite (14.7) as

$$\mathbf{y}^\top \mathbf{D}^{-1} \mathbf{y} = \frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} \leq C^2 \quad (14.9)$$

Choosing λ_1 as the largest eigenvalue, and defining

$$a \triangleq C\sqrt{\lambda_1} \quad (14.10)$$

$$b \triangleq C\sqrt{\lambda_2} \quad (14.11)$$

and choosing the equality in (14.7), we obtain the canonical elliptical equation

$$\frac{y_1^2}{a^2} + \frac{y_2^2}{b^2} = 1 \quad (14.12)$$

14.1.2 Determining the Eigenvalues of \mathbf{P}

To determine the eigenvalues and eigenvectors of the symmetric matrix \mathbf{P} , examine the eigenvalue equation

$$\mathbf{P}\mathbf{v} = \lambda\mathbf{v} \quad (14.13)$$

or

$$[\mathbf{P} - \lambda\mathbf{I}]\mathbf{v} = \mathbf{0} \quad (14.14)$$

The polynomial $f(\lambda) = |\mathbf{P} - \lambda\mathbf{I}|$ is called the characteristic polynomial of \mathbf{P} . The roots of $f(\lambda) = 0$ yield the eigenvalues λ_i

$$|\mathbf{P} - \lambda\mathbf{I}| = \begin{vmatrix} P_{11} - \lambda & P_{12} \\ P_{12} & P_{22} - \lambda \end{vmatrix} = 0 \quad (14.15)$$

which leads to a solutions

$$\lambda_1 = \frac{P_{11} + P_{22}}{2} + \frac{1}{2}\sqrt{(P_{11} - P_{22})^2 + 4P_{12}^2} \quad (14.16)$$

$$\lambda_2 = \frac{P_{11} + P_{22}}{2} - \frac{1}{2}\sqrt{(P_{11} - P_{22})^2 + 4P_{12}^2} \quad (14.17)$$

14.1.3 Determining the Error Ellipse Rotation Angle

For the eigenvector \mathbf{v}_1 corresponding to the eigenvalue λ_1 , (14.14) becomes

$$\begin{bmatrix} P_{11} - \lambda_1 & P_{12} \\ P_{12} & P_{22} - \lambda_1 \end{bmatrix} \mathbf{v}_1 = \mathbf{0} \quad (14.18)$$

Let

$$\mathbf{v}_1 \triangleq \begin{bmatrix} \xi_{11} \\ \xi_{12} \end{bmatrix} \quad (14.19)$$

$$\mathbf{v}_2 \triangleq \begin{bmatrix} \xi_{21} \\ \xi_{22} \end{bmatrix} \quad (14.20)$$

where (ξ_{11}, ξ_{12}) and (ξ_{21}, ξ_{22}) are the Cartesian components of \mathbf{v}_1 and \mathbf{v}_2 , respectively. Solving (14.18) for ξ_{12} in terms of ξ_{11} yields

$$\xi_{12} = -\frac{P_{12}}{P_{22} - \lambda_1} \xi_{11} \quad (14.21)$$

The rotation angle of the ellipse relative to the Cartesian axes is depicted in Figure 14.1. It follows immediately from the figure that the error ellipse rotation angle is given by

$$\varphi = \tan^{-1} \left(\frac{\xi_{12}}{\xi_{11}} \right) = \tan^{-1} \left(-\frac{P_{12}}{\lambda_1 - P_{22}} \right) \quad (14.22)$$

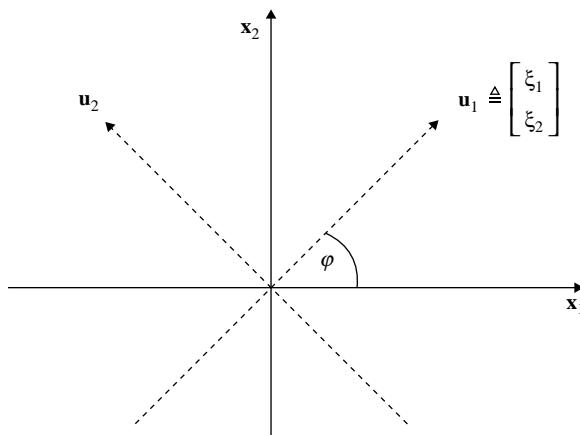


FIGURE 14.1 Error ellipse rotation angle.

14.1.4 Determination of the Containment Area

Since the components of \mathbf{x} are jointly Gaussian, as seen by (14.1), then the component of \mathbf{y} are also, so we can write

$$\mathbf{y} \sim \mathcal{N}(\mathbf{y}; \mathbf{0}, \mathbf{D}) \quad (14.23)$$

and the pdf of \mathbf{y} can be written as

$$p(\mathbf{y}) = \frac{1}{2\pi\sqrt{\lambda_1\lambda_2}} e^{-\left[\frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2}\right]} \quad (14.24)$$

Thus, the containment probability can be expressed as

$$P_C = P\left\{\frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} \leq C^2\right\} = \iint_{\frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} \leq C^2} p(\mathbf{y}) d\mathbf{y} \quad (14.25)$$

Now, let

$$r^2 \triangleq \frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} \leq C^2 \quad (14.26)$$

which leads to

$$y_1 = \sqrt{\lambda_1} r \cos \varphi \quad (14.27)$$

$$y_2 = \sqrt{\lambda_2} r \sin \varphi \quad (14.28)$$

$$dy_1 dy_2 = \sqrt{\lambda_1 \lambda_2} r dr d\varphi \quad (14.29)$$

The containment probability (14.25) now becomes

$$\begin{aligned} P_C &= \frac{1}{2\pi} \int_0^{2\pi} d\varphi \int_0^C r e^{-\frac{1}{2}r^2} dr \\ &= \int_0^C r e^{-\frac{1}{2}r^2} dr \\ &= 1 - e^{-\frac{1}{2}C^2} \end{aligned} \quad (14.30)$$

Solving for C results in

$$C = \sqrt{-2 \ln(1 - P_C)} \quad (14.31)$$

Thus, selecting a containment probability completely determines the value of C .

14.1.5 Parametric Plotting of Error Ellipse

A canonical ellipse (one in the form of (14.12)) can be expressed as the path of points $(x_e(\theta), y_e(\theta))$, where

$$x_e(\theta) = x_c + a \cos \theta \cos \varphi - b \sin \theta \sin \varphi \quad (14.32)$$

$$y_e(\theta) = y_c + a \cos \theta \sin \varphi + b \sin \theta \cos \varphi \quad (14.33)$$

Here, (x_c, y_c) is the center of the ellipse, φ is the angle between the Cartesian x -axis and the major axis of the ellipse, and the parametric vector θ is defined as the row vector $\theta \triangleq [0, 0.01, \dots, 2\pi]$.

Define the parametric matrix

$$\mathbf{Q}(\theta) = \begin{bmatrix} \mathbf{q}_1(\theta) \\ \mathbf{q}_2(\theta) \end{bmatrix} = \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} \quad (14.34)$$

From (14.10) and (14.11), we can identify

$$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} = C \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix} = C\mathbf{D}^{1/2} \quad (14.35)$$

and

$$\mathbf{V} = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix} = \begin{bmatrix} \xi_{11} & \xi_{21} \\ \xi_{12} & \xi_{22} \end{bmatrix} \quad (14.36)$$

The parametric equation for the path of the ellipse can now be written in vector-matrix form as

$$\mathbf{x}_e(\theta) = \begin{bmatrix} x_e(\theta) \\ y_e(\theta) \end{bmatrix} = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} + \begin{bmatrix} \xi_{11}C\sqrt{\lambda_1}\mathbf{q}_1(\theta) + \xi_{12}C\sqrt{\lambda_2}\mathbf{q}_2(\theta) \\ \xi_{21}C\sqrt{\lambda_1}\mathbf{q}_1(\theta) + \xi_{22}C\sqrt{\lambda_2}\mathbf{q}_2(\theta) \end{bmatrix} \quad (14.37)$$

where we have placed the center of the ellipse at (\hat{x}, \hat{y}) . Or, we could express the parametric equation as

$$\mathbf{x}_e(\theta) = \hat{\mathbf{x}} + \mathbf{V}\mathbf{C}\mathbf{D}^{1/2}\mathbf{Q}(\theta) \quad (14.38)$$

with $\hat{\mathbf{x}} \triangleq [\hat{x}, \hat{y}]$.

Remark 14.1 *The eigenvalue and eigenvector matrices, V and D , respectively, can be obtained using the Matlab function call $[V,D] = \text{eig}(P)$.*

A procedure for generating error ellipse data that can be plotted over a Cartesian track estimate plot is as follows:

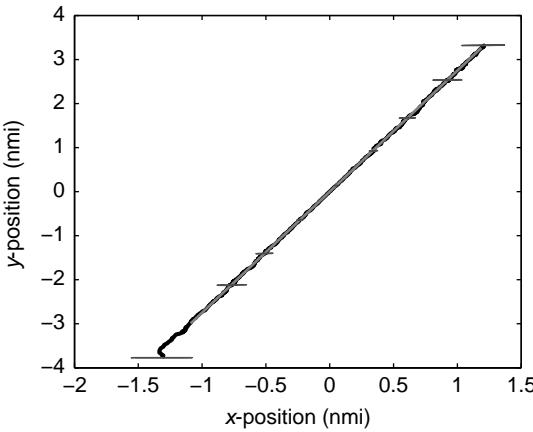


FIGURE 14.2 An example of error ellipses overplotted on the DIFAR UKF track output.

1. Pick a containment probability, P_C , usually 0.9, and calculate C using (14.31)
2. Generate the parametric vector $\theta \triangleq [0, 0.01, \dots, 2\pi]$. Note that one can use any desirable interval other than 0.01.
3. Generate $\mathbf{Q}(\theta)$ from (14.34)
4. Generate the eigenvalue and eigenvector matrices \mathbf{V} and \mathbf{D} of the covariance matrix $\mathbf{P}_{n|n}^{\text{xx}}$
5. Generate $\mathbf{x}_e(\theta) = \hat{\mathbf{x}}_{n|n} + \mathbf{V}\mathbf{C}\mathbf{D}^{1/2}\mathbf{Q}(\theta)$.

14.1.6 Error Ellipse Example

The plotting of an error ellipse at every 100th data point for the DIFAR UKF track output is shown in Figure 14.2. This plot allows one to evaluate the estimated uncertainty in the track estimates at various points along the track when the true track is unknown.

14.2 ROOT MEAN SQUARED ERRORS

Using (3.16), at time t_n we can write the state covariance matrix as

$$\mathbf{P}_{n|n}^{\text{xx}} = \mathcal{E} \left\{ (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}) (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top \right\} \quad (14.39)$$

When \mathbf{x}_n is known, as it would be in a simulation, and $\hat{\mathbf{x}}_{n|n}$ is unbiased, the square root of the diagonal elements $\mathbf{P}_{n|n}^{\text{xx}}$ can be *approximated* by conditional root mean squared errors (or the square root of the mean of the squared errors).

As noted in Chapter 4, the first task of a simulation is to generate a truth *state* trajectory, $\{\mathbf{x}_n, n = 0, 1, \dots, N\}$, for all times of interest $\{t_n, n = 0, 1, \dots, N\}$.

A truth *observation* trajectory is then generated from the truth state trajectory and independent noise is added to each observation to create a set of *simulated observations*, $\{\mathbf{z}_n, n = 0, 1, \dots, N\}$. To compute the RMS errors, multiple sets of observations are created in a Monte Carlo fashion, so that there are M sets of independent observations, $\{\mathbf{z}_{n,m}; n = 0, 1, \dots, N; m = 1, 2, \dots, M\}$. Note that the observation noise must be independent from time-sample to time-sample ($\mathbf{z}_{i,m}$ must be independent from $\mathbf{z}_{j,m}$, $\forall i \neq j$) and from Monte Carlo run to Monte Carlo run ($\mathbf{z}_{n,l}$ must be independent from $\mathbf{z}_{n,k}$, $\forall l \neq k$).

Each set of Monte Carlo observations are now used in a given tracking filter to produce a set of track file outputs, $\hat{\mathbf{x}}_{n|n}(m)$ and a mean squared error matrix at each n can be obtained from

$$\mathbf{S}(\hat{\mathbf{x}}_{n|n}) \triangleq \frac{1}{M} \sum_{m=1}^M [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}(m)] [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}(m)]^\top \quad (14.40)$$

The *RMS error* are then obtained from

$$\mathbf{e}_{\text{RMS}}(\mathbf{x}_n) = \text{diag} \left\{ \left[\frac{1}{M} \sum_{m=1}^M [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}(m)] [\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}(m)]^\top \right]^{1/2} \right\} \quad (14.41)$$

where $\text{diag}\{\bullet\}$ means “take the diagonal terms and place them in a column vector.” Or, to put it another way, for a four-dimensional state vector $\mathbf{x}_n = [x_{1,n}, x_{2,n}, x_{3,n}, x_{4,n}]^\top$

$$\mathbf{e}_{\text{RMS}}(\mathbf{x}_n) = \begin{bmatrix} e_{\text{RMS}}(x_{1,n}) \\ e_{\text{RMS}}(x_{2,n}) \\ e_{\text{RMS}}(x_{3,n}) \\ e_{\text{RMS}}(x_{4,n}) \end{bmatrix} \quad (14.42)$$

with

$$e_{\text{RMS}}(x_{i,n}) \triangleq \left[\frac{1}{M} \sum_{m=1}^M [x_{i,n} - \hat{x}_{i,n|n}(m)]^2 \right]^{1/2} \quad (14.43)$$

14.3 DIVERGENT TRACKS

One has to be careful when calculating the RMS error performance, since even one divergent track among 100 Monte Carlo runs will skew the RMS error statistic. Divergent tracks are to be considered outliers that do not conform to the Gaussian assumption that the RMS error is an unbiased sampled data estimate of the standard deviation. Unfortunately, what constitutes a divergent track is a subjective decision that is usually scenario dependent.

For the DIFAR case-study performance analysis, a track estimate was considered divergent if either the x or y -axis estimate at the final time deviated from the truth

value by more than 2 nmi. As seen in Chapters 7–12, for the DIFAR case study the number of divergent tracks increased as the SNR decreased below 5 dB.

14.4 CRAMER–RAO LOWER BOUND

We can now ask the question, what is the BEST (smallest) achievable RMS error for a given scenario (set of dynamic and observation models and state trajectories). Since the observation noise is always present, the best achievable RMS errors will occur when the dynamic model is used without any dynamic noise.

Why are lower bounds useful? Assessing achievable estimation algorithm (tracker) performance may be difficult and a lower bound will give an indication of the performance. In many cases lower bounds on performance are required to determine whether a set of imposed system performance requirements are realistic or not.

Much of the material in this section is based on Refs [1–3], and the references contained therein. We stress here the Cramer–Rao lower bound, but [2] goes beyond this bound to consider a wide variety of different bounds applicable to problems other than the Bayesian estimation methods addressed in this book.

14.4.1 The One-Dimensional Case

Consider the one-dimensional case where the process (dynamic) noise is zero making x deterministic. Then, the only contributor to errors in \hat{x} will be the observation noise, which is captured by the likelihood function. For a normalized likelihood function, we know that

$$\int p(z|x) dz = 1 \quad (14.44)$$

It immediately follows that the expected value of $\hat{x}(z)$ is given by

$$\int \hat{x}(z) p(z|x) dz = x \quad (14.45)$$

Based on this, since x is deterministic we can write

$$\int (\hat{x}(z) - x) p(z|x) dz = 0 \quad (14.46)$$

Taking the derivative with respect to x leads to the progression

$$\frac{\partial}{\partial x} \int (\hat{x} - x) p(z|x) dz = 0 \quad (14.47)$$

$$-\int p(z|x) dz + \int (\hat{x} - x) \frac{\partial p(z|x)}{\partial x} dz = 0 \quad (14.48)$$

$$\int (\hat{x} - x) \frac{\partial p(z|x)}{\partial x} dz = \int p(z|x) dz \quad (14.49)$$

$$\int (\hat{x} - x) \frac{\partial p(z|x)}{\partial x} dz = 1 \quad (14.50)$$

Noting that

$$\frac{\partial}{\partial x} \ln p(z|x) = \frac{1}{p(z|x)} \frac{\partial p(z|x)}{\partial x} \quad (14.51)$$

(14.50) becomes

$$\int (\hat{x} - x) p(z|x) \frac{\partial}{\partial x} \ln p(z|x) dz = 1 \quad (14.52)$$

or

$$1 = \int \left[(\hat{x} - x) \sqrt{p(z|x)} \right] \left[\frac{\partial}{\partial x} \ln p(z|x) \sqrt{p(z|x)} \right] dz \quad (14.53)$$

Taking the square of both sides yields

$$1 = \left\{ \int \left[(\hat{x} - x) \sqrt{p(z|x)} \right] \left[\frac{\partial}{\partial x} \ln p(z|x) \sqrt{p(z|x)} \right] dz \right\}^2 \quad (14.54)$$

Using the Swartz inequality, we obtain

$$1 \leq \int (\hat{x} - x)^2 p(z|x) dz \int \left[\frac{\partial}{\partial x} \ln p(z|x) \right]^2 p(z|x) dz \quad (14.55)$$

or

$$\int (\hat{x} - x)^2 p(z|x) dz \geq \frac{1}{\int \left[\frac{\partial}{\partial x} \ln p(z|x) \right]^2 p(z|x) dz} \quad (14.56)$$

The left-hand side of (14.56) is just the variance, σ_x^2 , and the denominator on the right-hand side can be identified as the *Fisher information* $J \triangleq \mathcal{E}\left\{\left[\frac{\partial}{\partial x} \ln p(z|x)\right]^2\right\}$. We can now write

$$\sigma_x^2 \geq \frac{1}{\mathcal{E}\left\{\left[\frac{\partial}{\partial x} \ln p(z|x)\right]^2\right\}} \triangleq \frac{1}{J} \quad (14.57)$$

where we define the CRLB for the one-dimensional case as

$$\text{CRLB} = \frac{1}{\mathcal{E}\left\{\left[\frac{\partial}{\partial x} \ln p(z|x)\right]^2\right\}} \quad (14.58)$$

Since the variance can be estimated by the RMS error in a Monte Carlo simulation, the square root of the CRLB becomes the lower bound on the RMS error.

14.4.2 The Multidimensional Case

For a multidimensional case, the Fisher information becomes the Fisher information matrix \mathbf{J}_n and the CRLB becomes \mathbf{J}_n^{-1} . When process noise is included in the state dynamic model, the inverse of the Fisher information matrix is the lower bound of $\mathbf{P}_{n|n}^{\text{xx}}$

$$\mathbf{P}_{n|n}^{\text{xx}} \triangleq \mathcal{E}\left\{\left[\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}\right] \left[\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}\right]^T\right\} \geq \mathbf{J}_n^{-1} \quad (14.59)$$

Consider the true state trajectory defined as $\mathbf{x}_{1:n} \triangleq \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n\}$. Assume that the initialization of the state estimate, \mathbf{x}_0 , prior to any observations, has the probability $p(\mathbf{x}_0)$. The unbiased estimate of the state \mathbf{x}_n at time t_n is given by the filter output $\hat{\mathbf{x}}_{n|n}$, which has made use of the observations $\mathbf{z}_{1:n} \triangleq \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$. The CRLB for the trajectory $\mathbf{x}_{1:n}$ is given by the inverse of the *trajectory information matrix* $\mathbf{J}_{1:n}$

$$\mathcal{E}\left\{\left[\mathbf{x}_{1:n} - \hat{\mathbf{x}}_{1:n|1:n}\right] \left[\mathbf{x}_{1:n} - \hat{\mathbf{x}}_{1:n|1:n}\right]^T\right\} \geq \mathbf{J}_{1:n}^{-1} = \text{CRLB}_{\text{traj}} \quad (14.60)$$

Extending (14.57) to the multidimensional case yields the *first form* of $\mathbf{J}_{1:n}$

$$\mathbf{J}_{1:n} \triangleq \mathcal{E}\left\{\left[\nabla_{\mathbf{x}_{1:n}} \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})\right] \left[\nabla_{\mathbf{x}_{1:n}} \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})\right]^T\right\} \quad (14.61)$$

Noting that (14.51) can be written in the multidimensional form

$$\nabla_{\mathbf{x}_{1:n}}^T \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n}) = \frac{1}{p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})} \nabla_{\mathbf{x}_{1:n}}^T p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n}) \quad (14.62)$$

After algebra, using (14.62) in (14.61) yields the *second form* of $\mathbf{J}_{1:n}$

$$\mathbf{J}_{1:n} \triangleq -\mathcal{E}\left\{\nabla_{\mathbf{x}_{1:n}} \nabla_{\mathbf{x}_{1:n}}^T \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})\right\} \quad (14.63)$$

Here, we must note that

- $p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})$ is used if the target state is stochastic due to the inclusion of process noise \mathbf{v}_n .
- $p(\mathbf{z}_{1:n}|\mathbf{x}_{1:n})$ is used if the target state is deterministic (zero process noise).
- $\nabla_{\mathbf{x}_{1:n}}$ is evaluated at the *true value* of $\mathbf{x}_{1:n}$.

14.4.3 A Recursive Approach to the CRLB

First, it would be useful to have a method for writing $\mathbf{J}_{1:n}$ in terms of \mathbf{J}_n and $\mathbf{J}_{1:n-1}$. Such a method was developed by Tichavský et al. [2,4]. Noting that we can decompose $\mathbf{x}_{1:n}$ and $\nabla_{\mathbf{x}_{1:n}}$ into

$$\mathbf{x}_{1:n} = \begin{bmatrix} \mathbf{x}_{1:n-1} \\ \mathbf{x}_n \end{bmatrix} \quad (14.64)$$

and

$$\nabla_{\mathbf{x}_{1:n}} = \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-1}} \\ \nabla_{\mathbf{x}_n} \end{bmatrix} \quad (14.65)$$

then $\nabla_{\mathbf{x}_{1:n}} \nabla_{\mathbf{x}_{1:n}}^\top$ can be decomposed into

$$\begin{aligned} \nabla_{\mathbf{x}_{1:n}} \nabla_{\mathbf{x}_{1:n}}^\top &= \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-1}} \\ \nabla_{\mathbf{x}_n} \end{bmatrix} \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-1}}^\top & \nabla_{\mathbf{x}_n}^\top \end{bmatrix} \\ &= \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-1}} \nabla_{\mathbf{x}_{1:n-1}}^\top & \nabla_{\mathbf{x}_{1:n-1}} \nabla_{\mathbf{x}_n}^\top \\ \nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_{1:n-1}}^\top & \nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_n}^\top \end{bmatrix} \end{aligned} \quad (14.66)$$

Now, using (14.66) in (14.63), we can decompose $\mathbf{J}_{1:n}$

$$\mathbf{J}_{1:n} = \begin{bmatrix} \mathbf{A}_{1:n-1|1:n-1} & \mathbf{B}_{1:n-1|n} \\ \mathbf{B}_{1:n-1|n}^\top & \mathbf{C}_{n|n} \end{bmatrix} \quad (14.67)$$

with

$$\mathbf{A}_{1:n-1|1:n-1} \triangleq -\mathcal{E}\{\nabla_{\mathbf{x}_{1:n-1}} \nabla_{\mathbf{x}_{1:n-1}}^\top \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})\} \quad (14.68)$$

$$\mathbf{B}_{1:n-1|n} \triangleq -\mathcal{E}\{\nabla_{\mathbf{x}_{1:n-1}} \nabla_{\mathbf{x}_n}^\top \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})\} \quad (14.69)$$

$$\mathbf{C}_{n|n} \triangleq -\mathcal{E}\{\nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_n}^\top \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n})\} \quad (14.70)$$

Now, we can calculate $\mathbf{J}_{1:n}^{-1}$ by taking the inverse of the partitioned matrix (14.67) using (2.19) results in

$$\mathbf{J}_{1:n}^{-1} = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{T}_{21} & \mathbf{T}_{22} \end{bmatrix} \quad (14.71)$$

where

$$\mathbf{T}_{11} \triangleq \left(\mathbf{A}_{1:n-1|1:n-1} - \mathbf{B}_{1:n-1|n} \mathbf{C}_{n|n}^{-1} \mathbf{B}_{1:n-1|n}^\top \right)^{-1} \quad (14.72)$$

$$\begin{aligned} \mathbf{T}_{12} &\triangleq -\mathbf{A}_{1:n-1|1:n-1} \mathbf{B}_{1:n-1|n}^{-1} \\ &\times \left(\mathbf{C}_{n|n} - \mathbf{B}_{1:n-1|n}^\top \mathbf{A}_{1:n-1|1:n-1}^{-1} \mathbf{B}_{1:n-1|n} \right)^{-1} \end{aligned} \quad (14.73)$$

$$\mathbf{T}_{21} \triangleq -\mathbf{C}_{n|n}^{-1} \mathbf{B}_{1:n-1|n}^\top \left(\mathbf{A}_{1:n-1|1:n-1} - \mathbf{B}_{1:n-1|n} \mathbf{C}_{n|n}^{-1} \mathbf{B}_{1:n-1|n}^\top \right)^{-1} \quad (14.74)$$

$$\mathbf{T}_{22} \triangleq \left(\mathbf{C}_{n|n} - \mathbf{B}_{1:n-1|n}^\top \mathbf{A}_{1:n-1|1:n-1}^{-1} \mathbf{B}_{1:n-1|n} \right)^{-1} \quad (14.75)$$

The lower right-hand block of $\mathbf{J}_{1:n}^{-1}$ can be identified as \mathbf{J}_n^{-1} that is the Cramer–Rao lower bound for $\hat{\mathbf{x}}_{n|n}$

$$\mathbf{J}_n^{-1} = \left(\mathbf{C}_{n|n} - \mathbf{B}_{1:n-1|n}^\top \mathbf{A}_{1:n-1|1:n-1}^{-1} \mathbf{B}_{1:n-1|n} \right)^{-1} \quad (14.76)$$

It follows immediately that

$$\mathbf{J}_n = \mathbf{C}_{n|n} - \mathbf{B}_{1:n-1|n}^T \mathbf{A}_{1:n-1|1:n-1}^{-1} \mathbf{B}_{1:n-1|n} \quad (14.77)$$

Note that (14.76) requires the inverse of the large matrix $\mathbf{A}_{1:n-1|1:n-1}$. To get around this, we now develop a recursion relationship that generates \mathbf{J}_n from \mathbf{J}_{n-1} that does not require a large matrix inverse. First, consider the joint density found in (14.61). It follows immediately that

$$\begin{aligned} p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n}) &= p(\mathbf{x}_n, \mathbf{z}_n, \mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) \\ &= p(\mathbf{z}_n | \mathbf{x}_n, \mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) p(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) \\ &\quad \times p(\mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) \end{aligned} \quad (14.78)$$

Since \mathbf{z}_n only depends on \mathbf{x}_n , and \mathbf{x}_n only depends on \mathbf{x}_{n-1} , this becomes

$$p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n}) = p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) \quad (14.79)$$

If we now decompose $\nabla_{\mathbf{x}_{1:n}}$ as

$$\begin{aligned} \nabla_{\mathbf{x}_{1:n}} \nabla_{\mathbf{x}_{1:n}}^T &= \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-2}} \\ \nabla_{\mathbf{x}_{n-1}} \\ \nabla_{\mathbf{x}_n} \end{bmatrix} \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-2}}^T & \nabla_{\mathbf{x}_{n-1}}^T & \nabla_{\mathbf{x}_n}^T \end{bmatrix} \\ &= \begin{bmatrix} \nabla_{\mathbf{x}_{1:n-2}} \nabla_{\mathbf{x}_{1:n-2}}^T & \nabla_{\mathbf{x}_{1:n-2}} \nabla_{\mathbf{x}_{n-1}}^T & \nabla_{\mathbf{x}_{1:n-2}} \nabla_{\mathbf{x}_n}^T \\ \nabla_{\mathbf{x}_{n-1}} \nabla_{\mathbf{x}_{1:n-2}}^T & \nabla_{\mathbf{x}_{n-1}} \nabla_{\mathbf{x}_{n-1}}^T & \nabla_{\mathbf{x}_{n-1}} \nabla_{\mathbf{x}_n}^T \\ \nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_{1:n-2}}^T & \nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_{n-1}}^T & \nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_n}^T \end{bmatrix} \end{aligned} \quad (14.80)$$

the trajectory information matrix (14.63) becomes

$$\mathbf{J}_{1:n} = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} & \mathbf{S}_{13} \\ \mathbf{S}_{21} & \mathbf{S}_{22} & \mathbf{S}_{23} \\ \mathbf{S}_{31} & \mathbf{S}_{32} & \mathbf{S}_{33} \end{bmatrix} \quad (14.81)$$

Here

$$\begin{aligned} \mathbf{S}_{11} &= -\mathcal{E}\left\{\nabla_{\mathbf{x}_{1:n-2}} \left[\nabla_{\mathbf{x}_{1:n-2}}^T \ln p(\mathbf{x}_{1:n}, \mathbf{z}_{1:n}) \right] \right\} \\ &= -\mathcal{E}\left\{\nabla_{\mathbf{x}_{1:n-2}} \left[\nabla_{\mathbf{x}_{1:n-2}}^T \left[\ln p(\mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) \right. \right. \right. \\ &\quad \left. \left. + \ln p(\mathbf{z}_n | \mathbf{x}_n) + \ln p(\mathbf{x}_n | \mathbf{x}_{n-1}) \right] \right] \right\} \\ &= -\mathcal{E}\left\{\nabla_{\mathbf{x}_{1:n-2}} \left[\nabla_{\mathbf{x}_{1:n-2}}^T \ln p(\mathbf{x}_{1:n-1}, \mathbf{z}_{1:n-1}) \right] \right\} \\ &= \mathbf{A}_{1:n-2|1:n-2} \end{aligned} \quad (14.82)$$

Similarly

$$\mathbf{S}_{22} = \mathbf{C}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}^{11} \quad (14.83)$$

$$\mathbf{S}_{12} = \mathbf{B}_{1:n-2|n-1} \quad (14.84)$$

$$\mathbf{S}_{31} = \mathbf{0} \quad (14.85)$$

$$\mathbf{S}_{32} = \mathbf{D}_{n|n-1}^{12} \quad (14.86)$$

$$\mathbf{S}_{22} = \mathbf{D}_{n|n}^{22} \quad (14.87)$$

with $\mathbf{A}_{1:n-2|1:n-2}$, $\mathbf{B}_{1:n-2|n-1}$, and $\mathbf{C}_{n-1|n-1}$ defined by (14.68), (14.69), and (14.70), respectively, and

$$\mathbf{D}_{n-1|n-1}^{11} \triangleq -\mathcal{E}\{\nabla_{\mathbf{x}_{n-1}} \nabla_{\mathbf{x}_{n-1}}^\top \ln p(\mathbf{x}_n | \mathbf{x}_{n-1})\} \quad (14.88)$$

$$\mathbf{D}_{n-1|n}^{21} \triangleq -\mathcal{E}\{\nabla_{\mathbf{x}_{n-1}} \nabla_{\mathbf{x}_n}^\top \ln p(\mathbf{x}_n | \mathbf{x}_{n-1})\} \quad (14.89)$$

$$\mathbf{D}_{n|n-1}^{12} = [\mathbf{D}_{n-1|n}^{21}]^\top \quad (14.90)$$

$$\begin{aligned} \mathbf{D}_{n|n}^{22} \triangleq & -\mathcal{E}\{\nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_n}^\top \ln p(\mathbf{x}_n | \mathbf{x}_{n-1})\} \\ & -\mathcal{E}\{\nabla_{\mathbf{x}_n} \nabla_{\mathbf{x}_n}^\top \ln p(\mathbf{z}_n | \mathbf{x}_n)\} \end{aligned} \quad (14.91)$$

We can now write (14.81) as

$$\mathbf{J}_{1:n} = \begin{bmatrix} \mathbf{A}_{1:n-2|1:n-2} & \mathbf{B}_{1:n-2|n-1} & \mathbf{0} \\ \mathbf{B}_{1:n-2|n-1}^\top & \mathbf{C}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}^{11} & \mathbf{D}_{n|n-1}^{12} \\ \mathbf{0} & \mathbf{D}_{n-1|n}^{21} & \mathbf{D}_{n|n}^{22} \end{bmatrix} \quad (14.92)$$

Calculating $\mathbf{J}_{1:n}^{-1}$ by taking the inverse of the partitioned matrix (14.92) using (2.19) and looking at only the inverse of the lower right-hand block results in

$$\begin{aligned} \mathbf{J}_n &= \mathbf{D}_{n|n}^{22} - [\mathbf{0} \quad \mathbf{D}_{n-1|n}^{21}] \\ &\times \begin{bmatrix} \mathbf{A}_{1:n-2|1:n-2} & \mathbf{B}_{1:n-2|n-1} \\ \mathbf{B}_{1:n-2|n-1}^\top & \mathbf{C}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}^{11} \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{D}_{n|n-1}^{12} \end{bmatrix} \\ &= \mathbf{D}_{n|n}^{22} - \mathbf{D}_{n-1|n}^{21} \left[\mathbf{C}_{n-1|n-1} + \mathbf{D}_{n-1|n-1}^{11} \right. \\ &\quad \left. - \mathbf{B}_{1:n-2|n-1}^\top \mathbf{A}_{1:n-2|1:n-2}^{-1} \mathbf{B}_{1:n-2|n-1} \right]^{-1} \\ &\quad \times \mathbf{D}_{n|n-1}^{12} \end{aligned} \quad (14.93)$$

Remembering (14.77), (14.93) becomes the recursion for \mathbf{J}_n

$$\mathbf{J}_n = \mathbf{D}_{n|n}^{22} - \mathbf{D}_{n-1|n}^{21} \left[\mathbf{J}_{n-1} + \mathbf{D}_{n-1|n-1}^{11} \right]^{-1} \mathbf{D}_{n|n-1}^{12} \quad (14.94)$$

This recursion can be initialized from (14.61) by setting

$$\mathbf{J}_0 = \mathcal{E}\{\nabla_{\mathbf{x}_0} \ln p(\mathbf{x}_0) \nabla_{\mathbf{x}_0} \ln p(\mathbf{x}_0)^\top\} \quad (14.95)$$

14.4.4 The Cramer–Rao Lower Bound for Gaussian Additive Noise

Consider the case where the dynamic and observation models are Gaussian, that is,

$$\mathbf{x}_n = \mathbf{f}_{n-1}(\mathbf{x}_n) + \mathbf{v}_{n-1} \quad (14.96)$$

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_{n-1} \quad (14.97)$$

where

$$\mathbf{v}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \quad (14.98)$$

$$\mathbf{w}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \quad (14.99)$$

If the initial distribution is Gaussian, with

$$p(\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0; \hat{\mathbf{x}}_0, \mathbf{P}_0^{\mathbf{xx}}) \quad (14.100)$$

then

$$\begin{aligned} \nabla_{\mathbf{x}_0} \ln p(\mathbf{x}_0) &= \nabla_{\mathbf{x}_0} \left\{ c - \frac{1}{2} \left[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T [\mathbf{P}_0^{\mathbf{xx}}]^{-1} (\mathbf{x}_0 - \hat{\mathbf{x}}_0) \right] \right\} \\ &= -[\mathbf{P}_0^{\mathbf{xx}}]^{-1} (\mathbf{x}_0 - \hat{\mathbf{x}}_0) \end{aligned} \quad (14.101)$$

Now, from (14.95), and noting that $[\mathbf{P}_0^{\mathbf{xx}}]^{-T} = [\mathbf{P}_0^{\mathbf{xx}}]^{-1}$, \mathbf{J}_0 becomes

$$\begin{aligned} \mathbf{J}_0 &= \mathcal{E} \left\{ [\mathbf{P}_0^{\mathbf{xx}}]^{-1} (\mathbf{x}_0 - \hat{\mathbf{x}}_0) (\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T [\mathbf{P}_0^{\mathbf{xx}}]^{-1} \right\} \\ &= [\mathbf{P}_0^{\mathbf{xx}}]^{-1} \mathcal{E} \left\{ (\mathbf{x}_0 - \hat{\mathbf{x}}_0) (\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T \right\} [\mathbf{P}_0^{\mathbf{xx}}]^{-1} \\ &= [\mathbf{P}_0^{\mathbf{xx}}]^{-1} \end{aligned} \quad (14.102)$$

Similarly,

$$\nabla_{\mathbf{x}_{n-1}} \ln p(\mathbf{x}_n | \mathbf{x}_{n-1}) = [\nabla_{\mathbf{x}_{n-1}} \mathbf{f}_{n-1}^T(\mathbf{x}_{n-1})] \mathbf{Q}_{n-1}^{-1} \mathbf{v}_{n-1} \quad (14.103)$$

$$\nabla_{\mathbf{x}_n} \ln p(\mathbf{z}_n | \mathbf{x}_n) = [\nabla_{\mathbf{x}_n} \mathbf{h}_n^T(\mathbf{x}_n)] \mathbf{R}_n^{-1} \mathbf{w}_n \quad (14.104)$$

And (14.88)–(14.91) become

$$\mathbf{D}_{n-1|n-1}^{11} = \mathcal{E} \left\{ \tilde{\mathbf{F}}_{n-1}^T \mathbf{Q}_{n-1}^{-1} \tilde{\mathbf{F}}_{n-1} \right\} \quad (14.105)$$

$$\mathbf{D}_{n-1|n}^{12} = -\mathcal{E} \left\{ \tilde{\mathbf{F}}_{n-1}^T \right\} \mathbf{Q}_{n-1}^{-1} \quad (14.106)$$

$$\mathbf{D}_{n|n-1}^{21} = -\mathbf{Q}_{n-1}^{-1} \mathcal{E} \left\{ \tilde{\mathbf{F}}_{n-1} \right\} \quad (14.107)$$

$$\mathbf{D}_{n|n}^{22} = \mathbf{Q}_{n-1}^{-1} + \mathcal{E} \left\{ \tilde{\mathbf{H}}_n^T \mathbf{R}_n^{-1} \tilde{\mathbf{H}}_n \right\} \quad (14.108)$$

where

$$\tilde{\mathbf{F}}_{n-1} \triangleq [\nabla_{\mathbf{x}_{n-1}} \mathbf{f}_{n-1}^T(\mathbf{x}_{n-1})]^T_{\mathbf{x}_{n-1}(\text{TRUE})} \quad (14.109)$$

$$\tilde{\mathbf{H}}_n \triangleq [\nabla_{\mathbf{x}_n} \mathbf{h}_n^T(\mathbf{x}_n)]^T_{\mathbf{x}_n(\text{TRUE})} \quad (14.110)$$

In summary, for nonlinear models with Gaussian noise, the CRLB can be computed recursively from (14.95) and (14.94) with (14.105)–(14.108) replacing (14.88)–(14.91).

Two problems can arise in the computation of the CRLB. The first is the calculation of the $\mathcal{E}\{\bullet\}$ terms in (14.105)–(14.108). For the Gaussian case, one can use a Monte Carlo method to estimate these expectation terms, since $\tilde{\mathbf{F}}_{n-1}$ and $\tilde{\mathbf{H}}_n$ are evaluated at the trajectory points \mathbf{x}_{n-1} and \mathbf{x}_n , respectively. By using (14.96), where the noise is sampled from $\mathbf{v}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, one can generate M sets of Monte Carlo trajectories values of $\{\mathbf{x}_{m,n}; n = 1, \dots, N; m = 1, \dots, M\}$. The expectation values can be estimated by averaging across the Monte Carlo trajectories at each time point.

The second problem occurs when \mathbf{Q}_n is singular and an inverse does not exist. Methods for handling this special case is beyond the scope of this book, but can be found in Ref. [4] (which can also be found starting on page 686 of Ref. [2]).

14.4.5 The Gaussian Cramer–Rao Lower Bound with Zero Process Noise

When the process noise is very small, we can evaluate the CRLB assuming zero process noise. When the process noise in (14.96) is zero, the state vector evolution is completely deterministic. Therefore, the expectation operation in (14.105)–(14.108) can be ignored. Now, (14.94) becomes

$$\mathbf{J}_n = \mathbf{Q}_{n-1}^{-1} + \tilde{\mathbf{H}}_n^\top \mathbf{R}_n^{-1} \tilde{\mathbf{H}}_n - \mathbf{Q}_{n-1}^{-1} \tilde{\mathbf{F}}_{n-1} \left[\mathbf{J}_{n-1} + \tilde{\mathbf{F}}_{n-1}^\top \mathbf{Q}_{n-1}^{-1} \tilde{\mathbf{F}}_{n-1} \right]^{-1} \tilde{\mathbf{F}}_{n-1}^\top \mathbf{Q}_{n-1}^{-1} \quad (14.111)$$

Using the matrix inversion lemma (2.13), this reduces to

$$\mathbf{J}_n = \left(\mathbf{Q}_{n-1} + \tilde{\mathbf{F}}_{n-1} \mathbf{J}_{n-1}^{-1} \tilde{\mathbf{F}}_{n-1}^\top \right)^{-1} + \tilde{\mathbf{H}}_n^\top \mathbf{R}_n^{-1} \tilde{\mathbf{H}}_n \quad (14.112)$$

Since the process noise is zero, it follows that

$$\mathbf{J}_n = \left(\tilde{\mathbf{F}}_{n-1}^{-1} \right)^\top \mathbf{J}_{n-1} \tilde{\mathbf{F}}_{n-1}^{-1} + \tilde{\mathbf{H}}_n^\top \mathbf{R}_n^{-1} \tilde{\mathbf{H}}_n \quad (14.113)$$

By looking at the inverse \mathbf{J}_n^{-1} and using the matrix inversion lemma, it can be shown that (14.113) reduces to a form identical to the Kalman filter covariance update equation (5.30), with the Jacobians evaluated at the true value of \mathbf{x}_n . Thus, in the absence of process noise, the CRLB recursion for nonlinear filtering is identical to the covariance update equations of the EKF, with the Jacobians evaluated at the true state vector \mathbf{x}_n .

14.4.6 The Gaussian Cramer–Rao Lower Bound with Linear Models

Consider the case where (14.96) and (14.97) are replace by the linear equations

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{v}_{n-1} \quad (14.114)$$

$$\mathbf{z}_n = \mathbf{H}\mathbf{x}_n + \mathbf{w}_{n-1} \quad (14.115)$$

The Jacobians (14.109) and (14.110) now become

$$\tilde{\mathbf{F}}_{n-1} \triangleq [\nabla_{\mathbf{x}_{n-1}} \mathbf{x}_n^\top \mathbf{F}^\top]_{\mathbf{x}_{n-1}(\text{TRUE})}^\top = \mathbf{F} \quad (14.116)$$

$$\tilde{\mathbf{H}}_n \triangleq [\nabla_{\mathbf{x}_n} \mathbf{x}_n^\top \mathbf{H}^\top]_{\mathbf{x}_n(\text{TRUE})}^\top = \mathbf{H} \quad (14.117)$$

Thus, in the *linear Gaussian* case, the Jacobians are independent of the target state and the expectations in (14.105)–(14.108) can be dropped yielding

$$\mathbf{D}_{n-1|n-1}^{11} = \mathbf{F}^\top \mathbf{Q}_{n-1}^{-1} \mathbf{F} \quad (14.118)$$

$$\mathbf{D}_{n-1|n-1}^{12} = -\mathbf{F}^\top \mathbf{Q}_{n-1}^{-1} \quad (14.119)$$

$$\mathbf{D}_{n-1|n-1}^{21} = -\mathbf{Q}_{n-1}^{-1} \quad (14.120)$$

$$\mathbf{D}_{n-1|n-1}^{11} = \mathbf{Q}_{n-1}^{-1} + \mathbf{H}^\top \mathbf{R}_n^{-1} \mathbf{H} \quad (14.121)$$

Now, using these in (14.94) and using the block matrix inversion lemma from Section 2.1.4, (14.94) reduces to

$$\mathbf{J}_n^{-1} = \mathbf{P}_{n|n}^{\mathbf{xx}} \quad (14.122)$$

Thus, for the Gaussian linear case, the CRLB reduces to the update state covariance matrix of the linear Kalman filter, indicating that the linear Kalman filter is the optimal filter for this case.

14.5 PERFORMANCE OF KALMAN CLASS DIFAR TRACK ESTIMATORS

The track estimators used for the DIFAR case study include the EKF, UKF, SSKF, GHKF, and MCKF. In this section, we compare the DIFAR tracking x -axis and y -axis RMS errors for each track estimator. Figures 14.3–14.7 show the RMS x -axis and y -axis RMS errors for the five different track algorithms, along with the CRLB, for signal SNRs from 20 dB down to 0 dB in increments of 5 dB. One can immediately see from these figures that there is very little difference among the tracker algorithms when applied to the DIFAR problem at all SNRs. The CRLB is the gray line across the bottom of each figure. Based on these performance measures, for the DIFAR problem, one should use the EKF since it minimizes the computational burden. It is readily apparent from these figures that for all SNRs the filters have their best performance when the target ship enters the buoy field near the $(0, 0)$ position. But when the target is far from the buoy field, track performance degradation increases with range from the buoy field center and decreasing SNR. In addition, at an SNR of 5 dB or below, the SSKF tracker became unstable. At an SNR of 0 dB, the MCKF also became unstable. These cases are not included in the RMS plots.

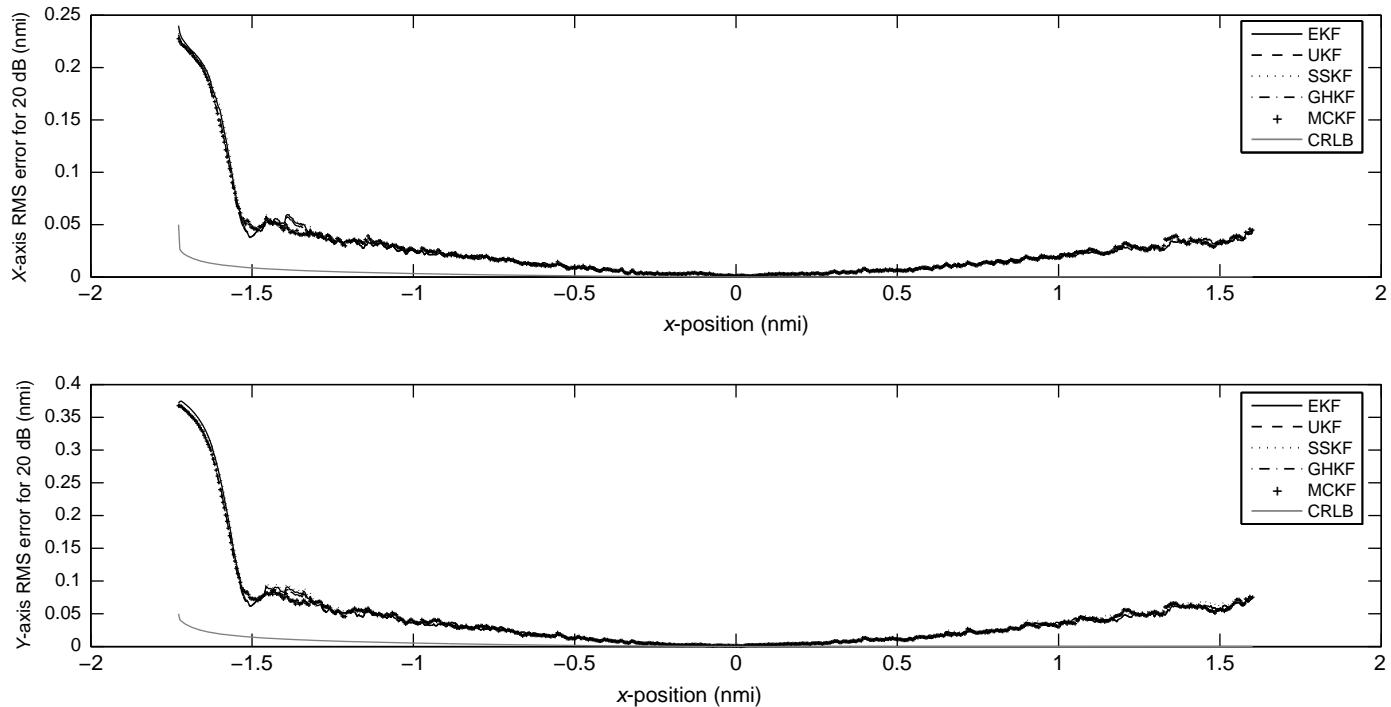


FIGURE 14.3 Comparison of the RMS errors for five different track estimation algorithms with the signal SNR at 20 dB.

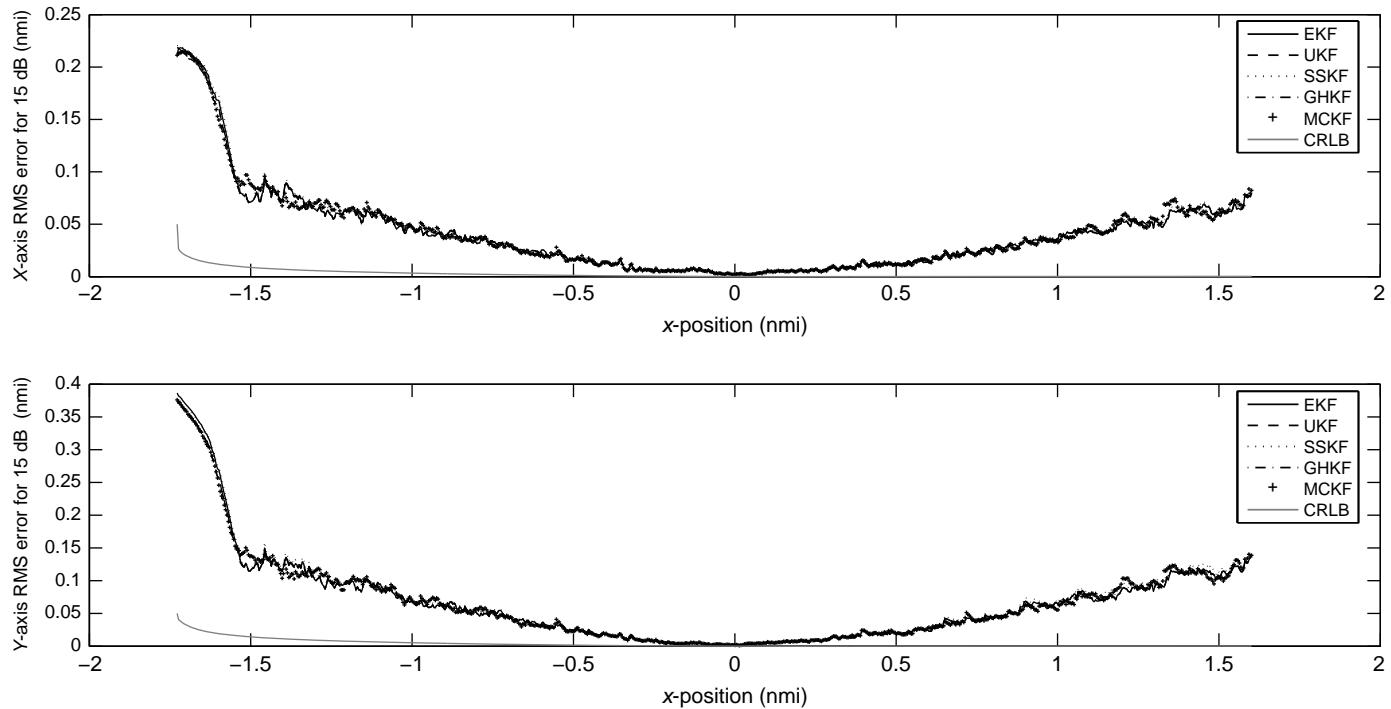


FIGURE 14.4 Comparison of the RMS errors for five different track estimation algorithms with the signal SNR at 15 dB.

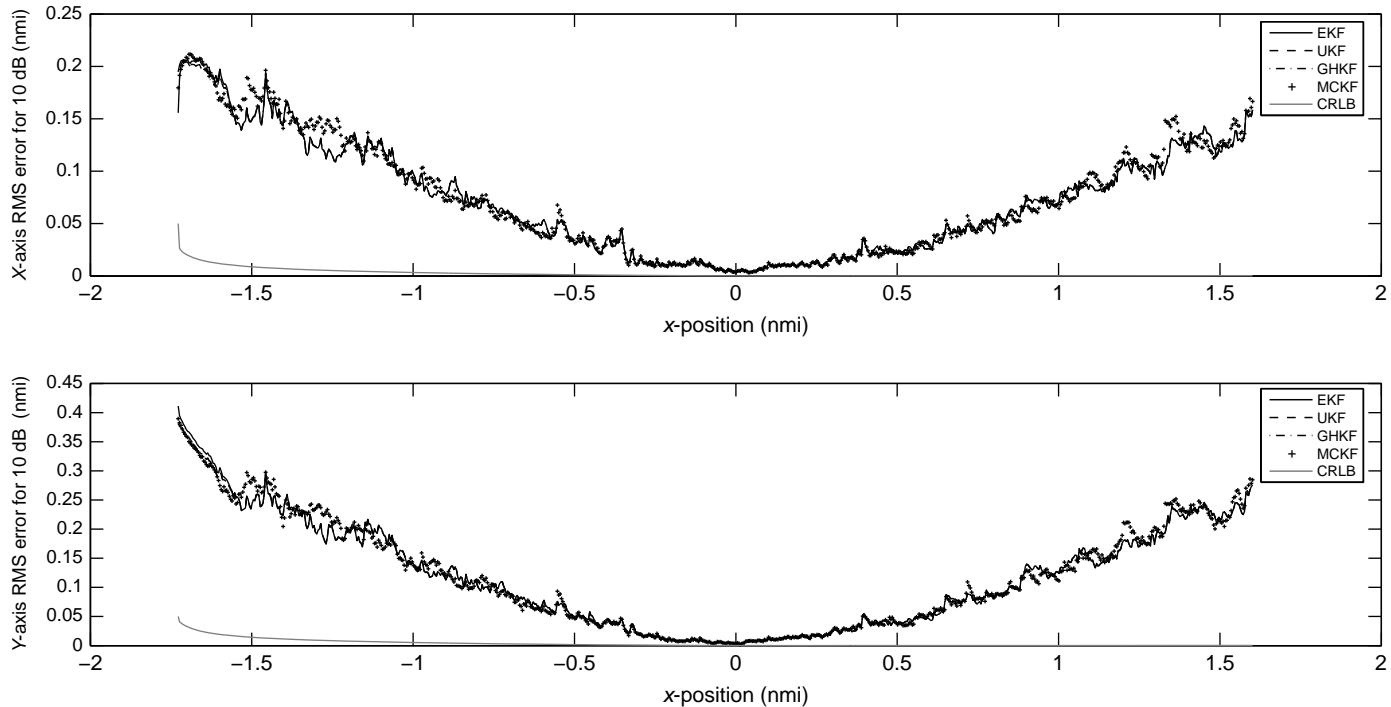


FIGURE 14.5 Comparison of the RMS errors for five different track estimation algorithms with the signal SNR at 10 dB.

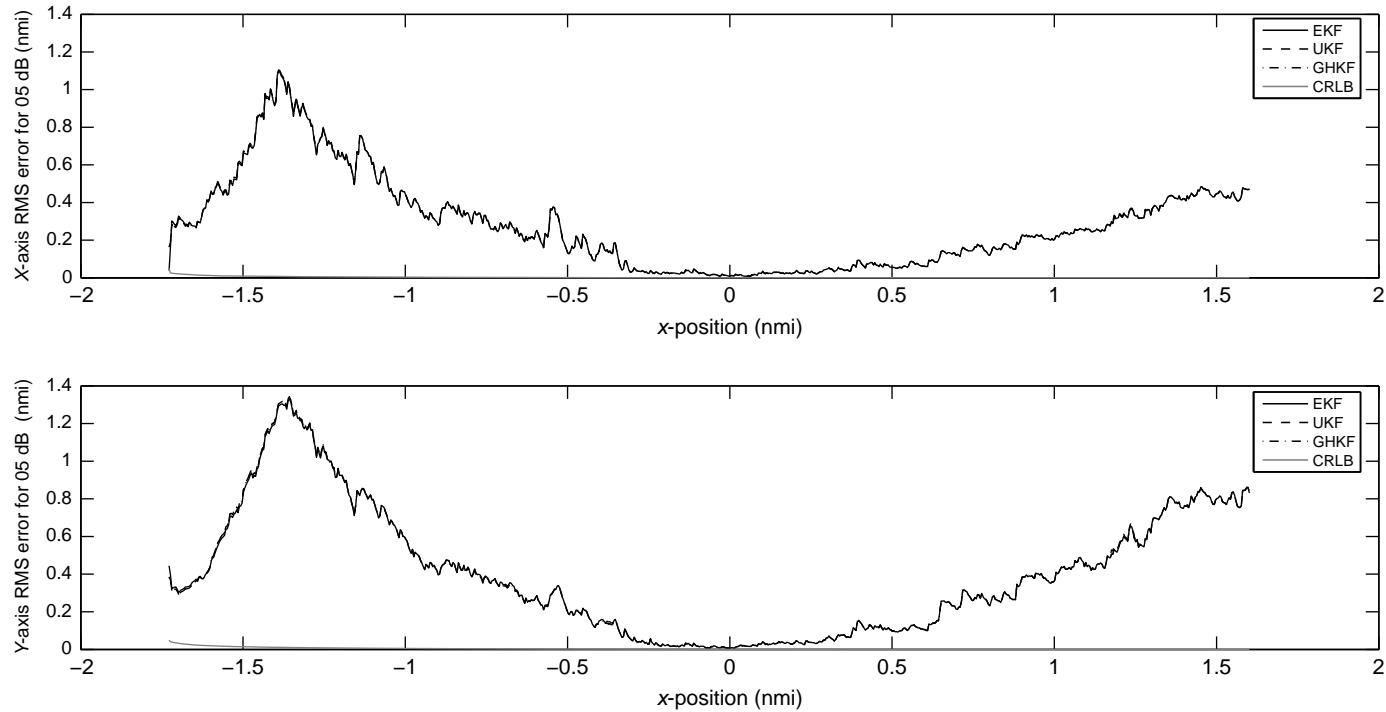


FIGURE 14.6 Comparison of the RMS errors for five different track estimation algorithms with the signal SNR at 5 dB.

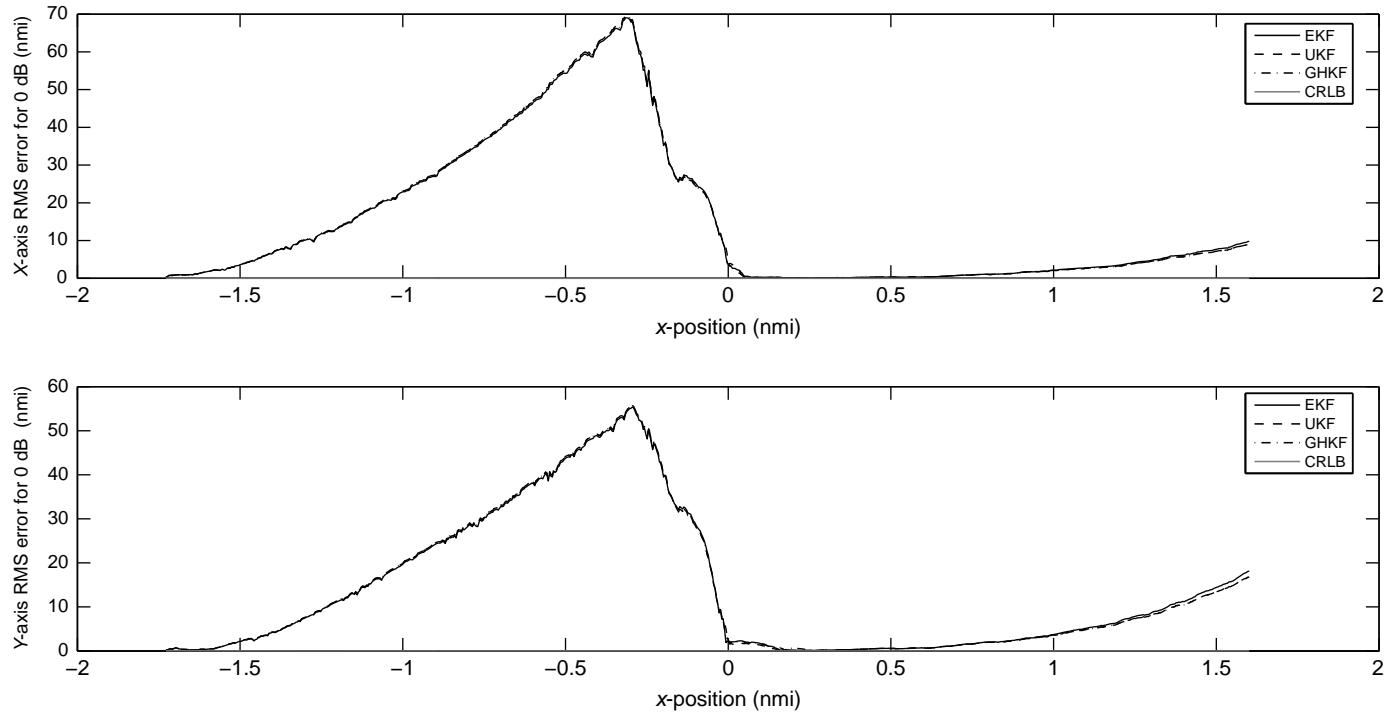


FIGURE 14.7 Comparison of the RMS errors for five different track estimation algorithms with the signal SNR at 0 dB.

REFERENCES

1. Van Trees HL. *Detection, Estimation, and Modulation theory, Part I: Detection, Estimation, and Linear Modulation Theory*. Wiley; 1968.
2. Van Trees HL, Bell KL, editors. *Bayesian Bounds for Parameter Estimation and Nonlinear Filtering/Tracking*. Wiley Interscience; 2007.
3. Ristic B, Arulampalam S, Gordon N. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House; 2004.
4. Tichavský P, Muavchik CH, Nehorai A. Posterior Cramér–Rao bounds for discrete-time nonlinear filtering. *IEEE Trans. Signal Process.* 1998;46(5):1386–1396.

PART III

MONTE CARLO METHODS

15

INTRODUCTION TO MONTE CARLO METHODS

In Part II, we examined numerical methods for evaluating Gaussian-weighted moment integrals containing nonlinear functions. Most of the methods that we investigated incorporate approximations of the nonlinear function using some form of expansion in a multidimensional polynomial. The Gaussian moment integrals were then evaluated at a variety of sets of deterministic vector points resulting in the family of sigma point Kalman filters. In Chapter 12, an alternate method was presented where, instead of making an approximation of the nonlinear function, the multidimensional Gaussian density was approximated by a set of discrete random samples, allowing the moment integrals to be evaluated as a weighted sum of the nonlinear function evaluated at the sample points. As the number of sample points increased, the approximation becomes more exact. It is possible to apply this Monte Carlo sampling method directly for some density functions that have an analytical form, such as Student, Rayleigh, and chi-squared. But the question that should immediately come to mind is whether or not such Monte Carlo integration methods can be applied when the density is unknown or hard to sample from.

In this chapter, we first consider a method for estimating the form of a density based solely on a set of random samples from that density. This, of course, begs the question of how those samples were obtained in the first place. The method presented consists of the generation of multidimensional histograms.

This is followed immediately by the development of a Kernel density approximation of the histogram, providing a smoother estimation of the density function. The usefulness of these Kernel approximations becomes obvious when used for

visualization of multimodal or highly non-Gaussian densities and estimation of the moments does not provide any meaningful information. See Richardson et al. [1] for an example where the density is multimodal in two dimensions and the moments are not representative of the peaks of any of the modes and would therefore give a false impression of the probable location of the target of interest. We will also find Kernel density approximations useful when we discuss resampling methods in the next chapter. After the discussion on kernel density estimation methods we return to the question of how to generate samples from an unknown density by introducing the concept of importance sampling. This chapter is concluded with a summary.

