DISTRIBUTED DATA FUSION ACROSS MULTIPLE
HARD AND SOFT MOBILE SENSOR PLATFORMS

A Dissertation in
Electrical Engineering

by

Gregory L. Sinsley

© 2012 Gregory L. Sinsley

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2012
The dissertation of Gregory L. Sinsley was reviewed and approved\(^1\) by the following:

Lyle N. Long  
Distinguished Professor of Aerospace Engineering and Mathematics  
Dissertation Advisor  
Co-Chair of Committee

W. Kenneth Jenkins  
Professor of Electrical Engineering  
Co-Chair of Committee

David Miller  
Professor of Electrical Engineering

David Hall  
Professor of Information Sciences and Technology

John Yen  
Professor of Information Sciences and Technology

Joseph Horn  
Associate Professor of Aerospace Engineering

Kultegin Aydin  
Professor of Electrical Engineering  
Head of the Department of Electrical Engineering

\(^1\)Signatures on file in the Graduate School.
Abstract

One of the biggest challenges currently facing the robotics field is sensor data fusion. Unmanned robots carry many sophisticated sensors including visual and infrared cameras, radar, laser range finders, chemical sensors, accelerometers, gyros, and global positioning systems. By effectively fusing the data from these sensors, a robot would be able to form a coherent view of its world that could then be used to facilitate both autonomous and intelligent operation. Another distinct fusion problem is that of fusing data from teammates with data from onboard sensors. If an entire team of vehicles has the same worldview they will be able to cooperate much more effectively. Sharing worldviews is made even more difficult if the teammates have different sensor types. The final fusion challenge the robotics field faces is that of fusing data gathered by robots with data gathered by human teammates (soft sensors). Humans sense the world completely differently from robots, which makes this problem particularly difficult. The advantage of fusing data from humans is that it makes more information available to the entire team, thus helping each agent to make the best possible decisions.

This thesis presents a system for fusing data from multiple unmanned aerial vehicles, unmanned ground vehicles, and human observers. The first issue this thesis addresses is that of centralized data fusion. This is a foundational data fusion issue, which has been very well studied. Important issues in centralized fusion include data association, classification, tracking, and robotics problems. Because these problems are so well studied, this thesis does not make any major contributions in this area, but does review it for completeness. The chapter on centralized fusion concludes with an example unmanned aerial vehicle surveillance problem that demonstrates many of the traditional fusion methods.

The second problem this thesis addresses is that of distributed data fusion. Distributed data fusion is a younger field than centralized fusion. The main issues in distributed fusion that are addressed are distributed classification and distributed tracking. There are several well established methods for performing distributed fusion that are
first reviewed. The chapter on distributed fusion concludes with a multiple unmanned vehicle collaborative test involving an unmanned aerial vehicle and an unmanned ground vehicle.

The third issue this thesis addresses is that of soft sensor only data fusion. Soft-only fusion is a newer field than centralized or distributed hard sensor fusion. Because of the novelty of the field, the chapter on soft only fusion contains less background information and instead focuses on some new results in soft sensor data fusion. Specifically, it discusses a novel fuzzy logic based soft sensor data fusion method. This new method is tested using both simulations and field measurements.

The biggest issue addressed in this thesis is that of combined hard and soft fusion. Fusion of hard and soft data is the newest area for research in the data fusion community; therefore, some of the largest theoretical contributions in this thesis are in the chapter on combined hard and soft fusion. This chapter presents a novel combined hard and soft data fusion method based on random set theory, which processes random set data using a particle filter. Furthermore, the particle filter is designed to be distributed across multiple robots and portable computers (used by human observers) so that there is no centralized failure point in the system.

After laying out a theoretical groundwork for hard and soft sensor data fusion the thesis presents practical applications for hard and soft sensor data fusion in simulation. Through a series of three progressively more difficult simulations, some important hard and soft sensor data fusion capabilities are demonstrated. The first simulation demonstrates fusing data from a single soft sensor and a single hard sensor in order to track a car that could be driving normally or erratically. The second simulation adds the extra complication of classifying the type of target to the simulation. The third simulation uses multiple hard and soft sensors, with a limited field of view, to track a moving target and classify it as a friend, foe, or neutral.

The final chapter builds on the work done in previous chapters by performing a field test of the algorithms for hard and soft sensor data fusion. The test utilizes an unmanned aerial vehicle, an unmanned ground vehicle, and a human observer with a laptop. The test is designed to mimic a collaborative human and robot search and
rescue problem. This test makes some of the most important practical contributions of the thesis by showing that the algorithms that have been developed for hard and soft sensor data fusion are capable of running in real time on relatively simple hardware.
Table of Contents

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

List of Figures ................................................................. xiv

List of Algorithms ............................................................. xxi

Nomenclature ................................................................. xxiii

Acknowledgments ............................................................... xxvi

Chapter 1. Introduction ....................................................... 1
  1.1 Background ............................................................... 3
    1.1.1 Distributed data fusion for robotics applications ............... 5
    1.1.2 Incorporating soft sensors into data fusion systems .......... 6
  1.2 Problem statement ..................................................... 7
  1.3 Search and rescue mission ............................................. 8
    1.3.1 The search and rescue grand challenge problem ............... 8
    1.3.2 Data fusion portion of the search and rescue mission ....... 10
  1.4 Scope of thesis ......................................................... 10
  1.5 Contributions of thesis ............................................... 12
    1.5.1 Contribution to robotics community ............................ 12
    1.5.2 Contribution to fusion community .............................. 13

Chapter 2. Traditional data fusion approaches ............................. 14
  2.1 Data association ....................................................... 14
    2.1.1 Hypothesis generation ......................................... 15
      2.1.1.1 Pattern recognition methods .............................. 16
      2.1.1.2 Gating .................................................... 18
    2.1.2 Hypothesis evaluation ......................................... 21
2.6 UAV surveillance tests ........................................... 59
  2.6.1 Simulation setup ............................................ 59
    2.6.1.1 Range Sensor Model ................................. 60
    2.6.1.2 Camera Model ........................................ 61
  2.6.2 Flight test setup ......................................... 63
    2.6.2.1 Aircraft .............................................. 63
    2.6.2.2 Autopilot ............................................. 64
    2.6.2.3 Airborne processor .................................. 64
    2.6.2.4 Flying field ......................................... 65
  2.6.3 Classification results .................................... 65
    2.6.3.1 Fuzzy logic classifier ............................... 66
    2.6.3.2 Crisp classifier ...................................... 66
    2.6.3.3 Analysis .............................................. 68
  2.6.4 Simulation results ....................................... 70
    2.6.4.1 Terrain mapping ..................................... 70
    2.6.4.2 Target state estimation .............................. 71
  2.6.5 Flight test results ...................................... 74
    2.6.5.1 Single target ....................................... 75
    2.6.5.2 Multiple targets .................................... 75

Chapter 3. Distributed data fusion ..................................... 81
  3.1 Distributed data fusion problem ................................ 82
    3.1.1 Network issues ......................................... 83
    3.1.2 Fusing correlated information .......................... 84
  3.2 Approaches to distributed classification ...................... 87
    3.2.1 Probabilistic approaches ............................... 87
    3.2.2 Dempster-Shafer theory ................................ 91
    3.2.3 Heuristic techniques ................................... 94
  3.3 Approaches to distributed tracking .............................. 95
    3.3.1 Naive fusion ........................................... 96
    3.3.2 Cross-covariance fusion ................................ 97
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3 Information matrix fusion</td>
<td>98</td>
</tr>
<tr>
<td>3.3.4 Maximum a posteriori fusion</td>
<td>99</td>
</tr>
<tr>
<td>3.3.5 Covariance intersection</td>
<td>99</td>
</tr>
<tr>
<td>3.4 Multiple robot collaboration</td>
<td>102</td>
</tr>
<tr>
<td>Chapter 4. Soft sensor data fusion</td>
<td>106</td>
</tr>
<tr>
<td>4.1 Past approaches</td>
<td>106</td>
</tr>
<tr>
<td>4.2 Fuzzy logic for soft sensor data fusion</td>
<td>108</td>
</tr>
<tr>
<td>4.2.1 Soft fusion graphical user interface</td>
<td>108</td>
</tr>
<tr>
<td>4.2.2 Fuzzy logic representation of soft sensor data</td>
<td>110</td>
</tr>
<tr>
<td>4.2.3 Fuzzy logic based fusion</td>
<td>114</td>
</tr>
<tr>
<td>4.3 Soft data fusion tests</td>
<td>119</td>
</tr>
<tr>
<td>4.3.1 Simulations</td>
<td>121</td>
</tr>
<tr>
<td>4.3.2 Field test</td>
<td>123</td>
</tr>
<tr>
<td>Chapter 5. Hard and soft sensor data fusion</td>
<td>130</td>
</tr>
<tr>
<td>5.1 Past approaches</td>
<td>130</td>
</tr>
<tr>
<td>5.2 Random set theory for data fusion</td>
<td>131</td>
</tr>
<tr>
<td>5.2.1 Theoretical basis for random set theory</td>
<td>132</td>
</tr>
<tr>
<td>5.2.2 Formal Bayes modeling</td>
<td>133</td>
</tr>
<tr>
<td>5.2.3 The generalized Bayes filter</td>
<td>136</td>
</tr>
<tr>
<td>5.3 Random set theory uncertainty representations</td>
<td>139</td>
</tr>
<tr>
<td>5.3.1 Random set representation of hard sensor data</td>
<td>139</td>
</tr>
<tr>
<td>5.3.2 Random set representation of soft sensor data</td>
<td>140</td>
</tr>
<tr>
<td>5.4 Processing random set data using particle filters</td>
<td>141</td>
</tr>
<tr>
<td>5.4.1 The generalized particle filter</td>
<td>142</td>
</tr>
<tr>
<td>5.4.2 The hybrid particle filter</td>
<td>143</td>
</tr>
<tr>
<td>5.4.3 The regularized particle filter</td>
<td>147</td>
</tr>
<tr>
<td>5.4.4 The distributed particle filter</td>
<td>149</td>
</tr>
<tr>
<td>5.4.5 The distributed, hybrid, generalized particle filter</td>
<td>153</td>
</tr>
</tbody>
</table>
Chapter 6. Hard and soft sensor fusion simulations .................................. 162

6.1 Simulation one: Single hard sensor, single soft sensor, single target class ......................................................... 162
  6.1.1 Target motion model ...................................................... 163
  6.1.2 Sensor models ............................................................. 166
  6.1.3 Simulation approach .................................................... 172
  6.1.4 Tracking algorithm ..................................................... 174
  6.1.5 Results ........................................................................ 176
    6.1.5.1 Hard sensor only .................................................. 178
    6.1.5.2 Soft sensor only ................................................... 182
    6.1.5.3 Hard and soft sensors ............................................. 185
    6.1.5.4 Analysis ............................................................... 189

6.2 Simulation two: multiple target classes ........................................... 195
  6.2.1 Target motion models .................................................. 196
  6.2.2 Sensor measurement models ......................................... 197
  6.2.3 Tracking algorithm ..................................................... 198
  6.2.4 Results ........................................................................ 200

6.3 Simulation three: distributed hard and soft sensors ......................... 215
  6.3.1 Target motion models .................................................. 217
  6.3.2 Sensor measurement models ......................................... 219
  6.3.3 Tracking algorithm ..................................................... 222
  6.3.4 Results ........................................................................ 228

6.4 Conclusions ..................................................................... 243

Chapter 7. Hard and soft sensor fusion hardware demonstration ............. 245

7.1 Fusion algorithm .............................................................. 246
  7.1.1 Motion models ............................................................ 246
  7.1.2 Sensor processing ....................................................... 249

7.2 Test setup ....................................................................... 251

7.3 Test results ..................................................................... 254
  7.3.1 UGV only ................................................................. 255
7.3.2 Soft sensor only ........................................ 255
7.3.3 Combined soft sensor and UGV .............. 258
7.3.4 Flight test ........................................... 261
7.4 Conclusions ............................................. 273

Chapter 8. Conclusions ........................................ 274
  8.1 Future work ........................................... 276

Appendix A. State estimation ................................. 279
  A.1 Bayes filter ........................................... 279
  A.2 Linear Kalman filters ............................... 280
  A.3 Extended Kalman filter ......................... 282
  A.4 Particle filter ....................................... 282

Appendix B. Fuzzy logic ....................................... 288
  B.1 Fuzzy sets ........................................... 288
  B.2 Fuzzy logic operations ......................... 290
  B.3 Fuzzy mapping rules ........................... 293

Appendix C. Random set theory ............................. 295
  C.1 Introduction to random set theory .............. 295
  C.2 Random set uncertainty representations ....... 295
    C.2.1 Fuzzy sets ................................... 296
    C.2.2 Dempster-Shafer events ..................... 296
    C.2.3 Fuzzy Dempster-Shafer Theory ............. 297
  C.3 Random set likelihoods ............................ 298
    C.3.1 Statistical .................................. 299
    C.3.2 Fuzzy ......................................... 299
    C.3.3 Dempster-Shafer ............................. 299
    C.3.4 Fuzzy Dempster-Shafer ...................... 299
  C.4 Bayesian data fusion ............................... 300
    C.4.1 Dempster’s Combination ...................... 300
C.4.2  Copula fuzzy conjunctions .......................... 301
C.5  Bayes-invariant conversions .......................... 303
  C.5.1  Fuzzy to fuzzy Dempster-Shafer .................. 303
  C.5.2  Fuzzy Dempster-Shafer to fuzzy .................. 304
  C.5.3  Fuzzy Dempster-Shafer to probabilistic ........... 304
  C.5.4  Probabilistic to fuzzy .......................... 305

Bibliography .................................................. 306
List of Tables

2.1 SIG Kadet Specifications ............................................. 64
2.2 Accuracy of each classifier ........................................... 70
2.3 Comparison of the two filters ......................................... 74

4.1 Numeric values corresponding to button presses ................. 113
4.2 Class membership function values for several user inputs .... 113
4.3 FFN membership function values for several user inputs ...... 113
4.4 Errors in localization for soft data test (coordinates are in east, north reference frame) ........................................... 129

5.1 Example discrete hard sensor likelihood table ..................... 139

6.1 Attributes reported by the soft sensor, along with possible values for each attribute ....................................................... 167
6.2 Values for “erratic” membership function .......................... 171
6.3 Performance of various tracking scenarios .......................... 194
6.4 Time to correctly classify the erratic trajectory ................... 195
6.5 Values for classification membership functions .................. 198
6.6 Performance of tracker on each trajectory; each tracker used 1000 particles; results were averaged over 100 runs .................. 215
6.7 Fuzzy membership function values for friend, foe, and neutral classification ................................................................. 222
6.8 RMS errors in north and east position and velocity for simulation 3, averaged over 48 runs, units are meters and meters per second .......... 243
List of Figures

2.1 Assignment matrix created by the hypothesis generation step . . . . . . 16
2.2 Gate which takes target maneuvering into account. (Figure 4-3 from
Blackman [1].) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.3 Distance matrix created by the hypothesis evaluation step . . . . . . . 21
2.4 Crisp classifier based on color filtering . . . . . . . . . . . . . . . . . . . 45
2.5 CINet for identifying a red barrel . . . . . . . . . . . . . . . . . . . . . 46
2.6 Fuzzy membership functions . . . . . . . . . . . . . . . . . . . . . . . . . 47
2.7 SIG Kadet Senior UAV . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
2.8 Model aircraft flying field . . . . . . . . . . . . . . . . . . . . . . . . . . 65
2.9 Image processing results . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
2.10 Image processed by the crisp classifier with the red ball and barrel cor-
rectly identified and the orange windsock incorrectly identified . . . . . 68
2.11 Image processed by the crisp blue tarp classifier . . . . . . . . . . . . . . 69
2.12 Comparison of the two terrain maps (units are meters relative to the
reference origin) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
2.13 Time history of EKF errors . . . . . . . . . . . . . . . . . . . . . . . . . 72
2.14 Time history of particle distribution (units are degrees latitude and lon-
gitude) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
2.15 Final particle distribution after re-sampling . . . . . . . . . . . . . . . . 74
2.16 Errors in the North and East direction for the flight test of the moving
target EKF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
2.17 Plot of barrel locations (large red circles), red ball location (red ‘x’), and
geolocation results (dots) before and after running k-means . . . . . . 79
2.18 Plot of barrel locations (red stars), red ball (red ‘x’), and location esti-
mates as determined by the tracker with data association (blue circles) . 80
3.1 Network architectures for distributed data fusion [2] . . . . . . . . . . 85
3.2 Information graph for hierarchical fusion without feedback [2] . . . . . 88
6.2 Maneuvering vehicle trajectory, with an ‘x’ at the sensor location . . . . 164
6.3 Erratic vehicle trajectory and speed ................................. 165
6.4 Fuzzy membership functions for “near,” “medium,” and “far” distances 168
6.5 Fuzzy membership functions formed through applying complement and hedges to “near” ......................................................... 169
6.6 Fuzzy membership functions for turning left and turning right ......... 170
6.7 Root mean square (RMS) position and velocity errors for 100 runs of simulation one with constant velocity trajectory, hard sensor only, and 1200 particles (units are meters and meters per second) .............. 179
6.8 Root mean square (RMS) position and velocity errors for 100 runs of simulation one with maneuver trajectory, hard sensor only, and 1575 particles (units are meters and meters per second) .................. 180
6.9 Average number of particles in each mode for 100 runs of simulation one with maneuver trajectory, hard sensor only, and 1575 particles .......... 181
6.10 Root mean square (RMS) position and velocity errors for 100 runs of simulation one with erratic trajectory, hard sensor only, and 1500 particles (units are meters and meters per second) ......................... 183
6.11 Average number of particles in each mode for 100 runs of simulation one with erratic trajectory, hard sensor only, and 1500 particles ........ 184
6.12 Root mean square (RMS) position and velocity errors for 100 runs of simulation one with erratic trajectory, soft sensor only, and 10,000 particles (units are meters and meters per second) ..................... 186
6.13 Average number of particles in each mode for 100 runs of simulation one with erratic trajectory, soft sensor only, and 10,000 particles .......... 187
6.14 Time history of particle distribution, along with a red ‘x’ at the actual target location, for simulation one with erratic trajectory, hard and soft sensors, and 1500 particles (units are meters) ......................... 188
6.15 Position and velocity errors, with one sigma contours, for a single run of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles (units are meters and meters per second) ............ 190
6.16 Number of particles in each mode for a single run of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles

6.17 Root mean square (RMS) position and velocity errors for 100 runs of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles (units are meters and meters per second)

6.18 Average number of particles in each mode for 100 runs of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles

6.19 Trajectory, heading and speed for the tank simulation

6.20 Root mean square (RMS) position and velocity errors for 100 runs of simulation two with tank trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)

6.21 Average number of particles in each mode for 100 runs of simulation two with tank trajectory, hard and soft sensors, and 1000 particles

6.22 Trajectory and speed for the motorcycle simulation

6.23 Root mean square (RMS) position and velocity errors for 100 runs of simulation two with motorcycle trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)

6.24 Average number of particles in each mode for 100 runs of simulation two with motorcycle trajectory, hard and soft sensors, and 1000 particles

6.25 Normal person trajectory with a red ‘x’ at the sensor location (units are meters)

6.26 Root mean square (RMS) position and velocity errors for 100 runs of simulation two with normal person trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)

6.27 Average number of particles in each mode for 100 runs of simulation two with normal person trajectory, hard and soft sensors, and 1000 particles

6.28 Trajectory and heading for the lost person simulation

6.29 Root mean square (RMS) position and velocity errors for 100 runs of simulation two with lost person trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)
6.30 Average number of particles in each mode for 100 runs of simulation two
with lost person trajectory, hard and soft sensors, and 1000 particles . . 214
6.31 Sensor locations and viewpoints (all units are meters) . . . . . . . 216
6.32 Normal vehicle trajectory, along with sensor locations with a blue ‘x’ for
a UAV, a red ‘*’ for a UGV, and a green ‘+’ for a soft sensor (all units
are meters). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229
6.33 Single sensor position and velocity errors (blue) along with one sigma
contour (dashed red), for normal trajectory with 1000 particles. . . . . 231
6.34 Single sensor motion regime and classification weights, for normal tra-
jectory with 1000 particles. . . . . . . . . . . . . . . . . . . . . . . . . . . . 232
6.35 RMS position and velocity errors (blue) along with one sigma contour
(dashed red), averaged over 48 runs, for normal trajectory with 1000
particles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 233
6.36 RMS position and velocity errors (blue) along with one sigma contour
(dashed red) for normal trajectory with 1000 particles, with outlier re-
moved. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235
6.37 Average motion regime and classification weights, averaged over 48 runs,
for normal trajectory with 1000 particles. . . . . . . . . . . . . . . . . . 236
6.38 Maneuver vehicle trajectory, along with sensor locations with a blue ‘x’
for a UAV, a red ‘*’ for a UGV, and a green ‘+’ for a soft sensor (all
units are meters). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
6.39 RMS position and velocity errors (blue) along with one sigma contour
(dashed red), averaged over 48 runs, for maneuver trajectory with 1000
particles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238
6.40 Average motion regime and classification weights, averaged over 48 runs,
for maneuver trajectory with 1000 particles. . . . . . . . . . . . . . . . . 239
6.41 Erratic vehicle trajectory, along with sensor locations with a blue ‘x’ for
a UAV, a red ‘*’ for a UGV, and a green ‘+’ for a soft sensor (all units
are meters). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
6.42 RMS position and velocity errors (blue) along with one sigma contour (dashed red), averaged over 48 runs, for erratic trajectory with 1000 particles.

6.43 Average motion regime and classification weights, averaged over 48 runs, for erratic trajectory with 1000 particles.

7.1 MMP-5 mobile robot

7.2 Target correctly classified by the UGV

7.3 Errors in the north and east direction for test with UGV only

7.4 Weights of the CP and IOU regimes for the UGV

7.5 Errors in the north and east direction for test with soft sensor only

7.6 Weights for motion regimes and classifications for the soft sensor only

7.7 Errors in the north and east direction for the soft sensor in the combined UGV and soft sensor test

7.8 Weights for motion regimes and classifications for the soft sensor in the combined UGV and soft sensor test

7.9 Errors in the north and east direction for the UGV in the combined UGV and soft sensor test

7.10 Weights for motion regimes and classifications for the UGV in the combined UGV and soft sensor test

7.11 Target correctly classified by the UAV

7.12 Errors in the north and east direction for the soft sensor in the flight test

7.13 Errors in the north and east direction for the UAV in the flight test

7.14 Errors in the north and east direction for the UGV in the flight test

7.15 Weights for motion regimes and classifications for the soft sensor in the flight test

7.16 Weights for motion regimes and classifications for the UAV in the flight test

7.17 Weights for motion regimes and classifications for the UGV in the flight test
B.1 Fuzzy membership functions ........................................... 291
List of Algorithms

2.1 Iterative procedure for determining a target’s location .................. 49
2.2 Algorithm for the terrain map ............................................. 51
2.3 Procedure for initializing the static target particle filter ................. 55
2.4 Static particle filter algorithm ............................................. 56
4.1 Fuzzy logic data association algorithm ................................... 118
4.2 Algorithm to compare the states of two tracks using fuzzy logic ...... 118
4.3 Algorithm to fuse two tracks .............................................. 119
5.1 The generalized particle filter ............................................ 144
5.2 The hybrid particle filter [7] .............................................. 146
5.3 Regime transition algorithm [7] .......................................... 147
5.4 Regularized particle filter algorithm [7] ................................ 150
5.5 Salmond’s joining algorithm [8] .......................................... 154
5.6 Algorithm for the fusion of Gaussian mixtures by covariance intersection [9] 156
5.7 Algorithm to update particles in the generalized, hybrid particle filter .. 158
5.8 Mode conditioned regularized resampling algorithm ................... 160
5.9 Algorithm to generate mode conditioned Gaussian mixtures ........... 160
5.10 Algorithm to fuse mode conditioned Gaussian mixtures ............. 161
5.11 Algorithm to sample from mode conditioned Gaussian mixtures .... 161
6.1 Particle filter for simulation one ........................................ 177
6.2 Algorithm to initialize filter from a camera measurement ............. 223
6.3 Algorithm to initialize filter from a stereo camera measurement ..... 224
6.4 Algorithm to initialize the regimes of particles ........................ 225
6.5 Algorithm to initialize filter from a soft sensor measurement .......... 226
7.1 Algorithm to initialize filter from a UGV camera measurement ....... 250
7.2 Algorithm to initialize the regimes of particles ........................ 250
7.3 Algorithm to initialize filter from a UAV camera measurement ....... 252
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>Algorithm to initialize filter from data entered into the GUI by a human operator</td>
<td>253</td>
</tr>
<tr>
<td>7.5</td>
<td>Algorithm to initialize the regimes of particles from a soft sensor measurement</td>
<td>253</td>
</tr>
<tr>
<td>A.1</td>
<td>The recursive Bayes filter [10]</td>
<td>280</td>
</tr>
<tr>
<td>A.2</td>
<td>The Kalman filter [10]</td>
<td>283</td>
</tr>
<tr>
<td>A.3</td>
<td>Information form of Kalman filter [10,11]</td>
<td>284</td>
</tr>
<tr>
<td>A.5</td>
<td>The particle filter [10]</td>
<td>287</td>
</tr>
</tbody>
</table>
Nomenclature

\((o_x, o_y)\) Pixel location of image center

\((x_{im}, y_{im})\) Point in pixel coordinates

\(\alpha_{PS}\) Dempster-Shafer agreement

\(\delta_k\) Kronecker delta function

\(\eta(x)\) An unambiguously generated ambiguous observation of the state \(x\)

\(\gamma(x_{nk})\) Gaussian mixture model responsibility parameter

\(\hat{x}_k^+\) Kalman filter \textit{a posteriori} state estimate

\(\hat{x}_k^-\) Kalman filter \textit{a priori} state estimate

\(\hat{y}\) Predicted measurement from a sensor

\(M_{\text{int}}\) Camera intrinsic parameter matrix

\(R\) Rotation matrix

\(I\) Information matrix

\(\mu_A(x)\) Fuzzy membership function (degree to which \(x\) belongs to fuzzy set \(A\))

\(\mu_k\) Gaussian mixture model component mean

\(\pi_k\) Gaussian mixture model component mixing coefficient

\(\Sigma_A(\mu)\) The random set representation of the fuzzy membership function \(\mu\)

\(\Sigma_k\) Gaussian mixture model component covariance

\(\Sigma_m\) The random set representation of the Dempster-Shafer basic mass assignment \(m(A)\)

\(\sigma_o^2\) Observation variance
\( \sigma^2_p \)  
Predication variance

\( \sigma_r \)  
Residual standard deviation

\( \hat{y} \)  
Kalman filter innovation vector

\( \xi \)  
Information vector

\( \{C_k\} \)  
The set of classes or clusters which a point can belong to in a pattern recognition problem

\( a_1 \land_{A_1, A_2} a_2 \)  
The copula of \( A_1 \) and \( A_2 \)

\( d_{G_{i,j}} \)  
Probabilistic distance between two points

\( E(f(x)) \)  
Expected value of \( f(x) \)

\( f \)  
Camera focal length

\( h_{\text{terrain}} \)  
Terrain elevation

\( h_{\text{UAV}} \)  
UAV altitude

\( I_{i,j} \)  
Indicator function for items \( i \) and \( j \)

\( K_{Gl} \)  
Gating constant

\( p(x_k|Y_{k-1}) \)  
Bayes filter \textit{a priori} pdf

\( p(x_k|Y_k) \)  
Bayes filter \textit{a posteriori} pdf

\( P^+_k \)  
Kalman filter \textit{a posteriori} covariance matrix

\( P^-_k \)  
Kalman filter \textit{a priori} covariance matrix

\( Q_k \)  
Kalman filter process noise covariance matrix

\( R_k \)  
Kalman filter measurement noise covariance matrix

\( s(x, y) \)  
Fuzzy s-norm (disjunction)

\( S \)  
Kalman filter residual covariance matrix
\( s_x \) Size of a pixel in the horizontal direction

\( s_y \) Size of a pixel in the vertical direction

\( t(x, y) \) Fuzzy t-norm (conjunction)

\( u_k \) The control vector at time \( k \)

\( U_k \) The set of all controls up till time \( k \)

\( x \) A state vector or data point

\( X_k \) The set of all states up till time \( k \)

\( y \) Measurement from a sensor

\( Y_k \) The set of all measurements up till time \( k \)

\( \text{Bel} \) Dempster-Shafer belief function

\( m(A) \) Dempster-Shafer basic mass assignment

\( \text{Pl} \) Dempster-Shafer plausibility function

\( Q \) Dempster-Shafer commonality function

\( ABC \) Aircraft-body coordinates

\( NED \) North-east-down coordinates

\( \epsilon \) Elevation angle

\( \eta \) Normalizing constant

\( \phi_r \) UAV roll

\( \psi_y \) UAV yaw

\( \theta_p \) UAV pitch

\( \varphi \) Azimuth angle
I am indebted to a number of people who have helped with this thesis in various ways.

I would like to thank my adviser Dr. Lyle Long for all of his advice and support throughout my time at Penn State.

I would like to thank my committee members Dr. Ken Jenkins, Dr. David Miller, Dr. David Hall, Dr. John Yen, and Dr. Joe Horn for all of the comments and enthusiasm.

Also, I have received a great deal of help from the entire Penn State UAV Team: former students Brian Geiger, James Ross, and Eric Schmidt; current students Mark DeAngelo and Sean Quinn Marlow; faculty advisers Dr. Lyle Long, Dr. Joe Horn, and Dr. Jack Langelaan; and Al Niessner and Mike Roeckel from the Applied Research Lab. I would like to thank Al in particular for all of his help keeping the UAVs flying and for serving as our safety pilot during flight tests.

I am also grateful for the financial support that I have received from the Penn State Applied Research Lab during my time as a graduate student.

Last, but certainly not least, I would like to thank my family for all of their support throughout my graduate program. I am especially grateful for my parents for all of their encouragement and for my wife Alissa, who has been right there beside me this whole time. I truly could not have done this without her.
Chapter 1

Introduction

Unmanned vehicles are used everywhere for diverse applications such as finding missing persons, disaster relief, patrolling borders, disposing of improvised explosive devices, and protecting convoys moving through hostile areas. Despite the accomplishments of unmanned vehicles in recent years, almost all unmanned systems share the same limitation: they need constant human intervention in order to accomplish their missions. It would be much more useful if unmanned vehicles could become more autonomous. According to Bekey [12]:

“Autonomy refers to systems capable of operating in the real-world environment without any form of external control for extended periods of time.”

By increasing the autonomy of unmanned vehicles, it would be possible to free up operators from engaging in monotonous tasks and allow them to do things they are better suited for. It would also allow fewer operators to control more vehicles because the time they spend controlling any one vehicle would be less. Autonomy is a distinct property from intelligence, which is also important. Gottfredson [13] defines intelligence as follows:

“Intelligence is a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience.”

An unmanned vehicle which is both intelligent and autonomous would be ideal, because operators could simply give it a mission, then allow the vehicle to reason and how plan to best accomplish this mission. In such a setting, unmanned vehicles would shift from being used as tools to being used as teammates. People and robots would share information with one another so that they could work together to accomplish a given goal.

The first step in moving toward more capable unmanned vehicles is to create an accurate model of the world that the vehicles are operating in so that their controllers will
have the data they need to make intelligent decisions. The vehicles’ worldview is formed using data from a number of sensors. Each of these sensors provides measurements that are noisy, incomplete, and potentially erroneous. Additionally, different sensors may provide conflicting data. Another problem comes when different vehicles in a team have different worldviews, which must be fused in order to facilitate teamwork. When human beings (soft sensors) are added to the team, the problem becomes even more difficult. Soft sensors do not provide the same type of information as hard (physics-based) sensors do. Information from them is usually fuzzy in nature and must be handled differently.

Sensing is an extremely important aspect of any autonomous system. Without sensing, the system will have no perception of the world around it and cannot possibly operate autonomously. In a recent workshop on cyber-physical systems (systems in which computational and physical components are tightly integrated) a position paper by Atkins [14] lists several challenges for cyber-physical systems in the aerospace domain. One such challenge is that:

“Perceptual capabilities of cyber-physical systems are limited to the specific sensor signals and processing algorithms possessed by these systems. How can a distributed cyber-physical team develop, adapt, and share an accurate world model of the environment and of each other? How can agents reliably detect anomalies, and how can a team repair an agent or compensate for its absence when a debilitating failure occurs?”

Although all of these problems are very important, this work focuses on the problem of the team developing, adapting, and sharing an accurate world model. This problem is foundational, because without an accurate shared world model, the agents will not be able to cooperate with one another effectively. Once this problem is fully solved, other issues such as cooperative behavior and recovery from failures will become possible to solve.
1.1 Background

This work focuses on techniques for performing distributed data fusion across multiple unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), and human observers (soft sensors). A great deal of research has focused on centralized data fusion [15–19]. In centralized fusion, one highly capable computer processes data from multiple sensors in order to perform tasks such as classifying targets, building a world map, and tracking moving targets. Applications of centralized fusion include ballistic missile defense [15], robotics [16,17], and structural health monitoring systems [18]. Techniques for doing so include neural networks [16], fuzzy logic [18], and Kalman filtering (and its variations) [17].

Data fusion problems have historically been modeled by the Joint Directors Laboratories (JDL) Model [19], which divides fusion problems into five levels:

- Level 0: Source Preprocessing/Subobject Refinement
- Level 1: Object Refinement
- Level 2: Situational Refinement
- Level 3: Impact Assessment (or Threat Refinement)
- Level 4: Process Refinement
- Level 5: Cognitive Refinement (or User Refinement)

These levels need not occur in order, nor do all of them have to be present in all data fusion systems. They do, however, represent a hierarchy that starts with raw sensor data and moves up to information which is meant to be used by some “consumer of information” (such as a commander on a battlefield). There is often overlap between the different levels, which means that some fusion methods are difficult to place into a single level. Therefore, this work does not often speak of a certain level, unless it is helpful to do so, but it often distinguishes between low-level data (levels 0-2) and high-level information (levels 3-5).
The data fusion problem becomes more difficult when sensors are located on multiple platforms (such as unmanned vehicles). Here issues that have to be dealt with include limited processing power onboard any one vehicle and limited communications bandwidth between different vehicles. Another more subtle problem arises due to the fact that the standard Kalman filter assumes different sensors provide independent information. This is a reasonable assumption in a standard centralized architecture, where all sensors independently send information to a centralized processor. It can be a problem, however, in a distributed architecture where information can flow in complex cyclic graphs.

There are several methods in the literature for performing distributed data fusion [3, 20–23]. Studies have mostly focused on distributed classification or distributed tracking. Classification is dealt with in [22, 23], which frame distributed classification as a statistical optimization problem. Two of the methods for distributed tracking are information matrix fusion [21] and covariance intersection [3]. It is also possible to combine the two methods [20]. These techniques have all performed well in simulation, but there have been few attempts to field them on a real world system.

The difficulty of the problem is increased when human agents (soft sensors) become involved. There have been some early attempts at hard sensor soft sensor fusion [4, 19, 24–27] that have shown promise, but there is still a great deal of research to do. Hard sensors are best modeled statistically, but this is difficult to accomplish with soft sensors. First of all, it is not always natural for a person to express ideas statistically, and even if it were, people are often bad at estimating probabilities. Therefore fuzzy logic [28–30] is often a more natural choice for dealing with information from soft sensors. It is much easier for a person to say that an object is small than it is for him to say that it is 1 meter in size with a standard deviation of 0.1 meters. Therefore, most attempts so far at fusing data from hard and soft sensors have focused on the mathematics of fusing fuzzy information with statistical information [24, 25].
1.1.1 Distributed data fusion for robotics applications

Traditionally, the type of sensor processing done onboard robots falls into the categories of localization (determining where the robot is in the world), mapping (making a map of the world which the robot is operating in), or simultaneous localization and mapping (SLAM) [11]. The most common example of localization is determining a robot’s position by triangulating its position using ranges from a number of beacons (such as GPS satellites) [31]. A common example of mapping is the occupancy grid, where a robot makes a map of its environment by marking space as either occupied or unoccupied [32]. (Note that in this type of map no other high-level detail is collected, space is either occupied or unoccupied.) In SLAM the robot must both determine its location and determine the locations of features in the environment relative to itself [33, 34]. Some SLAM approaches utilize high-level features in the map, but many are still geared toward simple occupancy-style maps. All of these approaches are probabilistic in nature and utilize some form of a Bayes Filter (see Appendix A).

The most important limitation of these methods is that they were almost all designed for low level features. In the case of occupancy grid mapping, grid cells are either occupied or unoccupied; no information is collected about what a grid cell is occupied by. In SLAM applications, there is more emphasis on features such as corners or edges, but still not on high-level properties. There are some SLAM implementations that utilize computer vision methods such as the scale invariant feature transformation (SIFT) to identify important features [34], but SIFT is still limited to identifying features that have been seen before. This focus on low level features is valuable for robot navigation, but for use in a truly intelligent system or use by a human operator, high level properties are important. Ideally, a robot should be able to classify the objects it sees, determine their velocity if they are moving, and at a very high level use this information to reason about its environment.

Recently, interest has shifted from single-robot problems to distributing localization, mapping, and SLAM across several platforms. This is an important problem because without sharing information, vehicles cannot cooperate. It is a difficult problem due to resource constraints. Sharing an entire occupancy grid map, for example,
utilizes a great deal of communication resources. Fusing an occupancy grid map from another robot with one's own would require a great deal of computational resources. Even when maps are feature based, data association is still a problem. Traditional SLAM approaches based on the extended Kalman filter do not work well when there are many robots in a team, so alternative methods such as the sparse extended information filter are used [35]. Distributed robotics problems become more difficult when robot teams are heterogeneous (possessing different sensing capabilities) [36]. In this case, two robots could have trouble sharing information because their sensors are too different to find correspondences between the features.

All of the traditional robotics approaches mentioned so far have been focused on using one very capable sensor (such as a range finder or stereo camera) or a small number of similar sensors. As robot sensing becomes more sophisticated and many different types of sensors such as tactile, chemical, or sound are used, sensor fusion becomes very important. In a recent issue of Aerospace America [37] Roland Menassa, Manager of Advanced Robotics at General Motors states,

“In robotics today, you find a lot of talk about vision and sensors. But the challenge is sensor fusion. Humans use many different sensors – when vision is occluded you can rely on touch.”

The field of multisensor data fusion [19] contains a wealth of algorithms for collecting data from multiple sensors, combining the data, and ultimately extracting high-level information which can be used by human decision makers or an intelligent system. There has also been research done on distributed data fusion [23, 38] that shows a great deal of promise, but these algorithms are less well-developed than the centralized fusion algorithms.

1.1.2 Incorporating soft sensors into data fusion systems

Traditionally, most data fusion systems have focused on fusing hard sensor information (such as data from a camera, radar, laser, etc.). The field of soft sensor data fusion is much newer. It deals with information from sources such as domain experts, human observers on the ground, and information published in newspapers and online [27].
Recently there has been a great deal of interest in fusing hard information with soft information [4,19,24–27,39]. It is obvious why doing so is important: it will make more information available to those who need it. Unfortunately, the hard/soft data fusion problem is a difficult one for which an optimal solution has not yet been found.

Most attempts at hard/soft fusion have focused on high-level battlefield situational awareness [40,41] or counter terrorism [26]. There have been few (if any) applications of hard/soft fusion to teams of autonomous robots. Incorporating soft sensor information into a robot’s worldview will give it more information, which should allow it to make better decisions. Some kinds of information that would be very difficult to gather with hard sensors can easily be gathered by soft sensors. For instance, by interviewing locals, it may be possible to determine that hostile forces were recently seen in a given area. This is valuable information for the robot in its planning, but it would be nearly impossible for the robot to gather this information using its own sensors.

There has also been a great deal of interest in moving from autonomous robots (robots which are capable of operation without external control) to intelligent robots (those which can reason to solve problems and learn from experience) and ultimately to conscious robots (robots which are self-aware) [42]. A robot which only possessed low level information from its sensors could be considered autonomous (if it is able to use this sensor data in order to operate without commands from some operator), but a truly intelligent robot would have to be able to extract higher level type information from its own sensors and combine that with high level information from other robots and from people. For instance, if a robot sees an unfamiliar vehicle, and the operator informs the robot that the vehicle is a hostile tank, an intelligent robot will be able to learn from this experience how to identify a hostile tank.

1.2 Problem statement

Based on the previous section, it is clear that there is a need in the robotics domain for more sophisticated data fusion. Data fusion is important for one robot with many sensors, but it is even more important for heterogeneous teams of robots and is essential for human-robot teams. This thesis describes the creation of a data fusion system
that is capable of running onboard multiple small heterogeneous unmanned vehicles and interacts with human observers (soft sensors). This system possesses several important capabilities. These capabilities include the ability to classify and locate static targets, the ability to track moving targets, and the ability to incorporate information from human observers into the classification, localization, and tracking. The fusion system is also a truly distributed system, i.e. it is able to dynamically allocate communication, processing, and sensing across the entire vehicle swarm. These capabilities are demonstrated through a combination of simulations and field tests.

1.3 Search and rescue mission

This section introduces a “grand challenge” problem to demonstrate the power of autonomy in a real world situation. The first subsection introduces this problem: multiple-vehicle, multiple-soft sensor, autonomous, collaborative search and rescue. Because this is a broad problem dealing with many issues outside the scope of this thesis, including: optimal search, path planning and obstacle avoidance, the second subsection describes the narrower sub-problem addressed in this thesis.

1.3.1 The search and rescue grand challenge problem

The Penn State Applied Intelligent Systems Lab was originally founded as a collaboration between the Penn State Applied Research Lab (ARL) and the Penn State Aerospace Engineering Department to study autonomy, intelligence, and collaboration onboard unmanned vehicles [43, 44]. Early work focused on unmanned aerial vehicles (UAVs), but the work has since been extended to unmanned ground vehicles (UGVs), and “soft” human sensors. The lab is a diverse group with expertise in areas such as intelligent control [45–47], data fusion [48], computer vision [49–51], trajectory optimization [52–58], and obstacle avoidance [59,60]. In order to demonstrate the group’s many capabilities in a single demonstration, a “grand challenge” problem has been developed: searching for and rescuing a missing pilot using unmanned aerial vehicles, unmanned ground vehicles, and human searchers, all while detecting and avoiding hostile forces.
Specifically, the experiment mimics finding a downed pilot in a hostile area. In general, finding a person from the air is a difficult computer vision problem. To avoid some computer vision difficulties, some simplifying assumptions have been made. Namely, the pilot will be able to use flares, or some other means of identifying himself. (This could be as simple as using a red ball to simulate a flare.) Searching for flares (or a ball) makes the computer vision problem tractable, and allows this demonstration to focus on data fusion and autonomy issues. If more sophisticated vision methods are latter required, they can easily be added. (See Hanford et al. [61] for an example robotics problem using advanced computer vision methods.) Also, to make the experiment easier to set up, it will be conducted at a model aircraft flying field, rather than in a forest. The field already contains a number of obstacles, including buildings and trees, that a ground vehicle would have to navigate. Additional obstacles can be created by parking cars in its path or using a blue tarp to simulate a body of water.

Everything listed so far can easily be accomplished by a robot-only team. The addition of hostile forces to the problem makes it significantly more difficult and necessitates the addition of human personal. A UAV can easily detect cars, but it is much harder for a UAV to determine that a car is driving erratically, and therefore is acting in a suspicious manner. A ground vehicle would be able to find people (a UAV could as well with a quality camera), but the ground vehicle would have trouble identifying suspicious behavior. A trained person should have significantly less trouble with such tasks. He should not only be able to identify suspicious activity, he should be able to quantify how suspicious it is (i.e. somewhat suspicious, suspicious, and very suspicious). This information would be difficult for the robots to collect, but would be vital in their mission. In the case of a search and rescue mission, the ground robot would seek to avoid a suspicious group of people, but the UAV may seek to keep track of where they are. In the case of this particular mission, vehicles and people not associated with the search party will be used to simulate hostile forces. It will be up to the human observer to classify any people that he or the robot sees as either friendly, hostile, or neutral and to associate a confidence with the classification.
Combining these items gives the following tractable, but still realistic scenario:
One UAV, equipped with a camera and GPS, will search for a flare (or simulated flare).
A UGV, equipped with a camera, GPS, and LIDAR (or stereo camera) and a human
observer will simultaneously search on the ground for the pilot himself, while avoiding
hostile forces and other obstacles. Also, another person will observe video feeds from
the vehicles to classify items that the vehicles were unable to classify. In other words,
the person and the vehicle will form a hybrid cognition system [39] that has both hard
and soft aspects. Once the pilot is found, the UAV will provide surveillance, while the
UGV and person converge on the target, while avoiding contact with any hostile forces.
During the course of this entire experiment, sensor reports, human observer reports, and
ground truth data will be collected so that the entire data set can be made available
online.

1.3.2 Data fusion portion of the search and rescue mission

There are many facets to the search and rescue problem discussed in the previous
section. This thesis focuses nearly exclusively on the data fusion aspects of the problem.
As such, the central focus is to develop algorithms capable of running on autonomous
vehicles, which allow them to gather data from their sensors and fuse that data with
information from other vehicles and human observers, in order to form a coherent view
of their world. From this worldview, vehicle intelligent controllers and path planners can
work to find the optimal way to find the missing pilot while avoiding hostile forces and
other obstacles. The field tests and simulations in this thesis seek to address various
aspects of this problem, including: single UAV search, multiple robot search, soft sensor
only data fusion, and combined hard and soft sensor data fusion. These tests cumulate in
a demonstration of finding a target using a UAV, a UGV, and a soft sensor in Chapter 7.

1.4 Scope of thesis

The previous sections outlined the main problem that this research addresses,
i.e. the need for more sophisticated data fusion in robotics applications. This is a very
broad problem. This section outlines the specific issues this thesis addresses and how
they contribute to the state of the art in both the robotics community and the fusion community. Where appropriate, it also describes important issues that are left for future work.

The first issue this thesis addressed here is that of centralized data fusion (Chapter 2). This is a foundational data fusion issue, which has been very well studied. Important issues in centralized fusion include data association [1,19,62–64], classification [29,65–71], tracking [1,3,7,20–23,63,64,72–75], and robotics problems [11,32,34,69,76–78]. Because these problems are well studied, this thesis does not make any major contributions in this area, but does review it for completeness. The chapter on centralized fusion concludes with an example UAV surveillance problem which demonstrates many of the traditional fusion methods (Sections 2.5 and 2.6).

The second problem addressed here is that of distributed data fusion (Chapter 3). Distributed data fusion is a younger field than centralized fusion. The main issues in distributed fusion that are addressed are distributed classification [19,79,80] and distributed tracking [2,3,21,81,82]. There are several well established methods for performing distributed fusion that are first reviewed. Then, a multiple vehicle collaborative test involving an unmanned aerial vehicle and an unmanned ground vehicle is presented (Section 3.4).

The third issue this thesis addresses is that of soft sensor only data fusion (Chapter 4). Soft-only fusion is a newer field than centralized or distributed hard sensor fusion, but there are still some very promising early results in the field [4,19,24–27,83]. Because of the novelty of the field, the chapter on soft only fusion contains less background information and instead focuses on some new results in soft sensor data fusion. Specifically, it discusses a novel fuzzy logic based soft sensor data fusion method (Section 4.2). This new method is tested in both simulation and in the field (Section 4.3).

The major issue addressed in this thesis is that of combined hard and soft fusion (Chapter 5). Fusion of hard and soft data is the newest area for research in the data fusion community [24,25,27,39,84]; therefore, some of the largest theoretical contributions in this thesis are in this chapter. This chapter presents a novel combined hard and soft data fusion method based on random set theory [5,25,85–88], which processes random set
data using a particle filter. Furthermore, the particle filter is designed to be distributed across multiple robots and portable computers (used by human observers) so that there is no centralized failure point in the system.

After laying out a theoretical groundwork for hard and soft sensor data fusion in the previous chapter, Chapter 6 deals with practical applications for hard and soft sensor data fusion in simulation. Through a series of three progressively more difficult simulations, some important hard and soft sensor data fusion capabilities are demonstrated. The first simulation demonstrates fusing data from a single soft sensor and a single hard sensor in order to track a car that could be driving normally or erratically. The second simulation adds the extra complication of classifying the type of target to the simulation. The third simulation uses multiple hard and soft sensors, with a limited field of view, to track a moving target and classify it as a friend, foe, or neutral.

Chapter 7 builds on the work done in previous chapters by performing a field test of the algorithms that were described in Chapter 5 and tested in simulation in Chapter 6. The test utilizes a UAV, a UGV, and a human observer with a laptop. The test is designed to mimic the search and rescue problem described in Section 1.3. Whereas Chapter 5 contains some of the most important theoretical contributions in this thesis, this chapter makes some of the most important practical contributions by showing that the algorithms that have been developed for hard and soft sensor data fusion are capable of running in real time on relatively simple hardware.

1.5 Contributions of thesis

This section describes the major contributions of this thesis. Because this work deals with the overlap between the robotics domain and the data fusion domain, significant contributions have been made to both communities.

1.5.1 Contribution to robotics community

This research contributes to the robotics community, especially in the field of human-robot interaction (HRI) [89,90]. HRI is a multidisciplinary field which studies
how humans and robots can work together to solve problems more effectively than either party can alone. The main HRI problem this thesis seeks to address is how to use high-level information from human sources (soft sensors) in order to aid robots in forming a worldview which can help them to perform their tasks. This is distinct from other important problems, such as how humans and robots should communicate, or how humans and robots should work together to complete a task. Although these problems are all important, the actual fusion problem addressed here (i.e. how does the robot use information from human sources) is of primary importance. This is because a robot with the best communications protocol (such as a robot which can recognize natural language and human gestures) is still not extremely valuable if it cannot use the information it is receiving effectively. Likewise, without an effective fusion system, teamwork between humans and robots is not possible.

1.5.2 Contribution to fusion community

The topic of fusing hard and soft data has generated a great deal of recent interest in the data fusion community [4,19,24–27,39]. The fundamental issue that most studies have dealt with is how to fuse information coming from hard sensors (i.e. cameras, radar, chemical sensors) with data coming from soft sensors (i.e. troops on a battlefield, interviews with locals, information from the Internet) in order to give human decision makers the best possible information. Much work remains to be done on this topic, and the research presented here contributes to the knowledge of this fundamental question. At the same time, this thesis addresses a variation of the fundamental problem; namely, how to fuse data from multiple robots (hard sensors) with the data from humans on the ground (soft sensors) in order to aid decision making not only by a central commander, but by the ground troops and robots themselves. This is a significant contribution because there have been few, if any, studies where the hard sensors are autonomous robots, which are not just sources of information, but also consumers of information.
Chapter 2

Traditional data fusion approaches

Multisensor data fusion is a broad field which utilizes techniques from many different disciplines (such as pattern recognition, estimation theory, and artificial intelligence) to solve problems with diverse application areas (including robotics, military problems, remote sensing, and medical diagnosis). According to Hall and McMullen [19],

“Data fusion involves combining information from multiple sources or sensors to achieve inferences not possible using a single sensor or source.”

It is not feasible to cover the entire data fusion field in one chapter (or even an entire book). However this chapter outlines some important data fusion techniques, especially those which are useful for robotics problems. For a thorough treatment of the subject see Hall and McMullen [19]; for a collection on recent research results see Liggins et al. [91]. Important algorithms include data association techniques, classification techniques, tracking techniques, and techniques developed specifically for robotics applications including localization, mapping, and simultaneous localization and mapping (SLAM). The chapter concludes with an example problem which utilizes many of the algorithms described here.

2.1 Data association

Data association [1, 19, 62–64] is the problem of determining which data from sensors correspond to the same physical item. It is an important problem in classification, tracking, and mapping. If a classifier makes an incorrect data association (such as attempting to fuse a visual image of one item with the radar cross section of a completely different item), incorrect classifications can be made. Likewise, if a tracker incorrectly associates a new sensor reading with the wrong object being tracked, it can lead to poor tracking results. In simultaneous localization and mapping problems, it is important for
a correct data association to be made in order for a robot to “close a loop”, i.e. determine that it has reached a location which it has already visited. The process of data association can be divided into three interconnected steps [19]: hypothesis generation, hypothesis evaluation, and hypothesis selection.

### 2.1.1 Hypothesis generation

The first step in the data association problem is to generate feasible hypotheses (where a hypothesis is a possible assignment of one data item to another), first by enumerating all potential hypotheses, then by identifying which of these hypotheses are feasible. There are many possible ways to structure these hypotheses; Table 3.2 of Hall and McMullen [19], provides a nearly exhaustive list. A few examples that are pertinent for this thesis include report-to-report, report-to-track, track-to-track, and multi-spectral. Report-to-report hypotheses state that two reports from the same sensor (usually received at different times) correspond to the same entity. A report-to-track hypothesis states that a new sensor report corresponds to a given entity which is currently being tracked. Track-to-track hypotheses are similar to report-to-track except that the sensor provides a state vector instead of a raw report. This would be the case if the “sensor” was really another robot in a distributed problem. Finally, multi-spectral hypotheses are generated when two sensors which give different types of data (such as camera and radar) provide reports about the same item (at the same time or at different times).

Once a set of potential hypotheses has been created, it must be pruned to only include those hypotheses which are actually feasible. Here, the term “feasible” is problem dependent, it may be dictated by the rules of physics (such as a sensor report for an airborne target should not be associated with a car, but could be associated with a plane). Feasible may simply mean that highly unlikely associations should be ruled out. Two important groups of techniques include pattern recognition methods (especially clustering) and gating. These two techniques are reviewed below, other methods are listed in Table 3.3 in Hall and McMullen [19]. Figure 2.1 shows an example of the output of the hypothesis generation procedure. The rows of the table represent items
currently being tracked and the columns represent observations received from sensors. A one in a cell represents a feasible hypothesis and an x represents an infeasible hypothesis.

<table>
<thead>
<tr>
<th></th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>T4</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

Fig. 2.1: Assignment matrix created by the hypothesis generation step

2.1.1.1 Pattern recognition methods

When a large group of data points must be partitioned into groups (such as in report-to-report and multi-spectral hypotheses), clustering is often a good choice for finding associations. Clustering is an unsupervised learning technique which seeks to break a dataset into groupings of similar data. The best known clustering technique is the K-means algorithm [92–94]. Given the number of clusters, $K$, and an initial guess of the cluster centers, the K-means algorithm seeks to partition a dataset into $K$ clusters such that the distance from each point to the center of its assigned cluster is minimized. There are several deficiencies of the K-means algorithm. The first is that the algorithm may converge to a local minimum. Therefore, K-means is often run several times with different initializations, and the solution which best partitions the data is accepted. The second problem is that there is no way to optimally choose $K$, it must be chosen by trial and error. The third problem is that K-means performs a “hard” partitioning of the data set, i.e. points are assigned to one and only one cluster.
An extension to the traditional K-means algorithm is the fuzzy K-means algorithm [95]. Fuzzy K-means makes a “soft” partition, so that data points can be assigned to multiple clusters. For each data point, \( x \), a parameter \( \mu_{C_i}(x) \) between zero and one, determines the degree to which point \( x \) belongs to cluster \( C_i \), where \( C_i \) is now a fuzzy set (see Appendix B for more on fuzzy sets). A value of one indicates complete membership and a value of zero indicates no membership.

Another “soft” clustering method is the mixture of Gaussians [71]. A mixture of Gaussians assumes that data points are generated by \( K \) different Gaussian probability distributions. Each component of the probability distribution is defined by three parameters: \( \mu_k \), the mean of \( k \)th Gaussian, \( \Sigma_k \), the covariance of the \( k \)th Gaussian, and \( \pi_k \), the mixing coefficient of the \( k \)th Gaussian. The mixing coefficients represent how likely a new random point will come from a given distribution. They must all sum to one. Given these parameters, it is possible to find the responsibility, \( \gamma(x_{nk}) \), which represents the likelihood that point \( x_n \) was generated by Gaussian \( C_k \) as follows:

\[
\gamma(x_{nk}) = \frac{\pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n | \mu_j, \Sigma_j)}. \tag{2.1}
\]

Now each point, \( x \), is assigned to the class with the highest responsibility, \( \gamma(x_{nk}) \). Fuzzy K-means and mixture of Gaussians both seek to accomplish similar goals (to create a “soft” partition of the data set), but they do so differently and yield different results. Which method works better in a given situation is problem dependent.

Nearest neighbor classifiers are similar to clustering techniques in that they attempt to group similar data points together, but they are a supervised technique that use labeled data points rather than unlabeled points. The labels often come from a priori knowledge. For instance, in report-to-track and track-to-track hypotheses, the labeled data points correspond to items which are already being tracked. Hypothesis enumeration then refers to enumerating the tracks which a new point could feasibly be assigned to. Two useful nearest neighbor classifiers are K-nearest neighbor and fuzzy K-nearest neighbor. For information on these and other classifiers see Section 2.2.
2.1.1.2 Gating

Gating [1,62,64] is another group of techniques for reducing the set of enumerated hypotheses to a feasible set of hypotheses. Gating techniques seek to eliminate highly unlikely hypotheses by evaluating how likely various hypotheses are and eliminating the least likely ones. Gating is often used in tracking problems, so this section will assume that all hypotheses are report-to-track (although track-to-track is a simple extension). A very simple heuristic gate can be based on the kinematics of a target. For instance, if it is known that a car can not travel more than 90 meters per second, and a car is known to be at position $x$, then one second later, only points within a 90 meter radius of $x$ could be associated with the car. Of course, it is possible to make this gate even narrower if it is known that the car is traveling in a certain direction with a certain speed. Then the car’s ability to accelerate and turn (its maneuvering capabilities) dictate that it’s more likely to be somewhere along the line it was traveling in (or a few degrees off in either direction) than it is for it to be in the complete opposite direction. Figure 2.2 shows an example gate which takes the maneuvering characteristics of a target into account.

The limiting factor in these simple gates is that they do not take into account uncertainty in the estimated state of an item (the track) or uncertainty due to sensor noise. Because of this limitation, probabilistic gates can be used [1]. These gates are based on state estimation using a Kalman filter (see Appendix A). A rectangular gate is the simplest gate of this type to implement, because it only deals with one dimension at a time. Specifically, if the Kalman filter predicts that the next measurement from a sensor should be $\hat{y}$, and the actual measurement from the sensor is $y$, then the innovation (or residual) vector $\tilde{y}$ is defined as follows:

$$\tilde{y} = y - \hat{y}. \quad (2.2)$$

Once the innovation vector has been defined, each component $\tilde{y}_l$ of the vector must satisfy the relation

$$|y_l - \tilde{y}_l| = |\tilde{y}_l| \leq K_G l \sigma_r \quad (2.3)$$
in order for the measurement to satisfy the gate. Here, $\sigma_r$ is the residual standard deviation, which is defined as

$$\sigma_r = \sqrt{\sigma_o^2 + \sigma_p^2},$$

where the observation variance ($\sigma_o$) is a characteristic of the sensor, and the prediction variance ($\sigma_p$) is taken from the Kalman filter covariance matrix. The parameter $K_{Gl}$ is known as the gating constant, and is a tunable parameter. Blackman [1] shows that if the gating parameter is a constant $K_{Gl} = K_G = 3$ in every dimension, then for a four-dimensional radar measurement, a valid observation will satisfy the gating test 99 percent of the time.

The main advantage of the rectangular gate is that it is relatively efficient to compute. Unfortunately, as the dimensionality of the problem increases, the probability that a rectangular gate will accept extraneous results increases. In these cases, an ellipsoidal gate is more desirable [1]. For an ellipsoidal gate, a measurement satisfies the gate if the
norm $d^2$ of the residual vector $\tilde{y}$ satisfies the following criterion:

$$d^2 = \tilde{y}^T S^{-1} \tilde{y} \leq G,$$  \hspace{1cm} (2.5)

where $G$ is a tunable parameter and $S$ is the residual covariance matrix (as calculated by the Kalman filter). There are several methods for choosing $G$. The simplest method is to make $G$ a constant based on the dimensionality of the problem. Blackman [1] shows how to use a chi-square table to choose $G$ such that a valid measurement will satisfy the gating test 99 percent of the time. Alternatively [96, 97], a maximum likelihood gate $G_0$ can be found such that an observation falling within the gate is more likely to be from the track than an extraneous source. The disadvantages of this gate are that it requires more computations and it requires the probability of detection and the new source density to be known.

Note that various hybrid gates are possible. For instance, observations may be first gated by a coarse, but computationally cheap rectangular gate, then the observations which satisfy the rectangular gate are tested using a finer, but more computationally expensive ellipsoidal gate. There are also cases where an ellipsoidal gate is used for non-maneuvering targets and a heuristic gate such as the one in Figure 2.2 is used for maneuvering targets.

The purpose of the hypothesis generation and evaluation process is to use a simple test to find feasible hypotheses so that fewer possibilities need to be evaluated in the more sophisticated hypothesis evaluation and selection steps. Unfortunately, the gating test described above requires that every observation be compared to every track. In cases where there are many items to be tracked and many observations to correlate to tracks, even the simplest gating strategy can lead to a computational bottleneck [62,64]. A better technique is to only compute the gating test if a report and track are “near” each other. For instance, it is possible to divide the state space into grid cells and only compare tracks with observations that fall within adjacent grid cells. This technique can still be inefficient because it depends heavily on the size of the cells and the target density. If cells are too large, there may be many observations within on cell. Conversely,
if the cells are too small, the algorithm may spend too much time examining empty grid cells. Uhlmann [62, 64] proposes what is essentially an adaptive grid, which overcomes these difficulties. The method for adapting the grid is closely related to the binary search and priority kd-tree algorithms from computer science.

2.1.2 Hypothesis evaluation

The goal of the hypothesis evaluation step is to score each of the potential associations created by the hypothesis enumeration step, so that the hypothesis selection step may select the optimal association (or at least an acceptable suboptimal association). There are many possible hypothesis evaluation metrics which may be used. For a nearly exhaustive list see Hall and McMullen [19]. Figure 2.3 shows the table from Figure 2.1 after hypothesis evaluation has been applied. The infeasible hypotheses are still marked with an x, but now the feasible hypotheses are given a numerical score as opposed to all being set to one.

<table>
<thead>
<tr>
<th></th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>x</td>
<td>3</td>
<td>x</td>
<td>9</td>
</tr>
<tr>
<td>T2</td>
<td>x</td>
<td>5</td>
<td>7</td>
<td>x</td>
</tr>
<tr>
<td>T3</td>
<td>8</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>T4</td>
<td>x</td>
<td>x</td>
<td>4</td>
<td>x</td>
</tr>
</tbody>
</table>

Fig. 2.3: Distance matrix created by the hypothesis evaluation step

Probabilistic techniques are one important class of hypothesis evaluation methods. The most commonly used measure of correlation between two random variables \( x_1 \) and \( x_2 \), is to assume that they are Gaussian distributed with covariances \( P_1 \) and \( P_2 \),
respectively. The probability of correlation is then \[1,64,98\]

\[
P_{\text{association}}(x_1, x_2) = \frac{1}{\sqrt{2\pi|P_1 + P_2|}} \exp \left( -\frac{1}{2} (x_1 - x_2)(P_1 + P_2)^{-1}(x_1 - x_2)^T \right). \tag{2.6}
\]

Note that Equation (2.5) is proportional to the logarithm of this probability (where \(S = P_1 + P_2\)) \[64,98\]. Therefore, in this case the hypothesis enumeration and selection steps can be combined into a single step. This is an example of a more general probabilistic technique known as likelihood, which seeks to compute \(P(R|H)\), where \(R\) is a report and \(H\) is a hypothesis \[99\]. Similar to the likelihood technique is the Bayes a posteriori technique, which uses Bayes rule to calculate the probability of association as follows \[100, 101\]:

\[
P(H|R) = \frac{P(R|H)P(H)}{P(R)}. \tag{2.7}
\]

Although the probabilistic techniques are the most commonly used techniques in practice, other possibilities also exist. Possibilistic techniques \[102,103\] seek to correct some of the deficiencies in the traditional Bayesian techniques. They are useful in situations where it is difficult to assign a probability distribution to the items to be correlated. This is especially important in cases where the data comes from a human user (either in the form of rules in an expert system or when a human acts as a “soft sensor” by providing reports to the system). The most common possibilistic techniques are Dempster-Shafer (the theory of evidence) \[80,104,105\] and fuzzy logic \[28,106\]. Dempster-Shafer is often used when a probability can be assigned to lie in an interval, but can not be precisely defined. For instance, if it is known that the probability of an event is greater than 0.5, but less than 0.9, then Dempster-Shafer theory would be a good choice for dealing with this information. Fuzzy logic is best suited for situations where there are no precise probabilities defined, such as natural language statements (see Appendix B).

These possibilistic techniques work well for the type of data which they were designed for, but it is very difficult to combine possibilistic data with probabilistic data (or even data which has been expressed using theory of evidence with data from fuzzy logic \[24\]). Therefore, the need has arose to create a unified theory of uncertainty which
encompasses all the other forms of uncertainty. One of the most promising techniques so far is random set theory [5, 25, 103, 106]. This is an ongoing research area, which will be explored further in this thesis. Appendix C provides some mathematical details behind random set theory, and Chapter 5 will explore using random set theory for fusing hard data with soft data.

2.1.3 Hypothesis selection

The final part of the data association process is hypothesis selection. The goal of the hypothesis selection step is to find the optimal assignment of observations to tracks. In this sense, optimal refers to minimizing the total expected error of the assignment problem. For instance, if the output of the hypothesis evaluation step is a probability of association (such as the probabilistic distance in Equation (2.6)), then maximizing the probability of association is equivalent to minimizing the following distance [1]:

\[ d_{G_{i,j}} = d_{i,j}^2 + \ln |S_i|, \]  

(2.8)

where, as in Equation (2.5), \( d_{i,j} \) is the norm of the residual vector and \( S_i \) is the sum of the covariance matrices. Note that the output of hypothesis evaluation step will often be this distance, which is easier to compute than the true probability of association. The goal of the hypothesis selection step is then to minimize the total cost of an association

\[ C = \sum_i \sum_j d_{i,j} I_{i,j}, \]  

(2.9)

where \( I_{i,j} \) is an indicator function which is one if observation \( j \) is assigned to track \( i \) and zero otherwise. Note that this assumes that all reports are assigned to one and only one track. If reports can belong to new tracks or no track at all, then this must also be accounted for in the cost function.
2.1.3.1 Heuristic techniques

Finding the optimal solution to the assignment problem is a combinatorial optimization problem [19]. For a small number of tracks and observations, the problem can be solved by an exhaustive search. For larger sized data sets, the problem becomes an NP-hard optimization, and finding an optimal solution becomes intractable. Often the problem is solved by simple heuristics. The simplest such technique is nearest neighbor, where each observation is assigned to the track it is nearest to (in terms of a distance such as the one defined in Equation (2.8)). This can lead to many problems, the most obvious of which is when there is a tie. Another problem can be seen by referring to Figure 2.3. Using a nearest neighbor assignment, O1 will be assigned to T3, O2 will be assigned to T1, O3 will be assigned to T4, and O4 will be assigned to T1. Therefore, two observations will be assigned to T1 and no observations will be assigned to T2. If we know a priori that there is a one to one correspondence between tracks and observations, this can not be the case.

Two more robust heuristic solutions are given by Blackman [1]. The first is given as follows:

1. “An observation that validates with a singly validated track is rejected by any multiply validated track.”

2. “A multiply validated observation is rejected by any track that validates with a singly validated observation.”

3. “Whether or not a track is multiply validated is determined again after each application of Rule 1 affecting it. A track that becomes singly validated is again subject to Rule 1.”

4. “Whether or not an observation is multiply validated is determined again after each application of Rule 2 affecting it. An observation that becomes singly validated is again subjected to Rule 2.”

5. “For each remaining multiply validated track, choose the observation with minimum distance.”
6. “For each remaining multiply validated observation, choose the track with minimum distance.”

In this case a validated observation track pair refers to one which passed the gating test. Multiply validated observations (or tracks) refer to observations (tracks) which could correspond to more than one track (or observation). This assignment method avoids situations where no observations are assigned to a track or when multiple observations are assigned to the same track. Blackman’s second solution is given as follows:

1. “Search the assignment matrix for the closest (minimum distance) observation-to-track pair and make the indicated assignment.”

2. “Remove the observation-to-track pair identified above from the assignment matrix and repeat Rule 1 for the reduced matrix.”

This is just a more principled way of performing the nearest neighbor test described above. It prevents multiple assignments, but still has some problems. The first suboptimal assignment appears to be the most sophisticated, but Blackman shows situations where even it fails to find the optimal assignment.

2.1.3.2 Optimization techniques

Since the assignment problem is an optimization problem, it makes sense to use techniques from mathematical optimization when simple heuristics fail. Techniques for doing so include [19]: integer programming [72], set partitioning [107], dynamic programming [108], and modern heuristic techniques (simulated annealing, genetic algorithms, neural networks, etc.) [109]. These techniques are useful when there are a large number of associations to be made, but they are unnecessary for the applications of interest in this thesis, therefore they will not be explored further. For more information on all of these algorithms see Chapter 3 of Hall and McMullen [19].

2.1.3.3 Tracking techniques

A different approach is taken in multiple hypotheses tracking (MHT) [1,110,111]. In cases where it is difficult to make an assignment, MHT delays the assignment decision
until more data is available. For example, if at time 1 two observations are received, each of which could belong to a new track or to false alarms, four hypotheses are formed (O1 new track, O1 false alarm, O2 new track, O2 false alarm). Obviously, the false alarm tracks do not need to be maintained, so at step 2 there will be two distinct tracks, T1 and T2. If two more observations are received, then there are eight possible hypotheses (O3 is a new track, O3 is a false alarm, O3 corresponds to T1, O3 corresponds to T2, and four more for O4). Obviously, for even a simple problem where only two observations are received at each time step, the number of tracks quickly becomes too large. Therefore, only hypotheses that pass the gating test will be initiated, unlikely tracks will be pruned, and similar tracks will be merged. Techniques for performing these operations are described in Blackman [1].

A similar approach is seen in joint probabilistic data association (JPDA) [1, 112, 113], which seeks to modify the Kalman filter to account for uncertainty in observation track pairs. Rather than maintaining multiple hypotheses such as in MHT, the JPDA filter assigns multiple observations to a track (where the observations are weighted by how likely they are). By doing so, the JPDA filter avoids the exponential increase in tracks seen by MHT. Although they were originally formulated for the Kalman filter, both MHT and JPDA can be extended to other state estimators such as the interacting multiple model filter [63] and the particle filter [7].

2.2 Multi-sensor classification

Classification is a problem from the broad field of pattern recognition. Bishop [71] describes pattern recognition as follows,

“The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.”
In the case of multisensor data fusion problems, classification involves taking in data from multiple sensors and using that data to determine the identity of the entity being sensed.

When classifying data from multiple sensors, the fusion may take place at the data level, the feature level, or the decision level [19]. At the data level, data from sensors is fused directly, then features are extracted and a classification decision is made. This method has a somewhat limited applicability because it requires sensors to be identical or commensurate; therefore, it will not be discussed further here. Fusion occurs at the feature level when important features are extracted from the raw sensor data, then the features from multiple sensors are used to make a classification. Techniques to perform feature extraction will be discussed in Subsection 2.2.1, and algorithms to perform feature level fusion will be discussed in Subsection 2.2.2. Finally, decision level fusion utilizes a separate classifier for each sensor, then the classifications from each individual sensor are fused. Decision level fusion is especially important in distributed data fusion because it is much easier to communicate a decision to a partner than it is to send a feature vector (or worse yet raw data). Algorithms for performing decision level fusion will be discussed in Subsection 2.2.3

2.2.1 Feature extraction

The purpose of feature extraction is to extract important information (where important in this case describes features that are useful for classification) from the complete data set. (For instance, in an image the foreground is often more important than the background.) Doing so will lead to a loss of data [19], but this is acceptable if it means a simpler classifier can be used with similar accuracy to using the raw data. If data comes in the form of a time series, feature extraction techniques will come from the signal processing domain [114, 115]. For image data (which is the main concern of this thesis), the field of image processing provides a wealth of techniques for preforming feature extraction [116]. There are also many techniques for feature extraction from the pattern recognition field which are general enough to work on all types of data [71].
Signal processing techniques for feature extraction deal with signals that are expressed as a function of time. The best known feature extraction technique is probably the Fourier transform [115]. The Fourier transform maps a time domain signal into the frequency domain. If there are a few important frequencies of interest in a signal, then this representation will be much more compact than the time domain representation. The Fourier transform is useful for deterministic signals; its analog for random signals is the power spectrum, which is simply the discrete-time Fourier transform of a wide sense stationary random signal’s autocorrelation sequence [114]. The autocorrelation sequence of a random process is itself a feature (a time domain representation of the second-order moment of the original random sequence).

When data comes in the form of a two or three dimensional image (which could include infrared, LIDAR, and synthetic aperture radar in addition to more traditional visual images), techniques from image processing are best suited for feature extraction [116]. Here, a few important feature extraction techniques are briefly mentioned. For more details see Trucco and Verri [116]. Edge detection seeks to find areas in an image where there is a sharp change in intensity. Such regions often correspond to the boundary of objects in the original scene. One of the most commonly used edge detectors in practice is the Canny edge detector [117]. Corners are also important features in images because they are stable across a sequence of images; therefore, they are useful or finding the same point in different images (such as in a stereo vision application) [116]. An example of a useful corner detection routine is given in Tomasi and Kanade [118]. Another common computer vision method is the Hough transform [119]. The Hough transform was originally developed to detect straight lines, but it has also been applied to more general geometric shapes [116]. SIFT [34,61] is an important image transformation for robotics problems. It seeks to identify unique features in an image that should be visible from multiple angles and at multiple scales.

Whereas the above techniques were designed with specific applications in mind, there are many feature extraction techniques which are general enough to use on any type of data. Some of them can be applied to data which has already been processed by another feature extraction technique. Principal component analysis (PCA) seeks to
project a high dimensional dataset onto a lower dimensional linear space that preserves the important information in the original space [71, 120, 121]. Unlike many of the previously mentioned techniques, PCA requires a training set in order to find the correct projections. It is an unsupervised technique, however; so the training set can be unlabeled. Fisher’s linear discriminant [122] (also known as linear discriminant analysis, or LDA) also seeks to reduce the dimensionality of a dataset, but it uses labeled data points. It specifically seeks to maximize the distance between the two classes in the reduced dimensionality set. Although originally formulated for two classes, it is possible to extend LDA to multiple classes [71, 123].

2.2.2 Feature level fusion

When data is fused at the feature level, each sensor’s data is processed by its own feature extractor; features are processed by some data association technique, as described in Section 2.1; and features corresponding to the same item are given to the classifier. In practice, many classifiers work at the feature level (even those that were designed to work at the data level can be used at the feature level if the feature vector is treated as the data).

One of the simplest classifiers is the linear discriminant function [71]. For a feature vector $x$, the linear discriminant function is

$$y(x) = w^T x + w_0,$$

(2.10)

where $w$ is a weight vector and $w_0$ is a bias. The feature vector is assigned to class $C_1$ if $y(x) \geq 0$ and to class $C_2$ otherwise. If there are more than two classes that a feature vector can belong to, the problem can still be solved by creating $K$ classifiers of the form

$$y_k(x) = w_k^T x + w_{k0},$$

(2.11)

The point is then assigned to class $C_k$ if $y_k(x) > y_j(x)$ for all $j \neq k$. The optimal set of weights can be found based on training data using least squares [71], or the weighting
matrix for Linear Discriminant Analysis described in Section 2.2.1 could be used. (In which case, feature extraction and classification can be combined into one step.)

Another important type of classifier is the nearest neighbor classifier. The simplest nearest neighbor classifier is the “hard” nearest neighbor (NN) classifier. The NN classifier classifies a new item based on which labeled item it is closest to (either in terms of distance or in terms of some more general similarity metric). The main advantage of this classifier is its simplicity. In cases where all targets are well defined, the nearest neighbor classifier may be all that is necessary to correlate new reports to existing tracks. If boundaries between classes are not well defined, it can break down, however. It may be that the nearest neighbor to a new feature vector is from class $B$, but that there are many other points near it which belong to class $A$; in this case, it would make more sense to assign the new point to class $A$.

For this reason the K-nearest neighbor (K-NN) algorithm was created. The K-NN algorithm finds the $K$ closest labeled points to a new data point, where $K$ is an integer which must be chosen by trial and error. (Low values of $K$ correspond to less computation, high values correspond to more accuracy.) The point is then assigned to the class which gets the most “votes”, i.e. the class that most of its neighbors belong to. At first, K-NN appears to be an ad hoc method, but Bishop [124] gives a Bayesian interpretation of K-NN. In particular, he shows that given a data point $x$, the posterior probability of $x$ being in class $C_k$ is

$$p(C_k|x) = \frac{K_k}{K},$$

(2.12)

where $K_k$ is the number of the $K$ nearest neighbors from class $C_k$. Therefore, assigning $x$ to the class which most of its neighbors belong to corresponds to maximizing the posterior probability of class membership.

The K-NN algorithm still suffers from one drawback, it gives distant neighbors the same weight as nearby neighbors. To overcome this limitation, the fuzzy K-NN algorithm was created [125]. The fuzzy K-NN algorithm assumes that the labeled points, $x_j$, are members of fuzzy sets, $C_i$, with membership functions $\mu_{C_i}(x_j)$. A new point $x$ is then
classified by finding its $K$ nearest neighbors then assigning it to fuzzy set $C_i$ with the following membership function

$$
\mu_{C_i}(x) = \frac{\sum_{j=1}^{K} \mu_{C_i}(x_j) \frac{1}{\|x - x_j\|^{2/(m-1)}}}{\sum_{j=1}^{K} \frac{1}{\|x - x_j\|^{2/(m-1)}}},
$$

where $m$ is a tunable parameter, which determines the degree to which far away labeled points should effect the classification. In the limit as $m$ approaches one the fuzzy K-NN classifier becomes a simple NN classifier.

Neural networks [69, 70] are trained to classify patterns based on a large set of training data. A mapping from the input space (often pre-processed sensor data) to the output space (which is interpreted as the class which the data falls into) is learned and stored in the structure of the network. Neural networks can be very powerful because they can represent any nonlinear mapping [126] and, in certain cases, may have a probabilistic interpretation [66, 68].

All of the previous methods focus on classifying patterns based upon training examples. A different approach is taken in knowledge-based fuzzy pattern recognition [29]. Similar to fuzzy K-nearest neighbor, input data is assigned to a class $C_i$ using a fuzzy membership function $\mu_{C_i}(x)$. However, rather than the mapping coming from training data, natural language statements, such as “big” or “small” and logic statements such as AND, OR, and NOT are transformed into mathematical models. For classification problems that are easily described in words, fuzzy logic can be a good choice. One area where it has been applied in the robotics domain is in drawing inferences from sensor readings using a technique called the continuous inference network (CINet) [65,67]. Because fuzzy logic deals with linguistic statements, it is also a useful technique for communicating with humans-in-the-loop [127]. In cases where expert knowledge is available, but it is incomplete, or where certain items are difficult to classify using knowledge based approaches, it is possible to combine knowledge based approaches with other approaches to form hybrid
approaches such as multi-stage fuzzy systems and neuro-fuzzy systems [29]. There are also cases where a fuzzy model may be learned from training data [29,128].

### 2.2.3 Decision level fusion

The final fusion method fuses data at the decision level. In other words, each sensor has its own individual classifier that provides a preliminary classification. Then the individual classifications are fused to reach a final result. There are several reasons for doing this. The first is that simpler classifiers may be used at each stage. The second is that in a distributed data fusion problem, it is much easier for partners to communicate decisions with one another than it is for them to communicate feature vectors with one another. Some major methods for decision level fusion include Bayesian inference [79], Dempster-Shafer theory [80, 104], and heuristic techniques such as voting [19]. All of these techniques are especially well suited for distributed data fusion; therefore, they are described in detail in Chapter 3.

Decision level fusion techniques are closely related to committee machines [70]. Whereas decision level fusion techniques usually assume that each classifier only has a subset of the input data to work with, committee machines usually assume that each classifier has the same input data. Therefore, traditional committee machines actually belong at the feature level (although most committee machines could be modified in order to operate at the decision level). One particularly interesting committee machine is called boosting [129, 130]. Boosting works by first training a simple classifier (which could use any of the techniques described in Section 2.2.2) on a subset of the data. Then another classifier is trained on a new dataset made up of features which the first classifier could not classify. This can be repeated as long as there are sufficient training examples. The utility of the algorithm comes from the fact that even if each classifier has a high error rate (as long as it is slightly better than random guessing), the final error rate can still be much lower.
2.3 Multi-sensor tracking

One of the most well studied problems in data fusion is the tracking problem (also known as state estimation) [5, 7, 10, 19, 63, 74, 75, 82, 131–136]. The purpose of tracking is usually to determine the position and velocity of a moving target. In linear Gaussian systems, it is well known that the optimal state estimator is the Kalman filter [10, 131] (see Appendix A, Algorithm A.2). It can be shown [5] that the Kalman filter is a special case of the optimal Bayes filter, which is applicable to any properly modeled system. Details on the Bayes filter and Kalman filters are given in Appendix A. Unfortunately, the optimal Bayes filter is usually computationally intractable (except, for the special case of a linear Gaussian system, i.e. the Kalman filter). This section gives details of some important suboptimal (but computationally tractable) filters which are commonly used for data fusion.

One important issue that is not dealt with in this section is what to do when dealing with multiple targets. Almost all trackers, except for a few exceptions such as multiple hypothesis tracking and joint probabilistic data association (see Section 2.1.3), assume that each measurement uniquely belongs to one and only one track. Therefore, as is commonly done in practice, it is assumed that the data association problem (Section 2.1) and the tracking problem can be addressed separately.

Another problem comes about when multiple sensors sense the same target. If the sensors are correlated, their correlation must be determined and accounted for in the fusion. This is a difficult problem, which often arises in distributed sensing problems; therefore, it is dealt with in detail in Section 3.3. If it can be assumed that the sensors are uncorrelated (as is often the case in a centralized fusion system), then the Kalman filter can be used simply by grouping all the sensor measurements together into a single measurement vector. If there are many more measurements than states (as would be the case when there are many sensors tracking a target that is constrained to move in two dimensions), it is more efficient to use the dual of the Kalman filter, known as the information form of the Kalman filter. (Full details of the information filter algorithm are given in Algorithm A.3). In the information form, processing new measurements is a simple additive process (it is even more efficient if measurements only effect a subset
of the state space) [11]. There is, of course, a trade-off for this efficiency; state updates become less efficient than the Kalman filter; therefore, it is not applicable to systems with a high-dimensional state space (such as when tracking many correlated objects).

### 2.3.1 Nonlinear approximations

Although the Kalman filter is optimal for linear systems, most real systems are nonlinear; therefore, most applications use an extended Kalman filter (EKF), which linearizes the model about the current best estimate of the state [10, 132]. (See Algorithm A.4 for details on how the EKF is implemented.) The EKF works well for systems where the nonlinearities are minor, and noise is approximately Gaussian, but performance can greatly degrade for highly nonlinear systems with non-Gaussian noise. Nonetheless, because the EKF is computationally tractable and has been around for a long time, it is used in most situations.

In situations where nonlinearities are severe (such as a bearings-only tracking problem [7]), the unscented Kalman filter (UKF) [82] is often a better choice than the EKF. The UKF works by maintaining an estimate of the state and covariance as a set of “sigma points”. This is similar to a particle filter, but the sigma points are chosen deterministically, whereas the particles in a particle filter are chosen statistically. This means that fewer sigma points are needed to represent the state, thus typically making it more efficient than a particle filter. It can be shown that the UKF approximates that nonlinearity in system to the third order, as opposed to the EKF which represents the nonlinearity to the first order [10].

### 2.3.2 Gaussian sum filters

Another variation of the Kalman filter is the Gaussian sum filter [7,133]. Similar to how the Gaussian mixture model in pattern recognition (see Section 2.1.1.1) represented a probability distribution as a weighted sum of Gaussian components, the Gaussian sum filter represents the state of a target as a mixture of Gaussians:

\[
p(x_k | Y_k) \approx \sum_{i=1}^{q_k} w_k^i \mathcal{N} \left( \hat{x}_k^i; \hat{x}_{k|k}^i, P_{k|k}^i \right),
\]  

(2.14)
where \( p(x_k|Z_k) \) is the optimal Bayes estimator posterior probability density function (pdf), as defined in Appendix A. In general, the Gaussian sum filter can represent probability distributions which are multi-modal, but where each mode is approximately Gaussian. One problem with the Gaussian sum filter is that the number of components can grow exponentially with time. One approach to combat this is to prune all but the most likely components [137]. In other cases, certain assumptions can be made in order to make the problem computationally tractable.

In the static multiple model estimator [7,134], it is assumed that the target is in one of a finite number of regimes (such regimes could include the flight phases of a ballistic missile or maneuvering vs. non-maneuvering flight for an airplane). It is further assumed that the target does not change regimes. A different filter (which could be an EKF, UKF, or particle filter) is designed for each regime. The output of the filter is given by a weighted sum of Gaussians, one for each possible regime. Initially, the weights are given by how likely each regime is perceived to be. As time progresses, the weights are updated by how well each filter predicts new measurements. In the limit, the weight for the correct filter will approach one and the weights for all other filters will approach zero.

In the interacting multiple model (IMM) filter [7,135,136], it is once again assumed that a target can be in one of a finite number of regimes. This time, however, the target can switch from one regime to another with a certain known probability. Just as in the static case, one filter is designed for every possible regime, but instead of each filter being updated only by its own estimate of the target’s state and covariance, it must be updated by a weighted average of all the filters’ estimates. This time, the weights are determined not only by each individual estimator’s likelihood of being correct, but also by the probability that the target transitions from one state to another. If the target stays in one regime for a long time, all but one of the weights will become small, but they will not be zero due to the fact that the transition probabilities are nonzero. If the target suddenly changes regimes, the filter will quickly react by adjusting the weights for each regime. Therefore, the IMM estimator is well suited for tracking targets that may
stay on a certain course for some time, but then make a sudden maneuver. (An example of such a case is given in Bar-Shalom and Blair [63].)

2.3.3 Particle filters

Because the EKF and even the UKF can perform poorly when nonlinearities are severe, the particle filter was developed [7, 74, 75]. The particle filter is a Monte Carlo method which represents the probability distribution of the target’s state as a set of particles instead of a single mean and covariance as in the Kalman filter. This means that the particle filter can represent more general probability distributions than the Kalman filter, which assumes that probabilities are Gaussian. The basic algorithm for the particle filter is given in Appendix A as Algorithm A.5.

An example of a particle filter tracking a ballistic object on reentry is given by Ristic et al. [7]. The particle filter is compared with both an EKF and a UKF. The EKF showed unacceptably large bias (on the order of 100 m in altitude) and high error standard deviation (over 100 m in altitude) in a Monte Carlo simulation with 200 runs. The UKF and particle filter were both unbiased (i.e. their average error approached zero), but the particle filter had slightly better error standard deviation (on the order of 15 m in altitude, for instance). In this test, the UKF required 5 times more CPU time than the EKF, and the particle filter required 100 times more CPU time that the EKF. Therefore, in this case, the UKF was most likely worth the computational overhead, but whether or not the particle filter was worth the overhead would depend on the desired accuracy and the speed of the computer.