15.1 APPROXIMATING A DENSITY FROM A SET OF MONTE CARLO SAMPLES

15.1.1 Generating Samples from a Two-Dimensional Gaussian Mixture Density

In this section, we discuss several methods for estimating the form of a density based on only a set of samples drawn from that density. It is often the case that we must make inferences about an unknown density from a set of samples (data) from that density, so we must adopt a nonparametric approach. Such nonparametric density estimation methods can provide visual cues that reveal skewness in distributions or the presence of multiple modes in the data and may provide the basis for important physical interpretations of the observations [2]. By way of example, consider the dual mode Gaussian mixture density

$$p(\mathbf{x}) = w_1 \mathcal{N}(\mathbf{x}_1; \hat{\mathbf{x}}_1, \Sigma_1) + w_2 \mathcal{N}(\mathbf{x}_2; \hat{\mathbf{x}}_2, \Sigma_2) \quad (15.1)$$

where $w_1 + w_2 = 1$. Samples can be easily drawn from such a mixture density. If the total number of desired samples is N , then draw $w_1 N$ samples from $\mathcal{N}(\mathbf{x}_1; \hat{\mathbf{x}}_1, \Sigma_1)$ and $w_2 N$ samples from $\mathcal{N}(\mathbf{x}_2; \hat{\mathbf{x}}_2, \Sigma_2)$. These samples for $N = 1000$ and $(w_1, w_2) = (0.3, 0.7)$ are shown in Figure 15.1. We have also set

$$\begin{aligned}\hat{\mathbf{x}}_1 &= (-5, -8) \\ \hat{\mathbf{x}}_2 &= (7, 7) \\ \Sigma_1 &= \begin{bmatrix} 100 & -5 \\ -5 & 1 \end{bmatrix} \\ \Sigma_2 &= \begin{bmatrix} 4 & 8 \\ 8 & 64 \end{bmatrix}\end{aligned}$$

15.1.2 Approximating a Density by Its Multidimensional Histogram

Since we are interested in developing a visualization of the probability density function, it is important to note that the probability of a sample falling within a given area

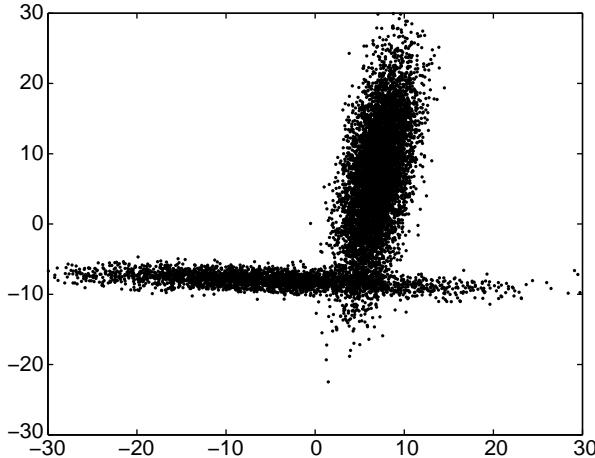


FIGURE 15.1 Samples drawn from a two-dimensional Gaussian mixture density.

is given by the normalized density of the samples that fall within that area. This leads to a normalized two-dimensional histogram as the first estimation and visualization method considered because it actually measures the probability of observing a sample in a particular two-dimensional area interval. A multidimensional histogram density estimate can be defined by

$$\hat{p}(\mathbf{x}) = \frac{v_{ik}}{NA_{ik}} \quad (15.2)$$

where v_{ik} is the number of samples falling in the hypervolume bin A_{ik} . For example, for a two-dimensional density, $A_{ik} = (x_{i+1} - x_i)(y_{k+1} - y_k)$. If all of the bins are the same size, we can let $h = (x_{i+1} - x_i) = (y_{k+1} - y_k)$, $\forall i, k$ and

$$\hat{p}(\mathbf{x}) = \frac{v_{ik}}{Nh^2} \quad (15.3)$$

Thus, for a general n_x -dimensional histogram

$$\hat{p}(\mathbf{x}) = \frac{v_i}{Nh^{n_x}} \quad (15.4)$$

with v_i the number of samples falling into the i th hypervolume bin. This latter histogram has only one free parameter, the bin width h . The histogram can asymptotically approximate any continuous density and hence can be called a nonparametric estimate. The selection of h , called the bandwidth, has been the emphasis of much recent research that will not be presented here. See Scott [3] and the references contained therein. Scott shows how to use the integrated mean squared error (IMSE) and the mean integrated squared error (MISE) to estimate the optimal bandwidth based on the sample data.

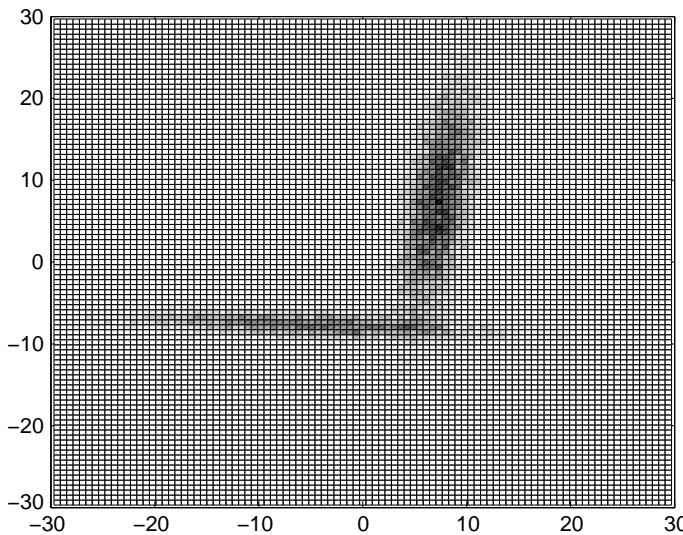


FIGURE 15.2 Two-dimensional histogram based on samples from a Gaussian mixture distribution.

A set of Matlab routines that generates and plots a two-dimensional histogram can be downloaded from Matlab Central [4]. Care must be taken with these subroutines. They must be adapted to the specific data set and they are also unnormalized as written. The two-dimensional normalized histogram generated from the data in Figure 15.1 is shown in Figure 15.2. The bandwidth was selected to produce 99 bins in each dimension based on the maximum range of the data.

15.1.3 Kernel Density Approximation

An estimate of the probability of a sample falling within a given two-dimensional bin centered at $\mathbf{x} \in \mathbb{R}^2$ is given by (15.3). In the limit, as the bin area goes to zero, we reach a point where each bin contains, at most, one sample. In this limit, we can write the sample density function as

$$p_s(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \delta(\mathbf{x} - \mathbf{x}^{(i)}) \quad (15.5)$$

This can be seen as a density for discrete samples because

$$\int_{\mathbb{R}^{n_x}} p_s(\mathbf{x}) d\mathbf{x} = \frac{1}{N} \sum_{i=1}^N \int_{\mathbb{R}^{n_x}} \delta(\mathbf{x} - \mathbf{x}^{(i)}) d\mathbf{x} = 1 \quad (15.6)$$

The histogram shown in Figure 15.2 is blocky and has discontinuities at each bin boundary. In addition, its derivatives are nonzero only at the bin boundaries and are

zero everywhere else. This difficulty can be overcome by replacing the histogram estimator with an estimator that convolves $p_s(\mathbf{x})$ with a smoothing scaled kernel density function, $\mathcal{K}_h(x)$. That is, let

$$\begin{aligned}\hat{p}(\mathbf{x}) &= (p_s * \mathcal{K}_h)(\mathbf{x}) \\ &\triangleq \int_{\mathbb{R}^{n_x}} p_s(\mathbf{u}) \mathcal{K}_{\mathbf{H}}(\mathbf{x} - \mathbf{u}) d\mathbf{u} \\ &= \int_{\mathbb{R}^{n_x}} \frac{1}{N} \sum_{i=1}^N \delta(\mathbf{u} - \mathbf{x}^{(i)}) \mathcal{K}_{\mathbf{H}}(\mathbf{x} - \mathbf{u}) d\mathbf{u} \\ &= \frac{1}{N} \sum_{i=1}^N \mathcal{K}_{\mathbf{H}}(\mathbf{x} - \mathbf{x}^{(i)})\end{aligned}\tag{15.7}$$

where the scaled kernel is defined in terms of the kernel by

$$\mathcal{K}_{\mathbf{H}}(\mathbf{t}) \triangleq \frac{1}{|\mathbf{H}|^{1/2}} \mathcal{K}(\mathbf{H}^{-1/2} \mathbf{t})\tag{15.8}$$

Here $\mathcal{K}(\mathbf{u})$ is a continuous symmetric Kernel function that satisfies the conditions

$$\int_{\mathbb{R}^n} \mathcal{K}(\mathbf{u}) d\mathbf{u} = 1\tag{15.9}$$

$$\int_{\mathbb{R}^n} \mathbf{u}^j \mathcal{K}(\mathbf{u}) d\mathbf{u} = 0, \quad j = 1, \dots, k-1\tag{15.10}$$

$$\int_{\mathbb{R}^n} \mathbf{u}^k \mathcal{K}(\mathbf{u}) d\mathbf{u} \neq 0\tag{15.11}$$

$$\mathcal{K}(\mathbf{u}) > 0, \quad \forall \mathbf{u}\tag{15.12}$$

where k is the order of the kernel function. The components of the matrix \mathbf{H} are the bandwidth (or smoothing) parameters of the kernel and \mathbf{H} functions in a manner similar to a covariance matrix. Two very good introductions to kernel density estimation can be found in the monograms by Fukunaga [5] and Silverman [6].

15.1.3.1 Univariate Kernel Density Estimation. Consider first the univariate case. A variety of univariate kernel function have been considered in the literature [7] and some of the most popular are listed in Table 15.1. In Table 15.1, the indicator function I is defined by

$$I(|u| \leq 1) = \begin{cases} 1, & \text{for } -1 \leq u \leq 1 \\ 0, & \text{elsewhere} \end{cases}\tag{15.13}$$

To illustrate the use of a kernel function in one-dimensional density estimation, consider the small set of 12 one-dimensional data points

TABLE 15.1 Common Univariate Kernel Functions of Order 2

Kernel	$\mathcal{K}(u)$
Uniform	$\frac{1}{2} I(u \leq 1)$
Triangle	$(1 - u) I(u \leq 1)$
Epanechnikov	$\frac{3}{4} (1 - u^2) I(u \leq 1)$
Quartic	$\frac{15}{16} (1 - u^2)^2 I(u \leq 1)$
Triweight	$\frac{35}{32} (1 - u^2)^3 I(u \leq 1)$
Cosine	$-\frac{\pi}{4} \cos\left(\frac{\pi}{2}u\right) I(u \leq 1)$
Gaussian	$\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}u^2\right)$

$\{-7.5, -6, -3, -2, -1.5, -1, -0.3, 0, 3.5, 4, 6, 8\}$. Let the number of original data points be labeled N_{old} . Using the Gaussian kernel from Table 15.1 in (15.7) results in the kernel density estimate

$$\begin{aligned}\hat{p}(x) &= \frac{1}{N} \sum_{i=1}^N \mathcal{K}_h(x - x^{(i)}) \\ &= \sum_{i=1}^N \frac{1}{Nh} \mathcal{K}\left(\frac{x - x^{(i)}}{h}\right) \\ &= \sum_{i=1}^N \frac{1}{N} \frac{1}{\sqrt{2\pi}h} \exp\left(-\frac{1}{2} \left[\frac{x - x^{(i)}}{h}\right]^2\right)\end{aligned}\quad (15.14)$$

We can see immediately that the bandwidth parameter h is just the standard deviation of the Gaussian density. We can also immediately see that the density estimate $\hat{p}(x)$ is nothing more than a Gaussian mixture density with equal weights for each individual Gaussian density centered around one of the original sample points.

Define the *individual* sample points kernels as

$$K_i(x) = \frac{1}{N} \frac{1}{\sqrt{2\pi}h} \exp\left(-\frac{1}{2} \left[\frac{x - x^{(i)}}{h}\right]^2\right) = \frac{1}{N} \mathcal{N}(x; x^{(i)}, h^2) \quad (15.15)$$

Both the full estimated kernel density and the individual kernels for two different values of h are plotted in Figure 15.3 over the range of x within the limits $-15 \leq x \leq 15$.

It is obvious from Figure 15.3 that the value chosen for h is very important. In the first plot, when $h = 0.9$, the density is smooth and produces a good approximation of the density of the samples, while in the second plot for $h = 0.3$, the density estimate is much too fractured with peaks at almost every sample. It has been commented that the choice of h is much more important than the choice of \mathcal{K} [7].

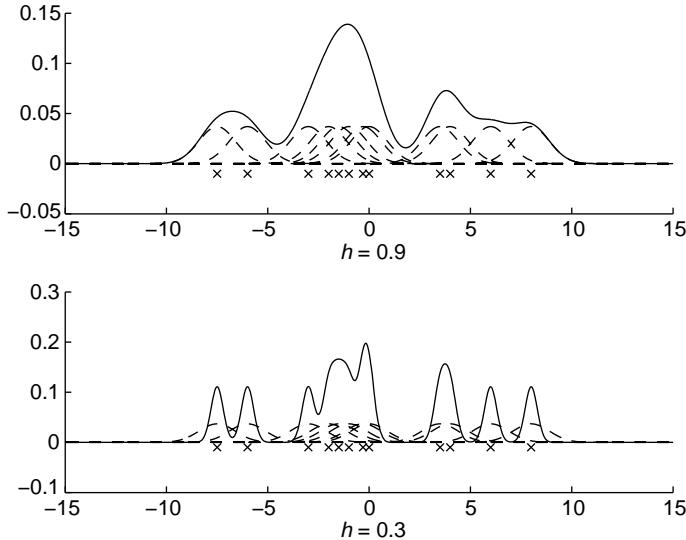


FIGURE 15.3 Gaussian kernel density estimate for sparse sample data.

Now, assume that the original sample set was drawn from a Gaussian distribution (even though it may be far from the truth). That is, assume that the true density can be approximated by $\mathcal{N}(x; x_s, \sigma_s^2)$, where x_s and σ_s^2 are the sample mean and variance. If we make the transformation

$$\tilde{x} = \frac{x}{\sigma_s} \quad (15.16)$$

then it follows immediately that

$$\begin{aligned} \mathcal{N}(x; x_s, \sigma_s^2) &= \frac{1}{\sigma_s} \mathcal{N}(\tilde{x}; \tilde{x}_s, 1) \\ &= \frac{1}{\sigma_s} \mathcal{N}\left(\frac{x}{\sigma_s}; \frac{x_s}{\sigma_s}, 1\right) \end{aligned} \quad (15.17)$$

Now, (15.15) becomes

$$\begin{aligned} K_i(\tilde{x}) &= \frac{1}{N\sigma_s} \mathcal{N}\left(x; x^{(i)}, \sigma_s^2 h^2\right) \\ &= \frac{1}{N} \frac{1}{\sqrt{2\pi} h \sigma_s} \exp\left(-\frac{1}{2} \left[\frac{x - x^{(i)}}{\sigma_s h}\right]^2\right) \end{aligned} \quad (15.18)$$

Applying the affine transformation

$$\tilde{x} = x^{(i)} + \sigma_s h c \quad (15.19)$$

TABLE 15.2 One-Dimensional Kernel Sample Generation

1. Subdivide the interval $(0, 1]$ into N_{old} increments	$q_i = (i - 1)/N_{\text{old}}; i = 1, \dots, N_{\text{old}} + 1.$
2. Generate N_{new} sample from a uniform distribution	$u_j \sim \mathcal{U}(0, 1], j = 1, \dots, N_{\text{new}}$
3. For u_j , determine its appropriate q_i interval	Find i such that $q_i < u_j \leq q_{i+1}$
4. Generate sample for c	$c^{(j)} \sim \mathcal{N}(c; 0, 1)$
5. Generate new sample of x	$\tilde{x}^{(j)} = x^{(i)} + \sigma_s h c^{(j)}$
6. Repeat Steps 2 - 5 for $j = 1, \dots, N_{\text{new}}$	

modifies (15.18) so that

$$K_i(\tilde{x}) = \frac{1}{N} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}c^2\right) \quad (15.20)$$

and (15.14) reduces to

$$\hat{p}(\tilde{x}) = \sum_{i=1}^N \frac{1}{N} \mathcal{N}(c; 0, 1) \quad (15.21)$$

Now, a new set of samples $\{x^{(j)}, j = 1, \dots, N_{\text{new}}\}$, where N_{new} can be much greater than N_{old} , can be drawn from $\hat{p}(\tilde{x})$ using the procedure shown in Table 15.2.

This procedure (called regularization) is so simple only because we are taking advantage of the fact that the kernel estimation method results in a mixture density with equal weights! As we will show below, it can be generalized to include the multivariate case with unequal weights. It is important to remember that in areas where there is a high density of samples, there will be more resampled values chosen.

It should be noted that resampling from a kernel density estimate, when kernels other than the Gaussian are used, is sometimes much more difficult. For example, generating samples from an Epanechnikov kernel consists of generating $\sqrt{\beta}u$, where β follows a beta distribution with parameters $(1/2, 2)$ and u is a sample from the uniform distribution as shown in step 2 above [8].

Further discussion of multidimensional kernel methods can be found in Refs [9–12].

15.1.3.2 Multivariate Kernel Density Estimation. For a multivariate Gaussian kernel, (15.18) results in the density estimator

$$\hat{p}(\mathbf{x}) = \sum_{i=1}^N \frac{1}{N} \frac{1}{(2\pi)^{n_x/2} |\Sigma\mathbf{H}|^{1/2}} \exp\left\{-\frac{1}{2} (\mathbf{x} - \mathbf{x}^{(i)})^\top [\Sigma\mathbf{H}]^{-1} (\mathbf{x} - \mathbf{x}^{(i)})\right\} \quad (15.22)$$

Thus, the multivariate individual kernels are obviously the Gaussian densities $\mathcal{N}(\mathbf{x}; \mathbf{x}^{(i)}, \Sigma_{\mathbf{H}})$ divided by N .

Consider the two-dimensional case $n_x = 2$, where

$$\mathbf{H} = \begin{bmatrix} h_1^2 & h_{12} \\ h_{12} & h_2^2 \end{bmatrix} \quad (15.23)$$

For this case we can have three different smoothing parameters, one for dimension 1, one for dimension 2, and one for the correlation between the two dimensions. But this makes choosing the “best” smoothing parameters more complex than in the one-dimensional case. If we assume that the underlying density of the data is Gaussian, that is, $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \Sigma)$, we can generate a sample covariance matrix Σ_s from the data. Making the matrix decomposition $\Sigma_s = \mathbf{D}_s \mathbf{D}_s^\top$, we can normalize the data using the transformation

$$\tilde{\mathbf{x}} = \mathbf{D}_s^{-1} \mathbf{x} \quad (15.24)$$

and it follows immediately that

$$\tilde{\mathbf{x}} \sim |\mathbf{D}_s|^{-1} \mathcal{N}\left(\mathbf{D}_s^{-1} \mathbf{x}; \mathbf{D}_s^{-1} \mathbf{x}_s, \mathbf{I}\right) \quad (15.25)$$

Since the data has now been transformed so that it has a unit diagonal sample covariance matrix, the off-diagonal terms of \mathbf{H} can be set to zero and we can set $h = h_1 = h_2$ and thus $|\mathbf{H}|^{1/2} = h^2$. Thus, for this two-dimensional case, (15.22) becomes

$$\begin{aligned} \hat{p}(\tilde{\mathbf{x}}) &= \sum_{i=1}^N \frac{1}{N} \frac{1}{2\pi |\mathbf{D}_s| h^2} \exp \left\{ -\frac{1}{2} \left(\mathbf{D}_s^{-1} \mathbf{x} - \mathbf{D}_s^{-1} \mathbf{x}^{(i)} \right)^\top \mathbf{H}^{-1} \right. \\ &\quad \times \left. \left(\mathbf{D}_s^{-1} \mathbf{x} - \mathbf{D}_s^{-1} \mathbf{x}^{(i)} \right) \right\} \end{aligned} \quad (15.26)$$

If a final affine transformation is made such that

$$\mathbf{x} = \mathbf{x}^{(i)} + \mathbf{D}_s h \mathbf{c} \quad (15.27)$$

then (15.26) reduces to

$$\hat{p}(\tilde{\mathbf{x}}) = \sum_{i=1}^N \frac{1}{N} \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) \quad (15.28)$$

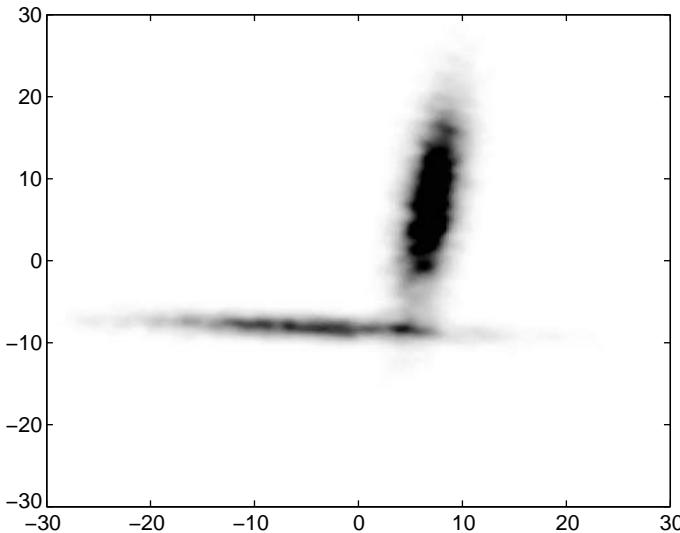
It is simple to extend the above to more than two dimensions and will be left as an exercise for the reader.

A two-dimensional Gaussian Kernel density estimator subroutine can be downloaded from Matlab Central [12]. It was modified and used to generate a smoothed density estimation from the data contained in Figure 15.1. The resulting visualization of the estimated pdf is shown in Figure 15.4.

In a manner similar to the one-dimensional case, samples from the multidimensional Gaussian kernel are easy to obtain using the procedure shown in Table 15.3.

TABLE 15.3 Multidimensional Kernel Sample Generation

1. Subdivide the interval $(0, 1]$ into N_{old} increments	$q_i = (i - 1)/N_{\text{old}}; i = 1, \dots, N_{\text{old}} + 1$
2. Generate a sample from a uniform distribution	$u_j \sim \mathcal{U}(0, 1], j = 1, \dots, N_{\text{new}}$
3. For u_j , determine its appropriate q_i interval	Find i such that $q_i < u_j \leq q_{i+1}$ Select $\mathbf{x}^{(i)}$
4. Generate sample for \mathbf{c}	$\mathbf{c}^{(j)} \sim \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I})$
5. Generate new sample of \mathbf{x}	$\mathbf{x}^{(j)} = \mathbf{x}^{(i)} + \mathbf{D}_s h \mathbf{c}^{(j)}$
6. Repeat Steps 2 - 5 for $j = 1, \dots, N_{\text{new}}$	

**FIGURE 15.4** Visualization of the Gaussian kernel estimate of a density generated from random samples drawn for that density.

That is, starting with a set of random samples $\{\mathbf{x}^{(i)}, i = 1, \dots, N_{\text{old}}\}$, a new set of samples $\{\mathbf{x}^{(j)}, j = 1, \dots, N_{\text{new}}\}$ is generated, where N_{new} can be much greater than N_{old} .

15.2 GENERAL CONCEPTS IMPORTANCE SAMPLING

As seen in Part II of this book, almost all of the Gaussian estimation (tracking) methods for nonlinear systems involve some kind of numerical evaluation of density-weighted moment integrals. Some of these numerical methods consist of weighted sums of a function at specific *deterministic* sigma points.

If the weighting density is multivariate and non-Gaussian or unknown, in general deterministic numerical methods cannot be used except for some cases with known analytical densities [13]. Recent (over the past 30 years) developments in Monte Carlo methods of integration for general density-weighted integrals have opened the floodgates for applications to estimation and tracking. The most influential of these methods revolve around the concept of *independent importance sampling*.

For a more complete discussion of the origins and fundamental developments of both general Monte Carlo integration methods and sequential importance sampling methods, the reader is referred to the books by Robert and Casella [14] and Doucet et al. [15] and the articles by Doucet [16] and Liu and Chen [17] and the references contained therein.

In Chapter 12, we solved the Gaussian density-weighted integrals by generating Monte Carlo samples from Gaussian densities thus reducing the integrals to weighted sums of functions evaluated at the sample points. For unknown or non-Gaussian densities, the process of generating Monte Carlo samples from that density can be accomplished most efficiently using importance sampling. In importance sampling, one generates a set of independent Monte Carlo samples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N_s)}\}$ from a known importance density $q(\mathbf{x})$ and then takes a weighted sum (average) of the integrand evaluated at the sample values. There is a great deal of freedom in the choosing of $q(\mathbf{x})$, with the better choices generating sample values that lie in the region that is important for the value of the integral, that is not in a region where the integrand is negligible.

The concept of importance sampling [13] began with the desire to evaluate integrals of the form

$$\hat{\mathbf{f}}(\mathbf{x}) = \int \mathbf{f}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \quad (15.29)$$

First, consider the case where $p(\mathbf{x})$ is Gaussian. One way to approximate the integral is to generate a set of samples $\{\mathbf{x}^{(i)}, i = 1, \dots, N\}$ from the Gaussian distribution $\mathcal{N}(\mathbf{x}; \mathbf{x}, \mathbf{P}_x)$ and replace $p(\mathbf{x})$ by its discrete sample version

$$\begin{aligned} p(\mathbf{x}) &\simeq \frac{1}{N} \sum_{i=1}^N \delta(\mathbf{x} - \mathbf{x}^{(i)}) \\ &= \sum_{i=1}^N w_i \delta(\mathbf{x} - \mathbf{x}^{(i)}) \end{aligned} \quad (15.30)$$

where $w_i = 1/N, \forall i$. Now the integral in (15.29) becomes

$$\begin{aligned} \hat{\mathbf{f}}(\mathbf{x}) &\simeq \int \mathbf{f}(\mathbf{x}) \sum_{i=1}^N w_i \delta(\mathbf{x} - \mathbf{x}^{(i)}) d\mathbf{x} \\ &= \sum_{i=1}^N w_i \mathbf{f}(\mathbf{x}^{(i)}) \end{aligned} \quad (15.31)$$

Here, w_i represents the importance of $\mathbf{x}^{(i)}$ to the sample mean (sum). Since we sampled directly from a known density, in this case the Gaussian density, all samples are equally important to the summation for the sample mean and therefore $w_i = 1/N$. Note that the weight w_i does not represent the probability of $\mathbf{x}^{(i)}$, just its importance to the weighted sum approximation to the moment integral. It can be shown [13] that (15.31) converges to the exact solution as $N \rightarrow \infty$.

Suppose that $p(\mathbf{x})$ is an unknown PDF or one that is difficult to sample. Also suppose that we have a second, well known and easily sampled PDF $q(x)$, referred to as the *importance density* for reasons that will be explained below. For example, $p(\mathbf{x})$ could be an unknown density and $q(x)$ could be a Gaussian density. Assuming that the support of $q(x)$ is greater than or equal to the support of $p(\mathbf{x})$, then $p(\mathbf{x})$ is proportional to $q(x)$ at each value of \mathbf{x} . Since $p(\mathbf{x})$ is a normalized PDF, $q(x)$ must be a scaled version of $p(\mathbf{x})$ with a different scaling factor at each x . This is absolutely true for any two PDFs with the same support. Now, we can define a proportionality or weighting factor as

$$w(\mathbf{x}) = \frac{p(\mathbf{x})}{q(x)} \quad (15.32)$$

It is imperative that $q(x)$ have a support that is greater than or equal to that of $p(\mathbf{x})$. For this reason, in many instances, a multidimensional Gaussian distribution is the best choice for $q(x)$ since the span of the components of \mathbf{x} are $-\infty \leq x_i \leq \infty$, $i = 1, \dots, n_x$.

An illustrative one-dimensional example is presented in Figure 15.5, where $p(x)$ is multimodal (solid line in top chart), $q(x)$ is a Gaussian PDF that is broader than $p(x)$ (dashed line in top chart), and $w(x)$ is shown in the bottom chart.

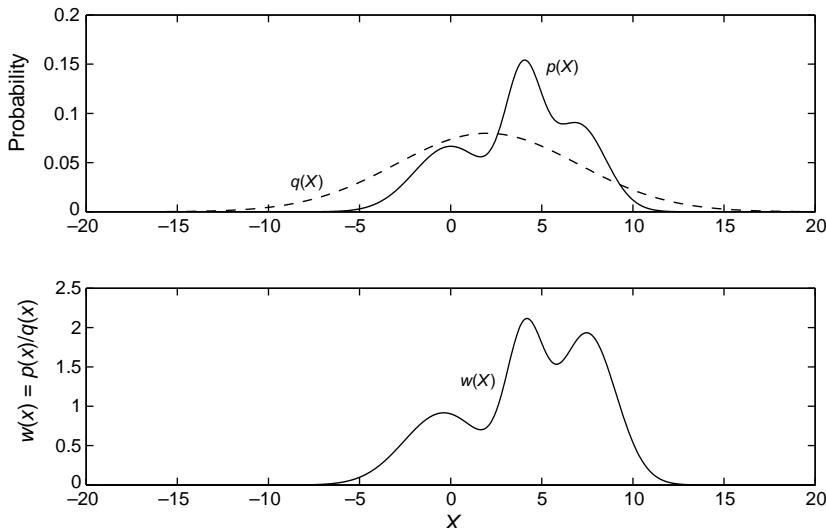


FIGURE 15.5 Example of the creation of $w(x)$.

At this point it must be emphasized that $w(\mathbf{x})$ is NOT a density function but the unnormalized ratio of two densities. Even when normalized, $w(\mathbf{x})$ cannot be used as a substitute for $p(\mathbf{x})$. This becomes important if one is using a maximum a posteriori (MAP) estimate of \mathbf{x} . If the peak of $q(\mathbf{x})$ does not coincide with the peak of $p(\mathbf{x})$, then the peak of $w(\mathbf{x})$ is not the MAP estimate for $p(\mathbf{x})$.

Now, suppose a sample $x^{(i)}$ is drawn from $q(x)$. Assuming that $q(x)$ is Gaussian with $q(x) = \mathcal{N}(x; \mu, \sigma^2)$, we can calculate the probability of x_i under $p(x)$ in the following manner

$$q(x = x^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{1}{2\sigma^2} ((x = x^{(i)}) - \mu)^2 \right\} \quad (15.33)$$

and therefore

$$w(x = x^{(i)}) = \frac{p(x = x^{(i)})}{q(x = x^{(i)})} \quad (15.34)$$

For multidimensional samples $\{\mathbf{x}^{(i)}, i = 1, \dots, N\}$ from the known distribution $q(\mathbf{x})$, use some *as yet to be determined method* to generate w_i for each sample and replace $p(\mathbf{x})$ by its discrete version

$$p(\mathbf{x}) \simeq \sum_{i=1}^N w_i \delta(\mathbf{x} - \mathbf{x}^{(i)}) \quad (15.35)$$

Consider again the integral (15.29) that can now be rewritten as

$$\begin{aligned} \hat{\mathbf{f}}(\mathbf{x}) &= \int \mathbf{f}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \\ &= \int \mathbf{f}(\mathbf{x}) w(\mathbf{x}) q(\mathbf{x}) d\mathbf{x} \\ &= \int \mathbf{f}(\mathbf{x}) \sum_{i=1}^N w_i \delta(\mathbf{x} - \mathbf{x}^{(i)}) d\mathbf{x} \\ &= \sum_{i=1}^N w_i \mathbf{f}(\mathbf{x}^{(i)}) \end{aligned} \quad (15.36)$$

This has the exact same form as (15.31), but now the weights w_i are no longer uniformly equal to $1/N$. Since the weights are part of the definition of the discrete density approximation given in (15.35), it is obvious that we must normalize the weights so that $\sum w_i = 1$.

Consider the first-order Markov process \mathbf{x}_n whose estimated moments are governed by the conditional posterior density $p(\mathbf{x}_n | \mathbf{z}_{1:n})$, with an initial posterior, prior to any observations, given by $p(\mathbf{x}_0)$ (See Section 3.3). A general expression for the moments

of a nonlinear function under an arbitrary posteriori pdf is given by

$$I(\mathbf{x}_n) = \int \mathbf{g}(\mathbf{x}_n) p(\mathbf{x}_n | \mathbf{z}_{1:n}) d\mathbf{x}_n \quad (15.37)$$

Here $\mathbf{g}(\mathbf{x}_n)$ is a functional that can represent any moment generating function. For example, if we wish to find the mean of a function $\mathbf{f}(\mathbf{x}_n)$, then we let $\mathbf{g}(\mathbf{x}_n) = \mathbf{f}(\mathbf{x}_n)$ leading to $I(\mathbf{x}_n) = \hat{\mathbf{f}}(\mathbf{x}_n)$. If we wish to compute the covariance of $\mathbf{f}(\mathbf{x}_n)$, let $\mathbf{g}(\mathbf{x}_n) = [\mathbf{f}(\mathbf{x}_n) - \hat{\mathbf{f}}(\mathbf{x}_n)][\mathbf{f}(\mathbf{x}_n) - \hat{\mathbf{f}}(\mathbf{x}_n)]^\top$, which leads to $I(\mathbf{x}_n) = \mathbf{P}_n^{\text{ff}}$.

Assume that $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ is unknown or difficult to sample and that we have an easily sampled importance density $q(\mathbf{x}_n | \mathbf{z}_{1:n})$ that has the same support as $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ and has its maximum in roughly the same place as $p(\mathbf{x}_n | \mathbf{z}_{1:n})$. Now, (15.37) can be written as

$$\begin{aligned} I(\mathbf{x}_n) &= \int \mathbf{g}(\mathbf{x}_n) \frac{p(\mathbf{x}_n | \mathbf{z}_{1:n})}{q(\mathbf{x}_n | \mathbf{z}_{1:n})} q(\mathbf{x}_n | \mathbf{z}_{1:n}) d\mathbf{x}_n \\ &= \int \mathbf{g}(\mathbf{x}_n) w(\mathbf{x}_n) q(\mathbf{x}_n | \mathbf{z}_{1:n}) d\mathbf{x}_n \end{aligned} \quad (15.38)$$

where the weight function $w(\mathbf{x}_n)$ is defined by

$$w(\mathbf{x}_n) \triangleq \frac{p(\mathbf{x}_n | \mathbf{z}_{1:n})}{q(\mathbf{x}_n | \mathbf{z}_{1:n})} \quad (15.39)$$

Using Bayes' rule (3.23) for $p(\mathbf{x}_n | \mathbf{z}_{1:n})$, (15.39) becomes

$$w(\mathbf{x}_n) = \frac{p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{z}_{1:n-1})}{p(\mathbf{z}_n | \mathbf{z}_{1:n-1})} \frac{1}{q(\mathbf{x}_n | \mathbf{z}_{1:n})} \quad (15.40)$$

Noting that $p(\mathbf{z}_n | \mathbf{z}_{1:n-1})$ is just a normalization term, we can write

$$w(\mathbf{x}_n) \propto \frac{p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{z}_{1:n-1})}{q(\mathbf{x}_n | \mathbf{z}_{1:n})} \quad (15.41)$$

where \propto represents a proportionality. Thus (15.41) formalizes the general weight function in terms of the likelihood function $p(\mathbf{z}_n | \mathbf{x}_n)$, the prior pdf $p(\mathbf{x}_n | \mathbf{z}_{1:n-1})$ and the importance pdf $q(\mathbf{x}_n | \mathbf{z}_{1:n})$.

If we were to select a set of samples $\{\mathbf{x}_n^{(i)}, i = 1, \dots, N_s\}$ from the importance density $q(\mathbf{x}_n | \mathbf{z}_{1:n})$, we can approximate $q(\mathbf{x}_n | \mathbf{z}_{1:n})$ by the set of Dirac delta function (see (15.5)) given by

$$q(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \delta(\mathbf{x}_n - \mathbf{x}_n^{(i)}) \quad (15.42)$$

and (15.38) now becomes

$$\begin{aligned} I(\mathbf{x}_n) &= \int \mathbf{g}(\mathbf{x}_n) w(\mathbf{x}_n) \frac{1}{N_s} \sum_{i=1}^{N_s} \delta(\mathbf{x}_n - \mathbf{x}_n^{(i)}) d\mathbf{x}_n \\ &= \frac{1}{N_s} \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{g}(\mathbf{x}_n^{(i)}) \end{aligned} \quad (15.43)$$

where $w_n^{(i)} \triangleq w(\mathbf{x}_n^{(i)})$. From (15.39) we obtain

$$w_n^{(i)} = \frac{p(\mathbf{x}_n^{(i)} | \mathbf{z}_{1:n})}{q(\mathbf{x}_n^{(i)} | \mathbf{z}_{1:n})} \quad (15.44)$$

Or, using (15.41), a set of unnormalized weights are given by

$$\tilde{w}_n^{(i)} = \frac{p(\mathbf{z}_n | \mathbf{x}_n^{(i)}) p(\mathbf{x}_n^{(i)} | \mathbf{z}_{1:n-1})}{q(\mathbf{x}_n^{(i)} | \mathbf{z}_{1:n})} \quad (15.45)$$

The weights are unnormalized because we have replaced the proportionality of (15.41) by an equality. The weights can now be normalized by forcing their sum to one using

$$w_n^{(i)} \triangleq \frac{1/N_s \tilde{w}_n^{(i)}}{1/N_s \sum_{i=1}^{N_s} \tilde{w}_n^{(i)}} \quad (15.46)$$

Now, (15.43) becomes

$$I(\mathbf{x}_n) = \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{g}(\mathbf{x}_n^{(i)}) \quad (15.47)$$

It is important to emphasize again that $w_n^{(i)}$ is not the probability of $\mathbf{x}_n^{(i)}$ but rather the importance of $\mathbf{x}_n^{(i)}$ to the discrete moment integral sum.

15.3 SUMMARY

In this chapter we have presented two main concepts, methods for the representation of a density function based on a set of samples drawn from that density, and the general concept of Monte Carlo importance sampling in the numerical integration of functions weighted by an arbitrary pdf. At present, a practical method for estimation and tracking cannot be presented based solely on these results and additional modifications must be made to develop practical methods.

Tracking filters based on a sequential generation of the recursive sets of Monte Carlo points and weights $\{\mathbf{x}_n^{(i)}, w_n^{(i)}, i = 1, \dots, N_s, n = 1, \dots, N\}$ have come to be called Sequential Monte Carlo (SMC) methods. They offer a number of significant

advantages over many of the other techniques currently available as a result of the generality of the SMC approach. They allow for an inference of the full posterior distribution in terms of general state space models including both nonlinear and non-Gaussian. SMC methods therefore allow for computation of all kinds of moments, quantiles, and maximum posterior density regions, whereas the Gaussian Kalman filter variants allow only approximations of the first- and second-order moments. In addition, by appropriately specifying the state space model, the SMC methods can incorporate constraints on the state space such as limits on the range of the state space, representing such physical limitations such as speed constraints or hard boundaries.

In the next chapter we will introduce a more restrictive approach to SMC, where the weight for each sample is generated in a sequential manner giving rise to a family of sequential importance sampling (SIS) particle filters. Later, in Chapter 17, we will introduce a less restrictive method that recomputes the weight functions with every iteration.

REFERENCES

1. Richardson HR, Stone LD, Monach WR, Discenza JH. Early maritime applications of particle filtering. *Signal and Data Processing of Small Targets, Proceedings of SPIE*, Vol. 5204, 2003, pp. 165–174.
2. Agnew DC, Constable C. *Geophysical Data Analysis*. [Online] <http://mahi.ucsd.edu/cathy/Classes/SIO223/Part1/>. Chapter 9. 2005.
3. Scott DW. Multivariate density estimation and visualization. In *Handbook of Statistics, Vol. 23: Data Mining and Computational Statistics*. Springer; 2004.
4. Patlolla R. 2-Dimensional histogram. [Online] <http://www.mathworks.com/matlabcentral/fileexchange/8422>. 2005.
5. Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed. Academic Press, 1990.
6. Silverman BW. *Density Estimation for Statistics and Data Analysis*. New York: Chapman and Hall; 1986.
7. Turlach BA. Bandwidth selection in Kernel density estimation: a review. *In CORE Institut Stat* 1993;23–49.
8. Musso C, Oudjane N, Le Gland F. Improving regularized particle filters. In Doucet A, de Freita N, Gordon G, editors. *Sequential Monte Carlo Methods in Practice*. Springer; 2001.
9. Wand MP, Jones MC. Comparison of smoothing parameterization in bivariate Kernel density estimation. *J. Am. Stat. Soc.* 1993;88(422):520–528.
10. Zhang X, King ML, Hyndman RJ. Bandwidth selection for multivariate Kernel density estimation using MCMC. *Comp. Stat. Data Anal.* 2006;50:3009–3031.
11. Wikipedia. [Online] [http://en.wikipedia.org/wiki/Kernel_\(statistics\)](http://en.wikipedia.org/wiki/Kernel_(statistics)).
12. Botev Z. Two-Dimensional Kernel Density Estimator Function. [Online] <http://www.mathworks.com/matlabcentral/fileexchange/17204-kernel-density-estimation>.
13. Evans M, Swartz T. *Approximating Integrals via Monte Carlo and Deterministic Methods*. New York: Oxford University Press; 2005.

14. Robert CP, Casella G. *Monte Carlo Statistical Methods*. Springer; 2004.
15. Doucet A, de Freita N, Gordon G, editors. *Sequential Monte Carlo Methods in Practice*. Springer; 2001.
16. Doucet A. On Sequential Simulation-Based Methods for Bayesian Estimation. Technical Report CUED/F-INFENG/TR.310; 1998.
17. Liu JS, Chen R. Sequential Monte Carlo methods for dynamic systems. *J. Am. Stat. Soc.* 1998;93(443):1032–1044.

16

SEQUENTIAL IMPORTANCE SAMPLING PARTICLE FILTERS

16.1 GENERAL CONCEPT OF SEQUENTIAL IMPORTANCE SAMPLING

Remember that in all of the Gaussian Kalman filter estimators developed in Part II, the estimates at time t_n depended in an analytical way on the estimates at time t_{n-1} . We would like to develop a similar recursive formulation for the importance sampling approach for estimation when the noise densities are non-Gaussian.

Recursive estimation methods based on the sequential estimation of the *weights* are called sequential importance sampling (SIS) particle filters. Using the Chapman–Kolmogorov theorem (3.24), $p(\mathbf{x}_n | \mathbf{z}_{1:n-1})$ and $q(\mathbf{x}_n | \mathbf{z}_{1:n})$ in (15.41) can be expanded so that (15.41) becomes

$$w(\mathbf{x}_n) \propto \frac{p(\mathbf{z}_n | \mathbf{x}_n) \int p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_{1:n-1}) p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1}}{\int q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_{1:n}) q(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1}} \quad (16.1)$$

The reason we have $q(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1})$ in the denominator instead of $q(\mathbf{x}_{n-1} | \mathbf{z}_{1:n})$ results from the fact that \mathbf{x}_{n-1} cannot be conditioned on a future observation. Since the state evolution equation is assumed to be a first-order Markov process independent of the observations, we can let $p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_{1:n-1}) \rightarrow p(\mathbf{x}_n | \mathbf{x}_{n-1})$ resulting in

$$w(\mathbf{x}_n) \propto \frac{p(\mathbf{z}_n | \mathbf{x}_n) \int p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1}}{\int q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_{1:n}) q(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1}} \quad (16.2)$$

For a Markov process, we can also make the assumption that

$$q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_{1:n}) = q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) \quad (16.3)$$

so that the importance density depends on only \mathbf{x}_{n-1} and \mathbf{z}_n .

Now assume that we have a set of random samples $\{\mathbf{x}_{n-1}^{(i)}, i = 1, \dots, N_s\}$ from $q(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1})$ (in the last chapter, we generated samples from $q(\mathbf{x}_n | \mathbf{z}_{1:n})$), that is, $\mathbf{x}_{n-1}^{(i)} \sim q(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1})$. Now, (16.2) becomes

$$\tilde{w}(\mathbf{x}_n) = \sum_{i=1}^{N_s} \frac{p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}) p(\mathbf{x}_{n-1}^{(i)} | \mathbf{z}_{1:n-1})}{q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n) q(\mathbf{x}_{n-1}^{(i)} | \mathbf{z}_{1:n-1})} \quad (16.4)$$

From the definition of $w(\mathbf{x}_n)$ given in (15.39), it follows that

$$\tilde{w}(\mathbf{x}_n) = \sum_{i=1}^{N_s} w_{n-1}^{(i)} \frac{p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \quad (16.5)$$

This leads to an expression for the posterior distribution

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \sum_{i=1}^{N_s} w_{n-1}^{(i)} \frac{p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} p(\mathbf{z}_n | \mathbf{x}_n) q(\mathbf{x}_n | \mathbf{z}_{1:n}) \quad (16.6)$$

Generating samples $\{\mathbf{x}_n^{(i)}, i = 1, \dots, N_s\}$ from $q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)$ such that

$$\mathbf{x}_n^{(i)} \sim q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n) \quad (16.7)$$

$$q(\mathbf{x}_n | \mathbf{z}_{1:n}) = \frac{1}{N_s} \delta(\mathbf{x}_n - \mathbf{x}_n^{(i)}) \quad (16.8)$$

results in

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \sum_{i=1}^{N_s} \frac{\tilde{w}_{n-1}^{(i)}}{N_s} \frac{p(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} p(\mathbf{z}_n | \mathbf{x}_n^{(i)}) \delta(\mathbf{x}_n - \mathbf{x}_n^{(i)}) \quad (16.9)$$

After normalizing $\tilde{w}_{n-1}^{(i)} / N_s$ using

$$w_{n-1}^{(i)} = \frac{\tilde{w}_{n-1}^{(i)} / N_s}{\sum_{i=1}^{N_s} \tilde{w}_{n-1}^{(i)} / N_s} \quad (16.10)$$

we obtain

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \sum_{i=1}^{N_s} w_n^{(i)} \delta\left(\mathbf{x}_n - \mathbf{x}_n^{(i)}\right) \quad (16.11)$$

where the SIS recursive weight update equation is given by

$$w_n^{(i)} \triangleq w_{n-1}^{(i)} \frac{p\left(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}\right)}{q\left(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n\right)} p\left(\mathbf{z}_n | \mathbf{x}_n^{(i)}\right) \quad (16.12)$$

Returning to the general moment Equation (15.37), the first two moments of \mathbf{x}_n , with respect to $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ can be obtained from

$$\begin{aligned} \hat{\mathbf{x}}_n &\triangleq \mathcal{E}\{\mathbf{x}_n | \mathbf{z}_{1:n}\} = \int_{\mathbb{R}} \mathbf{x}_n p(\mathbf{x}_n | \mathbf{z}_{1:n}) d\mathbf{x}_n \\ &= \int_{\mathbb{R}} \mathbf{x}_n \sum_{i=1}^{N_s} w_n^{(i)} \delta\left(\mathbf{x}_n - \mathbf{x}_n^{(i)}\right) \\ &= \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{x}_n^{(i)} \end{aligned} \quad (16.13)$$

and similarly

$$\begin{aligned} \mathbf{P}_n^{\mathbf{xx}} &\triangleq \mathcal{E}\{(\mathbf{x}_n - \hat{\mathbf{x}}_n)(\mathbf{x}_n - \hat{\mathbf{x}}_n)^T | \mathbf{z}_{1:n}\} \\ &= \sum_{i=1}^{N_s} w_n^{(i)} \left(\mathbf{x}_n^{(i)} - \hat{\mathbf{x}}_n\right) \left(\mathbf{x}_n^{(i)} - \hat{\mathbf{x}}_n\right)^T \end{aligned} \quad (16.14)$$

The procedure for the SIS particle filter is quite simple and is presented in Table 16.1.

There are several significant practical problems hidden in the SIS particle filter procedure. The first problem occurs in Step 2 of the SIS procedure, where analytical expressions for the likelihood function $p(\mathbf{z}_n | \mathbf{x}_n)$, predictive density $p(\mathbf{x}_n | \mathbf{x}_{n-1})$ and importance density $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$ are required. The likelihood function is based on the observation noise density that is usually assumed to be Gaussian. However, when the observation equation is highly nonlinear, this assumption can easily be violated, as has been shown in Section 4.4 for the DIFAR buoy case study. The predictive density is based on the noise in the dynamic transition equation and may be Gaussian, non-Gaussian or completely unknown. Finally, the importance density is usually some known analytical density from which samples can be easily drawn. But note that $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$ is conditioned on the current observation. For this reason, the importance density can usually be taken as the posterior density at the output of a nonlinear Gaussian Kalman filter and applied to each particle where the update

TABLE 16.1 General Sequential Importance Sampling Particle Filter

<i>A. Initialize filter</i>	
Initialize state vector samples	$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$
Initialize weights	$w_0^{(i)} = \frac{1}{N_s}$
<i>B. Sequential importance sampling</i>	
1. Draw new samples	$\mathbf{x}_n^{(i)} \sim q(\mathbf{x}_n \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n), \forall i$
2. Generate unnormalized importance weights	$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} \times \frac{p(\mathbf{z}_n \mathbf{x}_n^{(i)}) p(\mathbf{x}_n^{(i)} \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n^{(i)} \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)}$
3. Normalize the weights	$w_n^{(i)} = \tilde{w}_n^{(i)} / \sum_{i=1}^N \tilde{w}_n^{(i)}$
4. Time step and return to 1	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}$ $w_n^{(i)} \rightarrow w_{n-1}^{(i)}$
<i>C. Calculate moments if desired</i>	
First moment: state vector mean	$\hat{\mathbf{x}}_n = \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{x}_n^{(i)}$
Second moment: covariance	$\mathbf{P}_n^{\mathbf{x}\mathbf{x}} = \sum_{i=1}^{N_s} w_n^{(i)} \times (\mathbf{x}_n^{(i)} - \hat{\mathbf{x}}_n) (\mathbf{x}_n^{(i)} - \hat{\mathbf{x}}_n)^T$

equations use the latest observations. Consideration of issues related to these three densities for SIS particle filters are very important and must be considered carefully.

The second problem with SIS particle filters concerns the divergence of the particles. Notice that particles are generated *only* during the initialization step. Although Step 2 appears to be a resampling step, in practice, usually a nonlinear Gaussian Kalman filter is used for the importance density with each particle processed separately. In this case, the original particles are just passed through without resampling. Thus, the original particles are propagated through Steps 1–4, but no new particles are created after particle creation during initialization. It is for this reason that these Monte Carlo estimation methods have the name *particle* filter, since the particles trajectories represent temporal paths through the state space. The particles are created as samples from some initializing importance density and then propagated through the SIS process over and over again, with noise added at each iteration. This causes the spread of the particle based discrete posterior density to diverge from that of the true posterior density, reducing the weights to zero for all but a few central particles. This problem is often referred to as the SIS *degeneracy* problem. After each particle propagation, the weights for all particles are used to estimate the moments, but fewer and fewer particles have a high enough weight (importance) to contribute to the discrete sum, increasing the variance of the moment estimates. Reducing this growth in estimation variance caused by this effect constitutes the bulk of SIS particle filter methodology.

It should also be noted that the initialization importance density $q(\mathbf{x}_0)$ need not even be the same as the importance density $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$ used in SIS Step 1. One

wants the initialization density $q(\mathbf{x}_0)$ to be as uninformative as possible so as to include Monte Carlo samples in all possible regions of the posterior. But one wants $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$ to be highly localized through the use of the latest observations \mathbf{z}_n .

16.2 RESAMPLING AND REGULARIZATION (MOVE) FOR SIS PARTICLE FILTERS

The SIS sample degeneracy problem can be moderated by a process called resampling. The basic idea of resampling is to eliminate state space particles with small weights by replacing them with replicated values of particles with larger weights. For this process, the *weights* are treated as a one-dimensional discrete probability distribution and the inverse transform method [1,2] is used to resample the particles associated with the inverse of the cumulative distribution of the weights.

However, this leads to a new problem, a loss of diversity among the particles, since the resultant sample set will contain many repeated particles for any given weight. This problem has been labeled in the literature as *sample impoverishment*. In severe cases, all particles migrate to one sample point. To rectify the sample impoverishment due to resampling, after each resampling process a kernel density estimate of the particle density can be used to resample the particles a second time. In this process, each new particle (Monte Carlo sample) is selected from the resampled particles based on a draw from a uniform distribution and then the sample point is moved a small amount based on a draw from the local kernel. This process tends to concentrate the particles in the region of highest probability and separates them in a random fashion. This method of reducing sample impoverishment is called *regularization* [3] and constitutes the *move* part of a *resample and move process*.

There are several alternatives to the resample and move method discussed above [4–8], including a Markov Chain Monte Carlo (MCMC) sampling method that utilizes the *Metropolis–Hastings* acceptance algorithm instead of a regularization step and a Gibbs sampling method similar to MCMC. In general, these methods prove to be too computationally intensive for real-time recursive tracking applications and will therefore not be considered here. However, in cases where the posterior density has large tail probabilities, as is the case for alpha-stable distributions such as a Levy distribution, the standard SIS particle filter methods may fail due to difficulties in the selection of an appropriate importance density. In such instances, the use of an MCMC method in particle filters provides an alternative for building efficient high dimensional proposal distribution. For a discussion of the MCMC particle filter, see Ref. [4] and the references contained therein. Other applications where these methods are useful are static parameter estimation and smoothing methods [5], neither of which will be addressed in this book.

16.2.1 The Inverse Transform Method

First, let's examine the one-dimensional cumulative distribution function $P(x)$ shown in Figure 2.7 for the analytical Gaussian distribution $\mathcal{N}(-5, 6^2)$. From the figure it

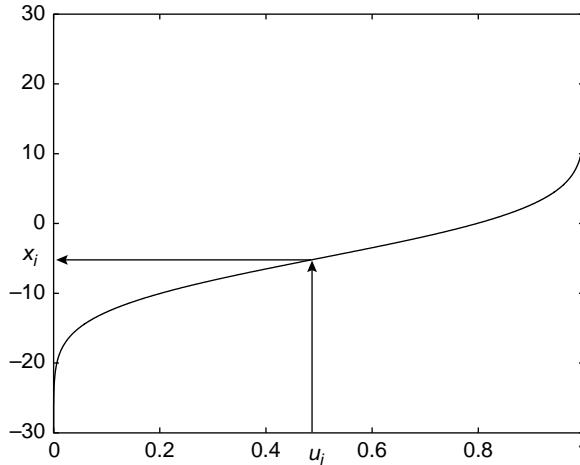


FIGURE 16.1 Inverse of the Gaussian cumulative distribution.

is obvious that the cumulative probabilities associated with the analytical CDF are uniformly distributed over the interval $(0,1]$. If we invert the CDF, we obtain $P^{-1}(u)$, which is shown in Figure 16.1. Now, if we generate a sample (u_i) from a uniform distribution on $(0,1]$ and use it to select a value along the vertical x -axis (x_i), for this example we would generate a sample from $\mathcal{N}(-5, 6^2)$, as shown in the figure. Note that for this example, very few samples of x will be selected outside of the range $-5 - 13 \leq x \leq -5 + 13$.

In general, the cumulative distribution and its inverse have the following properties [1,2,9]:

1. $P^{-1}(U)$ is monotonically increasing
2. $P(P^{-1}(U)) \geq u$ and $P(P^{-1}(X)) \leq x$
3. $u \leq P(X)$ if and only if $P^{-1}(U) \leq x$
4. P^{-1} is left continuous

Theorem 16.1 if $u \sim U(0, 1)$, then $x \sim P^{-1}(U) = P(X)$

Proof: $P(X \leq x) = P(X \leq P^{-1}(U)) = P(P^{-1}(u)) = u = P(X)$ ■

This general theorem applies to both analytical and empirical one-dimensional distributions only. It can only be applied to multidimensional distributions in some special cases. We illustrate this inverse distribution sampling method with a simple example. First we use Matlab to generate 50,000 samples from $\mathcal{N}(-5, 6^2)$ using the method described in Chapter 2. Next we form an empirical pdf by creating a 13 bin histogram that is normalized by the total number of samples. A CDF is then formed from a bin by bin cumulative sum. Figure 16.2 is then formed by plotting $P(x)$ versus

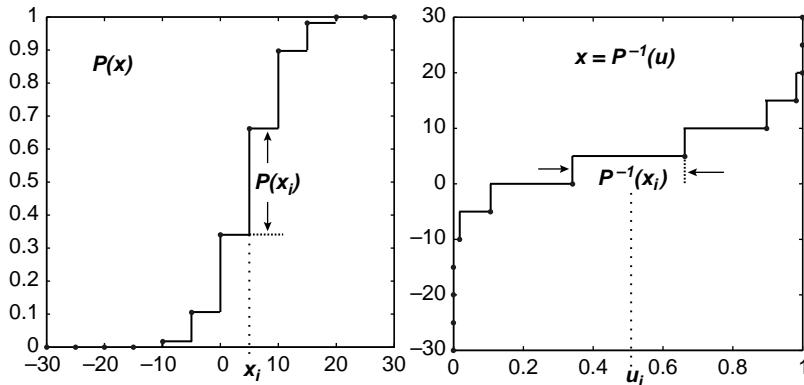


FIGURE 16.2 Comparison of a Gaussian CDF with its inverse.

x and x versus $P^{-1}(u)$. Note that the process of resampling for this simple example can generate only seven unique values of x .

There are several numerical approaches to implementing the inverse method for sample generation. We will consider only three here: random sampling, stratified sampling and systematic sampling. However, research has shown that the “best” results are obtained with either *stratified* or *systematic* sampling [10].

For *random resampling*, one generates samples $\{u_i, i = 1, \dots, n\}$ from a uniform distribution $U(0, 1)$ and then selects the values $\{x_i, i = 1, \dots, n\}$ such that $x_i = P^{-1}(u_i)$. Some simple Matlab code that accomplishes this is presented in Listing 16.1.

Listing 16.1 Random Sampling Matlab Snippet

```
% Random Resampling
u = cumprod(rand(1,Ns)./[Ns:-1:1]);
u = u(N:-1:1);
wc = cumsum(w);
label = nan(1,Ns);
k = 1;

for i = 1:Ns
    while(wc(k) < u(i))
        k = k+1;
    end
    label(i) = k;
end

% Resample particles
particles = particles(:,label);
w = ones(1,Ns)./Ns;
```

In *stratified resampling*, based on ideas used in survey sampling, the interval $(0, 1]$ is prepartitioned into n uniform adjoint sets or strata, $(0, 1] = (0, 1/n] \cup (1/n, 2/n] \cup \dots \cup (n - 1/n, 1]$. Then, in each strata, a separate uniform random number is generated such that for the i th strata, $u_i \sim U(i/n, i + 1/n)$. Then one sets $x_i = P^{-1}(u_i)$. Some simplified Matlab code that accomplishes this is shown in Listing 16.2.

Listing 16.2 Stratified Sampling Matlab Snippet

```
% Stratified Resampling
u = ([0:Ns - 1] + rand(1,Ns))/Ns;
wc = cumsum(w);
label = nan(1,Ns);
k = 1;

for i = 1:Ns
    while(wc(k) < u(i))
        k = k+1;
    end
    label(i) = k;
end

% Resample particles
particles = particles(:,label);
w = ones(1,Ns)./Ns;
```

Finally, in *systematic resampling*, the uniform region is again subdivided into n uniform adjoint strata, as in stratified sampling, but only a single random number draw is used for all strata. Listing 16.3 contains a snippet of Matlab code that accomplishes this.

Listing 16.3 Systematic Sampling Matlab Snippet

```
% Systematic Resampling
u = ([0:Ns - 1] + rand(1,Ns))/Ns;
wc = cumsum(w);
label = nan(1,Ns);
k = 1;

for i = 1:Ns
    while(wc(k) < u(i))
        k = k+1;
    end
    label(i) = k;
end

% Resample particles
particles = particles(:,label);
w = ones(1,Ns)./Ns;
```

For all of the particle filters requiring resampling used in our case studies, we used systematic resampling.

16.2.2 SIS Particle Filter with Resampling

The inverse transform method can be used for resampling in SIS particle filters. We show this by first forming the discrete approximation of the *cumulative* distribution $P(\mathbf{x}_n | \mathbf{z}_{1:n})$ for the discrete $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ shown in (16.11). From the definition of the cumulative distribution given in (2.80) we can write

$$\begin{aligned} P\left(\mathbf{x}_n^{(j)} | \mathbf{z}_{1:n}\right) &= \Pr\left\{\mathbf{y}_n | \mathbf{z}_{1:n} \leq \mathbf{x}_n^{(j)} | \mathbf{z}_{1:n}\right\} = \int_{-\infty}^{\mathbf{x}_n^{(j)}} p(\mathbf{y}_n | \mathbf{z}_{1:n}) d\mathbf{y}_n \\ &= \int_{-\infty}^{\mathbf{x}_n^{(j)}} \sum_{i=1}^{N_s} w_n^{(i)} \delta\left(\mathbf{y}_n - \mathbf{y}_n^{(i)}\right) d\mathbf{y}_n \\ &= \sum_{i=1}^j w_n^{(i)} \end{aligned} \quad (16.15)$$

Thus, each particle of the discrete $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ contributes its weight to the CDF primarily because the CDF is obtained from the PDF through integration. This is a very interesting conclusion. In Chapter 2, we presented a multidimensional CDF where the components of the CDF contained the same number of dimensions as the PDF. For the discrete $p(\mathbf{x}_n | \mathbf{z}_{1:n})$, although the state vector \mathbf{x}_n and $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ are multidimensional, a one-dimensional CDF can be formed from just the weights. It is for this reason we can utilize the inverse transform method to resample the discrete multidimensional $p(\mathbf{x}_n | \mathbf{z}_{1:n})$.

These particle filter resampling steps modify the weighted approximation density to create an unweighted density by eliminating particles having low importance weights. Each low weight particle eliminated is replaced by replicating a particle having a higher importance weight, so that the number of particles remains constant. Thus

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \sum_{i=1}^{N_s} w_n^{(i)} \delta\left(\mathbf{x}_n - \mathbf{x}_n^{(i)}\right) \quad (16.16)$$

is replace by

$$p\left(\mathbf{x}_n^* | \mathbf{z}_{1:n}\right) = \sum_{i=1}^{N_s} \frac{1}{N_s} \delta\left(\mathbf{x}_n - \mathbf{x}_n^*\right) = \sum_{i=1}^{N_s} \frac{n_i}{N_s} \delta\left(\mathbf{x}_n - \mathbf{x}_n^{(i)}\right) \quad (16.17)$$

where n_i is the number of copies of particle $\mathbf{x}_n^{(i)}$ in the resampled set of particles $\{\mathbf{x}_n^*\}$. See Ref. [10] for a complete evaluation of these and other SIS particle filter resampling methods.

In many applications of SIS particle filters, the degeneracy caused by migration of the spread of the discrete particle density away from that of the true density is not severe, so resampling need not be done on every iteration. A suitable measure of degeneracy for a specific application is the effective sample size N_{eff} that can be

TABLE 16.2 Sequential Importance Sampling Particle Filter with Resampling

<i>A. Initialize filter</i>	
1. Initialize state vector samples	$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$
2. Initialize weights	$w_0^{(i)} = \frac{1}{N_s}$
<i>B. Sequential importance sampling</i>	
1. Draw new samples	$\mathbf{x}_n^{(i)} \sim q(\mathbf{x}_n \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n), \forall i$
2. Generate unnormalized importance weights	$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} \times \frac{p(\mathbf{z}_n \mathbf{x}_n^{(i)}) p(\mathbf{x}_n^{(i)} \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n^{(i)} \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)}$
3. Normalize the weights	$w_n^{(i)} = \tilde{w}_n^{(i)} / \sum_{i=1}^{N_s} \tilde{w}_n^{(i)}$
<i>C. Calculate N_{eff}</i>	$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_n^{(i)})^2}$
<i>D. If $N_{\text{eff}} \ll N_s$</i>	Resample using one of the resampling Matlab example snippets
1. Resample	$w_n^{(i)} = 1/N_s$
2. Reset the weights	
<i>end if</i>	
<i>E. Calculate moments if desired</i>	
1. First moment: state vector mean	$\hat{\mathbf{x}}_n = \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{x}_n^{(i)}$
2. Second moment: covariance	$\mathbf{P}_n^{\text{xx}} = \sum_{i=1}^{N_s} w_n^{(i)} \times (\mathbf{x}_n^{(i)} - \hat{\mathbf{x}}_n) (\mathbf{x}_n^{(i)} - \hat{\mathbf{x}}_n)^T$
<i>F. Time step and return to B.1</i>	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}; w_n^{(i)} \rightarrow w_{n-1}^{(i)}$

estimated from [11]

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_n^{(i)})^2} \quad (16.18)$$

It follows immediately that if the weights are all uniform, $w_n^{(i)} = 1/N_s$, and $N_{\text{eff}} = N_s$. However, if there is serious degeneracy, $w_n^{(i)} = 1$ for one value of i and is zero for all others, so $N_{\text{eff}} = 1$. Hence $N_{\text{eff}} \ll N_s$ indicates severe degeneracy while $N_{\text{eff}} \sim N_s$ indicates very little degeneracy.

The procedure for a *resampling* SIS particle filter is presented in Table 16.2.

16.2.3 Regularization

After resampling, the resultant sample set of particles may contain many repeated particles for any given weight. This problem has been labeled in the literature as *sample impoverishment*. To rectify the sample impoverishment due to resampling,

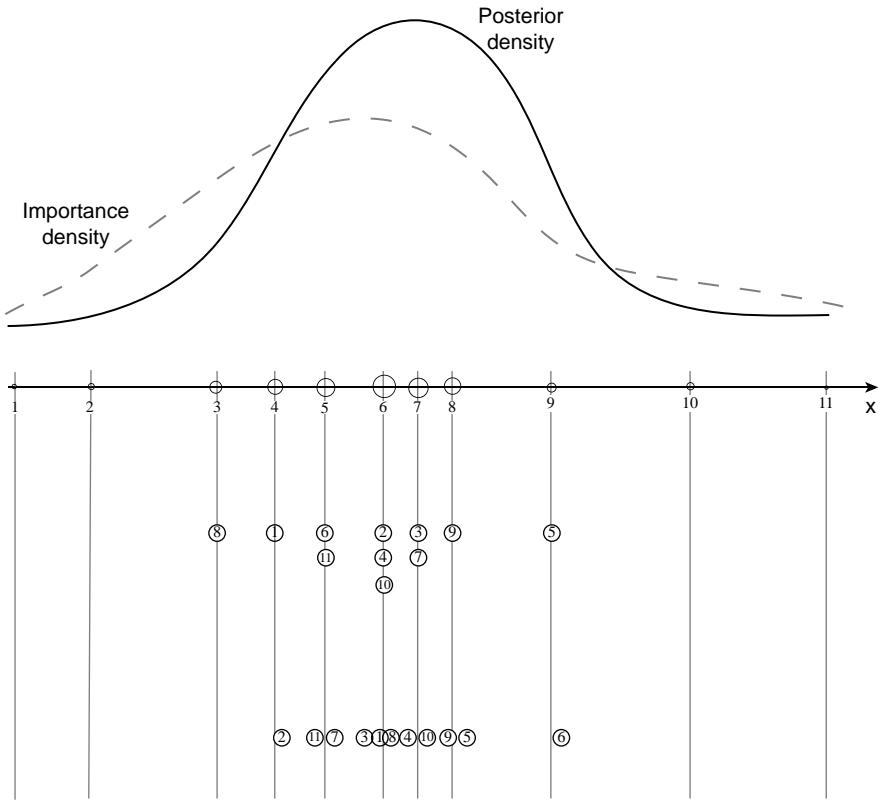


FIGURE 16.3 Principle of resampling. Top: Samples drawn from q (dashed line) with associated normalized importance weights depicted by bullets with radii proportional to the normalized weights (the target density corresponding to p is plotted as a solid line). Middle: After resampling, all particles have the same importance weight, and some have been duplicated. Bottom: Samples have been regularized using a Kernel density estimator that resamples and moves.

after each resampling process a kernel density estimate of the particle density can be used to resample the particles a second time. In this process, each new particle (Monte Carlo sample) is selected from the original resampled particles based on a draw from a uniform distribution and then the sample point is moved a small amount (dithered) based on a draw from the local individual kernel. This regularization process tends to concentrate the particles in the region of highest probability and separates them in a random fashion. An example of the complete results of this procedure is depicted on Figure 16.3.