The particle filter is very versatile, with many possible variations. Ristic et al. [7] provide numerous examples of particle filter variations. For instance, the IMM filter can be combined with a particle filter to create a multiple model particle filter. It is also possible to take into account information such as terrain using particle filters. Another popular method is to combine particle filters with other estimators (such as the EKF or UKF) to form hybrid estimators, which usually require fewer particles than the standard version of the particle filter.
2.4 Data fusion for robotics problems

The methods in the previous sections were designed primarily for general data fusion problems. In the special case of a robotics problem many of the techniques are more specialized. The reason is that most robots typically carry similar sensors (range finders, cameras, and stereo cameras) and have similar motion models (i.e. some form of odometer). In addition, robots, no matter the application, often have similar tasks that they all most do; such as reaching a goal while avoiding obstacles. Often special forms of the Bayes filter (see Appendix A) are used to accomplish these tasks, but there are also cases where fuzzy logic is used [138]. For the probabilistic methods, in the state update step, the most important quantity will be $p(x_k|u_k, x_{k-1})$, the probability of a new state (robot pose) given the previous state and the robot’s (known) control input, which is calculated using the known motion model and the known noise parameters. Likewise, in the measurement update, the robot calculates $p(y_k|x_k, m)$, the probability of receiving a measurement given the robot’s pose and the map. In fuzzy logic, similar quantities will be defined as fuzzy membership functions.

The three main types of problems studied in the robotics domain are localization (finding out where a robot is in the world), mapping (creating a model of the world based on the robot’s known position), and simultaneous localization and mapping (determining the robot’s location with respect to landmarks whose locations are also unknown). A brief review of these methods is presented in the following sections. For a much broader overview of the robotics field see Thrun et al. [11].

2.4.1 Localization

The localization problem seeks to determine a robot’s position with respect to a known map. Cox [139] has described localization as, “the most fundamental problem to providing a mobile robot with autonomous capabilities.” The most well known localization method is the global positioning system (GPS) [31]. GPS can be used to precisely determine the location of a robot (or anything equipped with a GPS receiver) by determining its range from multiple satellites whose positions are known precisely. In general,
mobile robots often operate indoors where GPS signals are unavailable, therefore some other technique for localization must be used. The two main categories of localization techniques are probability based and fuzzy logic based.

2.4.1.1 Probabilistic localization

In probabilistic localization, the goal is to compute bel($x_k$), the belief that the true position is $x_k$ from the known (probabilistic) motion and measurement models $p(x_k|u_k, x_{k-1})$ and $p(y_k|x_k, m)$, respectively [11]. The main methods for expressing these probabilities utilize the extended Kalman filter (EKF), grid methods, or particle filters. In EKF based approaches [140], the state $x$ is the current position and orientation of the robot and the measurement $y$ is the range or bearing to a set of known landmarks. Using this information, an EKF tracks the position of a robot using the robot’s odometry and measurements of feature locations. The EKF technique works well when there is little uncertainty in the robot’s position, but it can fail for global localization problems (i.e. when all the robot knows is that it is somewhere in a building) or when the robot is kidnapped (i.e. it suddenly finds itself somewhere other than where it expected to be).

The problems with the EKF are due to the fact that an EKF can only represent uni-modal (i.e. Gaussian) probability distributions, but often the probability distribution bel($x_k$) is more complicated than that. In order to correct this shortcoming, grid based filters have been applied to localization problems [141]. The grid filter approximates the probability distributions of a Bayes filter (see Appendix A) as a grid of discrete locations. Such an approach can yield good accuracy, but suffers from computational problems because it must expend resources in areas of the state space with low probability. To overcome these computational problems, particle filters were applied to the localization problem, creating a technique known as Monte Carlo localization [142,143]. By representing the robot’s uncertain location as a set of particles, Monte Carlo localization can represent arbitrary probability distributions while avoiding some of the computational problem of the grid based localization procedure.
2.4.1.2 Fuzzy localization

In fuzzy logic based localization, either the map, the sensor readings, or the robot’s position is fuzzy [138]. The fuzzy logic based methods work best when precise statistical models of uncertainty are unavailable. One example of a fuzzy logic approach is in Demirli and Turksen [144], which determines the location of the robot by a triangulation procedure that utilizes two or more ranges to walls with known locations. Both distances and angles are treated as fuzzy sets, and the robot’s position is also fuzzy. An even more general approach is taken in Saffiotti and Wesley [145], which treats odometry and external information as fuzzy sets in addition to sensor measurements. These items are combined using fuzzy conjunction in order to determine the robot’s position. Saffiotti and Wesley also allow more general maps where the locations of various features (such as doors, windows, and walls) can be fuzzy.

2.4.2 Mapping

Mapping problems are well studied in the literature [11, 32, 34, 69, 76–78, 138, 146–149]. The goal of any mapping problem is to determine the location of important objects (such as obstacles) in the world. In most cases, this is done probabilistically; in which case the goal is to calculate $p(m_k|Y_{k},X_{k})$. In other words, to calculate the probability of a certain map given all previous measurements and states.

2.4.2.1 Occupancy grid mapping

Early attempts at mapping [32] assumed that the robot’s position was known exactly and that the map at one point in the world was independent of the map at other points. This assumption lead to a formulation known as the occupancy grid map, where the world is broken into a discrete grid (i.e. $m = \{m_i\}$), and each cell is assigned a probability of being occupied, $p(m_i = 1)$. (By convention, $m_i = 1$ corresponded to an occupied cell.) Because it is assumed that the map at one location is independent of the map at all locations, the estimation problem can be broken into many simpler estimation problems (one per grid cell): $p(m_i|Y_{k},X_{k})$. By the independence assumption,
the posterior distribution for the whole map can then be given by

\[ p(m|Y_k, X_k) = \prod_i p(m_i|Y_k, X_k). \]  \hspace{1cm} (2.15)

This method is highly general, but it lacks robustness, because it assumes that the robot’s position is known exactly. This may be an acceptable assumption for a UAV, or a ground robot that operates outdoors, because they probably have access to GPS. This is a problem, however, for a robot operating indoors or in an urban environment where GPS will not be available. It also suffers from the drawback that each cell is assumed to be independent of all others. This is not reasonable if obstacles typically span multiple cells, in which case one cell would be more likely to be occupied if its neighbors were occupied. Keeping track of this information would be more accurate, but would lead to an increase in computational time. The final problem with an occupancy grid map is that it can only designate a cell as occupied or unoccupied, no other high-level information is available.

### 2.4.2.2 Fuzzy mapping

Fuzzy logic can also be applied to mapping problems [138]. One approach is to create a fuzzy logic version of the occupancy grid described above [146, 149]. In this case, the value of a grid cell is no longer the probability of being occupied, rather it is the membership of that cell in the fuzzy set “Occupied”. Because possibilities (unlike probabilities) need not add up to one, another fuzzy set “Unoccupied” could also be defined. There are cases where the two are not necessary the complement of one another. For instance, if the membership functions for both sets are close to zero in a cell, then the state of the cell is uncertain. If the membership functions are both close to one, then there is a conflict at that point. This information could be utilized by an intelligent planner, which seeks to explore unexplored regions, resolve conflict situations to build a better map, or simply safely reach a goal [149]. It is also possible to make the grid itself fuzzy as in Tunstel [147]. In this case, the fact that an obstacle exists is certain,
but its location is imprecise. All that can be said about the obstacle is that it is in the northeast corner of the area, for instance.

It is also possible to use fuzzy logic to create other forms of maps. For instance, in Gasos and Martin [148] fuzzy logic is used to process ultrasonic data in order to find objects with flat faces. This information is then used to find the boundaries of various objects (where the position and orientation of the boundaries are themselves fuzzy). This data can also be used for localization by comparing a local map with a previously obtained global map. A similar process can be used to find straight lines in visual images [150] by utilizing a fuzzy version of the Hough transform. By doing so, it is possible to find correspondences between lines in different images in order to map the environment.

2.4.2.3 Topological maps

Both occupancy grids and feature-based maps are classified as metric maps, since they represent the world in terms of distances between the robot and various landmarks and/or obstacles. A different approach to mapping uses topological maps. Topological maps are more interested in high level properties, such as intersections [78] or open regions and corridors [76]. The topological map may be learned directly from sensors, as in Kelley et al. [78], or by processing a metric map, as in Thrun [76]. The advantages of topological maps are that they may be easier to process using path planners or artificial intelligence techniques, and they are also more compact, which makes them easier to send over a communications channel.

2.4.3 Simultaneous localization and mapping

Because of the limitations of occupancy grid maps, a technique known as simultaneous localization and mapping (SLAM) was developed [11]. In SLAM, the robot creates a map of the environment, and localizes itself in that map at the same time. In other words, it seeks to compute the probability of the state and map given all previous measurements and controls \( p(x_k, m | Y_k, U_k) \). Note that in this formulation, it can not be assumed that \( x_k \) and \( m \) are statistically independent (because they are both being
estimated by the same sensor data). Therefore, instead of separating the two problems, SLAM must deal with the combined state vector, which will have three states for the robot (x and y position and orientation) plus two states for each item in the map. For a realistic problem with many landmarks, this is an extremely high dimensional problem. The other crucial problem in SLAM implementations is data association (see Section 2.1). In order for the robot to uniquely determine its location, it must view the same landmark from multiple angles (i.e. it must triangulate its location); therefore, it is essential that the robot recognize features that it has seen before.

Most SLAM implementations use some form of landmarks as their map, the vehicle is then localized relative to these landmarks. This means the robot must have some means of identifying the landmarks, such as a neural network [69] or techniques from computer vision such as the scale invariant feature transform (SIFT) [34]. Traditionally, the locations of these landmarks (along with the robot’s state) will be treated as a combined state vector by an extended Kalman filter (EKF). This procedure is conceptually simple, but it suffers from large computational problems when the size of the map becomes large. It also suffers from the drawback that the map can only represent the locations of discrete features. If higher level properties were needed, they would have to be represented some other way.

A technique called FastSLAM [33] uses a particle filter [74] instead of an EKF. FastSLAM takes advantage of the fact that the estimates of the locations of various features are conditionally independent given the robot’s position for all times. This means that the estimation problem can be factored as follows [11]:

\[
p(X_k, m|Y_k, U_k) = p(X_k|Y_k, U_k) \prod_{n=1}^{N} p(m_n|X_k, Y_k, U_k),
\]

where \(m_n\) is the location of an individual feature. This factorization is always possible if the robot’s position is known exactly. If the robot’s position is unknown, then it is still possible if the robot’s position is estimated using a particle filter. For each particle, the robot’s position is fixed, and the positions of features in the map can be estimated as if the robot’s location where known. This means that each particle will have \(N\)
EKF's, one for each feature; therefore, there will be $1 + NM$ total filters, where $M$ is the number of particles. Although, there are many filters, each is only two (or in the case of the robot three) dimensional. Therefore, the computational time is much lower than the EKF based approach. Another advantage of the FastSLAM algorithm is that each particle makes its own data association decisions. Therefore, it is likely that at least a few particles will make the correct data associations. These particles are then more likely to be chosen in the resampling stage than the ones that made incorrect data associations. This means that incorrect data associations will not lead to catastrophic failure. FastSLAM is also more general than the EKF approach in that it can also be used with occupancy grid maps instead of feature maps [77].

2.5 Multisensor data fusion algorithms for UAV surveillance

There are many different applications for the traditional data fusion algorithms just discussed. One example is UAV surveillance. UAV surveillance is one part of the larger search and rescue problem described in the first chapter, but it is also an important problem in its own right. This section shows how centralized fusion methods can be used for typical UAV surveillance situations, such as detecting and classifying objects on the ground, finding the locations of ground targets, determining the elevation of the terrain under the UAV, tracking moving targets, and associating measurements to tracks. The algorithms in this section are often somewhat simplistic. Many complex nonlinear sources of noise (such as the errors in GPS location estimates and UAV attitude) are either assumed to be small or folded into a single noise term. The examples in this section are meant to provide context for the future chapters on hard and soft sensor fusion by first illustrating traditional hard sensor fusion algorithms. Better results could likely be obtained by using higher fidelity motion and measurement models, but the basic algorithms would remain the same.

2.5.1 Classification algorithms

All of the classification results in this section are based on a color filtering scheme which detects objects of a certain size and color [48, 50, 51]. This method is best suited
for detecting objects which are of a distinct color from their background. Three types of targets are detected using this method: red balls, red barrels and blue tarps. All of these targets are well suited for detection from a UAV because they are visible from the air and are a distinct color from green or brown grass. There are two different versions of the color filtering method: a crisp classifier and a fuzzy logic classifier.

The crisp classifier is shown in Figure 2.4. The classifier starts by converting an image to the hue, saturation, value (HSV) color space. This makes color filtering simpler, because all color information is stored in the hue channel. The next step is to find pixels with a hue in a certain range. Then, pixels with a saturation above a certain threshold are marked. This step is important because it eliminates pixels which are “washed out”; in other words, they may have the correct hue, but they are actually a gray color because of the low saturation. Next, the algorithm searches for connected blobs of pixels that pass both the hue and saturation tests. Finally the blobs whose size (in pixels) are within the given range are marked as targets.

This algorithm has five tunable parameters: $H_{\text{min}}$, $H_{\text{max}}$, $S_{\text{min}}$, $s_{\text{min}}$, and $s_{\text{max}}$. $H_{\text{min}}$ and $H_{\text{max}}$ define the acceptable range of colors for a target, this obviously varies depending on whether the target is a red ball or blue tarp. Note also that hue is often expressed in angular units, so a red hue is actually a hue above 330 degrees or below 30 degrees, which is the same as in the range −30 degrees to 30 degrees. The parameter $S_{\text{min}}$, which eliminates washed out pixels, is usually the same for all target types, but it can vary slightly from camera to camera. The parameters $s_{\text{min}}$ and $s_{\text{max}}$, which select targets of the correct size vary according to target size and the height at which the UAV is flying. Usually, in order to make sure targets are detected at all altitudes, $s_{\text{min}}$ is set just high enough to eliminate any false positives due to noise (which tend to only be a few pixels in size), but low enough to still detect targets even at the UAV’s highest altitude. Conversely, $s_{\text{max}}$ is set to be large enough to detect all targets at the UAV’s lowest possible altitude.

The fuzzy logic classifier uses a technique known as a Continuous Inference Network (CINet) [67]. A CINet is a network of fuzzy AND and OR nodes that determine a confidence that a certain property exists. The CINet for determining if a colored blob
Convert image to HSV color space

Find pixels with hue $H$ in the range $H_{\text{min}} < H < H_{\text{max}}$

Find pixels with saturation $S$ greater than $S_{\text{min}}$

Find connected blobs of pixels that satisfy both the hue and saturation tests

Mark blobs with size $s$ in the range $s_{\text{min}} < s < s_{\text{max}}$ as targets

Fig. 2.4: Crisp classifier based on color filtering
in an image is a red barrel is shown in Figure 2.5. One advantage of fuzzy logic is that it utilizes natural language statements. For instance, the barrel CI\textit{Net} can be read from right to left as follows, “an object is a barrel if it is high or low hue, high saturation, and the size of a barrel”.

The first layer of the CI\textit{Net} is a “fuzzification” step that translates pixel values in the image into fuzzy confidence levels. For example, in order to translate the hue of a pixel into a red confidence level the hue is passed through the trapezoidal membership functions shown in Figures 2.6a and 2.6b. The trapezoidal membership function classifies hue values below 25 degrees as “definitely low” (confidence of 1), hue values between 25 degrees and 45 degrees as “possibly low”, and hue values above 45 degrees as “definitely not low” (confidence of 0). Figure 2.6c shows a similar membership function for saturation. For determining if a blob is an appropriate size, the number of pixels in a colored blob are counted and passed through the membership function shown in Figure 2.6d.

The second and third layers of the CI\textit{Net} represent fuzzy logic functions. Stover \textit{et al.} [67] describes elaborate fuzzy AND and OR functions with weights. In this section, much simpler membership functions are used. A min is used for AND and a max is used for OR. These simple functions were found to work fine for identifying the red barrels.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{./images/figure2.5.png}
\caption{CI\textit{Net} for identifying a red barrel}
\end{figure}
2.5.2 Target localization

When a new target is detected, the filter must be initialized. This involves finding an estimate of the target’s position, and the covariance of the uncertainty in the position. To estimate the position, first the pixel value of the target is converted to a set of bearings as follows:

\[
\varphi = \arctan \left( \frac{(o_x - x_{im})s_x}{(o_y - y_{im})s_y} \right) \tag{2.17a}
\]

\[
\epsilon = \arctan \left( \sqrt{\frac{(o_x - x_{im})^2 s_x^2 + (o_y - y_{im})^2 s_y^2}{f}} \right), \tag{2.17b}
\]
where $\varphi$ is the azimuth angle, $\epsilon$ is the elevation angle, $(x_{im}, y_{im})$ is the target's location in pixel coordinates, $(o_x, o_y)$ is the pixel location of the image center, $s_x$ and $s_y$ are the size of a single pixel (in millimeters) in the horizontal and vertical directions respectively, and $f$ is the focal length of the camera (in millimeters). A discussion of these and other camera parameters can be found in an image processing book such as Trucco and Verri [116].

Given a bearing to the target and a range to the target, the target's location can be uniquely determined. Unfortunately a range is not available, all that is available is an estimate of the UAVs height above ground level. Therefore, the target's location must be determined using the iterative procedure given in Algorithm 2.1.

Once the target's position in the UAV NED frame is found, it is translated into local coordinate frame that is fixed at a given point on the ground.

This procedure only works correctly if there is an estimate of terrain elevation at every point on the ground in the area of the UAV. This data can be obtained for North America from the United States Geological Survey (USGS) Seamless Server [152] at one third arc-second (about 10 meter) resolution. Unfortunately, the data files from USGS are too large to store elevation data for more than a small area onboard the UAV. Another option is to build the terrain map from altimeter data, as in Section 2.5.3. A final option is use altitude above ground level data coming from the Piccolo autopilot [153]. This data is based on the Shuttle radar topography mission (SRTM) [154, 155]. It is relatively low resolution (one arc-second for the United States and three arc-seconds worldwide) compared to other data sources, but it provides worldwide coverage. Also, the data is not directly available from the autopilot, only an estimate of the current height above ground level is available. Because of these limitations, the iterative procedure just described is not used when this is the only elevation data source; instead, the estimate after one iteration is used to initialize the tracker. It then becomes much more important to get multiple views of the target in order to get a good estimate of its position.

The initial uncertainty in the state estimate is somewhat difficult to estimate. There are many possible sources of error, including the UAV's location, roll, pitch, and yaw, the uncertainty in the pixel location of the target, and the uncertainty in the terrain
Algorithm 2.1 Iterative procedure for determining a target’s location

1: Express the initial guess of the target’s location in UAV aircraft-body coordinate (ABC) frame as follows:

\[ z = h_{UAV} - h_{\text{terrain}} \quad (2.18a) \]
\[ x = z \tan(\epsilon) \cos(\varphi) \quad (2.18b) \]
\[ y = z \tan(\epsilon) \sin(\varphi), \quad (2.18c) \]

where \( h_{\text{terrain}} \) is the terrain map’s estimate of the height of the terrain directly below the UAV. Note that these equations assume that the camera’s axis align with the UAV’s. If this is not the case, an additional rotation will have to be performed.

2: repeat

3: Rotate the ABC location into the UAV’s north-east-down (NED) frame using the following Equation [151]:

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}_{\text{NED}} = \mathbf{R} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}_{\text{ABC}}, \quad (2.19)
\]

where

\[
\mathbf{R} = \begin{bmatrix}
  \cos(\psi_y) & -\sin(\psi_y) & 0 \\
  \sin(\psi_y) & \cos(\psi_y) & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  \cos(\theta_p) & 0 & \sin(\theta_p) \\
  0 & 1 & 0 \\
  -\sin(\theta_p) & 0 & \cos(\theta_p)
\end{bmatrix} \begin{bmatrix}
  1 & 0 & 0 \\
  0 & \cos(\phi_r) & -\sin(\phi_r) \\
  0 & \sin(\phi_r) & \cos(\phi_r)
\end{bmatrix} \quad (2.20)
\]

and \( \phi_r, \theta_p, \psi_y \) are the UAV roll, pitch, and yaw, respectively.

4: Set \( h_{\text{terrain}} \) to the terrain map’s estimate of the height of the terrain at location \((x, y)\).

5: Calculate the new estimate of the target’s location in UAV aircraft-body coordinate (ABC) frame using Equation 2.18.

6: until The new estimate of the target’s location is less than a certain distance from the old estimate
map. Even if all these errors are independent and Gaussian, they can create correlated, non-Gaussian errors in the target location due to the nonlinearity of the measurement. Fortunately, it has been found that setting the diagonal elements of the covariance matrix to some large value (on the order of 100 meters squared), and the off-diagonal elements to some small value (on the order of 10 meters squared) works for this application.

### 2.5.3 Terrain mapping

There are several options for range sensors which could be used for aerial terrain mapping. The most elaborate are scanning laser range finders, such as the ones made by SICK [156]. Another solution that is suitable for UAVs is Cloud Cap Technology’s AGL laser altimeter [157]. This sensor interfaces directly with Cloud Cap’s Piccolo autopilots, and provides very accurate measurements of the vehicle’s height above ground level. The AGL laser altimeter useful range is limited to 370 meters. A third alternative and by far the cheapest is to use a camera to compute optical flow. In Marlow and Langelaan [59] it is shown that given a moving camera’s velocity and angular rotation, the range to a stationary obstacle can be determined using optical flow. Optical flow measurements tend to be very noisy, but have the advantage of using an inexpensive sensor. In this section a simulated sensor is used. It provides measurements of the form

\[ z = h_{UAV} - h_{\text{terrain}} + v_z \]  

(2.21)

where \( z \) is the measurement, \( h_{UAV} \) is the height of the UAV, \( h_{\text{terrain}} \) is the height of the terrain directly below the UAV, and \( v_z \) is zero mean Gaussian noise, with variance \( \sigma_z^2 \).

In order to build the terrain map, the flying area is first divided into a grid. The finer the grid is, the more accurate the map will be, but at the expense of additional memory and processing requirements. In this section, the grid cell size is fixed at 10 meters by 10 meters. Each grid cell is initialized with a mean height \( \mu_h \) and a height standard deviation \( \sigma_h \). If some information about the terrain is known in advance, it should be reflected in these values. Otherwise, the mean should be given a reasonable
value (for the flying area used in this section all grid cells are set to 390 meters above sea level), and the standard deviation should be given a large value (in this case 25 m).

Because the system is linear, and has no process noise or dynamics, it can be updated by a simple recursive least squares algorithm [10]. The fact that each grid cell is one dimensional (i.e. terrain height is the only relevant variable) makes the estimation problem even more simple. Each time a new estimate comes in, the grid is updated using Algorithm 2.2.

**Algorithm 2.2** Algorithm for the terrain map

1: Use range sensor to obtain an altitude measurement \( z = h_{UAV} - h_{\text{terrain}} + v_z \).
2: Determine which grid cell the UAV is directly over.
3: Update the mean and variance of the grid cell using the recursive least squares estimator:

\[
K = \frac{\sigma_h^2}{\sigma_h^2 + \sigma_z^2} \\
\mu_h = \mu_h + K[(h_{UAV} - z) - \mu_h] \\
\sigma_h^2 = (1 - K)\sigma_h^2.
\]

2.5.4 Tracking

When multiple target detections are received, the position estimate can be improved by fusing the measurements using a tracker. Three different trackers are used in this section. A static target EKF and a static target particle filter are used in simulation. A moving target EKF is used in the flight tests. Each of these trackers are described in the following sections.

2.5.4.1 Static target EKF

Each time a new measurement is received, the target’s state estimate is updated using the EKF equations (Appendix A.3). Normally, the EKF is used on a system with dynamics in the state equation. If we are only interested in a stationary target (such
as a red barrel), the state equation can be omitted. (i.e. $f(x) = x$ is used in the state update step of the EKF.)

The measurement equation for localizing ground targets using a UAV is:

$$
\begin{bmatrix}
\varphi \\
\epsilon 
\end{bmatrix} = h(x_{target}, x_{UAV}(k), v(k)) \tag{2.23}
$$

The function $h$ can be determined from the camera model described in Section 2.5.2. The equation should account for all sources of uncertainty through a nonlinear dependence on the noise term $v$, but due to the difficulty of doing so, it is assumed that the measurement noise is additive, in which case

$$
y = h(x_{target}, x_{UAV}(k)) + v(k). \tag{2.24}
$$

From this equation, the following matrices of partial derivatives are calculated:

$$
H = \frac{\partial h}{\partial x_{target}} \bigg|_{x_{target}(k)} \tag{2.25a}
$$

$$
M = \frac{\partial h}{\partial v} \bigg|_{x_{target}(k)} \tag{2.25b}
$$

In the case where the measurement equation is linear in the noise parameter $v$ the second matrix is simply

$$
M = \begin{bmatrix}
1 & 0 \\
0 & 1 
\end{bmatrix}. \tag{2.26}
$$

Also the covariance of the noise must be considered. It is assumed that $v$ is zero mean and Gaussian and has covariance

$$
R = E(vv^T) \tag{2.27}
$$

Once these quantities are defined, the standard EKF algorithm (Algorithm A.4) can be used to update the estimated target location at each time step.
2.5.4.2 Static target particle filter

A particle filter has also been implemented for tracking static targets. The particle filter follows the basic procedure outlined in Appendix A.4, with the exception that the state equation (Equation (A.22)) is given by \( f_k(x_k, w_k) = x_k \). In other words, there is no process noise in the system. Such a situation can lead to difficulties when the resampling step is carried out. In the worst case, known as sample impoverishment, all but one particle may be eliminated. The reason this can happen in systems with no process noise is that sampling from the state transition probability will not induce any diversity into the sample set. The easiest way to counteract this issue is to only resample occasionally instead of at every step. In this case, the weight of each particle is updated by the following equation during steps where resampling does not take place:

\[
  w_k^{[n]} = p(y_k|x_k^{[n]}) w_k^{[n]} \quad (2.28)
\]

In this example, resampling only takes place when a target that has been in the camera’s field of view for multiple image frames finally leaves the camera’s field of view.

Another way to add more diversity to the particle set is to add a small number of particles according to the measurement model instead of the motion model with the following probability [158]:

\[
  x_k^{[n]} \sim p(y_k|x_k) \quad (2.29)
\]

It is difficult to determine what the importance weight of these new particles should be. In Thrun et al. [158] it is suggested that the weights be determined by the integral

\[
  w_k^{[n]} = \int p(x_k^{[n]}|u_k, x_{k-1}) bel(x_{k-1}) dx_{k-1} \quad (2.30)
\]

where \( bel(x_{k-1}) \) is the belief calculated at the previous step. This integral can be difficult to compute, but simply setting the weights of these new particles to the average of the weights of the previously generated particles works well. There are also other methods for dealing with particle impoverishment such as regularization or Markov chain Monte Carlo (MCMC) move step [7, 159, 160]. These were not explored further for this application,
because the EKF proved to be effective for this problem. Given these mathematical preliminaries, a particle filter can be applied to tracking a stationary target using a terrain map and a visual camera.

The first time a target is seen, it is initialized using Algorithm 2.3. Note that this initialization process is different from that of the EKF described previously, because it explicitly takes uncertainty in the terrain map into account. This makes direct comparison of the two difficult, but it does allow the filter to better account for all possible sources of uncertainty in the initial estimate. Once the filter is initialized, it can be updated every time a new measurement comes in using the procedure in Algorithm 2.4.

2.5.4.3 Moving target EKF

The moving target extended Kalman filter utilizes a relative state vector [7], equal to the target’s state minus the UAV’s state:

\[
x \triangleq x^t - x^{UAV} = [x \ y \ z \ \dot{x} \ \dot{y} \ \dot{z}]^T, \quad (2.39)
\]

where \(x\), \(y\), and \(z\) are all expressed in the local North, East, Down coordinate frame. The reason for using this equation is that it keeps all dependencies on the UAV’s position in the state equation, so that the measurement equation can depend only on the current states. Also, one motion model can be used for a moving target with a static sensor, a static target with a moving sensor, or a moving target with a moving sensor. For these reasons, the moving target EKF was the only filter used in UAV surveillance field tests.

The state equation for the moving target EKF is as follows:

\[
x_{k+1} = F_k x_k + \Gamma_k v_k - U_{k,k+1}, \quad (2.40)
\]
Algorithm 2.3 Procedure for initializing the static target particle filter

1: Convert the camera’s pixel measurement to a bearing measurement using Equation (2.17).
2: Use Algorithm 2.1 to determine an initial guess of the target’s position in the NED frame.
3: Generate $N$ samples of the following random variables

\[ v_h \sim N(0, \sigma_h^2) \] \hspace{1cm} (2.31a)
\[ v_\varphi \sim N(0, \sigma_\varphi^2) \] \hspace{1cm} (2.31b)
\[ v_\epsilon \sim N(0, \sigma_\epsilon^2) \] \hspace{1cm} (2.31c)

where $\sigma_h^2$ is the uncertainty in the terrain map, as described in Section 2.5.3 and $\sigma_\varphi^2$ and $\sigma_\epsilon^2$ are the noise in the azimuth and elevation measurements, which are dependent on the camera and the image processing routine.

4: From the $N$ samples of the noise variables, generate the ABC coordinates of the $N$ particles using the following equations:

\[ \tilde{z} = h_{UAV} - h_{terrain} + v_h \] \hspace{1cm} (2.32a)
\[ \tilde{\varphi} = \varphi + v_\varphi \] \hspace{1cm} (2.32b)
\[ \tilde{\epsilon} = \epsilon + v_\epsilon \] \hspace{1cm} (2.32c)
\[ \tilde{x} = \tilde{z} \tan(\tilde{\epsilon}) \cos(\tilde{\varphi}) \] \hspace{1cm} (2.32d)
\[ \tilde{y} = \tilde{z} \tan(\tilde{\epsilon}) \sin(\tilde{\varphi}) \] . \hspace{1cm} (2.32e)

5: Transform the particle coordinates from the ABC frame to the NED frame using Equation (2.19), then translate the target location to the local frame.
6: Set the weight of each particle to

\[ w^{[n]} = \frac{1}{N}. \] \hspace{1cm} (2.33)
Algorithm 2.4 Static particle filter algorithm

1: Generate \( \tilde{n} \) new particles using the procedure described previously.
2: Set the likelihood of each of these particles to \( \frac{1}{N} \), where \( N \) is the previous number of particles.
3: Update the the value of \( N \):
   \[ N_{new} = N_{old} + \tilde{n} \]  
   (2.34)
4: For each particle, \( n \), determine the expected measurement
   \[ \begin{bmatrix} \varphi \\ \epsilon \end{bmatrix}_{\text{expected}}^{[n]} = h(x^{[n]}, x_{UAV}, 0), \]  
   (2.35)
where \( h \) is the measurement equation described in Section 2.5.4.1.
5: Evaluate the likelihood of each particle given the latest measurement
   \[ p(y_k|x^{[n]}_k) = \eta \exp \left\{ -\frac{1}{2} \left( \begin{bmatrix} \varphi \\ \epsilon \end{bmatrix} - \begin{bmatrix} \varphi \\ \epsilon \end{bmatrix}_{\text{expected}}^{[n]} \right)^T \Sigma^{-1} \left( \begin{bmatrix} \varphi \\ \epsilon \end{bmatrix} - \begin{bmatrix} \varphi \\ \epsilon \end{bmatrix}_{\text{expected}}^{[n]} \right) \right\}, \]  
   (2.36)
where \( \eta \) is a normalizing constant that ensures that \( p \) is a valid probability density, and the noise covariance \( \Sigma \) is given by
   \[ \Sigma = \begin{bmatrix} \sigma^2_{\varphi} & 0 \\ 0 & \sigma^2_{\epsilon} \end{bmatrix}. \]  
   (2.37)
6: Update the weight of each particle using Equation (2.30).
7: Normalize all the weights using the following formula
   \[ w_k^{[n]} = \frac{w_k^{[n]}}{\sum_{i=1}^{\tilde{n}} w_i^{[n]}}, \]  
   (2.38)
   (This step is not strictly necessary, but it does prevent weights from growing smaller and smaller as time goes on.)
8: Finally, once the target leaves the UAV’s field of view, resample according to the procedure given in Algorithm A.6.
where the state transition matrix $F_k$ is given by

$$
F_k = \begin{bmatrix}
1 & 0 & 0 & T & 0 & 0 \\
0 & 1 & 0 & 0 & T & 0 \\
0 & 0 & 1 & 0 & 0 & T \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
$$

(2.41)

(where $T$ is the sampling time), the first input matrix $\Gamma$ is given by

$$
\Gamma_k = \begin{bmatrix}
T^2/2 & 0 & 0 \\
0 & T^2/2 & 0 \\
0 & 0 & T^2/2 \\
T & 0 & 0 \\
0 & T & 0 \\
0 & 0 & T \\
\end{bmatrix}
$$

(2.42)

and $v_k$ is zero mean white noise, which enters the system as a disturbance on the velocity. The input $U_{k,k+1}$ is deterministic and accounts for the acceleration of the UAV. It is given by:

$$
U_{k,k+1} = \begin{bmatrix}
x_{UAV}^{k+1} - x_{UAV}^k - T \ddot{x}_{UAV}^k \\
y_{UAV}^{k+1} - y_{UAV}^k - T \ddot{y}_{UAV}^k \\
z_{UAV}^{k+1} - z_{UAV}^k - T \ddot{z}_{UAV}^k \\
\dot{x}_{UAV}^{k+1} - \dot{x}_{UAV}^k \\
\dot{y}_{UAV}^{k+1} - \dot{y}_{UAV}^k \\
\dot{z}_{UAV}^{k+1} - \dot{z}_{UAV}^k \\
\end{bmatrix}
$$

(2.43)

All of the terms in $U_{k,k+1}$ can be obtained from the UAV autopilot.

In the special case that the target is not moving, the initial target velocity, the uncertainty in target velocity, and the process noise $v_k$ are all set to zero. This means
that the velocity terms in the state are simply the velocity of the UAV. Conversely, if
the sensor does not move (for instance if it were on a helicopter rather than a fixed-wing
aircraft) the input $U_{k,k+1}$ would be zero and the velocity terms of the state would all be
due to the target motion.

The measurement equation for the moving target EKF is as follows:

$$h(x, w) = \begin{bmatrix} \tan^{-1}\left(\frac{y}{x}\right) \\ \tan^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right) \end{bmatrix} + w \quad (2.44)$$

This is similar to the measurement equation given in Section 2.5.4.1 except that now
$x$, $y$, and $z$ are given in the North, East, Down frame rather than in the aircraft body
frame. This means that an extra rotation must be performed by the image processing
software, but it allows the measurement equation in the EKF to depend only on the
states of the system. Linearization of this measurement equation for use in the EKF
proceeds identically to Equation (2.25).

### 2.5.5 Data association

As discussed in Section 2.1 and in Hall and McMullen [19], there are three steps
to performing data association: hypothesis generation, hypothesis evaluation, and hy-
pothesis selection. For the tracking problem discussed in this section, new measurements
are associated with existing tracks; therefore, the hypotheses take on the report-to-track
form. For the hypothesis generation step a rectangular gate (see Section 2.1.1.2 and
Blackman [1]) is used. Because there is a relatively small number of targets (less than
thirty), they are spaced at least ten meters apart, and measurements are only two dimen-
sional, the rectangular gate is favored over the more powerful, but less computationally
efficient ellipsoidal gate.

In the hypothesis evaluation step all report to track pairs that pass through the
rectangular gate are scored according to the likelihood that the report was generated by
the track (see Section 2.1.2). For this example, it is assumed that reports and tracks
are Gaussian; therefore, Equation (2.6) can be used to compute this probability. As
discussed in Blackman and Popoli [161], hypotheses corresponding to new tracks are also added to the table of distances. These new tracks are given distances that are larger than the distances for any of the gated hypotheses, but less than the un-gated hypotheses. This ensures that existing gated tracks are favored over new tracks, but that new tracks are favored over un-gated tracks.

Once these distances are calculated, hypothesis selection is carried out by using the method referred to in Section 2.1.3.1 as Blackman’s [1] suboptimal assignment two. Basically, this technique finds the most likely (i.e. minimum distance) report to track pair, assigns the report to the track, then eliminates the report and track from the table of possible hypotheses and repeats. Despite the fact that this heuristic method is suboptimal, it is chosen over more sophisticated methods due to its simplicity and the fact that targets are not extremely dense in this problem.

2.6 UAV surveillance tests

The algorithms described in the previous section were tested through a combination of simulations [48], flight tests, and post-processing data from flight tests on the ground. This section will present the results from all of these tests. The first subsection deals with how the simulations were setup, and the second deals with how the flight tests were setup. The next three subsections discuss the results for classification, simulations, and flight tests, respectively.

2.6.1 Simulation setup

In order to validate the fusion system for UAV surveillance prior to flight testing it onboard a UAV, a Matlab simulation was designed. The simulation of the UAV is quite simple. It is assumed that the UAV flies a raster scan pattern about the flying area at a constant speed and altitude. It is also assumed that the UAV is flying with its wings level while over the target area. The reason for this simplifying design choice was so that the simulation could focus on the fusion system without the extra complication of UAV dynamics. If a more faithful vehicle dynamic simulation is required, a simulation such as the one in Ross et al. [51] could be used. In order to simulate noise in the vehicle’s
GPS measurements, the fusion system only has access to a noise-corrupted version of the vehicle’s position.

2.6.1.1 Range Sensor Model

The first sensor that must be modeled is the range sensor. The range sensor is expected to provide measurements of the form

\[
z = h_{UAV} - h_{terrain} + v_z
\]

where \(v_z\) is zero mean Gaussian noise. The variance of the noise will depend on what type of sensor is being used. A laser has an accuracy of less than a meter, whereas optical flow will be much noisier. For the results presented in Section 2.6.4, a standard deviation of 1 meter was used.

The simulated range sensor uses terrain elevation information from the United States Geological Survey (USGS) National Map Seamless Server [152]. The data from USGS has been converted to an ArcInfo ASCII Grid data file. This file contains a grid of elevation values at 1/3 arc-second (about 10 meter) increments. Given this information, a simulated range measurement can be created by the following procedure:

1. Determine latitude, longitude, and altitude of the UAV.
2. Determine which grid cell the UAV is inside of.
3. Calculate the true range:

\[
r = h_{UAV} - h_{terrain}
\]

4. Generate a random number, \(v \sim N(0, \sigma_z)\).
5. Generate the corrupted measurement:

\[
z = r + v
\]
2.6.1.2 Camera Model

The camera model is meant to simulate a generic color camera with a resolution of 640x480. The simulation supplies the fusion system with a noise corrupted pixel location of the target, similar to the results the image processing program described in Section 2.5.1 would return. The camera model takes as an input the world location of a target and the world location of the UAV (from the UAV simulation). Using this information and values for the camera intrinsic parameters [116], the camera model determines the pixel coordinates of the target through a series of coordinate transformations.

For the purposes of the simulation, it has been assumed that both the UAV’s position and the target’s position have been expressed in a North, East, Down (NED) coordinate frame with its origin at the lower right corner of the flying area. If this is not the case, they can be transformed using equations such as those found in Stevens and Lewis [151]. Given the position of the target and the UAV in the local NED frame, the location of the target in the UAV’s NED frame is given by

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}_{NED} = \begin{bmatrix}
  n \\
  e \\
  d
\end{bmatrix}_{target} - \begin{bmatrix}
  n \\
  e \\
  d
\end{bmatrix}_{UAV}. \tag{2.48}
\]

Next, the target’s location is rotated into the UAV’s aircraft-body coordinate (ABC) frame by the following equation:

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}_{ABC} = R^{-1} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}_{NED}, \tag{2.49}
\]

where \( R \) is given in Equation (2.20). If the simplifying assumption is made that the UAV is flying with its wings level no angle of attack, the equation for \( R^{-1} \) takes the
much simpler form

\[
R^{-1} = \begin{bmatrix}
\cos(\psi_y) & \sin(\psi_y) & 0 \\
-\sin(\psi_y) & \cos(\psi_y) & 0 \\
0 & 0 & 1
\end{bmatrix},
\]

(2.50)

where \(\psi_y\) is the yaw of the UAV.

Now that the point is expressed in the UAV’s coordinate frame, it must undergo a nonlinear transformation into pixel coordinates [116]. The transformation is given by

\[
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix} = M_{\text{int}} \begin{bmatrix}
x \\
y \\
z
\end{bmatrix}_{ABC}.
\]

(2.51)

The matrix \(M_{\text{int}}\) is the camera intrinsic parameters matrix, which is given by

\[
M_{\text{int}} = \begin{bmatrix}
-\frac{f}{s_x} & 0 & o_x \\
0 & -\frac{f}{s_y} & o_y \\
0 & 0 & 1
\end{bmatrix},
\]

(2.52)

where \((o_x, o_y)\) is the pixel location of the image center, \(s_x\) and \(s_y\) are the size of a single pixel (in millimeters) in the horizontal and vertical directions respectively, and \(f\) is the focal length of the camera (in millimeters). The pixel coordinates of the image point are then given by

\[
x_{im} = \frac{x_1}{x_3}
\]

(2.53)

\[
y_{im} = \frac{x_2}{x_3}.
\]

(2.54)

Finally, to simulate inaccuracy in the pixel coordinates, zero mean Gaussian noise is added to \(x_{im}\) and \(y_{im}\).
2.6.2 Flight test setup

The UAV surveillance algorithms were tested onboard the Penn State UAV Lab’s UAV Testbed [45, 47]. The testbed consists of a working UAV system and a model aircraft flying field, which is used for flight tests. Major components of the UAV system include the airframe, an autopilot for low level stabilization and control, an onboard computer that runs the fusion software, and a number of different sensors. Each of these components is described in detail below.

2.6.2.1 Aircraft

The airborne platform utilizes a heavily modified R/C trainer aircraft, the SIG Kadet Senior (Figure 2.7). The Kadet was chosen due to its stable flight characteristics, slow flying speed, and good payload capacity. The specifications of the aircraft are shown in Table 2.1.

The flight control surfaces are driven by standard Futaba servos. Electric power for the flight controls is provided by a 4.8 volt, 0.5 amp-hour Nickel Metal Hydride (NiMH) battery pack. Power for the autopilot and computer is provided by a 11.1 volt, 4 amp-hour Lithium Polymer (LiPo) battery pack. The aircraft required several modifications in order to accommodate all the necessary sensors, and the additional weight that it has to carry. These include the following:

1. Increased fuel capacity for extended flight times (up to one hour)
2. Installation of heavy duty main and nose gear to support the heavy payload
3. Movement of the servos out of the central area of the fuselage to create an open fuel and payload space
4. Installation of the autopilot and control processors
5. Installation of pitot static tube mount
6. Installation of the GPS and communications antennas
7. Installation of a larger engine to accommodate higher payloads
Table 2.1: SIG Kadet Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wingspan</td>
<td>80 inches</td>
</tr>
<tr>
<td>Wing Area</td>
<td>1180 sq. inches</td>
</tr>
<tr>
<td>Length</td>
<td>64 3/4 inches</td>
</tr>
<tr>
<td>Empty Weight</td>
<td>6 1/2 pounds</td>
</tr>
<tr>
<td>Payload</td>
<td>7 1/2 pounds</td>
</tr>
<tr>
<td>Propulsion</td>
<td>0.91 cubic inch 4-stroke</td>
</tr>
</tbody>
</table>

2.6.2.2 Autopilot

The UAVs use a Piccolo Plus autopilot that is commercially available from Cloud Cap Technologies. The autopilot is used for flight stability and low level control. It is housed in a mount that provides shock and vibration isolation. The mount is located near the center of gravity of the aircraft. The autopilot connects to a GPS antenna in order to receive position information. It also communicates to the ground station via a 900 MHz radio link. It houses both inertial and air data sensors, and provides pulse width modulated servo outputs.

2.6.2.3 Airborne processor

The airborne processor is an Ampro ReadyBoard 800 Single Board Computer. The unit contains a 1.4 GHz Intel Pentium Processor running Microsoft Windows XP. The processor runs the data fusion software. It interfaces with the autopilot by an RS232
serial port and with other vehicles or the ground station via an 802.11b wireless card. It also interfaces with a Logitech Webcam through a USB port.

2.6.2.4 Flying field

The UAV surveillance tests were conducted at a model aircraft flying field in Centre Hall, Pennsylvania (Figure 2.8). The field provides a large grass runway, which is necessary for takeoff and landing. The field is also lined with red barrels, which are used as targets for the detection program. Additional targets could be added by placing a red exercise ball or a blue tarp on the ground. (Note the two red balls in the lower right corner of the image.)

![Model aircraft flying field](image)

Fig. 2.8: Model aircraft flying field

2.6.3 Classification results

The target identification subroutines were tested using flight data collected onboard the UAV described in Section 2.6.2. The UAV was equipped with a Logitech QuickCam webcam with a resolution of 640x480. The crisp classifier was tested in real time onboard the UAV in flight. Due to speed issues, the fuzzy logic classifier was tested by post-processing images collected from a previous flight [48].
2.6.3.1 Fuzzy logic classifier

An example image from the webcam is shown in Figure 2.9a. Figure 2.9b shows the results of passing this image through the membership functions “low hue OR high hue,” i.e. it finds all pixels with a red hue. White pixels are definitely red, black pixels are definitely not red, and gray pixels are in between. Note that a number of pixels that don’t appear to be red in the original image, are nonetheless marked as red. This is due to poor lighting. The saturation function takes care of this; Figure 2.9c shows the same image passed through the saturation membership function, now all the “washed out” pixels are excluded. Figure 2.9d shows the results of passing the two images through a fuzzy AND function. Notice that this image looks similar to the saturation image. This is because for most pixels the saturation confidence is lower than the hue confidence. Since a fuzzy AND is a minimum, the lower value dominates. If this had been a well lit image with many colors, hue would have dominated the calculation instead. Finally, Figure 2.9e shows the final processed image with the three red barrels circled.

In a sequence of 105 images, with a total of 278 targets, there were 6 missed detections and 125 false positives. The missed detections are not a very serious concern since most barrels appear in multiple images, and are usually detected in at least one of those images. The false positives are a greater concern because they can start false tracks (i.e. start to track the location of an object that does not exist) or they can incorrectly update an existing track. The best way to deal with these false alarms would be to require a higher confidence that a target is a red barrel or red ball before accepting it.

2.6.3.2 Crisp classifier

Figure 2.10 shows an example image that has been processed by the crisp red ball and barrel classifier. Note that the classifier correctly classifies the red barrel (at the bottom center of the image) and the red ball (in the middle), but that it incorrectly classifies the orange windsock (in the upper right) as a target. Because false positives are a greater concern than false negatives, the crisp classifier was tuned to be much more selective than the fuzzy classifier was. In a series of 477 images with 218 targets, there were 99 missed detections and only 10 false positives.
Fig. 2.9: Image processing results
Figure 2.11 shows an example image that was processed by the crisp blue tarp classifier. The blue tarp classifier was tested on 1016 images with 41 targets. The reason that there are so many images with so few targets is that there was only one blue tarp on the field as opposed to twenty red barrels and balls on the field. In these tests, there were only five missed detections and one false alarm.

Fig. 2.10: Image processed by the crisp classifier with the red ball and barrel correctly identified and the orange windsock incorrectly identified

### 2.6.3.3 Analysis

Table 2.2 summarizes the performance of each classifier. Here sensitivity is defined as

\[
S = \frac{TP}{TP + FN},
\]  

where TP is the number of true positives (targets correctly identified as targets) and FN is the number of false negatives (targets that are not detected), it is a measure of what percentage of the targets that are present are detected. Positive prediction value (PPV) is defined as

\[
PPV = \frac{TP}{TP + FP},
\]  

where FP is the number of false positives (items that are not targets that are classified as targets). It is a measure of the percentage of the items that are classified as targets
Fig. 2.11: Image processed by the crisp blue tarp classifier

that actually are targets. Accuracy is defined as

$$A = \frac{TP}{TP + FP + FN}.$$ (2.57)

Accuracy is the total percentage of items (targets and non-targets) that are correctly classified. Traditionally accuracy would include a term for true negatives (non-targets correctly identified as non-targets), but it is difficult to define what is a true negative in this problem because some images contain multiple targets and some contain none.

Because the three classifiers are tested on different datasets, and they are meant to classify different things, it is difficult to say which one is definitively better or worse. Some general observations are that the fuzzy logic barrel classifier and the crisp barrel classifier have comparable accuracy, but the fuzzy classifier was tuned to have a high sensitivity at the cost of allowing many false alarms, whereas the crisp classifier was tuned to have a lower false alarm rate at the cost of lower sensitivity. The blue tarp classifier has a high accuracy and a low false alarm rate, with a sensitivity that is higher than the crisp barrel classifier and lower than the fuzzy barrel classifier. The higher accuracy of the tarp classifier has less to do with the fact that its a better classifier and more to do with the fact that the blue tarp is larger than the barrels and is distinct from other objects on the field (the orange windsock and certain buildings can appear red, but there is nothing else that is blue at the field). Therefore, since the main focus
of this thesis is data fusion, not classification, the blue tarp will usually be used as a target. Also, due to performance issues with the fuzzy logic classifier and the fact that similar accuracy can be obtained from the crisp classifier, the crisp classifier will usually be used.

Table 2.2: Accuracy of each classifier

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Sensitivity</th>
<th>PPV</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy barrel</td>
<td>98%</td>
<td>69%</td>
<td>67%</td>
</tr>
<tr>
<td>Crisp barrel</td>
<td>78%</td>
<td>94%</td>
<td>74%</td>
</tr>
<tr>
<td>Crisp tarp</td>
<td>88%</td>
<td>97%</td>
<td>86%</td>
</tr>
</tbody>
</table>

2.6.4 Simulation results

The terrain mapping method described in Section 2.5.3, the static target EKF (Section 2.5.4.1), and the static target particle filter (Section 2.5.4.2) were all tested using the simulation described in Section 2.6.1. None of these methods were tested in flight because they depended on an accurate range sensor, which was not available for the flight tests.

2.6.4.1 Terrain mapping

The terrain mapping function was tested using the simulation described in Section 2.6.1.1. To evaluate the performance of the terrain mapping, the UAV flew a raster scan flight pattern about the flying area and recorded range measurements 10 times a second. The generated map after one complete flight pattern was then compared to terrain data given in the USGS Seamless Data Distribution System [152].

Figure 2.12a shows the map generated by the terrain mapping system. Figure 2.12b shows the map as found in the USGS database. The units of both figures are meters from the reference origin. Qualitatively the two maps are similar, but the estimated one is much rougher. To quantify the performance of the mapping both maps were divided into 10 m grids, and the difference in height between the two maps for each
grid point was calculated. The mean square error was found to be 0.54 meters squared. This is sufficient accuracy for most tasks. In terms of efficiency, a single update takes an average of 0.006 seconds in Matlab, with a worst-case update time of 0.045 seconds. Therefore, update frequency will be more a function of the sensor update time than the computing time.

![Estimated terrain map](image1.png) ![Actual terrain map](image2.png)

Fig. 2.12: Comparison of the two terrain maps (units are meters relative to the reference origin)

### 2.6.4.2 Target state estimation

The target state estimation routine was tested using the simulation described in Section 2.6.1.2. Both the extended Kalman filter and the particle filter were tested by taking measurements 10 times a second as the UAV made a single pass over the target. At the end of the pass, the position estimate was compared to the actual position to determine the accuracy of the method.

Figure 2.13 shows a time history of the EKF for a single pass over the target. The solid lines show the x (north), y (east), and z (down) errors of the estimation. The dash-dot lines represent one standard deviation from the mean (as represented in the diagonal elements of the state covariance matrix). Notice that all three errors approach zero as the number of measurements increase, as is expected. Unfortunately, many of
the errors are more than one standard deviation from the mean. This is most likely due to the non-linear nature of the system.

Figure 2.13: Time history of EKF errors

Figure 2.14a shows the initial particle distribution for the particle filter. The ‘x’ represents the actual barrel location, the dots are particle locations, and the circle represents the weighted (by likelihood) average of the particle locations. The units are in degrees latitude and longitude. Figures 2.14b through 2.14f shows the time history of the particle distributions at one second increments. As time goes on, the average gets closer to the actual location, until the circle is right on top of the x. Figure 2.15 shows the final particle distribution after a re-sampling step takes place. Note that the distribution collapses into just a few particles. This is a common issue with particle filters, known as sample impoverishment [10]. This can be a problem for tracking a maneuvering target because the sample set lacks the diversity to account for a maneuver. It is not such a major issue in this case since the target is stationary.
Fig. 2.14: Time history of particle distribution (units are degrees latitude and longitude)
Fig. 2.15: Final particle distribution after re-sampling

Table 2.3 compares the performance of the two filters. The mean values are the average over 100 runs. The run times are for Matlab running on a 1.4 GHz processor. As is expected, the EKF is much faster than the particle filter (although the particle filter would still be suitable for real-time implementation if it was coded in C++ as opposed to Matlab). Also, the particle filter is slightly more accurate than the EKF. For this simple system it appears the EKF is a better choice due to its efficiency. In a more complicated system, however, the particle filter may yield better accuracy.

Table 2.3: Comparison of the two filters

<table>
<thead>
<tr>
<th></th>
<th>Mean Runtime (seconds)</th>
<th>Mean Error (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF</td>
<td>$6.3 \times 10^{-4}$</td>
<td>1.1</td>
</tr>
<tr>
<td>Particle Filter</td>
<td>0.17</td>
<td>0.78</td>
</tr>
</tbody>
</table>

2.6.5 Flight test results

Two different UAV surveillance flight tests were performed. In one flight test a single blue tarp was detected, located, and tracked. In the other flight test, a red ball and multiple red barrels were detected and located. Both tests were performed in real-time onboard the UAV, with an update rate of two images per second. The data from the second test was later post-processed using a tracker with data association.
2.6.5.1 Single target

The moving target EKF described in Section 2.5.4.3 was tested in flight onboard the Penn State UAV Lab’s UAV platform (Section 2.6.2). The moving target EKF was only tested using a static target and a moving sensor (the UAV). This was due to the fact that it was not possible to drive a vehicle across the flying area. Because the target was static, the initial velocity of the target and the uncertainty in the velocity estimate were both set to zero. The initial uncertainty in position was more difficult to determine. The geolocation routine uses altitude above ground level data coming from the Piccolo autopilot, which is based on the Shuttle radar topography mission (SRTM) [154]. The SRTM elevation data has an error of 9.0 meters over North America [155]. Therefore, the uncertainty in the position estimate was set to 10 meters in all directions.

For the flight test, a blue tarp was placed on the flying field, and the plane flew a rectangular pattern around the field, making several passes over the tarp. The tarp was detected using the crisp classifier described in Section 2.5.1, then the pixel location was used to initialize the tracker (as described in Section 2.5.2). When additional measurements were obtained, they were fed into the tracker in order to improve the position estimate.

Figure 2.16 shows the position errors in the north and east direction for the tarp tracking along with the Kalman filter’s two-sigma error contour. The errors and the uncertainty are both constant for an extended period of time while the target is out of the camera’s field of view, then change when the target comes back into view. Note that receiving multiple measurements does not always improve the state estimate, due to noise in the measurements. In fact, the measurement at 100 seconds slightly improves the east position estimate, but makes the north position estimate worse. However the measurement at 200 seconds does improve each estimate.

2.6.5.2 Multiple targets

When multiple targets must be located and tracked, some method is needed to distinguish between the targets. This was the case in the flight test which detected red barrels and a red ball, because there were 18 barrels and 1 ball for a total of 19 targets.
Fig. 2.16: Errors in the North and East direction for the flight test of the moving target EKF
In this test, the plane flew over the same flying area as in the last section, this time using the classifier which was tuned to find red objects. Initially, no tracking was performed. Instead, the localization routine described in Section 2.5.2 was used to translate each detection into a target location estimate. Figure 2.17a shows the results of performing this geolocation. Each blue dot is an estimated target location, the red ‘x’ is the location of the red ball, and the red circles are the locations of the red barrels. Note that there are many location estimates near the actual ball and barrel locations, but there are also a number that are several meters from the barrel locations due to noise, and a few that are nowhere near a barrel due to false positives. Also note that because some barrels were never seen by the camera, there are not any estimates near them. Figure 2.17b shows the results of running a k-means [92–94] clustering on the raw geolocation data. In this figure the dots are color coded according to which cluster they belong to and the small colored circles represent the cluster means. Some clusters correspond very well to targets. For instance, the cluster closest to the red ball is 7.3 meters from the correct location. Other clusters are much further from any of the targets, which is likely due to noisy measurements or false positives.

The data from this flight test was later post-processed using a tracker with data association. The tracker was the moving target EKF and the data association method followed the procedure described in Section 2.5.5. In this case, a gating constant of 7 was chosen by trial and error. Doing so generated 44 tracks. Figure 2.18 shows the final estimated location from each track along with the locations of the ball and barrels. From this figure, it is apparent that there are many false tracks which are due to false positive classifications. There are also several targets which have multiple tracks near them. This is due to the fact that a very noisy measurement would not satisfy the gate and would cause initiation of a new track. The nearest track to the red ball is 14.7 meters away from the actual location.

It appears that taking raw measurements and clustering them using k-means yields better results than performing data association and tracking. The downside to using k-means, however, is that it is a batch process, which can only be performed once all measurements are received. The data association and tracking algorithm, on
the other hand, processes measurements sequentially. If the targets had been moving, it would have been essential to estimate their velocity, therefore the tracker with data association would have to have been used.
Fig. 2.17: Plot of barrel locations (large red circles), red ball location (red ‘x’), and geolocation results (dots) before and after running k-means.

(a) Scatter plot of geolocation results

(b) Scatter plot after performing k-means clustering; dots are color coded by the cluster that they belong to; small colored circles correspond to cluster centers.
Fig. 2.18: Plot of barrel locations (red stars), red ball (red ‘x’), and location estimates as determined by the tracker with data association (blue circles)
Chapter 3

Distributed data fusion

All of the techniques described so far for performing data fusion have assumed that all processing occurs at one centralized location. This was typical in the early days of data fusion research, which focused on topics such as air traffic control and ballistic missile tracking. As sensors and processors grew smaller and cheaper, data fusion started to be applied to new domains, such as teams of mobile robots and sensor networks [162]. The techniques described in the previous chapter were still used for processing onboard each node, but new methods had to be developed for combining the data from multiple nodes.

An important type of distributed system is a decentralized data fusion system [163]. According to Durrant-Whyte a decentralized system must satisfy three characteristics:

- “There is no single central fusion center; no one node should be central to the successful operation of the network.”

- “There is no common communication facility; nodes cannot broadcast results and communication must be kept on a strictly node-to-node basis.”

- “Sensor nodes do not have any global knowledge of sensor network topology; nodes should only know about connections in their own neighborhood.”

These constraints are difficult to realize in practice, but they have important benefits. Durrant-Whyte list three such benefits related to the three constraints:

- “Eliminating the central fusion center and any common communication facility ensures that the system is scalable as there are no limits imposed by centralized computational bottlenecks or lack of communication bandwidth.”

- “Ensuring that no node is central and that no global knowledge of the network topology is required for fusion means that the system can be made survivable”
to the on-line loss (or addition) of sensing nodes and to dynamic changes in the
network structure.”

• “As all fusion processes must take place locally at each sensor site and no global
knowledge of the network is required a priori, nodes can be constructed and pro-
grammed in a modular fashion.”

There are great advantages to the decentralized approach, but because it is difficult to
achieve, most previous work has focused on distributed systems where there is some
global knowledge. Nonetheless, because scalability, survivability, and modularity are so
important, recent work has focused more on the decentralized case.

This chapter reviews some of the most important methods for distributed data
fusion. The first section describes some of the unique problems faced by a distributed
data fusion system. The second describes approaches for distributed classification, while
the third describes approaches for distributed tracking. In the final section, results from
a multiple robot search and rescue mission are presented. This mission builds on the
results shown in the previous chapter by incorporating a unmanned ground vehicle.

3.1 Distributed data fusion problem

There are many issues which must be dealt with in a distributed data fusion
situation that do not arise in a centralized system. One issue is a lack of processing
power onboard small mobile nodes. A small node simply does not have the processing
power of a large fusion center. As Moore’s law has progressed, this has become less of
an issue, but the related issue of power consumption still remains [164]. In particular, a
node may have sufficient computing power to perform a complex task, but doing so could
drain its battery, which would render it inoperable. Another issue unique to distributed
problems relates to the communications network. Section 3.1.1 deals with these types
of problems, which can include bandwidth, network delays, node failures, and network
topology. One of the most difficult issues in distributed fusion is the problem of fusing
correlated information. When nodes share information with one another, the data that
one node possesses becomes correlated with the data another node possesses. If this
correlation is not accounted for, performance can degrade. Section 3.1.2 describes this problem in more detail and provides examples of where it could arise.

3.1.1 Network issues

In distributed systems the structure of the communications network between nodes is an important issue. Of particular importance is communications bandwidth. Communications bandwidth could be limited by the given medium (because most distributed systems are wireless, this would depend on how many devices are communicating on a given frequency). Communications bandwidth could also be limited by power consumption, because it takes power to transmit data. Related to bandwidth is the problem of delays in communication. If it takes too much time for a piece of data to reach a fusion node, the node must have some way of incorporating this new data as if it had arrived at the correct time. Node failures are also an important issue in distributed systems. If a node fails, not only is the data from that node lost, but if it is the unique link between two sub-networks, then the two sub-networks would become separated and would no longer share data with one another.

Figure 3.1 [2] shows several possible network architectures for data fusion. In the figure, circles represent sensors and squares represent fusion nodes. The arrows indicate the direction which information flows in. Figure 3.1a is the centralized architecture described in the previous chapter. It is theoretically optimal because the fusion center has access to all the sensor data, but it is non-robust as it has a single failure point. It also puts a high computational burden on a single node when there are many sensors.

Figure 3.1b shows a hierarchical architecture without feedback. In this architecture, each fusion node processes local sensor data, then sends the processed data to other fusion nodes. This architecture alleviates many of the processing and communications issues that the centralized case possessed because processed data is more compact than raw sensor data. The price paid for this efficiency is that it lacks the optimality that the centralized case possessed, because the global fusion center does not have access to raw sensor data. The hierarchical architecture still has a single failure point at the top node in the tree, but if this node fails lower nodes could still carry on using only local data.
By adding feedback to the hierarchical architecture (Figure 3.1c), the lower nodes can gain access to the information that the global fusion center possesses. The price paid for this is an increase in communications load and a network that is harder to analyze in terms of correlation (see Section 3.1.2).

Hierarchical networks are the simplest distributed networks to analyze, but they lack robustness. The most robust communications architecture is a broadcast architecture (see Figure 3.1d). In a broadcast architecture all nodes communicate with all other nodes. Therefore, there is no single failure point to the broadcast architecture, and every node has access to every other node’s information in one time step. Unfortunately, for a large number of nodes, the communications burden of a broadcast architecture becomes infeasible.

In general, the communications architecture may not be known ahead of time. For instance, an architecture such as the one in Figure 3.1e may naturally arise if F1, F2, and F3 were sensing an area, then F4 came along later and started communicating with F2. An example of a network where the topology is constantly changing is given by Chong et al. [165], where a moving target is always tracked by the sensing nodes which are nearest to it, thus avoiding long distance communications. Because the structure of these types of networks are not known ahead of time, they are the most difficult to analyze [162].

### 3.1.2 Fusing correlated information

An important issue in distributed data fusion is dealing with the common information in two pieces of data to be fused. As an example, let $x$ be the state (e.g. the location of a target) to be estimated based on the measurements collected by two sensing nodes, which are capable of communicating with one another. Furthermore, assume that the individual measurements are conditionally independent given $x$, in other words the joint likelihood is given as

$$p(z_{ij}, z_{mn} | x) = p(z_{ij} | x)p(z_{mn} | x).$$

At a given point in time fusion node one’s measurement set is given as $Z_1 = \{z_{11}, z_{12}, z_{13}, \ldots \}$ and fusion node two’s measurements are given as $Z_2 = \{z_{21}, z_{22}, z_{23}, \ldots \}$. The goal of the fusion process
Fig. 3.1: Network architectures for distributed data fusion [2]
is to estimate the probability density of the state $x$ based on all of the available measurements. Adopting the notation of Liggins et. al. [166] the likelihood of the entire set of measurements given the state is

$$p(Z_1 \cup Z_2 | x) = \frac{p(Z_1 | x)p(Z_2 | x)}{p(Z_1 \cap Z_2 | x)}, \quad (3.1)$$

where the short hand notation

$$p(x|Z_1) \triangleq p(x|z_{11}, z_{12}, z_{13}, \ldots) \quad (3.2)$$

has been employed. This equation means the the likelihood of all the measurements is equal to the likelihood at each individual fusion node divided by the likelihood of the common measurements (the common measurements occur because of communications between the two sensing nodes). If there are no common measurements, the term in the denominator is simply one. In order to obtain a joint posterior probability density, Bayes’ rule and the result above are used to obtain [2,164,166–168]:

$$p(x|Z_1 \cup Z_2) = \frac{p(Z_1 \cup Z_2 | x)p(x)}{p(Z_1 \cup Z_2)} = C^{-1} \frac{p(x|Z_1)p(x|Z_2)}{p(x|Z_1 \cap Z_2)}, \quad (3.3)$$

where $C$ is a normalization constant. Here $p(x|Z_1)$ is the estimate of the posterior probability density at node one, $p(x|Z_2)$ is the estimate at node two, and $p(x|Z_1 \cap Z_2)$ accounts for the common information at each node. If there are no common measurements, the denominator simplifies to the prior $p(x)$.

Much of the literature on the correlation problem deals with tracking. In the distributed tracking problem, there are two sources of common information: common process noise and common prior errors [2]. Common process noise errors occur when two fusion centers, which are tracking the same target, both use the same dynamic model. These estimates are correlated because they will have the same errors in their dynamic model. Correlation due to common prior errors is best illustrated using an information graph [2,164]. Figure 3.2 shows an example information graph for the hierarchical fusion without feedback network in Figure 3.1b. Note that in the figure, F1’s estimate of the
target state at point A is communicated to F3. F1 continues to update its state based on this information as well as the sensor data it receives. Meanwhile, F3 fuses this new data with its own local estimate at point B, then continues to estimate the state using data from other sources. At point E, F3 receives a new estimate, C, from F1, which must be combined with its own estimate, D. Both of these estimates depend on the data at point A, therefore this common information must be addressed. There are a number of techniques for accomplishing this, which will be discussed in Section 3.3.

3.2 Approaches to distributed classification

The study of distributed classification is largely based on decision-level fusion (Section 2.2.3). Traditional approaches to decision-level fusion assumed a centralized architecture, where several simple classifiers would make decisions, then a master classifier would fuse these decisions. More modern approaches have studied what happens when the classification problem is distributed across a network. Many of the centralized techniques can still be applied as long as issues such as bandwidth and latency are considered. Some major methods for decision level fusion include probabilistic approaches [79], Dempster-Shafer theory [80, 104, 169], and heuristic techniques such as voting [19]. Each of these techniques will be reviewed below.

3.2.1 Probabilistic approaches

Probabilistic approaches are the most well-developed mathematical techniques for performing distributed classification. Probabilistic techniques can broadly be divided into classical inference and Bayesian inference [19, 79]. The most significant problem with any probabilistic technique is that they require that certain probabilities be known, at least subjectively, in order for any optimality results to hold. Nonetheless, they have found wide acceptance because of their mathematical rigor and because in most cases the true probabilities can at least be approximated. Most of the early research on these techniques focused on the problem of decision-level fusion (see Section 2.2.3); therefore, the studies were more concerned with how to fuse decisions at some central location, than with the communications issues that accompany distributed fusion problems. With
Fig. 3.2: Information graph for hierarchical fusion without feedback [2]
the advent of wireless sensor networks, distributed fusion became more important and studies began to concentrate on issues such as network structure [22, 170] and fusion in systems with noisy communications channels [171]. Most techniques for performing distributed classification assume that a fusion node receives all relevant information, then makes a decision, but Hussain [172] has studied the case where data arrives sequentially then a decision is made when sufficient evidence to support one decision over all others has arrived.

One popular classical inference technique is the Neyman-Pearson (NP) technique, which maximizes the power to discriminate against false alarms [19]. Much of the research on distributed detection has focused on maximizing the probability of detection while holding the probability of false alarms at some fixed value [170, 173], a task for which the NP formulation is well-suited. Tay et al. [22] shows that for the NP formulation, a fusion tree with sufficiently many nodes, arranged in a tree of bounded height, has the same performance as a parallel network. These results do not necessarily hold for other formations (such as the Bayesian formulation); thus a major advantage of the NP formulation is that architecture is not a critical issue.

There are several limitations of classical inference [19, 79]: the first is that only two hypotheses can be tested at at time, the hypothesis $H_0$ and the null hypothesis, $H_1$; another important limitation is that classical inference techniques do not provide a method to take advantage of an a priori likelihood. This is important in cases where there are two possible hypotheses, but one is known to be more likely than the other. In this case, it should take more evidence to confirm the unlikely hypothesis than the likely one. Because of these limitations, Bayesian inference has been applied to the distributed classification problem. All Bayesian inference techniques are based on Bayes theorem [71, 79]:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)},$$

(3.4)

where $P(y|x)$ is the likelihood and $P(x)$ is the prior. In words these symbols mean:

$$\text{posterior} \propto \text{likelihood} \times \text{prior},$$

(3.5)
which gives an intuitive way of understanding how to update prior probabilities with new information. The denominator, $P(y)$, in the above equation is a normalizing term, which is given by:

$$P(y) = \sum_{i} P(y|x_i)P(x_i).$$  \hfill (3.6)

When given multiple statistically independent observations $\{y_n\}$, and a prior probability distribution $P(x)$, Bayes combination rule can be used to fuse information to form a new probability distribution [5,19]:

$$P(x|y_1, \ldots, y_n) = \frac{P(y_1|x) \cdots p(y_n|x)P(x)}{\sum_{i} P(y_1|x_i) \cdots P(y_n|x_i)P(x_i)}. \hfill (3.7)$$

In Bayesian inference problems, it is common to attempt to express the decision making problem as an optimization problem. Specifically, risks are assigned to each possible course of action. High risks are assigned to incorrect decisions (often false negatives are given higher risks than false positives, but this need not be the case), and little to no risk is assigned to correct decisions. In a two class problem the Bayes risk is [23,79]:

$$R = \sum_{i=0}^{1} \sum_{j=0}^{1} C_{ij} P_j P(H_i|H_j), \hfill (3.8)$$

where $R$ is the risk, $C_{ij}$ is the cost of declaring hypothesis $H_i$, given that hypothesis $H_j$ is true; $P_j$ is the prior probability of hypothesis $H_j$ being true; and $P(H_i|H_j)$ is the probability of declaring hypothesis $H_i$, given that hypothesis $H_j$ is true. This formula can further be simplified by assigning the probabilities of false alarm, detection, and missed detection as follows [23]:

$$P_F = P(H_1|H_0) \quad \hfill (3.9a)$$

$$P_D = P(H_1|H_1) \quad \hfill (3.9b)$$

$$P_M = P(H_0|H_1) = 1 - P_D. \quad \hfill (3.9c)$$
Using these quantities, Bayes risk is then expressed as [23]:

\[ R = C + C_F P_F - C_D P_D, \]  

(3.10)

where \( C_f = P_0(C_{10} - C_{00}), \) \( C_D = (1 - P_0)(C_{10} - C_{11}), \) and \( C = P_0 C_{00} + (1 - P_0) C_{01}, \) assuming the cost of an incorrect decision is higher than the cost of a correct decision.

The objective of the optimization problem is to find the decision rules at each individual sensor and at the fusion center that jointly minimize this risk. This is often done by first optimizing each individual sensor rule, assuming that all other sensor rules and the fusion rule are already optimized; then by optimizing the global fusion rule [23, 174–176]. One interesting aspect of this optimization is that even if sensors are statistically independent and identical, the decision rules at each sensor may not be the same. Carrying out this optimization is difficult for a large number of sensors, but techniques such as particle swarm optimization can be applied to solve the optimization problem [177].

In practice, this optimization may not be possible, because the sensors that will observe an entity may not be known a priori. Instead, a fusion center may simply receive a list of decisions \( \{D_i\} \) from a group of sensors along with the probability that this decision was reached given that the true observed entity was \( O_j \). Using this information, the fusion center can build a matrix with entries \( P(D_i|O_j) \), i.e. the probability that sensor \( i \) makes decision \( D_i \) when the actual target is \( O_j \). Given this data, a joint decision can be reached using Bayes combination rule (3.7). An example of using this technique in an identification-friend-foe-neutral (IFFN) system is given by Wilson [178] and reviewed in Hall et al. [19].

3.2.2 Dempster-Shafer theory

Dempster-Shafer theory [80, 104, 169] is meant to be a generalization of Bayesian probability theory. In traditional probability theory, probability is assigned to each element \( x \in X \) in a probability space \( X \). In Dempster-Shafer theory, a basic mass assignment \( m(A) \) is defined for certain subsets \( A \subseteq X \) of the probability space, known
as the focal sets. Unlike in probability theory, the focal sets are not constrained to be exhaustive or mutually exclusive. The basic mass assignment must satisfy the following properties [5, 80]:

- $m(A) = 0$ for all but a finite number of $A$ (called the focal sets of $m$).
- $\sum_A m(A) = 1$.
- $m(\emptyset) = 0$.

Note that in many cases, some of the mass is assigned to the set $A = X$, which basically means that it is uncertain where this mass should be allocated. In the extreme case that the only focal element is $A = X$ and $m(A) = 1$, there is total uncertainty. In the case where the sets $A$ are the singletons, i.e. $A_i = x_i$, the basic mass assignment is also a probability distribution.

From the basic mass assignment three more quantities: belief, plausibility and, commonality, can be derived for any subset of $X$ (whether or not it is a focal element). The belief, plausibility, and commonality are defined as [5, 80]:

\[
\text{Bel}_m(A) = \sum_{B \subseteq A} m(B) \quad (3.11a)
\]

\[
\text{Pl}_m(A) = 1 - \text{Bel}_m(A^c) = \sum_{B \cap A \neq \emptyset} m(B) \quad (3.11b)
\]

\[
Q_m(A) = \sum_{B \supseteq A} m(B). \quad (3.11c)
\]

The belief is the minimum amount of evidence committed to $A$; it can be thought of as a lower bound on the probability of $A$, but it does not necessarily form an lower bound on probability in the rigorous sense [80]. The plausibility of $A$ is the amount of evidence that does not contradict $A$; therefore, it can be thought of as an upper bound on the probability of $A$. Note that if the focal elements are the singletons, the belief and plausibility are equal. The basic mass assignment can be recovered from the belief by
the Mobius transform [5]:

$$m(A) = \sum_{B \subseteq A} (-1)^{|A-B|} Bel(B),$$

(3.12)

where $|\cdot|$ indicates cardinality of a set.

Two basic mass assignments can be fused using Dempster’s combination rule [5, 169]:

$$(m_1 * m_2)(A) = \alpha^{-1} \sum_{A_1 \cap A_2 = A} m_1(A_1) \cdot m_2(A_2),$$

(3.13)

where the agreement $\alpha$ is defined as:

$$\alpha = \alpha_{DS}(m_1, m_2) = \sum_{A_1 \cap A_2 \neq \emptyset} m_1(A_1) \cdot m_2(A_2).$$

(3.14)

If the agreement is zero, the mass assignments are totally in conflict with one another and cannot be fused. A more compact way of expressing the combination uses the commonality [80,104]:

$$Q_{m_1 * m_2}(A) = \alpha^{-1} Q_{m_1}(A) Q_{m_2}(A).$$

(3.15)

There are numerous variations on Dempster’s combination rule, including cases that take into account prior information. When the only focal elements of one or both mass assignments are singletons, Dempster’s rule can be thought of as a variation on Bayes’ rule [5].

In distributed fusion cases, it is usually assumed that the output of each sensor is a basic mass assignment, then the mass assignments from each sensor are fused using Dempster’s rule [179,180]. Unfortunately, Dempster’s rule can be very inefficient to compute compared to Bayes’ rule [180]. It also does not necessarily yield greater accuracy than Bayes’ rule in many situations [179,180].

Xu et al. give another use of Dempster’s rule in decision fusion, where sensors make hard decisions, but the accuracy of each sensor is known [181]. Xu et al. empirically calculates two quantities: $\epsilon_r$, the sensor’s recognition rate (the rate at which it makes
correct decisions) and $\epsilon_s$, the substitution rate (the rate at which it makes incorrect decisions). The sensor is also permitted to reject a data point if it cannot reach a decision; therefore, the sum of these quantities can be less than one. When the sensor makes a hard decision $H_i$ on a data point, it assigns mass $\epsilon_r$ to that decision, mass $\epsilon_s$ to all other possible classifications, and mass $1 - \epsilon_r - \epsilon_s$ to the entire set of possible classifications. In the case where the sensor rejects the data point, mass 1 is assigned to the entire set. Because of the special structure of these mass assignments, Dempster's combination rule can efficiently be computed.

Because of its computational complexity, Dempster-Shafer theory has found only limited application in hard sensor data fusion. Dempster-Shafer theory appears to be most useful when operating on nontraditional information sources, such as data from soft sensors [24,84]. In cases where people may only be able to say that a probability lies in an interval, Dempster-Shafer theory is a good candidate for using this information. Fuzzy data can also be approximated in Dempster-Shafer theory using alpha cuts of the fuzzy membership function [24]. Dempster-Shafer theory is also a special case of random set theory [5], which will be explored as a method for fusing hard and soft data later in this thesis.

### 3.2.3 Heuristic techniques

Heuristic techniques for performing distributed classification are often chosen for their computational convenience [19]. They usually do not make any claims of optimality, rather they seek to make good decisions within the given computational constraints. The simplest heuristic is voting [181,182]. Voting models human group decision making by choosing the class that the majority of sensors believe a data point belongs to. The biggest problem with voting is that it weighs all classifiers equally, without taking into account how reliable individual classifiers are. Two possible extensions of voting are to completely eliminate classifiers which are known to be unreliable, or to weigh classifiers based upon their reliability. This can be done in a number of ways: for instance, a second classifier, known as a critic [182], can be trained to identify potentially erroneous classifications; it is also possible to weigh classifications based on empirical probabilities
of correct identification (found using the training data or test data) using Bayesian or Dempster-Shafer methods [181].

Another heuristic technique is ranking [183]. In ranking, the individual sensors rank each of the possible classes that a data point can belong to from the most likely class down to the least likely class. The role of the fusion center is then to either create a new ranking based upon the rankings of each sensor, or to create a set of possible classifications that is very likely to contain the true classification. This can be done by finding the union or intersection of the top $k$ hypotheses from each classifier. Ranking is more powerful than voting, because it uses more information, but it is also less general, because it imposes a structure on the individual classifiers [181].

A final class of heuristic techniques uses logic functions, such as AND or OR [184, 185]. AND fusion minimizes false positives at the expense of many false negatives, whereas OR fusion minimizes false negatives at the expense of many false positives. Asymmetric strategies are possible, which use AND for classes where it is desirable to minimize false positives and OR for classes where it is desirable to minimize false negatives [185]. In cases with more than two sensors, results can be combined in stages. For instance, two similar sensors may be combined with AND, then the result is fused with a different sensor using OR [184]. In cases where sensors output fuzzy values, as opposed to binary values, fuzzy logic [28] can be used, instead of Boolean logic. With fuzzy logic, it is even possible to fuse data using weighted AND and OR functions [67].

3.3 Approaches to distributed tracking

As discussed in Section 3.1.2, when data items to be fused are correlated, this correlation must be accounted for prior to combining the data, or results will be over-confident. This problem is especially prevalent in tracking problems; therefore, most of the research in this area has focused on tracking. Also, due to the popularity of the Kalman filter, most of the research has assumed that the data to be fused consists of a mean and covariance. The main techniques for performing fusion in tracking problems include naive fusion, cross-covariance fusion, information matrix fusion, maximum a posteriori fusion, and covariance intersection [2]. Naive fusion assumes that there is no
common information in the two estimates to be fused. It is optimal in such cases, but yields very poor performance when common information is present. Cross-covariance fusion assumes that the common information in two data items is due to process noise. It works well in situations where process noise is high (i.e. maneuvering targets), but it does not work well when the two estimates to be fused are both based on a common prior. Information matrix fusion ignores common process noise, but removes common prior information. Because it is relatively efficient, and yields very good results in low process noise systems when the prior information in two estimates to be fused is known, cross-covariance fusion has become a very popular method. Maximum a posteriori fusion accounts for both process noise and common prior information. This makes it very general, but it is not as commonly used as information matrix fusion because it is more complicated. Covariance intersection is a very conservative approach to fusion. It assumes that the two estimates to be fused have some correlation, but that it is unknown. It calculates an estimate that is consistent no matter what the prior information happens to be. This yields performance that is inferior to the other methods in situations where the common information is known, but superior performance when the common information is unknown.

3.3.1 Naive fusion

Naive fusion [2] assumes that the common information in two tracks to be fused is negligible. In other words, the term \( p(x|Z_1 \cap Z_2) \) in Equation (3.3) simply becomes \( p(x|\emptyset) \) (because \( Z_1 \) and \( Z_2 \) do not intersect). For the traditional linear-Gaussian case this reduces to the following:

\[
\hat{x} = P(P_i^{-1}\hat{x}_i + P_j^{-1}\hat{x}_j) \tag{3.16a}
\]

\[
P = (P_i^{-1} + P_j^{-1})^{-1}. \tag{3.16b}
\]

If the information filter is being used for tracking (Algorithm A.3) then the information matrix is defined as \( \mathcal{I} = P^{-1} \) and the information vector is defined as \( \xi = \mathcal{I}x. \)
Then Equation (3.16) can be rewritten as:

\[ \hat{\xi} = \hat{\xi}_i + \hat{\xi}_j \] (3.17a)
\[ T = T_i + T_j, \] (3.17b)

which may be simpler to compute if many data items must be fused.

In track fusion, track to track data association becomes important. Therefore, each fusion method has a corresponding association metric to determine which tracks to fuse, while taking into account any prior information. In the case of naive fusion, the association metric is the familiar Mahalanobis distance metric [2,186]:

\[ C_{ij} = \|\hat{x}_i - \hat{x}_j\|^2_{(P_i + P_j)^{-1}}. \] (3.18)

Naive fusion is simple to compute, and only requires communication of two pieces of information, the estimate and covariance. In cases where common information is negligible, it can yield satisfactory results, but when this is not the case, results can be very poor. The tracker can become overconfident in an erroneous measurement or association, and the track will eventually diverge from the correct value [2]. Therefore, more complicated methods for performing track fusion were developed.

### 3.3.2 Cross-covariance fusion

Cross-covariance fusion [2] takes into account the common process noise between two items to be fused, but it ignores common prior information. The equation for cross-covariance fusion is:

\[ \hat{x} = \hat{x}_i + (P_i - P_{ij})(P_i + P_j - P_{ij} - P_{ji})^{-1}(\hat{x}_j - \hat{x}_i) \] (3.19a)
\[ P = P_i - (P_i - P_{ij})(P_i + P_j - P_{ij} - P_{ji})^{-1}(P_i - P_{ji}), \] (3.19b)
where \( P_{ij} \) and \( P_{ji} \) are the cross-covariance terms between \( \hat{x}_i \) and \( \hat{x}_j \). The corresponding performance metric to determine which tracks to fuse is given by the \( \chi^2 \) test [2,186]:

\[
C_{ij} = \|\hat{x}_i - \hat{x}_j\|^2 (P_i + P_j - P_{ij} - P_{ji})^{-1}.
\]  

(3.20)

Cross-covariance fusion is an improvement over naive fusion, especially in cases where process noise is high. The price paid for this increase in performance is that it requires more information to be communicated. It can be shown that cross-covariance fusion is optimal in the maximum likelihood sense [81], but stronger optimality is not possible. Because cross-covariance fusion does not account for prior information, it is a poor choice when the prior information is not negligible.

### 3.3.3 Information matrix fusion

Information matrix fusion [2,21] takes into account the common prior information, but ignores process noise. It is optimal in situations where target motion is deterministic and when fusion nodes communicate after every update [81]. The equations for information matrix fusion are as follows:

\[
\hat{x} = P(P_i^{-1} \hat{x}_i + P_j^{-1} \hat{x}_j - \bar{P}^{-1} \bar{x})
\]  

(3.21a)

\[
P = (P_i^{-1} + P_j^{-1} - \bar{P}^{-1})^{-1},
\]  

(3.21b)

where \( \bar{x} \) and \( \bar{P} \) represent the common prior information. To see why this is referred to as information matrix fusion, express Equation(3.21) in information filter form (Algorithm A.3):

\[
\dot{\xi} = \dot{\xi}_i + \dot{\xi}_j - \xi_i
\]  

(3.22a)

\[
I = I_i + I_j - \bar{I}.
\]  

(3.22b)

The fusion algorithm then effectively subtracts out the common information terms, hence the name.
The performance metric for information matrix fusion is given by [2,186]:

\[
C_{ij} = \|\hat{x} - \hat{x}_i\|^2_{(P_i)^{-1}} + \|\hat{x} - \hat{x}_j\|^2_{(P_j)^{-1}} - \|\hat{x} - \bar{x}\|^2_{(\bar{P})^{-1}} \tag{3.23}
\]

Information matrix fusion is a very popular fusion method because it only requires one additional piece of information (compared to naive fusion) to be communicated. It yields very good results in low process noise systems and systems where target maneuvers can quickly be accounted for (such as in multiple model filtering).

### 3.3.4 Maximum a posteriori fusion

Maximum a posteriori fusion is the most complete fusion algorithm because it accounts for both common prior information and common process noise [2]. The equations for maximum a posterior fusion are more complicated than cross-covariance fusion and information matrix fusion. It also requires that more information be communicated at every time step. Because of this, it has not found as wide of acceptance as the other two methods. According to Liggins and Chang [2] the advent of adaptive multiple model fusion has made information matrix fusion the best choice out of naive fusion, cross-covariance fusion, information matrix fusion, and maximum a posteriori fusion.

### 3.3.5 Covariance intersection

In situations such as decentralized fusion, where nodes may be added or subtracted at any time, keeping track of cross-covariance information and prior information can become quite cumbersome. Rather than attempting to keep track of cross-covariance information or prior information, the covariance intersection method seeks to form a conservative estimate that is consistent no matter what the correlations are [3,82]. There are cases where it may be possible to keep track of correlations, but if they are poorly modeled or statistical independence assumptions are violated, results may still be unsatisfactory. According to Julier and Uhlmann [3], “statistical independence is an extremely rare property.” In these cases, covariance intersection is a good choice.
The basic equation for covariance intersection fusion is [3]:

\[ \hat{x}_{CI} = P_{CI} \left[ \omega P_i^{-1} \hat{x}_i + (1 - \omega) P_j^{-1} \hat{x}_j \right] \]  \hspace{1cm} (3.24a)

\[ P_{CI} = \left[ \omega P_i^{-1} + (1 - \omega) P_j^{-1} \right]^{-1}, \]  \hspace{1cm} (3.24b)

where \( \omega \) lies between zero and one and is chosen to minimize the trace or determinant of \( P_{CI} \). In other words, covariance intersection finds a convex combination of the two estimated covariances. Note that when \( \omega = 0.5 \) covariance intersection yields the same estimate as naive fusion, but with twice the covariance [2]. Julier and Uhlmann have proved that the covariance intersection algorithm is mathematically consistent (but not necessarily optimal) [3,187].