In Figure 16.3, both the posterior density and the importance density are shown. Samples are drawn from the importance density and their position along the x -axis is depicted with bullets. Each particle is assigned an unnormalized weight given by the ratio $w_n^{(i)} = p(x_n^{(i)})/q(x_n^{(i)})$, with the bullet radii shown as the relative size of

the weights in the top line of Figure 16.3. The numbers are place holders used to show ancestral lineage, that is, where each individual particle ends up at the end of each step. The second line shows the particles after resampling, with all particles having equal weight and the lower weight particles turned into replications of existing higher weight samples. The third line depicts the samples after regularization, which resamples again and moves the particles so that no two are at the exact same value of x .

The procedure for a *resample and regularize* (move) SIS particle filter for a multi-dimensional state vector is presented in Table 16.3. However, note that computation of the empirical covariance matrix \mathbf{D}_s must be carried out prior to the resampling step and is therefore a function of both $\mathbf{x}_n^{(i)}$ and $w_n^{(i)}$. Since the regularization step resamples from the posterior density, all particles now have equal importance in the summation

TABLE 16.3 Sequential Importance Sampling Particle Filter with Resampling and Regularization

A. Initialize filter	
1. Initialize state vector samples	$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$
2. Initialize weights	$w_0^{(i)} = \frac{1}{N_s}$
B. Sequential importance sampling	
1. Draw new samples	$\mathbf{x}_n^{(i)} \sim q(\mathbf{x}_n \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n), \forall i$
2. Generate unnormalized importance weights	$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} \times \frac{p(\mathbf{z}_n \mathbf{x}_n^{(i)}) p(\mathbf{x}_n^{(i)} \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n^{(i)} \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)}$
3. Normalize the weights	$w_n^{(i)} = \tilde{w}_n^{(i)} / \sum_{i=1}^{N_s} \tilde{w}_n^{(i)}$
C. Calculate N_{eff}	$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_n^{(i)})^2}$
D. If $N_{\text{eff}} \ll N_s$	
a. Calculate the empirical mean	$\boldsymbol{\mu}_s = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathbf{x}_n^{(i)}$
b. Calculate the empirical covariance	$\Sigma_s = \frac{1}{N_s} \sum_{i=1}^{N_s} (\mathbf{x}_n^{(i)} - \boldsymbol{\mu}_s)^T (\mathbf{x}_n^{(i)} - \boldsymbol{\mu}_s)$
c. Resample the particles	Resample using one of the resampling Matlab example snippets
d. Regularize the resampled particles	Regularize the particles using the procedure from Table 15.3
e. Time step and return to B.1	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}; w_n^{(i)} = 1/N$
Else	
Time step and return to B.1	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}; w_n^{(i)} \rightarrow w_{n-1}^{(i)}$
end if	

approximation of the moment integrals, so all of the weights are reset to $w_n^{(i)} = 1/N$. At the end of each iteration, the moments can be calculated prior to stepping in time.

One can see that the SIS particle filter with resampling and regularization is still not in a form that can be applied to a tracking problem. We still have the problem of identifying a suitable importance distribution $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$ and the analytical form of the likelihood function $p(\mathbf{z}_n | \mathbf{x}_n)$.

16.3 THE BOOTSTRAP PARTICLE FILTER

One of the easiest to implement, and thus one of the most widely used, resampling SIS particle filters is the bootstrap particle filter (BPF) introduced in [12]. In the BPF, the transition density is chosen as the importance density, that is

$$q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) = p(\mathbf{x}_n | \mathbf{x}_{n-1}) \quad (16.19)$$

For this choice of importance density, the weight update equation (16.12) becomes

$$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} p\left(\mathbf{z}_n | \mathbf{x}_n^{(i)}\right) \quad (16.20)$$

The BPF has the distinctive feature that the incremental weights do not depend on the past trajectory of the particles but only on the conditional likelihood of the observation $p(\mathbf{z}_n | \mathbf{x}_n)$. For the BPF, sampling is very straightforward, with the state transition equation (5.1) used to generate new particles by merely transitioning the particle from the previous time step. This is usually followed by the resample and move steps. The complete procedure for the BPF is shown in Table 16.4. Once again, the first and second moments of the distribution can be calculated at the end of each time step.

One important point to note in Table 16.4 is the absence of the noise covariance matrix \mathbf{Q} in the state covariance computation. This is not needed here because samples from the noise term $\mathbf{v}_n^{(i)} \sim p(\mathbf{v}_n)$ are included in the transition equation for the state vector. Thus, there is also the unstated need for knowledge of the dynamic noise density. Since our original assumption was for an unknown posterior density, assuming an analytical form for the dynamic noise is a violation of our original assumption. The usual method to get around this is to ignore the noise term and transition the particles without adding the noise.

There still remains the need for an analytical expression for the likelihood function $p(\mathbf{z}_n | \mathbf{x}_n)$. As noted previously, $p(\mathbf{z}_n | \mathbf{x}_n)$ is usually taken to be Gaussian. But for highly nonlinear observation equations, the observation noise can be non-Gaussian, as shown in the DIFAR case study.

One of the main problems with application of the BPF to real-world problems occurs when $p(\mathbf{z}_n | \mathbf{x}_n)$ is a very narrow function, as would occur if the variances of the observation noise are small. When this occurs, many of the transitioned particles will produce very small values for $p(\mathbf{z}_n | \mathbf{x}_n^{(i)})$, making their corresponding updated weights very small. For such cases, the resample and move steps should be executed

TABLE 16.4 Bootstrap SIS Particle Filter with Resampling and Regularization

<i>A. Initialize filter</i>	
1. Initialize state vector samples	$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$
2. Initialize weights	$w_0^{(i)} = \frac{1}{N_s}$
<i>B. Sequential importance sampling</i>	
1. Draw new samples	$\mathbf{v}_n^{(i)} \sim p(\mathbf{v}_n)$ $\mathbf{x}_n^{(i)} = \mathbf{f}(\mathbf{x}_{n-1}^{(i)}) + \mathbf{v}_n^{(i)}, \forall i$
2. Generate unnormalized importance weights	$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} p(\mathbf{z}_n \mathbf{x}_n^{(i)})$
3. Normalize the weights	$w_n^{(i)} = \tilde{w}_n^{(i)} / \sum_{i=1}^{N_s} \tilde{w}_n^{(i)}$
<i>C. Calculate N_{eff}</i>	$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_n^{(i)})^2}$
<i>D. If $N_{\text{eff}} \ll N_s$</i>	
a. Calculate the empirical mean	$\boldsymbol{\mu}_s = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathbf{x}_n^{(i)}$
b. Calculate the empirical covariance	$\Sigma_s = \frac{1}{N_s} \sum_{i=1}^{N_s} (\mathbf{x}_n^{(i)} - \boldsymbol{\mu}_s) (\mathbf{x}_n^{(i)} - \boldsymbol{\mu}_s)^T$ $\Sigma_s = \mathbf{D}_s \mathbf{D}_s^T$
c. Resample the particles	Resample using one of the resampling Matlab example snippets
d. Regularize the resampled particles	Regularize the particles using the procedure from Table 15.3
e. Time step and return to B.1	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}; w_n^{(i)} = 1/N$
<i>Else</i>	
Time step and return to B.1	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}; w_n^{(i)} \rightarrow w_{n-1}^{(i)}$
<i>end if</i>	

for every update. On the other hand, if $p(\mathbf{z}_n | \mathbf{x}_n)$ is assumed to be Gaussian when in fact it is not, uninformative outliers can occur in the observations such that $p(\mathbf{z}_n | \mathbf{x}_n^{(i)})$ is zero for all particles resulting in an unstable filter.

16.3.1 Application of the BPF to DIFAR Buoy Tracking

In Chapter 4, we presented the likelihood function $p(\theta_{n,m} | \vartheta_m)$ for a DIFAR buoy (4.34). This likelihood function represented the likelihood of the m th DIFAR element bearing measurement at time t_n , $\theta_{n,m}$, conditioned on the whole space of possible target bearings $\{-\pi \leq \vartheta_m \leq \pi\}$ as a function of the signal SNR at the buoy input. We showed that the normalized likelihood was Gaussian for high SNR and approached a uniform distribution for low SNR. This should not be a surprising results since, in the absence of a signal (low SNR), the input to the buoy would be uniformly distributed

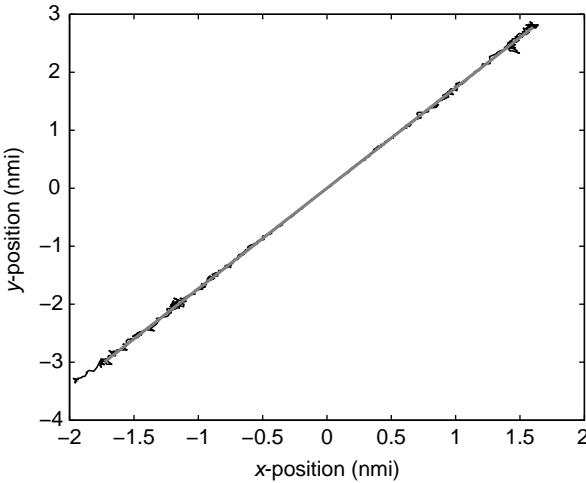


FIGURE 16.4 Target track generated from a DIFAR buoy field using a bootstrap particle filter.

noise over $-\pi \leq \theta_{n,m} \leq \pi$. Assuming that the noise is independent from buoy to buoy, we can write the buoy field likelihood function as the product of the individual buoy likelihood functions and use it in Step 2 of BPF process shown in Table 16.4.

To initialize the BPF we use the same procedure outlined in Section 7.4.3. After computing an estimate of the initial target state \mathbf{x}_0 , a set of initial particles $\{\mathbf{x}_0^{(i)}, i = 1, \dots, N_s\}$ are generated such that

$$\mathbf{x}_0^{(i)} = \mathbf{x}_0 + \mathbf{v}_n^{(i)} \quad (16.21)$$

with $\mathbf{v}_n^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. The remaining steps in Table 16.4 are then carried out sequentially to produce the estimated target track plotted against the truth track as shown in Figure 16.4. For this case, we left the SNR at 20 dB and initialized the range at 4 nmi and the speed at 30 knots (true initial range and speed were approximately 3.5 nmi and 30 knots, respectively). For this example, we set the number of particles to 1000.

The BPF track outputs for six different input signal SNRs (20 dB down to -5 dB in increments of 5 dB) with the same initialization as all of the previous DIFAR track estimation plots are presented in Figure 16.5. For this example, 3000 particles were used for the BPF. Comparison of a similar plot for the EKF shown in Figure 7.3 reveals that the EKF produces more accurate tracks than the BPF at all SNRs. In addition, the 100 BPF Monte Carlo runs produced divergent tracks at every SNR. Some of the reasons for this lack of performance for the BPF are discussed in the next paragraph.

Unlike the Gaussian Kalman filters, the performance of the BPF was highly sensitive to the range initialization value. If we initialized the range to any value greater

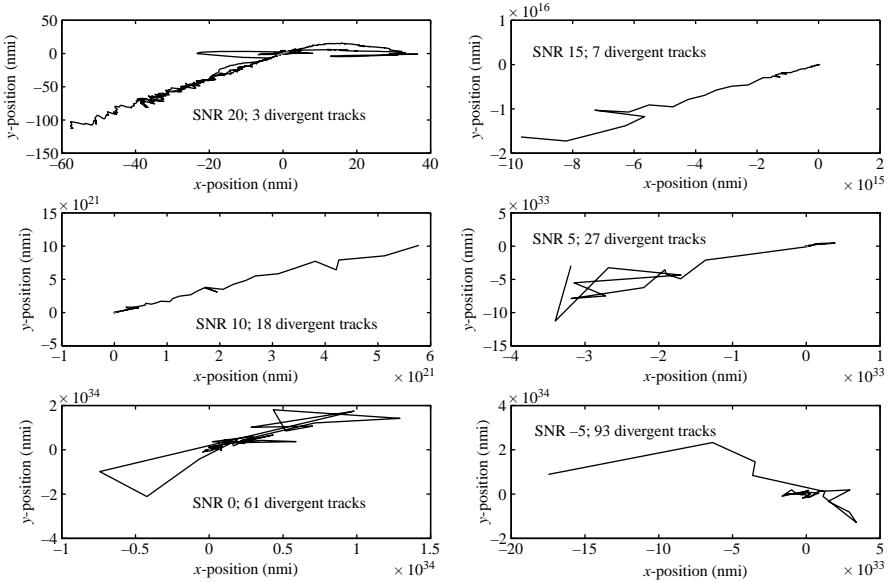


FIGURE 16.5 A comparison of track outputs at six different SNRs for the BPF tracker.

than 5 nmi, the BPF tracks diverged. Increasing the number of particles decreased this problem somewhat, but we have found that the BPF can be difficult to use because of this initialization sensitivity. There appeared to be little sensitivity to variations in the initial speed. We even tried initializing the particles by choosing uninformative uniformly distributed ranges and speeds, over the range 2–5.9 in increments of 0.1 nautical miles for range and 24–30 in increments of 0.5 knots for speed, with most of the ensuing track estimates highly divergent. Although rarely mentioned in the literature, this initialization sensitivity makes the BPF difficult to use in a real-world tracking scenario, unless an alternate method is used to generate “good” initializations. In addition, it must be noted that for the BPF algorithm, the importance density does not take into account the current observation, making the BPF a highly suboptimal filtering choice. Although very easy to implement, it has some serious deficiencies relative to some of the particle filter methods to be presented below.

16.4 THE OPTIMAL SIS PARTICLE FILTER

To overcome some of the problems of the BPF, one needs to choose the importance density more wisely. Thus, the choice of an importance density is the most critical issue in the design of a particle filter. The *optimal* choice for an importance density is to choose one that minimizes the variance of the weights. Consider the mean and

variance of the weights as follows:

$$\mathcal{E}_{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \left\{ w_n^{*(i)} \right\} = \int w_n^{*(i)} q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n) d\mathbf{x}_n \quad (16.22)$$

$$\begin{aligned} \text{Var}_{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \left\{ w_n^{*(i)} \right\} &= \mathcal{E}_{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \left\{ \left[w_n^{*(i)} \right]^2 \right\} \\ &\quad - \left[\mathcal{E}_{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \left\{ w_n^{*(i)} \right\} \right]^2 \end{aligned} \quad (16.23)$$

If we first examine the mean equation (16.22), substitution of (16.12), the recursion equation for $w_n^{*(i)}$, results in

$$\begin{aligned} \mathcal{E}_{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \left\{ w_n^{*(i)} \right\} &= w_{n-1}^{*(i)} \int p(\mathbf{z}_n|\mathbf{x}_n) p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}) d\mathbf{x}_n \\ &= w_{n-1}^{*(i)} p(\mathbf{z}_n|\mathbf{x}_{n-1}^{(i)}) \end{aligned} \quad (16.24)$$

Using the same procedure on the first term in the variance equation (16.23) and using (16.24) to evaluate the second term in (16.23), we obtain

$$\begin{aligned} \text{Var}_{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \left\{ w_n^{*(i)} \right\} &= \left[w_{n-1}^{*(i)} \right]^2 \left\{ \int \frac{\left[p(\mathbf{z}_n|\mathbf{x}_n) p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}) \right]}{q(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} d\mathbf{x}_n \right. \\ &\quad \left. - \left[p(\mathbf{z}_n|\mathbf{x}_{n-1}^{(i)}) \right]^2 \right\} \end{aligned} \quad (16.25)$$

It follows immediately from Bayes' law that (16.25) reduces identically to zero [13] if

$$q(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n) \quad (16.26)$$

$$= \frac{p(\mathbf{z}_n|\mathbf{x}_n, \mathbf{x}_{n-1}^{(i)}) p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)})}{p(\mathbf{z}_n|\mathbf{x}_{n-1}^{(i)})} \quad (16.27)$$

Now, the expression for the importance weight update (16.12) becomes

$$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} \frac{p(\mathbf{z}_n|\mathbf{x}_n^{(i)}) p(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)})}{p(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \quad (16.28)$$

$$= w_{n-1}^{(i)} p(\mathbf{z}_n|\mathbf{x}_{n-1}^{(i)}) \quad (16.29)$$

Comparing the BPF weight Equation (16.20) with that of the OPF (16.29) we see that the OPF importance weight $\tilde{w}_n^{(i)}$ no longer depends on $\mathbf{x}_n^{(i)}$. Thus, the importance weights at time t_n can be computed before the particles are propagated to time t_n .

Use of the optimal importance density suffers from two drawbacks [14]. First, one must be able to sample from the importance density $p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)$, and then one must be able to evaluate $p(\mathbf{z}_n | \mathbf{x}_{n-1}^{(i)})$ up to a normalization constant, where $p(\mathbf{z}_n | \mathbf{x}_{n-1}^{(i)})$ is given by the integral

$$p(\mathbf{z}_n | \mathbf{x}_{n-1}^{(i)}) = \int p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}) d\mathbf{x}_n \quad (16.30)$$

In general, either task may prove very difficult in the general case.

However, there are some interesting special cases where the optimal SIS particle filter can be used. As noted in Refs [3,15], the first special case occurs when \mathbf{x}_n is a member of a finite set. The integral in (16.30) then becomes a sum and sampling from $p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)$ then becomes possible. The example given for such a special case is that of a linear jump-Markov process applied to a maneuvering target. The second case, to be treated below, is a class of models for which $p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)$ is Gaussian [13,14].

16.4.1 Gaussian Optimal SIS Particle Filter

Consider the case where the system dynamic transition model is nonlinear with additive Gaussian noise and the observation equation is linear with additive Gaussian noise. Such a system can be characterized by

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{v}_{n-1} \quad (16.31)$$

$$\mathbf{z}_n = \mathbf{H}_n \mathbf{x}_n + \mathbf{w}_n \quad (16.32)$$

where

$$\mathbf{v}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_n) \quad (16.33)$$

$$\mathbf{w}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_n) \quad (16.34)$$

Now the densities $p(\mathbf{x}_n | \mathbf{x}_{n-1})$ and $p(\mathbf{z}_n | \mathbf{x}_n)$ can be identified as

$$p(\mathbf{x}_n | \mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n; \mathbf{f}(\mathbf{x}_{n-1}), \mathbf{Q}_{n-1}) \quad (16.35)$$

$$p(\mathbf{z}_n | \mathbf{x}_n) = \mathcal{N}(\mathbf{z}_n; \mathbf{H}_n \mathbf{x}_n, \mathbf{R}_n) \quad (16.36)$$

From Bayes' Law, we can write

$$p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) = \frac{p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1})}{p(\mathbf{z}_n | \mathbf{x}_{n-1})} \quad (16.37)$$

which results in the equality

$$p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) = p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) p(\mathbf{z}_n | \mathbf{x}_{n-1}) \quad (16.38)$$

Since the product on the left is a product of Gaussian distributions, the product on the right must also be a product of Gaussians. Therefore, let

$$p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) = \mathcal{N}(\mathbf{x}_n; \mathbf{a}_n, \Sigma_n) \quad (16.39)$$

$$p(\mathbf{z}_n | \mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{z}_n; \mathbf{b}_n, \mathbf{S}_n) \quad (16.40)$$

Now (16.38) can be written in terms of the Gaussian densities as

$$\mathcal{N}(\mathbf{z}_n; \mathbf{H}_n \mathbf{x}_n, \mathbf{R}_n) \mathcal{N}(\mathbf{x}_n; \mathbf{f}(\mathbf{x}_{n-1}), \mathbf{Q}_{n-1}) = \mathcal{N}(\mathbf{z}_n; \mathbf{b}_n, \mathbf{S}_n) \mathcal{N}(\mathbf{x}_n; \mathbf{a}_n, \Sigma_n) \quad (16.41)$$

Equating the terms in the exponents of the analytical Gaussian densities, it follows (after much algebra) that

$$\mathbf{b}_n = \mathbf{H}_n \mathbf{f}(\mathbf{x}_{n-1}) \quad (16.42)$$

$$\mathbf{S}_n = \mathbf{H}_n \mathbf{Q}_{n-1} \mathbf{H}_n^\top + \mathbf{R}_n \quad (16.43)$$

$$\mathbf{a}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \Sigma_n \mathbf{H}_n^\top \mathbf{R}_n^{-1} (\mathbf{z}_n - \mathbf{b}_n) \quad (16.44)$$

$$\Sigma_n = \mathbf{Q}_{n-1} - \mathbf{Q}_{n-1} \mathbf{H}_n^\top \mathbf{S}_n^{-1} \mathbf{H}_n \mathbf{Q}_{n-1} \quad (16.45)$$

Note that if \mathbf{Q}_{n-1} , \mathbf{R}_n , and \mathbf{H}_n are independent of time, then \mathbf{S}_n and Σ_n are also time independent and can be computed offline.

Remembering that this is a particle filter, we can insert the particles $\{\mathbf{x}_{n-1}^{(i)}, i = 1, \dots, N\}$ into (16.42) and (16.44) and thus we must let $\mathbf{b}_n \rightarrow \mathbf{b}_n^{(i)}$ and $\mathbf{a}_n \rightarrow \mathbf{a}_n^{(i)}$. The complete procedure for the OPF is shown in Table 16.5. In the table, we have used the notation $\mathbf{x}_{n|n-1}^{(i)}$ to indicate the predicted particle in the State Prediction and Intermediate calculations steps. Also, in the Intermediate calculations step, \mathbf{z}_n represents the observations at time t_n . Since this is a SIS particle filter, resampling is required and we have assumed that resampling occurs at every time step. This can be modified to only resample when $N_{\text{eff}} \ll N_s$ if desired. Note that the importance weights at time t_n are calculated in the weight update step *before* the particles are propagated forward in the importance sampling step. In addition, the old particles can also be resampled and the weights reset to $1/N_s$ before particles are propagated forward in time.

16.4.2 Locally Linearized Gaussian Optimal SIS Particle Filter

When both the dynamic and observation equations are nonlinear, the OPF can still be used if the nonlinear observation equation is linearized in some way [15]. Consider an observation equation of the form

$$\mathbf{z}_n = \mathbf{h}(\mathbf{x}_n) + \mathbf{w}_n \quad (16.46)$$

Expanding $\mathbf{h}(\mathbf{x}_n)$ in a multidimensional Taylor series about $\mathbf{f}(\mathbf{x}_{n-1})$ using (2.62) results

$$\mathbf{h}(\mathbf{x}_n) \simeq \mathbf{h}(\mathbf{f}(\mathbf{x}_{n-1})) + \hat{\mathbf{H}}_n (\mathbf{x}_n - \mathbf{f}(\mathbf{x}_{n-1})) \quad (16.47)$$

TABLE 16.5 Optimal SIS Particle Filter with Resampling and Regularization

<i>A. Initialize filter</i>	
1. Initialize state vector samples	$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$
2. Initialize weights	$w_0^{(i)} = \frac{1}{N_s}$
<i>B. State prediction</i>	
	$\mathbf{x}_{n n-1}^{(i)} = \mathbf{f}(\mathbf{x}_{n-1}^{(i)})$
	$\mathbf{b}_n^{(i)} = \mathbf{H}_n \mathbf{x}_{n n-1}^{(i)}$
	$\mathbf{S}_n = \mathbf{H}_n \mathbf{Q}_{n-1} \mathbf{H}_n^\top + \mathbf{R}_n$
<i>C. Weights update</i>	
a. Likelihood function	$p(\mathbf{z}_n \mathbf{x}_{n-1}^{(i)}) = \mathcal{N}(\mathbf{z}_n; \mathbf{b}_n^{(i)}, \mathbf{S}_n)$
b. Weight update	$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} p(\mathbf{z}_n \mathbf{x}_{n-1}^{(i)})$
c. Weight normalization	$w_n^{(i)} = \tilde{w}_n^{(i)} / \sum_{i=1}^{N_s} \tilde{w}_n^{(i)}$
<i>D. Resample the particles</i>	Resample using one of the resampling Matlab example snippets
<i>E. Move the resampled particles</i>	Regularize the particles using the procedure from Table 15.3
<i>Reset the weights</i>	$w_n^{(i)} = 1/N$
<i>F. Importance sampling</i>	
a. Intermediate calculations	$\mathbf{a}_n^{(i)} = \mathbf{x}_{n n-1}^{(i)} + \Sigma_n \mathbf{H}_n^\top \mathbf{R}_n^{-1} (\mathbf{z}_n - \mathbf{b}_n^{(i)})$ $\Sigma_n = \mathbf{Q}_{n-1} - \mathbf{Q}_{n-1} \mathbf{H}_n^\top \mathbf{S}_n^{-1} \mathbf{H}_n \mathbf{Q}_{n-1}$
b. Importance sampling	$\mathbf{x}_n^{(i)} \sim p(\mathbf{x}_n \mathbf{x}_{n-1}, \mathbf{z}_n) = \mathcal{N}(\mathbf{x}_n; \mathbf{a}_n^{(i)}, \Sigma_n)$
<i>G. Moment calculations</i>	
a. Mean	$\hat{\mathbf{x}}_n = \sum_i w_n^{(i)} \mathbf{x}_n^{(i)}$
b. Covariance	$\mathbf{P}_n^{\mathbf{xx}} = \sum_i w_n^{(i)} \mathbf{x}_n^{(i)} \mathbf{x}_n^{(i)\top} - \hat{\mathbf{x}}_n \hat{\mathbf{x}}_n^\top$
<i>H. Time step and return to B</i>	$\mathbf{x}_n^{(i)} \rightarrow \mathbf{x}_{n-1}^{(i)}$

where the Jacobian $\hat{\mathbf{H}}_n$ is defined by

$$\hat{\mathbf{H}}_n \triangleq [\nabla_{\mathbf{x}_n}^\top \mathbf{h}(\mathbf{x}_n)]_{\mathbf{x}_n=\mathbf{f}(\mathbf{x}_{n-1})}^\top \quad (16.48)$$

Now (16.46) becomes

$$\tilde{\mathbf{z}}_n \simeq \hat{\mathbf{H}}_n \mathbf{x}_n + \mathbf{w}_n \quad (16.49)$$

where

$$\tilde{\mathbf{z}}_n \triangleq \mathbf{z}_n + \hat{\mathbf{H}}_n \mathbf{f}(\mathbf{x}_{n-1}) - \mathbf{h}(\mathbf{f}(\mathbf{x}_{n-1})) \quad (16.50)$$

Thus, we have turned the observation equation into one that is linear in \mathbf{x}_n at the expense of adding additional terms to the observation, that is, $\mathbf{z}_n \rightarrow \tilde{\mathbf{z}}_n$. We can now

use the procedure in Table 16.5 by simply letting $\mathbf{H}_n \rightarrow \hat{\mathbf{H}}_n$ and $\mathbf{z}_n \rightarrow \tilde{\mathbf{z}}_n$, with the additional steps of calculating the Jacobian $\hat{\mathbf{H}}_n$ and the additional terms in $\tilde{\mathbf{z}}_n$ and we make the replacement $\mathbf{x}_n \rightarrow \mathbf{x}_n^{(i)}$ and $\mathbf{x}_{n-1} \rightarrow \mathbf{x}_{n-1}^{(i)}$ in (16.49) and (16.50).

Now, the OPF can be applied to any tracking problem that has nonlinear dynamic and observation equations as long as all densities are Gaussian. For most tracking problems with these characteristics, nothing is gained by using the OPF over any of the methods presented in Part II of this book. However, there are instances where one would like to apply constraints on the track solutions, such as limiting the track estimates to a predetermined road grid or limiting the maneuverability or speed of the target states. It is for these type of problems that the OPF will be most amenable to solving the tracking problem. Since our DIFAR tracking case study does not have any restrictions on the target state, we will not apply the locally linearized Gaussian optimal SIS particle filter to the case study.

16.5 THE SIS AUXILIARY PARTICLE FILTER

The use of a well chosen SIS proposal distribution, $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$, should ensure that the current observation \mathbf{z}_n is incorporated into the proposal procedure so that particles are not moved blindly into regions of the state space that are unlikely given that observation. Obviously, the BPF fails to achieve this objective since the current observation is not considered when $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) \stackrel{\text{BPF}}{=} p(\mathbf{x}_n | \mathbf{x}_{n-1})$. As we discussed previously, the OPF chooses $q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n) \stackrel{\text{OPF}}{=} p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{z}_n)$, but it is difficult to implement except in special cases. The SIS auxiliary particle filter (APF) is a particle filter that incorporates the current observations into the proposal distribution in a suboptimal way.

In the standard resampling SIS particle filter, including the BPF, the particles are resampled *after* the particles are drawn from the importance density and the weights have been updated by the current observation. In the OPF, on the other hand, resampling takes place *before* the particles are sampled from the importance density. This can be verified by comparing Table 16.5 with either Table 16.2 or Table 16.4. In Ref. [16] and Chapter 13 of Ref. [17], Pitt and Shephard proposed the APF as a variant of the standard SIS where the resampling step is performed at time t_{n-1} using the observation from time t_n . In this way, the APF attempts to mimic the method used by the optimal particle filter. The clearest description of the APF procedure can be found in Refs [7,8]. Much of the material presented in this section is taken directly from Ref. [7].

The idea behind the APF is to augment the particles $\mathbf{x}_{n-1}^{(i)}$ that have large weights (the “good” particles) in the sense that the *predictive* likelihoods $p(\mathbf{z}_n | \mathbf{x}_{n-1}^{(i)})$ are large for these particles. From Bayes’ Law (3.23), the posterior density can be written as

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) \propto p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) \quad (16.51)$$

Using the Chapman–Kolmogorov equation (3.24) this becomes

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) \propto p(\mathbf{z}_n | \mathbf{x}_n) \int p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \quad (16.52)$$

If

$$p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) = \sum_{i=1}^{N_s} w_{n-1}^{(i)} \delta(\mathbf{x}_{n-1} - \mathbf{x}_{n-1}^{(i)}) \quad (16.53)$$

Then

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) \propto \left[\sum_{i=1}^{N_s} w_{n-1}^{(i)} p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}) \right] p(\mathbf{z}_n | \mathbf{x}_n) \quad (16.54)$$

The product $w_{n-1}^{(i)} p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})$ is treated as a combined probability that contributes to the filtered density sum.

One way to sample from the empirical prediction density is to think of $\sum_i w_{n-1}^{(i)} p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})$ as a “prior” *mixture* density $\hat{p}(\mathbf{x}_n | \mathbf{z}_{1:n-1})$ that is combined with the likelihood $p(\mathbf{z}_n | \mathbf{x}_n)$ to produce a posterior. Samples $\mathbf{x}_n^{(i)}$ can be drawn from $\hat{p}(\mathbf{x}_n | \mathbf{z}_{1:n-1})$ by choosing $\mathbf{x}_n^{(i)}$ with probability $w_{n-1}^{(i)}$ and then drawing from $p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})$.

To understand the APF method, we first examine the SIS update equation (16.12) before sampling from $q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)$ and rewrite it as

$$\tilde{w}_n = w_{n-1}^{(i)} \frac{p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})}{q(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)} \quad (16.55)$$

Now introduce an auxiliary variable ξ ($\xi \in \{1, \dots, N_s\}$) that indexes into one component of the “prior” mixture density. Define the joint conditional indexed posterior density $p(\mathbf{x}_n, \xi = i | \mathbf{z}_{1:n})$ by

$$\begin{aligned} p(\mathbf{x}_n, \xi = i | \mathbf{z}_{1:n}) &\triangleq p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n, \xi = i | \mathbf{z}_{1:n-1}) \\ &= p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \xi = i, \mathbf{z}_{1:n-1}) p(i | \mathbf{z}_{1:n-1}) \end{aligned} \quad (16.56)$$

Comparison with (16.54) allows us to identify

$$p(\mathbf{x}_n | \xi = i, \mathbf{z}_{1:n-1}) = p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}) \quad (16.57)$$

$$p(i | \mathbf{z}_{1:n-1}) = w_{n-1}^{(i)} \quad (16.58)$$

and we obtain the posterior probability of \mathbf{x}_n given $\{\mathbf{z}_n, \mathbf{x}_{n-1}^{(i)}, w_{n-1}^{(i)}\}$

$$\beta_n^{(i)} \triangleq p(\mathbf{x}_n, \xi = i | \mathbf{z}_{1:n}) = p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}) w_{n-1}^{(i)} \quad (16.59)$$

Samples can be drawn from the joint density (16.59) by neglecting ξ . Assume that a set of particles $\{\mathbf{x}_n^{(i)}\}_{i=1}^{N_s}$ are drawn from the marginalized density $p(\mathbf{x}_n|\mathbf{z}_{1:n})$ and the indices ξ are simulated with probabilities proportional to $p(\xi|\mathbf{z}_{1:n-1}) = w_{n-1}^{(i)}$. Now, (16.54) can be approximated by

$$p(\mathbf{x}_n|\mathbf{z}_{1:n}) \propto \sum_{i=1}^{N_s} w_{n-1}^{(i)} p\left(\mathbf{z}_n|\mathbf{x}_n, \xi^{(i)}\right) p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}\right) \quad (16.60)$$

where $\xi^{(i)}$ denotes the index of the particle $\mathbf{x}_n^{(i)}$ at time step t_{n-1} (namely $\xi^{(i)} \triangleq \{\xi = i\}$).

The proposal distribution used to draw samples $\{\mathbf{x}_n^{(i)}, i = 1, \dots, N_s\}$ is chosen as a factorized form

$$q(\mathbf{x}_n, \xi|\mathbf{z}_{1:n}) \propto q(\xi|\mathbf{z}_{1:n}) q(\mathbf{x}_n|\xi, \mathbf{z}_{1:n}) \quad (16.61)$$

where

$$q\left(\xi^{(i)}|\mathbf{z}_{1:n}\right) = w_{n-1}^{(i)} p\left(\mathbf{z}_n|\boldsymbol{\mu}_n^{(i)}\right) \quad (16.62)$$

$$q\left(\mathbf{x}_n|\xi^{(i)}, \mathbf{z}_{1:n}\right) = p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}\right) \quad (16.63)$$

where $\boldsymbol{\mu}_n^{(i)}$ is a value (e.g., mean, mode, or sample value) associated with $p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)})$ from which the i th particle is drawn.

Thus, the true posterior is further approximated by

$$p(\mathbf{x}_n|\mathbf{z}_{1:n}) \propto \sum_{i=1}^{N_s} w_{n-1}^{(i)} p\left(\mathbf{z}_n|\boldsymbol{\mu}_n^{(\xi=i)}\right) p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(\xi=i)}\right) \quad (16.64)$$

Now the unnormalized weights from (16.55) become

$$\begin{aligned} \tilde{w}_n^{(i)} &= w_{n-1}^{(\xi=i)} \frac{p\left(\mathbf{z}_n|\mathbf{x}_n^{(i)}\right) p\left(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(\xi=i)}\right)}{q\left(\mathbf{x}_n^{(i)}, \xi^{(i)}|\mathbf{z}_{1:n}\right)} \\ &= \frac{p\left(\mathbf{z}_n|\mathbf{x}_n^{(i)}\right)}{p\left(\mathbf{z}_n|\boldsymbol{\mu}_n^{(\xi=i)}\right)} \end{aligned} \quad (16.65)$$

The process flow for the auxiliary particle filter is shown in Table 16.6. To sum up the process flow of the SIS APF, part B amounts to using a bootstrap particle filter procedure to calculate a new set of weights and then resampling and regularizing the particle set $\{\mathbf{x}_{n-1}^{(i)}, i = 1, \dots, N_s\}$ using these new weights. This is followed in Part E with a propagation of the resampled particles forward in time and computation of a new set of weights using (16.65) with normalization.

TABLE 16.6 Auxiliary Particle Filter Process Flow

A. Initialize filter	$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$
1. Initialize state vector samples	
2. Initialize weights	$w_0^{(i)} = \frac{1}{N_s}$
B. Calculate	$\boldsymbol{\mu}_n^{(i)}$ mean or draw from $p(\mathbf{x}_n \mathbf{x}_{n-1}^{(i)}), \forall i$
	$\tilde{\beta}_n^{(i)} = w_{n-1}^{(i)} p(\mathbf{z}_n \boldsymbol{\mu}_n^{(i)}), \forall i$
C. Normalize the weights	$\beta_n^{(i)} = \tilde{\beta}_n^{(i)} / \sum_{i=1}^{N_s} \tilde{\beta}_n^{(i)}$
D. Resample $\mathbf{x}_{n-1}^{(i)}$ using $\beta_n^{(i)}$	Resample $\mathbf{x}_{n-1}^{(i)}$ using one of the resampling Matlab example snippets using $\beta_n^{(i)}$ instead of $w_{n-1}^{(i)}$
Move the resampled particles	Regularize the particles using the procedure from Table 15.3
E. Importance sampling	$\mathbf{x}_n^{(i)} = p(\mathbf{x}_n \mathbf{x}_{n-1}^{(i)}, \xi^{(i)})$
	$w_n^{(i)} = \frac{p(\mathbf{z}_n \mathbf{x}_n^{(i)})}{p(\mathbf{z}_n \boldsymbol{\mu}_n^{(i)})}$
F. Normalize	$w_n^{(j)} = \tilde{w}_n^{(j)} / \sum_{i=1}^{N_s} \tilde{w}_n^{(j)}$
G. Time step and return to B	$\mathbf{x}_n^{(j)} \rightarrow \mathbf{x}_{n-1}^{(i)}; w_n^{(j)} \rightarrow w_{n-1}^{(i)}$

The APF is essentially a two-stage procedure: At the first stage, simulate the particles with large predictive likelihoods; at the second stage, reweight the particles and draw augmented states. This is equivalent to making a proposal that has a high conditional likelihood *a priori*, thereby avoiding inefficient sampling. Thus, the APF takes advantage beforehand of the information from the likelihood model to avoid inefficient sampling since the particles with low likelihood are less informative. In other words, the particles to be sampled are intuitively pushed to the higher likelihood region.

Some remarks about the APF are as follows:

- In conventional SIS particle filters, estimation is usually performed *after* the resampling step, which is less efficient because resampling introduces extra random variations in the current state. The APF overcomes this problem by doing one-step ahead estimation based on the point estimate $\boldsymbol{\mu}_n^{(i)}$ that characterizes $p(\mathbf{x}_n | \mathbf{x}_n^{(i)})$.
- When process noise is small, the performance of the APF is usually better than that of the SIS filters. However, when process noise is large, the point estimate $\boldsymbol{\mu}_n^{(i)}$ doesn't provide sufficient information about $p(\mathbf{x}_n | \mathbf{x}_n^{(i)})$ and the superiority of the APF is not guaranteed.

- In the APF, the importance density is proposed as a mixture density that depends on the past state and the most recent observation.
- The idea of the APF is identical to that of the local Monte Carlo method proposed in Ref. [18], where the authors proposed two methods for sample draws $\{\mathbf{x}, \xi\}$, based on either joint or marginal distributions.
- The disadvantage of the APF is that the sampling is drawn from an augmented (thus higher) space. If the auxiliary index varies a lot for a fixed prior, the gain is negligible and the variance of the importance weights will be higher.

16.5.1 Application of the APF to DIFAR Buoy Tracking

When the APF is used as the tracking filter for the DIFAR case study, it was hoped that the results would show an improvement over the Gaussian Kalman filters. But one glance at the resulting Monte Carlo tracks for the six gradually decreasing SNRs, shown in Figure 16.6 reveals that the APF does not perform any better. In hindsight, the reason for this is almost obvious. Particle filters are designed to perform estimation when the posterior density (read *dynamic model*) is non-Gaussian. But for the DIFAR case study it is the likelihood function that is non-Gaussian while the dynamic model is linear and Gaussian. So, as we found out, the APF does not outperform the Gaussian Kalman filters under these conditions.

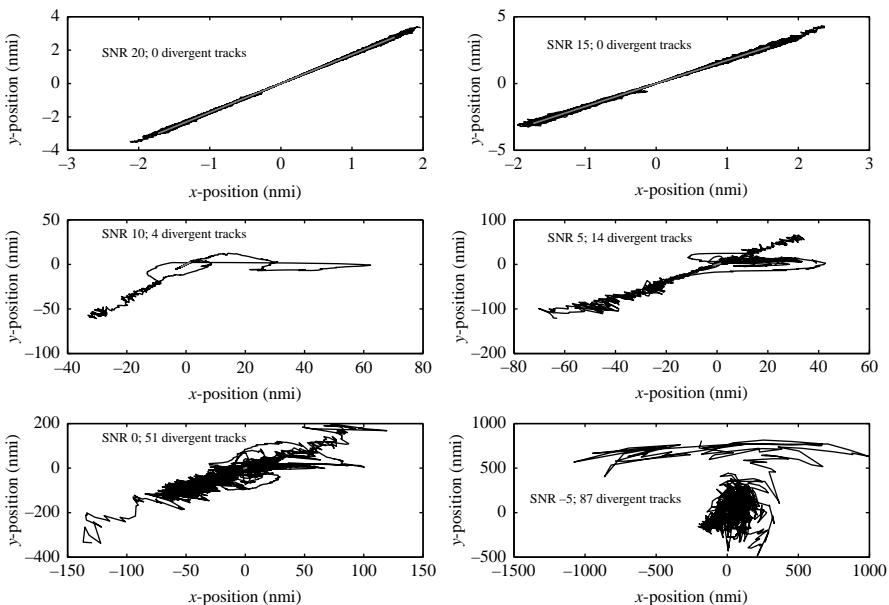


FIGURE 16.6 Track estimation results for the auxiliary particle filter applied to the DIFAR tracking case study.

16.6 APPROXIMATIONS TO THE SIS AUXILIARY PARTICLE FILTER

An alternate approach to the SIS particle filter has been proposed that is similar to the APF presented above. Suppose the posterior density $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ and the optimal importance density $q(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_{1:n})$ are approximated by Gaussian densities. That is, the posterior density is approximated as

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) \simeq \mathcal{N}\left(\mathbf{x}_n; \bar{\mathbf{x}}_n, \hat{\mathbf{P}}_n\right) \quad (16.66)$$

and the proposal distribution is approximated by a Gaussian proposal distribution for each particle

$$q\left(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_{1:n}\right) \simeq \mathcal{N}\left(\mathbf{x}_n^{(i)}; \bar{\mathbf{x}}_n^{(i)}, \hat{\mathbf{P}}_n^{(i)}\right) \quad (16.67)$$

That is, at time t_{n-1} , one uses one of the Gaussian Kalman filter variations from Part II of this book, along with the new observation, to compute the mean and covariance of the importance distribution for each particle. Then one can sample the i th particle from this distribution.

16.6.1 The Extended Kalman Particle Filter

In the extended Kalman particle filter (EKPF), first proposed by Merwe et al. [19], both the nonlinear dynamic and observation equations are expanded in Taylor series and used in an EKF to generate the first two moments for the i th particle $(\bar{\mathbf{x}}_n^{(i)}, \hat{\mathbf{P}}_n^{(i)})$ used in (16.67) to defined the Gaussian proposal distribution. Then, the i th sample is drawn from the proposal distribution

$$\hat{\mathbf{x}}_n^{(i)} = q\left(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_{1:n}\right) = \mathcal{N}\left(\mathbf{x}_n^{(i)}; \bar{\mathbf{x}}_n^{(i)}, \hat{\mathbf{P}}_n^{(i)}\right) \quad (16.68)$$

Now, $\hat{\mathbf{x}}_n^{(i)}$ is used to update the weights using (16.12),

$$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} \frac{p\left(\mathbf{z}_n | \hat{\mathbf{x}}_n^{(i)}\right) p\left(\hat{\mathbf{x}}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}\right)}{q\left(\hat{\mathbf{x}}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n\right)} \quad (16.69)$$

and the weights are normalized. Since this method is almost identical to the Gaussian optimal SIS particle filter presented above, the reader is referred to Ref. [19] for further information on the specific steps associated with this method.

16.6.2 The Unscented Particle Filter

For highly nonlinear systems, the UKF has been shown to be more accurate than the EKF, so in Ref. [19], the unscented particle filter (UPF) was proposed. It replaced the EKF with the UKF for the generation of the first two moments used in the Gaussian proposal distribution. The UPF method is straightforward and easy to implement, as

shown in Table 16.7. Table 13.2 should be used for the calculation of sigma point weights. In addition, some of the remaining steps have been omitted from the table because they are the same as the previous particle filters. Additional information related to the UPF can be found in Refs [20,21]. A square root version of the UPF can be found in Ref. [22].

TABLE 16.7 Unscented Particle Filter with Resampling and Regularization

A. Initialize filter

1. Initialize state vector samples
2. Initialize weights

$$\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0), i = 1, \dots, N_s$$

$$w_0^{(i)} = \frac{1}{N_s}$$

B. Importance sampling

For $i = 1, \dots, N_s$

- a. Calculate the j Sigma points

$$\chi_{n-1|n-1}^{(i,j)} = \bar{\mathbf{x}}_{n-1|n-1}^{(i)}$$

$$+ \sqrt{\frac{n_x}{1-w_0}} \mathbf{P}_{n-1|n-1}^{\mathbf{xx}(i)} \mathbf{r}^{(j)}, j = 0, 1, \dots, 2n_x$$

$$\hat{\mathbf{x}}_{n|n-1}^{(i)} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\chi_{n-1|n-1}^{(i,j)})$$

$$\mathbf{P}_{n|n-1}^{\mathbf{xx}(i)} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\chi_{n-1|n-1}^{(i,j)}) \mathbf{f}^\top(\chi_{n-1|n-1}^{(i,j)})$$

$$- \hat{\mathbf{x}}_{n|n-1}^{(i)} \hat{\mathbf{x}}_{n|n-1}^{\top(i)} + \mathbf{Q}$$

- b. State prediction

$$\chi_{n|n-1}^{(i,j)} = \hat{\mathbf{x}}_{n|n-1}^{(i)}$$

$$+ \sqrt{\frac{n_x}{1-w_0}} \mathbf{P}_{n|n-1}^{\mathbf{xx}(i)} \mathbf{r}^{(j)}, j = 0, 1, \dots, 2n_x$$

$$\hat{\mathbf{z}}_{n|n-1}^{(i)} = \sum_{j=0}^{2n_x} w_j \mathbf{h}(\chi_{n|n-1}^{(i,j)})$$

$$\mathbf{P}_{n|n-1}^{\mathbf{zz}(i)} = \sum_{j=0}^{2n_x} w_j \mathbf{h}(\chi_{n|n-1}^{(i,j)}) \mathbf{h}^\top(\chi_{n|n-1}^{(i,j)})$$

$$- \hat{\mathbf{z}}_{n|n-1}^{(i)} \hat{\mathbf{z}}_{n|n-1}^{\top(i)} + \mathbf{R}$$

- c. Observation prediction

$$\mathbf{P}_{n|n-1}^{\mathbf{xz}(i)} = \sum_{j=0}^{2n_x} w_j \mathbf{f}(\chi_{n-1|n-1}^{(i,j)}) \mathbf{h}^\top(\chi_{n-1|n-1}^{(i,j)})$$

$$- \hat{\mathbf{x}}_{n|n-1}^{(i)} \hat{\mathbf{z}}_{n|n-1}^{\top(i)}$$

- d. Kalman filter update

$$\hat{\mathbf{x}}_{n|n}^{(i)} = \hat{\mathbf{x}}_{n|n-1}^{(i)} + \mathbf{K}_n^{(i)} (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}^{(i)})$$

$$\mathbf{P}_{n|n}^{\mathbf{xx}(i)} = \mathbf{P}_{n|n-1}^{\mathbf{xx}(i)} - \mathbf{K}_n^{(i)} \mathbf{P}_{n|n-1}^{\mathbf{zz}(i)} \mathbf{K}_n^{\top(i)}$$

$$\tilde{w}_n^{(i)} = w_{n-1}^{(i)} \frac{p(\mathbf{z}_n | \hat{\mathbf{x}}_n^{(i)}) p(\hat{\mathbf{x}}_n^{(i)} | \mathbf{x}_{n-1}^{(i)})}{q(\hat{\mathbf{x}}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{z}_n)}$$

$$w_n^{(j)} = \tilde{w}_n^{(j)} / \sum_{i=1}^{N_s} \tilde{w}_n^{(i)}$$

E. Update the importance weights

F. Resample and move

G. Moment Calculations

H. Time step and return to B

Note that the UKF is only one of the class of sigma point Kalman filters that could be used in this application. We could just as easily develop a spherical simplex or Gauss–Hermite SIS particle filter by substituting the weights and sigma points for those filters into Table 16.7. In fact, for a high-dimensional state vector, using the spherical simplex Kalman filter in place of the unscented Kalman filter would be a good idea. On the other hand, for low-dimensional state vectors and highly nonlinear models, substituting the Gauss–Hermite Kalman filter would make more sense. For completeness, one could even use the Monte Carlo Kalman filter for this application, without any loss of generality.

16.7 REDUCING THE COMPUTATIONAL LOAD THROUGH RAO-BLACKWELLIZATION

In many state estimation applications where the state vector is of high dimensionality, the computational load for a SIS particle filter can become extremely costly and the estimation accuracy may deteriorate rapidly. Particle filter-based methods become quite inefficient when applied to a high-dimensional state space since prohibitively large number of samples may be required to approximate the underlying density functions to the desired accuracy. When the components of the state vector can be subdivided into two classes, one that follows a linear temporal transition and is Gaussian and a second class that is non-Gaussian, then the computational load can be reduced using what is known as Rao–Blackwellization [23–25].

The key idea of the Rao–Blackwell method is a dimensional reduction that makes use of the structure of the models to split the conditional posterior into two separate parts, with the Gaussian part conditioned on the non-Gaussian part. One first uses a particle filter to generate the non-Gaussian posterior and then uses a Kalman filter on the conditioned Gaussian part. Assume that the state vector components can be divided into two groups $\mathbf{x}_n = [\mathbf{q}^T, \mathbf{r}^T]^T$ such that

$$\begin{aligned} p(\mathbf{x}_n | \mathbf{z}_{1:n}) &= p(\mathbf{q}_n, \mathbf{r}_n | \mathbf{z}_{1:n}) \\ &= p(\mathbf{q}_n | \mathbf{r}_n, \mathbf{z}_{1:n}) p(\mathbf{r}_n | \mathbf{z}_{1:n}) \end{aligned} \quad (16.70)$$

In Ref. [24] it was pointed out that $p(\mathbf{q}_n | \mathbf{r}_n, \mathbf{z}_{1:n})$ can be efficiently updated using a linear Kalman filter when the initial uncertainty for \mathbf{q}_n is Gaussian and the conditional probabilities of the observation model and system dynamics for \mathbf{q}_n are Gaussian. For utilization of the Rao–Blackwell methodology, the reader is referred to Refs [3,24,25].

REFERENCES

1. Ripley BD. *Stochastic Simulation*. Wiley; 1987.
2. Online at: http://en.wikipedia.org/wiki/Inverse_transform_sampling.
3. Ristic B, Arulampalam S, Gordon N. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House; 2004.
4. Andrieu C, Doucet A, Holenstein R. Particle Markov chain Monte Carlo methods. *J. R. Stat. Soc. B*. 2010;72(3):269–324.

5. Cappé O, Godsill SJ, Moulines E. An overview of existing methods and recent advances in sequential Monte Carlo. *Proc. IEEE* 2007;95(5):899–924.
6. Liu JS, Chen R. Sequential Monte Carlo methods for dynamic systems. *J. Am. Stat. Assoc.* 1998;93(443):1032–1044.
7. Chen Z. Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond. Adaptive Systems Laboratory, McMaster University, Hamilton, ON, Canada. [Online], <http://citeseerx.ist.psu.edu>; 2003.
8. Fearnhead P. Sequential Monte Carlo Methods in Filter Theory. Dissertation. Oxford UK: University of Oxford, MertonCollege; 1998.
9. Ross SM. *Introduction to Probability Models*, 4th ed. Academic Press; 1989.
10. Hol DH, Schön TB, Gustafsson F. On resampling algorithms for particle filters. In *Proceedings of Nonlinear Statistical Signal Processing Workshop*, Cambridge, UK; September 2006.
11. Kong A, Liu JS, Wong WH. Sequential computations and Bayesian missing data problems. *J. Am. Stat. Assoc.* 1994;89(425):278–288.
12. Gordon N, Salmond D, Smith AF. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proc. F. Radar Signal Process.* 1993;140:107–113.
13. Doucet A. Monte Carlo Methods for Bayesian Estimation of Hidden Markov Models. Application to Radiation Signals, Dissertation. University of Paris-Sud, Orsay; 1997.
14. Doucet A. On Sequential Simulation-Based methods for Bayesian Filtering. Technical Report CUED/F-INFENG/TR.310, University of Cambridge; 1998.
15. Doucet A, Gordon N, Krishnamurthy V. Particle filters for state estimation of jump Markov linear systems. *IEEE Trans. Sig. Proc.* 2001;49:613–624.
16. Pitt K, Shephard N. Filtering via simulation: auxiliary particle filter. *J. Am. Stat. Assoc.* 1999;94(446):590–599.
17. Oitt MK, Shephard N. Auxiliary variable based particle filters. In *Sequential Monte Carlo Methods in Practice*. Springer; 2001.
18. Wu W, Hu X, Hu D, Wu M. Comments on “Gaussian Particle Filtering”. *IEEE Trans. Sig. Proc.* 2005;53(8):3350–3351.
19. van der Merwe R, Doucet A, de Freitas N, Wam E. The Unscented Particle Filter. Cambridge University Technical Report CUED/F-INFENG/TR 380; 2000.
20. van der Merwe R, Doucet A, de Freitas N, Wam E. The unscented particle filter. *Adv. Neural Inform. Process. Syst.* 2001;13:584–590.
21. Rui Y, Chen Y. Better Proposal distributions: object tracking using unscented particle filter. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR 2001; Vol. II, 2001, pp. 786–793.
22. Zandara S, Nicholson AE. Square Root Unscented Particle Filtering for Grid Mapping. *6th World Congress on Intelligent Control and Automation*; 2006, pp. 1548–1552.
23. Bolviken E, Storik G. Deterministic and Stochastic Particle Filters for State-Space Models. In *Sequential Monte Carlo Methods in Practice*. Springer; 2001.
24. Murphy K, Russell S. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Sequential Monte Carlo Methods in Practice*. Springer; 2001.
25. Casella C, Rober CP. Rao-Blackwellisation of sampling schemes. *Biometrika* 1996;83(1):81–94.

17

THE GENERALIZED MONTE CARLO PARTICLE FILTER

All of the SIS class of particle filters discussed in the previous chapter are based on (16.12), which defines the recursive update of the importance weights. The combination of this recursive importance weight update with the resample and move steps provide the framework for powerful nonlinear non-Gaussian estimation methods dependent only on the choice of importance density and likelihood function. However, there are many disadvantages in using the SIS particle filters:

- The BPF does not use the latest measurement during importance sampling leaving it susceptible to increased variance and instability for many applications.
- All SIS particle filters require the resample and move steps that tend to increase the estimated state error covariances. They also add a significant computational burden to the estimation procedure, since they cannot be parallelized.
- The SIS APF and its alternatives may require the execution of a Gaussian Kalman filter for *every* particle during *every* time step, increasing the computational burden even further.

A second class of particle filters, the *generalized monte carlo* (GMC) particle filters, have been developed that do not use a recursive weight update, but instead compute new weights with every sequential update of the filter. An example of this more general particle filter class include the Gaussian particle filter (GPF) first proposed by Kotechha and Djurić [1,2].

17.1 THE GAUSSIAN PARTICLE FILTER

Let us return once again to the fundamental equations of Bayes estimation, remembering that it is a two-step process, with the *filtering* step given by the estimation of the posterior (filtering or update) density

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = cp(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) \quad (17.1)$$

and the *predictive* step encompassing the estimation of the predictive (or prior, because it uses all of the prior measurements) density

$$p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) = \int p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{z}_{1:n-1}) d\mathbf{x}_{n-1} \quad (17.2)$$

The fundamental concept of the GPF is to make the *assumption* that both the posterior and prior densities are Gaussian. That is, let

$$p(\mathbf{x}_n | \mathbf{z}_{1:n}) = \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_n, \Sigma_n) \quad (17.3)$$

$$p(\mathbf{x}_n | \mathbf{z}_{1:n-1}) = \mathcal{N}(\mathbf{x}_n; \bar{\boldsymbol{\mu}}_n, \bar{\Sigma}_n) \quad (17.4)$$

Assume that at time t_0 , prior to any observations, we have prior information about the initial density and that

$$p(\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_0, \Sigma_0) \quad (17.5)$$

To initialize the GPF, we first draw samples $\{\tilde{\mathbf{x}}_0^{(i)}\}_{i=1}^{N_s}$ from $p(\mathbf{x}_0)$ and propagate them forward in time using the dynamic equation

$$\tilde{\mathbf{x}}_{n|n-1}^{(i)} = \mathbf{f}_n \left(\tilde{\mathbf{x}}_{n-1|n-1}^{(i)} \right) + \boldsymbol{\mu}_n \quad (17.6)$$

Then we compute $\bar{\boldsymbol{\mu}}_1$ and $\bar{\Sigma}_1$ from

$$\bar{\boldsymbol{\mu}}_n = \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{\mathbf{x}}_{n|n-1}^{(i)} \quad (17.7)$$

$$\bar{\Sigma}_n = \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{\mathbf{x}}_{n|n-1}^{(i)} \tilde{\mathbf{x}}_{n|n-1}^{(i)\top} - \bar{\boldsymbol{\mu}}_n \bar{\boldsymbol{\mu}}_n^\top + \mathbf{Q} \quad (17.8)$$

This now defines the prior (predictive) density (17.4)

Returning to (15.41), we can write the importance weight calculation algorithm in a noniterative fashion as

$$w_n \propto \frac{p(\mathbf{z}_n | \mathbf{x}_n) \mathcal{N}(\mathbf{x}_n; \bar{\boldsymbol{\mu}}_n, \bar{\Sigma}_n)}{q(\mathbf{x}_n | \mathbf{z}_{1:n})} \quad (17.9)$$

After sampling from the (currently unspecified) importance density, $\mathbf{x}_n^{(i)} \sim q(\mathbf{x}_n | \mathbf{z}_{1:n})$, $i = 1, \dots, N_s$, this becomes

$$\tilde{w}_n^{(i)} = \frac{p(\mathbf{z}_n | \mathbf{x}_n = \mathbf{x}_n^{(i)}) \mathcal{N}(\mathbf{x}_n = \mathbf{x}_n^{(i)}; \bar{\boldsymbol{\mu}}_n, \bar{\Sigma}_n)}{q(\mathbf{x}_n = \mathbf{x}_n^{(i)} | \mathbf{z}_{1:n})} \quad (17.10)$$

which is followed by a normalization step

$$w_n^{(i)} = \frac{\tilde{w}_n^{(i)}}{\sum_{i=1}^{N_s} \tilde{w}_n^{(i)}} \quad (17.11)$$

The moments of the posterior can now be computed from

$$\hat{\mathbf{x}}_n = \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{x}_n^{(i)} \quad (17.12)$$

$$\mathbf{P}_n^{\mathbf{xx}} = \sum_{i=1}^{N_s} w_n^{(i)} \mathbf{x}_n^{(i)} \mathbf{x}_n^{(i)\top} - \hat{\mathbf{x}}_n \hat{\mathbf{x}}_n^\top \quad (17.13)$$

and the Gaussian particle filter process is completed, as shown in Figure 17.1. Note that this generalized Monte Carlo GPF still requires additional analytical expressions for both the likelihood function and the importance density.

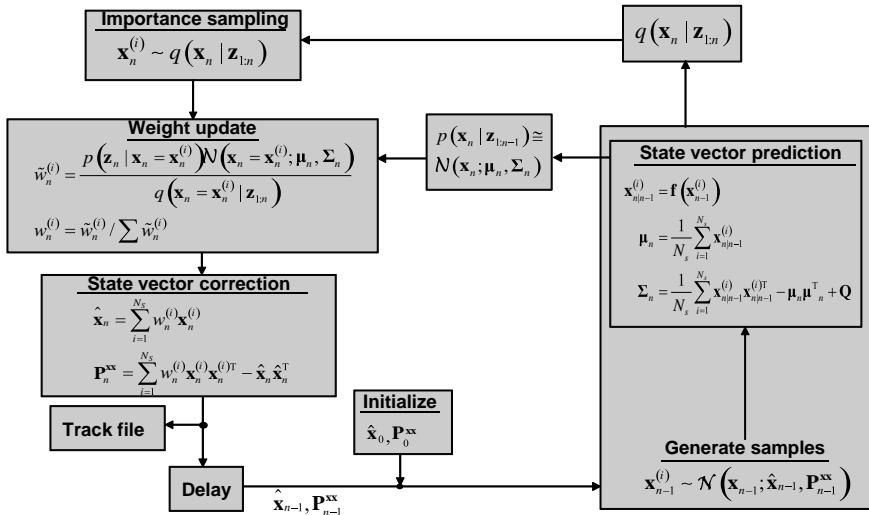


FIGURE 17.1 Process flow diagram for the Gaussian particle filter.

17.2 THE COMBINATION PARTICLE FILTER

In Figure 17.1, the importance density is not specified. Thus, the GPF is not complete and still requires the specification of an importance density. In Ref. [1], it is suggested that a Gaussian distribution be used as the importance density, that is, let $q(\mathbf{x}_n | \mathbf{z}_{1:n}) = \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_{n|n}^x, \Sigma_{n|n}^{xx})$, where $\boldsymbol{\mu}_{n|n}^x$ and $\Sigma_{n|n}^{xx}$ are obtained from the prior density using any of the nonlinear Kalman filters developed in Part II of this book. This approach combines the GPF with one of the nonlinear Kalman filters and is therefore called a generalized Monte Carlo combination particle filter (CPF).

The CPF can be subdivided into two classes, those that use an EKF to generate the importance density and those that use a sigma point Kalman filter. The sigma point Kalman filter class can be further subdivided into the four possible numerical integration Kalman filters, the MCKF, UKF, SSKF, or GHKF. The recursive process flow diagram for the CPF-EKF estimation filter is shown in Figure 17.2 while that of the CPF that uses a sigma point Kalman filter is presented in Figure 17.3. Along the bottom of the figures the Gaussian particle filter structure can be identified and along the right side the nonlinear Kalman filter structure can be seen. In Ref. [3], an unscented particle filter is presented that is similar, but does not include the Gaussian approximation for the prior. It should be noted that [3] contains a significant number of errors, making it very difficult to follow.

These generalized Monte Carlo combination particle filters work very well for most applications because the covariance associated with the importance density will

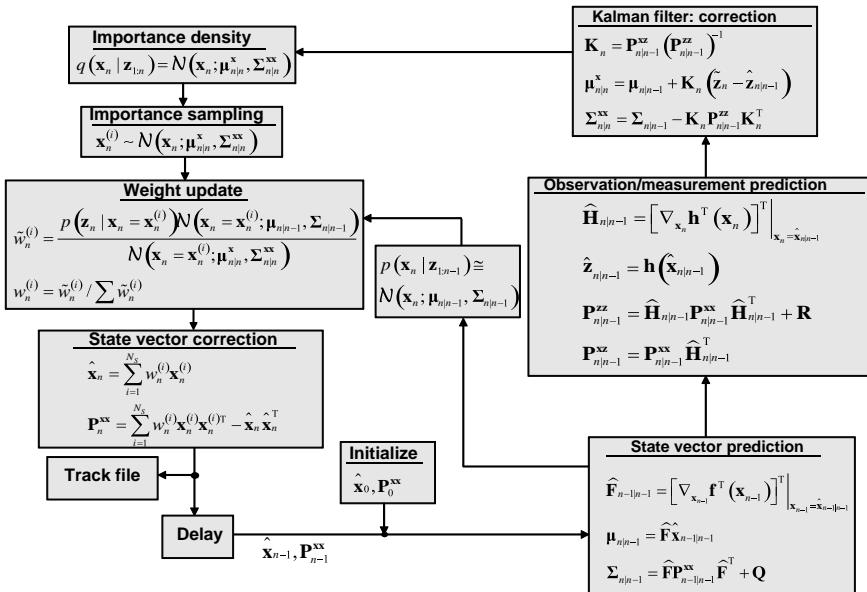


FIGURE 17.2 Combination particle filter that uses an EKF as an importance density.

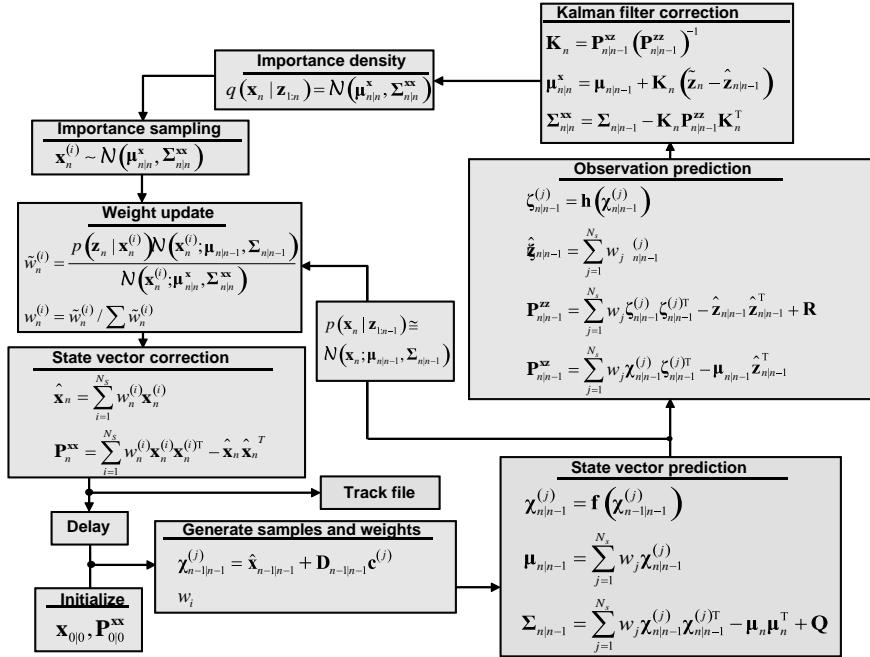


FIGURE 17.3 Combination particle filter that uses a sigma point kalman filter as an importance density.

usually be very broad and will encompass a major portion of the support of the posterior density. With the exception of applications where the posterior has large tail distributions, we have found that the combination particle filters are close to being the “best” particle filter for any application. Since they do not require the sample and move steps and they execute the nonlinear Kalman filter only once during every iteration, they offer a significant computational improvement over most of the SIS particle filters.