Covariance intersection can be visualized using a geometric interpretation [3]. It is well known that in two dimensions, the contours of equal probability of a Gaussian distribution are ellipses. Therefore, the covariances of the two estimates to be fused can be represented as ellipses. Figure 3.3 illustrates this. Sub-figure 3.3a shows two covariances to be fused. It can be shown that the covariance of any consistent fusion of the two must lie in the intersection of the two ellipses. Therefore, a conservative estimate should contain the entire intersection of the two ellipses. The remaining sub-figures show the covariance ellipses for different choices of \( \omega \). All choices are consistent, but which choice is best depends on the performance criterion (minimum trace or minimum determinate).

There are many possible variations on covariance intersection. Chong and Mori [188] provide a set theoretic formulation of covariance intersection that yields a tighter bound on the uncertainty. Efficient means of computing the parameter \( \omega \) also exist [38]. Covariance union [189] is a complementary technique to covariance intersection, which can be used to remove conflicting information from a system. Covariance intersection has also been applied with success to problems such as simultaneous localization and mapping (SLAM) [73]. Because covariance intersection is such a conservative method, it can yield suboptimal results in cases where there is known prior information; this has
(a) Two covariances to be fused
(b) Fused covariance for $\omega = 0.1$
(c) Fused covariance for $\omega = 0.5$
(d) Fused covariance for $\omega = 0.9$

Fig. 3.3: Example of geometric interpretation of covariance intersection [3]
motivated research into combining covariance intersection with information matrix fusion [20]. This yields better performance than either individual method in cases where some correlations are known and others are not. Although it was originally developed for uni-modal probability distributions, covariance intersection can be applied to more general probability distributions by expressing them as a Gaussian mixture [9]. Data from particle filters can also be fused by first converting the particle set to a Gaussian mixture then using covariance intersection [6].

3.4 Multiple robot collaboration

One important application for distributed data fusion is in multiple robot collaboration. The more data robots are able to share with one another, the better they will be able to collaborate. This section presents a simple multiple robot collaboration experiment designed as a first step toward distributed data fusion using both unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs). This mission builds on the UAV based search problem described in the first chapter by incorporating a UGV. It is also another step toward the full multiple robot, multiple human observer, search and rescue problem described in the Introduction. The specific mission presented in this section consists of the following steps:

1. A UAV, which is equipped with a webcam, flies over a field, which has a blue tarp placed somewhere in it.

2. When the UAV finds the tarp, it estimates the tarp’s location and sends the location to a UGV.

3. The UGV then proceeds to the location of the tarp.

The UAV for this experiment is the same SIG Kadet Senior model aircraft that was used in Section 2.6.2. The UAV detected the blue tarp using the classification method described in Section 2.5.1, then it estimated the tarp’s location using the procedure described in Section 2.5.2. Once it located the target, it sent the target location to the UGV using TCP/IP over an ad hoc wireless link.
Figure 3.4 shows the UGV, nicknamed the Tankbot, used in this experiment, along with the SIG Kadet Senior UAV. The Tankbot is a tread-propelled vehicle, which uses tank steering. It is equipped with a GPS receiver, a compass, and a single board computer. The single board computer runs the Penn State University Applied Research Lab Intelligent Controller (IC) program [67,190,191], which processes inputs from sensors and messages from other vehicles, and makes appropriate response decisions. When the IC receives the message from the UAV with the target’s location, it activates the traverse action, which guides the vehicle to the target’s location. A flowchart for the traverse action is shown in Figure 3.5.

![Tankbot UGV along with SIG Kadet Senior UAV.](image)

Fig. 3.4: Tankbot UGV along with SIG Kadet Senior UAV.

Figure 3.6 shows the results from the UAV and Tankbot collaborative search. The UAV correctly identified the target, and estimated its location to within ten meters (as shown by the green dot in the figure). Recall from Section 2.5.2 that the localization is based on projecting the image coordinates of the target onto a terrain map; therefore this is acceptable accuracy for such a simple procedure. The UAV then sent a message to the Tankbot’s IC, which activated the traverse action. The Tankbot then traveled from its initial location (the blue ‘x’ in the lower part of the figure) to a point seven meters from the actual target location (the blue circle in the figure).
Calculate the bearing from the vehicle to the target

Steer toward the target by speeding up the right tread and slowing down the left tread in proportion to the heading error

Drive forward at a set velocity

Target reached?

Stop

no

yes

Fig. 3.5: Tankbot IC traverse action
Fig. 3.6: Results from the UAV and Tankbot collaborative search, all units are meters north and east of the target location.
Chapter 4

Soft sensor data fusion

Just as hard sensor data fusion is an important problem in its own right, soft only data fusion is also a large area for research. Soft sensor fusion deals with information from sources such as domain experts, human observers on the ground, and information published in newspapers and online [27, 39]. These types of data are very different from the data that is traditionally handled by hard sensor data fusion techniques. According to Hall et. al. [39], “humans do not act as traditional sensors, and their accuracy, biases, and levels of observation are quite different from traditional sensors.” Data reported by humans is often in the form of natural language and can involve fuzzy descriptions and mixed Boolean and fuzzy logic reasoning [192].

This chapter focuses on algorithms for soft only data fusion. First, past approaches for soft sensor data fusion will be reviewed. Then, the method for performing soft only data fusion utilized in this thesis will be presented. The chapter will close by describing a series of tests of the soft fusion system in both simulation and the field.

4.1 Past approaches

There are many techniques for performing soft data fusion [4, 19, 24–27, 83]. One technique involves forming graphs of relations [4]. The relations are linguistic statements in the form of subject, predicate, object. The subject and object (nouns) are nodes in the graph and the predicate (verb) forms a directed arc which points from the subject to the object. For instance, the statement “a large gathering of youths are chanting anti-US slogans” would form the graph shown in Figure 4.1. As more data is collected, the graphs grow in size by adding new nodes and arcs to the graph. In addition to the graphs formed from analyzing data, there are also template graphs which represent certain situations (for instance an eminent terrorist attack), which are created by a
subject matter expert. Obviously, building the template graphs can be a very human-intensive task. In order to make inferences about the situation, the data graphs are matched to the template graphs by finding common nodes between the graphs. For instance, if there is a template graph corresponding to a youth uprising that contains the nodes “Large gathering”, “Youth”, and “Anti-US” then it could be inferred that the data graph in Figure 4.1 could correspond to a youth uprising.

![Diagram](image)

**Fig. 4.1: Example soft sensor data graph [4]**

Another graph-based approach utilizes fuzzy cognitive maps [26]. In this technique, a domain expert builds a graph with a series of inference relations. (For instance, if there is a fire in the biology lab and a suspicious person was seen near the biology lab prior to the fire, it may be possible to infer that the fire was caused by that person.) The fuzziness comes from the fact that relations may not be one to one. (For instance, just because there was a fire in the lab and a suspicious person was present does not mean that the person definitely set the fire; it could have still been an accident). In order to use this technique, items that are known to exist are set to 1, items that aren’t known to exist are set to 0, and the value of all unknown items are determined using the cause-effect relations in the graph. (With the unknown items being what is desired be inferred, i.e. “Is a terrorist attack eminent?”.) This technique can be used for hard/soft fusion since a fuzzy cognitive map shares some similarities (although there are also some important differences) with a continuous inference network [65,67], which has proved to
be useful for robotic systems. The big drawbacks seems to be that building the cognitive map is human intensive, and that the map can not represent situations the domain expert has never thought of.

Probabilistic methods have also been used for soft data fusion. In particular, Nevell et. al. [83] use Bayesian inference in order to fuse data from various intelligence analysts. By assigning a level of trust to each analysts, the authors are able to fuse information from multiple analysts, even when some are untrustworthy. Their model can take into account prior estimates of an analyst’s trustworthiness, or it can estimate the trust of each element by comparing their data to other analysts. For a small number of untrustworthy analysts, they obtain much better classification results than a model that does not estimate the trustworthiness of each analyst.

4.2 Fuzzy logic for soft sensor data fusion

For the research in this thesis a graphical user interface (GUI) was used to collect data from system operators. The system provided users with a display of data they entered as well as data collected from other operators. In order to fuse data from multiple users fuzzy logic was utilized. The reason for the choice of fuzzy logic over probabilistic methods was that fuzzy logic is better suited for situations where there is a lack of data for training a probabilistic model. The following sections explain how data was collected by the GUI and fused using fuzzy logic.

4.2.1 Soft fusion graphical user interface

In order to gather information from users of the system, a graphical user interface (GUI) was created using the Java programming language. The GUI accomplishes two tasks: first, it enables an operator to enter data using a familiar interface on a laptop or mobile device; second, it constrains the data that the user enters to a form that will be useful for the system. This second point is important. In a system where users are allowed to enter data in natural language they may enter a great deal of extraneous information, while at the same time omitting data that is potentially important. Dealing
with natural language issues such as this makes for an interesting research problem, but it was not explored here for the sake of keeping the research scope reasonable.

Initially, the GUI displays a set of cross-hairs, which represent the user’s current location (see Figure 4.2). The GUI has buttons at the top for exiting the program and entering a new report. There is also a button at the bottom for the user to tell the system where he is currently located (i.e. where to center the cross-hairs). In the future, this feature could be tied to GPS. When the user hits the “New” button, a dialog box similar to Figure 4.3a is displayed. This screen currently allows the user the opportunity to report seeing four different kinds of objects, but other types could easily be added. The next screen (Figure 4.3b) asks the user to enter his confidence in the classification on the previous screen. In a military situation, entities such as people, military units, and vehicles can be considered friendly, hostile, or neutral; therefore, the next screen (Figure 4.3c) gives the user opportunity to classify them as such. Note that the user also has the opportunity to classify entities as “Not a friend”, “Not a foe”, “Not neutral”, or completely unknown. The next screen (Figure 4.3d) asks for a confidence in this classification.

Once the user has entered an item’s classification, the GUI gathers data about the item’s state. In this example, the only state information collected deals with position, but the process for gathering velocity information is similar. Figure 4.3e shows the screen where the user enters a compass bearing to the target, the button at the center allows the user to enter unknown if a compass is not available. The next screen (Figure 4.3f) asks the user how far away the object is. The choices on this screen include: “Near”, “Medium”, “Far”, “Not Near”, “Not Far”, and “Unknown”. The “Not Near” choice is useful in situations where the user may know that the target is not close, but is not sure if it is medium or far (likewise for the “Not Far” choice). Once all this data is entered, the screen updates to a display similar to Figure 4.4. In this screen there is an icon for the entity type (a vehicle) and a polygon that outlines the area the entity could possibly be in (in this case, far away to the northeast). For ease of display, this polygon shows the area where the possibility is greater than 0.1, even though in reality this region is fuzzy. The red bar next to the icon is red to represent the fact that the
vehicle is hostile, the percentage of the bar that his shaded (in this case 70 percent) represents the classification confidence.

![Screenshot of GUI before any information is entered](image)

**Fig. 4.2:** Screenshot of GUI before any information is entered

### 4.2.2 Fuzzy logic representation of soft sensor data

When a user enters an item into the GUI a new sensor report is formed. The sensor report has two components: a classification and a state. The classification consists of seven membership function values, which correspond to all the possible classes that the item can belong to. Four of these values represent item type: \( \mu_{\text{person}}(x) \), \( \mu_{\text{vehicle}}(x) \), \( \mu_{\text{building}}(x) \), and \( \mu_{\text{intersection}}(x) \). Although these are membership functions, not probabilities, the fusion system still requires that they sum to one in order to be easily fused with probabilities later on. There are also three membership functions that
(a) Screen to enter classification

(b) Screen to enter classification confidence

(c) Screen to enter vehicle intent

(d) Screen to enter vehicle intent confidence

(e) Screen to enter vehicle location (bearing)

(f) Screen to enter vehicle location (distance)

Fig. 4.3: Screenshots of GUI screens for entering data
Fig. 4.4: Screenshot of GUI after a sensor report has been entered. The icon represents that the item is a vehicle. The color bar is colored red for foe. The color bar is 70 percent shaded to represent a medium confidence in the classification. The polygon around the icon represents the area where the car could be (with a possibility greater than 0.1).
represent the item’s friend, foe, neutral status, $\mu_{\text{friend}}(x)$, $\mu_{\text{foe}}(x)$, and $\mu_{\text{neutral}}(x)$, which also must sum to one.

The GUI accepts the user’s input and populates the membership functions with appropriate numeric values. Table 4.1 shows the numeric value corresponding to each possible confidence button. For instance, if the operator enters that the person confidence is high, the membership function for person will be set to 0.9 and all other membership function values will be set such that the total sums to one. The first row of Table 6.5 corresponds to this situation. The second two rows correspond to medium and low confidences for vehicle and building, respectively. Likewise, Table 4.3 shows the membership function values for several cases of friend, foe, or neutral confidences.

<table>
<thead>
<tr>
<th>Button pressed</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>0.9</td>
</tr>
<tr>
<td>Medium</td>
<td>0.7</td>
</tr>
<tr>
<td>Low</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 4.1: Numeric values corresponding to button presses

<table>
<thead>
<tr>
<th>Item class</th>
<th>Class confidence</th>
<th>$\mu_{\text{person}}(x)$</th>
<th>$\mu_{\text{vehicle}}(x)$</th>
<th>$\mu_{\text{building}}(x)$</th>
<th>$\mu_{\text{intersection}}(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>High</td>
<td>0.9</td>
<td>0.033</td>
<td>0.033</td>
<td>0.033</td>
</tr>
<tr>
<td>Vehicle</td>
<td>Medium</td>
<td>0.1</td>
<td>0.7</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Building</td>
<td>Low</td>
<td>0.17</td>
<td>0.17</td>
<td>0.5</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Table 4.2: Class membership function values for several user inputs

<table>
<thead>
<tr>
<th>Item FFN</th>
<th>FFN confidence</th>
<th>$\mu_{\text{friend}}(x)$</th>
<th>$\mu_{\text{foe}}(x)$</th>
<th>$\mu_{\text{neutral}}(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friend</td>
<td>High</td>
<td>0.9</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Foe</td>
<td>Medium</td>
<td>0.15</td>
<td>0.7</td>
<td>0.15</td>
</tr>
<tr>
<td>Neutral</td>
<td>Low</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 4.3: FFN membership function values for several user inputs

The second portion of the sensor report represents the state of the object. In this case, the state is simply the position of the object in a two dimensional world. Because of this, the state can be represented as a grid, with the value in each grid cell representing
the possibility that the item is in the given grid cell. If the state space were higher dimensional (such as when velocity is included in the state) a particle representation would be much more efficient. (See Section 5.4 for an example of such a case).

Figure 4.5 shows examples of fuzzy membership functions corresponding to various state parameters. The first three sub-figures correspond to the distance from the operator to the entity, which can be near, medium, or far. The horizontal axis is the distance from the user and the vertical axis is the confidence. For example, in Figure 4.5b a triangular membership function with its minimum at 100 meters, its peak at 300 meters, and its maximum at 500 meters represents medium distance. The final sub-figure (Figure 4.5d) represents a bearing measurement. A triangular membership function is once again used, but now the horizontal axis is a bearing in degrees.

As an example, consider when the operator enters that a target is a medium distance and to the northeast. The system first calculates the distance from the operator to the target at every grid point and the bearing from the operator to the target at every point. It then evaluates the two fuzzy membership functions $\mu_{\text{medium}}(\text{dist})$ and $\mu_{\text{NE}}(\text{bearing})$ and fuses them using fuzzy AND. In equation form:

$$
\mu_{\text{target}}(x) = \mu_{\text{medium}}(\text{dist}) \land \mu_{\text{NE}}(\text{bearing}).
$$

(4.1)

Figure 4.6 shows the final membership function for medium and northeast. Note that if performance is an issue these membership functions can be pre-computed and stored in memory, but it was not an issue in this system since operators tend to enter data much slower than the system calculates these functions.

### 4.2.3 Fuzzy logic based fusion

When an operator gathers more information about an entity in the world, or when two operators view the same target, the additional information must be fused with the information already in the system. This process can be divided into two steps: data association and fusion. Data association is the process by which reports from system
Fig. 4.5: Graphs of fuzzy membership functions
Fig. 4.6: Fuzzy membership functions for medium and northeast
operators are deemed to be of the same entity. Data fusion is the process by which new
data is used to update existing tracks.

Data association could be simple for a single user, who could assert that two
entities are the same. When more than one operator is present or a single user is not
sure if he has seen something before or not, data association becomes more difficult. This
is the situation that the following will focus upon. Algorithm 4.1 describes the process
by which a new sensor report is compared to all the tracks in the database in order to
determine if two tracks should be fused. For each track, the algorithm first compares
the classifications (friendly person, hostile person, friendly vehicle, etc.) of the track and
the new sensor report. If the classifications don’t match, the correlation is set to zero
and the next track is checked.

If the classifications match, the states of the two items are checked. Algorithm 4.2
describes how this is done. Recall that the state is represented as a grid where the value
in each grid point is the possibility that the item is located at that point. For each grid
point, the two possibilities are compared using fuzzy AND. Intuitively, this is checking to
see if both items could be located at this point. The correlation between the two points
is set to the highest such value (in other words if two items of the same classification are
very likely to be at the same point, then they are likely to be the same item).

Finally, the data association algorithm checks to see if any correlations exceed
a threshold. If multiple correlations are greater than the threshold, the largest one is
declared as a match. If none are greater than the threshold the new sensor report is
declared to be a new item and is added to the track database.

If the new sensor report matches one of the existing tracks, the two items are fused.
Algorithm 4.3 shows how this is performed. The first step is to fuse the classifications
using fuzzy AND. After the fusion, the membership function values are normalized so
that they all sum to one. Next, the state values are fused, once again using fuzzy AND;
this fusion is performed at every grid point. Once again, the values are normalized, but
this time the normalization ensures that the membership function has a value of one at
at least one grid point. The reason this normalization is used, rather than requiring that
all the grid points sum to one, is simply that all the values would end up being quite
Algorithm 4.1 Fuzzy logic data association algorithm

Require: New sensor report sr, set of existing tracks T

1: Determine most likely classification of sr: Class(sr) = max_c µ_c(sr)
2: for Each current track in database t ∈ T do
3: Check most likely classification of track: Class(t)
4: if Class(sr) = Class(t) then
5: Compare the states of sr and t
6: Set Correlation(sr, t) to the result of the comparison
7: else
8: Set Correlation(sr, t) = 0
9: end if
10: end for
11: if For each existing track t: Correlation(sr, t) < threshold then
12: Add sr to T
13: return No match
14: else
15: Find t* = arg max_t∈T Correlation(sr, t)
16: return Match(sr, t*)
17: end if

Algorithm 4.2 Algorithm to compare the states of two tracks using fuzzy logic

Require: Two states s_1 and s_2

1: Set Correlation(s_1, s_2) = 0
2: for Each grid point (x, y) in s_1 do
3: Evaluate µ_{s_1}(x, y) and µ_{s_2}(x, y), the possibilities that s_1 and s_2 are located at (x, y)
4: Set Correlation(s_1, s_2) = max(Correlation(s_1, s_2), µ_{s_1}(x, y) ∧ µ_{s_2}(x, y))
5: end for
6: return Correlation(s_1, s_2)
small if they were required to sum to one. In the end, the relative weights are much more important than the actual values.

**Algorithm 4.3** Algorithm to fuse two tracks

Require: Two tracks, $t_1$ and $t_2$

1: for All classifications, $c_i$ do
   2: Fuse classifications $\mu_{c_i}(\text{new}) = \mu_{c_i}(t_1) \wedge \mu_{c_i}(t_2)$
3: end for
4: for All classifications, $c_i$ do
5:   Normalize: $\mu_{c_i}(\text{new}) = \mu_{c_i}(\text{new}) / \sum_{c_j} \mu_{c_j}(\text{new})$
6: end for
7: for Each grid point $(x, y)$ in the new track do
8:   Fuse states $\mu_{\text{new}}(x, y) = \mu_{t_1}(x, y) \wedge \mu_{t_2}(x, y)$
9: end for
10: for Each grid point $(x, y)$ in the new track do
11:   Normalize: $\mu_{\text{new}}(x, y) = \mu_{\text{new}}(x, y) / \max_{(x,y)} \mu_{\text{new}}(x, y)$
12: end for

Figure 4.7 shows an example of how this fusion looks in the GUI. In Figure 4.7a a user views a building an unknown distance to the North. Figure 4.7b shows the same building viewed by an operator at a different location. When the operator at the second location enters that he views a building near him to his Northeast, the sensor report is fused with the original track, which results in Figure 4.7c. Note that in Figure 4.7c the uncertainty around the building’s location (the box around the building) is much smaller and that the confidence that the item is a building (the bar next to the icon) is much higher, because of the additional information in the second report.

### 4.3 Soft data fusion tests

In order to verify correct operation of the soft data fusion GUI, a series of tests were conducted. The basic setup for all of the tests was similar to Figure 4.8. In this example a single searcher or group of searchers attempts to find a missing pilot, while avoiding hostile forces. They may gather information from their own observations or by interviewing friendly villagers. Note that the purpose of this test was not necessarily to be completely realistic, but to show that the system functions correctly. The GUI was
(a) Building an unknown distance to the north

(b) Same building from a different angle

(c) Building after another report has been collected

Fig. 4.7: Screenshots of a location fusion example
tested through both a simulation on one or two computers over a wired network and in
the field using a laptop.

4.3.1 Simulations

The GUI was first tested on two computers over a wired network. The purposes
of this test were to verify that when operators entered data the reports appeared at the
correct place on the screen, and when the same target was viewed from multiple angles
the sensor reports were fused. There was not any way to evaluate the accuracy of the
system’s localization in simulation. Therefore, the hypothetical scenario which follows
sufficed to show that the system was working correctly and was ready for field trials.

Figure 4.9 shows a series of screenshots over the course of this test. In this
scenario, initially the two searchers only know that a missing pilot is somewhere to their
north. Figures 4.9a and 4.9b show the initial views for each searcher. They are the same
because each searcher is at the same location and has the same information. Next, the
first searcher spots an intersection nearby and a potentially hostile vehicle further ahead.
Figure 4.9c shows his view once this information is added. The second searcher’s screen
is identical.

At this point, the two searchers separated, with one going east and the other
going west. Figures 4.9d and 4.9e show each operator’s new view after moving. Notice
now that the two screens show the same information, but each user now sees the vehicle
and intersection relative to his own position.

Next, the first searcher meets a “friendly villager” who informs him that he saw an
aircraft go down far away to the northwest. He also informs him that the vehicle on the
road was indeed hostile and that their are also hostile ground troops a medium distance
away to the northwest. Figure 4.9f shows the first searcher’s screen after entering all
of this information. In particular, note that the blue polygon around the pilot is now
smaller because of the additional information from the villager. Also note that the red
bar next to the vehicle is larger and a brighter shade of red to indicate that the confidence
that the item is a hostile vehicle is higher.
Fig. 4.8: Setup for soft data fusion GUI tests
Because the first searcher is blocked by the hostile troops, the second searcher heads northeast to the area the pilot should be in, while avoiding all of the hostiles. When the pilot is spotted, the searcher enters that the pilot has been spotted near him to the east. Figure 4.9g shows the new display with the polygon around the pilot smaller yet to indicate this.

4.3.2 Field test

The soft fusion system was also tested outdoors on a laptop. Actual landmarks, with known positions, were used to represent the various entities, such as the pilot and the hostile vehicle. By doing so, it was possible to verify that the object locations estimated by the fuzzy logic system corresponded to their actual locations. The setup of this test was similar to the one done in simulation, except that there was only one user. Figure 4.10 shows a screenshot of the user’s display at the conclusion of the test.

Figure 4.11 shows the fuzzy logic representation of the location of the hostile vehicle, with a black “x” at the actual location. In the fuzzy logic representation, red represents areas of high possibility, while blue represents areas of low possibility. Units on the graph are meters east and north of an arbitrary reference location. As should be expected, the actual location of the target is near the area of highest possibility. Figure 4.12 shows the same type of graph for the pilot’s location. The actual location is near the area of maximum possibility.

In order to precisely evaluate the localization performance, the system’s fuzzy representations of the locations were defuzzified. To perform the defuzzification, the centroid defuzzification method was used [29]. The centroid defuzzification method converts a fuzzy set \( \mu_A(\cdot) \) to a crisp constant \( x \) as follows:

\[
x = \frac{\sum_i \mu_A(x_i) \cdot x_i}{\sum_i \mu_A(x_i)}
\]

(4.2)

In order to extend the centroid defuzzification method to two dimension, each dimension is defuzzified separately.
Fig. 4.9: Screenshots of each user’s GUI display during the simulation
Fig. 4.10: Screenshot of GUI during the field test
Fig. 4.11: Fuzzy representation of the location of the hostile vehicle
Fig. 4.12: Fuzzy representation of the location of the pilot
Figure 4.13 compares the actual locations of the hostile vehicle and the lost pilot with those calculated by the defuzzification method. The actual locations are very close to the estimated ones. Table 4.4 list the actual and estimated locations of the hostile vehicle and the pilot (once again in meters from an arbitrary reference), along with the error. The error in the location the hostile vehicle is 4 meters and the error in the location of the pilot is 13 meters.

Fig. 4.13: Actual and estimated locations of the pilot and the hostile vehicle
<table>
<thead>
<tr>
<th>Item</th>
<th>Actual location [m]</th>
<th>Estimated Location [m]</th>
<th>Error [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostile vehicle</td>
<td>$(-47, -38)$</td>
<td>$(-43, -39)$</td>
<td>4</td>
</tr>
<tr>
<td>Pilot</td>
<td>$(-77, -72)$</td>
<td>$(-68, -82)$</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4.4: Errors in localization for soft data test (coordinates are in east, north reference frame)
Chapter 5

Hard and soft sensor data fusion

This chapter presents methods for fusing data from both traditional hard sensors and human-based soft sensors. It begins by surveying some of the past approaches for doing so. Then, random set theory [5,25,85–88] is presented as a powerful mathematical tool for handling data from both hard and soft sources. Once random set theory has been introduced, mathematical formulas for representing various types of hard sensor and soft sensor data using random sets are presented. Finally, with these mathematical preliminaries in place, the final section shows some concrete algorithms of processing both hard and soft data using a particle filter, including a novel particle filter called the distributed, hybrid, generalized particle filter (DHGPF).

5.1 Past approaches

Combining hard and soft data is a challenging problem, but there are a number of techniques that show great promise [24, 25, 27, 84]. Most of these techniques focus on fusing fuzzy information with probabilistic information. The reason for this is that most hard fusion techniques are probabilistic and many soft techniques are fuzzy. Both fuzzy logic and probability deal with uncertainty, but they deal with it in different ways. Fuzzy logic deals with a degree of membership (i.e. “The item has a 60% membership in the class obstacle.”). Probability deals with the chances of a random event occurring (i.e. “There is a 6 in 10 chance that there is an obstacle in my path”). In Florea et al. [25], random set theory [5] is used to fuse fuzzy and probabilistic information. Random set theory is a very general method for dealing with uncertainty that encompasses traditional probability theory, fuzzy logic, and Dempster-Shafer theory [104]. A random set is a generalization of the traditional random variable. The value of a random set is a set, whereas the value of a random variable is a number. To fuse fuzzy and probabilistic
knowledge using random set theory, both types of knowledge are represented as a random set, then they are fused using the laws of random set theory.

It is also possible to combine fuzzy and probabilistic information using Dempster-Shafer theory (also known as evidence theory) [24]. Evidence theory is more general than probability theory in that it can deal with situations where probability is assigned to overlapping sets or when probabilities can lie in an interval. (See Section 3.2.2 for more details on Dempster-Shafer theory.) As in random set theory fusion, evidence theory involves converting both forms of information to a form usable by evidence theory (known as a belief function, not to be confused by the concept from fuzzy set theory with the same name) then combining them using a combination rule (there are many such rules). It is straightforward to convert probabilistic information to a belief function because evidence theory is a generalization of probability theory. Converting fuzzy information is slightly more difficult and often requires an approximate technique [24].

Another method for combining hard and soft data uses Dempster-Shafer theory directly, without using fuzzy logic or probability theory as an intermediary [84]. Using such a method avoids some complications since there is no need to convert between different ways of expressing information. The downside appears to be that if a hard fusion system that utilizes probability theory has already been designed it must be modified to use evidence theory. Also, when eliciting information from soft sensors, they must provide information in the form of a belief function, which may be more difficult to express than fuzzy information.

5.2 Random set theory for data fusion

The approach used here for fusing hard and soft data is based on random set theory [5, 25, 85–88, 193]. Random set theory is used because it is an extremely general method for dealing with uncertainty, whether that uncertainty is probabilistic, fuzzy [194, 195], or Dempster-Shafer [196–198] in nature. This section first discusses the theoretical basis behind using random set theory to fuse hard and soft data. Next, the random set theory-based modeling approach to data fusion known as formal Bayes modeling
is introduced. Finally, then the generalized Bayes filter is discussed as a method for processing random set data.

5.2.1 Theoretical basis for random set theory

The power of random set theory is that it models data that is difficult to statistically characterize due to ambiguity (either in the data itself or in how it is generated). It also provides a framework for using this data to construct state estimates. According to Mahler, there are four key ideas that random set theory uses to handle nontraditional data types [199]:

1. “Ambiguous data can be modeled by randomly varying closed subsets of measurement space.”

2. “Bayes’ rule is very general, in the sense that we can construct posterior distributions conditioned on non-Bayesian data.”

3. “Likelihood functions for ambiguous data can be constructed by matching observations to model signatures.”

4. “The nonlinear filtering equations [i.e. the Bayes’ filter of Appendix A.1, Algorithm A.1] can be generalized to incorporate ambiguous data.”

The first assertion is a well known property of random set theory [5, 200, 201], and is addressed in Appendix C.2. The second assertion essentially means that it is possible to use Bayes’ theorem to construct a posterior of the form $P_r(x|E)$ from a generalized likelihood function $P_r(E|x)$, where $x$ is a random vector and $E$ is any event. The third assertion describes how such a likelihood can be constructed, it should represent the degree to which the measurement “matches” the signature (i.e. the measurement that a given state is expected to produce). There are actually several forms such a likelihood could take on [199,200,202]. Appendix C.3 gives several examples of such likelihoods for the types of data dealt with in this research. The fourth point follows directly from the first three, and essentially means that ambiguous data (such as soft sensor data) can be handled within the familiar Bayes’ filter context.
There are several theoretical issues with this formulation. First, is that the approach is not strictly Bayesian, because it uses generalized likelihoods, rather than Bayesian likelihoods. Second, there is a heuristic element to how the likelihood is constructed, since the notion of the measurement “matching” the state is problem dependent. Third, is that the random set representation of the measurement is itself based on heuristics such as fuzzy sets or Dempster-Shafer evidence. Mahler argues that such heuristics are unavoidable due to the ambiguity inherent in the data [202]. If it were possible to precisely model human data statistically (such as by collecting sufficient training data to create a statistical model of what people mean by terms such as “near”), then there would be no need for using random set theory in the first place.

Despite these theoretical difficulties, random set theory has some desirable properties, which make it well suited for the fusion problem addressed in this research. The most important is that it provides a unified theoretical framework for handling all of the types of uncertainties (whether statistical, fuzzy, or possibilities) that are encountered in this work. This means, for instance, that the Bayes’ filter can be used process fuzzy data in the same manner as statistical data. Whether or not all of the optimality results of the Bayes’ filter hold in this framework is still an open research problem, but there are examples where the generalized Bayes’ filter yields the same results as a conventional Bayes’ filter. For instance, if the observations take the form of Gaussian-shaped fuzzy membership functions, then the generalized Bayes’ filter yields the same results as a conventional Bayes’ filter using Gaussian random variables [199]. This means that two data types with very different semantics (one represents the degree to which an item belongs to a fuzzy set, the other represents a random variable) can be handled the same way mathematically. This is a promising result, and helps to justify further exploration of random set theory as a unified fusion framework.

5.2.2 Formal Bayes modeling

Formal Bayes modeling is a term coined by Ronald Mahler to describe the process by which non-traditional states and measurements can be modeled in a principled (rather than ad hoc) mathematical framework [5, 85–88]. Once the data is modeled in such a
way, it can be processed using a generalized Bayes filter. There are seven stages to formal Bayes modeling [5].

- **State space:** “Carefully define a state space \( \mathcal{X} \) that uniquely and exhaustively specifies the possible states \( \xi \) of the physical system—meaning those aspects of physical objects that are of interest to us.” In this thesis the state space is always either Euclidean space \( \mathcal{X} = \mathbb{R}^N \) or Euclidean space along with a finite set \( \mathcal{X} = \mathcal{X}_0 = \mathbb{R}^N \times C \). For instance, if a target could be a car, truck, or motorcycle and we are interested in its two dimensional position and velocity the state space would be \( \mathcal{X} = \mathbb{R}^4 \times C \) where \( C = \{ \text{car, truck, motorcycle} \} \).

- **Measurement space:** “For each sensor carefully define a measurement space \( \mathcal{Z} \) that uniquely and exhaustively specifies the individual pieces of information \( \zeta \) that a sensor can provide.” For the traditional hard sensor fusion techniques described in earlier chapters the measurement space is \( \mathcal{Z} = \mathbb{R}^M \). When nontraditional sensor types (such as soft sensors) are included, the measurement space becomes a hyperspace, which consists of all random closed subsets \( \Theta \) of a “base” measurement space \( \mathcal{Z}_0 = \mathbb{R}^M \times D \), where \( D \) is finite.

- **Integration:** “Define integrals \( \int_\mathcal{S} f(\xi) d\xi \) and \( \int_\mathcal{T} g(\zeta) d\zeta \) of state and measurement variables, respectively.” In this thesis, the state space is the space \( \mathcal{X}_0 \). The integrals of functions of the state variables take on the following form:

\[
\int_\mathcal{S} f(\xi) d\xi = \int_\mathcal{S} f(x) dx \triangleq \sum_c \int_\mathcal{I}_S(u,c) f(u,c) du, \tag{5.1}
\]

where \( \mathcal{I}_S \) is the indicator function of set \( \mathcal{S} \). This is simply an extension of the traditional Lebesgue integral to spaces that have both continues and discrete components. The measurement space consists of random closed subsets \( \Theta \) of the base space \( \mathcal{Z}_0 \), but it is only necessary to define integration over the base space, which can be done analogously to the state space.

\[
\int_\mathcal{T} g(\zeta) d\zeta = \int_\mathcal{T} g(z) dz \triangleq \sum_d \int_\mathcal{I}_T(v,d) g(v,d) dv \tag{5.2}
\]
• **State-transition model:** “Construct a state-transition model (typically, a motion model) $\xi \leftarrow \xi'$ describing how the object-state may change from $\xi'$ at measurement collection time step $k$ to $\xi$ at measurement collection time step $k+1$.” In this thesis all such state models are motion models, which are nonlinear functions of the state and a white noise term of the form $x_{k+1} = \varphi_k(x_k, w_k)$ (as in Equation (A.1) with a slight change in notation).

• **State-transition density:** “From this model construct a state-transition density $f_{k+1}(\xi|\xi', Z^K)$ that is normalized (i.e., $\int f_{k+1}(\xi|\xi', Z^K)d\xi = 1$) and that describes the likelihood that the object will have state $\xi$ at time step $k+1$, if it had state $\xi'$ at time step $k$ and observations $Z^k : \zeta_1, \ldots, \zeta_k$ have been collected at previous time steps $i = 1, \ldots, k$.” In this thesis, the state is in the space $X_0$ and it is assumed that the predicted state of an object depends only on its current state; therefore, the transition density will take on the familiar form $f_{k+1}(x|x')$ (as in Equation (A.3)).

• **Measurement model:** “Construct a measurement model $\zeta \leftarrow \xi$ that describes how measurements $\zeta$ are generated by objects with specified states $\xi$.” In this thesis, all measurements fall into the class that Mahler refers to as unambiguously generated ambiguous, i.e. there is a precise equation of the form $z = \eta(x)$ that maps from states to measurements. Any uncertainty in the measurement is due to noise or imprecision in the sensor itself.

• **Likelihood function:** “From this model construct a likelihood function $L_{\zeta}(\xi) = f_{k+1}(\zeta|\xi, Z^K)$ that is normalized — $\int f_{k+1}(\zeta|\xi, Z^K)d\zeta = 1$ — and that describes the likelihood of observing $\zeta$ if an object with state $\xi$ is present and observations $Z^k$ have been previously collected.” In this thesis, the likelihood can take on two forms. For traditional hard sensor measurements (assuming the current prediction depends only on the current state), the likelihood takes on the form $f_{k+1}(z|x)$ (as in Equation (A.4)). For soft sensor data, the measurement is a random set; therefore, the likelihood takes on the following more general form:

$$f_{k+1}(\zeta|\xi, Z^K) = f_{k+1}(\Theta|x) \triangleq \Pr(\eta(x) \in \Theta) \quad (5.3)$$
(where it has once again been assumed that the current measurement depends only on the current state). This equation is further developed for several special cases in Appendix C.3).

5.2.3 The generalized Bayes filter

The generalized Bayes filter \[200\] is an extension of the Bayes filter (Section A.1) to nontraditional forms of data. It estimates the state of a dynamic system from a sequence of measurements by propagating the Bayes posterior probability density function \( f_{k|k}(\xi|Z^k) \) through time steps \( k = 0, 1, \ldots k \). Figure 5.1 illustrates the generalized Bayes filter graphically. The filter consists of the following steps \[5\]:

- **Initialization:** This involves selecting a prior density of the form \( f_{0|0}(\xi|Z^0) = f_{0|0}(\xi) \). In this thesis, because the state is in Euclidean space, the prior density takes on the same form as Equation (A.2), i.e. \( f(x_0|Z_0) = f(x_0) \).

- **Predictor:** The predictor equation accounts for the motion of the target and the increased ambiguity due to the state transition. In the most general case, it takes on the following form:

\[
f_{k+1|k}(\xi|Z^k) = \int f_{k+1}(\xi|\xi')f_{k+1|k}(\xi'|Z^k)d\xi
\]  

(5.4)

When the state is in Euclidean space, the predictor takes the special form of Equation (A.3):

\[
f_{k+1|k}(x|Z^k) = \int f_{k+1}(x|x')f_{k+1|k}(x'|Z^k)dx
\]  

(5.5)

- **Corrector:** The corrector updates the state based on a new measurement. In other words, it fuses previous information \( Z^k \) with new information \( \zeta_{k+1} \). The general form of the corrector equation is:

\[
f_{k+1|k+1}(\xi|Z^{k+1}) \propto f_{k+1}(\zeta_{k+1}|\xi)f_{k+1|k}(\xi|Z^k)
\]  

(5.6)
In the previous chapters, the Euclidean form of Equation (A.4) could be used, but when measurements are a random set form, as they are when soft sensors are involved, the more general form must be used. The crucial term in this equation is the likelihood $f_{k+1}(\zeta_{k+1}|\xi)$. It expresses the probability of receiving the measurement $\zeta_{k+1}$ given that the target has state $\xi$. Because in this thesis states are in Euclidean space, and measurements are random sets, the likelihood term takes on the following special form:

$$f_{k+1}(\zeta_{k+1}|\xi) = f_{k+1}(\Theta|x) \quad (5.7)$$

Specific likelihoods for various measurement types are given in Section C.3.

- **Fusion**: Data from multiple sensors are fused by using their joint likelihood in Bayes’ rule. If sensor one returns measurement $\zeta_{k+1}^1$ and sensor two returns measurement $\zeta_{k+1}^2$, their joint likelihood would be $\frac{1}{12} f_{k+1}(\zeta_{k+1}^1, \zeta_{k+1}^2|x)$. If the measurements are independent, then the joint likelihood reduces to the following:

$$\frac{1}{12} f_{k+1}(\zeta_{k+1}^1, \zeta_{k+1}^2|x) = \frac{1}{f_{k+1}(\zeta_{k+1}^1|x)} \cdot \frac{1}{f_{k+1}(\zeta_{k+1}^2|x)} \quad (5.8)$$

- **State estimation**: This step estimates the current state of the system using the posterior probability distribution. A suitable Bayes optimal state estimator is maximum a posteriori (MAP) estimator $\hat{\xi}_{k+1|k+1} = \arg\sup_{\xi} f_{k+1|k+1}(\xi|Z^{k+1})$. If the probability distribution is symmetric and uni-modal (such as a Gaussian distribution) then this estimate would be equivalent to the mean.

- **Error estimation**: The error estimation step calculates the statistical dispersion of $f_{k+1|k+1}(\xi|Z^{k+1})$. Typically the error will be estimated using the covariance or entropy of the probability distribution.
Fig. 5.1: Data fusion using formal Bayes modeling and the generalized Bayes filter [5]
5.3 Random set theory uncertainty representations

As described in the previous section, using random sets for fusing hard and soft information involves modeling both types of information as a random set and fusing the information using a generalized Bayes filter. This section shows the specific details of how each type of information can be expressed as a random set. It also shows the likelihood functions for each type of information.

5.3.1 Random set representation of hard sensor data

Hard sensor data is often trivial to express as a random set, because hard sensors usually return data that is already described statistically. Such statistical data is simply a random set with a single element. The following paragraphs will briefly describe how random set likelihoods are constructed for both discrete and continuous hard sensor data.

**Discrete** Hard sensor classification data usually takes on the form $z = d_i$ where $d_i \in D$ is a member of a discrete set. For instance, if the hard sensor classified the type of vehicle in an image, the discrete set could be $D = \{\text{car, truck, motorcycle}\}$. Traditionally, discrete probability distributions are given by simple enumeration, i.e. $p(c_1) = p_1$, $p(c_2) = p_2$, etc. Therefore, to construct the likelihood $p(d_j|c_i)$, i.e. the probability that the sensor classified the object as $d_j$ given that the actual class is $c_i$, a matrix such as Table 5.1 is constructed. Then the likelihood can be found in the appropriate row and column of the matrix. Because this likelihood is already probabilistic, it does not need to be transformed in order to be used in the generalized Bayes filter.

<table>
<thead>
<tr>
<th>$d_1$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(d_1</td>
<td>c_1)$</td>
<td>$p(d_1</td>
<td>c_2)$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$p(d_2</td>
<td>c_1)$</td>
<td>$p(d_2</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

Table 5.1: Example discrete hard sensor likelihood table
Continuous  Hard sensor measurements of quantities such as distance, angle, and speed are all continuous in nature. Pixel location data from an image is also treated as continuous, because even though pixels are discrete, they are small enough to be treated as if they were continuous. In general, a continuous hard sensor measurement takes on the form \( z = \rho(\eta(x), w) \). In other words, the measurement is a function of a nonlinear transformation of the state and a noise term \( w \). In most cases dealt with in this thesis, the measurements noise is additive, i.e. \( z = \eta(x) + w \). The likelihoods of continuous hard sensor measurements take on the form \( f(z|x) \), the same form as in Equation (A.4).

5.3.2 Random set representation of soft sensor data

Most of the soft sensor data dealt with in this thesis is fuzzy in nature. For instance, a soft sensor may give a measurement such as “the target is near me.” The concept of “near” can be translated into a fuzzy membership function \( \mu_{\text{near}}(z) \), which assigns to every distance \( z \) a confidence between zero and one that its “near.” In Appendix C.2.1 the *synchronous random set representation* [5] of a fuzzy set \( \mu \) is defined as

\[
\Sigma_A(\mu) \triangleq \{ u | A \leq \mu(u) \},
\]

(5.9)

where \( A \) is a uniformly distributed random number on \([0, 1]\). This random set contains the same information as the original fuzzy set, but it has an added random component.