For heavy tailed or multimodal prior and posterior densities, there are several alternatives that can be used to improve the performance of the generalized Monte Carlo particle filter class. For multimodal non-Gaussian densities, a Gaussian mixture particle filter has been proposed by Kotecha and Djurić [4] in which both the predictive and filtering distributions are approximated as Gaussian mixtures. These Gaussian sum particle filters (GSPF) have been shown to have improved performance over the GPF for several applications. Merwe and Wan presented a sigma point Gaussian mixture combination particle filter in Ref. [5] where it was used to estimate one-dimensional state in a highly nonlinear dynamic transition equation driven by Gamma noise. The state vector was augmented with the Gamma dynamic noise and the results were very promising, even when there were numerous outlier observations due to the heavy tails of the Gamma distribution.

Several modifications to the generalized Monte Carlo method can be used to handle heavy-tailed noise distributions. Instead of using a Gaussian approximation for the predictive and filtering densities, one could use a Levy, alpha-stable or Laplace distribution and replace the nonlinear Gaussian Kalman filter with a Kalman–Levy [6], generalized stable [7], or Laplace filter [8]. These filters have a Kalman filter-like symmetric structure that could readily be used in place of the nonlinear Gaussian Kalman filter in a combination particle filter application. The Kalman–Levy [9,10] and Laplace filters [8] have been successfully applied as tracking filters for maneuvering targets but have not yet been utilized in particle filter applications.

Like the Gaussian Kalman filter, all of these methods required a consistency of the analytical form of the density. Only a single analytical density had to be used throughout the filter. Another generalization of the nonlinear Kalman filter structure was proposed by Chen and Singpurwalla [11] in which the analytical density could vary for various parts of the filter. Their formulation utilized Gamma, Beta, and Pearson distributions for various parts of the filter, with the state prediction, observation prediction, and update formalism of the Kalman filter maintained throughout. In actual applications, a filter similar to the MCKF given in Chapter 12 was used. Their method could easily be adopted within a generalized particle filter formalism.

17.2.1 Application of the CPF–UKF to DIFAR Buoy Tracking

Application of a CPF–UKF to the DIFAR tracking problem resulted in the Monte Carlo tracking results for six SNRs shown in Figure 17.4. Comparisons with track

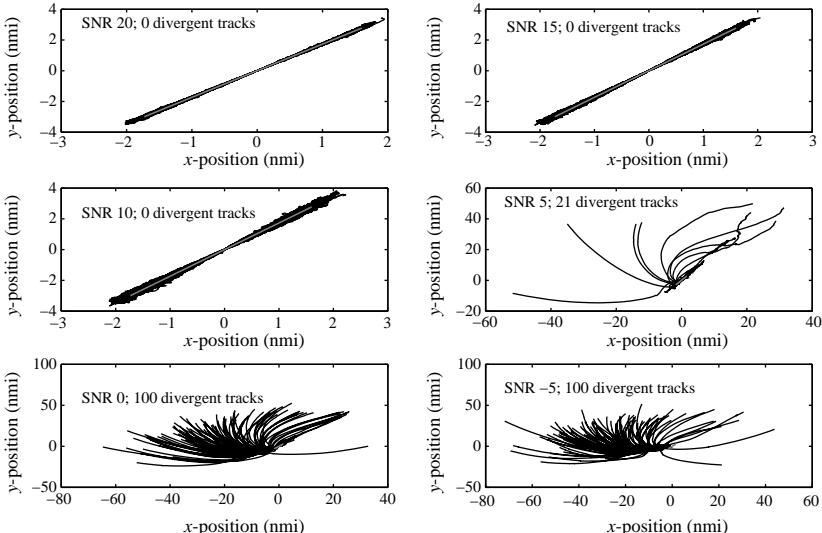


FIGURE 17.4 Monte Carlo track plots for the sigma point Gaussian particle filter for six SNRs.

plots for any of the other track filters shown earlier indicates that this filter is no better (and no worse) than any of the other filters.

17.3 PERFORMANCE COMPARISON OF ALL DIFAR TRACKING FILTERS

In this section, we present an analysis of the performance of many of the tracking filters applied to the DIFAR case study. We have delayed this performance analysis until all of the filters have been presented in Parts II and III. Since all of the results presented so far have been based on simulated observation data, the best method of assessing performance is by comparison of the x - and y -axis position RMSE for the various filters. For each signal SNR, a RMS position error comparison plot is presented for four of the Gaussian filters (EKF, UKF, SSKF, and GHKF) and three of the particle filters (BPF, APF, and GSPF). The acronym GSPF represents a combination particle filter with an UKF used as the importance density. For all particle filters, 3000 Monte Carlo samples were used.

As a first example, Figure 17.5 shows a comparison of both the x - and y -axis RMS position errors as a function of the true position of the target ship when the signal SNR is 20 dB. One can immediately see that the BPF RMS position errors grow quickly due to the three divergent tracks (out of 100 Monte Carlo runs) at 20 dB SNR, as can be seen in Figure 16.5. This is just another illustration of the problems associated with initialization sensitivity for the BPF. Since all of the filters were initialized with

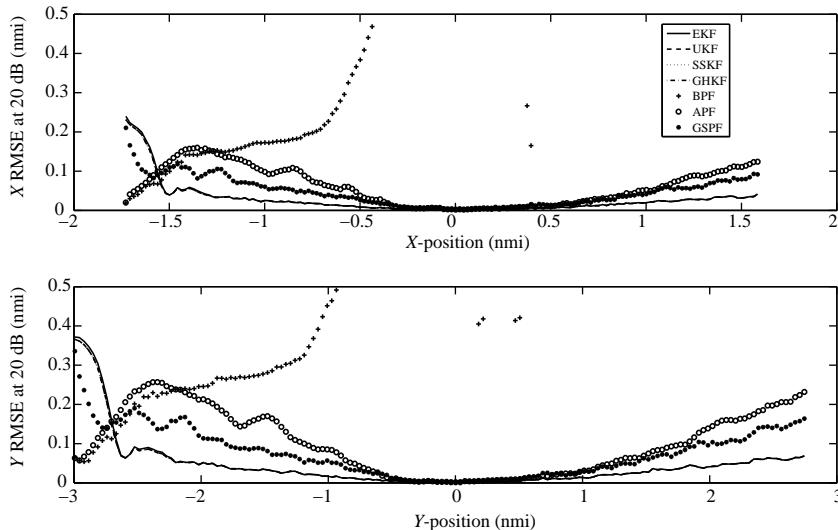


FIGURE 17.5 Comparison of the DIFAR case-study root mean squared position errors for a signal SNR of 20 dB.

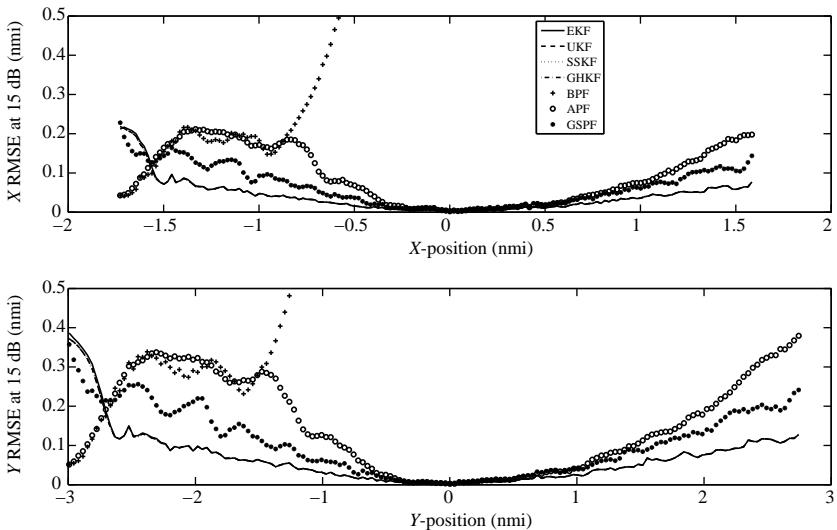


FIGURE 17.6 Comparison of the DIFAR case-study root mean squared position errors for a signal SNR of 15 dB.

the same set of initialization values, this additional tuning needed for the BPF is unacceptable for application to real-world tracking.

In addition, we note that all of the Gaussian Kalman filter variants have almost identical performance and that their performance is slightly better than that of either the APF or the GSPF. And as implied in the previous section, the GSPF has the best performance among the particle filters. Examination of the figure also shows that the RMSE track performance for most of the filters is highest shortly after tracking commences, reaches a minimum when the ship enters the buoy field and is almost symmetric about the origin. The initial high RMS errors occur due to poor initialization of the filters, but as soon as the filters have integrated a few observations they settle to a reasonable track estimate.

When the signal SNR is lowered to 15 dB, the RMS position errors increase relative to those of the 20 dB case, as can be seen in Figure 17.6. The BPF position RMSE diverges due to seven divergent tracks at this SNR. None of the other filters showed any divergent tracks in the 100 Monte Carlo runs. Once again, at this high SNR, the Gaussian filters showed almost identical RMS position errors with the APF and GSPF having slightly higher errors. Figures 14.3–14.7 show the RMS position errors for just the Gaussian filters from 20 dB down to 0 dB. These figures are scaled to the Gaussian filter RMS errors and therefore show a better comparison among the Gaussian filter errors and also show very little performance difference among the Gaussian filters for this application.

Figure 17.7 presents the RMS position errors for a 10 dB signal. At this signal level, the APF had four divergent tracks (see Figure 16.6) causing the RMS errors to grow. But the GSPF and all of the Gaussian filters are still stable so that their RMS

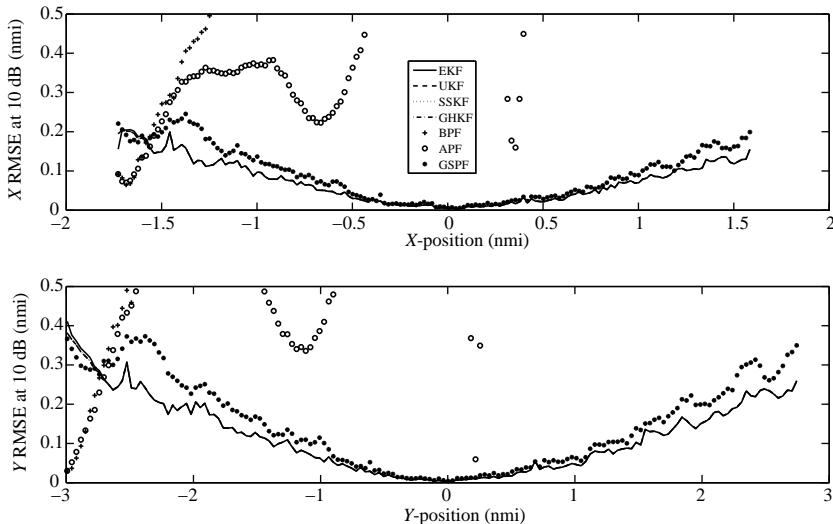


FIGURE 17.7 Comparison of the DIFAR case-study root mean squared position errors for a signal SNR of 10 dB.

errors are still acceptable. We will not show RMS position error plots for SNRs below 10 dB because the Monte Carlo track estimations for all of the filters showed some divergent tracks making the calculation of RMS errors invalid.

The performance analysis presented here and at the end of Chapter 14 indicate that tracking a ship transiting through a DIFAR buoy field is only viable if the SNR is greater than 5 dB. These results also point out that there is no advantage to using a particle filter if the dynamic noise is Gaussian regardless of the form of the density for the observations. One could argue that more particles were needed to improve the performance of the particle filters but we saw little improvement when we increased the number of particles from 1000 to 3000. Finally, it is clear from these results that the BPF is very sensitive to initialization, so care must be exercised when using the BPF.

REFERENCES

1. Kotecha JH, Djurić PM. Gaussian Particle Filtering. *Proceedings of the 11th Signal Processing Workshop on Statistical Signal Processing*; 2001, pp. 429–432.
2. Kotecha JH, Djurić PM. Gaussian particle filtering. *IEEE Trans. Sig. Proc.* 2003;51(10): 2592–2601.
3. Wan EA, van der Merwe R. The unscented Kalman filter for nonlinear estimation. *Adaptive Systems for Signal Processing, Communications and Control Symposium, AS-SPEC*; 2000, pp. 153–158.

4. Kotecha JH, Djurić PM. Gaussian sum particle filtering. *IEEE Trans. Sig. Proc.* 2003; 51(10):2602–2612.
5. van der Merwe R, Wan E. Gaussian Mixture Sigma-Point Particle Filters for Sequential Probabilistic Inference in Dynamic State-Space Models. ICASSP; 2003.
6. Sornette D, Ide K. The Kalman-Levy filter. *Physica D*, 115:2001;142–174.
7. Balakrishna N. Statistical signal extraction using stable processes. *Stat. Prob. Lett.* 2009; 79(7):851–856.
8. Wang D, Zhang C, Zhao X. Multivariate Laplace filter: a heavy-tailed model for target tracking. 19th International Conference on Pattern Recognition; 2008, pp. 1–4.
9. Gordon N, Percival J, Robinson M. The Kalman-Levy filter and heavy-tailed models for tracking maneuvering targets. *Proceedings of the 6th International Conference on Information Fusion*; 2003, pp. 1024–1031.
10. Sinha A, Kirubarajan T, Bar-Shalom Y. Application of the Kalman-Levy filter for tracking maneuvering targets. *Signal Data Process. Small Targets* 2004;5428;102–112.
11. Chen Y, Singpurwalla ND. A non-Gaussian Kalman filter model for tracking software reliability. *Stat. Sinica* 1994;4(2);535–548.

PART IV

ADDITIONAL CASE STUDIES

18

A SPHERICAL CONSTANT VELOCITY MODEL FOR TARGET TRACKING IN THREE DIMENSIONS

In this case study we address a very important estimation problem that is intrinsic to many engineering endeavors, the problem of tracking an object in three dimensional space. Many three-dimensional tracking algorithms use an inertial frame Cartesian constant velocity linear dynamic model for target motion and a nonlinear observation model that relates the Cartesian state vector \mathbf{x} to a set of spherical observations \mathbf{z} . If the dynamic and observation noises are considered to be Gaussian, Kalman filter implementation results in a set of *linear* Kalman filter state prediction equations and *nonlinear* observation prediction equations that can utilize any one of the suboptimal filtering techniques presented in Part II of this book.

The issue of tracking a maneuvering (accelerating) target has been addressed in a variety of ways. For tracking in Cartesian coordinates, the simplest approach is to remove the constant velocity constraint from the target dynamic model by including acceleration components in the state vector and replacing the constant velocity constraint with a constant acceleration constraint [1–3]. Problems with this method are encountered when it is applied to targets that only maneuver for short periods and are nonmaneuvering for the majority of their trajectory. To address these cases, several authors have proposed the use of a filter that switches from a constant velocity model to a constant acceleration model when a maneuver is detected, but detecting a maneuver is not always easy. Another approach that uses multiple banks of models, with each model used to track a different potential maneuver, have been proposed in Refs [2–4]. Still another model consists of an interactive multiple model (IMM) that uses a bank of filter models that only vary the dynamic noise from model to

model with an output that consists of a mixture distribution. Readers interested in understanding IMM methods are referred to these books and the references contained therein.

Although all of the above methods for tracking a maneuvering target work well for many tracking scenarios, as an alternative several authors have recast the target dynamic model into spherical polar coordinates maintaining the constraint of zero-acceleration in inertial Cartesian coordinates. In the methods proposed in Refs [5–8], a set of orthogonal Cartesian axes rotate relative to an inertial frame so that one axis continually aligns with the line-of-site axis (the range axis). Since the state vector is in rotating spherical coordinates, pseudo-accelerations must be introduced into the dynamic motion equations due to the Coriolis forces created by the rotating coordinate system. These pseudo-accelerations, coupled with the inertial frame Cartesian constant velocity assumption, result in a set of nonlinear fully coupled spherical dynamic equations. These filters give very poor performance when used for tracking a maneuvering target from a fixed inertial coordinate system [5].

In this chapter, we will develop a much simpler target dynamic model that works well in tracking a target following trajectories that include both maneuvering and nonmaneuvering sections. In our model, the inertial frame Cartesian constant velocity constraint is replaced by an inertial frame spherical constant velocity constraint where the orthogonal spherical velocity components are restrained to be constant. For a maneuvering target, constant velocity constraints will always be violated, but we will show that, during maneuvers, the RMS tracking errors will be smaller with the spherical constraint. Similar results have been previously been reported in Ref. [5].

In the latter part of this chapter, we will be comparing the performance of tracking algorithms that make the Cartesian constant velocity assumption with those that make a spherical constant velocity assumption. Therefore, we begin in Section 18.1 with a consideration of methods for tracking an object in Cartesian coordinates with the assumption of constant Cartesian velocity. The dynamic and observation equations are presented and both the extended and sigma point Kalman filters in Cartesian coordinates are described.

This is followed, in Section 18.2 with a similar development of the new set of nonlinear constant spherical velocity dynamic and linear observation equations, which are derived from first principles. Because the dynamic equations are referenced to an inertial nonrotating coordinate system, pseudo-accelerations never appear in the equations. Based on this model, spherical extended and sigma point Kalman filters are addressed. In addition, for large ranges, the general nonlinear spherical dynamic equations are shown to reduce to a set of linear constant *angular rate* dynamic equations, leading to a much simpler spherical linear Kalman filter.

Filter implementation issues are discussed in Section 18.3, beginning in Section 18.3.1 with a description of the meaning of the components of the dynamic noise term q (to be defined below) and how their values can be set based on knowledge of expected target acceleration limits.

All tracking filters require two essential sets of data as inputs: a set of observations and a set of initializations. In Section 18.3.2, we explain how observations are generated for a notional radar, one that is used for all of performance assessments

discussed in Section 18.4. The method we used to create initialization values based on the first few noisy observations is discussed in Section 18.3.3.

In Section 18.4 the performance of the Cartesian filters are compared to that of the spherical filters. Through a performance analysis with three simulated trajectories, where the level of target maneuvering increases from trajectory to trajectory, we show that the spherical track estimation filters have the better performance during target maneuvers and when the target approached zero range.

Finally, in Section 18.5, we present some observations about this case study and give some general guidelines for potential future expansion of the work presented here.

Two appendices are included at the end of this chapter because of their importance in understanding the development of three-dimensional tracking. Since we use simulations throughout this chapter, Appendix APPENDIX 18.A contains a discussion of a method for generating the trajectory of a maneuvering object in three dimensions. Usually this is accomplished using a realistic high-fidelity six-degree-of-freedom (6 DOF) simulation of the maneuvering object using a complete model of the forces that act on the object, such as thrust, friction, gravity, and so on. But since we are only interested in evaluating the performance of a variety of tracking algorithms, in Appendix APPENDIX 18.A we present a lower fidelity simulation of object motion, one that simply models the possible maneuvers through linked constant horizontal and vertical turn rate motions that produce a realistic looking trajectory based on a desired turn/dive accelerations. Utilizing this constant turn rate analytical model, three trajectories examples were created that have increasing amounts of target maneuvers. These trajectories are used in Section 18.4 to assess the performance of all trackers as the amount of target maneuvering increases.

In developing tracking algorithms to track an object in three dimensions one has a choice of coordinate systems, usually Cartesian or spherical. Since most sensors used in tracking applications, such as radars or sonars, produce measurements in spherical coordinates, six-dimensional (three position and three velocity components) coordinate transformation subroutines are essential tools. Most coordinate transformation subroutines treat just the state vector transformation. However, many tracking algorithms require coordinate transformation of a covariance matrix or require the calculation of a Jacobian. In Appendix APPENDIX 18.B we present complete coordinate transformations from Cartesian to spherical and from spherical to Cartesian, including the transformation of covariance matrices. As a by-product required for the covariance transformation, the Jacobian is also produced. Since we use an East-North-Up (x,y,z axes positive directions) Cartesian coordinate system in our analysis, transformations in this coordinate system are the ones presented. However, a general method is also presented that will allow the reader to derive such transformations for any desired three-dimensional coordinate system.

18.1 TRACKING A TARGET IN CARTESIAN COORDINATES

In this section, we discuss methods of tracking an object when our tracking filter uses a state vector that is defined in Cartesian coordinates. The section is divided into

subsections where we discuss the dynamic model used to define the expected object motion (Section 18.1.1), the observation vector and how it is related to the state vector (Section 18.1.2), and the various Cartesian Gaussian filter tracking algorithms that can applied to this problem (Section 18.1.3). A discussion of filter initialization and the creation of a Monte Carlo set of observations will be delayed until Sections 18.3.

18.1.1 Object Dynamic Motion Model

The standard method for tracking a moving object from a set of stationary radars in Cartesian coordinates assumes that the object travels at a *constant velocity*. This results in a linear dynamic model of the form

$$\mathbf{x}_n = \mathbf{F}_{n-1} \mathbf{x}_{n-1} + \mathbf{v}_{n-1} \quad (18.1)$$

where the Cartesian state vector at time t_n is defined by

$$\mathbf{x}_n \triangleq \mathbf{x}(t_n) = [r_{x,n}, v_x, r_{y,n}, v_y, r_{z,n}, v_z]^\top \quad (18.2)$$

and the transition matrix \mathbf{F}_n is defined as

$$\mathbf{F}_n = \begin{bmatrix} \mathbf{F}_{1,n} & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{F}_{1,n} & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & \mathbf{F}_{1,n} \end{bmatrix} \quad (18.3)$$

with

$$\mathbf{F}_{1,n} = \begin{bmatrix} 1 & T_n \\ 0 & 1 \end{bmatrix} \quad (18.4)$$

$\mathbf{0}_2$ a 2×2 matrix of zeros and T_n is defined as

$$T_n \triangleq t_n - t_{n-1} \quad (18.5)$$

Note that T_n can be considered time variable, to allow for the temporally asynchronous nature of multisensor tracking. However, in what follows in our analysis, we use only single sensor cases, so we always consider T as invariant with time.

The dynamic acceleration noise \mathbf{v}_n is considered to be independent zero-mean Gaussian acceleration noise defined by

$$\mathbf{v}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_n) \quad (18.6)$$

with

$$\mathbf{Q}_n = \begin{bmatrix} q_{x,n} \mathbf{Q}_1 & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & q_{y,n} \mathbf{Q}_1 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & q_{z,n} \mathbf{Q}_1 \end{bmatrix} \quad (18.7)$$

and, for an object undergoing continuous motion, \mathbf{Q}_1 is given by [3]

$$\mathbf{Q}_1 = \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix} \quad (18.8)$$

Here, the symbol \sim indicates that \mathbf{v}_n is drawn from the Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{Q}_n)$.

An important filter design parameter is $q_{i,n} \forall i$. We will show how to choose $q_{i,n}$ when we discuss the design of various filter options, based on the what we expect the characteristics of object motion will be, that is, will the object be maneuvering or traveling in a straight line.

18.1.2 Sensor Data Model

As noted in Section 18.3.2, most radar sensors output observation (measurement) data in two-dimensional polar or three-dimensional spherical coordinates, independent of the coordinate system chosen for the dynamic model. Consequently, we will define the complete sensor data vector as the following vector

$$\mathbf{z}_n = [R_n, \dot{R}_n, \theta_n, \phi_n]^\top \quad (18.9)$$

where the set $\{R_n, \dot{R}_n, \theta_n, \phi_n\}$ are the range, Doppler (range rate), bearing, and elevation relative to the fixed sensor position, respectively, of the object to be tracked. For sensors that do not have the full compliment of observation components, we merely drop those components that are not available for that particular radar. For example, in the simulations used for performance analysis below, the sensor model chosen contained only range, bearing, and elevation.

The observation vector is related to the Cartesian state vector through the following nonlinear spherical-to-Cartesian transformation, as defined by (18.160) and (18.161)

$$\mathbf{z}_n = \begin{bmatrix} R_n \\ \dot{R}_n \\ \theta_n \\ \phi_n \end{bmatrix} = \mathbf{h}_n(\mathbf{x}_n) = \begin{bmatrix} h_{1,n}(\mathbf{x}_n) \\ h_{2,n}(\mathbf{x}_n) \\ h_{3,n}(\mathbf{x}_n) \\ h_{4,n}(\mathbf{x}_n) \end{bmatrix} = \begin{bmatrix} \sqrt{r_x^2 + r_y^2 + r_z^2} \\ \frac{r_x v_x + r_y v_y + r_z v_z}{R} \\ \tan^{-1}\left(\frac{r_x}{r_y}\right) \\ \tan^{-1}\left(\frac{r_z}{\sqrt{r_x^2 + r_y^2}}\right) \end{bmatrix} \quad (18.10)$$

Generally, sensor noise is included in the observation model so that the observation model becomes

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_n \quad (18.11)$$

where the observation noise \mathbf{w}_n is considered to be independent zero-mean Gaussian noise defined by

$$\mathbf{w}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_n) \quad (18.12)$$

with \mathbf{R}_n usually considered to be independent of time and defined by the diagonal covariance matrix of sensor measurement variances given by

$$\mathbf{R} = \begin{bmatrix} \sigma_R^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{R}}^2 & 0 & 0 \\ 0 & 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & 0 & \sigma_\phi^2 \end{bmatrix} \quad (18.13)$$

For real sensor data, the sensor measurement standard deviations are usually output as part of the sensor data stream at every temporal sensor update, so the components of \mathbf{R} may be time dependent if the signal-to-noise ratio at the output of the sensor detector varies. For the *simulations* used for the performance analysis below, \mathbf{R} will be consider as invariant in time.

18.1.3 Gaussian Tracking Algorithms for a Cartesian State Vector

18.1.3.1 The Cartesian Linear State Prediction Equations. Since the Cartesian dynamic equation is linear, we can use the linear state and state covariance prediction equations from Table 6.1

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}_{n-1}\hat{\mathbf{x}}_{n-1|n-1} \quad (18.14)$$

$$\mathbf{P}_{n|n-1}^{\text{XX}} = \mathbf{F}_{n-1}\mathbf{P}_{n-1|n-1}^{\text{XX}}\mathbf{F}_{n-1}^\top + \mathbf{Q}_{n-1} \quad (18.15)$$

18.1.3.2 Cartesian Nonlinear Observation Prediction. On the other hand, from (18.10) we see that the observation model is nonlinear with respect to the state vector, so observation prediction must be performed with one of the suboptimal filters from Part II. For this case study, we used only the EKF and all of the sigma point KFs, with the exception of the MCKF.

18.1.3.2.1 The Cartesian Linearized EKF Observation Prediction Equations. From Table 7.2, the EKF observation prediction equations are given by

$$\hat{\mathbf{z}}_{n|n-1} = \mathbf{h}_n(\hat{\mathbf{x}}_{n|n-1}) \quad (18.16)$$

$$\mathbf{P}_{n|n-1}^{\text{ZZ}} = \hat{\mathbf{H}}_{n|n-1}\mathbf{P}_{n|n-1}^{\text{XX}}\hat{\mathbf{H}}_{n|n-1}^\top + \mathbf{R}_n \quad (18.17)$$

$$\mathbf{P}_{n|n-1}^{\text{XZ}} = \mathbf{P}_{n|n-1}^{\text{XX}}\hat{\mathbf{H}}_{n|n-1}^\top \quad (18.18)$$

where the components of the Jacobian matrix $\hat{\mathbf{H}}_{n|n-1}$ are presented in (18.164)–(18.190).

18.1.3.2.2 The Cartesian Nonlinear Sigma point Observation Prediction Equations. For the sigma point Kalman filter observation prediction equations shown in

Figure 13.3, we write

$$\mathbf{D}_{n|n-1} = [\mathbf{P}_{n|n-1}^{\mathbf{XX}}]^{1/2} \quad (18.19)$$

$$\boldsymbol{\chi}_{n|n-1}^{(j)} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{D}_{n|n-1} \mathbf{c}^{(j)} \quad (18.20)$$

$$\boldsymbol{\psi}_{n|n-1}^{(j)} = \mathbf{h}_n \left(\boldsymbol{\chi}_{n|n-1}^{(j)} \right) \quad (18.21)$$

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{N_s} w_j \boldsymbol{\psi}_{n|n-1}^{(j)} \quad (18.22)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\mathbf{ZZ}} &= \sum_{j=0}^{N_s} w_j \left(\boldsymbol{\psi}_{n|n-1}^{(j)} - \hat{\mathbf{z}}_{n|n-1} \right) \left(\boldsymbol{\psi}_{n|n-1}^{(j)} - \hat{\mathbf{z}}_{n|n-1} \right)^T \\ &\quad + \mathbf{R}_{n-1} \end{aligned} \quad (18.23)$$

$$\mathbf{P}_{n|n-1}^{\mathbf{XZ}} = \sum_{j=0}^{N_s} w_j \left(\boldsymbol{\chi}_{n|n-1}^{(j)} - \hat{\mathbf{x}}_{n|n-1} \right) \left(\boldsymbol{\psi}_{n|n-1}^{(j)} - \hat{\mathbf{z}}_{n|n-1} \right)^T \quad (18.24)$$

where the set $\{w_j, \mathbf{c}^{(j)}\}$ can be calculated offline using Listings 13.1 and 13.2 (see Chapter 13).

18.1.3.3 The Kalman Filter Update Equations. The final step in track estimation is applying the Kalman filter update equations (5.29)–(5.31)

$$\mathbf{K}_n = \mathbf{P}_{n|n-1}^{\mathbf{XZ}} [\mathbf{P}_{n|n-1}^{\mathbf{ZZ}}]^{-1} \quad (18.25)$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \quad (18.26)$$

$$\mathbf{P}_{n|n-1}^{\mathbf{XX}} = \mathbf{P}_{n|n-1}^{\mathbf{XX}} - \mathbf{K}_n \mathbf{P}_{n|n-1}^{\mathbf{ZZ}} \mathbf{K}_n^T \quad (18.27)$$

18.2 TRACKING A TARGET IN SPHERICAL COORDINATES

In this section, we discuss tracking an object using a spherical state vector defined as

$$\boldsymbol{\rho} = [R, v_r, \theta, v_h, \phi, v_v]^T \quad (18.28)$$

with the spherical velocity components $\{v_r, v_h, v_v\}$ to be defined below. As we did for the Cartesian case, where we assumed that the components of the Cartesian velocity were constant resulting in a Cartesian dynamic motion model given by (18.1), our objective will be to develop a three-dimensional dynamic motion model for an object with constant *spherical* velocity components.

18.2.1 State Vector Position and Velocity Components in Spherical Coordinates

Referring to Figure 18.25, the position of an object can be written as

$$\mathbf{r}_p = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z \quad (18.29)$$

or

$$\mathbf{r}_p = a_r\mathbf{e}_r + a_\theta\mathbf{e}_\theta + a_\phi\mathbf{e}_\phi \quad (18.30)$$

where $\{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$ and $\{\mathbf{e}_R, \mathbf{e}_\theta, \mathbf{e}_\phi\}$ are unit vector sets in Cartesian and spherical coordinates, respectively.

The spherical-to-Cartesian transformation of the position components are defined by (18.193), so we can rewrite (18.29) as

$$\mathbf{r}_p = \mathbf{e}_x R \sin \theta \cos \phi + \mathbf{e}_y R \cos \theta \cos \phi + \mathbf{e}_z R \sin \phi \quad (18.31)$$

By definition, the spherical unit vectors are given by

$$\mathbf{e}_r = \frac{1}{\left| \frac{\partial \mathbf{r}_p}{\partial R} \right|} \frac{\partial \mathbf{r}_p}{\partial R} \quad (18.32)$$

$$\mathbf{e}_\theta = \frac{1}{\left| \frac{\partial \mathbf{r}_p}{\partial \theta} \right|} \frac{\partial \mathbf{r}_p}{\partial \theta} \quad (18.33)$$

$$\mathbf{e}_\phi = \frac{1}{\left| \frac{\partial \mathbf{r}_p}{\partial \phi} \right|} \frac{\partial \mathbf{r}_p}{\partial \phi} \quad (18.34)$$

Using (18.31) in (18.32) results in the following expression for \mathbf{e}_r in terms of the set $\{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$

$$\mathbf{e}_r = \mathbf{e}_x \sin \theta \cos \phi + \mathbf{e}_y \cos \theta \cos \phi + \mathbf{e}_z \sin \phi \quad (18.35)$$

Similarly, we obtain

$$\mathbf{e}_\theta = \mathbf{e}_x \cos \theta - \mathbf{e}_y \sin \theta \quad (18.36)$$

and

$$\mathbf{e}_\phi = -\mathbf{e}_x \sin \theta \sin \phi - \mathbf{e}_y \cos \theta \sin \phi + \mathbf{e}_z \cos \phi \quad (18.37)$$

It follows immediately that the time derivative of the spherical unit vectors are given by

$$\begin{aligned} \dot{\mathbf{e}}_r &= \frac{\partial \mathbf{e}_r}{\partial R} \dot{R} + \frac{\partial \mathbf{e}_r}{\partial \theta} \dot{\theta} + \frac{\partial \mathbf{e}_r}{\partial \phi} \dot{\phi} \\ &= \mathbf{e}_\theta \dot{\theta} \cos \phi + \mathbf{e}_\phi \dot{\phi} \end{aligned} \quad (18.38)$$

$$\begin{aligned}\dot{\mathbf{e}}_\theta &= \frac{\partial \mathbf{e}_\theta}{\partial R} \dot{R} + \frac{\partial \mathbf{e}_\theta}{\partial \theta} \dot{\theta} + \frac{\partial \mathbf{e}_\theta}{\partial \phi} \dot{\phi} \\ &= -\mathbf{e}_r \dot{\theta} \cos \phi + \mathbf{e}_\phi \dot{\theta} \sin \phi\end{aligned}\quad (18.39)$$

and

$$\begin{aligned}\dot{\mathbf{e}}_\phi &= \frac{\partial \mathbf{e}_\phi}{\partial R} \dot{R} + \frac{\partial \mathbf{e}_\phi}{\partial \theta} \dot{\theta} + \frac{\partial \mathbf{e}_\phi}{\partial \phi} \dot{\phi} \\ &= -\mathbf{e}_\theta \dot{\theta} \sin \phi - \mathbf{e}_r \dot{\phi}\end{aligned}\quad (18.40)$$

Using (18.35), we can now write (18.31) as

$$\begin{aligned}\mathbf{r}_p &= \mathbf{e}_x R \sin \theta \cos \phi + \mathbf{e}_y R \cos \theta \cos \phi + \mathbf{e}_z R \sin \phi \\ &= R [\mathbf{e}_x \sin \theta \cos \phi + \mathbf{e}_y \cos \theta \cos \phi + \mathbf{e}_z \sin \phi] \\ &= R \mathbf{e}_r\end{aligned}\quad (18.41)$$

Since the velocity vector is the time derivative of the position vector, we obtain

$$\begin{aligned}\mathbf{v} &= \frac{d}{dt} (R \mathbf{e}_r) = \dot{R} \mathbf{e}_r + R \dot{\mathbf{e}}_r \\ &= \mathbf{e}_r \dot{R} + \mathbf{e}_\theta R \dot{\theta} \cos \phi + \mathbf{e}_\phi R \dot{\phi}\end{aligned}\quad (18.42)$$

we can now identify the components of \mathbf{v} in spherical coordinates as

$$\mathbf{v} = \begin{bmatrix} v_r \\ v_h \\ v_v \end{bmatrix} = \begin{bmatrix} \dot{R} \\ R \dot{\theta} \cos \phi \\ R \dot{\phi} \end{bmatrix} = \begin{bmatrix} \dot{R} \\ R_h \dot{\theta} \\ R \dot{\phi} \end{bmatrix}\quad (18.43)$$

where we have defined the horizontal range $R_h \triangleq R \cos \phi$. Using (18.43) as the velocity components gives us all of the components of the spherical state vector (18.28).

18.2.2 Spherical State Vector Dynamic Equation

For a constant velocity dynamic model of object motion, we assume that $\dot{v}_r = \dot{v}_h = \dot{v}_v = 0$. Taking the time derivative of each component in (18.43) yields

$$\dot{v}_r = \ddot{R} = 0\quad (18.44)$$

$$\dot{v}_h = \dot{R} \dot{\theta} \cos \phi + R \ddot{\theta} \cos \phi - R \dot{\theta} \dot{\phi} \sin \phi = 0\quad (18.45)$$

and

$$\dot{v}_v = \dot{R} \dot{\phi} + R \ddot{\phi} = 0\quad (18.46)$$

Or, after rearranging, we obtain

$$\ddot{\theta} = \dot{\theta} \left(\dot{\phi} \tan \phi - \frac{\dot{R}}{R} \right) \quad (18.47)$$

and

$$\ddot{\phi} = -\frac{\dot{R}}{R} \dot{\phi} \quad (18.48)$$

The components of the dynamic motion equation will now be developed. For range, we can write

$$R_n = R_{n-1} + T \dot{R}_{n-1} + \frac{T^2}{2} \ddot{R}_{n-1} \quad (18.49)$$

Using (18.44), and the definition of v_r from (18.43), this becomes

$$R_n = R_{n-1} + T v_{r,n-1} \quad (18.50)$$

For the radial speed we have

$$\begin{aligned} v_{r,n} &= v_{r,n-1} + T \dot{v}_r \\ &= v_{r,n-1} \end{aligned} \quad (18.51)$$

Writing an equation similar to (18.49) for the bearing, substituting (18.47) for $\ddot{\theta}_{n-1}$ and using the definitions of the velocity components from (18.43), we obtain

$$\theta_n = \theta_{n-1} + T \frac{v_{h,n-1}}{R_{h,n-1}} \left[1 + \frac{T}{2R_{n-1}} (v_{v,n-1} \tan \phi_{n-1} - v_{r,n-1}) \right] \quad (18.52)$$

In a manner similar to (18.51), it follows that

$$v_{h,n} = v_{h,n-1} \quad (18.53)$$

In addition, it follows that the elevation components are given by

$$\phi_n = \phi_{n-1} + T \frac{v_{v,n-1}}{R_{n-1}} \left[1 - \frac{T}{2} \frac{v_{r,n-1}}{R_{n-1}} \right] \quad (18.54)$$

and

$$v_{v,n} = v_{v,n-1} \quad (18.55)$$

The nonlinear dynamic motion equation for the spherical state vector can now be written as

$$\rho_n = \mathbf{f}(\rho_{n-1}) + \mathbf{n}_{n-1} \quad (18.56)$$

with the zero-mean Gaussian dynamic acceleration noise distribution

$$\mathbf{n}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_\rho) \quad (18.57)$$

The components of $\mathbf{f}(\rho_{n-1})$ are given by the nonlinear equations (18.50)–(18.55) and

$$\mathbf{Q}_\rho = \begin{bmatrix} q_r \mathbf{Q}_1 & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & q_h \mathbf{Q}_1 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & q_v \mathbf{Q}_1 \end{bmatrix} \quad (18.58)$$

with

$$\mathbf{Q}_1 = \begin{bmatrix} \frac{T^3}{3} & \frac{T^2}{2} \\ \frac{T^2}{2} & T \end{bmatrix} \quad (18.59)$$

18.2.2.1 Long-Range Limit of the Nonlinear Spherical Dynamic Equations. Using the components of (18.43), we can replace $v_{h,n-1}/R_{h,n-1}$ and $v_{v,n-1}/R_{n-1}$ in (18.52) and (18.54) by $\dot{\theta}_{n-1}$ and $\dot{\phi}_{n-1}$, respectively. Now, in the limit as $R_{n-1} \rightarrow \infty$, (18.50)–(18.55) become the linear constant rate dynamic equations

$$R_n = R_{n-1} + T \dot{R}_{n-1} \quad (18.60)$$

$$\dot{R}_n = \dot{R}_{n-1} \quad (18.61)$$

$$\theta_n = \theta_{n-1} + T \dot{\theta}_{n-1} \quad (18.62)$$

$$\dot{\theta}_n = \dot{\theta}_{n-1} \quad (18.63)$$

$$\phi_n = \phi_{n-1} + T \dot{\phi}_{n-1} \quad (18.64)$$

$$\dot{\phi}_n = \dot{\phi}_{n-1} \quad (18.65)$$

For this case, the spherical state vector at time t_n can be represented by

$$\mathbf{r}_n = [R_n, \dot{R}_n, \theta_n, \dot{\theta}_n, \phi_n, \dot{\phi}_n]^\top \quad (18.66)$$

and the dynamic equation becomes the linear equation

$$\mathbf{r}_n = \mathbf{F} \mathbf{r}_{n-1} + \mathbf{n}_{n-1} \quad (18.67)$$

where

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_1 & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{F}_1 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & \mathbf{F}_1 \end{bmatrix} \quad (18.68)$$

with

$$\mathbf{F}_1 = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \quad (18.69)$$

and

$$\mathbf{Q}_r = \begin{bmatrix} q_r \mathbf{Q}_1 & \mathbf{0}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & q_\theta \mathbf{Q}_1 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & q_\phi \mathbf{Q}_1 \end{bmatrix} \quad (18.70)$$

18.2.3 Observation Equations with a Spherical State Vector

The sensor data vector is given in (18.9). Based on this data vector, we can write the linear observation equation related to either form of the spherical dynamic equation, (18.56) or (18.67), as

$$\mathbf{z}_n = \mathbf{H}\rho_n + \mathbf{w}_{n-1} \quad (18.71)$$

$$= \mathbf{H}\mathbf{r}_n + \mathbf{w}_{n-1} \quad (18.72)$$

with

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (18.73)$$

18.2.4 Gaussian Tracking Algorithms for a Spherical State Vector

18.2.4.1 Track Estimation for Large R . Consider first the case where $R \gg 1$. We see that both the dynamic and observation equations, (18.67) and (18.72) respectively, are linear, so the linear Kalman filter developed in Chapter 6 is the optimal filter. As shown in Chapter 6, the LKF state prediction equations for this case are given by

$$\hat{\mathbf{r}}_{n|n-1} = \mathbf{F}\hat{\mathbf{r}}_{n-1|n-1} \quad (18.74)$$

$$\mathbf{P}_{n|n-1}^{rr} = \mathbf{F}\mathbf{P}_{n-1|n-1}^{rr}\mathbf{F}^\top + \mathbf{Q}_r \quad (18.75)$$

with \mathbf{F} and \mathbf{Q} shown in (18.68) and (18.70), respectively.

The observation prediction equations can be written as

$$\hat{\mathbf{z}}_{n|n-1} = \mathbf{H}\hat{\mathbf{r}}_{n|n-1} \quad (18.76)$$

$$\mathbf{P}_{n|n-1}^{zz} = \mathbf{H}\mathbf{P}_{n|n-1}^{rr}\mathbf{H}^\top + \mathbf{R} \quad (18.77)$$

$$\mathbf{P}_{n|n-1}^{xz} = \mathbf{P}_{n|n-1}^{rr}\mathbf{H}^\top \quad (18.78)$$

with \mathbf{H} and \mathbf{R} given by (18.73) and (18.13), respectively.

And finally, the Kalman filter update equations are the usual

$$\mathbf{K}_n = \mathbf{P}_{n|n-1}^{\text{xz}} [\mathbf{P}_{n|n-1}^{\text{zz}}]^{-1} \quad (18.79)$$

$$\hat{\mathbf{r}}_{n|n} = \hat{\mathbf{r}}_{n|n-1} + \mathbf{K}_n (\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}) \quad (18.80)$$

$$\mathbf{P}_{n|n}^{\text{rr}} = \mathbf{P}_{n|n-1}^{\text{rr}} - \mathbf{K}_n \mathbf{P}_{n|n-1}^{\text{zz}} \mathbf{K}_n^\top \quad (18.81)$$

18.2.4.2 Track Estimation for the Nonlinear Spherical dynamic Model. For this spherical state vector case, we have the exact opposite of what we had in the Cartesian state vector case. For the Cartesian state vector with constant Cartesian velocity motion, the state prediction equations were linear but the observation equations were nonlinear, requiring one of the nonlinear suboptimal tracking filters for observation prediction. Now, the state vector prediction equations are nonlinear while the observation prediction equations are linear and one of the nonlinear suboptimal tracking filters must be used for the state prediction step. Below, we will first examine the modifications required to utilize the linearized EKF for the state prediction step and then we will consider the sigma point methods.

18.2.4.2.1 Applying the EKF in Spherical Coordinates. In order to use the EKF for state prediction in spherical coordinates, we must first find the Jacobian of the nonlinear state transition equation

$$\boldsymbol{\rho}_n = \mathbf{f}(\boldsymbol{\rho}_{n-1}) \quad (18.82)$$

where the components of the transition are shown in (18.50)–(18.55). Now, the EKF state prediction equations are

$$\hat{\boldsymbol{\rho}}_{n|n-1} = \mathbf{f}(\hat{\boldsymbol{\rho}}_{n-1|n-1}) \quad (18.83)$$

$$\mathbf{P}_{n|n-1}^{\rho\rho} = \hat{\mathbf{F}}_{\rho,n-1} \mathbf{P}_{n-1|n-1}^{\rho\rho} \hat{\mathbf{F}}_{\rho,n-1}^\top + \mathbf{Q}_\rho \quad (18.84)$$

where $\hat{\mathbf{F}}_{\rho,n-1}$ is the Jacobian of the transition equation (18.82) at time t_{n-1} .

Applying the general methods found in Appendix APPENDIX 18.B, we find that the Jacobian of (18.82) is given by

$$\hat{\mathbf{F}}_\rho = \begin{bmatrix} 1 & T & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ F_{31} & F_{32} & 1 & F_{34} & F_{35} & F_{36} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ F_{51} & F_{52} & 0 & 0 & 1 & F_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (18.85)$$

with

$$F_{31} = -\frac{T v_h}{R R_h} \left[1 + \frac{T}{R} (v_v \tan \phi - v_r) \right] \quad (18.86)$$

$$F_{32} = -\frac{T^2 v_h}{2 R R_h} \quad (18.87)$$

$$F_{34} = \frac{T}{R_h} \left[1 + \frac{T}{2 R} (v_v \tan \phi - v_r) \right] \quad (18.88)$$

$$F_{35} = \frac{T^2 v_h}{2 R R_h} v_v \sec^2 \phi \quad (18.89)$$

$$F_{36} = \frac{T^2 v_h}{2 R R_h} \tan \phi \quad (18.90)$$

$$F_{51} = -\frac{T v_v}{R^2} \left[1 + \frac{T v_r}{R} \right] \quad (18.91)$$

$$F_{52} = \frac{T^2 v_v}{2 R^2} \quad (18.92)$$

$$F_{56} = \frac{T}{R} \left[1 + \frac{T v_r}{R} \right] \quad (18.93)$$

where we have removed the subscript n for clarity.

Since none of the components of the observation vector depend on v_h or v_v , the observation prediction and update equations are identical to those of the spherical LKF presented in Section 18.2.4.1.

18.2.4.2.2 Applying a Sigma Point Kalman Filter in Spherical Coordinates. The first step in the process is to generate a set of sigma points from the prior filter output $\{\hat{\rho}_{n-1|n-1}, \mathbf{P}_{n-1|n-1}^{\rho\rho}\}$ or from the initialization values using

$$\Pi_{n-1|n-1}^{(j)} = \hat{\rho}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}^{(j)}, \quad j = 0, 1, \dots, N_s \quad (18.94)$$

where

$$\mathbf{D}_{n-1|n-1} = \left[\mathbf{P}_{n-1|n-1}^{\rho\rho} \right]^{1/2} \quad (18.95)$$

The set $\{w_j, \mathbf{c}^{(j)}, j = 0, 1, \dots, N_s\}$ are generated using the subroutines presented in Listings 13.1 and 13.2. They will depend on the specific sigma point Kalman filter chosen and the dimension of the state vector.

Now, the state prediction equations for the sigma point Kalman filter are given by

$$\Pi_{n|n-1}^{(j)} = \mathbf{f} \left(\Pi_{n-1|n-1}^{(j)} \right) \quad (18.96)$$

$$\hat{\rho}_{n|n-1} = \sum_{j=0}^{N_s} w_j \Pi_{n|n-1}^{(j)} \quad (18.97)$$

$$\mathbf{P}_{n|n-1}^{\rho\rho} = \sum_{j=0}^{N_s} w_j \left(\Pi_{n|n-1}^{(j)} - \hat{\rho}_{n|n-1} \right) \left(\Pi_{n|n-1}^{(j)} - \hat{\rho}_{n|n-1} \right)^T \quad (18.98)$$

$$+ \mathbf{Q}_\rho \quad (18.99)$$

And, as with the spherical EKF, the linear observation prediction and the update equations are identical to those of the spherical LKF presented above.

18.3 IMPLEMENTATION OF CARTESIAN AND SPHERICAL TRACKING FILTERS

In order to execute any of the tracking filters presented above, one must

1. specify a set of values for q ,
2. generate a Monte Carlo set of noisy data that can be used as observations for the filters, and
3. create an initialization state vector and covariance matrix.

We address each of these in the sections below.

18.3.1 Setting Values for q

Consider the prediction equation for the Cartesian state covariance matrix given in (18.15). If we compare the upper left 2×2 set of components of the left-hand side of (18.15) with the similar set of components of \mathbf{Q} from (18.8), it follows that

$$\begin{bmatrix} P_{xx} & P_{x\dot{x}} \\ P_{x\dot{x}} & P_{\dot{x}\dot{x}} \end{bmatrix} = \begin{bmatrix} \sigma_x^2 & \bullet \\ \bullet & \sigma_{v_x}^2 \end{bmatrix} \propto \begin{bmatrix} q_x \frac{T^3}{3} & q_x \frac{T^2}{2} \\ q_x \frac{T^2}{2} & q_x T \end{bmatrix} \quad (18.100)$$

Thus, $\sigma_{v_x}^2 \propto q_x T$, and it follows immediately that q_x has units of [distance 2 /time 3]. Or, we could write this as [acceleration $^2 \times$ time] or [acceleration 2 /Hz].

Assuming that we know something about the maneuverability characteristics of the object we are trying to track, we can estimate the value required for q_x . Since our dynamic model is one of constant velocity with zero-mean Gaussian acceleration noise, the value for q_x can be set to twice the largest possible maneuver acceleration

of the object. It is doubled to account for turns to the left or right. So we set

$$q_x = (2a)^2 \quad (18.101)$$

If a is given in g 's (the acceleration due to gravity), we must convert the units to $(\text{nmi}/\text{s}^2)^2$. For example, if an object can execute a turn at a maximum of 3 g 's, we have approximately

$$\begin{aligned} q_x &= (2 \times 3 \text{ g})^2 \times \left(32.174 \text{ ft/s}^2/\text{g}\right)^2 \times \left(1.65 \times 10^{-4} \text{ nmi}/\text{ft}\right)^2 \\ &= 36 \times 2.80 \times 10^{-5} \left(\text{nmi}/\text{s}^2\right)^2 \end{aligned} \quad (18.102)$$

From (18.7) we see that separate values can be set for q for different components. Thus, for a Cartesian filter, we could set one value for horizontal maneuvers (q_x and q_y) and a different value for vertical maneuvers (q_z).

For the spherical filters we have the sets $\{q_r, q_h, q_v\}$ for constant spherical velocity filters and $\{q_r, q_\theta, q_\phi\}$ for the constant rate spherical filter. Generally, we would expect that we could set q_r and q_h to the same values that we set the Cartesian horizontal q 's and q_v to the value of the Cartesian vertical q_z . For the constant rate spherical filters, it is easy to show that the angular q 's should be set such that $q_\theta = q_r/R_h^2$ and $q_\phi = q_r/R^2$. However, as we will show later, higher values of the spherical q 's gives better performance.

18.3.2 Simulating Radar Observation Data

In order to conduct a performance comparison of a set of track estimation algorithms, each tracking algorithm must use the *identical* set of noisy observations for a given Monte Carlo run. In addition, for each Monte Carlo run, a different set of observations must be generated using a new set of noise data that is independent of the previous set. So, if we are going to use 100 Monte Carlo runs to compare the performance of multiple track estimation algorithms, we need 100 sets of noisy observations where the noise is independent from time sample to time sample and also independent from one Monte Carlo set to all others.

To track the object with a trajectory generated using the low-fidelity simulation presented in Appendix APPENDIX 18.A, we ignore the effects of atmospheric round-trip propagation on the radar signal as well as the radar detectors detection statistics and we will assume a detection probability of 100%. To generate observation sets, the characteristics of the radar must be set:

- What variables constitute the radar observation vector.
- What is the radar update rate.
- What is the observation noise standard deviation for each observation variable.

For a general radar, the observation vector can consist of range to the object (R) and the rate at which the range is changing (\dot{R}), the horizontal bearing to the object

relative to true North (θ) and the elevation of the object above the ground (ϕ). Thus, the observation vector can be

$$\mathbf{z} = [R, \dot{R}, \theta, \phi]^\top \quad (18.103)$$

Note that in our spherical polar coordinate system $-180^\circ \leq \theta \leq 180^\circ$ and $0^\circ \leq \phi \leq 90^\circ$.

It is usually assumed that the noise on each component of the observation vector are independent and Gaussian resulting in an observation noise covariance matrix given by

$$\mathbf{R} = \begin{bmatrix} \sigma_R^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{R}}^2 & 0 & 0 \\ 0 & 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & 0 & \sigma_\phi^2 \end{bmatrix} \quad (18.104)$$

where σ_i is the standard deviation of the noise for the i th observation vector component.

In the track estimation performance results presented in the next section, we use a radar model that measures only the range, bearing, and elevation. The observation uncertainties used to generate the noisy observations are

$$\sigma_R = 0.15 \text{ nmi} \quad (18.105)$$

$$\sigma_\theta = 0.3 \text{ deg} \quad (18.106)$$

$$\sigma_\phi = 0.3 \text{ deg} \quad (18.107)$$

Our simulated radar will have an update period of one observation every 4 s for an update rate of 0.25 Hz.

Using the methods of Appendix APPENDIX 18.A, several Cartesian target trajectories have been generated, with the simulated trajectory data generated at a 10 Hz rate. Since our radar model has a much lower update rate, the trajectory data must first be decimated down to the radar update rate so as to produce the truth values of the observations prior to adding observation noise. We could also add a variable start time, but for our purposes that is not necessary, so we will start the observations at $t_0 = 0$. If a variable start time is used, and the start time contains more than one decimal place (e.g., $t_0 = 0.12345$ s), one must use interpolation to generate the Cartesian samples from the trajectory data. Thus if our original trajectory contained 4000 samples, we will end up with 100 samples from which we generate observation sets.

Assume that the decimated spherical truth data is contained in the variable Polar Truth (of dimension $P \times N$, where P is the dimension of the observation vector and N is the number of time samples), then noise is added to the truth trajectory samples using the Matlab snippet contained in Listing 18.1 (Stream is a placeholder for the random number stream found in the latest version of randn function).

Listing 18.1 Adding Random Observation Noise

```
Observations = PolarTruth + sqrtm(R)*randn(Stream,P,N);
```

A separate set of observations is generated for each Monte Carlo run. The final set of observations therefore contains N samples for each Monte Carlo run. This procedure can be completed offline and independent of the tracker performance computations.

18.3.3 Filter Initialization

All sequential Kalman tracking filters require initialization of both the state vector and its covariance matrix, $\hat{x}_{0|0}$ and $P_{0|0}^{xx}$, respectively. Assuming that one has access to a stream of sensor data (either simulated, as described above, or real), the easiest way to generate the required initialization values is to utilize the first few sensor observation data values to create an initial estimate of a Cartesian state vector.

If a full noisy spherical polar six-dimensional state vector were available as an observation vector, we could generate an initial Cartesian state vector by simply averaging a few samples of noisy spherical state vector data and then use the polar-to-Cartesian transformation defined by (18.195). But we will have, *at most*, four of the six spherical components available in the data vector.

Therefore, since all components of a spherical state vector will never be available from the observations, one needs to set up default values for those spherical components that cannot be obtained directly from the available sensor dependent data. The easiest way to accomplish this is to generate a set of nominal values for all the components of both the spherical state vector and their corresponding standard deviations. Generally, all sensors provide bearing and bearing uncertainty so we will not set nominal values for these but instead assume that they will always be available from the sensor. We have created a nominal set of spherical initialization values that will be used for all track estimation filters used for performance analysis. These are listed in Table 18.1. All off-diagonal terms in the spherical initialization covariance matrix will be set to zero.

TABLE 18.1 Nominal Values for Initialization State Vector and Covariance Components

	Nominal Spherical Vector Components	Nominal Standard Deviations
Range (nmi)	10	0.1
Range rate (knots)	-600	5
Bearing (rad)	Available from sensor	Available from sensor
Bearing rate (rad/s)	0	10^{-4}
Elevation (rad)	0	10^{-2}
Elevation rate (rad/s)	0	10^{-4}

When observation data is available for one or more of the spherical components, initialization values can be computed from several successive samples of the data. For position data (R, θ, ϕ) , we generate an initialization value by averaging several consecutive data values, for example, for range we could average the first two data samples

$$R_0 = \frac{1}{2} (R_1 + R_2) \quad (18.108)$$

For the rate components, we could use two of the first three data points to compute the central difference formula from (2.44), for example, range rate could be computed from

$$\dot{R}_0 = \frac{1}{2} (R_3 - R_1) \quad (18.109)$$

For cases where one is processing asynchronous multisensor data, some modifications must be made to account for the fact that some components might be available from one sensor but not another.

Once complete spherical state vector and covariance matrices are available for initialization, a simple spherical-to-Cartesian transformation is made using the formulas from Appendix APPENDIX 18.B. This results in the initialization Cartesian state vector and covariance matrix. We always ensure that the initialization state vector and covariance matrix set is given in Cartesian coordinates and is the same for all Monte Carlo runs and all track estimation methods.

For the spherical constant rate filter, these Cartesian initialization values must be converted back to spherical to generate the set $\{\hat{\mathbf{r}}_{0|0}, \mathbf{P}_{0|0}^{\text{rr}}\}$. Once these are available, they can be converted to the initialization set $\{\hat{\rho}_{0|0}, \mathbf{P}_{0|0}^{\rho\rho}\}$ for the constant velocity spherical filters. Note that the transformation from \mathbf{r} to ρ is given by

$$\rho = \begin{bmatrix} R \\ v_R \\ \theta \\ v_h \\ \phi \\ v_v \end{bmatrix} = \mathbf{g}(\mathbf{r}) = \begin{bmatrix} R \\ \dot{R} \\ \theta \\ \dot{\theta}R \cos \phi \\ \phi \\ \dot{\phi}R \end{bmatrix} \quad (18.110)$$

and that

$$\mathbf{P}^{\rho\rho} = \hat{\mathbf{G}} \mathbf{P}^{\text{rr}} \hat{\mathbf{G}}^\top \quad (18.111)$$

where $\hat{\mathbf{G}}$ is the Jacobian of the transformation $\mathbf{g}(\mathbf{r})$. Evaluating the partials of the Jacobian leads to

$$\hat{\mathbf{G}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \dot{\theta} \cos \phi & 0 & 0 & R \cos \phi & -R\dot{\theta} \sin \phi & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \dot{\phi} & 0 & 0 & 0 & 0 & R \end{bmatrix} \quad (18.112)$$

18.4 PERFORMANCE COMPARISON FOR VARIOUS ESTIMATION METHODS

18.4.1 Characteristics of the Trajectories Used for Performance Analysis

For the purpose of analysis, three target motion scenarios were created using the method outlined in Appendix APPENDIX 18.A. The trajectories associated with each of the three scenarios are shown in Figures 18.1–18.3.

The first scenario, with the trajectory shown in Figure 18.1, is for a radially moving target that has a constant Cartesian velocity. For this trajectory, the target moves radially inbound at a constant height of 1 nmi (height not shown in Figure 18.1) and passes directly over the radar sensor located at $(0, 0, 0)$. The second scenario, with the trajectory shown in Figure 18.2, consists of a target that executes two turns, with a 90 deg 2 g turn in the first loop and a 90 deg 5 g turn in the second loop. The third

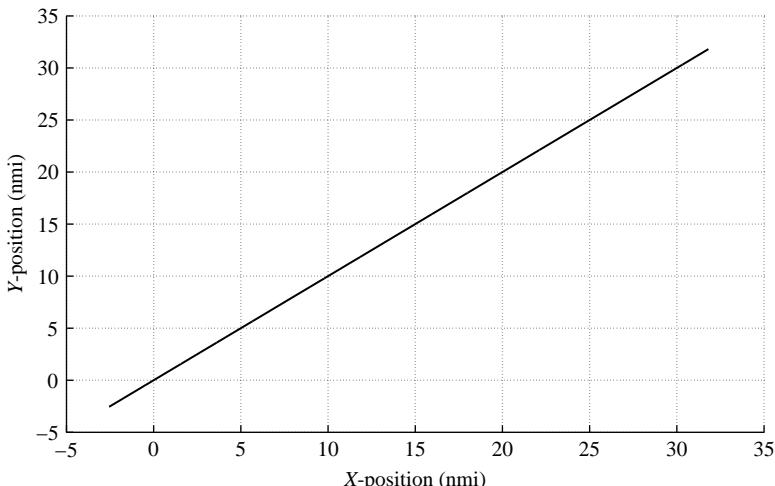


FIGURE 18.1 A simulated radially inbound target truth track.

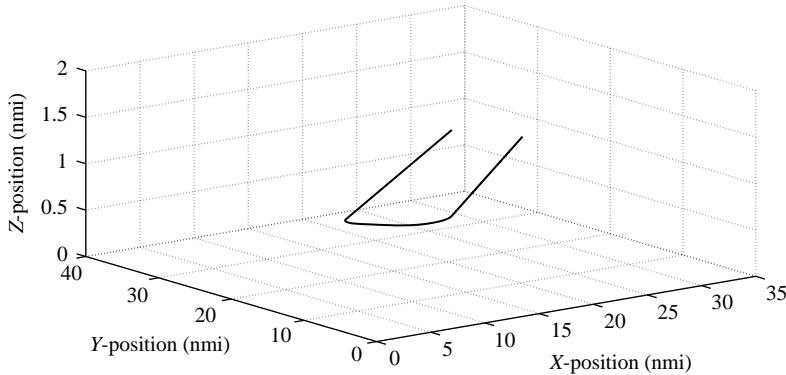


FIGURE 18.2 A simulated horizontally looping target truth track.

trajectory is for a highly maneuvering target that undergoes a 7 g turn followed by a 6 g turn, a deceleration and 5 g dive, a leveling off and an acceleration followed by a final 7 g turn, as shown in Figure 18.3. All three of these scenarios are similar to those considered by Zollo and Ristic [5].

The specific initial characteristics of all three scenarios are presented in Table 18.2. For all tracking filters used in this performance analysis, the values for the components of q were set to $\{q_x = q_y = 225; q_z = 1.5\}$ for the Cartesian trackers, $\{q_r = q_h = 225; q_v = 1.5\}$ for the spherical constant velocity trackers, and $\{q_r = 225; q_\theta = q_\phi = 1.5\}$ for the spherical constant rate filters. The q values of 225 or 1.5 correspond to expected maximum turn/dive accelerations of 7.5 g's or 0.61 g's, respectively.

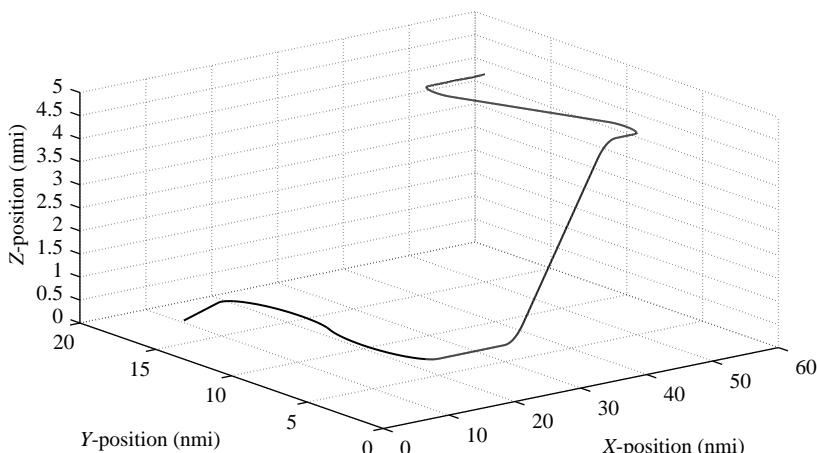
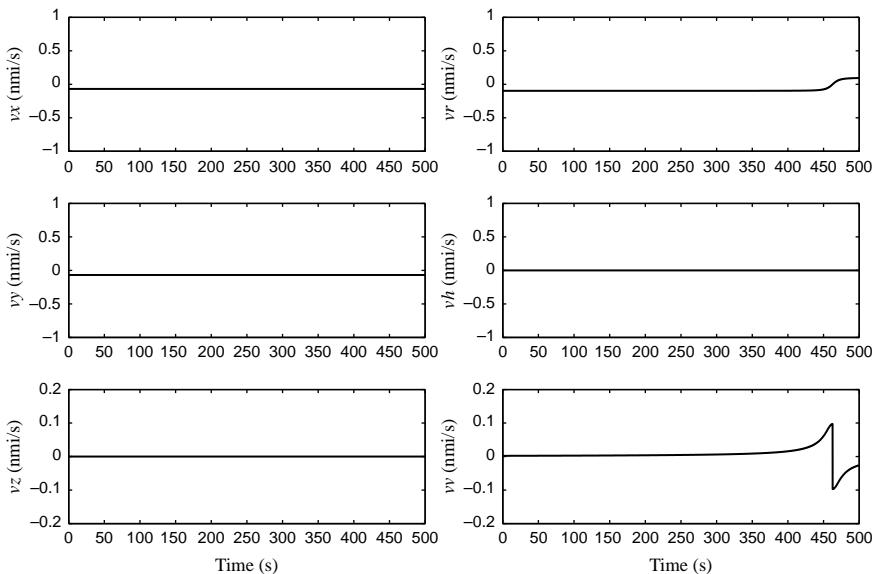


FIGURE 18.3 A simulated benchmark target truth track.

TABLE 18.2 Initial Characteristics of Three Simulated Scenarios

	Trajectory		
	Radial Trajectory	Loop	Benchmark
Speed (knots)	350	700	700
Initial position (nmi) (x, y, z)	(31.82, 31.82, 1)	(30.78, 30.78, 1)	(50, 20, 3)
Initial range (nmi)	45	40	58.58
Initial bearing	45°	45°	70°
Initial heading	-135°	-135°	-90°
Initial attitude	0°	0°	0°

We first examine the velocity characteristics of the trajectories in each scenario to demonstrate how well the trajectories conform to either of the constant velocity models. Figures 18.4–18.6 show the components of both the Cartesian and spherical velocities for the three scenarios. From Figure 18.4, it is obvious that the velocities are constant for the radially inbound target for the Cartesian velocities but spherical velocity components for range and vertical velocity show variations in the region where the track crosses over the sensor at the origin where they both undergo a sudden change in sign. Therefore, we should expect little difference in performance between the two tracker models in regions away from the origin but some difficulty with the spherical trackers as the object nears the origin. Although not shown, the

**FIGURE 18.4** Components of both the cartesian and spherical velocities for the radially inbound trajectory.

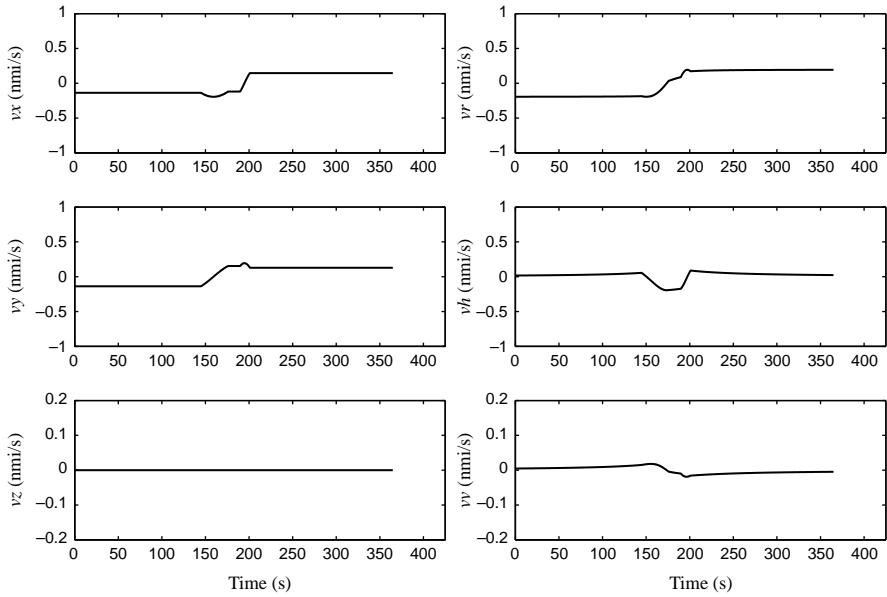


FIGURE 18.5 Components of both the Cartesian and spherical velocities for the loop trajectory.

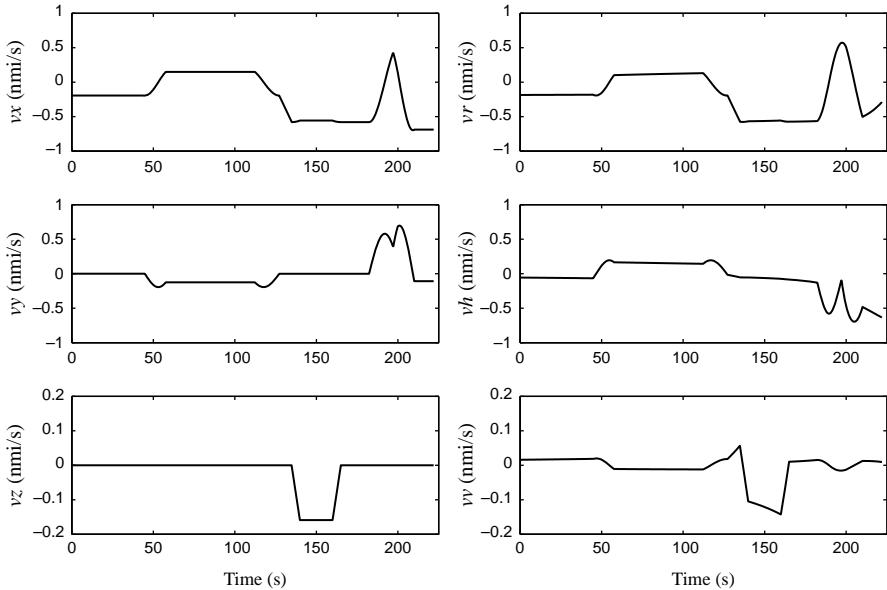


FIGURE 18.6 Components of both the Cartesian and spherical velocities for the benchmark trajectory.

rates for the constant rate spherical filter show large changes at short ranges due to sudden changes as the target goes through the origin.

The velocity comparison for the loop trajectory scenario, shown in Figure 18.5, indicates that the velocities show similar variations in both Cartesian and spherical, so there should be little difference in the tracker performance.

Finally, for the highly maneuvering trajectory, the velocity components shown in Figure 18.6 indicate that the spherical trackers should perform as well as the Cartesian trackers, due to the similar variations in the velocities as a function of time.

18.4.2 Filter Performance Comparisons

Since we use the same track estimation filters (EKF, UKF, SSKF, and GHKF) for both the Cartesian and spherical trackers, we differentiate them by using the precursors C- and S- for the Cartesian and spherical track algorithms, respectively. Four *Cartesian* Kalman filters, the C-EKF, C-UKF, C-SSKF, and C-GHKF, were used to generate track estimates for the Cartesian constantvelocity model and four *spherical* Kalman filters, S-EKF, S-UKF, S-SSKF, and S-GHKF, were utilized to generate estimated tracks for the spherical constant velocity model. The S-LKF was used for the spherical constant rate long-range approximation model.

In each case, 100 Monte Carlo runs were performed. For the Cartesian tracking algorithms, the Cartesian estimated track outputs were converted to spherical coordinates and for the spherical trackers, the spherical track outputs were converted to Cartesian coordinates, with all coordinate transformations accomplished using the algorithms contained in Appendix 18.B. Thus, for all trackers, both Cartesian and spherical track estimates were generated. Based on the 100 Monte Carlo runs, RMS errors were calculated for the position estimates in both Cartesian and spherical coordinates.

18.4.2.1 Track Performance Comparison for the Radially Inbound Target Trajectory. Figures 18.7 and 18.8 show the three-dimensional estimated tracks for the first Monte Carlo run of the radially inbound target for the Cartesian and spherical Kalman filters, respectively, along with the true track (in gray). One can immediately see from a comparison of these estimated track plots that the spherical trackers appear to perform better than the Cartesian trackers as the target passes over the sensor located at (0,0,0). With the exception of the C-EKF, all of the Cartesian filters perform poorly in the region near the origin.

The reason for this divergent behavior for the Cartesian track estimators near the origin can be found in the nonlinear observation equations (18.10) of the Cartesian track estimation methods. Although the particulars are beyond the scope of this book, it can be shown that the inverse tangent operations contained in (18.10) create this divergence. As the magnitude of the Cartesian position components approach the magnitude of the estimated position uncertainties, the probability density function of the inverse tangent of the ratio of any two position components approaches a uniform distribution. Thus predictions of both bearing and elevation are governed by larger and larger standard deviations, causing the innovations vector ($\mathbf{z}_n - \hat{\mathbf{z}}_{n|n-1}$) in (18.26) to become unstable. This instability in the innovations vector results in filter divergence.

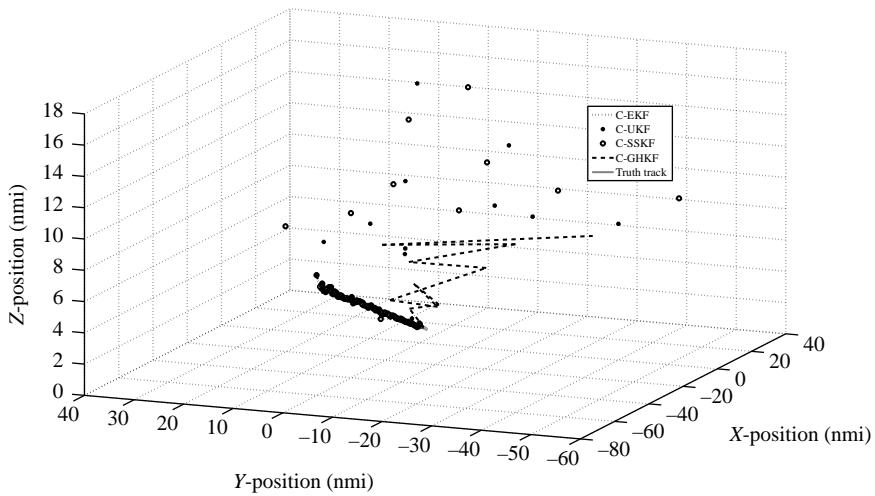


FIGURE 18.7 Estimated tracks for the radially inbound target trajectory for all Cartesian tracking methods.

However, in regions not near the origin, the Cartesian filters outperform the spherical filters, as expected.

For the radially inbound target trajectory, the RMS *Cartesian* position errors for the Cartesian and spherical tracking algorithms can be compared by examining Figure 18.9, while the RMS *spherical* position errors for the two classes of tracking algorithms can be seen in Figure 18.10. For these plots, we set $q_x = q_y = q_r = q_h = 225$

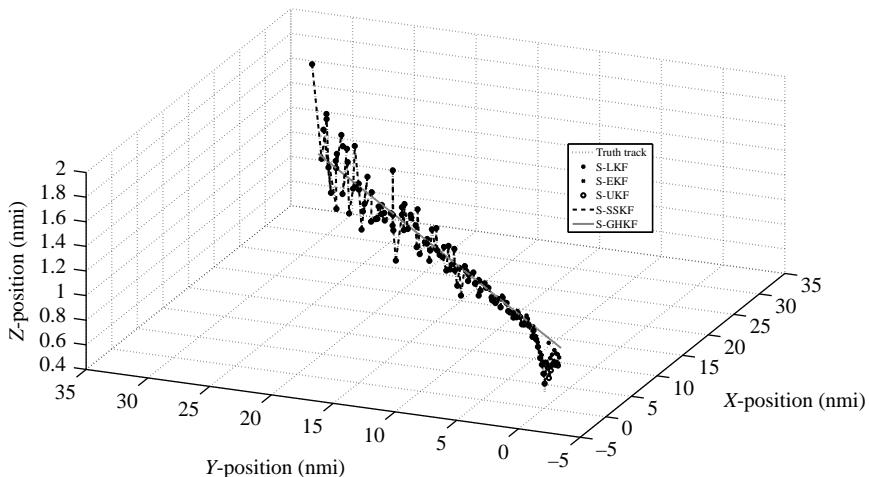


FIGURE 18.8 Estimated tracks for the radially inbound target trajectory for all spherical tracking methods.

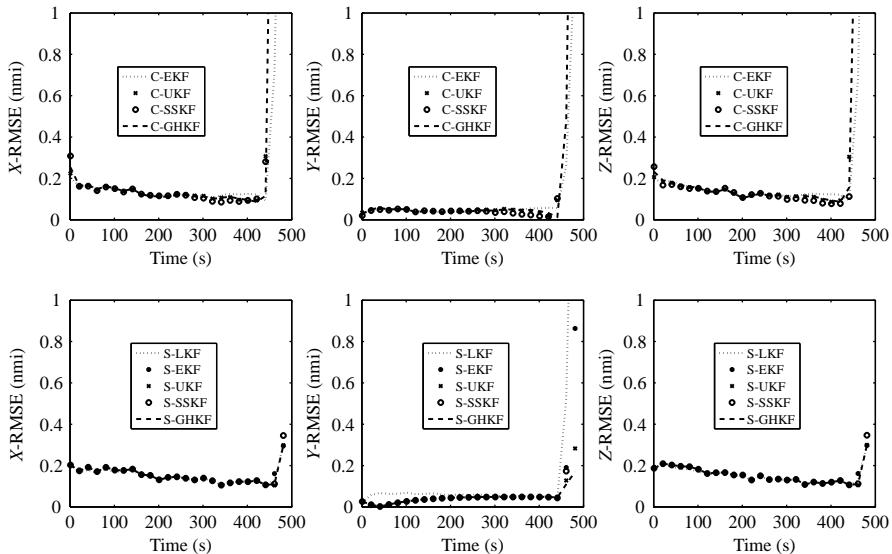


FIGURE 18.9 RMS Cartesian position errors of the radially inbound target trajectory for the Cartesian and spherical tracking algorithms.

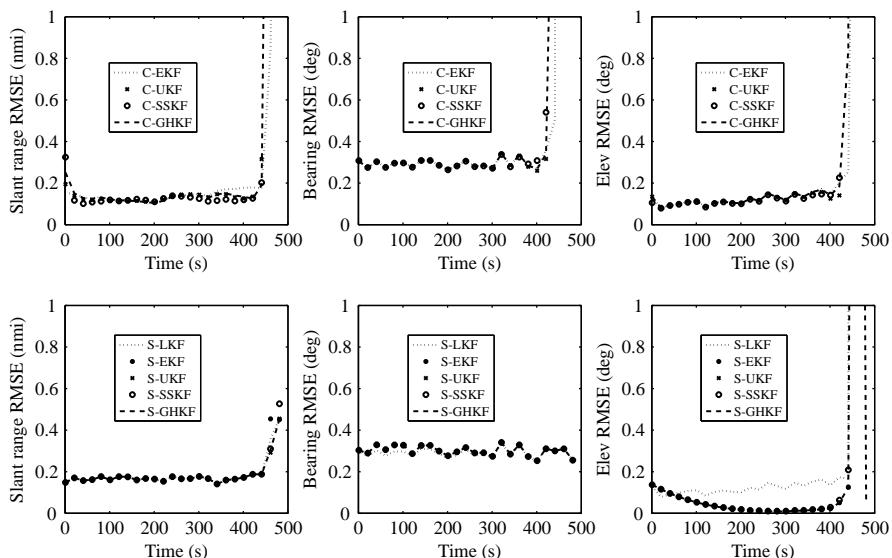


FIGURE 18.10 RMS spherical position errors of the radially inbound target trajectory for the Cartesian and spherical tracking algorithms.

and all other q values were set to 1.5. It is obvious that all trackers perform equally well in estimating the Cartesian tracks except in the region where the target passes over the radar, beginning at approximately 410 s. After this point, the spherical trackers have the better performance, with the S-GHKF performing the best. In this close-in tracking region, the spherical trackers exhibit large RMS errors in elevation but show nominal RMS errors in all other Cartesian and spherical variables.

We now examine the impact of variations in the q values on the RMS position error performance for a radially inbound target trajectory. First, notice that in Figures 18.9 and 18.10 there is little difference in performance among the Cartesian class of filters taken as a group or among the spherical filters. To make examination of the effects of varying q simpler, we only compare the performance results for Cartesian and spherical EKF algorithms as q varies. Although not shown here, an initial study examined variations in the values of q_z , q_{angular} , and q_v . We set them all to one of three values: 1.5, 36, and 225. It was clear from the analysis that the best choice over all trajectories was 1.5.

Since the radially inbound trajectory consists of an object that maintains a constant velocity throughout the trajectory, one would think that the RMS position errors would be proportional to the value of q , that is, as q increased, the RMS errors would increase. However, examination of Figures 18.11 and 18.12 indicate that the best performance for the Cartesian track estimators occurs when the horizontal values for q are 36 (upper graphs), while for the spherical trackers the best choice is 550 (lower graphs). Note that for the Cartesian filter algorithms, q_r and q_a represent the horizontal and vertical values of q , respectively. All of the filters seem to have large RMS errors for

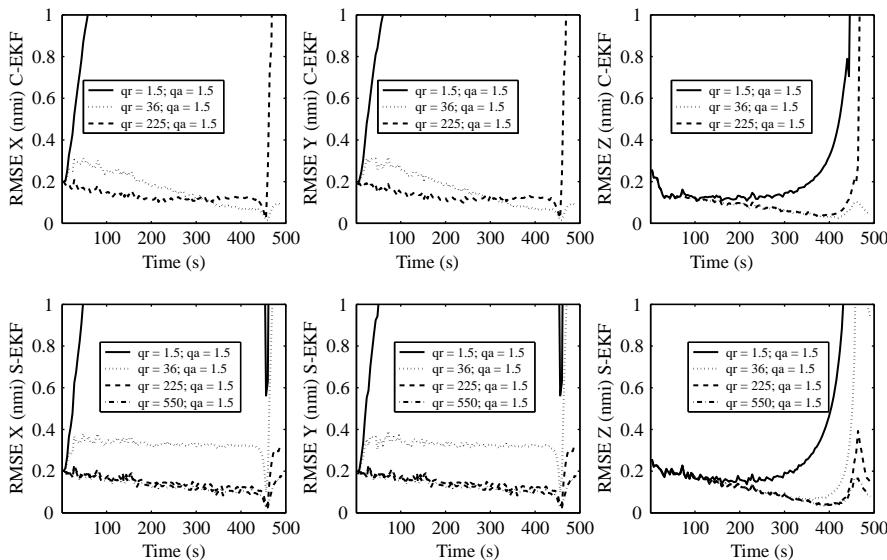


FIGURE 18.11 Comparison of the radially inbound trajectory RMS Cartesian position errors for varying values of q for both Cartesian and spherical EKF filters.

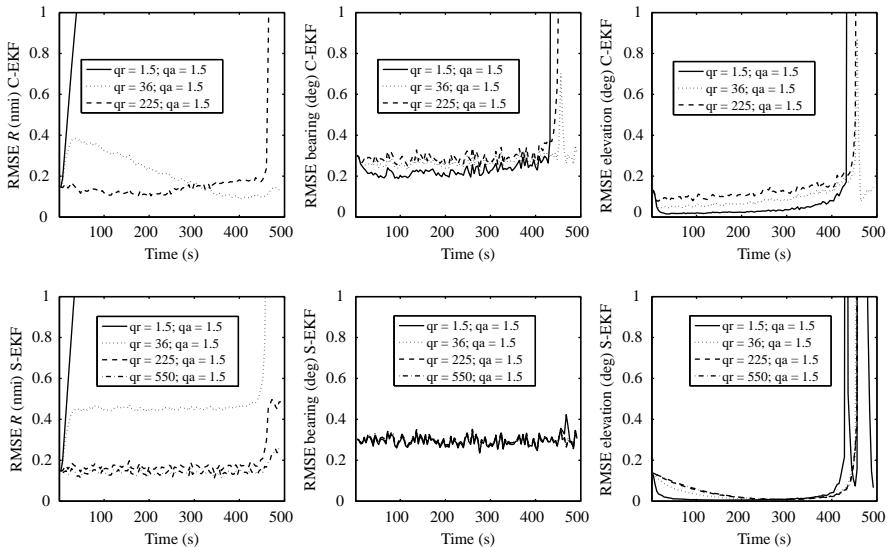


FIGURE 18.12 Comparison of the radially inbound trajectory RMS spherical position errors for varying values of q for both Cartesian and spherical EKF filters.

the estimation of elevation near the origin, with the spherical filters having slightly better performance. The best overall performance appears to be with the S-EKF with a $qr = 550$.

18.4.2.2 Track Performance Comparison for the Loop Target Trajectory. For the loop trajectory scenario, examples of one estimated track output along with the truth track are shown in Figures 18.13 and 18.14 for the Cartesian and spherical track algorithms, respectively. Examination of the figures shows that all trackers seem to do well but lag slightly in regions of high- g turns, with the Cartesian trackers showing less variation in regions away from the turns.

The charts for the RMS position errors, shown in Figures 18.15 and 18.16, reveal that all of the track estimation algorithms have similar performance, with the Cartesian track algorithms having slightly better performance overall. The C-SSKF is the worst performing track estimation algorithm among the Cartesian trackers and the S-LKF has the best performance among the spherical trackers. For these two plots, we set $qx = qy = qr = qh = 225$ and all other q values were set to 1.5.

Once again, to examine the effects of varying the value of q , we examine the performance of only the Cartesian and spherical EKF with q_z , q_{angular} , and q_v set to 1.5. The results shown in Figures 18.17 and 18.18 for the RMS Cartesian and spherical position errors, respectively, indicate that the best results are obtained when the horizontal q values are set to 225 and 550 for the Cartesian and spherical EKFs, respectively.

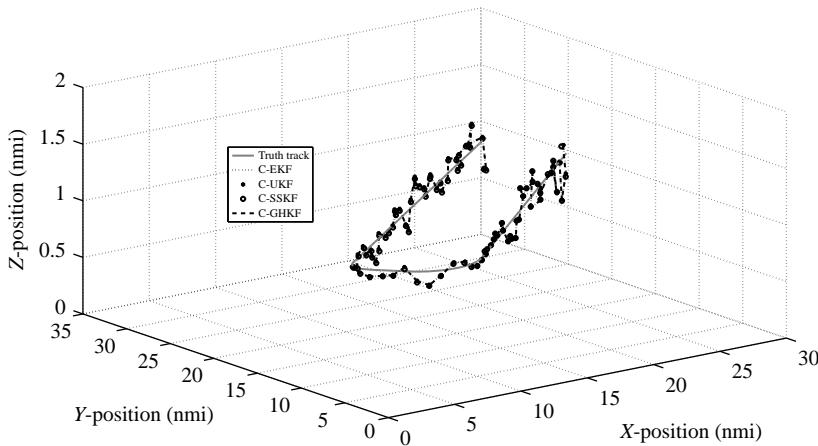


FIGURE 18.13 Estimated tracks for the looping target trajectory for all Cartesian tracking methods.

18.4.2.3 Track Performance Comparison for the Benchmark Highly Maneuvering Target Trajectory. An example of track estimates for the final trajectory, one with several high- g maneuvers and both deceleration and acceleration sections, are shown in Figures 18.19 and 18.20 for the Cartesian and spherical track algorithms, respectively. Comparison of the two figures show that all track algorithms, both Cartesian and spherical, appear to do well until the acceleration phase at the bottom of the charts. This results in an overshoot by the Cartesian trackers during the final turn after the acceleration section. On the other hand, the spherical trackers have no problems with

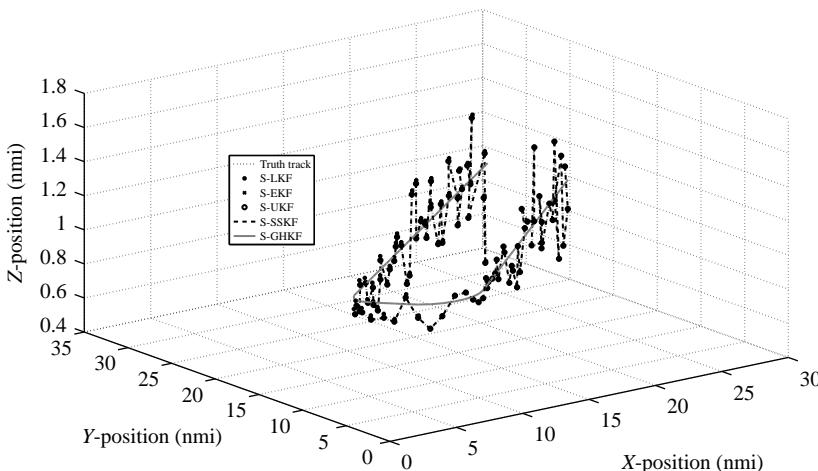


FIGURE 18.14 Estimated tracks for the looping target trajectory for all spherical tracking methods.

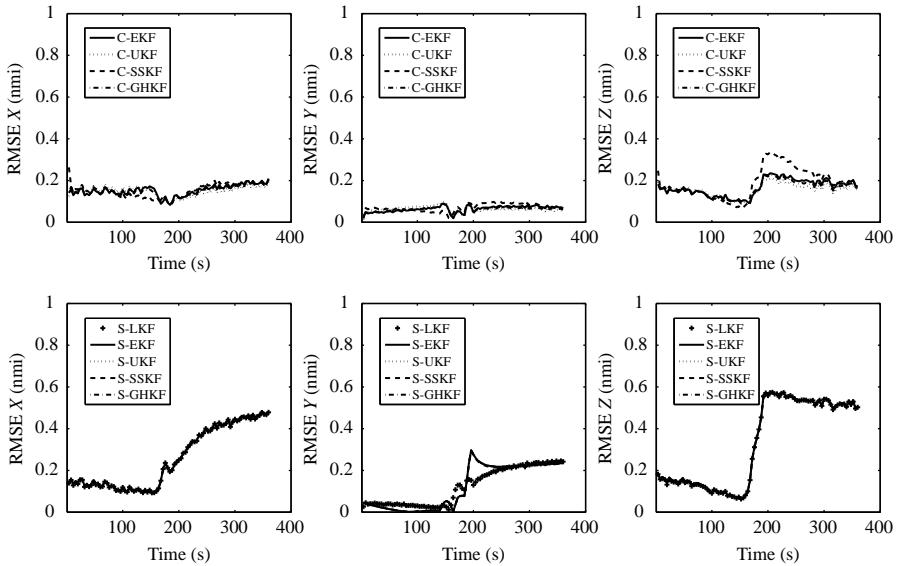


FIGURE 18.15 RMS Cartesian and spherical position errors of the loop target trajectory for the Cartesian tracking algorithms.

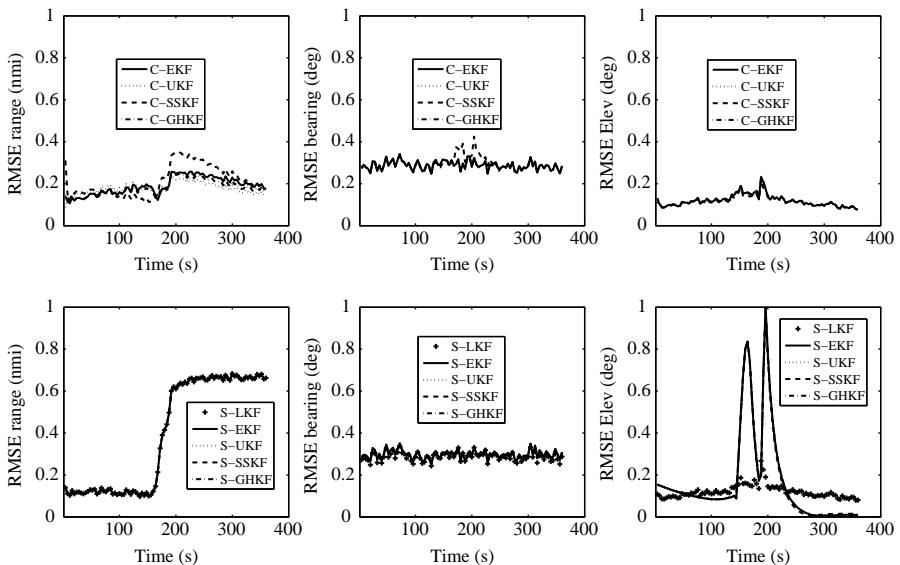


FIGURE 18.16 RMS Cartesian and spherical position errors of the loop target trajectory for the spherical tracking algorithms.

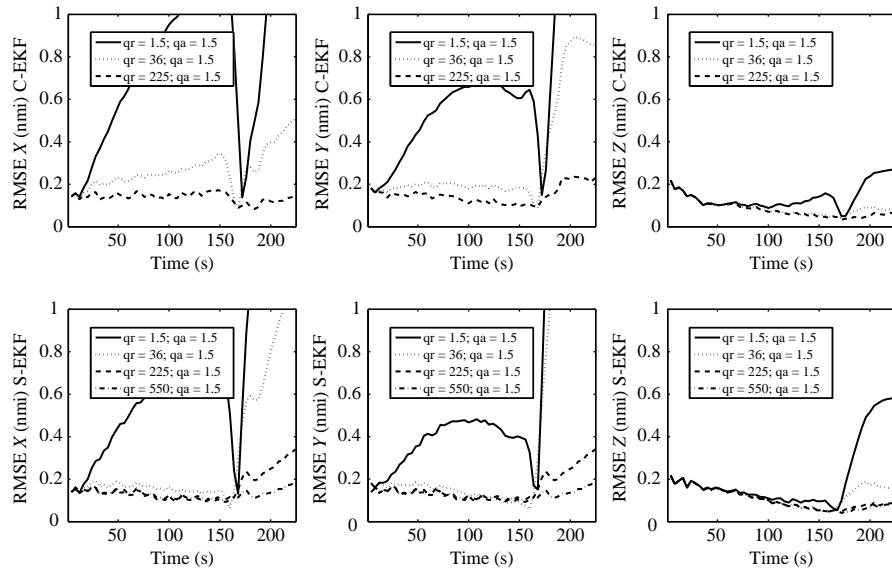


FIGURE 18.17 Comparison of RMS Cartesian position error performance as a function of q for the loop trajectory.

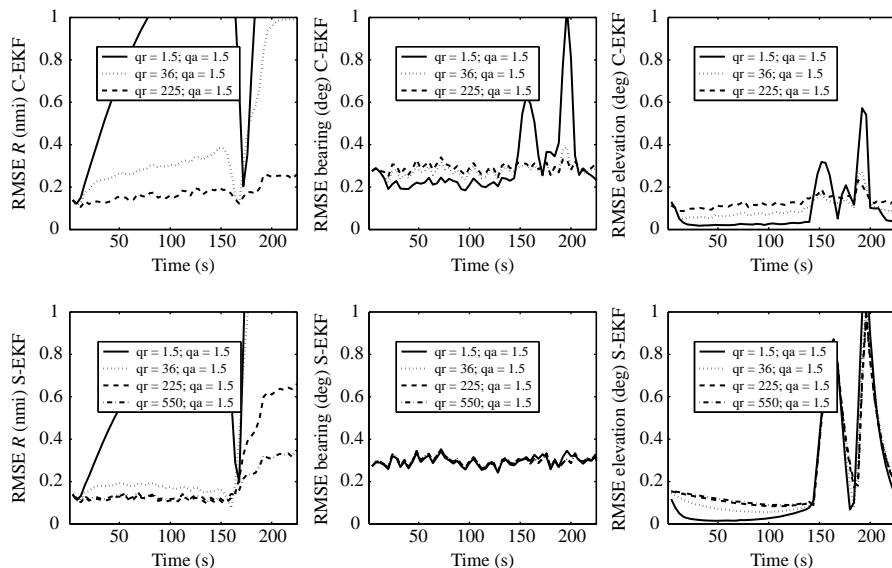


FIGURE 18.18 Comparison of RMS spherical position error performance as a function of q for the loop trajectory.

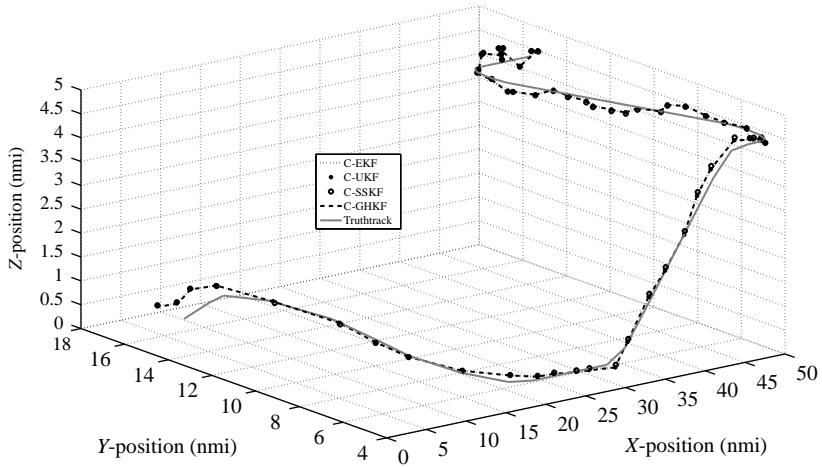


FIGURE 18.19 Estimated tracks for the benchmark target trajectory for all Cartesian tracking methods.

the final turn. From these figures it is hard to ascertain the best track performance among the track algorithms.

Comparison of the RMS position errors shown in Figures 18.21 and 18.22, for the Cartesian and spherical RMS position errors, respectively, indicate that for a highly maneuvering target the spherical tracking algorithms result in slightly lower RMS Cartesian and range position errors than do the Cartesian track algorithms. In addition, the spherical track estimation algorithms have better performance in

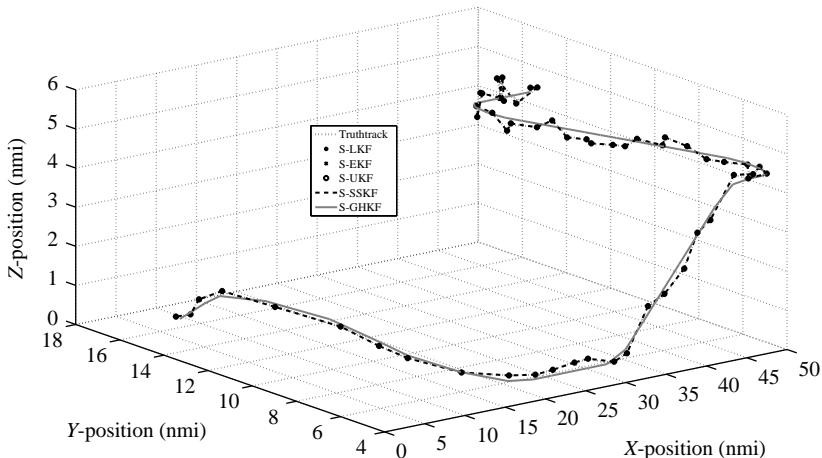


FIGURE 18.20 Estimated tracks for the benchmark target trajectory for all spherical tracking methods.

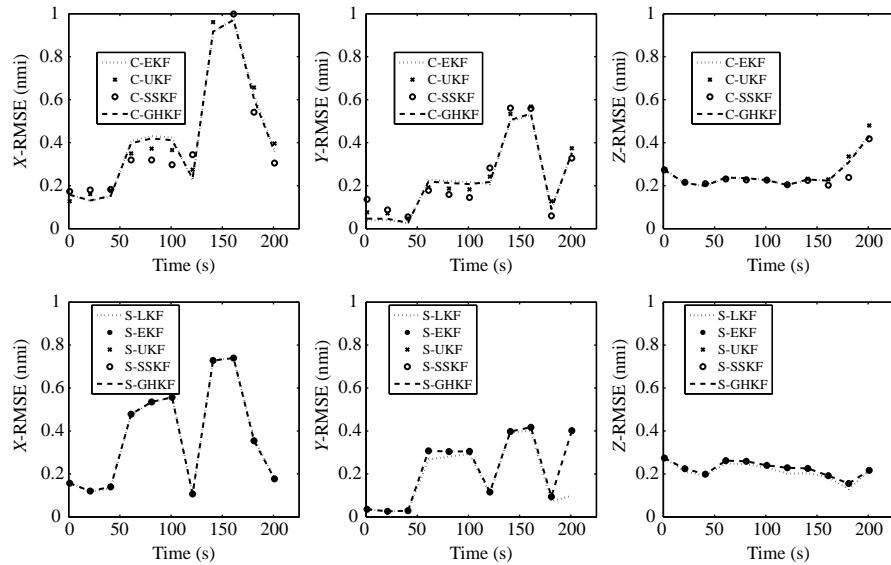


FIGURE 18.21 RMS Cartesian and spherical position errors of the benchmark target trajectory for the Cartesian tracking algorithms.

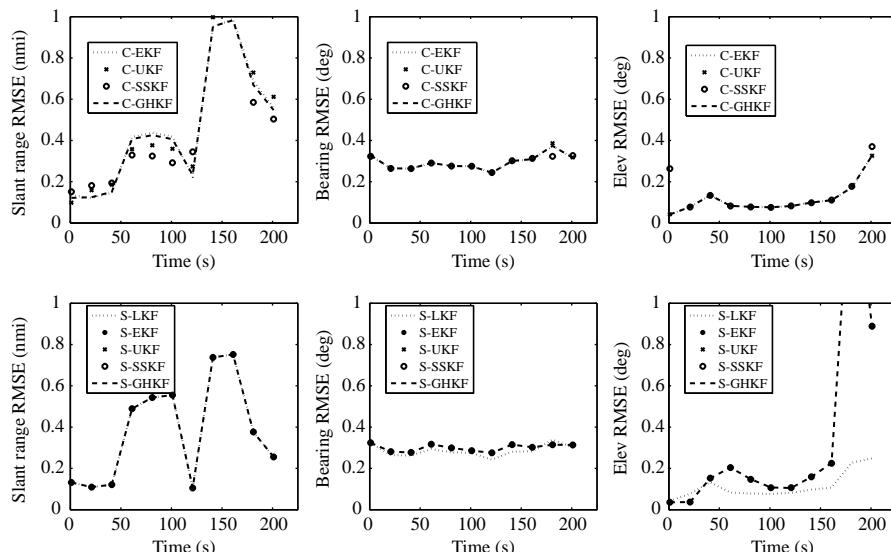


FIGURE 18.22 RMS Cartesian and spherical position errors of the benchmark target trajectory for the spherical tracking algorithms.

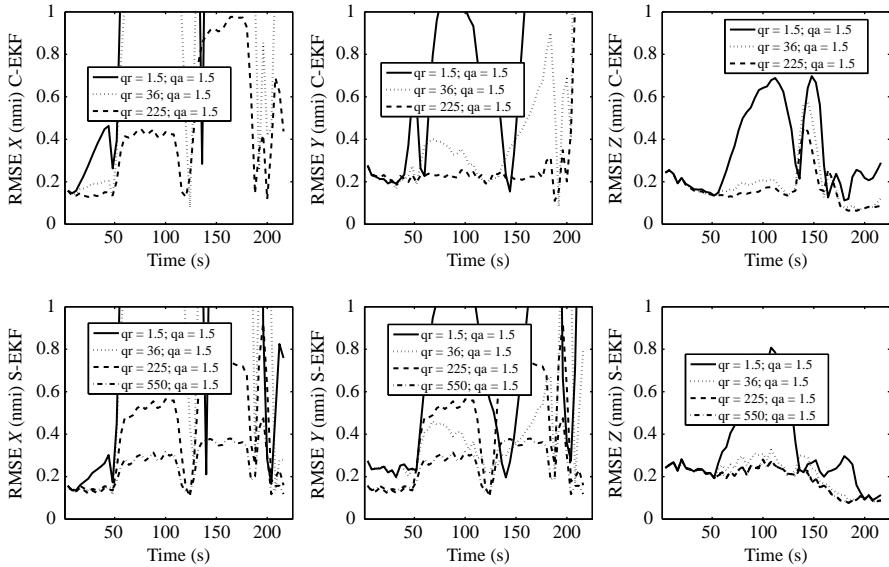


FIGURE 18.23 Comparison of RMS Cartesian position error performance as a function of q for the benchmark trajectory.

tracking bearing but worse performance in tracking elevation. Overall, it is obvious from these charts that the spherical track filter algorithms produce track estimates that are comparable and slightly better than the Cartesian track estimation algorithms for tracking a maneuvering target. It should also be noted that the constant spherical rate filter has the best performance among all of the filters. This is not surprising for this long-range tracking scenario where the target object comes no closer in range than 20 nmi. For these two plots, we set $q_x = q_y = q_r = q_h = 225$ and all other q values were set to 1.5.

In examining the effects of varying the value of q for this trajectory, once again we examine the performance of only the Cartesian and spherical EKF with q_z , q_{angular} , and q_v set to 1.5. The results shown in Figures 18.23 and 18.24 for the RMS Cartesian and spherical position errors, respectively, indicate that the best results are obtained when the horizontal q values are set to 225 and 550 for the Cartesian and spherical EKFs, respectively.

18.4.2.4 Conclusions. From a consideration of the tracker algorithm RMS position error performance for three different target trajectories with maneuver accelerations increasing from trajectory to trajectory, we can conclude that the new spherical constant velocity dynamic model offers advantages over the Cartesian constant velocity model for tracking a maneuvering target, as long as the values for the components of q are chosen properly. In addition, the new trackers show improved performance for

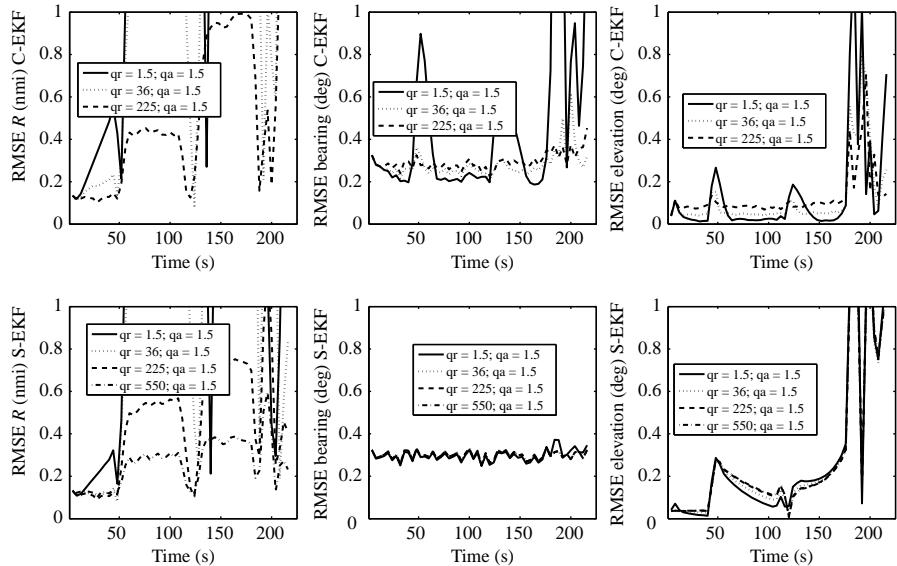


FIGURE 18.24 Comparison of RMS spherical position error performance as a function of q for the benchmark trajectory.

targets that approach the region where R is small, especially for estimation of bearing and elevation.

18.5 SOME OBSERVATIONS AND FUTURE CONSIDERATIONS

This case study considered a new constant spherical velocity dynamic target motion model and the resultant Kalman filter track algorithms that can be used with this model. Through a performance analysis with three simulated trajectories, where the level of target maneuvering increased from trajectory to trajectory, we showed that spherical track estimation filters had the better performance during target maneuvers and when the target approached zero range. We also showed that, with the exception of the S-LKF, all of the spherical track estimation algorithms had identical performance. This latter consequence leads to the conclusion that the S-EKF should be used for constant spherical velocity tracking since it has the shortest computation time.

In this case study, we did not fully explore the potential of this new dynamic model due to space and time limitations. Some useful expansions on this study could include the following:

- Exploring the potential use of a spherical IMM filter that would mix a high- q model with a low- q model, and
- Explore the impact of using range rate observations on the performance of the spherical filters relative to the Cartesian filters.

APPENDIX 18.A THREE-DIMENSIONAL CONSTANT TURN RATE KINEMATICS

In this section, we develop a method to generate a trajectory that can include constant turn-rate turns in both the horizontal and the vertical. This will allow us to create moving object trajectories that contain horizontal turns, vertical dives, and spirals.

Assume that an object at time t is moving at a horizontal heading $\vartheta(t)$ and at a vertical attitude of $\gamma(t)$. Consider the Cartesian East–North–Up coordinate system shown in Figure 18.25, where the East–North plane is the horizontal plane.

18.A.1 General Velocity Components for Constant Turn Rate Motion

The Cartesian components of velocity for constant speed v in an East–North–Up Cartesian system can be expressed as

$$\mathbf{v}(t) = \begin{bmatrix} v_x(t) \\ v_y(t) \\ v_z(t) \end{bmatrix} = \begin{bmatrix} v \sin \vartheta(t) \cos \gamma(t) \\ v \cos \vartheta(t) \cos \gamma(t) \\ v \sin \gamma(t) \end{bmatrix} \quad (18.113)$$

where ϑ and γ represent the heading ($-180^\circ \leq \vartheta < 180^\circ$) and attitude ($-90^\circ \leq \gamma \leq 90^\circ$), respectively.

For a constant rate turn, referring to Figure 18.26, we can write

$$\vartheta(t) = \vartheta(t_0) + \Omega(t - t_0) \triangleq \vartheta_0 + \Omega T \quad (18.114)$$

$$\gamma(t) = \gamma(t_0) + \Psi(t - t_0) \triangleq \gamma_0 + \Psi T \quad (18.115)$$

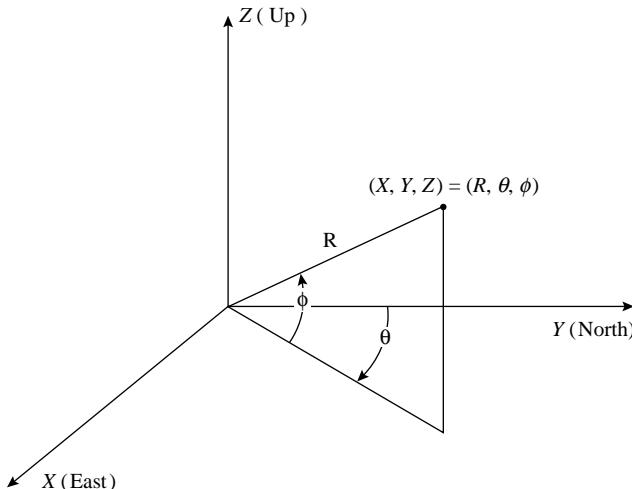


FIGURE 18.25 The East–North–Up Cartesian coordinate system.

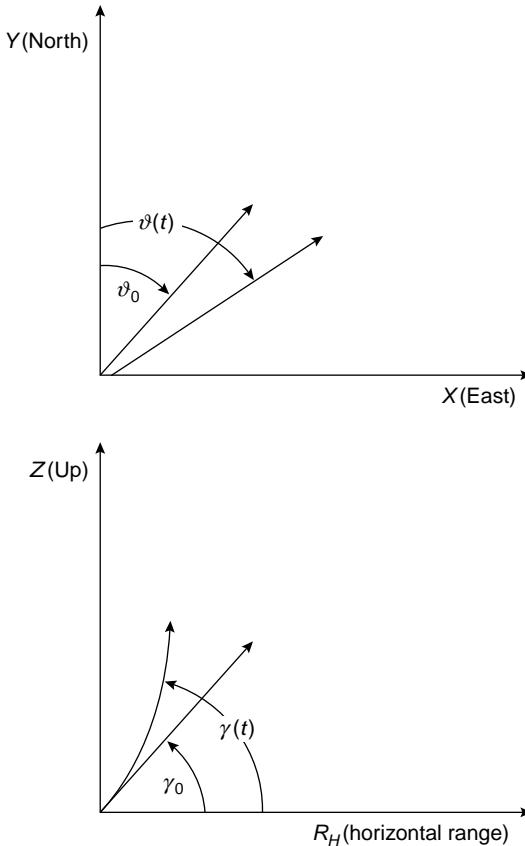


FIGURE 18.26 Depiction of a constant rate turn in both the horizontal and vertical planes.

where $T \triangleq t - t_0$ and $\vartheta(t_0) \rightarrow \vartheta_0$, $\gamma(t_0) \rightarrow \gamma_0$. Here, Ω and Ψ represent the constant turn rates in the horizontal and vertical directions, respectively. Note that Ω is positive for turns to the right relative to the direction of motion and Ψ is positive for a turn up. Substituting (18.114) and (18.115) into (18.113) results in

$$\begin{bmatrix} v_x(t) \\ v_y(t) \\ v_z(t) \end{bmatrix} = \begin{bmatrix} v \sin(\vartheta_0 + \Omega T) \cos(\gamma_0 + \Psi T) \\ v \cos(\vartheta_0 + \Omega T) \cos(\gamma_0 + \Psi T) \\ v \sin(\gamma_0 + \Psi T) \end{bmatrix} \quad (18.116)$$

Expanding the $v_x(t)$ component of the velocity using the trigonometric identities for sine and cosine of a sum of angles we obtain

$$\begin{aligned} v_x(t) &= v \sin(\vartheta_0 + \Omega T) \cos(\gamma_0 + \Psi T) \\ &= v [\sin \vartheta_0 \cos \Omega T + \cos \vartheta_0 \sin \Omega T] [\cos \gamma_0 \cos \Psi T - \sin \gamma_0 \sin \Psi T] \end{aligned}$$

$$\begin{aligned}
&= v \sin \vartheta_0 \cos \gamma_0 \cos \Omega T \cos \Psi T \\
&\quad - v \sin \vartheta_0 \sin \gamma_0 \cos \Omega T \sin \Psi T \\
&\quad + v \cos \vartheta_0 \cos \gamma_0 \sin \Omega T \cos \Psi T \\
&\quad - v \cos \vartheta_0 \sin \gamma_0 \sin \Omega T \sin \Psi T
\end{aligned} \tag{18.117}$$

Using the definition of the components of \mathbf{v} from (18.113), this becomes

$$\begin{aligned}
v_x(t) &= v_x(t_0) \cos \Omega T \cos \Psi T + v_y(t_0) \sin \Omega T \cos \Psi T \\
&\quad - v_z(t_0) [\cos \vartheta_0 \sin \Omega T \sin \Psi T \\
&\quad + \sin \vartheta_0 \cos \Omega T \sin \Psi T]
\end{aligned} \tag{18.118}$$

Now, using the trigonometric product-to-sum identities [9] this can be rewritten as

$$\begin{aligned}
v_x(t) &= \frac{1}{2} \left\{ v_x(t_0) \cos \xi_1 T + v_y(t_0) \sin \xi_1 T \right. \\
&\quad - v_z(t_0) [\cos \vartheta_0 \cos \xi_1 T - \sin \vartheta_0 \sin \xi_1 T] \\
&\quad + v_x(t_0) \cos \xi_2 T + v_y(t_0) \sin \xi_2 T \\
&\quad \left. + v_z(t_0) [\cos \vartheta_0 \cos \xi_2 T - \sin \vartheta_0 \sin \xi_2 T] \right\}
\end{aligned} \tag{18.119}$$

where

$$\xi_1 \triangleq \Omega - \Psi \tag{18.120}$$

$$\xi_2 \triangleq \Omega + \Psi \tag{18.121}$$

Similarly, we obtain

$$\begin{aligned}
v_y(t) &= \frac{1}{2} \left\{ -v_x(t_0) \sin \xi_1 T + v_y(t_0) \cos \xi_1 T \right. \\
&\quad + v_z(t_0) [\cos \vartheta_0 \sin \xi_1 T + \sin \vartheta_0 \cos \xi_1 T] \\
&\quad - v_x(t_0) \sin \xi_2 T + v_y(t_0) \cos \xi_2 T \\
&\quad \left. - v_z(t_0) [\cos \vartheta_0 \sin \xi_2 T + \sin \vartheta_0 \cos \xi_2 T] \right\}
\end{aligned} \tag{18.122}$$

Now, examine $v_z(t)$

$$\begin{aligned}
v_z(t) &= v \sin (\gamma_0 + \Psi T) \\
&= v \sin \gamma_0 \cos \Psi T + v \cos \gamma_0 \sin \Psi T \\
&= v_z(t_0) \cos \Psi T + v \cos \gamma_0 \sin \Psi T
\end{aligned} \tag{18.123}$$

Note that

$$\begin{aligned} v_h(t_0) &= \sqrt{v_x^2(t_0) + v_y^2(t_0)} \\ &= \sqrt{v^2 \sin^2 \vartheta_0 \cos^2 \gamma_0 + v^2 \cos^2 \vartheta_0 \cos^2 \gamma_0} \\ &= v \cos \gamma_0 \end{aligned} \quad (18.124)$$

Thus (18.123) becomes

$$v_z(t) = v_z(t_0) \cos \Psi T + v_h(t_0) \sin \Psi T \quad (18.125)$$

Define

$$c_i \triangleq \cos(\xi_i T), \quad i = 1, 2 \quad (18.126)$$

$$s_i \triangleq \sin(\xi_i T), \quad i = 1, 2 \quad (18.127)$$

We can now rewrite (18.119) and (18.122) as

$$\begin{aligned} v_x(t) &= \frac{1}{2} \left\{ (c_1 + c_2) v_x(t_0) + (s_1 + s_2) v_y(t_0) \right. \\ &\quad \left. + [(s_1 - s_2) \sin \vartheta_0 - (c_1 - c_2) \cos \vartheta_0] v_z(t_0) \right\} \end{aligned} \quad (18.128)$$

$$\begin{aligned} v_y(t) &= \frac{1}{2} \left\{ -(s_1 + s_2) v_x(t_0) + (c_1 + c_2) v_y(t_0) \right. \\ &\quad \left. + [(c_1 - c_2) \sin \vartheta_0 + (s_1 - s_2) \cos \vartheta_0] v_z(t_0) \right\} \end{aligned} \quad (18.129)$$

However, there are some special cases that must be considered.

1. Consider the case where $\Omega = \Psi = 0$. Now $c_i = 1$ and $s_i = 0$, $i = 1, 2$, resulting in $\mathbf{v}(t) = \mathbf{v}(t_0)$.
2. When $\Omega = 0$, $\Psi \neq 0$, $\xi_1 = -\Psi$ and $\xi_2 = \Psi$ leading to $c_1 = c_2 = \cos(\Psi T)$, and $s_1 = -\sin(\Psi T)$, $s_2 = \sin(\Psi T)$.
3. When $\Psi = 0$, $\Omega \neq 0$, $\xi_1 = \xi_2 = \Omega$ leading to $c_1 = c_2 = \cos(\Omega T)$, and $s_1 = s_2 = \sin(\Omega T)$.
4. If $\Omega = \Psi \neq 0$, $\xi_1 = 0$ and $c_1 = 1$ and $s_1 = 0$.
5. If $\Omega = -\Psi \neq 0$, $\xi_2 = 0$ and $c_2 = 1$ and $s_2 = 0$.

18.A.2 General Position Components for Constant Turn Rate Motion

To generate expressions for the position components, $\mathbf{r}(t)$, we know that

$$\mathbf{r}(t) = \mathbf{r}(t_0) + \int_{t_0}^t \mathbf{v}(u) du \quad (18.130)$$

First consider the integrals of the form

$$\begin{aligned}
 \int_{t_0}^t c_i du &= \int_{t_0}^t \cos(\xi_i(u - t_0)) du \\
 &= \int_{t_0}^t [\cos(\xi_i u) \cos(-\xi_i t_0) - \sin(\xi_i u) \sin(-\xi_i t_0)] du \\
 &= \cos(-\xi_i t_0) \int_{t_0}^t \cos(\xi_i u) du - \sin(-\xi_i t_0) \int_{t_0}^t \sin(\xi_i u) du \\
 &= \frac{1}{\xi_i} \sin(\xi_i t) \\
 &= \frac{1}{\xi_i} s_i
 \end{aligned} \tag{18.131}$$

Similarly

$$\begin{aligned}
 \int_{t_0}^t s_i du &= \int_{t_0}^t \sin(\xi_i(u - t_0)) du \\
 &= \frac{1}{\xi_i} (1 - \cos(\xi_i t)) \\
 &= \frac{1}{\xi_i} (1 - c_i)
 \end{aligned} \tag{18.132}$$

Define

$$g_i \triangleq \frac{1}{\xi_i} s_i \tag{18.133}$$

$$h_i \triangleq \frac{1}{\xi_i} (1 - c_i) \tag{18.134}$$

It follows immediately that, after performing the integrations, the components of (18.130) are given by

$$\begin{aligned}
 r_x(t) &= r_x(t_0) + \frac{1}{2} \{(g_1 + g_2)v_x(t_0) + (h_1 + h_2)v_y(t_0) \\
 &\quad + [(h_1 - h_2)\sin\vartheta_0 - (g_1 - g_2)\cos\vartheta_0]v_z(t_0)\}
 \end{aligned} \tag{18.135}$$

$$\begin{aligned}
 r_y(t) &= r_y(t_0) + \frac{1}{2} \{-(h_1 + h_2)v_x(t_0) + (g_1 + g_2)v_y(t_0) \\
 &\quad + [(g_1 - g_2)\sin\vartheta_0 + (h_1 - h_2)\cos\vartheta_0]v_z(t_0)\}
 \end{aligned} \tag{18.136}$$

$$r_z(t) = r_z(t_0) + \frac{1 - \cos\Psi T}{\Psi}v_h(t_0) + \frac{\sin\Psi T}{\Psi}v_z(t_0) \tag{18.137}$$

Now let us consider the special cases:

1. Consider the case where $\Omega = \Psi = 0$. Now $\mathbf{v}(t) = \mathbf{v}(t_0)$, so $\mathbf{r}(t) = \mathbf{r}(t_0) + T\mathbf{v}(t_0)$.

2. When $\Omega = 0, \Psi \neq 0$, leads to $g_1 = g_2 = \sin(\Psi T) / \Psi$, and $h_1 = -(1 - \cos(\Psi T)) / \Psi, h_2 = -h_1$.
3. When $\Psi = 0, \Omega \neq 0$, leads to $g_1 = g_2 = \sin(\Omega T) / \Omega$, and $h_1 = h_2 = -(1 - \cos(\Omega T)) / \Omega$.
4. If $\Omega = \Psi \neq 0, \xi_1 = 0$ and $g_1 = T$ and $h_1 = 0$.
5. If $\Omega = -\Psi \neq 0, \xi_2 = 0$ and $g_2 = 1$ and $h_2 = 0$.

18.A.3 Combined Trajectory Transition Equation

Combining (18.135) through (18.125) with (18.128), (18.129) and (18.125) into a single vector-matrix equation leads to

$$\mathbf{x}(t_n) = \mathbf{f}_n(\mathbf{x}(t_0)) \quad (18.138)$$

and

$$\mathbf{x}(t_n) \triangleq \mathbf{x}_n = [r_{x,n}, r_{y,n}, r_{z,n}, v_{x,n}, v_{y,n}, v_{z,n}]^\top \quad (18.139)$$

By setting way-points to divide a trajectory into segments and specifying the turn and dive rates for each segment, (18.138) can be used to create realistic trajectories. For all simulations used in this case study, trajectories were created at 0.1 s intervals or a sample rate of 10 Hz.

18.A.4 Turn Rate Setting Based on a Desired Turn Acceleration

For an object with a constant speed v , the velocity components as a function of heading and aspect are given by (18.113). The acceleration can then be depicted as

$$\begin{aligned} \mathbf{a}(t) &= \frac{d\mathbf{v}}{dt} = \begin{bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{bmatrix} \\ &= v \begin{bmatrix} \dot{\vartheta} \cos \vartheta(t) \cos \gamma(t) - \dot{\gamma} \sin \vartheta(t) \sin \gamma(t) \\ -\dot{\vartheta} \sin \vartheta \cos \gamma(t) - \dot{\gamma} \cos \vartheta(t) \sin \gamma(t) \\ \dot{\gamma} \cos \gamma(t) \end{bmatrix} \end{aligned} \quad (18.140)$$

The magnitude of the acceleration is now given by

$$|\mathbf{a}(t)| = \sqrt{a_x^2(t) + a_y^2(t) + a_z^2(t)} \quad (18.141)$$

with

$$\begin{aligned} a_x^2(t) &= v^2 \left[\dot{\vartheta}^2 \cos^2 \vartheta(t) \cos^2 \gamma(t) + \dot{\gamma}^2 \sin^2 \vartheta(t) \sin^2 \gamma(t) \right. \\ &\quad \left. - 2\dot{\vartheta}\dot{\gamma} \cos \vartheta(t) \sin \vartheta(t) \cos \gamma(t) \sin \gamma(t) \right] \end{aligned} \quad (18.142)$$

$$\begin{aligned} a_y^2(t) &= v^2 \left[\dot{\vartheta}^2 \sin^2 \vartheta(t) \cos^2 \gamma(t) + \dot{\gamma}^2 \cos^2 \vartheta(t) \sin^2 \gamma(t) \right. \\ &\quad \left. + 2\dot{\vartheta}\dot{\gamma} \cos \vartheta(t) \sin \vartheta(t) \cos \gamma(t) \sin \gamma(t) \right] \end{aligned} \quad (18.143)$$

$$a_z^2(t) = v^2 \dot{\gamma}^2 \cos^2 \gamma(t) \quad (18.144)$$

Thus

$$|\mathbf{a}(t)| = v \sqrt{\dot{\vartheta}^2 \cos \gamma(t) + \dot{\gamma}^2} \quad (18.145)$$

Identifying $\dot{\vartheta} = \Omega$ and $\dot{\gamma} = \Psi$, this becomes

$$|\mathbf{a}(t)| = v \sqrt{\Omega^2 \cos \gamma(t) + \Psi^2} \quad (18.146)$$

Consider the following special cases for constant rate turns/dives:

1. For a horizontal turn, $\gamma(t) \rightarrow \gamma_0$; $\Psi \rightarrow 0$, and

$$|\mathbf{a}(t)| = \Omega v \cos \gamma_0 \quad (18.147)$$

2. For a vertical dive: $\Omega \rightarrow 0$, and

$$|\mathbf{a}(t)| = \Psi v \quad (18.148)$$

Suppose now that we wish to specify a horizontal turn with an acceleration defined as a number of g 's (acceleration due to gravity), say x g 's. For a horizontal turn with an initial attitude $\gamma_0 = 0$, this leads to

$$\Omega = \frac{x}{v} \quad (18.149)$$

For example, if we specify a 6 g horizontal turn to the right for an object moving at a constant speed of 600 knots, the turn rate should be set to

$$\Omega = \frac{6g}{600 \text{ knots}} \times C \quad (18.150)$$

where the conversion factor C is defined by

$$C \triangleq \frac{[32.174 \text{ (ft/s}^2\text{)/}g]}{(6076.11549 \text{ (nmi/ft)} / 3600 \text{ (s/h)})} \quad (18.151)$$

Thus, for this case, we find that we must set $\Omega = 0.19$ rad/s to achieve the desired turn acceleration.

APPENDIX 18.B THREE-DIMENSIONAL COORDINATE TRANSFORMATIONS

When using a tracking filter that defines the state vector in Cartesian coordinates and the observation vector in spherical polar coordinates, both Cartesian-to-spherical and a spherical-to-Cartesian transformations are required. These transformations must be applied to the vector under transformation as well as the covariance matrix. In this section, we will derive these transformations and show how to develop a subroutine to achieve these transformation. As a side benefit, the Jacobian of the transformation is created and becomes part of the output.

First, let us present some general concepts about multidimensional coordinate transformations. A vector transformation from one coordinate system to another can be written as

$$\mathbf{r} = \mathbf{f}(\mathbf{x}) \quad (18.152)$$

where \mathbf{x} is the vector in its original coordinate system and \mathbf{r} is the vector in the transformed system.

To develop a transformation of the covariance matrix, examine the definition of the covariance matrix in the transformed system, given by

$$\mathbf{P}^{\mathbf{r}\mathbf{r}} = \mathcal{E}\{(\mathbf{r} - \hat{\mathbf{r}})(\mathbf{r} - \hat{\mathbf{r}})^T\} \quad (18.153)$$

where \mathcal{E} represents the expected value and $\hat{\mathbf{r}} = \mathcal{E}(\mathbf{r})$. From (18.152) it follows immediately that

$$\hat{\mathbf{r}} = \mathbf{f}(\hat{\mathbf{x}}) \quad (18.154)$$

Expanding $\mathbf{f}(\mathbf{x})$ in a Taylor series about $\hat{\mathbf{x}}$ to first order, we find from (2.70),

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= [\mathbf{f}(\mathbf{x})]_{\mathbf{x}=\hat{\mathbf{x}}} + [\nabla_{\mathbf{x}} \mathbf{f}^T(\mathbf{x})]_{\mathbf{x}=\hat{\mathbf{x}}}^T (\mathbf{x} - \hat{\mathbf{x}}) \\ &= \mathbf{f}(\hat{\mathbf{x}}) + \hat{\mathbf{F}}(\mathbf{x} - \hat{\mathbf{x}}) \end{aligned} \quad (18.155)$$

where we have defined the Jacobian $\hat{\mathbf{F}}$ as

$$\begin{aligned} \hat{\mathbf{F}} &= [\nabla_{\mathbf{x}} \mathbf{f}^T(\mathbf{x})]_{\mathbf{x}=\hat{\mathbf{x}}}^T \\ &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}_{\mathbf{x}=\hat{\mathbf{x}}} \end{aligned} \quad (18.156)$$

Here, n is the dimension of the vector \mathbf{x} and m is the dimension of the vector \mathbf{f} .