In Equation (5.3) the generalized likelihood of a random set measurement was defined as

\[
f(\Theta|x) \triangleq \Pr(\eta(x) \in \Theta).
\]

(5.10)

Based on this definition, it is possible to show [5] that when \( \Theta \) is a fuzzy synchronous random set

\[
f(\Theta_\mu|x) = \Pr(\eta(x) \in \Sigma_A(\mu)) = \mu(\eta(x)),
\]

(5.11)

where \( \eta(\cdot) \) is the function that maps states to measurements. For instance, for our definition of “near” \( \eta(x) = d(x) \), where \( d(x) \) is the distance from point \( x \) to the observer.
Therefore the likelihood of the measurement “near” would be given by

\[ f(\Theta_{\text{near}}|x) = \mu_{\text{near}}(d(x)). \]  
(5.12)

**Discrete**  
A discrete fuzzy set is created by enumerating for every possible value of \( d \) the value of \( \mu(d) \). For instance, returning to the vehicle classification example, \( d \) could be a member of the set \( D = \{\text{car, truck, motorcycle}\} \). Then for each vehicle type, the soft sensor would give a confidence between zero and one that the object being observed is that type of vehicle. Alternatively, the observer may assert a single hypothesis and a confidence in that hypothesis, for instance “I am moderately confident that the item is a car.” Then, a moderate value (for instance, 0.7) would be assigned to \( \mu(\text{car}) \) and a lower value (for instance, 0.15) would be assigned to \( \mu(\text{truck}) \) and \( \mu(\text{motorcycle}) \). In the case of this classification problem, the state is simply the actual class of the item; therefore, the function that maps states to measurements is \( \eta(c) = c \). So, in this case the likelihood of a state \( c \) is given by

\[ f(\Theta_{\text{car}}|c) = \mu_{\text{car}}(c). \]  
(5.13)

**Continuous**  
Continuous fuzzy sets are given by a function that maps values on the domain of interest to the interval \([0, 1]\). For instance, the notion of “medium speed” could be given by a triangular membership function such as the one in Figure 5.2. In this case, if the function \( s(x) \) gives the speed of a target with state \( x \), the likelihood of measurement “medium speed” given state \( x \) is

\[ f(\Theta_{\text{medium speed}}|x) = \mu_{\text{medium speed}}(s(x)), \]  
(5.14)

where \( \mu_{\text{medium speed}}(\cdot) \) is given by the triangular membership function.

### 5.4 Processing random set data using particle filters

With these mathematical preliminaries in place, it is now possible to provide some concrete algorithms that can be used to process data from both hard and soft sensors.
The first subsection introduces the generalized particle filter, which is a specialized particle filter for processing random set measurements. The next three subsections deal with the hybrid particle filter, the regularized particle filter, and the distributed particle filter. All three methods were originally designed for processing traditional hard sensor data, but can be generalized for processing soft sensor data. The final subsection presents a truly novel particle filtering algorithm, the distributed, hybrid, generalized particle filter (DHGPF). It draws on elements of the other four filters in order fuse data from both hard and soft sensors that are distributed over a network in order to track a target that can have both continuous and discrete states.

5.4.1 The generalized particle filter

The soft sensor measurements described in the previous section are “inherently very nonlinear and non-Gaussian” (Mahler page 199 [5]). If the soft sensor measurements were to take on the form of fuzzy Dempster-Shafer measurements with Gaussian focal sets, then a specialized Gaussian-mixture filter known as the Kalman evidential filter [5, 203, 204] could be used instead of a traditional Kalman filter. Unfortunately,
the measurements in the previous section do not take on this special form; therefore, a particle approximation to the generalized Bayes filter must be used.

In order to generalize the standard particle filter algorithm (Algorithm A.5 in Appendix A) to accommodate random set measurements, the likelihood step must be modified. In the traditional particle filter, each particle is weighted proportional to the measurement likelihood \( f(z|x) \). In the random set case, measurements are weighted instead by the generalized likelihood \( f(\Theta|x) \). This is the only major difference between the traditional particle filter and the generalized particle filter.

Algorithm 5.1 gives the complete generalized particle filter algorithm. The most significant difference between the generalized particle filter and Algorithm A.5 is that in Step 6 the likelihood is now a generalized likelihood. Another change is in Step 11, where the algorithm calculates the effective number of particles \( N_{eff} \) (essentially, the number of particles that have a significant weight) using Equation (5.18). It then only resamples if the effective number of particles falls below a certain threshold. This change is not a feature of the generalized particle filter, it can be done with any particle filter to help prevent sample impoverishment and to save computations by not resampling at every step [7]. Most of the particle filters in this section will use this resampling technique.

5.4.2 The hybrid particle filter

The hybrid particle filter is used in situations where the target state has both continuous and discrete components. It was originally designed for traditional sensor types, but it is also useful for hard and soft sensors. There are several reasons for having both continuous and discrete states. The discrete states could determine what dynamic model governs target dynamics (straight line, accelerating, turning, etc.); in this case, the filter is also referred to as the multiple-model particle filter [7, 159]. The multiple-model particle filter is a generalization of the interacting multiple model (IMM) filter [63, 135, 136]. The additional discrete states can also correspond to non-kinematic attributes of the target such as its class (airliner or fighter jet) or allegiance (friend or foe). In the most general case, the discrete states could represent both types of attributes. For
Algorithm 5.1 The generalized particle filter

1: Randomly generate $N$ particles by sampling from the pdf of the initial state, $f(x_0)$. These particles are denoted by $\{x_0^n\}$.
2: Set the weight of each particle to $w_0^i = \frac{1}{N}$.
3: for $k = 1, 2, \ldots$ do
4: for $i = 1$ to $N$ do
5: Draw a new particle according to the state update probability density function:

$$x_k^i \sim f(x_k^i|x_{k-1}^i)$$ (5.15)

6: Update the weight of the particle using the measurement likelihood:

$$\tilde{w}_k^i = w_{k-1}^i f(\Theta_k|x_k^i)$$ (5.16)

7: end for
8: for $i = 1$ to $N$ do
9: Scale the relative weights so that they sum to one as follows:

$$w_k^i = \frac{\tilde{w}_k^i}{\sum_{j=1}^{N} \tilde{w}_k^j}$$ (5.17)

10: end for
11: Calculate the effective number of particles $N_{\text{eff}}$ as follows:

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (w_k^i)^2}$$ (5.18)

12: if $N_{\text{eff}} < N_{\text{thr}}$ then
13: Resample using Algorithm A.6:

$$\{x_k^n, w_k^n\} = \text{RESAMPLE}(\{x_k^n, w_k^n\})$$ (5.19)
14: end if
15: end for
instance, in Boers and Driessen [205] a target could be either a fighter jet or an airliner, and the possible dynamic regimes of the target depend on its classification.

Algorithm 5.2 shows the hybrid particle filter algorithm. In the first step of the algorithm, the particles are randomly assigned an initial state and an initial regime \( r \). The initial regime of the particle is given by a discrete probability distribution \( f(r_0) \).

The next step of the filter conducts a regime transition using Algorithm 5.3. The regime transition accounts for the fact that targets can transition between modes (for instance, a target which is traveling in a straight line can start turning). The regime transition algorithm randomly assigns to each particle a new regime according to the transition probability. The transition probability is given by the matrix \( \Pi = [\pi_{ij}] \), where \( \pi_{ij} \) represents the probability that a target which is in mode \( r_{k-1} = i \) at time \( k - 1 \) transitions to mode \( r_k = j \) at time \( k \). The rows of \( \Pi \) must sum to one for the elements to be valid probabilities. In the special case that the regime simply represents the class of a target, the transition matrix would be an identity matrix since targets cannot change class.

Once the new regimes are selected, particles are updated using the regime conditioned state update (Equation (5.21)), which probabilistically assigns a new state to each particle based on its previous state and the regime. Because the state update is conditioned on the regime, it can use different dynamic models depending on which regime the particle is in.

The measurement update (Equation (5.22)) is also conditioned on the regime. This allows measurements of not only the target’s state, but also its identity to be incorporated into the likelihood. For instance, a radar could measure range, bearing, and radar cross section; then, a large radar cross section measurement is more likely due to an airliner than a fighter jet [205]. After performing the state and measurement updates, the hybrid particle filter checks the effective number of particles and resamples, if necessary.
Algorithm 5.2 The hybrid particle filter [7]

1: Generate \( N \) particles \( y_k = [x_k^T r_k]^T \) from the known pdf’s \( f(x_0|r_0) \) and \( f(r_0) \).
2: for \( k = 1, 2, \ldots \) do
3: Perform regime transition on the particles (Algorithm 5.3):

\[
\{r_k^n\} = \text{RT}(\{r_{k-1}^n\}) \quad (5.20)
\]

4: for \( i = 1 \) to \( N \) do
5: Draw a new particle according to the regime conditioned state update probability density function:

\[
x_k^i \sim f(x_k|x_{k-1}^i, r_k^i) \quad (5.21)
\]

6: Update the weight of the particle using the regime conditioned measurement likelihood:

\[
\tilde{w}_k^i = w_{k-1}^i f(z_k|x_k^i, r_k^i) \quad (5.22)
\]

7: end for
8: for \( i = 1 \) to \( N \) do
9: Scale the weights so that they sum to one as follows:

\[
w_k^i = \frac{\tilde{w}_k^i}{\sum_{j=1}^{N} \tilde{w}_k^j} \quad (5.23)
\]

10: end for
11: Calculate \( N_{eff} \) using Equation (5.18).
12: if \( N_{eff} < N_{thr} \) then
13: Resample using Algorithm A.6:

\[
\{y_k^n, w_k^n\} = \text{RESAMPLE}(\{y_k^n, w_k^n\}) \quad (5.24)
\]
14: end if
15: end for
Algorithm 5.3 Regime transition algorithm [7]

1: for $i = 1$ to $s$ do
2: $c_i(0) = 0$
3: for $j = 1$ to $s$ do
4: $c_i(j) = c_i(j - 1) + \pi_{ij}$
5: end for
6: end for
7: for $n = 1$ to $N$ do
8: Draw a random number $u_n \sim U[0, 1]$
9: Set $i = r_{r_{k-1}}^n$
10: $m = 1$
11: while $c_i(m) < u_n$ do
12: $m = m + 1$
13: end while
14: Set $r_k^n = m$
15: end for

5.4.3 The regularized particle filter

A major issue with particle filters is “sample impoverishment,” where resampling results in one or only a few particles. This is not unique to particle filters that process soft data. Any particle filter can suffer from this problem. One solution to the sample impoverishment problem is to resample from a continuous distribution, rather than a discrete distribution. This method is refereed to as the regularized particle filter [7,159]. Sampling from a continuous distribution rather than a discrete distribution is also useful for distributing a particle filter across multiple sensing platforms (see Section 5.4.4).

The key to the regularized particle filter is the process of approximating the discrete particle set as a continuous probability density function. The most common choice is to use a kernel approximation [7]:

$$p(x_k|Z_k) \approx \sum_{i=1}^{N} w_k^i K_h(x_k - x_k^i),$$  \hspace{1cm} (5.25)

where $\{x_k^i, w_k^i\}$ is the set of particles and weights and the rescaled kernel density $K_h(\cdot)$ is given by

$$K_h(x) = \frac{1}{h^{n_x}} K\left(\frac{x}{h}\right),$$  \hspace{1cm} (5.26)
where \( n_x \) is the dimension of the state vector. The Kernel density \( K(\cdot) \) is a symmetric probability density function with the following properties:

\[
\int xK(x)dx = 0 \tag{5.27a}
\]

\[
\int \|x\|K(x)dx < \infty \tag{5.27b}
\]

The kernel \( K(\cdot) \) and bandwidth \( h \) should be chosen to minimize the mean integrated square error between the true distribution and the kernel representation. Unfortunately, the underlying distribution is not known, this is the whole reason for the particle approximation. In the special case of equally weighted samples, the optimal choice is the Epanechnikov kernel [159], but for computational reasons another common kernel is the Gaussian kernel (i.e. a zero mean Gaussian probability density with unit covariance). If the Gaussian kernel is used, and the underlying probability density is itself Gaussian with a unit covariance matrix, the optimal bandwidth is [159]

\[
h_{opt} = AN^{-\frac{1}{n_x+4}} \tag{5.28}
\]

where

\[
A = \left( \frac{4}{n_x + 2} \right)^{\frac{1}{n_x+4}}, \tag{5.29}
\]

and \( N \) is the number of particles.

In practice, none of the assumptions underlying these optimal choices hold (if they did, there would be no need for using a particle filter); however, in the absence of any other information about the posterior, they are still used, resulting in a suboptimal filter [7]. Algorithm 5.4 gives the full regularized particle filter algorithm. Note that it is identical to the standard particle filter, except in the resampling step. In Step 13 an empirical covariance matrix is calculated based on both the particles and their weights. Then, in Step 18 a random variable drawn from the Gaussian kernel is multiplied by the bandwidth and the square root of the covariance matrix, then added to the resampled particle. This effectively jitters the resampled value, so that there are not several copies of the exact same particle. The reason for multiplying by the square root of the covariance matrix
matrix is that the optimal bandwidth is calculated by assuming that the underlying probability distribution has unit covariance matrix. Therefore, it must be scaled for other covariance matrices.

5.4.4 The distributed particle filter

There have been several efforts to distribute a particle filter across multiple platforms [6, 206–210]. Many approaches have been based on communicating measurements between sensing nodes. In the special case that the sensors collect independent measurements and communicate their measurements at every time step, distributed fusion would be identical to the centralized case. A more efficient approach is to send a quantized version of the measurement likelihood rather than the full measurement; this approach only works, however, if both filters are initialized with the same particles [207]. Rather than broadcasting the quantized likelihood to all partners, it is also possible to only send it to partners in the same clique [208]. Another similar method is to collect several measurements at the local sensing node, then send them all at once [209]. Doing so can save on communications overhead, but it requires some more sophisticated processing. All of these algorithms have one thing in common: every measurement is sent to every other partner (or every partner in a clique). A more principled method for fusion based on measurements is given in Rosencrantz et al., where sensors query each other for information only when they need it [206]. The query consists of one robot sending its current belief (expressed as a random subset of its particles) to another robot. The second robot then replies with its most informative recent measurement (where information is measured using the Kullback-Leibler divergence between the measurement and the belief sent by the first robot).

All of these measurement fusion based approaches require that measurements be independent. They also require that each node have some notion of the network architecture: in the cases where quantized likelihoods are sent a broadcast architecture is required; in the case of the query-response system the querying robot must know which partners are likely to have the needed information. Recall from Section 3.3.5 that in situations where nodes have no knowledge of network topology and independence of
Algorithm 5.4 Regularized particle filter algorithm [7]

1: Randomly generate $N$ particles by sampling from the pdf of the initial state, $f(x_0)$. These particles are denoted by $\{x_0^n\}$.

2: Set the weight of each particle to $w_i^0 = \frac{1}{N}$.

3: for $k = 1, 2, \ldots$ do

4: for $i = 1$ to $N$ do

5: Draw a new particle according to the state update probability density function:

$$x_k^i \sim f(x_k|x_{k-1}^i)$$ (5.30)

6: Update the weight of the particle using the measurement likelihood:

$$\tilde{w}_k^i = w_{k-1}^i f(z_k|x_k^i)$$ (5.31)

7: end for

8: for $i = 1$ to $N$ do

9: Scale the relative weights so that they sum to one as follows:

$$w_k^i = \frac{\tilde{w}_k^i}{N \sum_{j=1}^N \tilde{w}_k^j}$$ (5.32)

10: end for

11: Calculate $N_{eff}$ using Equation (5.18).

12: if $N_{eff} < N_{thr}$ then

13: Calculate the weighted empirical covariance matrix $S_k$ of $\{x_k^n,w_k^n\}$.

14: Compute the Cholesky factorization of $D_k$ of $S_k$, i.e. find $D_k$ such that $D_k D_k^T = S_k$.

15: Resample using Algorithm A.6:

$$\{x_k^n,w_k^n\} = \text{RESAMPLE}(\{x_k^n,w_k^n\})$$ (5.33)

16: for $i = 1$ to $N$ do

17: Draw $\epsilon^i \sim K$ from the Gaussian kernel.

18: Create a new particle $x_k^{i*} = x_k^i + h_{opt} D_k \epsilon^i$.

19: end for

20: end if

21: end for
measurements and state estimates cannot be guaranteed, covariance intersection is often a suitable candidate fusion algorithm. Covariance intersection was originally designed to fuse two Gaussian probability distributions when the correlation between them is unknown [3,82]. There is no guarantee that the fused estimate is optimal, but it will always be consistent (in other words it will not double count correlated information). It is also possible to generalize covariance intersection to non-Gaussian probability distributions using information theory [211]. It is even possible to use a variation of covariance intersection when the states are random finite sets, instead of random vectors [212]. These more general techniques work in theory for any probability distributions, but they require that the probability distributions be known, and there is no guarantee that there is a closed form solution to the fusion equations. A particle filter would violate the assumption that the probability distribution be known, so a different technique must be used to fuse state information from two particle filters using covariance intersection.

This thesis employs the technique developed by Ong et al. for fusing particle sets [6]. In this method, the particle set is first approximated by a Gaussian mixture, then the Gaussian mixtures are fused by covariance intersection. It would be possible to generate a Gaussian mixture approximation to the particle set through expectation maximization [71], but it is an iterative approach which is not necessarily efficient enough to run on a mobile robot. The regularized particle filter (Algorithm 5.4) provides a means for approximating a set of particles as a continuous probability distribution using a kernel function [159]. If a Gaussian kernel is used, then the resulting distribution would be a Gaussian mixture. Unfortunately, the resulting Gaussian mixture would have the same number of components as there are particles, which would be too inefficient to communicate over a network.

In order to reduce the number of mixture components that must be communicated, Salmond’s joining algorithm can be used [8]. This algorithm reduces the number of components in a mixture by first eliminating insignificant components, then fusing components that are “near” each other (in terms of the Mahalanobis distance). The fusion is carried out in such a way to keep the overall covariance of the entire mixture unchanged. Algorithm 5.5 gives Salmond’s joining algorithm. The algorithm has three
tunable parameters: $\pi_T$, $N_T$, and $d_T$. The parameter $\pi_T$ is the minimum weight that a component must have in order to be considered for fusion; it ensures that computation is not wasted on insignificant components. The maximum number of allowable components is given by $N_T$, in this thesis it has been set to the square root of the original number of components. The parameter $d_T$ is the minimum distance between components; it ensures that components that are closer than this distance are fused even if $N_T$ has already been reached. Note that the algorithm must compute the distance between all components, but this must only be done once. Then, when a new mixture is formed, only the row and column of the distance matrix corresponding to the new component must be recalculated.

Once the number of components in the mixture has been reduced, it is communicated to the other sensing nodes. Then, the mixtures are fused using a generalization of covariance intersection [9]. Algorithm 5.6 gives the method for performing the fusion. Essentially, the algorithm fuses every Gaussian in the first mixture with every other Gaussian in the second mixture. If there are $N$ Gaussians in the first mixture and $M$ in the second, this will result in $N \cdot M$ new Gaussians. This growth in the number of components is not a major concern, because ultimately a set of particles will be drawn from the new mixture. Note that weight $\omega_{ij}$ can be computed using any method, but in this thesis it has been computed by improved fast covariance intersection [38]. Also note that at each iteration several matrix inversions must be performed and several determinants must be calculated. To save on computation, the inverse of each covariance matrix and its determinant can be calculated and stored, so that only the determinant of their sum and the inverse of the new covariance matrix must be computed each time.

With these preliminaries in place, the complete algorithm for distributed fusion using a particle filter can be expressed [6, 210]. Figure 5.3 shows a flowchart of this algorithm. The block entitled “Time to communicate?” controls how often the filter sends data to partners. The communication can occur as a result to a query, at fixed time-steps, or after so many measurements are received. In this thesis, the filter always broadcasts its state to all partners, but this is not a limitation of the algorithm; the state can be sent to any subset of partners, who could then pass it on to their partners, without
keeping track of the information’s pedigree. Also note that in this thesis, resampling is carried out by regularized resampling. This is used because the regularization step already needed to be implemented in order to perform the communication. It is not clear what resampling method was used in the original paper [6].

5.4.5 The distributed, hybrid, generalized particle filter

With all of these preliminaries in place, this section presents a novel method for fusing hard and soft data over a distributed network: the distributed, hybrid, generalized particle filter (DHGPF). The DHGPF is useful in situations where targets can belong to more than one class or regime, when sensing nodes are distributed over a network, and when there is a mixture of traditional hard sensors and soft human sensors reporting on the target; these are precisely the situations that will be dealt with in Chapters 6 and 7.

Figure 5.4 shows a flowchart of the DHGPF algorithm. It appears to be similar to the distributed particle filter, but there are several important distinctions. The first thing to note is that rather than just initializing the kinematic states of particles, they are also assigned a random mode, just as they are in the hybrid particle filter (Algorithm 5.2).

The second difference comes from how particle states, modes, and weights are updated. Algorithm 5.7 shows how this update is performed: first there is a probabilistic regime transition and regime conditioned state update, just as in the hybrid particle filter algorithm (Algorithm 5.2); but then, rather than updating the particle weights using the traditional regime conditioned likelihood \( f(z_k | x_k^i, r_k^i) \), they are updated using the generalized regime conditioned likelihood \( f(\Theta_k | x_k^i, r_k^i) \). This likelihood function is quite powerful, because the term \( \Theta_k \) could represent traditional hard sensor data or fuzzy data from a soft sensor. Also, because the likelihood is conditioned on the kinematic state and the regime, sensors can report on not only the sensor’s physical attributes, but also its class, allegiance, and motion characteristics (such as smooth or erratic).

Because the DHGPF deals with both continuous and discrete states, the process for performing regularized resampling is more complicated than in the original case. The state of a particle is given as \( y_k = [x_k^T r_k]^T \), where \( x_k \) is continuous and \( r_k \) is discrete. The regularized particle filter seeks to create a continuous approximation to \( f(y_k | z_k) \), but
Algorithm 5.5 Salmond’s joining algorithm [8]

1: Eliminate all mixture components with weight $\pi_i < \pi_T$.
2: Find the covariance matrix $\Sigma$ of the entire mixture

$$
\Sigma = \sum_{i=1}^{N} \pi_i \left( \Sigma_i + (\mu_i - \hat{x}) (\mu_i - \hat{x})^T \right),
$$

(5.34)

where

$$
\hat{x} = \sum_{i=1}^{N} \pi_i \mu_i
$$

(5.35)

is the mean of the mixture.

3: for $i = 1$ to $N$ do
4: for $j = i + 1$ to $N$ do
5: Calculate the distance $d_{ij}$ between components $i$ and $j$ as follows

$$
\frac{d_{ij}^2}{\pi_i \pi_j} = \frac{\pi_i \pi_j}{\pi_i + \pi_j} (\mu_i - \mu_j)^T \Sigma^{-1} (\mu_i - \mu_j)
$$

(5.36)

6: end for
7: end for
8: Store the distances in an upper triangular matrix.
9: while $N > N_T$ or $\min(d_{ij}) < d_T$ do
10: Find the minimum distance $d_{ij}$.
11: Calculate the new weight $\pi'$, mean $\mu'$, and covariance $\Sigma'$ as follows:

$$
\pi' = \pi_i + \pi_j
$$

(5.37a)

$$
\mu' = \frac{1}{\pi} (\pi_i \mu_i + \pi_j \mu_j)
$$

(5.37b)

$$
\Sigma' = \frac{1}{\pi} \left( \left( \Sigma_i + \mu_i \mu_i^T \right) + \left( \Sigma_j + \mu_j \mu_j^T \right) \right) - \mu' \mu'^T
$$

(5.37c)

12: Eliminate the rows and columns of the distance matrix corresponding to components $i$ and $j$.
13: Set $N = N - 1$.
14: Calculate the distance between the new component and every other component using Equation (5.36).
15: end while
Initialize particle set

Send mixture data to partners

Partner data available?

yes

Generate local mixtures using Salmond’s joining algorithm (Algorithm 5.5)

Update particle states and weights (Equations (5.30), (5.31), and (6.20))

Generate local mixtures using Salmond’s joining algorithm (Algorithm 5.5)

Fuse local data with partner data using Gaussian mixture covariance intersection (Algorithm 5.6)

Sample $N$ new particles from the fused mixture

Time to communicate?

yes

Calculate $N_{eff}$ (Equation (5.18))

$N_{eff} < N_{thr}$?

no

no

yes

no

Perform regularized resampling (Algorithm 5.4)

Fig. 5.3: Distributed particle filter algorithm [6]
Algorithm 5.6 Algorithm for the fusion of Gaussian mixtures by covariance intersection [9]

1: for \( i = 1 \) to \( N \) do
2: \quad for \( j = 1 \) to \( M \) do
3: \quad \quad Use fast covariance intersection [38] to calculate the weight \( \omega_{ij} \):
4: \quad \quad \omega_{ij} = \frac{\det(\Sigma_i^{-1} + \Sigma_j^{-1}) - \det(\Sigma_j^{-1}) + \det(\Sigma_i^{-1})}{2 \det(\Sigma_i^{-1} + \Sigma_j^{-1})} \quad (5.38)
5: \quad \quad Generate the parameters \( \Sigma_{ij}, \mu_{ij}, \) and \( \pi_{ij} \) for the new mixture:
6: \quad \quad \quad \Sigma_{ij}^{-1} = \omega_{ij} \Sigma_i^{-1} + (1 - \omega_{ij}) \Sigma_j^{-1} \quad (5.39a)
7: \quad \quad \quad \mu_{ij} = \Sigma_{ij} \left( \omega_{ij} \Sigma_i^{-1} \mu_i + (1 - \omega_{ij}) \Sigma_j^{-1} \mu_j \right) \quad (5.39b)
8: \quad \quad \quad \pi_{ij} = \omega_{ij} \pi_i + (1 - \omega_{ij}) \pi_j \quad (5.39c)
9: \quad end for
10: end for

it makes no sense to create a continuous approximation to this probability distribution when it has both continuous and discrete states. Another possible approach would be to assume that \( x_k \) and \( r_k \) are conditionally independent given \( Z^k \); then the posterior could be factored as follows:

\[
f(y_k|Z^k) = f(x_k, r_k|Z^k) = f(x_k|Z^k) \cdot f(r_k|Z^k)
\] (5.40)

There are several problems with this factorization. First, it is not clear that it is valid, because the state could depend heavily on the mode. If the mode represents vehicle class, for instance, then items of class car would be constrained to be on the ground, but items of class airplane would not. Furthermore, particles with different motion models could have a different number of states. (For instance, in two dimensions it takes four states to represent a constant velocity target, five to represent a target with constant turn rate, and six to represent a constant acceleration target [161].) If there were a different number of states in each mode, it would make no sense to create a continuous approximation to \( f(x_k|Z^k) \), because the dimension of \( x_k \) would be undefined. Because of these issues, it is best not to assume that the kinematic state and regime are conditionally independent.
This results in the following factorization:

\[ f(y_k|Z^k) = f(x_k, r_k|Z^k) = f(x_k|r_k, Z^k) \cdot f(r_k|Z^k) \] (5.41)

From this factorization, it is now clear how to perform regularized resampling. First, the probability distribution \( f(r_k|Z^k) \) must be learned from the particle set. This step consists of adding up the weights of all particles in a given mode. Then, a continuous approximation to \( f(x_k|r_k, Z^k) \) must be learned. To do this, a continuous kernel distribution is created using only the particles in mode \( r^i_k \) (with the weights normalized so that the sum of the weights in each mode is one). To sample from this new distribution, a new mode \( r^i_k \) is drawn according to \( f(r_k) \), which is proportional to the sum of the weights of the particles in mode \( r^i_k \). Then, a particle is randomly drawn from the set of particles that are in mode \( r^i_k \). Finally, the particle is jittered by a random amount based on the kernel for mode \( r^i_k \). Algorithm 5.8 gives the complete algorithm for performing this operation.

Similar to the previous discussion for regularization, extra care must be taken in the fusion step to account for the fact that states may be both continuous and discrete. In particular, the posterior is once again factored as \( f(y_k|Z^k) = f(x_k|r_k, Z^k) \cdot f(r_k|Z^k) \), then the discrete and continuous states are handled separately. Algorithm 5.9 shows how this is done: the algorithm first partitions the particle set by mode, then it finds the sum of the weights for each mode, and finally generates a Gaussian mixture for each mode. This means that the filter must send several pieces of data to partners: a vector of total weights assigned to each mode and a Gaussian mixture corresponding to each mode. For a problem with many modes, this may result in too much communication overhead; if this is the case it may be necessary to assume that the kinematic state of the particle and its mode are independent, with an obvious loss in accuracy.

Algorithm 5.10 shows the method for fusing two mode conditioned Gaussian mixtures. First, note that the modes are fused using a minimum. The reason for choosing minimum is that it corresponds to fusing two perfectly correlating pieces of information using a copula conjunction [5]. (For more on copula conjunctions see Appendix C.4.2.)
Because no assumptions of independence are being made in this situation, the minimum
is a conservative choice for fusing discrete data, just as covariance intersection is for
continuous data. Next, the mode weights are normalized so that they continue to sum
to one. Finally, the Gaussian mixtures corresponding to each mode are fused by gener-
alized covariance intersection (Algorithm 5.6), which results in a new set of mixtures for
each mode. Algorithm 5.11 shows how new samples are drawn from the fused distribu-
tion. Similar to in Algorithm 5.8, a mode is drawn at random according the the fused
mode probability density; then, a state is randomly drawn from the Gaussian mixture
 corresponing to the given mode; finally, the weight of every particle is set to \( \frac{1}{N} \).

**Algorithm 5.7** Algorithm to update particles in the generalized, hybrid particle filter

1: Perform regime transition on the particles (Algorithm 5.3):
   \[
   \{r^n_k\} = RT(\{r^n_{k-1}\}) \tag{5.42}
   \]

2: for \( i = 1 \) to \( N \) do
3:     Draw a new particle according to the regime conditioned state update probability
density function:
   \[
   x^i_k \sim f(x_k|x^i_{k-1}, r^i_k) \tag{5.43}
   \]
4:     Update the weight of the particle using the regime conditioned generalized mea-
   surement likelihood:
   \[
   \tilde{w}^i_k = w^i_{k-1} f(\Theta_k|x^i_k, r^i_k) \tag{5.44}
   \]
5: end for
6: for \( i = 1 \) to \( N \) do
7:     Scale the weights so that they sum to one as follows:
   \[
   w^i_k = \frac{\tilde{w}^i_k}{\sum_{j=1}^{N} \tilde{w}^j_k} \tag{5.45}
   \]
8: end for
Initialize particle kinematic states and modes as in Algorithm 5.2

Send mixture data to partners

Partner data available?

yes

no

Generate mode conditioned local mixtures using Algorithm 5.9

Time to communicate?

yes

no

Update particle states, modes, and weights using Algorithm 5.7

Generate mode conditioned local mixtures using Algorithm 5.9

Fuse local data with partner data using mode conditioned Gaussian mixture covariance intersection (Algorithm 5.10)

Sample $N$ new particles from the fused mixture using Algorithm 5.11

Calculate $N_{eff}$ (Equation (5.18))

$N_{eff} < N_{thr}$?

no

yes

Perform mode conditioned regularized resampling (Algorithm 5.8)

Fig. 5.4: Distributed, hybrid, generalized particle filter (DHGPF) algorithm
Algorithm 5.8 Mode conditioned regularized resampling algorithm

1: Partition the particles and weights into \( s \) sets, where the set \( \{x^j_i, w^j_i\} \) consists of all particles in mode \( i \).
2: for \( i = 1 \) to \( s \) do
3: Find the total weight \( w^i_{\text{total}} \) of all the particles in mode \( i \)
\[
w^i_{\text{total}} = \sum_{j=1}^{N_i} w^j_i,
\]
where \( N_i \) is the number of particles in mode \( i \).
4: Normalize the weights of all particles in mode \( i \) so that they sum to one.
5: Calculate the weighted empirical covariance matrix \( S^i \) of the particles \( \{x^n, w^n\}^i \) in mode \( i \).
6: Compute the Cholesky factorization of \( D^i \) of \( S^i \), i.e. find \( D^i \) such that \( D^i D^{iT} = S^i \).
7: end for
8: for \( i = 1 \) to \( N \) do
9: Randomly choose the mode \( r^i \) of particle \( i \) according to the probability density
\[
f(r) = \sum_{j=1}^{s} w^j_i \delta(r - r^j),
\]
10: Randomly draw a particle \( x^i \) from among the particles in mode \( r^i \), i.e. \( \{x^j_i, w^j_i\}^{r^i} \), in proportion to the normalized particle weights.
11: Draw \( \epsilon^i \sim K^{r^i} \) from the Gaussian kernel corresponding to mode \( r^i \). (The kernels are identical unless different modes have a different number of states.)
12: Create a new particle \( x^{i*} = x^i + h^{opt} D^{r^i} \epsilon^i \).
13: Set the weight of the new particle to \( \frac{1}{N} \).
14: end for

Algorithm 5.9 Algorithm to generate mode conditioned Gaussian mixtures

1: Partition the particles and weights into \( s \) sets, where the set \( \{x^j_i, w^j_i\} \) consists of all particles in mode \( i \).
2: for \( i = 1 \) to \( s \) do
3: Find the total weight \( w^i_{\text{total}} \) of all the particles in mode \( i \)
\[
 w^i_{\text{total}} = \sum_{j=1}^{N_i} w^j_i,
\]
where \( N_i \) is the number of particles in mode \( i \).
4: Express the particles in mode \( i \) as a Gaussian mixture using regularization and Salmond’s joining algorithm (Algorithm 5.5).
5: end for
Algorithm 5.10 Algorithm to fuse mode conditioned Gaussian mixtures

1: for $i = 1$ to $s$ do
2:  Find the fused mode weights using minimum:

$$\tilde{w}_i^{\text{total}} = \min(w_i^{\text{total1}}, w_i^{\text{total2}})$$  \hspace{1cm} (5.48)

3: end for
4: for $i = 1$ to $s$ do
5:  Scale the total weights so that they sum to one as follows:

$$w_i^{\text{total}} = \frac{\tilde{w}_i^{\text{total}}}{\sum_{j=1}^{s} \tilde{w}_j^{\text{total}}}$$  \hspace{1cm} (5.49)

6:  Fuse the mixtures in mode $i$ using generalized covariance intersection (Algorithm 5.6).
7: end for

Algorithm 5.11 Algorithm to sample from mode conditioned Gaussian mixtures

1: for $i = 1$ to $N$ do
2:  Randomly choose the mode of particle $i$, $r^i$ in proportion to the probability density

$$f(r) = \sum_{j=1}^{s} w_j^{\text{total}} \delta(r - r^j).$$

3:  Randomly generate the continuous state vector $x^i$ of the particle by sampling from
the Gaussian mixture corresponding to mode $r^i$.
4:  Set the weight of the particle to $w^i = \frac{1}{N}$.
5: end for
Chapter 6

Hard and soft sensor fusion simulations

In this chapter the algorithms for hard and soft sensor data fusion developed in Chapter 5 are applied to some realistic problems in simulation. The simulations are meant to mimic various aspects of the search and rescue problem described in Section 1.3. The first simulation utilizes a single hard sensor and a single soft sensor to track a target. The novel aspect of this tracking problem is that the target can move either normally or “erratically,” which is a vague term that has meaning to a person, but not to a traditional hard sensor; therefore, the challenge of the first simulation is to fuse this vague information with the measurements from the hard sensor. In the second simulation, the same sensors track a target that can now belong to one of several classes. It is assumed that the hard sensor does not provide any classification information; therefore, the soft sensor must provide this information. The final simulation tracks a target using a swarm of sensors made up of unmanned aerial vehicles, unmanned ground vehicles, and soft sensors. Sensing and fusion is distributed across the entire swarm of vehicles using the distributed, hybrid, generalized particle filter (DHGPF), which was introduced in Section 5.4.5.

6.1 Simulation one: Single hard sensor, single soft sensor, single target class

In the first simulation, a radar and a soft sensor, both located at the same place, seek to track a single car. The car could follow one of three trajectories: normal (Figure 6.1), maneuver (Figure 6.2), or erratic (Figure 6.3). This simulation is not directly a search and rescue problem, but it demonstrates a very important capability necessary for the search and rescue problem: fusion of hard sensor data and soft sensor data in a very simple scenario. The radar sensor provides accurate estimates of the target’s location
and velocity, while the soft sensor provides fuzzy estimates of the target’s location and velocity, while also classifying the target as driving normally, turning, or driving erratically. Because the soft sensor is able to provide this higher level information, better tracking results are possible than with a hard sensor alone.

Fig. 6.1: Normal vehicle trajectory, with an ‘x’ at the sensor location

6.1.1 Target motion model

There are two basic motion models that govern the car’s motion: constant velocity and turning. The constant velocity model is governed by the following linear equation of motion:

\[
\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} \frac{T^2}{2} & 0 \\ 0 & \frac{T^2}{2} \\ T & 0 \\ 0 & T \end{bmatrix} \mathbf{v}_k, \tag{6.1}
\]
Fig. 6.2: Maneuvering vehicle trajectory, with an ‘x’ at the sensor location

where $v_k$ is zero mean white noise with $2 \times 2$ covariance matrix $Q_k$, $T$ is the sampling time, and the state $x$ is given as $[x \ y \ \dot{x} \ \dot{y}]^T$. The turning model utilizes the following non-linear equation of motion [7]:

$$
\begin{bmatrix}
1 & 0 & \frac{\sin(\omega_k T)}{\omega_k} & -\frac{1-\cos(\omega_k T)}{\omega_k} & 0 \\
0 & 1 & \frac{1-\cos(\omega_k T)}{\omega_k} & \frac{\sin(\omega_k T)}{\omega_k} & 0 \\
0 & 0 & \cos(\omega_k T) & -\sin(\omega_k T) & 0 \\
0 & 0 & \sin(\omega_k T) & \cos(\omega_k T) & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_k+1 \\
v_k
\end{bmatrix}
\begin{bmatrix}
\frac{T^2}{2} & 0 & 0 \\
0 & \frac{T^2}{2} & 0 \\
0 & 0 & T \\
0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
v_k
\end{bmatrix}
$$

where the turning rate $\omega_k$ is given by

$$
\omega_k = \frac{\alpha_k}{\sqrt{\dot{x}_k^2 + \dot{y}_k^2}}
$$
(a) Erratic vehicle trajectory, with an ‘x’ at the sensor location

(b) Erratic vehicle speed in meters per second

Fig. 6.3: Erratic vehicle trajectory and speed
\( \mathbf{v}_k \) is zero mean white noise with \( 3 \times 3 \) covariance matrix \( Q_k \), and the state \( \mathbf{x} \) is given as \( [x \ y \ \dot{x} \ \dot{y} \ a]^T \). Note that this model has an additional state \( a \) to represent the lateral acceleration.

### 6.1.2 Sensor models

The hard sensor is modeled as a two dimensional radar, which reports on range, bearing, and range-rate. The nonlinear observation equation for the hard sensor is given as:

\[
\mathbf{z}_k = \eta(\mathbf{x}_k) + \mathbf{w}_k = \begin{bmatrix}
\sqrt{x_k^2 + y_k^2} \\
\arctan\left(\frac{y_k}{x_k}\right) \\
\frac{x_k\dot{x}_k + y_k\dot{y}_k}{\sqrt{x_k^2 + y_k^2}}
\end{bmatrix} + \mathbf{w}_k,
\]

where \( \mathbf{w}_k \) is zero mean white noise with \( 3 \times 3 \) covariance matrix \( R_k \), and the measurement is given as \( [\rho \ \theta \ \dot{\rho}]^T \). Assuming that the additive noise \( \mathbf{w}_k \) is Gaussian, the likelihood for this measurement is simply:

\[
f(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\mathbf{z}_k - \eta(\mathbf{x}_k), R_k),
\]

i.e. the Gaussian pdf evaluated at the difference between the actual and expected measurement.

The soft sensor reports on six attributes of the target: distance, bearing, speed, heading, turn direction, and driving style. All of these attributes are fuzzy in nature, and are described with natural language statements. Table 6.1 shows the values that each attribute can take on. Note that distance and speed can each be modified with negation (not) and the hedges very and more or less (MOL) [29]. The hedges are meant to mimic qualitative information that can come from human observers, such as "the target is not very far." The possible values for bearing and heading are the cardinal and ordinal compass directions. Turn direction simply describes the direction that the target is turning and driving style describes the driving as normal or erratic.

The key to doing any useful processing with this data is to convert it into a numeric form. This is accomplished through a fuzzy membership function. Taking distance as an
Table 6.1: Attributes reported by the soft sensor, along with possible values for each attribute

<table>
<thead>
<tr>
<th>distance</th>
<th>bearing</th>
<th>speed</th>
<th>heading</th>
<th>turn direction</th>
<th>driving style</th>
</tr>
</thead>
<tbody>
<tr>
<td>near</td>
<td>north</td>
<td>slow</td>
<td>north</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td>medium</td>
<td>northeast</td>
<td>medium</td>
<td>northeast</td>
<td>left turn</td>
<td>erratic</td>
</tr>
<tr>
<td>far</td>
<td>east</td>
<td>fast</td>
<td>east</td>
<td>right turn</td>
<td></td>
</tr>
<tr>
<td>very near</td>
<td>southeast</td>
<td>very slow</td>
<td>southeast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>very far</td>
<td>southwest</td>
<td>very fast</td>
<td>southwest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOL near</td>
<td>west</td>
<td>MOL slow</td>
<td>west</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOL far</td>
<td>unknown</td>
<td>MOL fast</td>
<td>unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not near</td>
<td>unknown</td>
<td>not slow</td>
<td>unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not far</td>
<td></td>
<td>not fast</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not very near</td>
<td></td>
<td>not very slow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not very far</td>
<td></td>
<td>not very fast</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not MOL near</td>
<td></td>
<td>not MOL slow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not MOL far</td>
<td></td>
<td>not MOL fast</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td></td>
<td>stationary</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

example, Figure 6.4 shows the fuzzy membership functions for “near,” “medium,” and “far.” Note that “near” and “far” utilize sigmoidal membership functions, and “medium” uses a Gaussian membership function. (For more on fuzzy membership functions, see Section B.1.) These three basic fuzzy sets form the other ones through fuzzy complement and hedges [29]. The complement of a fuzzy set is defined as follows

\[ \mu_{\neg A}(x) = 1 - \mu_A(x). \] (6.6)

The hedge “Very,” which has the effect of narrowing the membership function, is defined by:

\[ \mu_{\text{very } A}(x) = [\mu_A(x)]^2. \] (6.7)

The hedge “more or less” has the effect of widening a membership function; it given by:

\[ \mu_{\text{MoreOrLess } A}(x) = \sqrt{\mu_A(x)}. \] (6.8)

Note that it only makes sense to define these hedges for “near” and “far;” it does not make sense to say “very medium.” Figure 6.5 shows the membership function for “near”
along with all the possible hedges that can be formed from it. Finally, the membership function for “unknown” is one for all distances.

Most of the fuzzy membership functions for speed are of the same form as the ones for distance, except that they are scaled to convey the concepts of “slow,” “medium,” and “fast.” The one exception is the membership function for “stationary;” it uses a triangular membership function that is 1.0 at 0 meters per second and decreases linearly to 0.0 at 1.0 meters per second. (Obviously, the concept of “stationary” is not actually fuzzy, something is either stationary or it is not, but this membership function reflects that the fact that targets which are moving very slowly could appear to be stationary when viewed from a distance.)

Fig. 6.4: Fuzzy membership functions for “near,” “medium,” and “far” distances

Heading and bearing are both described by triangular membership functions. For instance, the membership function “north” is 0 below −45 degrees, increases linearly until it is 1 at 0 degrees, then decreases linearly until it reaches 0 at 45 degrees. The
Fig. 6.5: Fuzzy membership functions formed through applying complement and hedges to “near”
membership functions for all of the other directions follow the same pattern. The membership function for “unknown” is one for every angle.

Figure 6.6 shows the membership function (as a function of lateral acceleration) for left and right turns. Since the constant velocity model does not have a lateral acceleration state, the lateral acceleration is always zero for targets in this mode. Left and right turns once again use sigmoidal membership functions. The membership function for “straight or unknown” is one for all lateral accelerations. It was chosen to combine straight and unknown, because a turning target may appear to be going straight if viewed from a certain angle. Also note that, although it may make sense to say things such as “turning gently” or “turning sharply,” no hedges were implemented on these membership functions.