Putting (18.155) into (18.153) we obtain

$$\begin{aligned}
 \mathbf{P}^{\text{rr}} &= \mathcal{E}\left\{\left[\mathbf{f}(\mathbf{x}) - \mathbf{f}(\hat{\mathbf{x}})\right] \left[\mathbf{f}(\mathbf{x}) - \mathbf{f}(\hat{\mathbf{x}})\right]^T\right\} \\
 &= \mathcal{E}\left\{\left[\mathbf{f}(\hat{\mathbf{x}}) + \hat{\mathbf{F}}(\mathbf{x} - \hat{\mathbf{x}}) - \mathbf{f}(\hat{\mathbf{x}})\right] \left[\mathbf{f}(\hat{\mathbf{x}}) + \hat{\mathbf{F}}(\mathbf{x} - \hat{\mathbf{x}}) - \mathbf{f}(\hat{\mathbf{x}})\right]^T\right\} \\
 &= \hat{\mathbf{F}}\mathcal{E}\left\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T\right\}\hat{\mathbf{F}}^T \\
 &= \hat{\mathbf{F}}\mathbf{P}^{\text{xx}}\hat{\mathbf{F}}^T
 \end{aligned} \tag{18.157}$$

Thus, the general equation for the coordinate transformation of a covariance matrix is given by (18.157), to first order. Note that (18.157) includes the computation of the Jacobian, so any subroutine designed to perform this coordinate transformation produces the Jacobian as a bonus.

18.B.1 Cartesian-to-Spherical Transformation

For the general three-dimensional coordinate systems shown in Figure 18.25, we can write a state vector as

$$\mathbf{x} = [r_x, v_x, r_y, v_y, r_z, v_z]^T \tag{18.158}$$

in Cartesian coordinates, or

$$\mathbf{r} = [R, \dot{R}, \theta, \dot{\theta}, \phi, \dot{\phi}]^T \tag{18.159}$$

in spherical polar coordinates.

From Figure 18.25, it follows that the transformation from Cartesian to spherical for the position coordinates is given by

$$\begin{bmatrix} R \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{r_x^2 + r_y^2 + r_z^2} \\ \tan^{-1}\left(\frac{r_x}{r_y}\right) \\ \tan^{-1}\left(\frac{r_z}{\sqrt{r_x^2 + r_y^2}}\right) \end{bmatrix} \tag{18.160}$$

Taking the derivative of each component with respect to time results in the spherical rate components

$$\begin{bmatrix} \dot{R} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{r_x v_x + r_y v_y + r_z v_z}{R} \\ \frac{r_y v_x - r_x v_y}{R_h^2} \\ \frac{R v_z - r_z \dot{R}}{R R_h} \end{bmatrix} \tag{18.161}$$

where the horizontal range is defined by $R_h \triangleq \sqrt{r_x^2 + r_y^2}$. In a three-dimensional system, R is usually referred to as the slant range. Thus, the complete Cartesian-to-

spherical vector transformation, defined as $\mathbf{h}(\mathbf{x})$, is given by

$$\mathbf{r} = \begin{bmatrix} R \\ \dot{R} \\ \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} = \mathbf{h}(\mathbf{x}) \triangleq \begin{bmatrix} h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ h_3(\mathbf{x}) \\ h_4(\mathbf{x}) \\ h_5(\mathbf{x}) \\ h_6(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \sqrt{r_x^2 + r_y^2 + r_z^2} \\ \frac{r_x v_x + r_y v_y + r_z v_z}{R} \\ \tan^{-1}\left(\frac{r_x}{r_y}\right) \\ \frac{r_y v_x - r_x v_y}{R^2 H} \\ \tan^{-1}\left(\frac{r_z}{\sqrt{r_x^2 + r_y^2}}\right) \\ \frac{R v_z - r_z \dot{R}}{R R_h} \end{bmatrix} \quad (18.162)$$

From (18.156) we can write the Jacobian of the Cartesian-to-spherical covariance transformation as

$$\hat{\mathbf{H}} = \begin{bmatrix} H_{11} & H_{12} & \cdots & H_{16} \\ H_{21} & H_{22} & \cdots & H_{26} \\ \vdots & \vdots & \ddots & \vdots \\ H_{61} & H_{62} & \cdots & H_{66} \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1}{\partial r_x} & \frac{\partial h_1}{\partial v_x} & \cdots & \frac{\partial h_1}{\partial v_z} \\ \frac{\partial h_2}{\partial r_x} & \frac{\partial h_2}{\partial v_x} & \cdots & \frac{\partial h_2}{\partial v_z} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_6}{\partial r_x} & \frac{\partial h_6}{\partial v_x} & \cdots & \frac{\partial h_6}{\partial v_z} \end{bmatrix}_{\mathbf{x}=\hat{\mathbf{x}}} \quad (18.163)$$

If (18.162) is evaluated first, we can use the spherical components in the evaluation of each partial, resulting in the Jacobian components

$$H_{11} = \frac{\partial}{\partial r_x} \left[(r_x^2 + r_y^2 + r_z^2)^{1/2} \right] = \frac{r_x}{R} \quad (18.164)$$

$$H_{12} = H_{14} = H_{16} = 0 \quad (18.165)$$

$$H_{13} = \frac{r_y}{R} \quad (18.166)$$

$$H_{15} = \frac{r_z}{R} \quad (18.167)$$

$$H_{21} = \frac{R v_x - r_x \dot{R}}{R^2} \quad (18.168)$$

$$H_{22} = H_{11} \quad (18.169)$$

$$H_{23} = \frac{R v_y - r_y \dot{R}}{R^2} \quad (18.170)$$

$$H_{24} = H_{13} \quad (18.171)$$

$$H_{25} = \frac{R v_z - r_z \dot{R}}{R^2} \quad (18.172)$$

$$H_{26} = H_{15} \quad (18.173)$$

$$H_{31} = \frac{r_y}{R_h^2} \quad (18.174)$$

$$H_{32} = H_{34} = H_{35} = H_{36} = 0 \quad (18.175)$$

$$H_{33} = -\frac{r_x}{R_h^2} \quad (18.176)$$

$$H_{41} = -\frac{v_y - 2r_x\dot{\theta}}{R_h^2} \quad (18.177)$$

$$H_{42} = H_{31} \quad (18.178)$$

$$H_{43} = \frac{v_x - 2r_y\dot{\theta}}{R_h^2} \quad (18.179)$$

$$H_{44} = H_{33} \quad (18.180)$$

$$H_{45} = H_{46} = 0 \quad (18.181)$$

$$H_{51} = -\frac{r_x r_z}{R^2 R_h} \quad (18.182)$$

$$H_{52} = H_{54} = H_{56} = 0 \quad (18.183)$$

$$H_{53} = -\frac{r_y r_z}{R^2 R_h} \quad (18.184)$$

$$H_{55} = \frac{R_h}{R^2} \quad (18.185)$$

$$H_{61} = \frac{1}{RR_h} \left[\frac{r_x}{R_h^2} (r_z \dot{R} - R v_z) + \frac{r_z}{R^2} (2r_x \dot{R} - R v_x) \right] \quad (18.186)$$

$$H_{62} = H_{51} \quad (18.187)$$

$$H_{63} = \frac{1}{RR_h} \left[\frac{r_y}{R_h^2} (r_z \dot{R} - R v_z) + \frac{r_z}{R^2} (2r_y \dot{R} - R v_y) \right] \quad (18.188)$$

$$H_{64} = H_{53} \quad (18.189)$$

$$H_{65} = \frac{1}{R^3 R_h} \left[(2r_z^2 + R^2) \dot{R} - r_z R v_z \right] \quad (18.189)$$

$$H_{66} = H_{55} \quad (18.190)$$

Now, using (18.157) we can write the covariance matrix transformation from Cartesian-to-spherical coordinates as

$$\mathbf{P}^{rr} = \hat{\mathbf{H}} \mathbf{P}^{xx} \hat{\mathbf{H}}^\top \quad (18.191)$$

where \mathbf{P}^{xx} and \mathbf{P}^{rr} are the covariance matrices in Cartesian and spherical coordinates, respectively.

18.B.2 Spherical-to-Cartesian Transformation

Occasionally, we will need the inverse of (18.162), which is defined as the spherical-to-Cartesian transformation

$$\mathbf{x} = \mathbf{g}(\mathbf{r}) \quad (18.192)$$

If we first examine the position components, using Figure 18.25 we find that

$$\begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} R \sin \theta \cos \phi \\ R \cos \theta \cos \phi \\ R \sin \phi \end{bmatrix} \quad (18.193)$$

taking the derivative of each component results in

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \sin \theta \cos \phi & R \cos \theta \cos \phi & -R \sin \theta \sin \phi \\ \cos \theta \cos \phi & -R \sin \theta \cos \phi & -R \cos \theta \sin \phi \\ \sin \phi & 0 & R \cos \phi \end{bmatrix} \begin{bmatrix} \dot{R} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} \quad (18.194)$$

Combining (18.193) and (18.194) results in the spherical-to-Cartesian transformation

$$\mathbf{x} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} g_1(\mathbf{r}) \\ g_2(\mathbf{r}) \\ g_3(\mathbf{r}) \\ g_4(\mathbf{r}) \\ g_5(\mathbf{r}) \\ g_6(\mathbf{r}) \end{bmatrix} = \begin{bmatrix} R \sin \theta \cos \phi \\ \dot{R} \sin \theta \cos \phi + \dot{\theta} R \cos \theta \cos \phi - \dot{\phi} R \sin \theta \sin \phi \\ R \cos \theta \cos \phi \\ \dot{R} \cos \theta \cos \phi - \dot{\theta} R \sin \theta \cos \phi - \dot{\phi} R \cos \theta \sin \phi \\ R \sin \phi \\ \dot{R} \sin \phi + \dot{\phi} R \cos \phi \end{bmatrix} \quad (18.195)$$

It follows immediately that the spherical-to-Cartesian transformation of the covariance matrix is given by

$$\mathbf{P}^{\mathbf{xx}} = \hat{\mathbf{G}} \mathbf{P}^{\mathbf{rr}} \hat{\mathbf{G}}^\top \quad (18.196)$$

where the components of the transformation Jacobian from (18.156) are given by

$$G_{11} = \sin \theta \cos \phi \quad (18.197)$$

$$G_{12} = G_{14} = G_{16} = 0 \quad (18.198)$$

$$G_{13} = R \cos \theta \cos \phi \quad (18.199)$$

$$G_{15} = -R \sin \theta \sin \phi \quad (18.200)$$

$$G_{21} = \dot{\theta} \cos \theta \cos \phi - \dot{\phi} \sin \theta \sin \phi \quad (18.201)$$

$$G_{22} = G_{11} \quad (18.202)$$

$$G_{23} = \dot{R} \cos \theta \cos \phi - \dot{\theta} R \sin \theta \cos \phi - \dot{\phi} R \cos \theta \sin \phi \quad (18.203)$$

$$G_{24} = G_{13} \quad (18.204)$$

$$G_{25} = -\dot{R} \sin \theta \sin \phi - \dot{\theta} R \cos \theta \sin \phi - \dot{\phi} R \sin \theta \cos \phi \quad (18.205)$$

$$G_{26} = G_{15} \quad (18.206)$$

$$G_{31} = \cos \theta \cos \phi \quad (18.207)$$

$$G_{32} = G_{34} = G_{36} = 0 \quad (18.208)$$

$$G_{33} = -R \sin \theta \cos \phi \quad (18.209)$$

$$G_{35} = -R \cos \theta \sin \phi \quad (18.210)$$

$$G_{41} = -\dot{\theta} \sin \theta \cos \phi - \dot{\phi} \cos \theta \sin \phi \quad (18.211)$$

$$G_{42} = G_{31} \quad (18.212)$$

$$G_{43} = -\dot{R} \sin \theta \cos \phi - \dot{\theta} R \cos \theta \cos \phi + \dot{\phi} R \sin \theta \sin \phi \quad (18.213)$$

$$G_{44} = G_{33} \quad (18.214)$$

$$G_{45} = -\dot{R} \cos \theta \sin \phi + \dot{\theta} R \sin \theta \sin \phi - \dot{\phi} R \cos \theta \cos \phi \quad (18.215)$$

$$G_{46} = G_{35} \quad (18.216)$$

$$G_{51} = \sin \phi \quad (18.217)$$

$$G_{52} = G_{53} = G_{54} = G_{56} = 0 \quad (18.218)$$

$$G_{55} = R \cos \phi \quad (18.219)$$

$$G_{61} = \dot{\phi} \cos \phi \quad (18.220)$$

$$G_{62} = G_{51} \quad (18.221)$$

$$G_{63} = G_{64} = 0 \quad (18.222)$$

$$G_{65} = \dot{R} \cos \phi - \dot{\phi} R \sin \phi \quad (18.223)$$

$$G_{66} = G_{55} \quad (18.224)$$

REFERENCES

1. Blackman SS. *Multiple-Target Tracking with Radar Applications*. Artech House; 1986.
2. Bar-Shalom Y, Fortman TE. *Tracking and Data Association*. Academic Press; 1988.
3. Bar-Shalom Y, Rong Li X, Kirubarajan T. *Estimation with Application to Tracking and Navigation*. Wiley; 2001.
4. Ristic B, Arulampalam S, Gordon N. *Beyond the Kalman filter: Particle Filter Applications*. Artech House; 2004.

5. Zollo S, Ristic B. On the Choice of the Coordinate System and Tracking Filter for the Track-while-scan Mode of an Airborne Radar. DSTO-TR-0926; 1999.
6. Lefferts RE. Alpha-beta filters in polar coordinates with acceleration constraints. *IEEE Trans. Aero. Elec. Syst.* 1988; 24(6): 693–699.
7. Blair WD, Watson GA, Rice TR. Interacting multiple model filter for tracking maneuvering targets in spherical coordinates. *Proc. Am. Control Conf.* 1991;1055–1059.
8. Kameda H, Nomoto K, Kosuge Y, Kondo M. Target tracking algorithm in radar reference coordinates using fully-coupled filters. *Electron. Commun. Jpn.* 1997;Pt1,10(7):327–333.
9. Swilling DZ. *CRC Standard Mathematical Tables and Formulae*, 30th ed. CRC Press; 1996.

19

TRACKING A FALLING RIGID BODY USING PHOTOGRAMMETRY

19.1 INTRODUCTION

Photogrammetry is the mathematical science of estimating an object's pose (three-dimensional position and orientation) and kinematic properties (translational and rotational velocities and accelerations) from a set of object features observed in a series of two-dimensional video image frame data from multiple cameras. Many approaches to this estimation problem have been applied, with the most common prior to the 1990s using a nonlinear least squares (NLLSQ) approach. These early NLLSQ methods were reviewed by Huang and Netravali [1]. In the NLLSQ method, at each time step, the residuals between the predicted video images and the actual current video images are used in an iterative NLLSQ procedure to arrive at a local residual error minimum producing an estimate of the rigid body state. Alternate, more modern, Bayesian estimation extended Kalman filter (EKF) methods were introduced by Iu and Wohn [2] and Gennery [3] with excellent results. These were followed by a single-constraint-at-a-time (SCAAT) EKF method introduced by Welch [4,5].

All of the above-mentioned methods had some drawbacks that affect the rigid body state estimation performance. The NLLSQ methods assumes a dynamic model without noise, is not recursive in time, and requires the computation of a Jacobian (matrix of partial derivatives) at each iteration. The observation set consists of video frame images and all image frames are reprocessed during each iteration, making the method very time consuming. In addition, because the NLLSQ, as used in photogrammetry, is a "single time-step at a time" method, translational velocities and

angular rates are not estimated in the NLLSQ but have to be calculated separately using a postprocessing smoothing step.

The EKF methods are also difficult to implement because the Jacobians of both the nonlinear observation and dynamic transition models have to be calculated at each time step in order to linearize the observation process (see Chapter 7). This analytical linearization has proven to be inadequate for many highly nonlinear problems.

In this case study, for the purpose of safety evaluation, we address the specific problem of estimating and tracking the time-varying pose of a rigid body store (a bomb-like object) dropped from an airplane pylon. We use image sequences that are recorded from multiple high-speed image sensors attached to a reference object. The image sensors or cameras are specifically oriented to reliably observe the track object's rigid body state trajectory. Multiple feature points are affixed to both the reference and rigid body object as shown in Figures 19.1 and 19.2. There are 19 cameras attached to the aircraft, including four cameras on each wingtip, four under the tail, and a camera under the aircraft's nose. The position of the feature points in the image frame of each camera are recorded in a temporal image sequence that is used in the estimation of the trajectory of the tracked object. Each image sensor has its own unique image sequence as a record of the object's motion from a particular point of view. Each image sequence must be properly combined with the image sequences from other cameras to compute the best track estimate of the trajectory at each time step. The track object's trajectory can then be used to perform miss distance analysis for safety purposes, or for validation and refinement of predictions made by other types of modeling and simulation. Since image sensors estimate the time-varying position



FIGURE 19.1 Photo of a US Navy F-18E super hornet aircraft preparing to release four Mk-62 stores.



FIGURE 19.2 Close-up of a MK-62 store release as viewed from a camera mounted under the aircraft's tail.

and orientation of a rigid body object in the 3D world, photogrammetric estimation is a logical first choice for accurate close range tracking [6].

In our estimation process we introduce two new methods that use an unscented Kalman filter (UKF) or a generalized UKF particle filter (see Chapters 9 and 17, respectively, for a complete derivation and discussion of these filters). These filters offer improved performance over both the EKF and the NLLSQ because they have the capability of estimating the full rigid body state kinematics including Cartesian and angular velocities and accelerations without the need to compute Jacobians.

In Section 19.2, we present a dynamic model of motion for a falling rigid body object including translational and rotational motion. This section also contains a discussion of the dynamic noise terms that are introduced to account for any small motion deviations from the dynamic model. The observation model introduced in Section 19.3 begins by calculating the location of the feature points on the rigid body surface, transforming (rotating and translating) these 3D feature points on the rigid body from the body's coordinate system into the reference (aircraft's) coordinates, then transforming the 3D points into a camera's coordinate system and projecting the 3D points into the camera's 2D image plane.

This is followed in Section 19.4 with a discussion of the estimation methods used, beginning with the NLLSQ in Section 19.4.1, the UKF in Section 19.4.2, and the UKF combination particle filter in Section 19.4.3. Initialization for all of these filters is presented in Section 19.4.5.

For performance analysis purposes, we have developed a synthetic rigid body model along with a synthetic motion model that emulates the motion of a rigid body store as it is released from an airplane pylon. This model and a method for using it to generate synthetic image frame data is contained in Section 19.5. The relative

performance of all of the estimation methods along with a summary of important results is presented in Section 19.6.

Appendix 19.A outlines some definitions and the mathematics associated with quaternions, axis-angle vectors and rotations, and is included as a brief review of the topics.

19.2 THE PROCESS (DYNAMIC) MODEL FOR RIGID BODY MOTION

Our motion model will describe the translational and rotational motion of a rigid body as a function of time. Consider an instantaneous state vector $\mathbf{x}(t) = [\mathbf{p}^\top(t), \mathbf{a}^\top(t)]^\top$ that defines both the translation of the center of mass of a rigid body relative to a fixed reference Cartesian coordinate frame, $\mathbf{p}(t) = [x(t), y(t), z(t)]^\top$, and the orientation of the body with respect to the reference frame given as an axis-angle vector $\mathbf{a}(t) = [a_x(t), a_y(t), a_z(t)]^\top$. Under certain conditions the vector \mathbf{a} can become numerically unstable (specifically when $\|\mathbf{a}\| = 2\pi n$ where n is a positive integer), so instead we use a unit quaternion $\mathbf{q} = [q_s, q_x, q_y, q_z]^\top$ that when combined with \mathbf{a} represents the orientation. We discuss the need for \mathbf{q} in more detail in Section 19.2.2 and in Appendix 19.A.2. In addition, in the dynamic (process) models developed below, we will modify the state vector to include rate vectors, acceleration vectors, and jerk (third time derivative) terms governing both translational and rotational motion.

Since the translational motion of a rigid body is independent of its rotational motion, we will treat the two separately. In the sections below we will develop the dynamic transition equations that take the rigid body state vector, defined above, from some time t_{n-1} to a later time t_n . First, in Section 19.2.1 we address the transition equations governing the translational part of the state vector, while in Section 19.2.2 we consider the transition equations for the orientation part of the state vector. In general, we can write the total transition equation in the form

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}, \mathbf{v}_{n-1}) \quad (19.1)$$

where \mathbf{v}_{n-1} is considered the noise driving the process. In all cases we will take the noise to be Gaussian and additive so that (19.1) becomes

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{v}_{n-1} \quad (19.2)$$

In Section 19.2.3, this composite pose transition model is discussed in more detail. In Section 19.2.4, we will discuss the properties of several noise models for \mathbf{v}_{n-1} .

19.2.1 Dynamic Transition of the Translational Motion of a Rigid Body

Let \mathbf{p}_n be a three-dimensional vector that represents the Cartesian position of the center of gravity (cg) of a rigid body at time t_n , that is, $\mathbf{p}_n = [x_n, y_n, z_n]^\top$. Expanding

$\mathbf{p}(t_n)$ in a temporal Taylor series about some prior time t_{n-1} we can write

$$\begin{aligned}\mathbf{p}_n &= \mathbf{p}_{n-1} + \left[\frac{d\mathbf{p}(t)}{dt} \right]_{t=t_{n-1}} (t_n - t_{n-1}) + \left[\frac{d^2\mathbf{p}(t)}{dt^2} \right]_{t=t_{n-1}} \frac{(t_n - t_{n-1})^2}{2!} \\ &\quad + \left[\frac{d^3\mathbf{p}(t)}{dt^3} \right]_{t=t_{n-1}} \frac{(t_n - t_{n-1})^3}{3!} + \dots\end{aligned}\quad (19.3)$$

$$= \mathbf{p}_{n-1} + T_n \dot{\mathbf{p}}_{n-1} + \frac{T_n^2}{2} \ddot{\mathbf{p}}_{n-1} + \frac{T_n^3}{6} \dddot{\mathbf{p}}_{n-1} + \dots\quad (19.4)$$

where $\dot{\mathbf{p}}$, $\ddot{\mathbf{p}}$, $\dddot{\mathbf{p}}$ represent the first three temporal derivatives of position and $T_n \triangleq t_n - t_{n-1}$. For this case study, all observing cameras are assumed to record their images in a synchronous fashion, so we will assume that T_n is a constant T . Keeping only terms to third order, we can rewrite (19.3) as the third-order Taylor polynomial

$$\mathbf{p}_n = \mathbf{p}_{n-1} + T \dot{\mathbf{p}}_{n-1} + \frac{T^2}{2} \ddot{\mathbf{p}}_{n-1} + \frac{T^3}{6} \dddot{\mathbf{p}}_{n-1}\quad (19.5)$$

For a *constant position* model, $\dot{\mathbf{p}}$, $\ddot{\mathbf{p}}$, and $\dddot{\mathbf{p}}$ are all zero so we define the rigid-body zeroth-order three-dimensional state vector containing only position components as $\mathbf{p}_n^{(0)} = \mathbf{p}_n = [x(t), y(t), z(t)]^\top$ and (19.5) becomes

$$\mathbf{p}_n^{(0)} = \mathbf{p}_{n-1}^{(0)} + \mathbf{v}_{n-1}^{(0)}\quad (19.6)$$

where we have added a zeroth-order process velocity noise term $\mathbf{v}_{n-1}^{(0)}$. Process noise is added to account for slight deviations of the objects position from our model due to unmodeled effects such as wind, friction, and so on.

For a *constant velocity* assumption, if we define a six-dimensional state vector having both position and velocity components, that is $\mathbf{p}_n^{(1)} = [\mathbf{p}_n^\top, \dot{\mathbf{p}}_n^\top]^\top$, (19.5) becomes the first-order process model

$$\mathbf{p}_n^{(1)} = \mathbf{F}^{(1)} \mathbf{p}_{n-1}^{(1)} + \mathbf{v}_{n-1}^{(1)}\quad (19.7)$$

where

$$\mathbf{F}^{(1)} = \begin{bmatrix} \mathbf{I}_3 & T\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix}\quad (19.8)$$

with \mathbf{I}_3 and $\mathbf{0}_3$ defined as the three-dimensional identity and zero matrices, respectively. $\mathbf{v}_{n-1}^{(1)}$ is the first-order acceleration process noise.

In a similar manner, for a *constant acceleration* model (19.5) the nine-dimensional state vector is $\mathbf{p}_n^{(2)} = [\mathbf{p}_n^\top, \dot{\mathbf{p}}_n^\top, \ddot{\mathbf{p}}_n^\top]^\top$ and the process model becomes

$$\mathbf{p}_n^{(2)} = \mathbf{F}^{(2)} \mathbf{p}_{n-1}^{(2)} + \mathbf{v}_{n-1}^{(2)}\quad (19.9)$$

with

$$\mathbf{F}^{(2)} = \begin{bmatrix} \mathbf{I}_3 & T\mathbf{I}_3 & \frac{T^2}{2}\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & T\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \quad (19.10)$$

$\mathbf{v}_{n-1}^{(2)}$ is the second-order jerk process noise.

And finally, for a *constant jerk* 12-dimensional model, $\mathbf{p}_n^{(3)} = [\mathbf{p}_n^\top, \dot{\mathbf{p}}_n^\top, \ddot{\mathbf{p}}_n^\top, \dddot{\mathbf{p}}_n^\top]^\top$ resulting in the process model

$$\mathbf{p}_n^{(3)} = \mathbf{F}^{(3)}\mathbf{p}_{n-1}^{(3)} + \mathbf{v}_{n-1}^{(3)} \quad (19.11)$$

with

$$\mathbf{F}^{(3)} = \begin{bmatrix} \mathbf{I}_3 & T\mathbf{I}_3 & \frac{T^2}{2}\mathbf{I}_3 & \frac{T^3}{6}\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & T\mathbf{I}_3 & \frac{T^2}{2}\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & T\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \quad (19.12)$$

and $\mathbf{v}_{n-1}^{(1)}$ is the third-order snap (or jounce) fourth derivative process noise.

19.2.2 Dynamic Transition of the Rotational Motion of a Rigid Body

The instantaneous orientation of a rigid body can be expressed in multiple ways. The orientation can be expressed as a set of *Euler rotation angles* relative to the three Cartesian reference frame axis. The right-handed Cartesian reference frame we choose is a frame with the x -axis pointing forward, the y -axis pointing to the right (starboard) looking forward, and the z -axis pointing down. The three orientation angles are $\{\phi, \theta, \psi\}$, with ϕ (roll) representing a counterclockwise (CCW) rotation about the x -axis, θ (pitch) representing a CCW rotation about the y -axis, and lastly φ (yaw) a CCW rotation about the z -axis, respectively. Unfortunately, using Euler angles to describe rigid body orientation can result in singularities under certain circumstances [3].

19.2.2.1 Conversions Between Rotation Representations. Let $\mathbf{a} = [a_x, a_y, a_z]^\top \in \mathbb{R}^3$ be used as an *axis-angle representation* of rotation where the direction of \mathbf{a} specifies the axis of rotation and $\|\mathbf{a}\| = \sqrt{a_x^2 + a_y^2 + a_z^2}$ specifies the magnitude of rotation (in radians).

Let $\mathbf{q} = [q_s, q_x, q_y, q_z]^\top \in \mathbf{H}$, where \mathbf{H} is the space of quaternions, be used as a *unit quaternion representation* of rotation, where q_s is real, q_x, q_y, q_z are coefficients of distinct imaginary numbers, and $\sqrt{q_s^2 + q_x^2 + q_y^2 + q_z^2} = 1$.

Let $\mathbf{M} \in \mathbb{R}^3 \times \mathbb{R}^3$ be a 3×3 orthonormal *rotation matrix*.

To convert from an axis-angle vector \mathbf{a} to a unit quaternion \mathbf{q} , we define the function $\mathcal{Q}_{\mathbf{a}} : \mathbb{R}^3 \rightarrow \mathbf{H}$ as

$$\mathbf{q} = \mathcal{Q}_{\mathbf{a}}(\mathbf{a}) = \left[\cos \frac{\|\mathbf{a}\|}{2}, \frac{\sin \frac{\|\mathbf{a}\|}{2}}{\|\mathbf{a}\|} [a_x, a_y, a_z]^\top \right]^\top \quad (19.13)$$

To convert from a unit quaternion \mathbf{q} to an axis-angle vector \mathbf{a} we define the function $\mathcal{A}_{\mathbf{q}} : \mathbf{H} \rightarrow \mathbb{R}^3$ as

$$\mathbf{a} = \mathcal{A}_{\mathbf{q}}(\mathbf{q}) = \frac{2 \cos^{-1} q_s}{\sqrt{1 - q_s^2}} [q_x, q_y, q_z]^\top \quad (19.14)$$

19.2.2.2 Quaternion Multiplication. Let \mathbf{q}_1 , \mathbf{q}_2 , and \mathbf{q}_3 be quaternions. Then the quaternion multiplication $\mathbf{q}_3 = \mathbf{q}_1 \mathbf{q}_2$ is defined as

$$\mathbf{q}_3 = \begin{bmatrix} q_{3,s} \\ q_{3,x} \\ q_{3,y} \\ q_{3,z} \end{bmatrix} = \begin{bmatrix} q_{1,s} & -q_{1,x} & -q_{1,y} & -q_{1,z} \\ q_{1,x} & q_{1,s} & -q_{1,z} & q_{1,y} \\ q_{1,y} & q_{1,z} & q_{1,s} & -q_{1,x} \\ q_{1,z} & -q_{1,y} & q_{1,x} & q_{1,s} \end{bmatrix} \begin{bmatrix} q_{2,s} \\ q_{2,x} \\ q_{2,y} \\ q_{2,z} \end{bmatrix} \quad (19.15)$$

19.2.2.3 Dynamic Model for Orientation. To express the dynamic temporal transition of the rigid body's orientation, we will use the axis-angle representation. The axis-angle orientation vector, \mathbf{a}_n , represents a small rotation relative to the prior orientation, that is, the rotation from time t_{n-1} to time t_n . This small axis-angle rotation is added (via a rotation) into an unambiguous *external* quaternion representation, \mathbf{q}_{n-1} , which contains the full rotation at time t_{n-1} for use in the observation model. Note that \mathbf{a} and its derivatives $\dot{\mathbf{a}}$, $\ddot{\mathbf{a}}$, and $\ddot{\mathbf{a}}$ (when present) are all with respect to the reference coordinate frame and not the body frame, thus \mathbf{q} does not need to appear in the dynamic model.

We will define the orientation state vector in terms of the axis-angle vector \mathbf{a} and its derivatives. For a *constant orientation* model, $\dot{\mathbf{a}}$, $\ddot{\mathbf{a}}$, and $\ddot{\mathbf{a}}$ are all zero, so consider the state vector for this zeroth-order model to be the 3D axis-angle vector $\mathbf{a}^{(0)}$ so that

$$\mathbf{a}_n^{(0)} = \mathbf{a}_{n-1}^{(0)} + \boldsymbol{\rho}_{n-1}^{(0)} \quad (19.16)$$

where $\boldsymbol{\rho}_{n-1}^{(0)}$ represents the three-dimensional zeroth-order angular velocity noise in the orientation.

For a *constant rotation-rate* model, we choose a six-dimensional state vector of the form $\mathbf{a}_n^{(1)} \triangleq [\mathbf{a}_n^\top, \dot{\mathbf{a}}_n^\top]^\top$ resulting in the first-order process model

$$\mathbf{a}_n^{(1)} = \mathbf{g}_{n-1}^{(1)} \left(\mathbf{a}_{n-1}^{(1)} \right) + \boldsymbol{\rho}_{n-1}^{(1)} \quad (19.17)$$

where $\boldsymbol{\rho}_{n-1}^{(1)}$ represents the three-dimensional first-order angular velocity noise in the orientation. To propagate the axis-angle vector \mathbf{a} forward over the time interval $T = t_n - t_{n-1}$ we cannot simply add $T \dot{\mathbf{a}}_{n-1}$ to \mathbf{a}_{n-1} to get a new vector \mathbf{a}_n because

adding together (or combining) two 3D rotations cannot be accomplished using vector addition of their axis-angle representations. Instead, we must convert the axis-angle vectors to unit quaternions, multiply them together (while remembering to reverse their order), and convert the results back into an axis-angle vector (alternatively, we could use rotation matrices instead of unit quaternions.) Specifically, let $\Delta \mathbf{a}$ be the incremental change to orientation produced by a constant rotation rate over the time interval T so that

$$\Delta \mathbf{a} = \int_{t_{n-1}}^{t_n} \dot{\mathbf{a}} dt = T\dot{\mathbf{a}} \quad (19.18)$$

Define $\mathbf{q}_{\mathbf{a}_{n-1}}$ and $\mathbf{q}_{\Delta \mathbf{a}}$ as

$$\mathbf{q}_{\mathbf{a}_{n-1}} = Q_{\mathbf{a}}(\mathbf{a}_{n-1}) \quad (19.19)$$

$$\mathbf{q}_{\Delta \mathbf{a}} = Q_{\mathbf{a}}(\Delta \mathbf{a}) = Q_{\mathbf{a}}(T\dot{\mathbf{a}}) \quad (19.20)$$

Now, we can multiply the two quaternions, taking care to reverse their order, and convert the results back into the axis-angle vector \mathbf{a}_n

$$\mathbf{a}_n = \mathcal{A}_{\mathbf{q}}(\mathbf{q}_{\Delta \mathbf{a}} \mathbf{q}_{\mathbf{a}_{n-1}}) \quad (19.21)$$

Since we are assuming constant rotation rate, it follows immediately that

$$\dot{\mathbf{a}}_n = \dot{\mathbf{a}}_{n-1} \quad (19.22)$$

Now, using (19.19) through (19.22), (19.17) becomes

$$\mathbf{a}_n^{(1)} = \begin{bmatrix} \mathbf{a}_n \\ \dot{\mathbf{a}}_n \end{bmatrix} = \begin{bmatrix} \mathcal{A}_{\mathbf{q}}(Q_{\mathbf{a}}(T\dot{\mathbf{a}}_{n-1}) Q_{\mathbf{a}}(\mathbf{a}_{n-1})) \\ \dot{\mathbf{a}}_{n-1} \end{bmatrix} + \boldsymbol{\rho}_{n-1}^{(1)} \quad (19.23)$$

For a *constant rotational-acceleration* model, we choose the nine-dimensional orientational state vector $\mathbf{a}_n^{(2)} \triangleq [\mathbf{a}_n^T, \dot{\mathbf{a}}_n^T, \ddot{\mathbf{a}}_n^T]^T$ resulting in the second-order process model

$$\mathbf{a}_n^{(2)} = \mathbf{g}_{n-1}^{(2)} \left(\mathbf{a}_{n-1}^{(2)} \right) + \boldsymbol{\rho}_{n-1}^{(2)} \quad (19.24)$$

where $\boldsymbol{\rho}_{n-1}^{(2)}$ represents the three-dimensional second-order angular velocity noise in the orientation. For this model, the incremental change to orientation over the time interval T is given by

$$\Delta \mathbf{a} = \int_{t_{n-1}}^{t_n} (\dot{\mathbf{a}} + t\ddot{\mathbf{a}}) dt = T\dot{\mathbf{a}} + \frac{T^2}{2}\ddot{\mathbf{a}} \quad (19.25)$$

Since $\dot{\mathbf{a}}$ and $\ddot{\mathbf{a}}$ are proper vectors, we can write

$$\dot{\mathbf{a}}_n = \dot{\mathbf{a}}_{n-1} + T\ddot{\mathbf{a}}_{n-1} \quad (19.26)$$

$$\ddot{\mathbf{a}}_n = \ddot{\mathbf{a}}_{n-1} \quad (19.27)$$

so that

$$\begin{aligned} \begin{bmatrix} \mathbf{a}_n \\ \dot{\mathbf{a}}_n \\ \ddot{\mathbf{a}}_n \end{bmatrix} &= \mathbf{g}_{n-1}^{(2)} \left(\mathbf{a}_{n-1}^{(2)} \right) + \boldsymbol{\rho}_{n-1}^{(2)} \\ &= \begin{bmatrix} \mathcal{A}_{\mathbf{q}} \left(\mathcal{Q}_{\mathbf{a}} \left(T \dot{\mathbf{a}}_{n-1} + \frac{T^2}{2} \ddot{\mathbf{a}}_{n-1} \right) \mathcal{Q}_{\mathbf{a}} (\mathbf{a}_{n-1}) \right) \\ \dot{\mathbf{a}}_{n-1} + T \ddot{\mathbf{a}}_{n-1} \\ \ddot{\mathbf{a}}_{n-1} \end{bmatrix} \\ &\quad + \boldsymbol{\rho}_{n-1}^{(2)} \end{aligned} \quad (19.28)$$

Finally, for a *constant rotational-jerk* model, we choose a 12-dimensional state vector $\mathbf{a}_n^{(3)} \triangleq [\mathbf{a}_n^T, \dot{\mathbf{a}}_n^T, \ddot{\mathbf{a}}_n^T, \dddot{\mathbf{a}}_n^T]^T$ resulting in the process model

$$\mathbf{a}_n^{(3)} = \mathbf{g}_{n-1}^{(3)} \left(\mathbf{a}_{n-1}^{(3)} \right) + \boldsymbol{\rho}_{n-1}^{(3)} \quad (19.29)$$

where $\boldsymbol{\rho}_{n-1}^{(3)}$ is the 12-dimensional angular noise for this model and

$$\mathbf{g}_{n-1}^{(3)} \left(\mathbf{a}_{n-1}^{(3)} \right) = \begin{bmatrix} \mathcal{A}_{\mathbf{q}} \left(\mathcal{Q}_{\mathbf{a}} \left(T \dot{\mathbf{a}}_{n-1}^T + \frac{T^2}{2} \ddot{\mathbf{a}}_{n-1}^T + \frac{T^3}{6} \dddot{\mathbf{a}}_{n-1}^T \right) \mathcal{Q}_{\mathbf{a}} (\mathbf{a}_{n-1}) \right) \\ \dot{\mathbf{a}}_{n-1}^T + T \ddot{\mathbf{a}}_{n-1}^T + \frac{T^2}{2} \dddot{\mathbf{a}}_{n-1}^T \\ \ddot{\mathbf{a}}_{n-1}^T + T \ddot{\mathbf{a}}_{n-1}^T \\ \ddot{\mathbf{a}}_{n-1}^T \end{bmatrix} \quad (19.30)$$

19.2.3 Combined Dynamic Process Model

The translational and orientational process models can be combined into a single, more general, process model. First, define a combined state vector as

$$\mathbf{x}_n^{(i)} = \begin{bmatrix} \mathbf{p}_n^{(i)} \\ \mathbf{a}_n^{(i)} \end{bmatrix} \quad (19.31)$$

and a state noise model as

$$\mathbf{v}_n^{(i)} = \begin{bmatrix} \mathbf{v}_n^{(i)} \\ \boldsymbol{\rho}_n^{(i)} \end{bmatrix} \quad (19.32)$$

Now we can write the full dynamic model as

$$\mathbf{x}_n^{(i)} = \mathbf{f}_{n-1}^{(i)} \left(\mathbf{x}_{n-1}^{(i)} \right) + \mathbf{v}_{n-1}^{(i)} \quad (19.33)$$

where

$$\mathbf{f}_{n-1}^{(i)} \left(\mathbf{x}_{n-1}^{(i)} \right) = \begin{bmatrix} \mathbf{F}^{(i)} \mathbf{p}_{n-1}^{(i)} \\ \mathbf{g}^{(i)} \left(\mathbf{a}_{n-1}^{(i)} \right) \end{bmatrix} \quad (19.34)$$

19.2.4 The Dynamic Process Noise Models

The noise is assumed to be zero-mean Gaussian so we can consider the noise to be of the form

$$\mathbf{v}_n^{(i)} \sim \mathcal{N} \left(\mathbf{0}_{3(i+3)}, \mathbf{Q}^{(i)} \right), \quad i = 0, 1, 2, 3 \quad (19.35)$$

where $\mathbf{0}_{3(i+3)}$ is a $3(i+3) \times 3(i+3)$ matrix of zeros.

From [3,7], we find that for the constant position/orientation model the noise term represents velocity white noise, which for a continuous process reduces to

$$\mathbf{Q}^{(0)} = \begin{bmatrix} q_t^{(0)} T \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & q_r^{(0)} T \mathbf{I}_3 \end{bmatrix} \quad (19.36)$$

where $q_t^{(0)}$ and $q_r^{(0)}$ are related to the standard deviation of translational and rotational velocity noise, respectively.

For constant velocity models, the noise covariance is given by [3,7]

$$\mathbf{Q}^{(1)} = \begin{bmatrix} q_t^{(1)} \mathbf{Q}_1 & \mathbf{0}_6 \\ \mathbf{0}_6 & q_r^{(1)} \mathbf{Q}_1 \end{bmatrix} \quad (19.37)$$

with

$$\mathbf{Q}_1 = \begin{bmatrix} \frac{T^3}{3} \mathbf{I} & \frac{T^2}{2} \mathbf{I} \\ \frac{T^2}{2} \mathbf{I} & T \mathbf{I} \end{bmatrix} \quad (19.38)$$

For the higher order position equations, the constant acceleration dynamic noise model can be written [3,7]

$$\mathbf{Q}^{(2)} = \begin{bmatrix} q_t^{(2)} \mathbf{Q}_2 & \mathbf{0}_9 \\ \mathbf{0}_9 & q_r^{(2)} \mathbf{Q}_2 \end{bmatrix} \quad (19.39)$$

with

$$\mathbf{Q}_2 = \begin{bmatrix} \frac{T^5}{20} \mathbf{I}_3 & \frac{T^4}{8} \mathbf{I}_3 & \frac{T^3}{6} \mathbf{I}_3 \\ \frac{T^4}{8} \mathbf{I}_3 & \frac{T^3}{3} \mathbf{I}_3 & \frac{T^2}{2} \mathbf{I}_3 \\ \frac{T^3}{6} \mathbf{I}_3 & \frac{T^2}{2} \mathbf{I}_3 & T \mathbf{I}_3 \end{bmatrix} \quad (19.40)$$

and from [8] we the constant jerk noise covariance is given by

$$\mathbf{Q}^{(3)} = \begin{bmatrix} q_t^{(3)} \mathbf{Q}_3 & \mathbf{0}_{12} \\ \mathbf{0}_{12} & q_r^{(3)} \mathbf{Q}_3 \end{bmatrix} \quad (19.41)$$

with

$$\mathbf{Q}_3 = \begin{bmatrix} \frac{T^7}{252} \mathbf{I}_3 & \frac{T^6}{72} \mathbf{I}_3 & \frac{T^5}{30} \mathbf{I}_3 & \frac{T^4}{24} \mathbf{I}_3 \\ \frac{T^6}{72} \mathbf{I}_3 & \frac{T^5}{20} \mathbf{I}_3 & \frac{T^4}{8} \mathbf{I}_3 & \frac{T^3}{6} \mathbf{I}_3 \\ \frac{T^5}{30} \mathbf{I}_3 & \frac{T^4}{8} \mathbf{I}_3 & \frac{T^3}{3} \mathbf{I}_3 & \frac{T^2}{2} \mathbf{I}_3 \\ \frac{T^4}{24} \mathbf{I}_3 & \frac{T^3}{6} \mathbf{I}_3 & \frac{T^2}{2} \mathbf{I}_3 & T \mathbf{I}_3 \end{bmatrix} \quad (19.42)$$

19.3 COMPONENTS OF THE OBSERVATION MODEL

Assume that there are J cameras used to gather data on the state of the rigid body and that they observe feature points designated by markers placed around the surface of the rigid body. The location of each feature point relative to the body center of mass is known to within very tight tolerances due to prior calibrations. Let $\mathbf{S} = \{\mathbf{s}_i = [s_{i,x}, s_{i,y}, s_{i,z}]^\top ; i = 1, \dots, N_s\}$ be a set of fixed 3D time-invariant locations of the feature points in the body coordinate system. Because each camera is placed so that it has a different field of view, each camera will view a different subset of markers. For the j th camera at time t_n , let the number of viewed feature points be $K_{j,n}$ and let the individual visible feature points for that camera be designated $k_{1,j,n}, k_{2,j,n}, \dots, k_{K_{j,n},j,n}$ so that each $k_{i,j,n}$ refers to the point $\mathbf{s}_{k_{i,j,n}} \in \mathbf{S}$, where $1 \leq k_{i,j,n} \leq N_s$. Assume that all of the image frames across all cameras are synchronized so that the cameras each record an image frame simultaneously every 0.005 s. Also assume that the position and orientation of all cameras is calibrated for every frame and that the *association* of marker images for each camera image frame with predictions of those marker image positions is managed manually. Figure 19.3 shows the coordinates systems that depict the projection of a feature point onto a camera's image plane.

The observation vector \mathbf{z}_n , at time t_n , will therefore consist of a total of $M_n = \sum_{j=1}^J K_{j,n}$ two-dimensional pixel image marker locations (u, v) and \mathbf{z}_n can be written as

$$\mathbf{z}_n = [z_{1,1,u,n}, z_{1,1,v,n}, z_{2,1,u,n}, z_{2,1,v,n}, \dots, z_{i,j,u,n}, z_{i,j,v,n}, \dots, z_{K_{j,n},J,u,n}, z_{K_{j,n},J,v,n}]^\top \quad (19.43)$$

The task now is to define the functional dependence of the components of the $2M_n$ -dimensional observation vector to the state vector \mathbf{x}_n . If we include observation noise, the observation can be written in terms of the state vector as

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_n \quad (19.44)$$

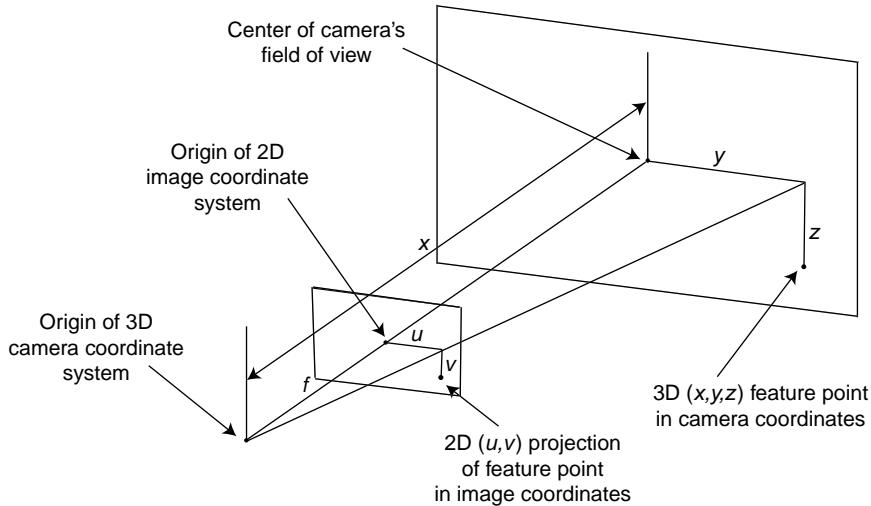


FIGURE 19.3 Projection of a feature point onto a camera’s image plane.

where $\mathbf{h}_n (\mathbf{x}_n)$ can be written as

$$\mathbf{h}_n (\mathbf{x}_n) = \left[\mathbf{h}_{1,1,n}^\top (\mathbf{x}_n), \mathbf{h}_{2,1,n}^\top (\mathbf{x}_n), \dots, \mathbf{h}_{i,j,n}^\top (\mathbf{x}_n), \dots, \mathbf{h}_{K,J,n}^\top (\mathbf{x}_n) \right]^\top \quad (19.45)$$

so that the task that remains is to define the individual subfunctions.

The observation noise is assumed to be a zero-mean Gaussian process, so $\mathbf{w}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, with \mathbf{R} defined as the diagonal noise covariance matrix of image pixel measurement variances given by

$$\mathbf{R} = \sigma_{pix}^2 \mathbf{I}_{2M_n} \quad (19.46)$$

where \mathbf{I}_{2M_n} is a $2M_n \times 2M_n$ identity matrix.

Note that \mathbf{z}_n , \mathbf{h}_n , and \mathbf{w}_n are of dimension $2M_n$, the dimension of \mathbf{x}_n depends on the order of the dynamic model, and M_n and \mathbf{h}_n are time-dependent because the number of features observed by each camera can change with time. For simplicity, the measurement noise variance, σ_{pix}^2 , of each pixel component, is assumed to be the same for each feature marker measurement.

Recall the difference between a point rotation and a frame rotation (see Appendix 19.A). To rotate and translate $\mathbf{s}_{k_i,j,n}$ from the rigid body’s coordinate system into the reference coordinate system, we perform a point rotation, then a translation

$$\mathbf{s}_{k_i,j,n}^{(ref)} = \mathbf{M}_{ref} \mathbf{s}_{k_i,j,n} + \mathbf{p}_n \quad (19.47)$$

where \mathbf{p}_n is the position vector of the center of mass at time t_n . The rotation matrix \mathbf{M}_{ref} can be obtained by recalling that the rotation of a feature marker point is a two step process. In the first step, the axis-angle adjustment to the rotation \mathbf{a}_n is combined with the external quaternion \mathbf{q}_{n-1} that represents the prior rotation. The axis-angle

vector \mathbf{a}_n is first converted to a rotation matrix $\mathbf{M}_{\mathbf{a}_n}$ using the function $\mathcal{M}_{\mathbf{a}}$ defined by (19.164)

$$\mathbf{M}_{\mathbf{a}_n} = \mathcal{M}_{\mathbf{a}}(\mathbf{a}_n) \quad (19.48)$$

and the external quaternion \mathbf{q}_{n-1} is converted to a rotation matrix $\mathbf{M}_{\mathbf{q}_{n-1}}$ using the function $\mathcal{M}_{\mathbf{q}}$ defined by (19.163)

$$\mathbf{M}_{\mathbf{q}_{n-1}} = \mathcal{M}_{\mathbf{q}}(\mathbf{q}_{n-1}) \quad (19.49)$$

then the two matrices are multiplied to produce \mathbf{M}_{ref}

$$\mathbf{M}_{\text{ref}} = \mathbf{M}_{\mathbf{a}_n} \mathbf{M}_{\mathbf{q}_{n-1}} \quad (19.50)$$

To rotate and translate $\mathbf{s}_{k_{i,j,n}}^{(\text{ref})}$ from reference coordinates into the j th camera's coordinate system, we define $\mathbf{p}_{Cj,n}$, $\mathbf{q}_{Cj,n}$, and $f_{Cj,n}$ to be the calibrated position, orientation, and focal length, in pixels, of camera C_j and we then perform a negative translation followed by a frame rotation

$$\mathbf{s}_{k_{i,j,n}}^{(C_j)} = \mathbf{M}_{Cj,n}^T \left(\mathbf{s}_{k_{i,j,n}}^{(\text{ref})} - \mathbf{p}_{Cj,n} \right) \quad (19.51)$$

where $\mathbf{M}_{Cj} = \mathcal{M}_{\mathbf{q}}(\mathbf{q}_{Cj})$.

Finally as shown in Figure 19.3, the 3D $\mathbf{s}_{k_{i,j,n}}^{(C_j)}$ is projected into the 2D image plane with

$$\mathbf{s}_{k_{i,j,n}}^{(\text{image})} = \begin{pmatrix} f_{Cj,n} \\ \mathbf{s}_{k_{i,j,n},x}^{(C_j)} \end{pmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{s}_{k_{i,j,n}}^{(C_j)} \quad (19.52)$$

where $\mathbf{s}_{k_{i,j,n}}^{(\text{image})}$ has components $[z_{i,j,u,n}, z_{i,j,v,n}]^T \in \mathbf{z}_n$ given by

$$\mathbf{z}_{i,j,n} = \begin{bmatrix} z_{i,j,u,n} \\ z_{i,j,v,n} \end{bmatrix} = \begin{bmatrix} s_{k_{i,j,n},u}^{(\text{image})} \\ s_{k_{i,j,n},v}^{(\text{image})} \end{bmatrix} \quad (19.53)$$

and (u, v) are image coordinates in pixels. Equation (19.53) represents the image pixel location of the i th feature point for *one* camera. Since the observation vector consists of a total of M_N feature point sets from J cameras, full observation vector becomes

$$\mathbf{z}_n = \left[\mathbf{z}_{1,1,n}^T, \mathbf{z}_{2,1,n}^T, \dots, \mathbf{z}_{i,j,n}^T, \dots, \mathbf{z}_{J,n,J,n}^T \right]^T \quad (19.54)$$

with the noise free $\mathbf{z}_{i,j,n} = \mathbf{h}_{i,j,n}(\mathbf{x}_n)$ given by the combination of (19.47) through (19.53).

19.4 ESTIMATION METHODS

Several estimation methods can be used for the recursive estimation of the state vector and its covariance matrix. In the sections below, we discuss the three specific methods that we applied to this photogrammetry tracking problem, including an NLSSQ solver, a Gaussian UKF, and an unscented combination particle filter that we call the unscented Gaussian particle filter (UGPF). Application of these three estimation methods are presented in this section and their performance will be compared in Section 19.6.

19.4.1 A Nonlinear Least Squares Estimation Method

The standard method used in the past for estimation of the position and orientation of a rigid body based on video image data has been the NLSSQ method that assumes a *noise-free* dynamic equation and produces an estimate that minimizes the mean squared difference between an actual measurement set and a prediction of that measurement set. In our implementation, the state vector is taken as only the pose of the rigid body at time t_n . We will drop all time indices because the pose is estimated anew for each time step, as will be shown below.

From (19.31), we define the NLSSQ state vector as

$$\mathbf{x} = [\mathbf{p}^\top, \mathbf{a}^\top]^\top \quad (19.55)$$

and we recall that the rigid body's orientation is represented by a combination of both \mathbf{a} and the quaternion \mathbf{q} that represents the prior (or initial) orientation.

The observation vector is shown in (19.54). The dependence of the observation vector on the state vector can be written as

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{w} \quad (19.56)$$

with $\mathbf{h}(\mathbf{x})$ decomposed into components $\mathbf{h}_{i,j}(\mathbf{x})$ as in (19.45), where each $\mathbf{h}_{i,j}(\mathbf{x})$ is given by the combination of (19.47) through (19.53) for each $\mathbf{z}_{i,j}$ component of \mathbf{z} . We will assume that $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ and there are $M = \sum_{j=1}^J K_j$ feature marker two-dimensional image measurements, making \mathbf{R} a $2M \times 2M$ diagonal covariance matrix where the same variance is used for all image pixel location variances. Thus we can write

$$\mathbf{R} = \sigma_{\text{pix}}^2 \mathbf{I}_{2M} \quad (19.57)$$

The NLSSQ method essentially uses a weighted least-squares estimator of \mathbf{x} given measurements \mathbf{z} by minimizing the cost function

$$q(\mathbf{x}) = [\mathbf{z} - \mathbf{h}(\mathbf{x})]^\top \mathbf{R}^{-1} [\mathbf{z} - \mathbf{h}(\mathbf{x})] \quad (19.58)$$

Thus, an estimate of \mathbf{x} , written as $\hat{\mathbf{x}}$, can be obtained from

$$\nabla_{\mathbf{x}} q(\mathbf{x}) = -2 [\nabla_{\mathbf{x}} \mathbf{h}^\top(\mathbf{x})] \mathbf{R}^{-1} [\mathbf{z} - \mathbf{h}(\mathbf{x})] = \mathbf{0} \quad (19.59)$$

Define \mathbf{H} as

$$\begin{aligned}\mathbf{H} &\triangleq \nabla_{\mathbf{x}} \mathbf{h}^T(\mathbf{x}) = \nabla_{\mathbf{x}} [\mathbf{h}_{1,1}(\mathbf{x}), \mathbf{h}_{2,1}(\mathbf{x}), \dots, \mathbf{h}_{K_J,J}(\mathbf{x})] \\ &= \begin{bmatrix} \frac{\partial \mathbf{h}_{1,1}}{\partial p_x} & \frac{\partial \mathbf{h}_{2,1}}{\partial p_x} & \dots & \frac{\partial \mathbf{h}_{K_J,J}}{\partial p_x} \\ \frac{\partial \mathbf{h}_{1,1}}{\partial p_y} & \frac{\partial \mathbf{h}_{2,1}}{\partial p_y} & \dots & \frac{\partial \mathbf{h}_{K_J,J}}{\partial p_y} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{h}_{1,1}}{\partial a_z} & \frac{\partial \mathbf{h}_{2,1}}{\partial a_z} & \dots & \frac{\partial \mathbf{h}_{K_J,J}}{\partial a_z} \end{bmatrix} \quad (19.60)\end{aligned}$$

with $\mathbf{h}_{i,j}$ the nonlinear function that transforms \mathbf{x} into the i th marker's location in camera J's image to produce $\mathbf{z}_{i,j}$. Now (19.59) can be rewritten as

$$\mathbf{H}\mathbf{R}^{-1}[\mathbf{z} - \mathbf{h}(\mathbf{x})] = \mathbf{0} \quad (19.61)$$

Given some initial estimate (guess) of the state, $\hat{\mathbf{x}}_0$, the nonlinear function $\mathbf{h}(\mathbf{x})$ can be expanded in a Taylor series about $\hat{\mathbf{x}}_0$

$$\mathbf{h}(\mathbf{x}) = \mathbf{h}(\hat{\mathbf{x}}_0) + [\nabla_{\mathbf{x}} \mathbf{h}^T(\mathbf{x})]_{\mathbf{x}=\hat{\mathbf{x}}_0}^\top [\mathbf{x} - \hat{\mathbf{x}}_0] + \dots, \quad (19.62)$$

or, keeping just the linear terms of the series

$$\mathbf{h}(\mathbf{x}) = \mathbf{h}(\hat{\mathbf{x}}_0) + \mathbf{H}_0 [\mathbf{x} - \hat{\mathbf{x}}_0] \quad (19.63)$$

where

$$\mathbf{H}_0 \triangleq [\nabla_{\mathbf{x}} \mathbf{h}^T(\mathbf{x})]_{\mathbf{x}=\hat{\mathbf{x}}_0}^\top \quad (19.64)$$

Equation (19.61) now becomes

$$\mathbf{H}\mathbf{R}^{-1}[\mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}_0) - \mathbf{H}_0 [\mathbf{x} - \hat{\mathbf{x}}_0]] = \mathbf{0} \quad (19.65)$$

Solving for $\hat{\mathbf{x}}_1 = \mathbf{x}$ and letting $\mathbf{H} \rightarrow \mathbf{H}_0$ yields

$$\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_0 + [\mathbf{H}_0 \mathbf{R}^{-1} \mathbf{H}_0^\top]^{-1} \mathbf{H}_0 \mathbf{R}^{-1} [\mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}_0)] \quad (19.66)$$

The process is repeated using the new estimate of \mathbf{x} as the initial estimate and iterating, until after p iterations, we obtain the Gauss–Newton normal equations

$$\hat{\mathbf{x}}_p = \hat{\mathbf{x}}_{p-1} + [\mathbf{H}_{p-1} \mathbf{R}^{-1} \mathbf{H}_{p-1}^\top]^{-1} \mathbf{H}_{p-1} \mathbf{R}^{-1} [\mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}_{p-1})] \quad (19.67)$$

It follows immediately that the covariance of $\hat{\mathbf{x}}_p$ is given by

$$\mathbf{P}_p = [\mathbf{H}_{p-1} \mathbf{R}^{-1} \mathbf{H}_{p-1}^\top]^{-1} \quad (19.68)$$

From (19.57) we see that \mathbf{R} is a diagonal matrix and these last two equations reduce to

$$\begin{aligned}\hat{\mathbf{x}}_p &= \hat{\mathbf{x}}_{p-1} + \left[\mathbf{H}_{p-1} \left(\sigma_{\text{pix}}^2 \mathbf{I}_{2M} \right)^{-1} \mathbf{H}_{p-1}^\top \right]^{-1} \mathbf{H}_{p-1} \left(\sigma_{\text{pix}}^2 \mathbf{I}_{2M} \right)^{-1} [\mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}_{p-1})] \\ &= \hat{\mathbf{x}}_{p-1} + \left(\mathbf{H}_{p-1}^\top \right)^{-1} [\mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}_{p-1})]\end{aligned}\quad (19.69)$$

and

$$\mathbf{P}_p = \sigma_{\text{pix}}^{-2} \left[\mathbf{H}_{p-1} \mathbf{H}_{p-1}^\top \right]^{-1} \quad (19.70)$$

The process is repeated until $\mathbf{e}_p \triangleq \hat{\mathbf{x}}_p - \hat{\mathbf{x}}_{p-1}$ is below a vector of threshold values.

Thus, at each time step t_n , a new set of images \mathbf{z}_n are obtained and this procedure is recalculated to generate an estimate $\hat{\mathbf{x}}_n = \hat{\mathbf{x}}_{p,n}$ having an estimated covariance $\mathbf{P}_{p,n}$. Note that in Matlab, this NLSSQ method is available as the subroutine “lsqnonlin” that takes the function (19.58) and the variable \mathbf{x} as inputs and outputs a value $\hat{\mathbf{x}}$ that minimizes the function. Although this function does not estimate the covariance matrix, certain forms of the function also return the Jacobian \mathbf{H}_{p-1} , allowing the covariance matrix estimate to be calculated.

Since this formulation of the NLSSQ is iterative but not recursive (in time), a dynamic model is not used, and the observations depend only on the position and orientation of the rigid body, only the position and orientation can be estimated. In order to estimate any of the time derivatives of position and orientation, some form of smoothing filter or finite difference filter must be used as a post processor in order to obtain the time derivatives of \mathbf{x} .

As an alternative for future implementations, one could include a dynamic model and take several successive-in-time measurement sets augmenting them to the vector $\mathbf{h}(\mathbf{x})$. This reformulation would allow the state vector to include derivative terms that could then be included in the estimation procedure. However, that would increase the size of both $\mathbf{h}(\mathbf{x})$ and \mathbf{H} increasing the computational burden.

19.4.2 An Unscented Kalman Filter Method

The UKF is part of a class of sigma point Kalman filters that require all noise densities to be Gaussian. For an i th-order model, the dynamic equation is nonlinear and is given by (19.33). The observation model is also nonlinear and is shown in (19.44). Both the dynamic and observation noise densities are assumed to be Gaussian. As shown in Part II of this book, if one makes an affine transformation of the nonlinear functions, and then expands them in a general polynomial, the Gaussian-weighted moment integrals reduce to a sum over a weighted set of sigma points, where the weights and sigma points depend on the sigma point filter chosen. We present the formulation of the general sigma point Kalman filter for estimation of the state vector and its covariance matrix that is based on a successive-in-time set of image feature marker observations.

The filter is started with an initial estimate for the state vector $\hat{\mathbf{x}}_0 = [\hat{\mathbf{p}}_0^\top, \hat{\mathbf{a}}_0^\top]^\top$, its associated quaternion \mathbf{q}_0 (as described in Section 19.4.4.2) and its covariance \mathbf{P}_0^{xx} . The dimensions of \mathbf{x} , \mathbf{P}^{xx} , and \mathbf{Q} will be dependent on the model order chosen, as described in Section 19.2.1 and 19.2.4.

The state sigma points are calculated from

$$\chi_{n-1|n-1}^{(j)} = \hat{\mathbf{x}}_{n-1|n-1} + \mathbf{D}_{n-1|n-1} \mathbf{c}^{(j)}, \quad j = 0, 1, \dots, N_s \quad (19.71)$$

where $\hat{\mathbf{x}}_{n-1|n-1} \rightarrow \hat{\mathbf{x}}_0$ and $\mathbf{P}_{n-1|n-1}^{\text{xx}} \rightarrow \mathbf{P}_0^{\text{xx}}$ for the initial set of sigma points and $\mathbf{D}_{n-1|n-1}$ is defined by

$$\mathbf{P}_{n-1|n-1}^{\text{xx}} = \mathbf{D}_{n-1|n-1} \mathbf{D}_{n-1|n-1}^\top \quad (19.72)$$

Also, N_s is the number of sigma points and $\mathbf{c}^{(j)}$ is the j th sigma point. The set $\{w_j, \mathbf{c}^{(j)}\}$, used by the sigma point filters, are dependent on the size of the state vector and the sigma point filter to be implemented. They can be generated using the subroutines presented in Listings 13.1 and 13.2.

From Chapter 13, the state prediction equations for all sigma point Kalman filters are given by

$$\chi_{n|n-1}^{(j)} = \mathbf{f}(\chi_{n-1|n-1}^{(j)}) \quad (19.73)$$

$$\hat{\mathbf{x}}_{n|n-1} = \sum_{j=0}^{N_s} w_j \chi_{n|n-1}^{(j)} \quad (19.74)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\text{xx}} = & \sum_{j=0}^{N_s} w_j (\chi_{n|n-1}^{(j)} - \hat{\mathbf{x}}_{n|n-1}) (\chi_{n|n-1}^{(j)} - \hat{\mathbf{x}}_{n|n-1})^\top \\ & + \mathbf{Q} \end{aligned} \quad (19.75)$$

The observation prediction equations are

$$\hat{\mathbf{z}}_{n|n-1} = \sum_{j=0}^{N_s} w_j \mathbf{h}(\chi_{n|n-1}^{(j)}) \quad (19.76)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\text{zz}} = & \sum_{j=0}^{N_s} w_j \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right] \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^\top \\ & + \mathbf{R} \end{aligned} \quad (19.77)$$

$$\begin{aligned} \mathbf{P}_{n|n-1}^{\text{xz}} = & \sum_{j=0}^{N_s} w_j \left[\chi_{n|n-1}^{(j)} - \hat{\mathbf{x}}_{n|n-1} \right] \\ & \times \left[\mathbf{h}(\chi_{n|n-1}^{(j)}) - \hat{\mathbf{z}}_{n|n-1} \right]^\top \end{aligned} \quad (19.78)$$

And finally, the state update equations are

$$\mathbf{K}_n \triangleq \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} (\mathbf{P}_{n|n-1}^{\mathbf{z}\mathbf{z}})^{-1} \quad (19.79)$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{z}_n^o - \hat{\mathbf{z}}_{n|n-1}) \quad (19.80)$$

$$\mathbf{P}_{n|n}^{\mathbf{x}\mathbf{x}} = \mathbf{P}_{n|n-1}^{\mathbf{x}\mathbf{x}} - \mathbf{K}_n \mathbf{P}_{n|n-1}^{\mathbf{z}\mathbf{z}} \mathbf{K}_n^\top \quad (19.81)$$

where \mathbf{z}_n^o is the observation at time t_n . Once the state vector $\hat{\mathbf{x}}_{n|n} = [\hat{\mathbf{p}}_{n|n}^\top, \hat{\mathbf{a}}_{n|n}^\top]^\top$ has been obtained at the end of a filter iteration, a new unit quaternion \mathbf{q}_n is produced from $\hat{\mathbf{a}}_{n|n}$ and \mathbf{q}_{n-1} after which $\hat{\mathbf{a}}_{n|n}$ is reset according to

$$\mathbf{q}_n = \mathcal{Q}_{\mathbf{a}}(\hat{\mathbf{a}}_{n|n}) \mathbf{q}_{n-1} \quad (19.82)$$

$$\hat{\mathbf{a}}_{n|n} \rightarrow [0, 0, 0]^\top \quad (19.83)$$

For higher order models and for a large number of feature marker image dimensions, the UKF is the best sigma point Kalman filter because the number of sigma points required goes as $N_s = 2n + 1$, where n is the dimension of the state vector. For all applications of this filter, we always choose $w_0 = 0$, so $N_s = 2n$ and $w_j = 1/2n$, $j = 1, \dots, 2n$. In addition, $\mathbf{e}^{(j)} = \sqrt{n}\mathbf{r}^{(j)} = \sqrt{n}[1] \in \mathbb{R}^n$ (see Section 2.2 for an explanation of this notation.) The spherical simplex Kalman filter (SSKF) has fewer points ($N_s = n + 1$) and we examined its performance briefly and discovered that it has performance results that are similar to those of the UKF but such results will not be presented here.