![Fig. 6.6: Fuzzy membership functions for turning left and turning right](image)

The final attribute reported by the soft sensor is the binary attribute driving style. The driving style can either be “normal or unknown” or “erratic.” Note that it also could have been possible to attach a confidence to the soft sensor’s classification of
the driving style as erratic, but that was not implemented in this simulation. The concept of “erratic” is modeled as a set of regimes in the tracking algorithm (see Section 6.1.4 for a description of the possible regimes), so the membership function for “erratic” is based on the regime $r_k$ of the target. Table 6.2 shows the values of $\mu_{\text{erratic}}(r_k)$ and $\mu_{\text{normal unknown}}(r_k)$ for different values of $r_k$.

<table>
<thead>
<tr>
<th>$r_k$</th>
<th>$\mu_{\text{erratic}}(r_k)$</th>
<th>$\mu_{\text{normal unknown}}(r_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal straight</td>
<td>0.1</td>
<td>1.0</td>
</tr>
<tr>
<td>normal left turn</td>
<td>0.2</td>
<td>1.0</td>
</tr>
<tr>
<td>normal right turn</td>
<td>0.2</td>
<td>1.0</td>
</tr>
<tr>
<td>erratic straight</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>erratic left turn</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>erratic right turn</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 6.2: Values for "erratic" membership function

Although they are more informative than linguistic descriptions, these fuzzy membership functions still cannot be used as an input to the generalized Bayes filter. In order to be used, they must be converted to generalized likelihoods. These likelihoods take on the form of $f(\Theta_k|x_k, r_k)$. In this case, $\Theta$ contains information about the distance, bearing, speed, heading, turn direction, and driving style of the car. If these items are assumed to be conditionally independent, given the state and regime, the likelihood can be factored as follows:

$$
 f(\Theta_k|x_k, r_k) = f(\theta_{\text{dist}}_k|x_k, r_k)f(\theta_{\text{bearing}}_k|x_k, r_k)f(\theta_{\text{speed}}_k|x_k, r_k)
$$

$$
 f(\theta_{\text{heading}}_k|x_k, r_k)f(\theta_{\text{turning}}_k|x_k, r_k)f(\theta_{\text{style}}_k|x_k, r_k).
$$

Because of this factorization, each factor in the likelihood can be handled separately, then the factors are multiplied to get the final likelihood.

Recall from Section 5.3.2 that the likelihood of a fuzzy measurement is given as:

$$
 f(\Theta_{\mu_k}|x_k) = \Pr(\eta(x_k) \in \Sigma_A(\mu_k)) = \mu_k(\eta(x_k)).
$$

(6.10)
Taking distance as an example, the function $\eta(x)$, which maps from the state space to the measurement space, is simply the distance from the target to the sensor:

$$\eta_d(x_k) = \sqrt{x_k^2 + y_k^2} \quad (6.11)$$

(assuming the sensor is at the origin). The fuzzy membership function $\mu(d)$ is one of the membership functions for distance given above. As an example, suppose the soft sensor reports that the target is “near,” then the likelihood of this measurement, given that the target’s state is $x_k$ is

$$f(\text{near}|x_k) = \mu_{\text{near}}(\eta_d(x_k)) = \mu_{\text{near}} \left( \sqrt{x_k^2 + y_k^2} \right), \quad (6.12)$$

where $\mu_{\text{near}}(d)$ is the fuzzy membership function shown in Figure 6.5a. The likelihoods for “medium,” “far,” “very near,” etc. are constructed identically. Likelihoods for bearing, speed, heading, and turning are constructed analogously, with

$$\eta_{\text{bearing}}(x_k) = \arctan \left( \frac{y_k}{x_k} \right) \quad (6.13a)$$

$$\eta_s(x_k) = \sqrt{x_k^2 + y_k^2} \quad (6.13b)$$

$$\eta_{\text{heading}}(x_k) = \arctan \left( \frac{\dot{y}_k}{\dot{x}_k} \right) \quad (6.13c)$$

$$\eta_{\text{turning}}(x_k) = a_k, \quad (6.13d)$$

and the fuzzy membership functions taking on the forms described previously. The discrete driving characteristic parameter operates on the regime of the target; therefore, $\eta_{\text{driving}}(r_k) = r_k$. The fuzzy membership functions $\mu_{\text{erratic}}(r_k)$ and $\mu_{\text{normal unknown}}(r_k)$ were given in Table 6.2.

### 6.1.3 Simulation approach

In order to simulate the motion of the vehicle, three trajectories were generated in Matlab: normal (Figure 6.1), maneuver (Figure 6.2), and erratic (Figure 6.3). In order
to simulate the different rates that sensors take measurements, the hard sensor model generated measurements at a rate of once per second and the soft sensor generated measurements at a rate of once every ten seconds. To simulate the hard sensor, the actual range, bearing, and range-rate were calculated, then noise were added to them.

Simulating the soft sensor was more involved, because human beings are difficult to model mathematically. What follows is not meant to be a high-fidelity simulation of how a person may report on the situation, but rather a proof of concept simulation. Taking distance as an example, the soft sensor simulator generates a fuzzy measurement using the following procedure:

1. Calculate the distance from the sensor to the target.

2. Evaluate the fuzzy membership functions for “near,” “medium,” “far,” “very near,” “very far,” “more or less near,” “more or less far,” “not near,” “not far,” “not very near,” “not very far,” “not more or less near,” “not more or less far,” and “unknown” at this distance. This represents the degree that the distance belongs to each of these fuzzy sets.

3. Normalize these possibilities to sum to one, so that they can be treated as if they were probabilities.

4. Choose one of the fuzzy sets at random according to these pseudo-probabilities; this is the simulated output of the sensor.

It is important to point out here, that there is no assumed randomness in the model of the soft sensor; randomness is simply used in the simulation in order to simulate how a soft sensor may behave. The soft sensor’s reports on bearing, speed, heading, and turning are all generated identically.

The classification of the driving style as erratic is handled slightly differently. Notice from Figure 6.3 that the erratic target travels in a straight line at constant velocity for a while before starting to accelerate, weave, and turn at random. It is assumed that the longer the target has been weaving, making sudden starts and stops, or making other aggressive maneuvers, the more likely the soft sensor is to classify its motion as erratic.
Therefore, the probability that the soft sensor will classify a target as erratic is given by

$$p_{\text{erratic}} = \min\left(\frac{t - t_e}{t}, 0\right),$$

(6.14)

where $t$ is the current time and $t_e$ is the time when the target made its first erratic maneuver ($t_e$ is hard coded in the simulation based on visually inspecting the trajectory). Once again, it is important to point out that the randomness here is simply for the sake of the simulation; there is no randomness in the soft sensor model itself.

An important assumption in this simulation is that the human sensor makes his or her classification of the driving style as erratic based only what he or she is observing, not on the track itself. This assumption keeps the problem tractable, without it the system would have to account for the degree to which the classification of the driving style is based on track data (and therefore is not statistically independent of the track). In an actual system, this may or may not be a valid assumption. Some people may act only as sensors, but not need to use the fused information; in their case this assumption would hold. Other people (such as battlefield commanders) may need access to the fused data in order to to make command decisions; if these people also act as sensors, then their data may not be statistically independent of the track data.

### 6.1.4 Tracking algorithm

Algorithm 6.1 shows the particle filtering algorithm used in simulation one. Note that there are many similarities between it and the hybrid particle filtering algorithm (Algorithm 5.2), but it has been adapted to process hard and soft measurements. In the initialization step of the algorithm, rather than assuming some initial probability on the state, the difference between the first two measurements are used in order to form an estimate of the target’s position and velocity.

From the two motion models described in Section 6.1.1 six possible motion regimes are formed: constant velocity, left turn, right turn, erratic constant velocity, erratic left turn, and erratic right turn. There are two major differences between the normal regimes and the erratic regimes: the erratic regimes have higher process noise variances...
and targets in the straight line erratic mode are much more likely to transition into one of the two turning modes than targets in the standard straight line mode are. These transition probabilities are embedded in the regime transition matrix $\Pi$.

The initial mode probabilities are set to $p(r_0) = [0.6 \ 0.08 \ 0.08 \ 0.16 \ 0.04 \ 0.04]$ for the modes normal constant velocity, normal left turn, normal right turn, erratic constant velocity, erratic left turn, and erratic right turn, respectively. The mode transition probabilities are given as

$$
\Pi = \begin{bmatrix}
0.8 & 0.05 & 0.05 & 0.1 & 0 & 0 \\
0.3 & 0.5 & 0.1 & 0 & 0.1 & 0 \\
0.3 & 0.1 & 0.5 & 0 & 0 & 0.1 \\
0 & 0 & 0 & 0.6 & 0.2 & 0.2 \\
0 & 0 & 0 & 0.35 & 0.4 & 0.25 \\
0 & 0 & 0 & 0.35 & 0.25 & 0.4
\end{bmatrix}.
$$

Taking the first row of this table as an example, a particle initially in the normal constant velocity regime has a 0.8 probability of staying in that regime, a 0.05 probability of transitioning to normal left turn, a 0.05 probability of transitioning to normal right turn, and a 0.1 probability of transitioning to erratic constant velocity. Note that in the upper left part of the table, which refers to the “normal” regimes, that particles are unlikely to go from straight line to turning and turning particles are likely to either continue turning the same direction or transition back to straight line. In the lower right portion of the table, which deals with the “erratic” regimes, particles in the constant velocity erratic regime are more likely to turn one way or another and particles in one of the erratic turning regimes are more likely to suddenly turn the other direction. A final item to note from this matrix is that the 0.1 values in the upper right corner mean that there is a chance for “normal” particle to become erratic (such as would happen if a suspect suddenly realized that he was being followed). The lower left corner, on the other hand, is all 0 because an erratically driving car is not likely to suddenly start driving normally.
In Step 7 of Algorithm 6.1 particles’ states are updated using one of the two regime conditioned motion equations. Normal and erratic constant velocity particles are updated using Equation (6.1), but the process noise for the erratic case is five times the process noise for the normal case. Likewise, the normal and erratic turning modes both use equation (6.2), but the erratic cases have five times the process noise and twice the lateral acceleration of the normal cases.

In Step 9 of Algorithm 6.1, if both hard and soft measurements are present, their likelihoods are multiplied to get a final likelihood. (This assumes that the hard and soft sensor measurements are conditionally independent, which should be a valid assumption as long as the soft sensor does not have access to the hard sensor data, which could bias his report). If no soft sensor measurement is available, the weights are simply updated using the hard sensor measurement.

6.1.5 Results

The tracking algorithm presented in this section was tested by running 100 Monte Carlo simulations of the target being tracked. The same trajectory was used across all 100 simulations, but the noise that was added to the hard sensor measurements was random, as were the decisions made by the soft sensor. For all of the simulations, the sensor noise standard deviation was 15 meters in range, 0.1 radians in bearing, and 5 meters per second in range rate. The soft sensor measurements were generated according to the procedure described in Section 6.1.3.

In all of the simulations, process noise (acceleration) standard deviation was 1 meters per second squared for the normal modes and 5 meters per second squared for the erratic modes. For the normal turning modes, the initial lateral acceleration was normally distributed with a mean of plus or minus 3 meters per second squared and a standard deviation of 2.5 meters per second squared. For the erratic turning modes, the mean lateral acceleration was 6 meters per second squared with a standard deviation of 5 meters per second squared.
Algorithm 6.1 Particle filter for simulation one

1: Generate $N$ particle states using the first two hard sensor measurements and a two-point differencing [7].
2: Randomly select the mode of each particle according to the probability $p(r_0)$.
3: Add random lateral acceleration to those particles in the turning mode.
4: for $k = 1, 2, \ldots$ do
5: Perform regime transition on the particles (Algorithm 5.3):
   \[ \{r^n_k\} = \text{RT}(\{r^n_{k-1}\}) \]  \hspace{1cm} (6.16)
6: for $i = 1$ to $N$ do
7: Draw a new particle according to the regime conditioned state update probability density function:
   \[ x^i_k \sim f(x_k|x^i_{k-1}, r^i_k) \]  \hspace{1cm} (6.17)
8: if Soft sensor measurement is available then
9: Update the weight of the particle using the regime conditioned hard and soft sensor generalized likelihoods:
   \[ \tilde{w}^i_k = w^i_{k-1} f(z_k|x^i_k, r^i_k) f(\Theta_k|x^i_k, r^i_k). \]  \hspace{1cm} (6.18)
else
10: Update the weight of the particle using the regime conditioned hard sensor measurement likelihood:
   \[ \tilde{w}^i_k = w^i_{k-1} f(z_k|x^i_k, r^i_k). \]  \hspace{1cm} (6.19)
end if
end for
for $i = 1$ to $N$ do
14: Scale the weights so that they sum to one as follows:
   \[ w^i_k = \frac{\tilde{w}^i_k}{\sum_{j=1}^{N} \tilde{w}^j_k} \]  \hspace{1cm} (6.20)
end for
16: Calculate $N_{\text{eff}}$ using Equation (5.18).
17: if $N_{\text{eff}} < N_{\text{thr}}$ then
18: Resample using Algorithm A.6:
   \[ \{y^n_k, w^n_k\} = \text{RESAMPLE}(\{y^n_k, w^n_k\}) \]  \hspace{1cm} (6.21)
end if
20: end for
6.1.5.1 Hard sensor only

For the sake of comparison, all three trajectories were processed using a hard sensor only. Figure 6.7 shows the root mean square (RMS) errors in position and velocity for the hard sensor tracking the constant velocity target (Figure 6.1) over 100 runs using 1200 particles. A few things to note are that the initial error is quite high, because the initial particles were generated using the difference between two noisy measurements. The RMS error decreases over the course of the simulation, until the target gets close to the sensor, then it increases as the target starts driving away again. The reason the error begins increasing again after that point is due to the fact that the farther the target is from the sensor, the larger effect noise in the bearing measurement can have on the position estimate.

Figure 6.8 shows the RMS position and velocity error for the maneuver target (Figure 6.2), averaged over 100 runs. This time, 1575 particles were used instead of 1200. The number of particles to use was chosen by slowly increasing the number of particles until a point came where increasing the number of particles stopped improving tracking results. Note that in these figures, there is an increase in both position and velocity errors starting at 40 seconds into the simulation; this corresponds to when the target makes a turn. Figure 6.11 shows the average number of particles in each mode, also averaged over 100 runs. Note that at 40 seconds into the run the number of particles in the right turn mode increases, while the number of particles in the straight and left turn modes decrease. Interestingly, the number of particles in the right turn mode doubles from 200 to 400, but there are still far more particles in the straight mode. It appears that it only takes a few particles in the turning mode for the filter to track the turn. Once the target stops turning, the RMS errors in position and velocity once again decrease, and the number of particles in the right turn mode decreases to 200.

Figure 6.10 shows the RMS position and velocity errors for the hard sensor tracking the erratic target (Figure 6.3) with 1500 particles, and Figure 6.11 shows the average number of particles in each mode. As could be expected, the errors for this simulation are much higher than the other two, because the target makes many aggressive maneuvers, including weaving, sharp turns, and accelerations and decelerations. This problem is
Fig. 6.7: Root mean square (RMS) position and velocity errors for 100 runs of simulation one with constant velocity trajectory, hard sensor only, and 1200 particles (units are meters and meters per second)
Fig. 6.8: Root mean square (RMS) position and velocity errors for 100 runs of simulation one with maneuver trajectory, hard sensor only, and 1575 particles (units are meters and meters per second)
Fig. 6.9: Average number of particles in each mode for 100 runs of simulation one with maneuver trajectory, hard sensor only, and 1575 particles.
exasperated because many of these aggressive maneuver occur far from the sensor. Note that in Figure 6.11, the tracker correctly identifies the target’s driving style as erratic. This happens even though the hard sensor cannot directly classify the target’s driving style, because the erratic particles approximate the target’s motion better and are more likely to be resampled.

6.1.5.2 Soft sensor only

For the sake of comparison, the erratic target was tracked using only a human observer. For the soft-only simulation, the soft sensor collected measurements at a rate of once every five seconds, instead of once every ten. Because of the slow update frequency of the soft sensor, and the fact that the soft sensor is extremely imprecise, the filter was still initialized using two hard sensor measurements. After this, all updates used only soft sensor measurements. Figure 6.12 shows the RMS position and velocity errors averaged over 100 runs. The first thing to note in these figures is that the scales are different from all of the other figures in this section. The errors for the soft sensor only were up to an order of magnitude higher than those for the hard sensor. This is even though the soft sensor used 10,000 particles to track the target. There are several reasons for such a large error: first is the fact that the soft sensor only updates once every ten seconds, rather than once a second; second is the extreme imprecision in the soft sensor’s measurements. The fuzzy membership function for “near” is above 0.5 for any distance between 0 and 400 meters and is above 0.1 for any distance less than 600 meters; therefore, the soft sensor could reasonably consider any distance below 600 meters “near.” Also, the soft sensor could apply hedges, such as “more or less near” or “not very near,” which are even more imprecise, or the soft sensor could simply return “unknown,” which yields no information about the target’s position. Figure 6.13 shows the average number of particles in each mode for the soft sensor. Somewhat paradoxically, the soft sensor takes longer than the hard sensor to determine that the target is driving erratically, even though the soft sensor has the ability to classify the driving style of the target.

From these results it is apparent that a human cannot track a target nearly as well as a radar. This why it does not make sense to do target tracking using only a
Fig. 6.10: Root mean square (RMS) position and velocity errors for 100 runs of simulation one with erratic trajectory, hard sensor only, and 1500 particles (units are meters and meters per second)
Fig. 6.11: Average number of particles in each mode for 100 runs of simulation one with erratic trajectory, hard sensor only, and 1500 particles.
human observer. A human observer is much better at making higher level inferences on properties such as classification or intent than it is at determining location and velocity. The reason for gathering location and velocity information from the soft sensor is not to precisely track the target, the radar is much better suited for this. By collecting this information it is possible to associate reports from the soft sensor with items being tracked by the hard sensor. This means the hard sensor’s tracks can be updated with data from the soft sensor (such as classification and intent) that is not readily available to the hard sensor.

6.1.5.3 Hard and soft sensors

Combined hard and soft sensor tracking was tested primarily using the erratic trajectory. Several different particle counts were used, but the results in this subsection focus on the 1500 particle case, because this was found to be a good trade off between accuracy and speed. Figure 6.14 shows the evolution of the particle set over the course of one run. Note from this figure that the particles are initially very spread out, due to the fact that they are initialized using two noisy measurements; then the particle cloud gets tighter as more measurements are collected; then as the target starts making some more aggressive maneuvers the particle cloud spreads out slightly. From the figure, it is apparent that the noise in bearing is much more severe than the noise in range, as is seen by the fact that the particles are usually spread out in an arc. Finally, note that the red ‘x’ representing the target’s actual location is usually somewhere in the particle cloud, although it is not always in the densest part.

Figure 6.15 shows the north and east position and velocity errors for a single run along with one-sigma error contours. The error contours were calculated by finding the covariance of the particle states. From these figures, note that the errors increase and decrease over the course of the run, but they do stay within one to two standard deviations of the mean. This is important, because it means that the filter is not overconfident in bad estimates, which can lead to filter divergence. Figure 6.16 shows the number of particles in each mode for the same run. From this figure, it is more apparent than in the averaged figures that particles are constantly changing between modes. The straight line
Fig. 6.12: Root mean square (RMS) position and velocity errors for 100 runs of simulation one with erratic trajectory, soft sensor only, and 10,000 particles (units are meters and meters per second)
Fig. 6.13: Average number of particles in each mode for 100 runs of simulation one with erratic trajectory, soft sensor only, and 10,000 particles
Fig. 6.14: Time history of particle distribution, along with a red ‘x’ at the actual target location, for simulation one with erratic trajectory, hard and soft sensors, and 1500 particles (units are meters)
mode, however, tends to be dominant until 100 seconds in, when an aggressive maneuver, a measurement from the soft sensor, or both cause the erratic mode to dominate.

Figure 6.17 shows the RMS position and velocity errors over the course of 100 runs and Figure 6.18 shows the average number of particle in each mode averaged over 100 runs. These figures appear to be very similar to the same figures for the hard sensor only. Therefore, it would appear from these figures that the addition of a soft sensor does not make a significant difference in terms of tracking results for this scenario. The next subsection will show, however, that the soft sensor can make a difference when there are fewer particles.

6.1.5.4 Analysis

Table 6.3 shows the RMS position and velocity errors, averaged over the entire trajectory and over all runs, for several different tracking scenarios. All of the results are for 100 runs. There are several things to note from the table. First, it is generally true that the more particles that are used to track the target, the smaller the errors will be. This makes perfect sense, since more particles should yield a better approximation to the posterior probability distribution. This increase is far from linear, however. Taking as an example the hard sensor only tracking the erratic target, increasing the number of particles from 200 to 300 decreases the RMS position error from 215 meters to 58.8 meters, but increasing the number of particles further to 1500 only decreases the RMS position error to 48 meters. Because computational effort does increase more or less linearly with the number of particles, there is definitely a point where increasing the number of particles further is not worth the additional computational effort.

The second important thing to note in the table is the difference in errors between the hard sensor only and the hard and soft sensors. For 300 or 1500 particles, the difference in errors is slight, on the order of 1 to 5 meters in position and 1 meter per second in velocity. When the number of particles is decreased, however, the difference is magnified. For 200 particles the hard sensor has an RMS position error of 215 meters and an RMS velocity error of 18 meters per second. For 200 particles the combined hard and soft sensors have an RMS position error of 38.7 meters and an RMS velocity
Fig. 6.15: Position and velocity errors, with one sigma contours, for a single run of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles (units are meters and meters per second)
Fig. 6.16: Number of particles in each mode for a single run of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles
Fig. 6.17: Root mean square (RMS) position and velocity errors for 100 runs of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles (units are meters and meters per second)
Fig. 6.18: Average number of particles in each mode for 100 runs of simulation one with erratic trajectory, hard and soft sensors, and 1500 particles
error of 12.6 meters per second. The combined hard and soft filter was able to track targets with as few as 133 particles, but the hard sensor only filter started to diverge for particle counts lower than 200. From this, it appears that even though the soft sensor is extremely poor at tracking on its own, when it is used to supplement a hard sensor it can improve the tracking performance. As the number of particles is increased, however, this benefit diminishes.

<table>
<thead>
<tr>
<th>Sensors</th>
<th>Trajectory</th>
<th>Number of particles</th>
<th>RMSEP [m]</th>
<th>RMSEV [m/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard-only</td>
<td>constant</td>
<td>400</td>
<td>65.6</td>
<td>7.1</td>
</tr>
<tr>
<td></td>
<td>maneuver</td>
<td>1200</td>
<td>38.7</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>500</td>
<td>35.8</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1575</td>
<td>31.9</td>
<td>6.7</td>
</tr>
<tr>
<td>soft-only</td>
<td>erratic</td>
<td>200</td>
<td>215</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>58.8</td>
<td>15.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1500</td>
<td>48</td>
<td>12</td>
</tr>
<tr>
<td>hard and soft</td>
<td>erratic</td>
<td>10,000</td>
<td>796</td>
<td>51.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>133</td>
<td>71.8</td>
<td>17.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>38.7</td>
<td>12.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>54</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1500</td>
<td>45.7</td>
<td>11.7</td>
</tr>
</tbody>
</table>

Table 6.3: Performance of various tracking scenarios

Another way to compare the hard and soft sensors is in terms of classification. In this simulation, the motion of a target was classified as normal, turning, or erratic. This classification influenced the motion of the target, and could have been used to help infer that a target was potentially hostile. Table 6.4 shows the time that it took to correctly classify a target as erratic. For these results, the target was considered classified when over fifty percent of the particles were in the erratic regime. This was based on examining the average number of particles per mode for 100 runs of the simulation.

The first thing to note from this table is that the addition of a soft sensor has little bearing on whether or not a target is classified as erratic. This result is counter-intuitive because the soft sensor can directly classify a target as erratic, whereas the hard sensor indirectly classifies the target based on its motion. Further inspection of Equation (6.14) helps to explain this behavior. The soft sensor randomly classifies the
target in proportion to how long the target has been traveling erratically (as a percentage of the total length of the trajectory). This makes the soft sensor very slow to respond to a change in trajectory. In simulation, it would be possible to change this behavior by making the soft sensor more responsive. A possible area for future research could be to evaluate how quickly various people are able to classify a target as traveling erratically so that more realistic values could be used in the simulation.

Another observation from Table 6.4 is that the target is classified faster when there are 300 particles, than when there are 667. In fact, the target is classed as erratic in 28 or 29 seconds, which is prior to the target making any aggressive maneuvers. In Figure 6.3 the first aggressive velocity change is not until about 50 seconds. In other words, this is actually a miss-classification. For more than 667 particles, the number of particles has little influence on the classification performance: the target is classified as erratic after 102 or 103 seconds. In Figure 6.3 this corresponds to the “hairpin” turn, when the target’s trajectory is unmistakeably erratic.

<table>
<thead>
<tr>
<th>Sensors</th>
<th>Number of particles</th>
<th>Time to classify trajectory [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard-only</td>
<td>300</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>667</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>1500</td>
<td>102</td>
</tr>
<tr>
<td>hard and soft</td>
<td>300</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>667</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>1500</td>
<td>103</td>
</tr>
</tbody>
</table>

Table 6.4: Time to correctly classify the erratic trajectory

6.2 Simulation two: multiple target classes

The second simulation uses a single radar and a single soft sensor, both located at the origin. This time, there are several different classes of targets that can be tracked. These include a normally driving car, an erratically driving car, a tank, a motorcycle, a normal person on foot, and a lost person on foot. By adding different target types, it is possible to evaluate both classification and tracking using a hard and soft sensor.
6.2.1 Target motion models

The normal car, erratic car, and motorcycle all utilize the constant velocity and maneuver motion models described in Section 6.1.1. The motion model for the tank had five states: north position, east position, heading, left tread velocity, and right tread velocity. It’s motion is described by the following nonlinear equations [213]

\[
x_{k+1} = x_k + \frac{BT}{6} \cos(\theta_k) + \frac{T}{3} \left(\frac{AT}{2} + B\right) \cos\left(\frac{CT^2}{4} + \frac{DT}{2} + \theta_k\right)
+ \frac{T}{6} (AT + B) \cos(CT^2 + DT + \theta_k)
\]

(6.22a)

\[
y_{k+1} = y_k + \frac{BT}{6} \sin(\theta_k) + \frac{T}{3} \left(\frac{AT}{2} + B\right) \sin\left(\frac{CT^2}{4} + \frac{DT}{2} + \theta_k\right)
+ \frac{T}{6} (AT + B) \sin(CT^2 + DT + \theta_k)
\]

(6.22b)

\[
\theta_{k+1} = \theta_K + CT^2 + DT
\]

(6.22c)

\[
v_{l_{k+1}} = v_{l_k} + a_{l_k} T
\]

(6.22d)

\[
v_{r_{k+1}} = v_{r_k} + a_{r_k} T,
\]

(6.22e)

where

\[
A = \frac{a_{l_k} + a_{r_k}}{2}
\]

(6.23a)

\[
B = \frac{v_{l_k} + v_{r_k}}{2}
\]

(6.23b)

\[
C = \frac{a_{r_k} - a_{l_k}}{2b}
\]

(6.23c)

\[
D = \frac{v_{r_k} - v_{l_k}}{b},
\]

(6.23d)

\[b\] is the width of the tank, \(T\) is the time step, and \(a_{l_k}\) and \(a_{r_k}\) are white noise inputs, with standard deviation \(\sigma_{ak}\). The motion model for both the normal person and the lost person has four states: north position, east position, heading, and velocity. It is governed by the following

\[
x_{k+1} = x_k + \frac{BT}{6} \cos(\theta_k) + \frac{T}{3} \left(\frac{AT}{2} + B\right) \cos\left(\frac{CT^2}{4} + \frac{DT}{2} + \theta_k\right)
+ \frac{T}{6} (AT + B) \cos(CT^2 + DT + \theta_k)
\]

(6.22a)

\[
y_{k+1} = y_k + \frac{BT}{6} \sin(\theta_k) + \frac{T}{3} \left(\frac{AT}{2} + B\right) \sin\left(\frac{CT^2}{4} + \frac{DT}{2} + \theta_k\right)
+ \frac{T}{6} (AT + B) \sin(CT^2 + DT + \theta_k)
\]

(6.22b)

\[
\theta_{k+1} = \theta_K + CT^2 + DT
\]

(6.22c)

\[
v_{l_{k+1}} = v_{l_k} + a_{l_k} T
\]

(6.22d)

\[
v_{r_{k+1}} = v_{r_k} + a_{r_k} T,
\]

(6.22e)
nonlinear state equations

\[
x_{k+1} = x_k + T v_k \cos(\theta_k) \quad (6.24a)
\]

\[
y_{k+1} = y_k + T v_k \sin(\theta_k) \quad (6.24b)
\]

\[
\theta_{k+1} = \theta_k + T \omega_k \quad (6.24c)
\]

\[
v_{k+1} = v_k + T a_k, \quad (6.24d)
\]

where \( T \) is once again the time step and \( \omega_k \) and \( a_k \) are white noise inputs with standard deviations \( \sigma_{\omega k} \) and \( \sigma_{a_k} \), respectively.

6.2.2 Sensor measurement models

The measurement model for the hard sensor in this simulation is identical to the one described in Section 6.1.2. Likewise, the soft sensor generates measurements of distance, bearing, speed, and heading the same way as in the last section. In this simulation, the soft sensor did not measure turn direction, because it is difficult to define lateral acceleration for the tank or the person on foot.

In place of the driving style classification of the previous section, the soft sensor returns a single discrete classification, which contains information about both the target’s class and behavior. It can take on one of the following values: “normal car,” “erratic car,” “tank,” “motorcycle,” “normal person,” “lost person,” or “unknown.” It could have been possible to attach a confidence to this classification, but that was not implemented at this time. In order to use these classifications in a likelihood, each possible classification had to be treated as a fuzzy set, with a fuzzy membership function. Table 6.5 shows the values for each fuzzy membership function, for every possible target class. Note that some of these concepts, such as “erratic” or “lost” are fuzzy in nature, so it makes perfect sense for a normal car to have a membership of 0.5 in the class “erratic car” or for a lost person to have a membership of 0.5 in the class “normal person.” Other items, which are less fuzzy are still treated as such here. The concept of a tank is not necessarily fuzzy (although one could possibly argue that a half-track can be considered
a tank or a truck), but it was still modeled as a fuzzy set. What’s being stated by the fuzzy set for tank is not necessarily that a normal car has a membership of 0.1 in the class “tank,” but that a car resembles a tank enough that a person may mistakenly call a car a tank, especially if he is fatigued or his view is partially obstructed. A final thing to note from the table is that the value for the fuzzy set “unknown” is 1.0 for every class of target. Having constructed this table, the likelihood of a fuzzy measurement is calculated exactly as in Section 6.1.2.

<table>
<thead>
<tr>
<th>Actual class</th>
<th>( \mu_{\text{n-car}} )</th>
<th>( \mu_{\text{e-car}} )</th>
<th>( \mu_{\text{tank}} )</th>
<th>( \mu_{\text{cycle}} )</th>
<th>( \mu_{\text{n-person}} )</th>
<th>( \mu_{\text{l-person}} )</th>
<th>( \mu_{\text{unknown}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal car</td>
<td>1.0</td>
<td>0.5</td>
<td>0.1</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>1.0</td>
</tr>
<tr>
<td>erratic car</td>
<td>0.5</td>
<td>1.0</td>
<td>0.1</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>1.0</td>
</tr>
<tr>
<td>tank</td>
<td>0.1</td>
<td>0.1</td>
<td>1.0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>1.0</td>
</tr>
<tr>
<td>motorcycle</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>1.0</td>
<td>0.1</td>
<td>0.1</td>
<td>1.0</td>
</tr>
<tr>
<td>normal person</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.1</td>
<td>1.0</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>lost person</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.1</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 6.5: Values for classification membership functions

### 6.2.3 Tracking algorithm

This simulation once again uses Algorithm 6.1 to track and classify targets, but the set of regimes are different. The car can once again be in six different regimes: “normal constant velocity,” “normal left turn,” “normal right turn,” “erratic constant velocity,” “erratic left turn,” and “erratic right turn.” The probability that the car is initially in each of these states is once again given by \( p(r_{\text{car}}) = [0.6 \ 0.08 \ 0.08 \ 0.16 \ 0.04 \ 0.04] \). The tank can only be in one regime so \( p(r_{\text{tank}}) = 1.0 \). The motorcycle can be in one of three regimes: “constant velocity,” “left turn,” or “right turn,” with initial probability \( p(r_{\text{motorcycle}}) = [0.8 \ 0.1 \ 0.1] \). The person can be in two regimes: “normal person” or “lost person,” with initial probability \( p(r_{\text{person}}) = [0.9 \ 0.1] \). If the class of the target were initially known, then one of these initial probability densities could be used to initialize all of the particles, ignoring all of the other classes. Due to the fact that the class of the target is not initially known, a composite initial density must be formed that accounts for all possible target classes. Assuming that each target class is equally likely,
it takes on the following form

\[ p(r_0) = \frac{[p(r_{\text{car }0}) p(r_{\text{tank }0}) p(r_{\text{motorcycle }0}) p(r_{\text{person }0})]}{4}. \]  

(6.25)

The mode transition probabilities are given as

\[
\Pi = \begin{bmatrix}
\Pi_{\text{car}} & 0_{6\times1} & 0_{6\times3} & 0_{6\times2} \\
0_{1\times6} & \Pi_{\text{tank}} & 0_{1\times3} & 0_{1\times2} \\
0_{3\times6} & 0_{3\times1} & \Pi_{\text{motorcycle}} & 0_{3\times2} \\
0_{2\times6} & 0_{2\times1} & 0_{2\times3} & \Pi_{\text{person}}
\end{bmatrix},
\]  

(6.26)

where \( \Pi_{\text{car}} \) was given in Equation (6.15) and

\[
\Pi_{\text{tank}} = [1] 
\]  

(6.27a)

\[
\Pi_{\text{motorcycle}} = \begin{bmatrix}
0.8 & 0.1 & 0.1 \\
0.3 & 0.6 & 0.1 \\
0.3 & 0.1 & 0.6
\end{bmatrix}
\]  

(6.27b)

\[
\Pi_{\text{person}} = \begin{bmatrix}
0.95 & 0.05 \\
0.0 & 1.0
\end{bmatrix}
\]  

(6.27c)

Because this matrix is block diagonal, it is basically stating that targets can change regimes within the same class, but they cannot change classes.

The other difference between this simulation and Algorithm 6.1 is the specific equations used in the regime conditioned state updates. The normal and erratic cars used the same state updates as they did before. The tank uses Equation (6.22) to update its state. The constant velocity motorcycle uses Equation 6.1 (the same as the constant velocity car), and the left and right turn motorcycles use Equation (6.2) (the same as the turning car), but with five times acceleration noise and three times the initial lateral acceleration, in order to account for the maneuverability of the motorcycle. The normal and lost person both use Equation (6.24), but the lost person has five times as much
noise in both speed and heading to account for the lost person turning around or starting and stopping.

### 6.2.4 Results

Simulation two was tested using the same basic procedure as simulation one. Results were once again averaged over 100 Monte Carlo simulations, and the update rate was once again 1 second. To make comparisons easier, all of the results presented in this section are for 1000 particles. One difference from simulation one was that the soft sensor updated once every 15 seconds instead of once every 10. (It was decided that this was a more realistic rate to update.) The soft sensor measurements were generated according to the procedure described in Section 6.1.3 (with appropriate changes made, since there is not a turning measurement and there is a new classification measurement). The hard sensor once again collected measurements at once per second with a measurement noise standard deviation of 15 meters in range, 0.1 radians in bearing, and 5 meters per second in range rate.

For the car targets, the process noise (acceleration) standard deviation was 1 meters per second squared for the normal car modes and 5 meters per second squared for the erratic car modes. For the normal car turning modes, the initial lateral acceleration was normally distributed with a mean of plus or minus 3 meters per second squared and a standard deviation of 1.5 meters per second squared. For the erratic turning modes, the mean lateral acceleration was 6 meters per second squared with a standard deviation of 3 meters per second squared. The tank process noise (which effects the acceleration of the left and right treads) was set to 1 meters per second squared. The constant velocity motorcycle used the same process noise parameters as the constant velocity normal car. The maneuvering motorcycle had a process noise of 5 meters per second squared in longitudinal and lateral acceleration. The initial lateral acceleration was normally distributed with a mean of plus or minus 9 meters per second squared with a standard deviation of 4.5 meters per second squared. The process noise standard deviation for the normal person was set to 1.2 radians per second in heading and 0.25
meters per second squared in acceleration. The lost person model used a process noise 6.0 radians per second in heading and 1.25 meters per second squared in acceleration.

Figure 6.19 shows the tank trajectory, heading and speed. Note that 50 seconds into the simulation, the tank turns in place (i.e. the heading changes, but the speed stays zero); this is a unique feature of the tank and person models. This is a difficult maneuver to track, because the heading of the tank is unobservable when it is stationary. Figure 6.20 shows the RMS position and velocity errors for tracking the tank. Note from the figures that the position and velocity errors both increase slightly at the end of the trajectory due to the maneuver, but the increase is slight, on the order of 10 meters in position and 10 meters per second in velocity. It’s apparent from this that due to the tank’s slow speed, it is relatively easy to track. Figure 6.21 shows the average number of particles in each mode for the tank. From the start of the simulation, tank is the most likely mode, but the number of particles in the tank mode increases at 15, 30, and 60 seconds; these jumps are most likely caused by soft sensor measurements.

Figure 6.22 shows the trajectory and speed for the motorcycle; this turned out to be the most difficult target to track (in terms of RMS error), due to its maneuvering capabilities. Figure 6.23 shows the RMS position and velocity errors for the motorcycle. Note that the RMS position error decreases until 30 seconds in, when the motorcycle makes its first of two aggressive turns and quickly accelerates out of the turn. The velocity error also increases from 40 seconds on. It is important to note, however, that though the error increases, the track doesn’t diverge; in fact, when the motorcycle begins approaching the sensor’s location the position error once again decreases (although the velocity error stays high). Figure 6.24 shows the average number of particles in each mode for this simulation. Due to the combination of the measurements from the soft sensor and the fact that the motorcycle is the only vehicle capable of the maneuvers it makes, the motorcycle mode quickly dominates.

Figure 6.25 shows the trajectory for the normal person. As should be expected, the person does not cover much ground in 60 seconds. Also, the person can make nearly instantaneous changes in heading. Figure 6.26 shows the RMS position and velocity errors for the person. As could be expected, these errors are small, due in large part
Fig. 6.19: Trajectory, heading and speed for the tank simulation

(a) Tank trajectory with a red ‘x’ at the sensor location (meters)

(b) Tank heading (degrees)

(c) Tank speed (meters per second)
Fig. 6.20: Root mean square (RMS) position and velocity errors for 100 runs of simulation two with tank trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)
Fig. 6.21: Average number of particles in each mode for 100 runs of simulation two with tank trajectory, hard and soft sensors, and 1000 particles.
(a) Motorcycle trajectory with a red ‘x’ at the sensor location (meters)

(b) Motorcycle speed (meters per second)

Fig. 6.22: Trajectory and speed for the motorcycle simulation
Fig. 6.23: Root mean square (RMS) position and velocity errors for 100 runs of simulation two with motorcycle trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)
Fig. 6.24: Average number of particles in each mode for 100 runs of simulation two with motorcycle trajectory, hard and soft sensors, and 1000 particles
to the fact that the person travels so slowly. Figure 6.27 shows the average number of particles in each mode for this trajectory. Note that the tank mode is actually quite likely until 15 seconds in, when the soft sensor classifies the target as a normal person. From this point on, an interesting phenomenon occurs: the number of particles in the normal person mode decreases, while the number of particles in the lost person mode increases, until a soft sensor measurement occurs (every 15 seconds), at which point the number of particles in the normal person mode jumps and the number of particles in the lost person mode decreases; this pattern then continues to repeat itself. The reason for this lies in the transition probabilities. Recall from Equation (6.27c), that the transition probabilities for the person had the following form

\[
\Pi_{\text{person}} = \begin{bmatrix} 0.95 & 0.05 \\ 0.0 & 1.0 \end{bmatrix}. \tag{6.28}
\]

This means that particles in the normal person mode have a probability of 0.05 of transitioning to the lost person mode (in other words, a person could get lost). On the other hand, particles in the lost person mode always stay in that mode (without help, it is rare for a lost person to suddenly find their way). This means that, all else being equal, 5 percent of the normal particles will switch to lost at every time step. If necessary, this behavior could be changed by modifying the transition probabilities so that there is a smaller probability of a normal person particle transitioning to lost, or by setting a very small probability for a lost person particle to transition to normal.

Figure 6.28 shows the trajectory and heading for the lost person simulation. As should be expected for a lost person, the subject is constantly changing heading and basically moves in a big circle. Figure 6.29 shows the RMS position and velocity errors for the lost person trajectory. Even though the lost person’s trajectory is quite erratic, the errors end up being very small because of how slowly the person moves. Figure 6.30 shows the average number of particles in each mode for the lost person trajectory. As was the case with the normal person, the tank mode is initially quite likely, but when the first soft sensor measurement occurs at 15 seconds, the number of particles in the lost person mode sharply increases. From this point on, lost person is the dominant mode.
Fig. 6.25: Normal person trajectory with a red ‘x’ at the sensor location (units are meters)
Fig. 6.26: Root mean square (RMS) position and velocity errors for 100 runs of simulation two with normal person trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)
Fig. 6.27: Average number of particles in each mode for 100 runs of simulation two with normal person trajectory, hard and soft sensors, and 1000 particles
(a) Lost person trajectory with a red ‘x’ at the sensor location (meters)

(b) Lost person heading (degrees)

Fig. 6.28: Trajectory and heading for the lost person simulation
Fig. 6.29: Root mean square (RMS) position and velocity errors for 100 runs of simulation two with lost person trajectory, hard and soft sensors, and 1000 particles (units are meters and meters per second)
Fig. 6.30: Average number of particles in each mode for 100 runs of simulation two with lost person trajectory, hard and soft sensors, and 1000 particles
Table 6.6 shows the average RMS position and velocity errors for each target type. Note that generally the errors are proportional to the maneuverability of the target: the tank, normal person, and lost person all have low errors; the normal and maneuver cars both have moderate errors; and the erratic car and motorcycle both have higher errors.

<table>
<thead>
<tr>
<th>Target</th>
<th>RMSEP</th>
<th>RMSEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal car</td>
<td>30.1</td>
<td>18.7</td>
</tr>
<tr>
<td>Maneuver car</td>
<td>35.4</td>
<td>16.3</td>
</tr>
<tr>
<td>Erratic car</td>
<td>50.5</td>
<td>20.1</td>
</tr>
<tr>
<td>Tank</td>
<td>9.6</td>
<td>8.8</td>
</tr>
<tr>
<td>Motorcycle</td>
<td>56.4</td>
<td>22.0</td>
</tr>
<tr>
<td>Normal person</td>
<td>7.7</td>
<td>4.8</td>
</tr>
<tr>
<td>Lost person</td>
<td>7.5</td>
<td>5.8</td>
</tr>
</tbody>
</table>

Table 6.6: Performance of tracker on each trajectory; each tracker used 1000 particles; results were averaged over 100 runs.

6.3 Simulation three: distributed hard and soft sensors

The third simulation was designed to approximate the search and rescue problem described in Section 1.3. As such, the radar and soft sensor that were placed at a centralized location and had a view of the entire search area were replaced with a swarm of distributed sensing agents. The sensing agents consisted of unmanned helicopters equipped with standard cameras, unmanned ground vehicles equipped with stereo cameras, and human observers. Each of these sensors possessed a limited field of view. For the purpose of this simulation, the sensing agents remained stationary throughout the course of the simulation, since control issues are outside the scope of this thesis. The sensors were laid out in a grid spaced 250 meters apart, with the UAVs located 200 meters above the ground. Figure 6.31a shows one possible layout for the sensing agents. UGVs and soft sensors both faced toward the origin of the coordinate system and the UAVs pointed their sensors straight down. Figure 6.31b shows all of the sensor fields of view superimposed over each other. Note that there are some areas that are viewed by multiple sensors and some that are not viewed by any sensors.
(a) Sensor locations with a blue ‘x’ for a UAV, a red ‘*’ for a UGV, and a green ‘+’ for a soft sensor.