19.4.3 Estimation Using the Unscented Combination Particle Filter

In Chapter 17 we introduced the combination particle filter, a particle filter method that does not require resampling. We use it here as another alternative to the NLSSQ. The process block diagram for a combination particle filter that uses an UKF to generate the importance samples is shown in Figure 17.3. In our application of this filter to the photogrammetry rigid body pose estimation problem, the likelihood function for each particle, $p(\mathbf{z}_n | \mathbf{x}_n^{(i)})$, is Gaussian because (19.44) can be rewritten as

$$\mathbf{w}_n^{(i)} = \mathbf{z}_n^o - \mathbf{h}_n(\mathbf{x}_n^{(i)}) \quad (19.84)$$

Now $p(\mathbf{z}_n | \mathbf{x}_n^{(i)}) = \mathcal{N}([\mathbf{z}_n^o - \mathbf{h}_n(\mathbf{x}_n^{(i)})], \mathbf{R})$, and we can write the particle likelihood function as

$$p(\mathbf{z}_n | \mathbf{x}_n^{(i)}) = c \exp \left\{ -\frac{1}{2} \left[\mathbf{z}_n^o - \mathbf{h}_n(\mathbf{x}_n^{(i)}) \right]^\top \mathbf{R}^{-1} \left[\mathbf{z}_n^o - \mathbf{h}_n(\mathbf{x}_n^{(i)}) \right] \right\} \quad (19.85)$$

The normalization c need not be included unless it is needed for numerical stability, since it will be canceled out when the weights are normalized during the weight update step. For our application of this filter, we used 20,000 particles. Since the full procedure is presented in Figure 17.3, we will not reiterate it here.

19.4.4 Initializing the Estimator

All of the estimation methods described above require an initial estimate $\hat{\mathbf{x}}$ and some also require an initial estimate of $\mathbf{P}^{\mathbf{xx}}$. There are, however, some differences in what the initial estimate represents between the NLLSQ estimator and all other estimators. In the subsections below, we address these issues.

19.4.4.1 Initializing the NLLSQ. Since the NLLSQ is an iterative filter (one that loops through a set of code until an error tolerance limit is reached), it must be reinitialized at every time step. For time steps prior to the store release event, the rigid body is at rest, so the position and orientation initial estimates can be taken as

$$\hat{\mathbf{x}}_0 = [\hat{\mathbf{p}}_0^\top, \hat{\mathbf{a}}_0^\top]^\top = [0, 0, 0, 0, 0, 0]^\top \quad (19.86)$$

Therefore, the initial estimate for $\hat{\mathbf{q}}_0$ is the quaternion

$$\hat{\mathbf{q}}_0 = (1, 0, 0, 0) \quad (19.87)$$

which represents a zero angle. For other situations that differ from our synthetic test, $\hat{\mathbf{p}}_0^\top$ and $\hat{\mathbf{q}}_0$ might take on some nominal nonzero value, while $\hat{\mathbf{a}}_0$ should still be set to $\hat{\mathbf{a}}_0^\top = [0, 0, 0]^\top$.

After the store release event, the rigid body is undergoing translation and rotation from one time step to the next, so the final estimate of position and rotation from the previous time step can be used to initialize the state vector for the current time step. Let $\hat{\mathbf{x}}_n^+ = [\hat{\mathbf{p}}_n^{+\top}, \hat{\mathbf{a}}_n^{+\top}]^\top$ be the posterior estimate from frame n , and let $\hat{\mathbf{q}}_{n-1}$ represent the entire rotation from frame $n - 1$. Then $\hat{\mathbf{q}}_n$ and $\hat{\mathbf{a}}_n$ should be set according to

$$\hat{\mathbf{q}}_n = Q_a(\hat{\mathbf{a}}_n^+) \hat{\mathbf{q}}_{n-1} \quad (19.88)$$

$$\hat{\mathbf{a}}_n = [0, 0, 0]^\top \quad (19.89)$$

The prior initial estimate for frame $n + 1$ should be set as $\hat{\mathbf{x}}_{n+1}^- = [\hat{\mathbf{p}}_{n+1}^{-\top}, \hat{\mathbf{a}}_{n+1}^{-\top}]^\top$, where

$$\hat{\mathbf{p}}_{n+1}^- = \hat{\mathbf{p}}_n^+ \quad (19.90)$$

$$\hat{\mathbf{a}}_{n+1}^- = \hat{\mathbf{a}}_n^+ = [0, 0, 0]^\top \quad (19.91)$$

19.4.4.2 Initialization of Bayesian Filters. For all filters other than the NLLSQ, initial values are needed for $\hat{\mathbf{x}}_0$, $\mathbf{P}_0^{\mathbf{xx}}$, and $\hat{\mathbf{q}}_0$. For consistency, we will use the same initialization for all of the filters. However, the required initialization will change from one dynamic model order to another. So, we will need different initialization for the different model orders: constant position/orientation, constant velocity, constant acceleration, and constant jerk. Since each lower order model can be thought of as a degenerative case of the next higher order model, we need only initialize the highest (third) order model variables $\hat{\mathbf{x}}_0^{(3)}$, $\mathbf{P}_0^{(3)}$, and $\hat{\mathbf{q}}_0$, then truncate if needed to initialize

a lower order model. Since the rigid body is initially at rest, the initial estimate of $\hat{\mathbf{x}}_0^{(3)}$ is

$$\hat{\mathbf{x}}_0^{(3)} = \begin{bmatrix} \hat{\mathbf{p}}_0^{(3)} \\ \hat{\mathbf{a}}_0^{(3)} \end{bmatrix} = \mathbf{0} \quad (19.92)$$

where $\mathbf{0}$ is a vector of 24 zeros.

The initial estimate of $\hat{\mathbf{q}}_0$ for the synthetic test is the identity quaternion

$$\hat{\mathbf{q}}_0 = (1, 0, 0, 0) \quad (19.93)$$

which represents a zero angle. For other situations that differ from our synthetic test, $\hat{\mathbf{p}}_0$ and $\hat{\mathbf{q}}_0$ might take on some nominal nonzero value, while $\hat{\mathbf{a}}_0$ should still be set to $\hat{\mathbf{a}}_0 = [0, 0, 0]^T$. Since normally the body is at rest, the derivatives of $\hat{\mathbf{p}}_0$ and $\hat{\mathbf{a}}_0$ should be still set to zero.

To initialize \mathbf{P}_0^{xx} , let σ_p , $\sigma_{\dot{p}}$, $\sigma_{\ddot{p}}$, and $\sigma_{\ddot{\ddot{p}}}$ be the standard deviation in translational position, velocity, acceleration, and jerk, respectively. Also let σ_a , $\sigma_{\dot{a}}$, $\sigma_{\ddot{a}}$, and $\sigma_{\ddot{\ddot{a}}}$ be the standard deviation in angular position, velocity, acceleration, and jerk, respectively. We set the standard deviations to

$$\sigma_p = 1, \sigma_{\dot{p}} = 5, \sigma_{\ddot{p}} = 1000, \sigma_{\ddot{\ddot{p}}} = 5000 \quad (19.94)$$

$$\sigma_a = 1, \sigma_{\dot{a}} = 1, \sigma_{\ddot{a}} = 50, \sigma_{\ddot{\ddot{a}}} = 2000 \quad (19.95)$$

where the translational units are inches and seconds and the angular units are degrees and seconds. Now, the initial translational covariance submatrix is

$$\mathbf{P}_{0,\text{trans}}^{\text{xx}(3)} = \begin{bmatrix} \sigma_p^2 \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \sigma_{\dot{p}}^2 \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \sigma_{\ddot{p}}^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \sigma_{\ddot{\ddot{p}}}^2 \mathbf{I}_3 \end{bmatrix} \quad (19.96)$$

and the initial angular covariance submatrix is

$$\mathbf{P}_{0,\text{ang}}^{\text{xx}(3)} = C_{d2r} \begin{bmatrix} \sigma_a^2 \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \sigma_{\dot{a}}^2 \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \sigma_{\ddot{a}}^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \sigma_{\ddot{\ddot{a}}}^2 \mathbf{I}_3 \end{bmatrix} \quad (19.97)$$

where $C_{d2r} = \pi/180$ converts degrees to radians. The two submatrices are combined to form the initial covariance matrix

$$\mathbf{P}_0^{\text{xx}(3)} = \begin{bmatrix} \mathbf{P}_{0,\text{trans}}^{\text{xx}(3)} & \mathbf{0}_{12} \\ \mathbf{0}_{12} & \mathbf{P}_{0,\text{ang}}^{\text{xx}(3)} \end{bmatrix} \quad (19.98)$$

The Bayesian estimators are rarely reinitialized after the filtering process has begun, so the above initialization is the only one considered.

19.5 THE GENERATION OF SYNTHETIC DATA

In order to evaluate the performance of the various tracking filter methods, we developed a synthetic rigid body and created synthetic trajectories indicative of the store release event. This gives us a synthetic “truth” trajectory that allows us to create synthetic observation sets as seen from synthetic cameras by adding Monte Carlo sets of pixel noise. The synthetic camera measurements can then be used to perform Monte Carlo RMS error analysis comparisons of the estimation (solver) methods.

19.5.1 Synthetic Rigid Body Feature Points

We first construct a rigid body with eight feature points out of a unit cube by constructing a 3×8 matrix \mathbf{S} , where the i th column of \mathbf{S} is the feature point $\mathbf{s}_i = (s_{i,x}, s_{i,y}, s_{i,z})$

$$\mathbf{S} = \begin{bmatrix} -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (19.99)$$

Then we apply an affine transformation that makes the cube long and thin

$$\mathbf{S} = \mathbf{T}\mathbf{S} \quad (19.100)$$

where

$$\mathbf{T} = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 30 \end{bmatrix} \quad (19.101)$$

19.5.2 Synthetic Trajectory

To build a synthetic trajectory we first generate a sequence of times \mathbf{t} , where

$$\mathbf{t} = \{t_n = 0.005 * n, n = 0, 1, 2, \dots, 200\} \quad (19.102)$$

Then we define a damped sinusoidal function (damped harmonic oscillator)

$$f(t_n, \alpha, \beta, \gamma) = \begin{cases} \alpha e^{-\beta(t_n - t_r)} \{\cos [\gamma(t_n - t_r)] - 1\}, & t_n \geq t_r \\ 0, & t_n < t_r \end{cases} \quad (19.103)$$

where t_r is the store release time (set to $t_r = 0.1$) and the constants α , β , and γ represent the amplitude, damping factor, and frequency, respectively. Prior to time t_r the body is motionless and after time t_r the body is falling.

The first two derivatives of $f(t_n, \alpha, \beta, \gamma)$ are given by

$$\dot{f}(t_n, \alpha, \beta, \gamma) = \begin{cases} -\alpha e^{-\beta(t_n-t_r)} \{ \beta \cos [\gamma(t_n - t_r)] - \beta \\ \quad + \gamma \sin [\gamma(t_n - t_r)] \}, & t_n \geq t_r \\ 0, & t_n < t_r \end{cases} \quad (19.104)$$

and

$$\ddot{f}(t_n, \alpha, \beta, \gamma) = \begin{cases} -\alpha e^{-\beta(t_n-t_r)} \{ (\beta^2 - \gamma^2) \cos [\gamma(t_n - t_r)] \\ \quad - \beta^2 + 2\beta\gamma \sin [\gamma(t_n - t_r)] \}, & t_n \geq t_r \\ 0, & t_n < t_r \end{cases} \quad (19.105)$$

Using the damped sinusoidal signal, we generate six sequences, one for each of the six parameters ($x, y, z, \phi, \theta, \varphi$), where ϕ, θ , and φ are the Euler angles *roll*, *pitch*, and *yaw*, and $t_n \in \mathbf{t}$

$$x(t_n) = f(t_n, \alpha_x, \beta_x, \gamma_x); (\alpha_x, \beta_x, \gamma_x) = (100, 2, 2) \quad (19.106)$$

$$y(t_n) = f(t_n, \alpha_y, \beta_y, \gamma_y); (\alpha_y, \beta_y, \gamma_y) = (-200, 3, 1) \quad (19.107)$$

$$z(t_n) = f(t_n, \alpha_z, \beta_z, \gamma_z); (\alpha_z, \beta_z, \gamma_z) = (-2000, 1.5, 0.5) \quad (19.108)$$

$$\phi(t_n) = f(t_n, \alpha_\phi, \beta_\phi, \gamma_\phi); (\alpha_\phi, \beta_\phi, \gamma_\phi) = (500, 1.5, 1.25) \quad (19.109)$$

$$\theta(t_n) = f(t_n, \alpha_\theta, \beta_\theta, \gamma_\theta); (\alpha_\theta, \beta_\theta, \gamma_\theta) = (-10, 2, 10) \quad (19.110)$$

$$\varphi(t_n) = f(t_n, \alpha_\varphi, \beta_\varphi, \gamma_\varphi); (\alpha_\varphi, \beta_\varphi, \gamma_\varphi) = (15, 0.8, 14) \quad (19.111)$$

We next generate $\dot{x}(t_n), \dot{y}(t_n), \dot{z}(t_n), \dot{\phi}(t_n), \dot{\theta}(t_n)$, and $\dot{\varphi}(t_n)$ as well as $\ddot{x}(t_n), \ddot{y}(t_n), \ddot{z}(t_n), \ddot{\phi}(t_n), \ddot{\theta}(t_n)$, and $\ddot{\varphi}(t_n)$ according to (19.104) and (19.105).

We can now easily construct the translational components of our synthetic trajectory

$$\mathbf{p}_{n,\text{true}} = [x(t_n), y(t_n), z(t_n)]^\top \quad (19.112)$$

$$\dot{\mathbf{p}}_{n,\text{true}} = [\dot{x}(t_n), \dot{y}(t_n), \dot{z}(t_n)]^\top \quad (19.113)$$

$$\ddot{\mathbf{p}}_{n,\text{true}} = [\ddot{x}(t_n), \ddot{y}(t_n), \ddot{z}(t_n)]^\top \quad (19.114)$$

For the rotational components, we must construct the axis-angle $\mathbf{a}_{n,\text{true}}$, and the unit quaternion $\mathbf{q}_{n,\text{true}}$. For $\mathbf{a}_{n,\text{true}}$, we write

$$\mathbf{a}_{n,\text{true}} = [0, 0, 0]^\top, \forall n \quad (19.115)$$

To construct $\mathbf{q}_{n,\text{true}}$, we first introduce the conversion function $\mathcal{Q}_{\text{Euler}}(\phi, \theta, \varphi) : \mathbb{R}^3 \rightarrow \mathbf{H}$ that converts Euler angles (ϕ, θ, φ) into a unit quaternion

$$\mathcal{Q}_{\text{Euler}}(\phi, \theta, \varphi) = \mathbf{q}_\phi \mathbf{q}_\theta \mathbf{q}_\varphi \quad (19.116)$$

where \mathbf{q}_ϕ , \mathbf{q}_θ , and \mathbf{q}_φ are the quaternions that represent each of the three Euler angles by themselves

$$\mathbf{q}_\phi = \left[\cos \frac{\phi}{2}, \sin \frac{\phi}{2}, 0, 0 \right]^\top \quad (19.117)$$

$$\mathbf{q}_\theta = \left[\cos \frac{\theta}{2}, 0, \sin \frac{\theta}{2}, 0 \right]^\top \quad (19.118)$$

$$\mathbf{q}_\varphi = \left[\cos \frac{\varphi}{2}, 0, 0, \sin \frac{\varphi}{2} \right]^\top \quad (19.119)$$

Now $\mathbf{q}_{n,\text{true}}$ can be defined as

$$\begin{aligned} \mathbf{q}_{n,\text{true}} &= \mathcal{Q}_{\text{Euler}}(\phi(t_n), \theta(t_n), \varphi(t_n)) \\ &= \mathbf{q}_\phi \mathbf{q}_\theta \mathbf{q}_\varphi \end{aligned} \quad (19.120)$$

where the “ n,true ” portion of the subscripts have been omitted for brevity and a quaternion multiplication from (19.15) is implied. Now, we set

$$\mathbf{x}_{n,\text{true}} = [\mathbf{p}_{n,\text{true}}^\top, \mathbf{a}_{n,\text{true}}^\top]^\top = [\mathbf{p}_{n,\text{true}}^\top, [0, 0, 0]^\top]^\top \quad (19.121)$$

To construct truth data for the purpose of comparison against $\dot{\mathbf{a}}_n$ and $\ddot{\mathbf{a}}_n$, we must express synthetic first and second derivatives of rotation. We do so in body-reference coordinates rather than in the fixed reference frame because we judge that using a body-referenced coordinate system for derivatives is more intuitive for humans. We shall call the body-referenced angular derivatives $\dot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})}$ and $\ddot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})}$.

We begin by recalling the well-known quaternion differentiation formula [9]

$$\dot{\mathbf{q}} = \frac{1}{2} \boldsymbol{\omega} \mathbf{q} \quad (19.122)$$

where \mathbf{q} is a unit quaternion and the quaternion $\boldsymbol{\omega} = [0, \omega_x, \omega_y, \omega_z]^\top$ is a body referenced rotation rate vector converted to a pure quaternion (“pure” because its scalar component is 0). The relationship between \mathbf{q} , $\dot{\mathbf{q}}$, and $\boldsymbol{\omega}$ for a *frame* rotation is specified by (19.122), whereas $\mathbf{q}_{n,\text{true}}$ is a *point* rotation. If \mathbf{q} is a frame rotation, and $\mathbf{q}_{n,\text{true}}$ is the equivalent point rotation, then $\mathbf{q}_{n,\text{true}} = \mathbf{q}^*$, where \mathbf{q}^* is the quaternion conjugate of \mathbf{q} . That is, if $\mathbf{q} = [q_s, q_x, q_y, q_z]^\top$, then $\mathbf{q}^* = [q_s, -q_x, -q_y, -q_z]^\top$. Before taking the conjugate of both sides of (19.124), recall three facts from quaternion algebra. First, if \mathbf{w} is a pure quaternion (where the scalar component $w_s = 0$), then $\mathbf{w}^* = -\mathbf{w}$. Second, for the product of quaternions, \mathbf{q}_1 and \mathbf{q}_2 , $(\mathbf{q}_1 \mathbf{q}_2)^* = \mathbf{q}_2^* \mathbf{q}_1^*$. Third, conjugating a quaternion twice yields the original quaternion, that is, $(\mathbf{q}^*)^* = \mathbf{q}$. Substituting $\mathbf{q}_{n,\text{true}}^*$ for \mathbf{q} and $-\boldsymbol{\omega}_{n,\text{true}}$ for $\boldsymbol{\omega}$ in (19.122) yields

$$\dot{\mathbf{q}}_{n,\text{true}}^* = -\frac{1}{2} \boldsymbol{\omega}_{n,\text{true}} \mathbf{q}_{n,\text{true}}^* \quad (19.123)$$

Solving for $\omega_{n,\text{true}}$ we have

$$\begin{aligned}\omega_{n,\text{true}} &= -2\dot{\mathbf{q}}_{n,\text{true}}^* (\mathbf{q}_{n,\text{true}}^*)^{-1} \\ &= -2\dot{\mathbf{q}}_{n,\text{true}}^* \frac{(\mathbf{q}_{n,\text{true}}^*)^*}{\|\mathbf{q}_{n,\text{true}}\|} \\ &= -2\dot{\mathbf{q}}_{n,\text{true}}^* \mathbf{q}_{n,\text{true}}\end{aligned}\quad (19.124)$$

where we have used the fact that $\mathbf{q}_{n,\text{true}}$ is a unit quaternion.

Now, taking the conjugate of both sides of (19.124) yields

$$\omega_{n,\text{true}}^* = -\omega_{n,\text{true}} = -2(\dot{\mathbf{q}}_{n,\text{true}}^* \mathbf{q}_{n,\text{true}})^* = -2\mathbf{q}_{n,\text{true}}^* \dot{\mathbf{q}}_{n,\text{true}} \quad (19.125)$$

so that

$$\omega_{n,\text{true}} = 2\mathbf{q}_{n,\text{true}}^* \dot{\mathbf{q}}_{n,\text{true}} \quad (19.126)$$

To evaluate $\omega_{n,\text{true}}$ we must find $\dot{\mathbf{q}}_{n,\text{true}}$. To do so, differentiate (19.120) using the product rule

$$\begin{aligned}\dot{\mathbf{q}}_{n,\text{true}} &= \frac{d}{dt} (\mathbf{q}_\varphi \mathbf{q}_\theta \mathbf{q}_\phi) \\ &= \dot{\mathbf{q}}_\varphi \mathbf{q}_\theta \mathbf{q}_\phi + \mathbf{q}_\varphi \dot{\mathbf{q}}_\theta \mathbf{q}_\phi + \mathbf{q}_\varphi \mathbf{q}_\theta \dot{\mathbf{q}}_\phi\end{aligned}\quad (19.127)$$

where $\dot{\mathbf{q}}_\phi$, $\dot{\mathbf{q}}_\theta$, and $\dot{\mathbf{q}}_\varphi$ are obtained by differentiating (19.117) through (19.119) resulting in

$$\dot{\mathbf{q}}_\phi = \frac{\dot{\phi}}{2} \left[-\sin \frac{\phi}{2}, \cos \frac{\phi}{2}, 0, 0 \right]^\top \quad (19.128)$$

$$\dot{\mathbf{q}}_\theta = \frac{\dot{\theta}}{2} \left[-\sin \frac{\theta}{2}, 0, \cos \frac{\theta}{2}, 0 \right]^\top \quad (19.129)$$

$$\dot{\mathbf{q}}_\varphi = \frac{\dot{\varphi}}{2} \left[-\sin \frac{\varphi}{2}, 0, 0, \cos \frac{\varphi}{2} \right]^\top \quad (19.130)$$

Inserting (19.127) into (19.126) yields

$$\omega_{n,\text{true}} = 2\mathbf{q}_{n,\text{true}}^* (\dot{\mathbf{q}}_\varphi \mathbf{q}_\theta \mathbf{q}_\phi + \mathbf{q}_\varphi \dot{\mathbf{q}}_\theta \mathbf{q}_\phi + \mathbf{q}_\varphi \mathbf{q}_\theta \dot{\mathbf{q}}_\phi) \quad (19.131)$$

Now, $\dot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})}$ can be written as

$$\dot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})} = [\omega_{x,n,\text{true}}, \omega_{y,n,\text{true}}, \omega_{z,n,\text{true}}]^\top \quad (19.132)$$

In a similar manner, to generate $\ddot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})}$ we must find $\dot{\boldsymbol{\omega}}_{n,\text{true}}$ by differentiating (19.126)

$$\begin{aligned}\dot{\boldsymbol{\omega}}_{n,\text{true}} &= \frac{d}{dt} \boldsymbol{\omega}_{n,\text{true}} = \frac{d}{dt} \left(2\mathbf{q}_{n,\text{true}}^* \frac{d\mathbf{q}_{n,\text{true}}}{dt} \right) \\ &= 2 \left[\frac{d\mathbf{q}_{n,\text{true}}^*}{dt} \frac{d\mathbf{q}_{n,\text{true}}}{dt} + \mathbf{q}_{n,\text{true}}^* \frac{d^2\mathbf{q}_{n,\text{true}}}{dt^2} \right]\end{aligned}\quad (19.133)$$

To evaluate $d\mathbf{q}_{n,\text{true}}^*/dt$ we observe that for all \mathbf{q} , $d(\mathbf{q}\mathbf{q}^*)/dt = 0$. Thus

$$\frac{d}{dt} (\mathbf{q}^* \mathbf{q}) = \frac{d\mathbf{q}^*}{dt} \mathbf{q} + \mathbf{q}^* \frac{d\mathbf{q}}{dt} = 0 \quad (19.134)$$

which leads to

$$\frac{d\mathbf{q}^*}{dt} = -\mathbf{q}^* \frac{d\mathbf{q}}{dt} \mathbf{q}^* = -\mathbf{q}^* \dot{\mathbf{q}} \mathbf{q}^* \quad (19.135)$$

Using (19.135) in (19.133) yields

$$\begin{aligned}\dot{\boldsymbol{\omega}}_{n,\text{true}} &= 2 \left[-\mathbf{q}_{n,\text{true}}^* \dot{\mathbf{q}}_{n,\text{true}} \mathbf{q}_{n,\text{true}}^* \dot{\mathbf{q}}_{n,\text{true}} + \mathbf{q}_{n,\text{true}}^* \ddot{\mathbf{q}}_{n,\text{true}} \right] \\ &= 2\mathbf{q}_{n,\text{true}}^* \ddot{\mathbf{q}}_{n,\text{true}} - \boldsymbol{\omega}_{n,\text{true}}^2\end{aligned}\quad (19.136)$$

To obtain $\ddot{\mathbf{q}}_{n,\text{true}}$ we differentiate (19.127) resulting in

$$\begin{aligned}\ddot{\mathbf{q}}_{n,\text{true}} &= \ddot{\mathbf{q}}_\phi \mathbf{q}_\theta \mathbf{q}_\phi + \mathbf{q}_\phi \ddot{\mathbf{q}}_\theta \mathbf{q}_\phi + \mathbf{q}_\phi \mathbf{q}_\theta \ddot{\mathbf{q}}_\phi \\ &\quad + 2\dot{\mathbf{q}}_\phi \dot{\mathbf{q}}_\theta \mathbf{q}_\phi + 2\dot{\mathbf{q}}_\phi \mathbf{q}_\theta \dot{\mathbf{q}}_\phi + 2\mathbf{q}_\phi \dot{\mathbf{q}}_\theta \dot{\mathbf{q}}_\phi\end{aligned}\quad (19.137)$$

where $\ddot{\mathbf{q}}_\phi$, $\ddot{\mathbf{q}}_\theta$, and $\ddot{\mathbf{q}}_\phi$ are obtained by differentiating (19.128) through (19.130) resulting in

$$\ddot{\mathbf{q}}_\phi = \frac{\ddot{\phi}}{2} \left[-\sin \frac{\phi}{2}, \cos \frac{\phi}{2}, 0, 0 \right]^T - \dot{\phi} \mathbf{q}_\phi \quad (19.138)$$

$$\ddot{\mathbf{q}}_\theta = \frac{\ddot{\theta}}{2} \left[-\sin \frac{\theta}{2}, 0, \cos \frac{\theta}{2}, 0 \right]^T - \dot{\theta} \mathbf{q}_\theta \quad (19.139)$$

$$\ddot{\mathbf{q}}_\varphi = \frac{\ddot{\varphi}}{2} \left[-\sin \frac{\varphi}{2}, 0, 0, \cos \frac{\varphi}{2} \right]^T - \dot{\varphi} \mathbf{q}_\varphi \quad (19.140)$$

Now, using (19.137) in (19.136) yields

$$\begin{aligned}\dot{\boldsymbol{\omega}}_{n,\text{true}} &= 2\mathbf{q}_{n,\text{true}}^* \left[(\ddot{\mathbf{q}}_\phi \mathbf{q}_\theta \mathbf{q}_\phi + \mathbf{q}_\phi \ddot{\mathbf{q}}_\theta \mathbf{q}_\phi + \mathbf{q}_\phi \mathbf{q}_\theta \ddot{\mathbf{q}}_\phi + 2\dot{\mathbf{q}}_\phi \dot{\mathbf{q}}_\theta \mathbf{q}_\phi \right. \\ &\quad \left. + 2\dot{\mathbf{q}}_\phi \mathbf{q}_\theta \dot{\mathbf{q}}_\phi + 2\mathbf{q}_\phi \dot{\mathbf{q}}_\theta \dot{\mathbf{q}}_\phi) - \boldsymbol{\omega}_{n,\text{true}}^2 \right]\end{aligned}\quad (19.141)$$

and $\ddot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})}$ can be written as

$$\ddot{\mathbf{a}}_{n,\text{true}}^{(\text{Body})} = [\dot{\omega}_{x,n,\text{true}}, \dot{\omega}_{y,n,\text{true}}, \dot{\omega}_{z,n,\text{true}}]^T \quad (19.142)$$

19.5.3 Synthetic Cameras

To generate synthetic data, we assume that there are only two synthetic cameras by setting $J = 2$ and giving them the following positions, orientations (in degrees), and focal lengths (in pixels)

$$(x_{C_1}, y_{C_1}, z_{C_1}) = (490, 100, 0) \quad (19.143)$$

$$(x_{C_2}, y_{C_2}, z_{C_2}) = (-490, 100, 0) \quad (19.144)$$

$$(\phi_{C_1}, \theta_{C_1}, \varphi_{C_1}, f_{C_1}) = (0, 0, -168, 2500) \quad (19.145)$$

$$(\phi_{C_2}, \theta_{C_2}, \varphi_{C_2}, f_{C_2}) = (0, 0, -12, 2500) \quad (19.146)$$

and then we set \mathbf{p}_{C_1} , \mathbf{q}_{C_1} , \mathbf{p}_{C_2} , \mathbf{q}_{C_2} as

$$\mathbf{p}_{C_1} = [x_{C_1}, y_{C_1}, z_{C_1}]^\top \quad (19.147)$$

$$\mathbf{q}_{C_1} = Q_{\text{Euler}}(\phi_{C_1}, \theta_{C_1}, \varphi_{C_1}) \quad (19.148)$$

$$\mathbf{p}_{C_2} = [x_{C_2}, y_{C_2}, z_{C_2}]^\top \quad (19.149)$$

$$\mathbf{q}_{C_2} = Q_{\text{Euler}}(\phi_{C_2}, \theta_{C_2}, \varphi_{C_2}) \quad (19.150)$$

19.5.4 Synthetic Measurements

Having constructed our synthetic rigid body in (19.100), we must specify which feature points are visible at each point in time by setting $K_{j,n}$ and $k_{i,j,n}$ for all i , j , and n . For simplicity, all feature points are visible to both cameras throughout the trajectory. Explicitly, $K_{j,n} = 8$ for $j = \{1, 2\}$ and $n = \{0, 1, \dots, 200\}$ and $k_{i,j,n} = i$ for $i = \{1, 2, \dots, 8\}$, $j = \{1, 2\}$, and $n = \{0, 1, \dots, 200\}$. Now we have expressions for every term necessary for construction of the measurement function $\mathbf{h}(\mathbf{x})$ as described in Section 3 and we use $\mathbf{h}(\mathbf{x})$ to generate the synthetic measurements. For each $n = \{0, 1, \dots, 200\}$, the measurement vector $\mathbf{z}_{n,\text{synth}}^o$ is generated by evaluating

$$\mathbf{z}_{n,\text{synth}}^o = \mathbf{h}_n(\mathbf{x}_{n,\text{true}}) + \mathbf{w}_{n,\text{synth}} \quad (19.151)$$

where

$$\mathbf{w}_{n,\text{synth}} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{\text{synth}}) \quad (19.152)$$

with

$$\mathbf{R}_{n,\text{synth}} = \sigma_{\text{pix}}^2 \mathbf{I}_{2M_n} = \sigma_{\text{pix}}^2 \mathbf{I}_{32} \quad (19.153)$$

Here (since $M_n = 16 = \sum_{j=1}^2 K_{j,n}$), \mathbf{I}_{32} is a 32×32 identity matrix and σ_{pix} is set to 2 pixels. Note that a new Monte Carlo sample is drawn from $\mathcal{N}(\mathbf{0}, \mathbf{R}_{\text{synth}})$ for each time step.

19.6 PERFORMANCE COMPARISON ANALYSIS

The performance analysis of all estimation filters presented in Section 19.4 was conducted using the synthetic test data described in Section 19.5. The performance of the UKF using zero-, first-, second-, and third-order dynamic models is compared against the NLLSQ and also against the UGPF using the second-order dynamic model. The performance metric used in all cases is the root mean squared (RMS) error. The performance results to be presented below indicate that the second-order UKF was the best filter to use.

Figures 19.4 through 19.6 show the results the estimated track results of a single run of the UKF filter using the second -order model compared against synthetic truth data. Observe that the *rotational* rates and accelerations shown in Figures 19.5 and 19.6 are in body-referenced coordinates. Figure 19.7 shows a close-up of a small section of the lateral position truth trajectory with the estimated trajectories from multiple solvers superimposed over the truth. Identical noisy observations and initialization were used as input to all estimation methods. Analysis of this data leads to two conclusions: first, we note that the NLLSQ and the UKF with a zeroth-order dynamic model generate nearly identical estimates. Since the zeroth-order model is a constant position model and the NLLSQ assumes a position-only state vector, the similarity

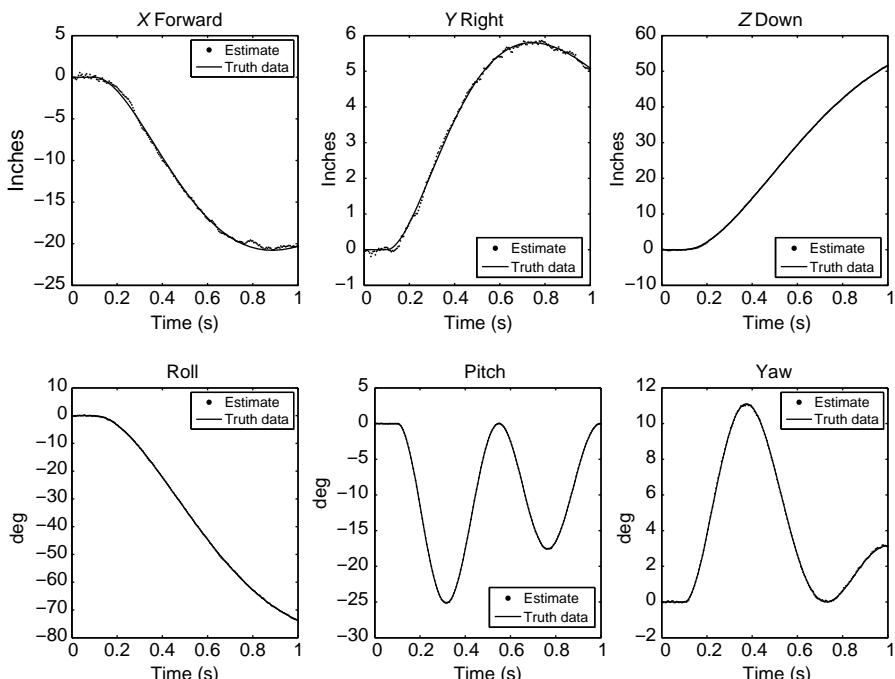


FIGURE 19.4 Translational and rotational position using a second-order model with a UKF filter.

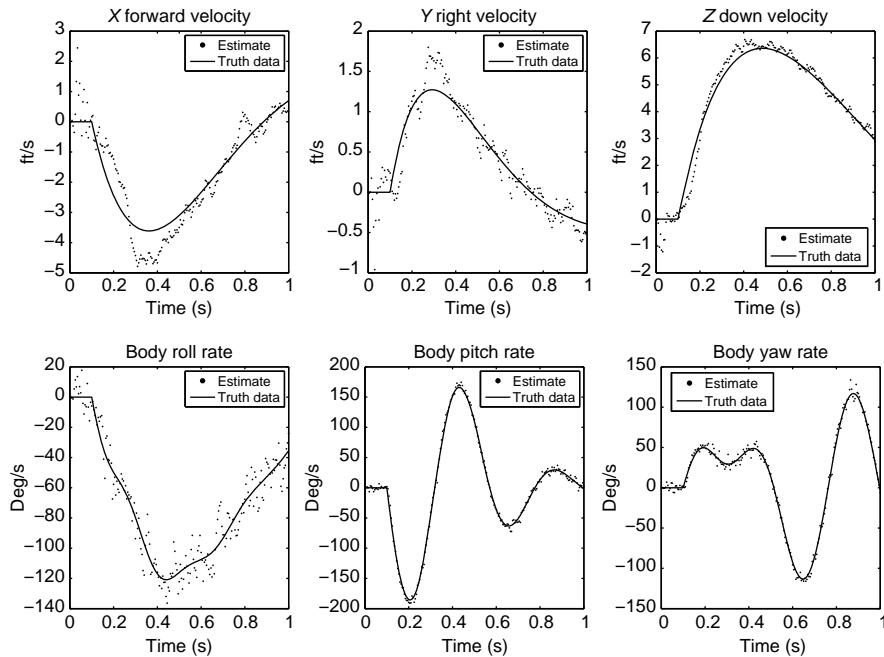


FIGURE 19.5 Translational and rotational velocity using a second-order model with a UKF filter.

in performance of the NLLSQ and UKF(zeroth order) is to be expected. The second conclusion from Figure 19.7 is that the filters that use higher order dynamic models have better performance than the NLLSQ or filtering using a zeroth-order model, primarily due to a reduction of the variation of their estimates about the truth trajectory. This preliminary conclusion will be born out in Section 19.5.2 where we present the RMS error comparisons for the various filters and models.

19.6.1 Filter Performance Comparison Methodology

The performance comparison methodology begins with a single noiseless state vector trajectory $\{\mathbf{x}_n, n = 0, 1, \dots, N\}$ generated synthetically as described in Section 19.5.2. This trajectory is then transformed into a set of noiseless synthetic truth observations $\{\mathbf{h}_n(\mathbf{x}_n), n = 0, 1, \dots, N\}$. Finally, $N_{\text{RMS}} = 100$ sets of noise observations are produced by adding independent zero-mean Gaussian noise to the synthetic truth observations to produce N_{RMS} sets of noisy observations $\{\mathbf{z}_{n,r}^0; n = 0, 1, \dots, N; r = 1, 2, \dots, N_{\text{RMS}}\}$. Each noisy observation set is used in each tracking filter to produce N_{RMS} sets of state vector estimates $\{\hat{\mathbf{x}}_{n|r}; n = 0, 1, \dots, N; r = 1, 2, \dots, N_{\text{RMS}}\}$.

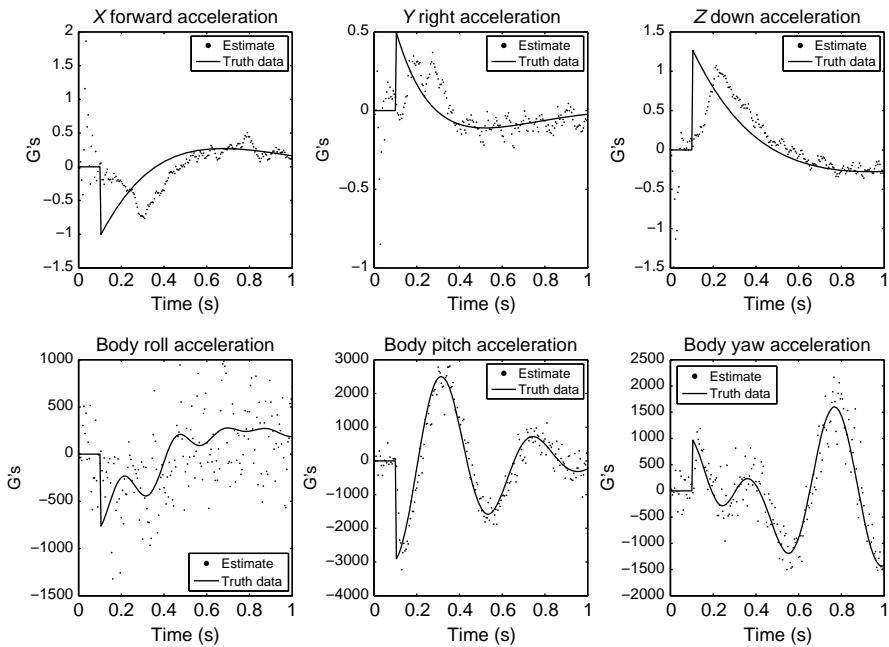


FIGURE 19.6 Translational and rotational acceleration using a second-order model with a UKF filter.

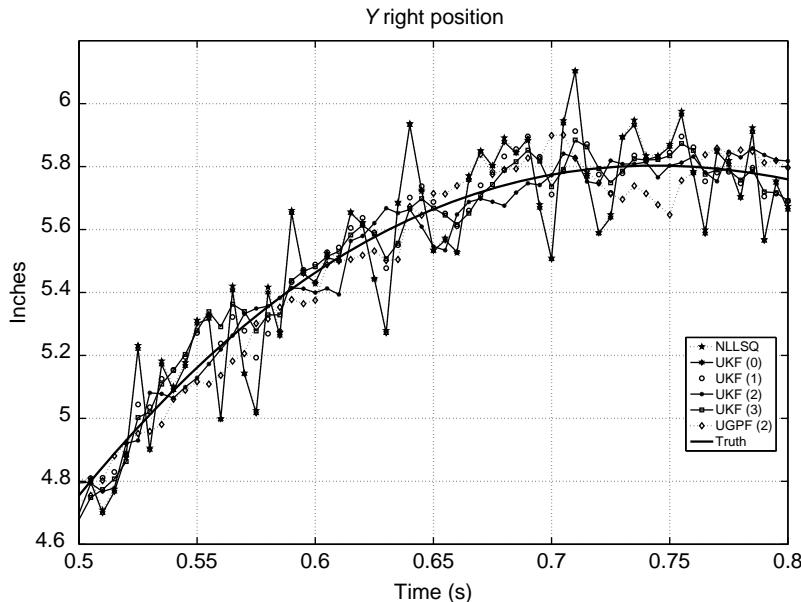


FIGURE 19.7 Lateral position estimates from multiple solvers using identical inputs.

As noted in Chapter 14, at time t_n , the covariance matrix of the full state vector estimate $\hat{\mathbf{x}}_{n|n}$ can be written as

$$\mathbf{P}_{n|n}^{\mathbf{XX}} = \mathcal{E} \left\{ (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n}) (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top \right\} \quad (19.154)$$

When \mathbf{x}_n is known, as it would be for our synthetic trajectory case, and $\hat{\mathbf{x}}_{n|n}$ is an unbiased estimate of \mathbf{x}_n , the square root of the diagonal terms of $\mathbf{P}_{n|n}^{\mathbf{XX}}$ can be approximated by their conditional RMS errors obtained from repeated filter runs using a different Monte Carlo set of noisy observation data for each run.

For example, the RMS error for the lateral position (y) estimate of a given filter at time t_n is given by

$$e_{y,n}^{(\text{filter})} = \sqrt{\frac{\sum_{r=1}^{N_{\text{RMS}}} (y_n - \hat{y}_{n|n,r}^{(\text{filter})})^2}{N_{\text{RMS}}}} \quad (19.155)$$

Since we intend to compare results between zero-, first-, and second-order terms obtained by each filter for our six-dimensional problem, we would need to calculate and analyze the RMS errors for 18 parameters. To simplify our analysis, we instead compare all three components of position together using the squared distance

$$\left\| \mathbf{p}_n - \mathbf{p}_{n|n,r}^{(\text{filter})} \right\|^2 = (x_n - \hat{x}_{n|n,r}^{(\text{filter})})^2 + (y_n - \hat{y}_{n|n,r}^{(\text{filter})})^2 + (z_n - \hat{z}_{n|n,r}^{(\text{filter})})^2 \quad (19.156)$$

This results in just six performance comparisons, one for each 3D position and its first two derivatives, and one for each 3D rotation and its first two derivatives.

For the three translational parameters comprising the position at time t_n , we compute the RMS position error as

$$e_{\mathbf{p},n}^{(\text{filter})} = \sqrt{\frac{\sum_{r=1}^{N_{\text{RMS}}} \left\| \mathbf{p}_n - \mathbf{p}_{n|n,r}^{(\text{filter})} \right\|^2}{N_{\text{RMS}}}} \quad (19.157)$$

For the first and second derivatives of position, we compute $e_{\dot{\mathbf{p}},n}^{(\text{filter})}$ and $e_{\ddot{\mathbf{p}},n}^{(\text{filter})}$ in a similar fashion.

The angular displacement RMS errors cannot be computed in the same way since a 3D angle is not a vector quantity. To find the difference between two orientations represented by the unit quaternions \mathbf{q}_1 and \mathbf{q}_2 we write

$$\mathbf{q}_\Delta = \mathbf{q}_2^* \mathbf{q}_1 \quad (19.158)$$

To calculate the magnitude of the difference, \mathbf{q}_Δ must be converted to its axis-angle representation before its norm can be computed. Thus, we can write the RMS angular error for a specific filter at time t_n as

$$e_{\mathbf{a},n}^{(\text{filter})} = \sqrt{\frac{\sum_{r=1}^{N_{\text{RMS}}} \left\| \mathcal{A}_{\mathbf{q}} (\mathbf{q}_n^* \mathbf{q}_{n|n,r}^{(\text{filter})}) \right\|^2}{N_{\text{RMS}}}} \quad (19.159)$$

where $\mathcal{A}_q : \mathbf{H} \rightarrow \mathbb{R}^3$ (see (19.163)) converts a unit quaternion into an axis-angle vector, and $\mathbf{q}_{n|n,r}^{(\text{filter})}$ is the quaternion representing the orientation estimate given by a specific filter at time t_n for run r and \mathbf{q}_n represents the true orientation at time t_n .

Since the rotation rate $\dot{\mathbf{a}}$ is a vector, we can compare $\dot{\mathbf{a}}_{n|n,r}^{(\text{filter})}$ with $\dot{\mathbf{a}}_n$ directly. Observe that the filters estimate $\dot{\mathbf{a}}$ in reference frame coordinates while the truth trajectory for $\dot{\mathbf{a}}$ is in body-referenced coordinates. Thus, a frame rotation (see Appendix 19.A.3) must be used to transform $\dot{\mathbf{a}}_{n|n,r}^{(\text{filter})}$ into body coordinates. Thus, the RMS angular rate error for a particular filter at time t_n is written as

$$e_{\dot{\mathbf{a}},n}^{(\text{filter})} = \sqrt{\frac{\sum_{r=1}^{N_{\text{RMS}}} \left\| \mathbf{M}_{n,r}^T \dot{\mathbf{a}}_{n|n,r}^{(\text{filter})} - \dot{\mathbf{a}}_n \right\|^2}{N_{\text{RMS}}}} \quad (19.160)$$

where $\mathbf{M}_{n,r} = \mathcal{M}_q(\mathbf{q}_{n|n,r}^{(\text{filter})})$ is the matrix representing the estimated orientation for the specific filter at time t_n during run r . $e_{\ddot{\mathbf{a}},n}^{(\text{filter})}$ can be computed in the same manner.

19.6.2 Filter Comparison Results

For each filter and model order, the RMS errors are computed as a function of time and then the mean and maximum errors (over time) are generated for each case. These mean and maximum values are summarized in Tables 19.1, 19.2, and 19.3, for the position, velocity, and acceleration RMS errors, respectively.

TABLE 19.1 Synthetic Data RMS Positional Error Summary

Filter Type	Model Order	Position (in.)		Orientation (deg)	
		Mean	Max	Mean	Max
NLLSQ	0	0.68	0.81	0.22	0.26
UKF	0	0.61	0.72	0.20	0.24
UKF	1	0.30	0.43	0.15	0.22
UKF	2	0.32	0.52	0.15	0.22
UKF	3	0.32	0.47	0.14	0.23
UGPF	2	0.32	0.48	0.15	0.22

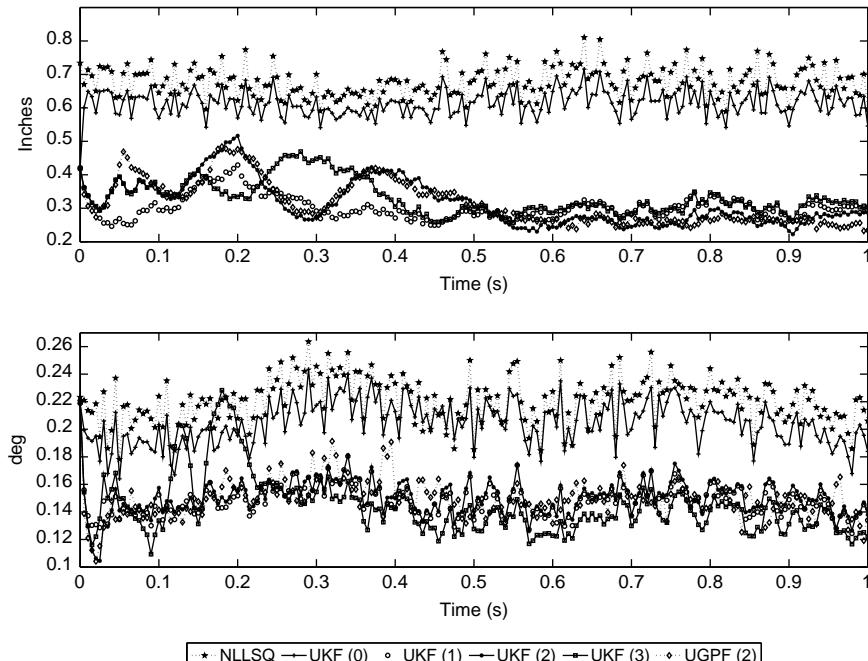
TABLE 19.2 Synthetic Data RMS Velocity Error Summary

Filter Type	Model Order	Translational Velocity (ft/s)		Angular Velocity (deg/s)	
		Mean	Max	Mean	Max
NLLSQ	0	N/A	N/A	N/A	N/A
UKF	0	N/A	N/A	N/A	N/A
UKF	1	0.74	1.43	14.04	22.76
UKF	2	0.71	1.63	12.34	22.18
UKF	3	0.89	1.78	10.56	34.37
UGPF	2	0.72	1.67	12.30	20.39

TABLE 19.3 Synthetic Data RMS Acceleration Error Summary

Filter Type	Model Order	Translational Acceleration (G's)		Angular Acceleration (deg/s ²)	
		Mean	Max	Mean	Max
NLLSQ	0	N/A	N/A	N/A	N/A
UKF	0	N/A	N/A	N/A	N/A
UKF	1	N/A	N/A	N/A	N/A
UKF	2	0.37	1.70	653.3	3005.9
UKF	3	0.57	1.77	563.9	3085.0
UGPF	2	0.38	1.71	652.6	2990.2

Figures 19.8, 19.9, and 19.10 show time histories of the performance of each filter/model order combination for positional and rotational RMS errors, position and rotation velocity RMS errors, and position and rotation accelerations, respectively (top plots). It is obvious from Tables 19.1 through 19.3 and Figures 19.8 through 19.10 that all of the filters that use a model order greater than zero have an average performance that is at least 50% better in position and 30% better in orientation. Because of the large acceleration transient at the time of store release at 0.1 s, the RMS error plots show the poorest performance for 200–300 ms after the release event.

**FIGURE 19.8** RMS errors for position and orientation of all tracking filters using synthetic data.

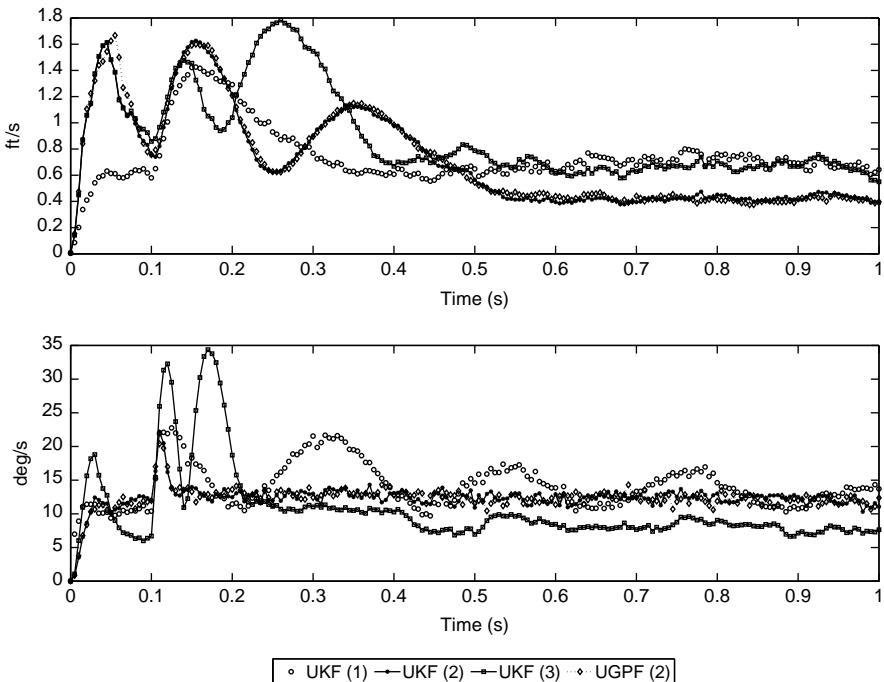


FIGURE 19.9 RMS errors for position and orientation velocities of all tracking filters using synthetic data.

The second-order unscented Gaussian particle filter (UGPF(2)) and the second-order unscented Kalman filter (UKF(2)) have nearly equal performance over all six parameter types. Since a Gaussian random error was used for the synthetic measurements, the near equal performance of UGPF(2) and UKF(2) indicates that the degree of nonlinearity in either the dynamic or measurement models is insufficient to warp the probability density of the state vector. Since the state vector density always remains Gaussian, the RMS errors for the UGPF(2) are of the same order of magnitude as those of the UKF(2). We therefore judge that the considerable computational expense related to the use of a particle filter is unnecessary in our problem space. We might reconsider this judgment if we encounter non-Gaussian measurement noise of sufficient severity in real-world use.

The third-order unscented Kalman filter (UKF(3)) has significant performance problems in comparison to UKF(1) and UKF(2). Although the mean RMS error of UKF(3) is significantly better for some parameter types, in some cases it performs much more poorly than either UKF(1) or UKF(2). The problem with the performance of our current third-order dynamic model merits further investigation.

The UKF(2) solver appears to have marginally better performance than UKF(1). RMS statistics are roughly equivalent for position, angle, and translational velocity.

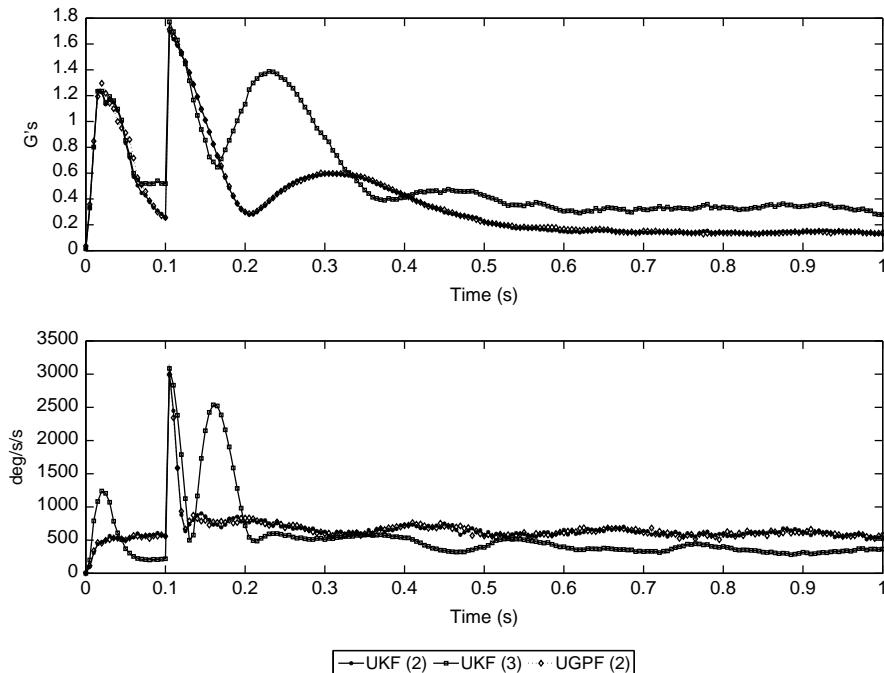


FIGURE 19.10 RMS errors for position and orientation accelerations of all tracking filters using synthetic data.

In the angular velocity time history, RMS error of the UKF(2) solver stabilizes after about 100 ms, while the UKF(1) solver continues to oscillate.

19.6.3 Conclusions and Future Considerations

The second-order unscented Kalman filter (UKF(2)) achieves the best performance of all of the solvers tested, and provides a marked improvement over the standard zeroth-order NLLSQ solver. Although the UGPF(2) particle filter has equivalent performance, its considerable computational expense makes it unnecessary.

All of the solvers tested whose order was greater than zero experienced degradation in performance immediately after the large jump in acceleration caused by the release event impulse. Accounting for this impulse in the dynamic model should significantly improve filter performance. The free-fall equations in our dynamic model are not sufficient to account for changes in the process (such as a forcing function) that are not attributable to noise.

An alternative approach to this degradation problem is to directly measure the accelerations causing the performance degradation. This is discussed in the next chapter, where we combine image measurements with accelerometer and rate gyroscope measurements.

Another approach we intend to investigate is using not only the 2D locations of feature points in images, but also the 2D image velocity of feature points measured from image sequences using techniques such as optical flow. Adding feature point image velocity to the measurement model should help to stabilize and increase the accuracy of the 3D translational and angular velocity estimates generated by the Bayesian filter.

APPENDIX 19.A QUATERNIONS, AXIS-ANGLE VECTORS, AND ROTATIONS

19.A.1 Conversions Between Rotation Representations

Let $\mathbf{a} = [a_x, a_y, a_z]^\top \in \mathbb{R}^3$ be used as an *axis-angle representation* of rotation where the direction of \mathbf{a} specifies the axis of rotation and $\|\mathbf{a}\|$ specifies the magnitude of rotation (in radians).

Let $\mathbf{q} = [q_s, q_x, q_y, q_z]^\top \in \mathbf{H}$, where \mathbf{H} is the space of quaternions, be used as a *unit quaternion representation* of rotation, where q_s is real and q_x, q_y, q_z are coefficients of distinct imaginary numbers, and $\sqrt{q_s^2 + q_x^2 + q_y^2 + q_z^2} = 1$.

Let $\mathbf{M} \in \mathbb{R}^3 \times \mathbb{R}^3$ be a 3×3 *rotation matrix*, such that \mathbf{M} is orthonormal, and $\det \mathbf{M} = +1$.

To convert from an axis-angle vector \mathbf{a} to a unit quaternion \mathbf{q} , we define the function $\mathcal{Q}_{\mathbf{a}} : \mathbb{R}^3 \rightarrow \mathbf{H}$ as

$$\mathbf{q} = \mathcal{Q}_{\mathbf{a}}(\mathbf{a}) = \left[\cos \frac{\|\mathbf{a}\|}{2}, \frac{\sin \frac{\|\mathbf{a}\|}{2}}{\|\mathbf{a}\|} [a_x, a_y, a_z]^\top \right]^\top \quad (19.161)$$

To convert from a unit quaternion \mathbf{q} to an axis-angle vector \mathbf{a} we define the function $\mathcal{A}_{\mathbf{q}} : \mathbf{H} \rightarrow \mathbb{R}^3$ as

$$\mathbf{a} = \mathcal{A}_{\mathbf{q}}(\mathbf{q}) = \frac{2 \cos^{-1} q_s}{\sqrt{1 - q_s^2}} [q_x, q_y, q_z]^\top \quad (19.162)$$

To convert from a unit quaternion \mathbf{q} to a rotation matrix \mathbf{M} , we define the function $\mathcal{M}_{\mathbf{q}} : \mathbf{H} \rightarrow \mathbb{R}^3 \times \mathbb{R}^3$ as

$$\mathbf{M} = \mathcal{M}_{\mathbf{q}}(\mathbf{q}) = \begin{bmatrix} 2(q_s^2 + q_x^2) - 1 & 2(q_x q_y - q_s q_z) & 2(q_x q_z + q_s q_y) \\ 2(q_x q_y + q_s q_z) & 2(q_s^2 + q_y^2) - 1 & 2(q_y q_z - q_s q_x) \\ 2(q_x q_z - q_s q_y) & 2(q_y q_z + q_s q_x) & 2(q_s^2 + q_z^2) - 1 \end{bmatrix} \quad (19.163)$$

To convert from an axis-angle vector \mathbf{a} to a rotation matrix \mathbf{M} , we derive the function $\mathcal{M}_{\mathbf{a}} : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \times \mathbb{R}^3$ as

$$\mathbf{M} = \mathcal{M}_{\mathbf{a}}(\mathbf{a}) = \mathcal{M}_{\mathbf{q}}(\mathcal{Q}_{\mathbf{a}}(\mathbf{a})) \quad (19.164)$$

To convert the Euler angles (ϕ, θ, φ) into a unit quaternion \mathbf{q} , we define the function $\mathcal{Q}_{\text{Euler}}(\phi, \theta, \varphi) : \mathbb{R}^3 \rightarrow$

$$\mathcal{Q}_{\text{Euler}}(\phi, \theta, \varphi) = \mathbf{q}_\varphi \mathbf{q}_\theta \mathbf{q}_\phi \quad (19.165)$$

where \mathbf{q}_ϕ , \mathbf{q}_θ , and \mathbf{q}_φ are the quaternions that represent each of the three Euler angles by themselves

$$\mathbf{q}_\phi = \left[\cos \frac{\phi}{2}, \sin \frac{\phi}{2}, 0, 0 \right]^\top \quad (19.166)$$

$$\mathbf{q}_\theta = \left[\cos \frac{\theta}{2}, 0, \sin \frac{\theta}{2}, 0 \right]^\top \quad (19.167)$$

$$\mathbf{q}_\varphi = \left[\cos \frac{\varphi}{2}, 0, 0, \sin \frac{\varphi}{2} \right]^\top \quad (19.168)$$

19.A.2 Representation of Orientation and Rotation

Rotation is represented by two quantities together, a 3D axis-angle vector \mathbf{a} that resides in the state vector, and an *external* unit quaternion \mathbf{q} that resides outside the state vector and is only used in the observation model. During an iteration of the filter, the quaternion \mathbf{q} represents the prior orientation of the rigid body relative to the reference coordinate system. The vector \mathbf{a} represents an adjustment to \mathbf{q} and \mathbf{a} by itself only partially represents the current orientation. When derivatives of rotation are present, they are represented in the state vector by the 3D vectors $\dot{\mathbf{a}}$, $\ddot{\mathbf{a}}$, $\dddot{\mathbf{a}}$, ...

The quaternion multiplication of two quaternions \mathbf{p} and \mathbf{q} resulting in the quaternion \mathbf{r} is represented by the operation

$$\mathbf{r} = \mathbf{pq} \quad (19.169)$$

or

$$\mathbf{r} = \begin{bmatrix} r_s \\ r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} p_s & -p_x & -p_y & -p_z \\ p_x & p_s & -p_z & p_y \\ p_y & p_z & p_s & -p_x \\ p_z & -p_y & p_x & p_s \end{bmatrix} \begin{bmatrix} q_s \\ q_x \\ q_y \\ q_z \end{bmatrix} \quad (19.170)$$

which can be written as the matrix vector multiplication

$$\mathbf{r} = \mathbf{Pq} \quad (19.171)$$

The external unit quaternion is not altered during a filter iteration and is only used in the observation model. Once the filter update step has completed, the corrected axis-angle portion of the state vector $\mathbf{a}_{n|n}$ is applied to update the external unit quaternion, after which $\mathbf{a}_{n|n}$ is reset

$$\mathbf{q}_n = \mathcal{Q}_{\mathbf{a}}(\mathbf{a}_{n|n}) \mathbf{q}_{n-1} \quad (19.172)$$

$$\mathbf{a}_{n|n} \leftarrow [0, 0, 0]^\top \quad (19.173)$$

19.A.3 Point Rotations and Frame Rotations

We draw a distinction between point rotations and frame rotations as follows. In general terms, when a point rotation is performed, the point or vector is rotated while the axis directions remains constant. With a frame rotation, the axes rotate while the point or vector remains fixed. To perform a point rotation of a 3D vector \mathbf{v} by an angular displacement specified by a unit quaternion \mathbf{q} , we construct a rotation matrix using the function $\mathcal{M}_\mathbf{q}$ specified in (19.164)

$$\mathbf{M}_\mathbf{q} = \mathcal{M}_\mathbf{q}(\mathbf{q}) \quad (19.174)$$

then we multiple to produce the rotated vector $\mathbf{v}^{(\text{rot})}$

$$\mathbf{v}^{(\text{rot})} = \mathbf{M}_\mathbf{q} \mathbf{v} \quad (19.175)$$

Now let $\mathbf{v}^{(f1)}$ be a vector whose components are specified with respect to coordinate frame $f1$, and let $\mathbf{q}_{f1 \rightarrow f2}$ specify the angular displacement between the coordinate frames $f1$ and $f2$. To perform the frame rotation that finds the coordinates of $\mathbf{v}^{(f1)}$ with respect to $f2$, we perform

$$\mathbf{M}_{\mathbf{q}_{f1 \rightarrow f2}} = \mathcal{M}_\mathbf{q}(\mathbf{q}_{f1 \rightarrow f2}) \quad (19.176)$$

and

$$\mathbf{v}^{(f2)} = \mathbf{M}_{\mathbf{q}_{f1 \rightarrow f2}} \mathbf{v}^{(f1)} \quad (19.177)$$

Observe that a frame rotation is the inverse of a point rotation. Since rotation matrices are orthonormal, if \mathbf{M} is a rotation matrix, then $\mathbf{M}^{-1} = \mathbf{M}^\top$. Also observe that the origin of both $f1$ and $f2$ are coincident.

We use rotations in two ways in the observation model. We first find the coordinates of feature points on a rigid body after it has been rotated and translated to the body's estimated location in the reference (aircraft) coordinate system. Since the points on the body are being rotated and then translated into new locations, we perform a point transformation

$$\mathbf{s}^{(\text{ref})} = \mathbf{M}_{\text{ref}} \mathbf{s} + \mathbf{p} \quad (19.178)$$

where \mathbf{M}_{ref} is the rotation matrix derived from the estimated rotation angle of the body, the 3D vector \mathbf{p} specifies the translation of the body's center of mass, and \mathbf{s} specifies the location of a 3D feature point in the body's coordinate system.

We then calculate the location of the point $\mathbf{s}^{(\text{ref})}$ with respect to a camera's coordinate system using a frame transformation (a negative translation followed by a frame rotation)

$$\mathbf{s}^{(\text{cam})} = \mathbf{M}_{\text{cam}}^\top \left(\mathbf{s}^{(\text{ref})} - \mathbf{p}_{\text{cam}} \right) \quad (19.179)$$

where \mathbf{M}_{cam} describes the camera's orientation and \mathbf{p}_{cam} its position.

Observe that the inverse of a point transformation is a frame transformation. So to invert (19.178) we could perform

$$\mathbf{s} = \mathbf{M}_{\text{ref}}^T \left(\mathbf{s}^{(\text{ref})} - \mathbf{p} \right) \quad (19.180)$$

or to invert (19.179) we could perform

$$\mathbf{s}^{(\text{ref})} = \mathbf{M}_{\text{cam}} \mathbf{s}^{(\text{cam})} + \mathbf{p}_{\text{cam}} \quad (19.181)$$

REFERENCES

1. Huang TS, Netravali AN. Motion and structure from feature correspondences: a review. *Proc. IEEE* 1994;82(2): 252–268.
2. Iu SL, Wohn K. Estimation of General Rigid Body Motion from a Long Sequence of Images. Department of Computer & Information Science Technical Report, University of Pennsylvania; 1990.
3. Gennery DB. Visual tracking of known three-dimensional objects. *Int. J. Comput. Vision* 1992;7(3):243–270.
4. Welch GF. SCATT: Incremental Tracking with Incomplete Information. TR96051, Dissertation. Department of Computer Science, UNC-Chapel Hill; 1996.
5. Welch G, Bishop G. SCATT: incremental tracking with incomplete information. In *Proceedings of the ACM Siggraph 97*. New York: ACM Press; 1997, pp. 333–344.
6. Forseman E, Schug D. Improved analysis techniques for more efficient weapon separation testing. In *Proceedings of the Society of Flight Test Engineers European Chapter*; 2008.
7. Bar-Shalom Y, Li XR, Kirubarajan T. *Estimation with Applications to Tracking and Navigation*. Wiley; 2001.
8. Mehrotra K, Mahapatra PR. A jerk model for tracking highly maneuvering targets. *IEEE Trans. Aero. Electron. Syst.* 1997;33(4):1094–1105.
9. Online at ae-www.technion.ac.il/admin/serve.php?id=18558, accessed August 2011.

20

SENSOR FUSION USING PHOTOGRAMMETRIC AND INERTIAL MEASUREMENTS

20.1 INTRODUCTION

In Chapter 19, we presented a method for tracking the pose trajectory (position and orientation as a function of time) of a rigid falling body using only external video camera sensors. In this method, video images from multiple cameras are used to view feature points positioned on the surface of the rigid body. Thus, the measurement set consists only of feature point positions viewed by the cameras, with feature point occlusions creating data dropouts for certain rigid body orientations. In addition, no velocity or acceleration measurements are directly obtainable, so estimates of higher order derivative terms related to the position and orientation of the rigid body have reduced accuracy.

In this chapter, we examine the utility of adding additional body-mounted inertial sensors that include multiaxis accelerometers and gyroscopes which, when packaged together, are referred to as an inertial measurement unit (IMU). An IMU provides measurements of linear accelerations along the three body axes and angular velocities about the three body axes. This presents a requirement to fuse the temporal stream of IMU measurement data with the temporal stream of video image data to produce a tracking filter that utilizes both in an asynchronous measurement data stream.

Methods for tracking a rigid body using only successive temporal images of feature points from a single camera (monocular imaging) were first implemented in Refs [1], [2]. In the paper by Iu and Wohn [1], they implemented a dynamic model that assumed

a constant translational and rotational accelerations while Lee et al. [2] developed a dynamic model with constant translation and rotational velocities. Methods were advanced that used multiple camera images of feature points to calculate the pose of a rigid body at each time step in Refs [3–6]. Gennery [3], Huang and Natravali [4], and Ude [5] all utilized a nonlinear least squares (NLLSQ) method for state vector estimation that assumes no dynamic noise. This method is not recursive in time, but is instead iterative at each time step. Halvorsen et al. proposed using an extended Kalman Filter (EKF) that was recursive in time and therefore could handle missing feature points in the video images from multiple cameras.

In Refs [8–12], methods for estimating the *orientation* of a rigid body using either an IMU or a magnetic, angular rate, and gravity (MARG) sensor were proposed. In Ref. [8], Algrain and Samiee use an accelerometer gyro linear Gaussian (AGLG) estimation technique that is essentially a linear least-squares iterative method that is not time recursive. A MARG sensor is used in Refs [9–12] in conjunction with an EKF to track the orientation of a rigid body. These four papers present slight modifications to the original method and are recursive in time methods for tracking human body hand or limb orientations.

Fusion of video and IMU data was the focus of the articles [13–15]. In all cases the IMU is mounted next to the camera and the purpose of the fusion was for calibration of the camera. Since the IMU was not mounted external to the object being tracked, the IMU data was not used directly as measurement data in any of the object state estimation methods proposed in these papers.

In what follows we present an approach that includes an IMU mounted on the rigid body with external video cameras viewing feature points on the rigid body. Data from the rigid body mounted IMU is fused with the video image data to create a single stream of asynchronous data that is used in the update step of the tracking algorithm. In our approach, an unscented Kalman filter (UKF) is used as the fusion tracking algorithm and its performance results are compared to those obtained from using either the video image or the IMU data alone.

A picture that shows several views of the IMU alone or mounted on a store (bomb-like rigid body) is presented in Figure 20.1.

20.2 THE PROCESS (DYNAMIC) MODEL FOR RIGID BODY MOTION

The dynamic motion model for the motion of a rigid body was described in Section 2 of Chapter 19. We refer the reader to Chapter 19 for a full description of those dynamic models. At the end of Chapter 19 we showed that the best relative performance was achieved using the second order *constant acceleration* model, where we modified the state vector to include rate and acceleration vectors governing both translational and rotational motion. We will use that model exclusively for the analysis in this chapter. Our motion model describes the translational and rotational motion of a rigid body as a function of time in terms of an instantaneous state vector $\mathbf{x}(t) = [\mathbf{p}^T(t), \dot{\mathbf{p}}^T(t), \ddot{\mathbf{p}}^T(t), \mathbf{a}^T(t), \dot{\mathbf{a}}^T(t), \ddot{\mathbf{a}}^T]^T$ that, along with the external unit quaternion $\mathbf{q}(t)$, defines both the translation of the center of mass of a rigid body relative to a fixed



FIGURE 20.1 A free-standing IMU with transmitting antenna attached, and an IMU mounted in the forward fuse well of a 500 pound store.

reference Cartesian coordinate frame, $\mathbf{p}(t) = [x(t), y(t), z(t)]^\top$, and the orientation of the body about the center of mass as given by the combination of the axis-angle vector, $\mathbf{a}(t) = [a_x(t), a_y(t), a_z(t)]^\top$ and the external unit quaternion $\mathbf{q}(t) = [q_s(t), q_x(t), q_y(t), q_z(t)]^\top$.

In general, we can write the state vector temporal transition equation in the form

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{v}_{n-1} \quad (20.1)$$

where \mathbf{v}_{n-1} is a noise that accounts for the uncertainty in the process. In all cases we will take the noise to be Gaussian and additive.

20.3 THE SENSOR FUSION OBSERVATIONAL MODEL

The task is to define the functional dependence for the transformation of the state vector \mathbf{x}_n into the components of the sensor fusion observation vector \mathbf{z}_n . If the observation noise is included, this transformation can be written as

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_n) + \mathbf{w}_n \quad (20.2)$$

where $\mathbf{w}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_n)$. The observations consist of two separate components, the IMU components and the photogrammetric components, which must be fused together to provide an asynchronous stream of observation data.

20.3.1 The Inertial Measurement Unit Component of the Observation Model

The IMU package outputs data from three orthogonal accelerometers and three orthogonal rate gyroscopes. The IMU observation vector at time t_n can be written as

$$\mathbf{z}_{\text{IMU},n} = [z_{x,n}, z_{y,n}, z_{z,n}, z_{\phi,n}, z_{\theta,n}, z_{\varphi,n}]^\top \quad (20.3)$$

where $z_{x,n}$, $z_{y,n}$, and $z_{z,n}$ are the accelerations (in G 's) measured by the x -forward, y -right, and z -down accelerometers, respectively, and $z_{\phi,n}$, $z_{\theta,n}$ and $z_{\psi,n}$ are rotation rates (in degrees) measured by the roll (counterclockwise about the x -axis), pitch (counterclockwise about the y -axis), and yaw (counterclockwise about the z -axis) rate gyroscopes, respectively. Note that since the IMU package is attached to the rigid body, it rotates as the body rotates so that the measurements in $\mathbf{z}_{\text{IMU},n}$ are taken with respect to the body coordinate system and not with respect to the reference (aircraft) coordinate system.

We can now write the IMU observation vector in terms of the state vector by looking at the transformation equation

$$\mathbf{z}_{\text{IMU},n} = \mathbf{h}_{\text{IMU},n}(\mathbf{x}_n) + \mathbf{w}_{\text{IMU},n} \quad (20.4)$$

where $\mathbf{w}_{\text{IMU},n} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{\text{IMU},n})$, with $\mathbf{R}_{\text{IMU},n}$ defined as the diagonal noise covariance matrix of accelerometer and rate gyroscope measurement variances given by

$$\mathbf{R}_{\text{IMU},n} = \begin{bmatrix} \sigma_{\text{acc}}^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \sigma_{\text{gyro}}^2 \mathbf{I}_3 \end{bmatrix} \quad (20.5)$$

with σ_{acc} and σ_{gyro} defined as the standard deviation of the accelerometer and angular rate measurement standard deviations, respectively, \mathbf{I}_3 is a 3×3 identity matrix, and $\mathbf{0}_3$ is a 3×3 matrix of zeros.

To construct $\mathbf{h}_{\text{IMU},n}(\mathbf{x}_n)$, let us first define $\mathbf{g}_{\text{ref}} = [g_{x,\text{ref}}, g_{y,\text{ref}}, g_{z,\text{ref}}]^T$ as the ambient acceleration (in G 's) felt by the reference coordinate system. At the surface of the earth the gravity vector would be $\mathbf{g}_{\text{ref}} = [0, 0, -1]^T$. But since the reference aircraft is maneuvering and undergoing a G load, \mathbf{g}_{ref} will vary with time. But, since the aircraft is much more massive than the rigid body and has much greater inertia, we choose \mathbf{g}_{ref} to be constant since it will vary slowly with time compared to the rigid body being tracked. In addition, we also ignore any change in orientation that the reference coordinate system might be undergoing due to an aircraft maneuver. In future implementations, we may modify our procedure to take into account these small changes. We also assume that the IMU is located at the center of the rigid body's coordinate system (normally its center of gravity or CG). A lever-arm correction should be added that compensates for the centripetal and tangential linear accelerations caused by rotations of the accelerometers about the body's CG.

The rotation matrix \mathbf{M}_{ref} that represents the body's current orientation with respect to the reference coordinate system can now be constructed in a fashion identical to that of the observation model presented in Chapter 19. Using (19.48) through (19.50), we can write

$$\mathbf{M}_{\text{ref}} = \mathbf{M}_{\mathbf{a}_n} \cdot \mathbf{M}_{\mathbf{q}_{n-1}} \quad (20.6)$$

where $\mathbf{M}_{\mathbf{a}_n} = \mathcal{M}_{\mathbf{a}}(\mathbf{a}_n)$ and $\mathbf{M}_{\mathbf{q}_{n-1}} = \mathcal{M}_{\mathbf{q}}(\mathbf{q}_{n-1})$. Since the measurements in \mathbf{z}_n are taken with respect to the body's coordinates system, the noiseless body-referenced

rotation rates $z_{\phi,n}$, $z_{\theta,n}$, and $z_{\varphi,n}$ can be produced as follows

$$\begin{bmatrix} z_{\phi,n} \\ z_{\theta,n} \\ z_{\varphi,n} \end{bmatrix} = \dot{\mathbf{a}}_n^{(\text{body})} = \mathbf{M}_{\text{ref}}^T \dot{\mathbf{a}}_n \quad (20.7)$$

The 3D acceleration measured by the x , y , and z accelerometers (in body-referenced coordinates) at time t_n is

$$\mathbf{g}_n^{(\text{body})} = \ddot{\mathbf{p}}_n^{(\text{body})} + \mathbf{g}_{\text{ref}}^{(\text{body})} \quad (20.8)$$

where $\ddot{\mathbf{p}}_n^{(\text{body})}$ is the translational acceleration of the body's center of gravity and $\mathbf{g}_{\text{ref}}^{(\text{body})}$ is the acceleration felt by the reference coordinate system rotated into body coordinates. Now $\ddot{\mathbf{p}}_n^{(\text{body})} + \mathbf{g}_{\text{ref}}^{(\text{body})}$ can be easily calculated from

$$\begin{aligned} \ddot{\mathbf{p}}_n^{(\text{body})} + \mathbf{g}_{\text{ref}}^{(\text{body})} &= \mathbf{M}_{\text{ref}}^T \ddot{\mathbf{p}}_n + \mathbf{M}_{\text{ref}}^T \mathbf{g}_{\text{ref}} \\ &= \mathbf{M}_{\text{ref}}^T (\ddot{\mathbf{p}}_n + \mathbf{g}_{\text{ref}}) \end{aligned} \quad (20.9)$$

The noiseless body-referenced acceleration measurements can therefore be expressed as

$$\begin{bmatrix} z_{x,n} \\ z_{y,n} \\ z_{z,n} \end{bmatrix} = \mathbf{g}_n^{(\text{body})} = \mathbf{M}_{\text{ref}}^T (\ddot{\mathbf{p}}_n + \mathbf{g}_{\text{ref}}) \quad (20.10)$$

Using (20.7) through (20.10), we can now express $\mathbf{h}_{\text{IMU},n}(\mathbf{x}_n)$ as

$$\begin{aligned} \mathbf{h}_{\text{IMU},n}(\mathbf{x}_n) &= [z_{x,n}, z_{y,n}, z_{z,n}, z_{\phi,n}, z_{\theta,n}, z_{\varphi,n}]^T \\ &= \begin{bmatrix} \mathbf{g}_n^{(\text{body})} \\ \dot{\mathbf{a}}_n^{(\text{body})} \end{bmatrix} \end{aligned} \quad (20.11)$$

20.3.2 The Photogrammetric Component of the Observation Model

To define the photogrammetric component of the observation model, we simply recall the observation model exactly as described in Section 3 of Chapter 19. First, we must rename the components derived in Chapter 19 as

$$\mathbf{z}_n \rightarrow \mathbf{z}_{\text{photo},n} \quad (20.12)$$

$$\mathbf{h}_n(\mathbf{x}_n) \rightarrow \mathbf{h}_{\text{photo},n}(\mathbf{x}_n) \quad (20.13)$$

$$\mathbf{w}_n \rightarrow \mathbf{w}_{\text{photo},n} \quad (20.14)$$

$$\mathbf{R}_n \rightarrow \mathbf{R}_{\text{photo},n} \quad (20.15)$$

Just as in Chapter 19, note that $\mathbf{z}_{\text{photo},n}$, $\mathbf{h}_{\text{photo},n}(\mathbf{x}_n)$, and $\mathbf{w}_{\text{photo},n}$ are of dimension $2M_n$, and $\mathbf{R}_{\text{photo},n}$ is of dimension $2M_n \times 2M_n$ where M_n is the number of feature

points visible from all cameras. We observe that $\mathbf{h}_{\text{photo},n}$ and M_n are time dependent because the number of feature points observed by each camera can change with time.

20.3.3 The Combined Sensor Fusion Observation Model

In order to indicate which type of measurement(s) are available at time t_n , we define an observation flag F_n as

$$F_n = \begin{cases} 1, & \text{Only IMU data is present} \\ 2, & \text{Only photogrammetric data is present} \\ 3, & \text{Both types of data are present} \end{cases} \quad (20.16)$$

The observation vector \mathbf{z}_n at time t_n can now be written as

$$\mathbf{z}_n = \begin{cases} \mathbf{z}_{\text{IMU},n}, & F_n = 1 \\ \mathbf{z}_{\text{photo},n}, & F_n = 2 \\ [\mathbf{z}_{\text{IMU},n}^\top, \mathbf{z}_{\text{photo},n}^\top]^\top, & F_n = 3 \end{cases} \quad (20.17)$$

Similarly, the transformation function $\mathbf{h}_n(\mathbf{x}_n)$ can be written as

$$\mathbf{h}_n(\mathbf{x}_n) = \begin{cases} \mathbf{h}_{\text{IMU},n}(\mathbf{x}_n), & F_n = 1 \\ \mathbf{h}_{\text{photo},n}(\mathbf{x}_n), & F_n = 2 \\ [\mathbf{h}_{\text{IMU},n}^\top(\mathbf{x}_n), \mathbf{h}_{\text{photo},n}^\top(\mathbf{x}_n)]^\top, & F_n = 3 \end{cases} \quad (20.18)$$

The observation noise, being a combination of its components, is also assumed to be the zero-mean Gaussian process $\mathbf{w}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_n)$, with \mathbf{R}_n defined as the diagonal noise covariance matrix

$$\mathbf{R}_n = \begin{cases} \mathbf{R}_{\text{IMU}}, & F_n = 1 \\ \mathbf{R}_{\text{photo}}, & F_n = 2 \\ \begin{bmatrix} \mathbf{R}_{\text{IMU}} & \mathbf{0}_{\text{SF}} \\ \mathbf{0}_{\text{SF}}^\top & \mathbf{R}_{\text{photo}} \end{bmatrix}, & F_n = 3 \end{cases} \quad (20.19)$$

where $\mathbf{0}_{\text{SF}}$ is a zero matrix of dimension $6 \times 2M_n$.

Finally, if the IMU clock is not synchronized with the photogrammetric clock, the data from the IMU may be asynchronous with that of the photogrammetric data and the time difference between successive measurements may not be uniform. Thus, a time-dependent T_n must be used in all tracking filters, with

$$T_n = t_n - t_{n-1} \quad (20.20)$$

20.4 THE GENERATION OF SYNTHETIC DATA

In order to evaluate the performance of the various tracking methods, in Chapter 19 we developed a synthetic rigid body and created a synthetic trajectory that is representative of a store release event. This gave us a synthetic “truth” trajectory that allowed us to create Monte Carlo sets of synthetic noisy observations as seen from a set of virtual cameras. The Monte Carlo synthetic measurement sets were then used in a variety of estimation (solver) methods and the root mean squared (RMS) track errors calculated from each estimation method allowed for comparative performance analysis.

The synthetic trajectories and Monte Carlo noisy camera observations sets from Chapter 19 will be reused in this chapter, and we will add noisy measurements as felt by a virtual IMU’s accelerometers and rate gyroscopes attached to the synthetic rigid body. Thus, we will be able to perform comparative analysis of various estimation methods and also analyze any improvements in performance due to the addition of IMU measurements.

20.4.1 Synthetic Trajectory

We shall use the same synthetic trajectory developed in Section 19.5.2, except that we generate a sequence of times \mathbf{t} that has a faster data rate

$$\mathbf{t} = \{t_n = 0.001 * n, n = 0, 1, 2, \dots, 1000\} \quad (20.21)$$

Although the IMU measurements are generated at the shorter period of 0.001 seconds, the image measurements are still generated at the slower period of 0.005 s. To indicate this, we set the flag variable F_n to be

$$F_n = \begin{cases} 3, & \{n, \text{ mod } 5\} = 0 \\ 1, & \text{otherwise} \end{cases} \quad (20.22)$$

The synthetic trajectory can now be constructed according to (19.106) through (19.111).

20.4.2 Synthetic Cameras

We construct two virtual cameras according to Section 19.5.3.

20.4.3 Synthetic Measurements

20.4.3.1 IMU Measurements. For each $n = \{0, 1, 2, \dots, 1000\}$ we generate the IMU synthetic measurement vector $\mathbf{z}_{\text{IMU},n,\text{synth}}^0$ as

$$\mathbf{z}_{\text{IMU},n,\text{synth}}^0 = \mathbf{h}_{\text{IMU},n}(\mathbf{x}_{n,\text{true}}) + \mathbf{w}_{\text{IMU},n,\text{synth}} \quad (20.23)$$

where

$$\mathbf{w}_{\text{IMU},n,\text{synth}} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{\text{IMU},\text{synth}}) \quad (20.24)$$

with

$$\mathbf{R}_{\text{IMU},\text{synth}} = \begin{bmatrix} \sigma_{\text{acc}}^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \sigma_{\text{gyro}}^2 \mathbf{I}_3 \end{bmatrix} \quad (20.25)$$

Here, σ_{acc} and σ_{gyro} are set to 0.1 G's and 5.0 deg/s, respectively.

20.4.3.2 Camera Measurements. Camera measurements are generated at every fifth point in the time sequence. For each $n = \{0, 1, 2, \dots, 200\}$ we generate the camera synthetic measurement vector $\mathbf{z}_{\text{photo},5n,\text{synth}}^o$ as

$$\mathbf{z}_{\text{photo},5n,\text{synth}}^o = \mathbf{h}_{\text{photo},5n}(\mathbf{x}_{5n,\text{true}}) + \mathbf{w}_{\text{photo},5n,\text{synth}} \quad (20.26)$$

where

$$\mathbf{w}_{\text{photo},5n,\text{synth}} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{\text{photo},5n,\text{synth}}) \quad (20.27)$$

with

$$\mathbf{R}_{\text{photo},5n,\text{synth}} = \sigma_{\text{pixel}}^2 \mathbf{I}_{2M_n} = \sigma_{\text{pixel}}^2 \mathbf{I}_{32} \quad (20.28)$$

Here, σ_{pixel} is set to 2 pixels.

20.4.3.3 Combined Measurements. For each $n = \{0, 1, 2, \dots, 1000\}$ we generate the combined synthetic measurement vector $\mathbf{z}_{n,\text{synth}}^o$ as

$$\mathbf{z}_{n,\text{synth}}^o = \begin{cases} \mathbf{h}_{c,n}(\mathbf{x}_{n,\text{true}}) + \mathbf{w}_{n,\text{synth}}, & \{n, \text{ mod } 5\} = 0 \\ \mathbf{h}_{\text{IMU},n}(\mathbf{x}_{n,\text{true}}) + \mathbf{w}_{n,\text{synth}}, & \text{otherwise} \end{cases} \quad (20.29)$$

where

$$\mathbf{h}_{c,n}(\mathbf{x}_{n,\text{true}}) \triangleq \left[\mathbf{h}_{\text{IMU},n}^\top(\mathbf{x}_{n,\text{true}}), \mathbf{h}_{\text{photo},n}^\top(\mathbf{x}_{n,\text{true}}) \right]^\top \quad (20.30)$$

with

$$\mathbf{w}_{n,\text{synth}} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{n,\text{synth}}) \quad (20.31)$$

and

$$\mathbf{R}_{n,\text{synth}} = \begin{cases} \begin{bmatrix} \mathbf{R}_{\text{IMU},\text{synth}} & \mathbf{0}_{\text{SF}} \\ \mathbf{0}_{\text{SF}}^\top & \mathbf{R}_{\text{photo},5n,\text{synth}} \end{bmatrix}, & \{n, \text{ mod } 5\} = 0 \\ \mathbf{R}_{\text{IMU},\text{synth}}, & \text{otherwise} \end{cases} \quad (20.32)$$

Here, $\mathbf{0}_{\text{SF}}$ is a zero matrix of dimensions 6×32 . Note that a new Monte Carlo sample is drawn from $\mathbf{R}_{n,\text{synth}}$ for each time step.

20.5 ESTIMATION METHODS

The estimation methods used for comparative analysis of the sensor fusion models include methods using photogrammetric data alone, methods that use IMU data alone and our method that fuses the video data with the IMU data. In Chapter 19, we concluded that the second-order unscented Kalman filter (UKF(2)) achieves the best performance of all of the solvers tested and provides a marked improvement over the standard zeroth-order NLLSQ solver when only photogrammetric video data is used. For completeness, the comparative performance analysis of Section 20.6 will include performance results from both the UKF(2) and the NLLSQ when only photogrammetric observation data is used. When only IMU data is used as observations, a second-order Predictor–Corrector method and the UKF(2) are implemented as tracking filters. To do this, we set $F_n = 1$, for $\forall n$ so that only IMU data is used as observations.

For a description of the NLLSQ and UKF(2) estimation processes when only photogrammetric video data is used, see Sections 19.4.1 and 19.4.2, respectively. A description of these methods will not be repeated here. In Section 20.5.1, we discuss the Predictor–Corrector estimation method used when only IMU data is being processed.

20.5.1 Initial Value Problem Solver for IMU Data

The accepted method for estimating the position and orientation of a rigid body using an IMU’s translational acceleration and angular rate measurements is to numerically solve a system of *ordinary differential equations* (ODE) using an *initial value problem* (IVP) solver. There are many IVP solvers to choose from and here we describe a simple but accurate iterative solver called the Euler Trapezoidal or *Predictor–Corrector* method [16,17]. In what follows, we construct a first-order system of ODEs that relate the IMU measurements to the position, orientation, and translational velocities of the rigid body. We then briefly describe how the solver is initialized and how it is used to solve the system of ODEs resulting in an estimate of the object’s trajectory (position and orientation) as a function of time.

20.5.1.1 The Predictor–Corrector IVP Solver. In the general case, the objective is to solve a first-order system of ODEs that can be cast as an initial value problem. If one wishes to estimate an unknown function $\mathbf{x}(t)$ from a vector-valued function $\mathbf{f}(t, \mathbf{x}(t))$ that expresses the relationship of $\mathbf{x}(t)$ to its first time derivative, given an initial value $\mathbf{x}(t_0) = \mathbf{x}_0$, we can write

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}(t)) \quad (20.33)$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \quad (20.34)$$

Euler’s method is perhaps the simplest quadrature rule for integrating an ODE. Let $\mathbf{t} = \{t_n, n = 0, 1, 2, \dots, N\}$ and identify $\mathbf{x}_n = \mathbf{x}(t_n)$. Using Euler’s rule, a prediction

of \mathbf{x}_n can be obtained from

$$\mathbf{x}_n^{\text{pred}} = \mathbf{x}_{n-1} + T\mathbf{f}(t_{n-1}, \mathbf{x}_{n-1}) \quad (20.35)$$

where $T = t_n - t_{n-1}$. Euler's method can be thought of as a rectangular integration rule that evaluates the integrand only once at the left-hand endpoint of the interval. The rule is exact if $\mathbf{f}(t, \mathbf{x})$ is constant but is not exact if $\mathbf{f}(t, \mathbf{x})$ is linear, with the error being proportional to the step size T . Small step sizes are needed in order to get an accurate estimate but there is no automatic way to determine the step size based on a required error budget. To reduce this uncertainty in the error, it is important to follow the Euler step with a second function evaluation at the right-hand endpoint of the interval. One such approach is called the Predictor–corrector method that uses the trapezoidal rule in an iterative fashion. First, an Euler step is taken across the interval to make an initial prediction using (20.35). Then the function is evaluated at the extrapolated point $\mathbf{x}_n^{\text{pred}}$ and the two slopes are averaged to obtain the first corrected *estimate* of \mathbf{x}_n

$$\hat{\mathbf{x}}_{1,n} = \mathbf{x}_{n-1} + \frac{T}{2} [\mathbf{f}(t_{n-1}, \mathbf{x}_{n-1}) + \mathbf{f}(t_n, \mathbf{x}_n^{\text{pred}})] \quad (20.36)$$

For $k > 1$, the correction step is iteratively repeated and the quality of the estimate $\hat{\mathbf{x}}_{k,n}$ is evaluated by comparing it to its predecessor using

$$e_k = \|\hat{\mathbf{x}}_{k,n} - \hat{\mathbf{x}}_{k-1,n}\|^2 \quad (20.37)$$

where

$$\hat{\mathbf{x}}_{k,n} = \mathbf{x}_{n-1} + \frac{T}{2} [\mathbf{f}(t_{n-1}, \mathbf{x}_{n-1}) + \mathbf{f}(t_n, \hat{\mathbf{x}}_{k-1,n})] \quad (20.38)$$

The iteration is terminated when e_k becomes sufficiently small relative to some threshold, that is, $e_k < \varepsilon$. Usually, we take $\varepsilon = \sqrt{\varepsilon_{\text{mach}}}$, with $\varepsilon_{\text{mach}} \simeq 2.2204 \times 10^{-16}$ that is the smallest double precision machine number larger than zero that when added to one gives a value discernibly greater than one. When the iteration reaches the threshold, we set $\mathbf{x}_n = \mathbf{x}_{k,n}^{\text{corr}}$ and proceed to the next time step.

20.5.1.2 A System of ODE's for the IMU Problem. We now construct a first order system of ODEs that relates body-referenced rotation rates measured by the IMU to rotation angles and rates in the reference coordinate system, and relates body-referenced translational accelerations to velocities and positions in aircraft coordinates. Let $\mathbf{z}^o = [(\mathbf{z}_\alpha^o)^\top, (\mathbf{z}_\omega^o)^\top]^\top$ be a measurement from the IMU, where $\mathbf{z}_\alpha^o = [z_x, z_y, z_z]^\top$ represents a measurement from the three orthogonal accelerometers and $\mathbf{z}_\omega^o = [z_\phi, z_\theta, z_\psi]^\top$ represents a measurement from the three orthogonal rate gyroscopes. Let $\mathbf{x} = [\mathbf{p}^\top, \dot{\mathbf{p}}^\top, \mathbf{q}^\top]^\top$ be the unknown state vector we wish to estimate, where $\mathbf{p} = [p_x, p_y, p_z]^\top$ represents the 3D position, $\dot{\mathbf{p}} = [\dot{p}_x, \dot{p}_y, \dot{p}_z]^\top$ represents the 3D velocity, and $\mathbf{q} = [q_s, q_x, q_y, q_z]^\top$ is a unit quaternion representing the 3D orientation. Note that \mathbf{z}^o is measured in body-referenced coordinates while \mathbf{x} is expressed in reference coordinates.

To write the ODE for \mathbf{q} , we begin with (19.122) and replace $\boldsymbol{\omega}$ with \mathbf{z}_ω^0 to obtain

$$\frac{d\mathbf{q}}{dt} = \frac{1}{2}\mathbf{q}\mathbf{z}_\omega^0 \quad (20.39)$$

where it is understood that in the context of the quaternion multiplication $\mathbf{q} \cdot \mathbf{z}_\omega^0$, the multiplicand \mathbf{z}_ω^0 is represented by the pure quaternion $[0, \mathbf{z}_\omega^0]^\top = [0, z_\phi, z_\theta, z_\varphi]^\top$.

Likewise, to relate the 3D position \mathbf{p} to the IMU measured body-referenced acceleration measurements \mathbf{z}_α^0 , the second-order ODE equation is

$$\frac{d^2\mathbf{p}}{dt^2} = \mathbf{M}_\mathbf{q}\mathbf{z}_\alpha^0 - \mathbf{g} \quad (20.40)$$

where the rotation matrix $\mathbf{M}_\mathbf{q} = \mathcal{M}_\mathbf{q}(\mathbf{q})$ rotates \mathbf{z}_α^0 into reference coordinates and \mathbf{g} is the gravitational acceleration acting on the reference coordinate system. This second-order equation can be reduced to a system of first-order equations by including the velocity equation

$$\frac{d\mathbf{p}}{dt} = \dot{\mathbf{p}} \quad (20.41)$$

Now, (20.40) can be rewritten using (20.41) to obtain

$$\frac{d\dot{\mathbf{p}}}{dt} = \mathbf{M}_\mathbf{q}\mathbf{z}_\alpha^0 - \mathbf{g} \quad (20.42)$$

Thus, (20.39), (20.41), and (20.42) constitute a 10-dimensional first-order system of ODEs that fully describe the relationship between the IMU measurements and the rigid body's position, orientation, and translational velocities. If we define $\mathbf{x} = [\mathbf{p}^\top, \dot{\mathbf{p}}^\top, \mathbf{q}^\top]^\top$, we can now write (20.33) as

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}) = \mathbf{f}\left(t, \begin{bmatrix} \mathbf{p} \\ \dot{\mathbf{p}} \\ \mathbf{q} \end{bmatrix}\right) = \begin{bmatrix} \dot{\mathbf{p}} \\ \mathbf{M}_\mathbf{q}\mathbf{z}_\alpha^0 - \mathbf{g} \\ \frac{1}{2}\mathbf{q}\mathbf{z}_\omega^0 \end{bmatrix} \quad (20.43)$$

20.5.1.3 Initializing and Using the Predictor–Corrector for Position and Orientation Estimation. The Predictor–Corrector can be initialized for estimation of the trajectory of the synthetic rigid body by setting

$$\begin{aligned} \mathbf{x}_0 &= [\mathbf{p}_0^\top, \dot{\mathbf{p}}_0^\top, \mathbf{q}_0^\top]^\top \\ &= [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]^\top \end{aligned} \quad (20.44)$$

where the last four components are the unit quaternion that represents a zero rotation. Note that in other contexts \mathbf{x}_0 might contain nonzero components, depending on the circumstances. Since the Predictor–Corrector is an unconstrained estimator, once an iteration has produced the estimate $\hat{\mathbf{x}}_n$ at time t_n , the quaternion component $\hat{\mathbf{q}}_n$ will not necessarily be unitary and would therefore not represent the orientation correctly.

To correct this, $\hat{\mathbf{q}}_n$ is normalized to unit length prior to the next temporal iteration by performing

$$\hat{\mathbf{q}}_n = \frac{\hat{\mathbf{q}}_n}{\|\hat{\mathbf{q}}_n\|} \quad (20.45)$$

20.6 PERFORMANCE COMPARISON ANALYSIS

The performance analysis to be presented below was conducted using the synthetic test data described in Section 20.4. Note that this is the same synthetic trajectory (with identical noise in the image data) as was used in Section 19.6. The performance of the sensor fusion estimation filter (UKF(2)) is compared against two estimators that operate only on photogrammetric data (NLLSQ and UKF(2)) and also against two estimators that use only IMU data (Predictor–Corrector and UKF(2)). The performance metric used in all cases is the root mean squared (RMS) error. As we will show below, the sensor fusion estimator had the lowest RMS errors and was therefore the best estimator.

Figures 20.2 through 20.4 show the estimated track components of a single run of the sensor fusion filter against the truth track components. Figure 20.5 shows a

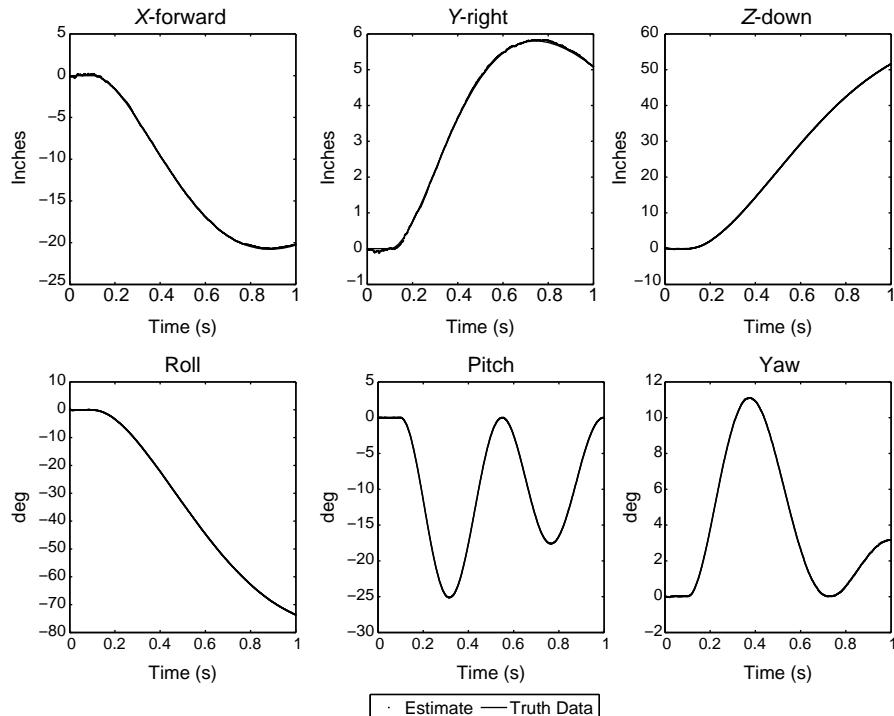


FIGURE 20.2 Translational and rotational position using the sensor fusion estimator.

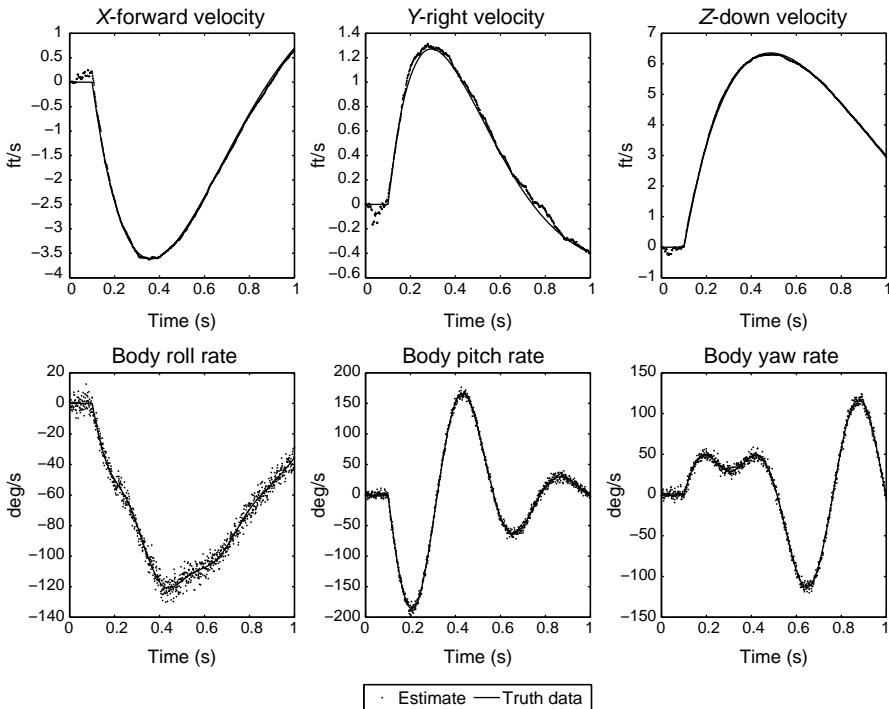


FIGURE 20.3 Translational and rotational velocity using the sensor fusion estimator.

close-up of a small section of the lateral position truth trajectory with the estimated trajectories from multiple estimators superimposed over the truth. Identical noisy observations and initialization were used as inputs to all estimation methods. Analysis of these figures leads to a number of observations.

Both estimation methods that use only IMU data are inaccurate, especially the UKF implementation. It is well known that estimators that rely on IMU data alone are sensitive to initial conditions and calibration biases, but in this (synthetic) case, the initial conditions and calibrations are exact. Rather, the inaccuracies appear to be caused by drift due to random noise in the observations, especially since one of the solver's estimate is too large and the other too small.

In addition, both estimators that use only photogrammetric data are accurate but imprecise. The UKF(2) appears to be more accurate (and more precise) than the NLLSQ due primarily to its inclusion of derivative terms in the state vector estimates. We conclude that the sensor fusion results are the most accurate of any of the tested estimation methods, because whereas the photogrammetry-only estimators measure spatial parameters and must estimate derivatives, and whereas the IMU-only estimators measure derivatives and must estimate spatial parameters, the fusion estimator uses both spatial and derivative measurements as the basis for estimation.

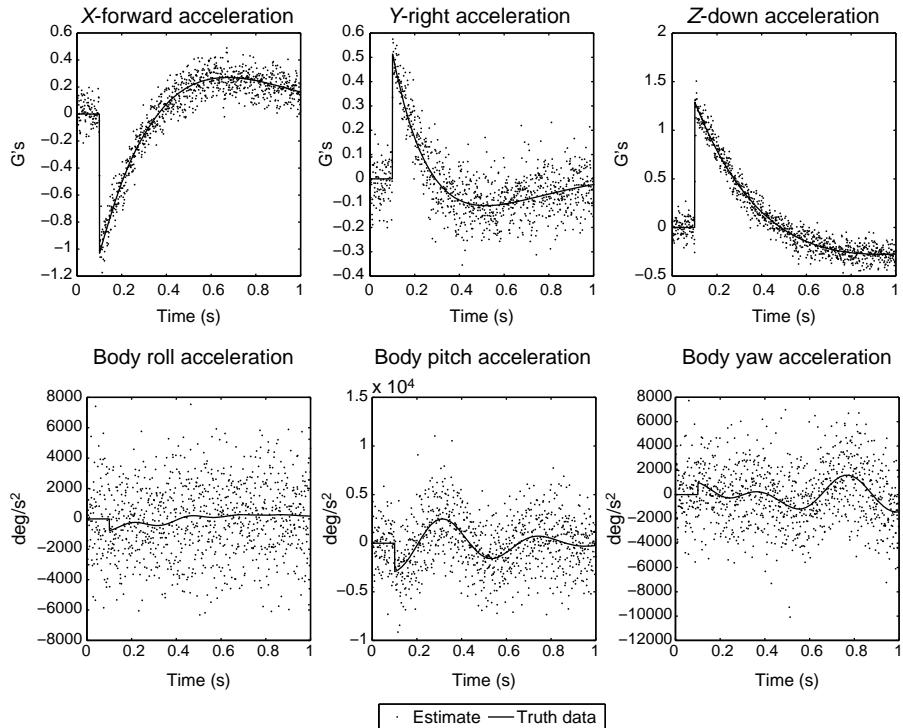


FIGURE 20.4 Translational and rotational acceleration using the sensor fusion estimator.

The sensor fusion estimator's behavior is apparent from Figure 20.5 by comparing its trajectory with the photogrammetry-only UKF, which is sampled at one-fifth the rate of the IMU data. Sensor fusion trends smoothly until at every fifth sample it shifts in one direction or the other when a photogrammetric observation arrives. It shifts in the same direction as the photogrammetry-only UKF, but with less magnitude due to the more accurate derivative estimates provided by the IMU data.

These preliminary conclusions will be tested in Section 20.6.2 where we present the RMS error comparisons for the various estimators.

20.6.1 Filter Performance Comparison Methodology

In Section 19.6.1, we presented a performance methodology that used RMS error as a metric. We will reuse that methodology here. The synthetic noiseless trajectory $\{\mathbf{x}_n, n = 0, 1, 2, \dots, N\}$ developed in Section 19.5 is reused for our analysis here, *except that now* $N = 1000$ as described in Section 20.4.1. That trajectory is transformed into a set of noiseless synthetic truth observations $\{\mathbf{h}_n(\mathbf{x}_n), n = 0, 1, 2, \dots, N\}$, where $\mathbf{h}_n(\mathbf{x}_n)$ can contain either photogrammetric or IMU measurements or both, based on

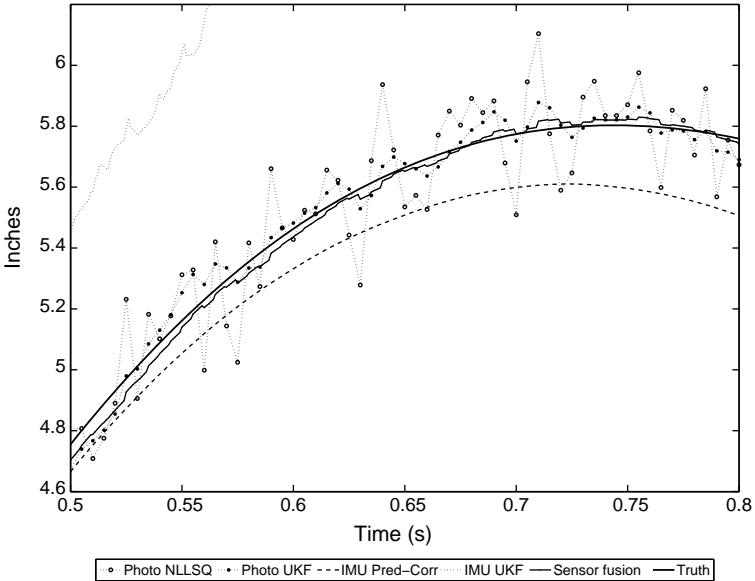


FIGURE 20.5 Lateral position estimates from multiple estimation filters using identical measurement inputs.

the flag F_n . Then, $N_{\text{RMS}} = 100$ sets of noisy observations $\{\mathbf{z}_{n,r}^o, n = 0, 1, 2, \dots, N; r = 1, 2, \dots, N_{\text{RMS}}\}$ are produced by adding independent zero-mean Gaussian noise to the synthetic truth observations to produce N_{RMS} sets of noisy observations. Each noisy observation set is used to exercise each tracking filter to produce N_{RMS} sets of state vector estimates $\{\hat{\mathbf{x}}_{n|r}, n = 0, 1, 2, \dots, N; r = 1, 2, \dots, N_{\text{RMS}}\}$. We then calculate the RMS error for each tracking filter as a function of time, producing $\mathbf{e}_{\mathbf{p},n}^{(\text{filter})}$ and $\mathbf{e}_{\mathbf{a},n}^{(\text{filter})}$ for 3D translational and angular position errors, $\mathbf{e}_{\dot{\mathbf{p}},n}^{(\text{filter})}$ and $\mathbf{e}_{\dot{\mathbf{a}},n}^{(\text{filter})}$ for 3D translational and angular velocity errors, and $\mathbf{e}_{\ddot{\mathbf{p}},n}^{(\text{filter})}$ and $\mathbf{e}_{\ddot{\mathbf{a}},n}^{(\text{filter})}$ for 3D translational and angular acceleration errors.

20.6.2 Filter Comparison Results

For each type of estimator, the RMS errors are computed as a function of time and then the mean and maximum errors (over time) are generated for each case. These mean and maximum values are summarized in Tables 20.1, 20.2, and 20.3, for the position, velocity, and acceleration RMS errors, respectively.

Figures 20.6, 20.7, and 20.8 show time histories of the performance of each of the five tested estimators for positional and rotational RMS errors, position and rotation velocity RMS errors, and position and rotation acceleration errors, respectively. It is obvious from Tables 20.1 through 20.3 and Figures 20.6 through 20.8 that the UKF(2)

TABLE 20.1 Synthetic Data RMS Positional Error Summary

Filter Type	Data Type	Position (in.)		Orientation (deg)	
		Mean	Max	Mean	Max
NLLSQ	Photo	0.68	0.81	0.22	0.26
UKF(2)	Photo	0.32	0.52	0.15	0.22
Pred/Corr	IMU	1.21	3.09	0.55	0.85
UKF(2)	IMU	1.50	5.15	1.69	2.10
UKF(2)	Both	0.16	0.42	0.06	0.22

TABLE 20.2 Synthetic Data RMS Velocity Error Summary

Filter Type	Data Type	Translational Velocity (ft/s)		Angular Velocity (deg/s)	
		Mean	Max	Mean	Max
NLLSQ	Photo	N/A	N/A	N/A	N/A
UKF(2)	Photo	0.71	1.63	12.34	22.18
Pred/Corr	IMU	0.30	0.48	8.66	9.75
UKF(2)	IMU	0.41	1.05	7.47	8.55
UKF(2)	Both	0.10	0.24	7.43	8.53

TABLE 20.3 Synthetic Data RMS Acceleration Error Summary

Filter Type	Data Type	Translational Acceleration (G's)		Angular Acceleration (deg/s ²)	
		Mean	Max	Mean	Max
NLLSQ	Photo	N/A	N/A	N/A	N/A
UKF(2)	Photo	0.37	1.70	653.3	3005.9
Pred/Corr	IMU	1.7	0.2	N/A	N/A
UKF(2)	IMU	0.16	0.24	4532.1	5106.0
UKF(2)	Both	0.15	0.89	4217.1	5119.2

estimator that uses data fused from both photogrammetric and IMU sensors has the best performance overall, except in the case of rotational accelerations.

20.7 CONCLUSIONS

We have demonstrated that the sensor fusion estimator produces results that are more accurate than either photogrammetry-based or IMU-based estimators alone. In the context of our UKF solver, the two types of measurements complement each other.

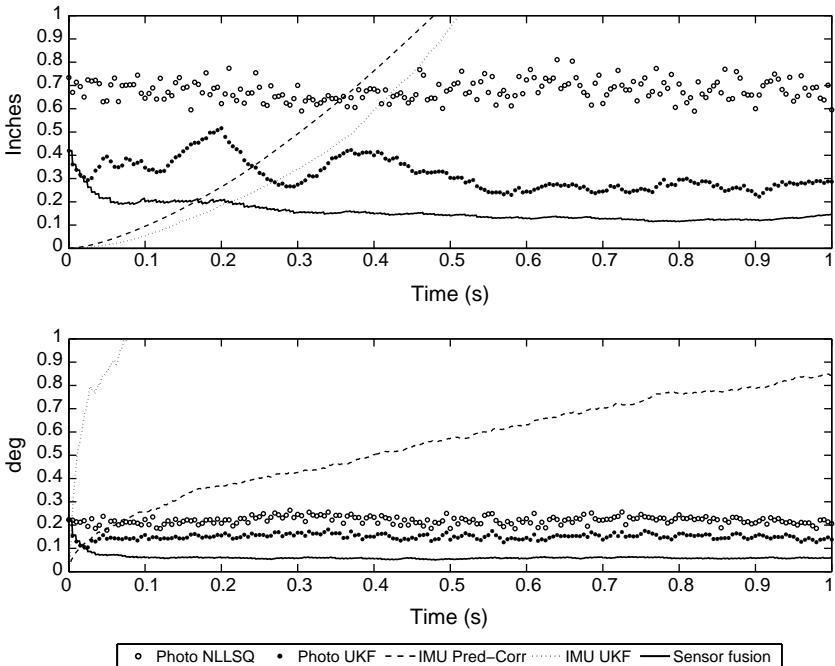


FIGURE 20.6 RMS errors for position and orientation of all tracking filters using synthetic data.

IMU measurements mitigate the noise present in the photogrammetric data, thus making positions and angles more accurate. And photogrammetry compensates for biases and drift in the IMU measurements, making translational velocities more accurate.

20.8 FUTURE WORK

The current measurement model assumes that the IMU is located at the center of the rigid body's coordinate system (CG). To make the sensor fusion estimator more practical, a lever-arm correction should be added to the measurement model that compensates for the centripetal and tangential linear accelerations caused by rotations of the accelerometers about the body's CG. We also intend to show that using sensor fusion to analyze photogrammetry and IMU data together improves fault tolerance, in that dropouts in either data source are mitigated by measurements from the other data source.

In addition to increased accuracy and fault tolerance, sensor fusion can potentially improve photogrammetry turnaround time. Collecting 2D tracking data is the most time-consuming step in the photogrammetric data reduction process. We intend to investigate restructuring that process by using sensor fusion to reduce the quantity

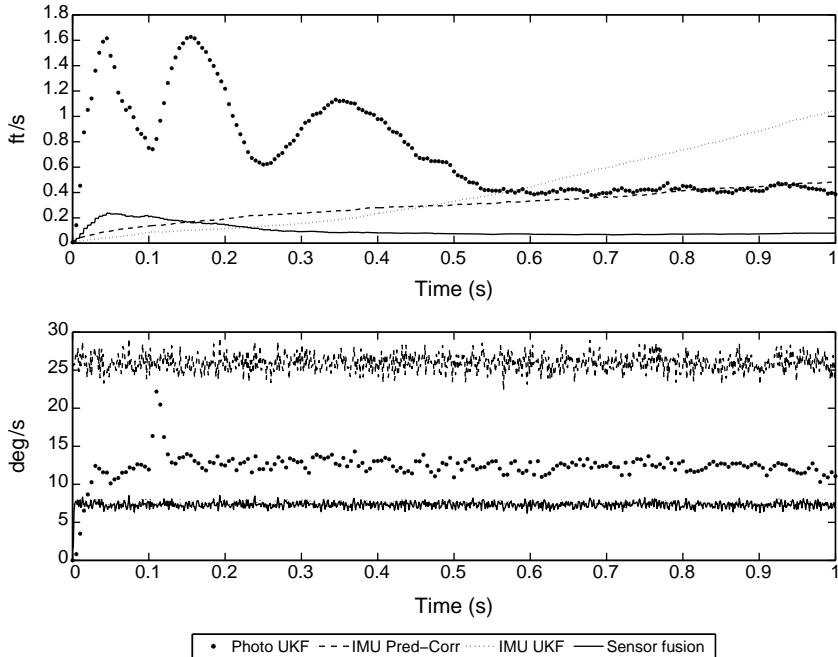


FIGURE 20.7 RMS errors for position and orientation velocities of all tracking filters using synthetic data.

of image measurements that are necessary to produce a 6 DOF solution of sufficient accuracy. The UKF uncertainty model provides a measure of the accuracy of a 6 DOF solution for a particular event. We can compare the uncertainties in the sensor fusion result with the photogrammetry-only result, and determine how much less photogrammetric data needs to be collected to obtain sufficient accuracy.

Although the rate gyroscopes in the current generation of IMUs appear to be very accurate, the accelerometers appear to exhibit a large enough calibration bias to induce significant error in the 6 DOF solution. We have begun to use sensor fusion to estimate calibration biases in order to further improve accuracy. Also, we currently assume that the forces the aircraft undergoes are constant during the brief period that the store is near the aircraft. We have begun incorporating data from the aircraft's INS system to determine if correcting for changes in aircraft accelerations and rotation rates cause a significant difference in solution results.

We also intend to address the problem of latency in the IMU data. While the imagery used in photogrammetry is time stamped as it is collected and recorded aboard the aircraft, the IMU data is transmitted from the store to the ground station and collected in a data buffer before being time stamped. These multiple data paths cause a time synchronization error of indeterminate length. We have begun trying to estimate the latency by examining the residual errors in the sensor fusion solution when the latency is allowed to vary.

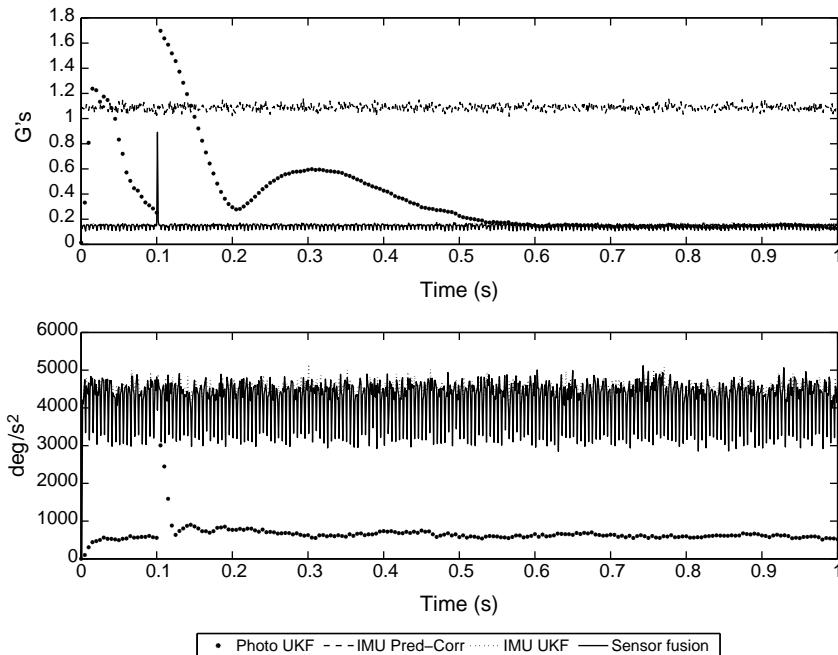


FIGURE 20.8 RMS errors for position and orientation accelerations of all tracking filters using synthetic data.

REFERENCES

1. Iu SL, Wohn K. Estimation of General Rigid Body Motion from a long Sequence of Images. Technical Report, Department of Computer Science, University of Pennsylvania; 1990.
2. Lee JW, Kim MS, Kweon IS. A Kalman filter based visual tracking algorithm for an object moving in 3D. *IROS '95 Proceedings International Conference on Intelligent Robots and Systems*, Vol. 1; 1995.
3. Gennery DB. Visual tracking of known three-dimensional objects. *Int. J. Comput. Vision* 1992;7(3): 243–270.
4. Huang TS, Netravali AN. Motion and structure from feature correspondences: a review. *Proc. IEEE* 1994;82(2): 252–268.
5. Ude A. Filtering in a unit quaternion space for model-based object tracking. *Robot. Autonom. Syst.* 1999;20: 163–172.
6. Halvorsen K, Söderström T, Stokes V, and Lanshammar H. Using an extended Kalman filter for rigid body pose estimation. *J. Biomech. Eng.* 2005;127: 475–483.
7. Algrain MC, Saniie J. Estimation of 3D Angular motion using gyroscopes and linear accelerometers. *IEEE Trans. Aero. Electron. Syst.* 1991;27(6): 910–920.

9. Marins JL, Yun X, Bachmann ER, McGhee RB, and Zyda MJ. An extended Kalman filter for quaternion-based orientation estimation using MARG Sensors. In *Proceedings of the 2001 International Conference on Intelligent Robots and Systems*, 2001, pp. 2003–2011.
10. Yun X, Lizarraga M, Bachmann ER, McGhee RB. An improved quaternion-based filter for real-time tracking of rigid body orientation. In *Proceedings of the 2003 International Conference on Intelligent Robots and Systems*, 2003, pp. 1074–1079.
11. Sabatini AM. Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing. *IEEE Trans. Biomed. Eng.* 2006;53(7): 1346–1356.
12. Yun X, Bachmann ER. Design, implementation, and experimental results of a quaternion-based Kalman filter for human body motion Tracking. *IEEE Trans. Robot.* 2006;22(6):1216–1227.
13. Kelly J, Sukhatme GS. Fast relative pose calibration for visual and inertial sensors. In, *Proceedings of the 11th International Symposium on Experimental Robotics (ROBIO '08)*, 2008.
14. Bleser G and Stricker D. Advanced tracking through efficient image processing and visual-inertial sensor fusion. *Comput. Graphics* 2009;33: 59–72.
15. Jeon S, Tomizuka M, Katou T. Kinematic Kalman filter (KKF) for robot end-effector sensing. *J. Dyn. Syst. Meas. Control* 2009; 021010:8.
16. Moler C. *Numerical Computing with Matlab*, 2nd ed. SIAM; 2004.
17. Van Loan CF. *Introduction to Scientific Computing*, 2nd ed. Prentice Hall; 2000.

INDEX

- affine transformation, 35, 80
- auxiliary particle filter
 - derivation, 238
 - process, 241
- angle-axis rotations, see rigid body rotation representations
- Bayesian estimation, 43, 46
- Bayes' rule, 31
- bootstrap particle filter
 - derivation, 230
 - process, 231
- Cartesian-to-spherical transformation, 302
- Chapman-Kolmogorov equation, 48
- constant Cartesian velocity dynamic model, 261
- constant spherical velocity dynamic model, 266
- constant turn-rate simulation, 294
- coordinate transformations in
 - multidimensional space, 301
- Cramer-Rao lower bound
 - application to DIFAR tracking, 192
 - Gaussian additive noise, 190
 - general derivation, 184
 - linear models, 191
 - recursive, 186
 - zero process noise, 191
- cumulative distribution function, 30, 34
- density estimation
 - histogram, 202
 - kernel density, 204
 - recursive Bayesian estimation, 46, 201
- DIFAR
 - dynamic model, 58
 - likelihood function, 67
 - observation model, 59
 - scenario simulator, 58, 65
 - signal processing, 62
 - single buoy bearing track estimation,
 - linear Kalman filter, 88
- track estimation results
 - auxiliary particle filter, 242
 - bootstrap particle filter, 231
 - extended Kalman filter, 109
 - Gauss-Hermite Kalman filter, 163
 - generalized Monte Carlo particle filter, 252
 - spherical simplex Kalman filter, 147
 - unscented Kalman filter, 137
- error ellipses, 176
- extended Kalman filter
 - alternate derivation of covariance prediction, 107
 - constant spherical velocity state prediction, 271
 - linearized, 105
 - observation prediction, 96, 102

- extended Kalman filter (*Continued*)
 - process block diagram, 169
 - radar observation prediction, 264
 - second order, 105
 - state prediction, 94, 99
 - transformation of predication equations, 96, 103
- extended Kalman particle filter, 243
- finite difference Kalman filter
 - alternate derivation of covariance prediction, 125
 - observation prediction, 118, 124
 - process, 119, 126
 - process block diagram, 170
 - simplified, 118
 - state prediction, 116, 120
- Gauss-Hermite Kalman filter
 - multidimensional derivation, 155
 - observation prediction, 155, 160
 - one-dimensional derivation, 148
 - process, 161
 - process block diagram, 170
 - sparse-grid approximation, 160
 - state prediction, 154, 160
- Gaussian cumulative density function, 34
- Gaussian filter, general, 80
- Gaussian moments, 38
- Gaussian moment integrals, 35
- Gaussian prediction equations, 78
- Gaussian probability density function, 32
- Gaussian weighted integrals, numerical methodology, 82, 128
- generalized Monte Carlo particle filters
 - Gaussian particle filter, 248
 - Gaussian particle filter block diagram, 249
 - combination particle filter, 250
 - combination particle filter block diagram, 250
 - sigma point combination particle filter
 - block diagram, 251
 - tracking a rigid body, 325
 - generating Gaussian samples, 40
- Hermite polynomial approximation, 28, 149
- hierarchy of Bayesian Estimators, 5
- importance sampling, 210
- inertial measurement unit observation model, 348
- initialization for rigid body tracking, 326
- Initialization using a predictor-corrector method, 356
- joint probability density, 30
- Kalman filter summary, 168
- Kalman filter update equations, 51, 76
- linear Kalman filter
 - linear models, 86
 - process, 89
 - process block diagram, 169
 - state prediction for constant Cartesian velocity, 264
 - state prediction for constant spherical rates, 271
- matrix operations
 - block matrix inversion, 14
 - conventions and notations, 11
 - matrix inversion, 13
 - matrix square root, 15
 - sums and products, 12
- marginal probability density, 31
- Monte Carlo Kalman filter
 - derivation, 164
 - observation prediction, 165
 - process, 166
 - state prediction, 165
- multinomial cubature integration rules, 128
- nonlinear least squares
 - general estimation, 321
 - initialization for rigid body tracking, 326
- observation prediction, 53, 75, 79
- optimal particle filter
 - derivation, 233
 - Gaussian, 235
 - locally linearized, 236
 - process, 237
- photogrammetric (video) observation model, 318

- photogrammetric rigid body tracking
 performance, 334
- point estimators, 43
- polynomial approximation, general, 19,
 23
- predictor-corrector initial value problem
 solver, 354
- probability density function, 30, 32
- quaternion, see rigid body rotation
 representations
- radar spherical observation model, 263
- Rao-Blackwellization, 245
- resampling, 222
- recursive estimation of moments, 49, 54
- resampling, 222
- regularization (move step), 227
- rigid body rotation representations, 342
- rigid falling body dynamics
 combined translation and rotation motion,
 316
- dynamic motion model, 317
- rotational motion, 313
- translational motion, 311
- root mean squared error, 182
- sensor fusion, 346
- sensor fusion observation model, 351
- sensor fusion rigid body tracking
 performance, 357
- sequential importance sampling
 general process, 221
- general theory, 218
- sequential importance sampling particle
 filter, 226
- sigma point Kalman filters, 170
- simplex polynomial approximation, 29
- spherical simplex Kalman filter
 process block diagram, 170
- derivation, 140
- process, 146
- spherical-to-Cartesian transformation, 305
- state prediction, 49, 74, 79
- Sterling's polynomial approximation, 21, 27
- synthetic rigid body
 synthetic trajectory generator, 328, 352
- photogrammetric synthetic
 measurements, 333
- inertial measurement unit observation
 measurements, 352
- Taylor polynomial approximation, 20 ,24
- tracking methodology, 57
- tracking in three dimensions
 filter implementation issues, 273
- filter initialization, 276
- tracking performance, 278
- unscented Kalman filter
 alternate version, 135
- background, 130
- constant spherical velocity state
 prediction, 271
- derivation, 131
- observation prediction, 135
- process, 136
- process block diagram, 170
- radar observation prediction, 264
- state prediction, 134
- tracking a rigid body, 323
- unscented particle filter
 derivation, 243
- process, 244
- vector point generators, 16, 173