(b) Sensor viewpoints with red corresponding to in view and blue corresponding to not in view.

Fig. 6.31: Sensor locations and viewpoints (all units are meters)
The other change from the last simulations deals with the class of the targets. The only possible class of target is a car, but the car could be a friend, foe, or neutral. This was done to reduce the need for many motion models, while still utilizing the soft sensor’s classification ability. Also, because of the addition of the UAVs, tracking was performed in three dimensions.

### 6.3.1 Target motion models

There were three different models that could govern the car’s motion: constant velocity, Singer maneuver model [214], and coordinated turn [215]. These models were chosen as they are standard models, which are often used for interacting multiple model filtering [161]. The constant velocity model uses the following linear equation of motion

\[
x_{k+1} = \begin{bmatrix}
  1 & 0 & 0 & T & 0 & 0 \\
  0 & 1 & 0 & 0 & T & 0 \\
  0 & 0 & 1 & 0 & 0 & T \\
  0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 & 1 \\
  0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}x_k + v_k,
\]

where \( v_k \) is zero mean white noise with covariance

\[
Q = \begin{bmatrix}
  \frac{T^3}{3} & 0 & 0 & \frac{T^2}{2} & 0 & 0 \\
  0 & \frac{T^3}{3} & 0 & 0 & \frac{T^2}{2} & 0 \\
  0 & 0 & \frac{T^3}{3} & 0 & 0 & \frac{T^2}{2} \\
  \frac{T^2}{2} & 0 & 0 & T & 0 & 0 \\
  0 & \frac{T^2}{2} & 0 & 0 & T & 0 \\
  0 & 0 & \frac{T^2}{2} & 0 & 0 & T \\
\end{bmatrix} \sigma_a^2
\]

\( T \) is the time step, and the state is given as \( x_k = [x \ y \ z \ v_x \ v_y \ v_z]^T \); this is essentially a three dimensional version of the constant velocity model from Section 6.1.
The Singer acceleration model, which accounts for sudden maneuvers by the target, is given by the following linear equation [161]

$$
\mathbf{x}_{k+1} = \begin{bmatrix}
1 & 0 & 0 & T & 0 & 0 & \frac{T^2}{2} & 0 & 0 \\
0 & 1 & 0 & 0 & T & 0 & 0 & \frac{T^2}{2} & 0 \\
0 & 0 & 1 & 0 & 0 & T & 0 & 0 & \frac{T^2}{2} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T
\end{bmatrix} \mathbf{x}_k + \mathbf{v}_k, \quad (6.31)
$$

where $\mathbf{v}_k$ is zero mean white noise with covariance

$$
Q = \begin{bmatrix}
\frac{T^5}{20} & 0 & 0 & \frac{T^4}{8} & 0 & 0 & \frac{T^3}{6} & 0 & 0 \\
0 & \frac{T^5}{20} & 0 & 0 & \frac{T^4}{8} & 0 & 0 & \frac{T^3}{6} & 0 \\
0 & 0 & \frac{T^5}{20} & 0 & \frac{T^4}{8} & 0 & 0 & \frac{T^3}{6} & 0 \\
\frac{T^4}{8} & 0 & 0 & \frac{T^3}{3} & 0 & 0 & \frac{T^2}{2} & 0 & 0 \\
0 & \frac{T^4}{8} & 0 & 0 & \frac{T^3}{3} & 0 & 0 & \frac{T^2}{2} & 0 \\
0 & 0 & \frac{T^4}{8} & 0 & \frac{T^3}{3} & 0 & 0 & \frac{T^2}{2} & 0 \\
\frac{T^3}{6} & 0 & 0 & \frac{T^2}{2} & 0 & 0 & T & 0 & 0 \\
0 & \frac{T^3}{6} & 0 & 0 & \frac{T^2}{2} & 0 & 0 & T & 0 \\
0 & 0 & \frac{T^3}{6} & 0 & \frac{T^2}{2} & 0 & 0 & T & 0
\end{bmatrix} \sigma_m^2, \quad (6.32)
$$

$T$ is the time step, and the state is given as $\mathbf{x}_k = [x \ y \ z \ v_x \ v_y \ v_z \ a_x \ a_y \ a_z]^T$. 
The coordinated turn model is given by the following nonlinear equation [161]

\[
\mathbf{x}_{k+1} = \begin{bmatrix}
1 & 0 & 0 & \frac{\sin(\omega_k T)}{\omega_k} & -\frac{1-\cos(\omega_k T)}{\omega_k} & 0 & 0 \\
0 & 1 & 0 & \frac{1-\cos(\omega_k T)}{\omega_k} & \frac{\sin(\omega_k T)}{\omega_k} & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & T & 0 \\
0 & 0 & 0 & \cos(\omega_k T) & -\sin(\omega_k T) & 0 & 0 \\
0 & 0 & 0 & \sin(\omega_k T) & \cos(\omega_k T) & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \mathbf{x}_k + \mathbf{v}_k,
\]

where \(\mathbf{v}_k\) is zero mean white noise with covariance

\[
Q = \begin{bmatrix}
\frac{T^4}{4} & 0 & 0 & \frac{T^3}{2} & 0 & 0 & 0 \\
0 & \frac{T^4}{4} & 0 & 0 & \frac{T^3}{2} & 0 & 0 \\
0 & 0 & \frac{T^4}{4} & 0 & 0 & \frac{T^3}{2} & 0 \\
\frac{T^3}{2} & 0 & 0 & T^2 & 0 & 0 & 0 \\
0 & \frac{T^3}{2} & 0 & 0 & T^2 & 0 & 0 \\
0 & 0 & \frac{T^3}{2} & 0 & 0 & T^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\sigma_a^2 T^2}{\sigma_a^2}
\end{bmatrix} \sigma_a^2,
\]

\(T\) is the time step, and the state is given as \(\mathbf{x}_k = [x \ y \ z \ v_x \ v_y \ v_z \ \omega]^T\); this is basically a three dimensional version of Equation (6.2), with the turn rate explicitly made a state (rather than the lateral acceleration).

### 6.3.2 Sensor measurement models

The first sensor type used in this simulation is a normal camera mounted on an unmanned helicopter. The camera has a limited range and angular field of view. The measurement model for the camera assumes that some sort of image processing has already been preformed internally (such as in Section 2.5.2), so that the camera returns an azimuth and elevation angle in the local north, east, down frame, with its origin at the sensor (as opposed to reporting measurements directly in the camera frame). Doing
so is possible because the camera is located onboard an autonomous vehicle equipped with an autopilot. This results in the following measurement equation

\[ z_k = \eta(x_k) + w_k = \begin{bmatrix} \tan^{-1}\left(\frac{y_k}{x_k}\right) \\ \tan^{-1}\left(\frac{z_k}{\sqrt{x_k^2 + y_k^2}}\right) \end{bmatrix} + w_k, \quad (6.35) \]

where \( w_k \) is zero mean white noise with \( 2 \times 2 \) covariance matrix \( R_k \). A final thing to note is that although cameras are generally well suited for classification, this model provides no classification information because the important attribute to classify in this scenario is friend, foe, or neutral, which any hard sensor is ill-suited for classifying.

The other hard sensor type used in this simulation is a stereo camera mounted onboard a ground robot. This sensor once again has a limited range and angular field of view and once again reports its measurements in the local north, east, down coordinate frame. The measurement equation for the stereo camera is

\[ z_k = \eta(x_k) + w_k = \begin{bmatrix} \sqrt{x_k^2 + y_k^2 + z_k^2} \\ \tan^{-1}\left(\frac{y_k}{x_k}\right) \\ \tan^{-1}\left(\frac{z_k}{\sqrt{x_k^2 + y_k^2}}\right) \end{bmatrix} + w_k, \quad (6.36) \]

where \( w_k \) is zero mean white noise with \( 3 \times 3 \) covariance matrix \( R_k \). (Note that in reality the noise in a stereo camera, particularly in range is not zero mean Gaussian, but it is modeled as such here for simplicity.)

For both of the hard sensors, the likelihood is simply given by the Gaussian pdf evaluated at the difference between the actual and expected measurement:

\[ f(z_k|x_k) = \mathcal{N}(z_k - \eta(x_k), R_k), \quad (6.37) \]

where \( \eta(\cdot) \) and \( R_k \) are the appropriate parameters for the type of measurement.

The soft sensors once again report on distance, bearing, speed, and heading the same way as in the previous simulations. Since targets are tracked in three dimensions in this simulation, two new reports are added: height and climb angle. Also, the report
on the target’s class is modified to report on friend, foe, or neutral rather than vehicle type. The likelihood for the soft sensor measurement takes on the exact same form as in Section 6.1.2.

The soft sensor describes the height of the target as “on the ground,” “low,” “medium,” “high,” “very high,” “more or less low,” “more or less high,” “not low,” “not high,” “not very low,” “not very high,” “not more or less low,” “not more or less high,” or “unknown.” The fuzzy membership functions for height take on the same form as the ones for speed described in Section 6.1.2, with “low” replacing “slow,” “high” replacing “fast,” and “on the ground” replacing stationary. They are also obviously scaled to convey the notions of height rather than speed.

The climb angle parameter operates on the angle \( \alpha \) that a target’s trajectory makes with the ground:

\[
\alpha_k = \eta(x_k) = \tan^{-1} \left( \frac{v_{zk}}{\sqrt{v_{xk}^2 + v_{yk}^2}} \right).
\]  

(6.38)

It can take on one of the following values: “down,” “level,” “up,” “very down,” “very up,” “more or less down,” “more or less up,” “not down,” “not up,” “not very down,” “not very up,” “not more or less down,” “not more or less up,” or “unknown.” The fuzzy membership functions for these parameters are formed identically to the fuzzy membership functions for distance described in Section 6.1.2, with “down” replacing “near,” “level” replacing “medium,” and “up” replacing “far.” Also, they are scaled to operate on angles rather than distances and translated so that “level” corresponds to 0 radians, “down” corresponds to negative angles and “up” corresponds to positive angles.

The final parameter returned by the soft sensors classifies the target as “friend,” “foe,” “neutral,” “not a friend,” “not a foe,” “not neutral,” or “unknown.” As with the other classification parameters, no confidences were attached to these classifications. Table 6.7 shows the fuzzy membership functions for these classifications as a function of the actual class.
<table>
<thead>
<tr>
<th>Actual class</th>
<th>Friend</th>
<th>Foe</th>
<th>Neutral</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_{\text{friend}}$</td>
<td>1.0</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>$\mu_{\text{foe}}$</td>
<td>0.01</td>
<td>1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>$\mu_{\text{neutral}}$</td>
<td>0.01</td>
<td>0.01</td>
<td>1.0</td>
</tr>
<tr>
<td>$\mu_{\text{not friend}}$</td>
<td>0.001</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$\mu_{\text{not foe}}$</td>
<td>1.0</td>
<td>0.001</td>
<td>1.0</td>
</tr>
<tr>
<td>$\mu_{\text{not neutral}}$</td>
<td>1.0</td>
<td>1.0</td>
<td>0.001</td>
</tr>
<tr>
<td>$\mu_{\text{unknown}}$</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 6.7: Fuzzy membership function values for friend, foe, and neutral classification

6.3.3 Tracking algorithm

This simulation uses the distributed, hybrid, generalized particle filter (DHGPF) algorithm described in Section 5.4.5. Several changes were made to the basic algorithm. The first difference came in the initialization; the filter is initialized by whichever sensor first sights the target. When the time comes for this sensor to communicate its state to its partners as a Gaussian mixture, the partners will initialize their particle sets by sampling from the Gaussian mixture. If multiple sensors sight the target before the first communication, they will initialize their filters separately; then at the first communication step all of the other partners will initialize their filters from the fusion of both mixtures.

Algorithm 6.2 gives the initialization procedure for the camera and Algorithm 6.3 gives the procedure for the stereo camera. Basically, these algorithms initialize the particles in a modified spherical frame by assuming that the azimuth, elevation, and range (for the stereo camera) are normally distributed about the sensor measurements, that the climb angle is zero mean Gaussian, and that all other parameters are uniformly distributed. The algorithms then convert the particles to the Cartesian frame and add zero mean Gaussian acceleration states. The regimes of the particles are then initialized using Algorithm 6.4. This algorithm essentially chooses a regime at random, then converts the particle’s state to have the correct form for the given regime.

Algorithm 6.5 gives the initialization procedure for the soft sensor. With the soft sensor, it is not possible to simply add noise to the measurements in order to initialize the particles, because the measurements are fuzzy. Instead, the particle set is created by resampling from a uniform particle set using the measurement likelihood as a weight.
Algorithm 6.2 Algorithm to initialize filter from a camera measurement

1: for $i = 1$ to $N$ do
2: Generate the range $\rho_i$ using a uniform random number between 0 and the camera’s maximum range.
3: Generate the azimuth $\varphi_i$ of the particle as follows
   \[ \varphi_i \sim \mathcal{N}(\varphi, \sigma^2_{\varphi}), \]  \hspace{1cm} (6.39)

   where $\varphi$ is the camera azimuth measurement and $\sigma^2_{\varphi}$ is the variance of the measurement.
4: Generate the elevation $\epsilon_i$ of the particle as follows
   \[ \epsilon_i \sim \mathcal{N}(\epsilon, \sigma^2_{\epsilon}), \]  \hspace{1cm} (6.40)

   where $\epsilon$ is the camera elevation measurement and $\sigma^2_{\epsilon}$ is the variance of the measurement.
5: Generate the particle speed $s_i$ uniformly between 0 and the target’s maximum speed.
6: Generate the particle course $\theta_i$ uniformly between 0 and $2\pi$.
7: Generate the particle climb angle $\alpha_i$ as follows
   \[ \alpha_i \sim \mathcal{N}(0, \sigma^2_\alpha), \]  \hspace{1cm} (6.41)

   where $\sigma^2_\alpha$ is the variance in climb angle.
8: Convert the particle state to Cartesian coordinates and translate it into the absolute frame as follows
   \[ x_i = x_{\text{cam}} + \rho_i \cos(\varphi_i) \sin(\epsilon_i) \]  \hspace{1cm} (6.42a)
   \[ y_i = y_{\text{cam}} + \rho_i \sin(\varphi_i) \sin(\epsilon_i) \]  \hspace{1cm} (6.42b)
   \[ z_i = z_{\text{cam}} + \rho_i \cos(\epsilon_i) \]  \hspace{1cm} (6.42c)
   \[ v_{xi} = s_i \cos(\theta_i) \cos(\alpha_i) \]  \hspace{1cm} (6.42d)
   \[ v_{yi} = s_i \sin(\theta_i) \cos(\alpha_i) \]  \hspace{1cm} (6.42e)
   \[ v_{zi} = s_i \sin(\alpha_i), \]  \hspace{1cm} (6.42f)

   where $(x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}})$ is the location of the camera.
9: Initialize the accelerations $(a_{xi}, a_{yi}, a_{zi})$ of the particle as follows:
   \[ a_i \sim \mathcal{N}(0, \sigma^2_a), \]  \hspace{1cm} (6.43)

   where $\sigma^2_a$ is the acceleration variance.
10: end for
11: Initialize the regimes of the particles using Algorithm 6.4.
**Algorithm 6.3** Algorithm to initialize filter from a stereo camera measurement

1: for $i = 1$ to $N$ do
2:   Generate the range $\rho_i$ of the particle as follows
3:     $\rho_i \sim \mathcal{N}(\rho, \sigma^2_\rho)$, \hspace{1cm} (6.44)
4:     where $\rho$ is the camera range measurement and $\sigma^2_\rho$ is the variance of the measurement.
5:   Generate the azimuth $\varphi_i$ of the particle as follows
6:     $\varphi_i \sim \mathcal{N}(\varphi, \sigma^2_\varphi)$, \hspace{1cm} (6.45)
7:     where $\varphi$ is the camera azimuth measurement and $\sigma^2_\varphi$ is the variance of the measurement.
8:   Generate the elevation $\epsilon_i$ of the particle as follows
9:     $\epsilon_i \sim \mathcal{N}(\epsilon, \sigma^2_\epsilon)$, \hspace{1cm} (6.46)
10:    where $\epsilon$ is the camera elevation measurement and $\sigma^2_\epsilon$ is the variance of the measurement.
11:   Generate the particle speed $s_i$ uniformly between 0 and the target’s maximum speed.
12:   Generate the particle course $\theta_i$ uniformly between 0 and $2\pi$.
13:   Generate the particle climb angle $\alpha_i$ as follows
14:     $\alpha_i \sim \mathcal{N}(0, \sigma^2_\alpha)$, \hspace{1cm} (6.47)
15:     where $\sigma^2_\alpha$ is the variance in climb angle.
16:   Convert the particle state to Cartesian coordinates and translate it into the absolute frame using Equation (6.42).
17:   Initialize the accelerations $(a_{xi}, a_{yi}, a_{zi})$ of the particle as follows:
18:     $a_i \sim \mathcal{N}(0, \sigma^2_a)$, \hspace{1cm} (6.48)
19:     where $\sigma^2_a$ is the acceleration variance.
20: end for
21: Initialize the regimes of the particles using Algorithm 6.4.
Algorithm 6.4 Algorithm to initialize the regimes of particles

1: for $i = 1$ to $N$ do
2:   Generate the regime $r_i$ of the particle using the initial regime probability density $p(r_0)$.
3:   if $r_i =$ “Constant velocity” then
4:      Remove the acceleration states from the particle.
5:   else if $r_i =$ “Coordinated turn” then
6:      Calculate the turn rate $\omega_i$ as follows
7:          \[
7:          \omega_i = \frac{v_{xi} \cdot a_{yi} - v_{yi} \cdot a_{xi}}{v_{xi}^2 + v_{yi}^2}.
8:          \]
9:      Eliminate the acceleration states from the particle and add the state for turn rate $\omega_i$.
10:  end if
11: end for

Rather than simply creating a six dimensional particle set to sample from, the dimension corresponding to each measurement is handled separately; then the one dimensional particles are combined to yield a composite particle. This was done to increase the diversity of the particle set: if samples were drawn in six dimensional Cartesian space, there could be 100 copies of the exact same particle drawn, but because each dimension is handled separately, there may be 100 copies of the same range drawn, but each one could be paired with a different bearing to create a unique particle. Once the particles are created in the measurement frame, they are converted to the Cartesian frame, given random accelerations, and converted to the appropriate regimes, just as with the hard sensors.

Another change from the standard DHGPF algorithm comes from the fact that sensors do not receive a measurement at every time step. Hard sensors collect measurements once every second, but only if the target is in view. Soft sensors collect measurements once every ten seconds, but only if the target is in view. Note that unlike the previous simulations, the measurements from the soft sensors need not be at multiples of ten seconds; if a target comes into the soft sensor’s view and it has been more than ten seconds since its last measurement, it will collect a measurement. This obviously more closely resembles how humans would report on a situation, they would likely not
Algorithm 6.5 Algorithm to initialize filter from a soft sensor measurement

1: \textbf{for} distance, bearing, height, speed, heading, and climb angle \textbf{do}
2: \hspace{1em} Generate $N$ one dimensional particles uniformly between the minimum and maximum distance (or bearing, height, etc.).
3: \hspace{1em} Attach a weight $w$ to each distance (or bearing, height, etc.) by evaluating the fuzzy membership function for the soft sensor measurement. (For example, if the distance measurement is “near” and the distance generated in the last step is $d_i$, then $w_i = \mu_{\text{near}}(d_i)$.)
4: \hspace{1em} Resample $N$ new one dimensional particles from the original set.
5: \textbf{end for}
6: \textbf{for} $i = 1$ to $N$ \textbf{do}
7: \hspace{1em} Create a composite particle in the measurement coordinate frame using the one dimensional particles generated in the previous steps: $x_{im} = [d_i \phi_i h_i s_i \theta_i \alpha_i]^T$.
8: \hspace{1em} Convert the particle state to Cartesian coordinates and translate it into the absolute frame as follows:

\begin{align}
    x_i &= x_s + d_i \cos(\phi_i) \\
    y_i &= y_s + d_i \sin(\phi_i) \\
    z_i &= z_s - h_i \\
    v_{xi} &= s_i \cos(\theta_i) \cos(\alpha_i) \\
    v_{yi} &= s_i \sin(\theta_i) \cos(\alpha_i) \\
    v_{zi} &= s_i \sin(\alpha_i),
\end{align}

where $(x_s, y_s, z_s)$ is the location of the soft sensor.
9: \hspace{1em} Initialize the accelerations $(a_{xi}, a_{yi}, a_{zi})$ of the particle as follows:

\begin{equation}
    a_i \sim \mathcal{N}(0, \sigma_a^2),
\end{equation}

where $\sigma_a^2$ is the acceleration variance.
10: \textbf{end for}
11: Initialize the regimes of the particles using Algorithm 6.4.
report at some fixed frequency, they would report as soon as they see something, then wait some time for the situation to change to report again.

Similarly, all sensors communicate at a ten second rate, but only if they have received a measurement in the past ten seconds, this avoids wasting effort communicating particle sets that contain no new information. Just as with the soft sensor measurements, this communication is asynchronous: if a sensor collects a measurement and it has not communicated in ten seconds, it will process the measurement then communicate with its partners. Because soft sensor measurements and all communications can occur asynchronously, the simulation is more difficult to implement and debug, but it makes it a much more faithful representation of what would occur in the real world.

The target regimes used in the DHGPF for this simulation are “constant velocity friend,” “Singer acceleration friend,” “coordinated turn friend,” “constant velocity foe,” “Singer acceleration foe,” “coordinated turn foe,” “constant velocity neutral,” “Singer acceleration neutral,” and “coordinated turn neutral.” These regimes correspond to the three target classes, friend, foe, and neutral, and the three motion model described in Section 6.3.1. The initial probabilities for each class are $\frac{1}{3}$ and the initial probabilities for each motion model are 0.75 for constant velocity, 0.2 for Singer acceleration, and 0.05 for coordinated turn. This gives a composite initial regime probability of

$$p(r_0) = \frac{[0.75 \ 0.2 \ 0.05 \ 0.75 \ 0.2 \ 0.05 \ 0.75 \ 0.2 \ 0.05]}{3}.  \quad (6.52)$$

The matrix of transition probabilities is given as follows

$$\Pi = \begin{bmatrix}
\Pi_1 & \Pi_2 & \Pi_2 \\
\Pi_2 & \Pi_1 & \Pi_2 \\
\Pi_2 & \Pi_2 & \Pi_1
\end{bmatrix},  \quad (6.53)$$

where

$$\Pi_1 = \begin{bmatrix}
0.78 & 0.1 & 0.1 \\
0.2 & 0.58 & 0.2 \\
0.15 & 0.05 & 0.78
\end{bmatrix}.  \quad (6.54)$$
The matrix $\Pi_1$ represents the probability of an object staying in the same class, but switching between motion models within that class. The matrix $\Pi_2$ allows a small number of particles to switch classes. It may not be realistic to assume that a friend can switch to a foe or neutral (although it could occasionally happen), but it does prevent a single incorrect classification from eliminating all of the particles in a certain class.

In general, the labels of “friend,” “foe,” and “neutral” do not have to be attached to particles, because they have no effect on the target motion model. It could have been equally plausible to classify the target by having all of the soft sensors vote on the identity of the target at every time step [181, 182]. The primary reason for attaching labels to particles rather than using a voting method was so that classification and tracking could be handled in one unified framework. A potential area for future work could be to allow the intent of the target to influence its motion. For instance, it is entirely plausible that a foe is more likely to make evasive maneuvers than a friend.

6.3.4 Results

This simulation was tested by running the particle filter 48 times on each of the three trajectories described in Section 6.1. For each run the sensors were placed at the same locations, but the sensor type was random. The noise in the sensor measurements was randomized, with a noise standard deviation of 0.1 radians in azimuth and elevation for both cameras and 15 meters in range for the stereo camera. Additionally, the decisions made by the soft sensors were random. The acceleration process noise for the constant velocity and coordinated turn models was set to $\sigma_a^2 = 1.0$; the process noise for the Singer acceleration model was set to $\sigma_m^2 = 0.2$; and the turn rate variance for the coordinated turn model was set to $\sigma_\omega^2 = 0.2$.

Figure 6.32 shows the normal trajectory from Section 6.1, along with one possible sensor configuration. Note that this trajectory appears rotated from before, because the
tracking in this section was performed in the three dimensional north, east, down (NED) frame; therefore, the ‘x’ coordinate is along the vertical axis and the ‘y’ coordinate is along the horizontal axis. (The ‘z’ axis goes into the page, but it was ignored in these test, because all of the test trajectories were for ground vehicles.)

Fig. 6.32: Normal vehicle trajectory, along with sensor locations with a blue ‘x’ for a UAV, a red ‘*' for a UGV, and a green ‘+' for a soft sensor (all units are meters).

Figure 6.33 shows the position and velocity errors for a single sensing agent along with the one sigma uncertainty contours. (The contours were created by calculating the covariance of the particle set.) These figures are for the first sensor to sight the target on this particular run. The sensor sights the target on the first time step, initializes its particle set, and begins tracking the target. The error and the uncertainty both decrease as the sensor collects more measurements, until the target leaves the sensor’s field of view around 10 seconds into the simulation. At this point, the uncertainty increases and the error drifts as the particle set spreads out in the absence of any measurements. At
15 seconds, a partner sends its data to this sensor. The new data is fused with the local data, and a new particle set is created. This causes the uncertainty and the error to both decrease. This effect is more pronounced in position than velocity; therefore, it is likely that the second sensor has gathered more information about the target’s position, but not its velocity. This same pattern then repeats itself throughout the simulation: uncertainty increases and error drifts, until partner data arrives and decreases the uncertainty.

Figure 6.34a shows the total weights of all particles in the three motion regimes, constant velocity, Singer acceleration, and horizontal turn, for the same sensor. Since this target moves in a straight line, constant velocity should be the most likely mode. This is mostly the case, except at 25 seconds when a message from a partner causes the weight of the horizontal turn mode to jump. As the simulation goes on, in the absence of any measurements indicating that the target is turning, these particles transition back to the constant velocity mode. Figure 6.34b shows the total weight of all particles that are in the classes friend, foe, and neutral. (Note that the actual class of the target was set to friend.) Because hard sensors cannot determine if the target is a friend, foe, or neutral, these weights drift until a soft sensor measurement arrives at 15 seconds. Then, more weight is assigned to the friend particles. At 35 seconds a miss-classification causes neutral to become the most likely class. This problem is corrected at 45 seconds when a correct classification causes friend to again become the most likely class. From that point on, friend stays the dominant class.

Figure 6.35 shows the root mean square (RMS) position and velocity error, along with the one sigma uncertainty contours for this simulation. The results are averaged over all 64 sensors for all 48 runs. Note that in these figures, the scales for the north position and velocity errors are different from all of the other results in this section, due to a huge spike in error 60 seconds into the simulation. It turns out that this spike was caused by a single bad sensor on a single run. Because the RMS error is calculated by squaring the error, a single outlier can have a large influence on the RMS error. Interestingly, at 80 seconds, when a communication likely occurs, the error quickly drops back to acceptable ranges. This shows that tracking system is robust to a single bad sensor.
Fig. 6.33: Single sensor position and velocity errors (blue) along with one sigma contour (dashed red), for normal trajectory with 1000 particles.
Fig. 6.34: Single sensor motion regime and classification weights, for normal trajectory with 1000 particles.
Fig. 6.35: RMS position and velocity errors (blue) along with one sigma contour (dashed red), averaged over 48 runs, for normal trajectory with 1000 particles.
In order to make comparisons with the other runs simpler, Figure 6.36 shows the RMS error for the same test, but with the bad run removed. Note that these figures show an average across all sensors, even the ones that are not currently sensing the target. That is why the uncertainty tends to increase with time as the particles drift due to the process noise. Periodically, the sensor that is viewing the target will broadcast a message and the errors and uncertainty will once again drop as this data is fused with the local data. Figure 6.37 shows the weights of the particles in each motion regime and each class. As should be expected, constant velocity is the dominant mode throughout the simulation. Also, friend is very quickly established as the dominant classification, as it should be.

Figure 6.38 shows the maneuver trajectory, along with a possible sensor configuration. Figure 6.39 shows the RMS position and velocity error for the maneuver trajectory. As was the case in Section 6.1, the tracking error increases at 40 seconds when the target makes its maneuver. Note that this issue is even more pronounced here, because most of the sensors are not viewing the target when the maneuver occurs, so their filters continue to assume that the target is moving in a straight line. As the sensors exchange messages, the local filters are corrected to account for the maneuver and the error returns to the pre-maneuver levels. Figure 6.40a shows the average weights assigned to each mode for this simulation. Note that there is not a significant increase in the weight for the maneuver mode at 40 seconds, because the average sensor does not know that the target is maneuvering. There are likely one or two sensors viewing the target during the maneuver, whose maneuver weights do increase, but they are averaged out. Figure 6.40b shows the average number of particles in each class for this simulation. The class was set to neutral, which was indeed the dominant classification.

Figure 6.41 shows the erratic trajectory, along with a possible sensor configuration. Note that this time the tracker does not have an explicit regime called “erratic,” rather it was hypothesized that the addition of the Singer acceleration mode would be sufficient to account for any sudden maneuvers by the target. Figure 6.42 shows the RMS position and velocity errors for the erratic trajectory. As was the case before, this is a difficult target to track, due to the aggressive maneuvers it makes. The important thing to note
Fig. 6.36: RMS position and velocity errors (blue) along with one sigma contour (dashed red) for normal trajectory with 1000 particles, with outlier removed.
Fig. 6.37: Average motion regime and classification weights, averaged over 48 runs, for normal trajectory with 1000 particles.
Fig. 6.38: Maneuver vehicle trajectory, along with sensor locations with a blue ‘x’ for a UAV, a red ‘*’ for a UGV, and a green ‘+’ for a soft sensor (all units are meters).
Fig. 6.39: RMS position and velocity errors (blue) along with one sigma contour (dashed red), averaged over 48 runs, for maneuver trajectory with 1000 particles.
Fig. 6.40: Average motion regime and classification weights, averaged over 48 runs, for maneuver trajectory with 1000 particles.
is that, although the errors do grow quite large, they nearly always stay within the one sigma uncertainty contour, so the track is never lost. Also, as has been the case with all of the trajectories, fusion with partner data results in drops in both the error and the uncertainty. Figure 6.43a shows the average weight of each regime; as has been the case, not much can be gleaned from this figure because averaging over 64 sensors and 48 runs smooths out any interesting regime changes. Figure 6.43b shows the average weight of each classification. The actual class of the target was set to foe, which indeed becomes the dominant classification within 25 seconds.

Table 6.8 shows the average (i.e. averaged over all time) RMS north and east position and velocity errors for all three trajectories for both 500 and 1000 particles. The normal trajectory with 1000 particles is shown with both the original results and with the outlier removed. Generally the results in this table are fairly intuitive: error
Fig. 6.42: RMS position and velocity errors (blue) along with one sigma contour (dashed red), averaged over 48 runs, for erratic trajectory with 1000 particles.
Fig. 6.43: Average motion regime and classification weights, averaged over 48 runs, for erratic trajectory with 1000 particles.
tends to decrease with an increase in particles; and erratic errors are highest, followed by maneuver and normal (with the outlier removed). Finally, note that this table should not be compared with the results from the other simulations, because even though it used the same trajectories, the fact that different sensors with a limited field of view were used makes the tracking problem very different.

<table>
<thead>
<tr>
<th>Trajectory</th>
<th>Particles</th>
<th>RMSE N</th>
<th>RMSE E</th>
<th>RMSE VN</th>
<th>RMSE VE</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>500</td>
<td>47.5</td>
<td>39.8</td>
<td>10.1</td>
<td>9.1</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>101.7</td>
<td>37.2</td>
<td>23.1</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>1000(^1)</td>
<td>44.8</td>
<td>33.2</td>
<td>8.6</td>
<td>4.6</td>
</tr>
<tr>
<td>maneuver</td>
<td>500</td>
<td>65.0</td>
<td>121.</td>
<td>28.0</td>
<td>57.0</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>44.0</td>
<td>36.1</td>
<td>8.1</td>
<td>6.0</td>
</tr>
<tr>
<td>erratic</td>
<td>500</td>
<td>88.1</td>
<td>93.0</td>
<td>30.9</td>
<td>27.5</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>58.3</td>
<td>54.5</td>
<td>10.7</td>
<td>8.2</td>
</tr>
</tbody>
</table>

Table 6.8: RMS errors in north and east position and velocity for simulation 3, averaged over 48 runs, units are meters and meters per second.

## 6.4 Conclusions

This chapter presented results from three different simulations involving hard and soft sensor data fusion. Many important things can be observed from the results of these simulations. The first is that the hard sensor alone does a very good job of tracking, even when a target can make erratic maneuvers, as in Section 6.1.5.1. Likewise, a soft sensor alone does a very poor job of target tracking, as was seen in Section 6.1.5.2; this is consistent with general intuition that people are not good at precisely measuring distances or speeds. Section 6.1.5.3 showed that the generalized particle filter algorithm presented in Section 5.4.1 does provide a consistent means for fusing hard and soft data. In the case of this particular simulation it turned out that the addition of a soft sensor did not significantly improve tracking results except when the number of particles was low.

The second simulation introduced the problem of multiple possible target classes. Because the first simulation showed that tracking performance was similar for both

\(^1\) outlier removed
the hard-only case and the hard and soft case when the number of particles was high, the second simulation focused entirely on the combined hard and soft case. The first thing to note from the second simulation was the error when tracking a target was roughly proportional to its maneuvering capability; this corresponds well with intuition. The second was that with the addition of the soft sensor, the target was nearly always successfully classified (except for the case where the normal person was often mistaken for a lost person); this also corresponds well with intuition.

Whereas the focus of the first two simulations was to study the fusion of hard and soft data in various situations, the purpose of the third simulation was to show that the distributed, hybrid, generalized particle filter (DHGPF) introduced in Section 5.4.5 was ready to be fielded in a real world system. Therefore, this simulation utilized a swarm of unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), and soft sensors to track a target. The important aspects of this simulation where not necessarily the magnitude of the tracking errors or the classification performance, but that each sensor consistently processed local sensor information and fused that information with information from partners. The fact that the sensors in this simulation did exactly that shows that the DHGPF is ready for field testing. Field test results using the DHGPF are presented in the following chapter.
Chapter 7

Hard and soft sensor fusion hardware demonstration

This chapter builds on the work done in previous chapters by performing a field test of the algorithms that were described in Chapter 5 and tested in simulation in Chapter 6. The test utilized an unmanned aerial vehicle, an unmanned ground vehicle, and a human observer with a laptop. The test was meant to emulate the search and rescue scenario described in the first chapter. The test proceeded as follows:

1. The human observer input into his GUI that there was a friendly person (in this case a missing pilot) an unknown distance away to the northeast. This data was used to initialize a track onboard the laptop, which was then sent to all partners.

2. The UAV was commanded to fly a search pattern in the area where the missing pilot was likely to be.

3. When the UAV found the target (in this case a blue tarp, which represented the pilot), it fused the data it collected about the target with the data from the human observer and sent an updated track to all of the partners.

4. When the UGV received the updated target coordinates from the UAV, it traveled to those coordinates in order to confirm the detection.

5. When the UGV detected the target, it updated the track with its own sensor data and sent an update to all partners.

The following sections describe in detail how this test was carried out. The first section describes how the distributed, hybrid, generalized particle filter (DHGPF), developed in Chapter 5, was applied to this problem. The next section details how the test was carried out in hardware. The final section presents the results from the test.
7.1 Fusion algorithm

In this demonstration, the missing person was tracked using the distributed, hybrid, generalized particle filter (DHGPF) described in Section 5.4.5. As in Section 6.3.3, several modifications were made to the basic algorithm to facilitate practical implementation. The first is that the tracker was initialized by whichever sensor first saw the target. The second is that measurements and communications did not occur synchronously. Measurements occurred whenever the target was spotted by a sensor and communications occurred at fixed intervals, but only if a measurement had occurred during that time period. The following two sections describe the specific motion and measurement models used by the DHGPF to track to missing person.

7.1.1 Motion models

Recall the the purpose of this demonstration was to mimic finding a missing person; therefore, the motion model should mimic the motion of a person on foot. In Section 6.2.1 two motion models were developed for a person on foot: one for a person walking normally and one for a lost person. These motion models were found to be inadequate for this demonstration because they both assumed that the person was constantly moving. This would be a poor assumption if a person was injured, or if he chose to stay in one place to increase his likelihood of being found. In order to account for the fact that a person could be stationary or moving, two motion models were used: a constant position model and a dynamic model.

**Constant position model** The constant position model (CP) model has three states: north position, east position, and down position. The model has no dynamics, the update equation is simply

\[
x_{k+1} = x_k \hspace{1cm} (7.1a)
\]

\[
y_{k+1} = y_k \hspace{1cm} (7.1b)
\]

\[
z_{k+1} = z_k. \hspace{1cm} (7.1c)
\]
Normally, using this state equation in a particle filter could result in sample impoverishment after resampling is carried out (as in Section 2.6.4.2), but because a multiple model particle filter was used, constant position particles could transition to dynamic particles, which added diversity to the particle set. Also, regularized resampling was used to further add diversity to the particle set.

**Dynamic model** The person dynamic model created in Section 6.2.1 was found to be inadequate for this demonstration for several reasons. The first was that one of the states was a heading, which is not directly observable. The second was that it did not account for vertical position or velocity (this is important when a target can go up or down a hill). The third, and most important reason, was that it assumed that the velocity had white noise added to it at every time step; this can produce unreasonably large velocities if the target is not viewed for a long period of time (as would be the case if a UAV viewed it once per pass). To deal with these issues, an Integrated Ornstein-Uhlenbeck (IOU) process, which allows for bounding the variance in velocity over time, was used \[6, 216\]. The IOU model has six states: north position, east position, down position, north velocity, east velocity, and down velocity. The state update is given by a linear equation of the form

\[
x_{k+1} = Fx_k + Gv_k.
\]  
(7.2)

The state transition matrix \(F\) is given by:

\[
F = \begin{bmatrix}
1 & 0 & 0 & T & 0 & 0 \\
0 & 1 & 0 & 0 & T & 0 \\
0 & 0 & 1 & 0 & 0 & T \\
0 & 0 & 0 & F_v & 0 & 0 \\
0 & 0 & 0 & 0 & F_v & 0 \\
0 & 0 & 0 & 0 & 0 & F_v \\
\end{bmatrix},
\]  
(7.3)

where \(T\) is the sampling time and

\[
F_v = e^{-T\gamma}.
\]  
(7.4)
The input matrix $G$ is given by:

$$G = \begin{bmatrix}
T^2/2 & 0 & 0 \\
0 & T^2/2 & 0 \\
0 & 0 & T^2/2 \\
\frac{1-F_k}{\gamma} & 0 & 0 \\
0 & \frac{1-F_k}{\gamma} & 0 \\
0 & 0 & \frac{1-F_k}{\gamma}
\end{bmatrix}.$$  \hspace{1cm} (7.5)

The noise parameter, $v_k$ is zero mean white noise with covariance matrix $Q$. The important aspect of this model is the parameter $\gamma$, which controls how new velocities are affected by previous velocities. As gamma approaches zero the velocity approaches the constant velocity model (Equation(6.1)), where the velocity process can grow unbounded. As gamma is increased, current velocities have less influence on future velocities, which keeps the velocity process bounded.

**Transition probabilities** In order to model the fact that a moving target may stop and stationary target may start moving, a hybrid particle filter was used for tracking. The discrete component of the state also accounted for the fact that a target may be a friend, foe, or neutral. This gave a total of six possible modes: CP Friend, IOU Friend, CP Foe, IOU Foe, CP Neutral, and IOU Neutral. Initially, each mode was set to be equally likely. The matrix of transition probabilities was constructed such that there was a moderately small (on the order of 0.04) probability that a CP particle could transition to IOU (or the other way around) and a very small probability (on the order of 0.004) that a Friend particle could transition to Foe or Neutral. The first probability accounted for the fact that a person may transition from stationary to moving; the second acted as a forgetting factor to reduce the influence of miss-classifications over long periods of time. These values were increased or decreased slightly depending on the sampling time.
7.1.2 Sensor processing

Two different measurement models were used to process sensor data: a hard sensor model that was used by the UAV and UGV, and a soft sensor model that was used by the human operator. Also, because the sensor that first viewed a target initialized the tracker from its measurement, each sensor had a unique initialization algorithm. Each of these measurement models and initialization procedures is described below.

**Hard sensor measurement model**  Both the UAV and UGV that were used in this test carried a webcam. The target (a blue tarp) was detected using the crisp classifier described in Section 2.5.1. Classifications were then transformed to bearing measurements using the procedure described in Section 2.5.2. Then, from this measurement, Equation (6.37) was used to evaluate the likelihood of the measurement given each particle.

**UGV sensor initialization**  Just as in Section 6.3.3, whichever sensor first viewed the target initialized the track, then communicated the track to its partners. When the UGV was the first sensor to view the target, it used Algorithm 7.1 to initialize the track. The algorithm first chooses a range to lie uniformly between zero and the maximum range, then it generates an azimuth and elevation by adding noise to the camera measurements. These values are then converted to Cartesian coordinates. The particle regimes are then initialized using Algorithm 7.2. This algorithm essentially randomly assigns a regime to each particle, then adds three extra velocity states (with normally distributed velocities) if the particle is in the IOU regime.

**UAV sensor initialization**  Algorithm 7.3 shows how the filter was initialized when the UAV was the first sensor to view the target. It is nearly identical to the UGV initialization procedure, except in how range is handled. Whereas the UGV has no information about the range to a target, the UAV can estimate the range to the target based on the current height of the UAV. This is accomplished by first generating the noise corrupted height estimate using Equation (7.12); then, the range is estimated from
Algorithm 7.1 Algorithm to initialize filter from a UGV camera measurement

1: for $i = 1$ to $N$ do
2:   Generate the range $\rho_i$ using a uniform random number between 0 and the camera’s maximum range.
3:   Generate the azimuth $\varphi_i$ of the particle as follows
   \[ \varphi_i \sim \mathcal{N}(\varphi, \sigma^2_\varphi), \] (7.6)
   where $\varphi$ is the camera azimuth measurement and $\sigma^2_\varphi$ is the variance of the measurement.
4:   Generate the elevation $\epsilon_i$ of the particle as follows
   \[ \epsilon_i \sim \mathcal{N}(\epsilon, \sigma^2_\epsilon), \] (7.7)
   where $\epsilon$ is the camera elevation measurement and $\sigma^2_\epsilon$ is the variance of the measurement.
5:   Convert the particle state to Cartesian coordinates and translate it into the absolute frame as follows
   \[ x_i = x_{\text{cam}} + \rho_i \cos(\varphi_i) \sin(\epsilon_i) \] (7.8a)
   \[ y_i = y_{\text{cam}} + \rho_i \sin(\varphi_i) \sin(\epsilon_i) \] (7.8b)
   \[ z_i = z_{\text{cam}} + \rho_i \cos(\epsilon_i), \] (7.8c)
   where $(x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}})$ is the location of the camera.
6: end for
7: Initialize the regimes of the particles using Algorithm 7.2.

Algorithm 7.2 Algorithm to initialize the regimes of particles

1: for $i = 1$ to $N$ do
2:   Generate the regime $r_i$ of the particle using the initial regime probability density $p(r_0)$.
3:   if $r_i = “\text{IOU}”$ then
4:     Initialize the velocities $(v_{xi}, v_{yi}, v_{zi})$ of the particle as follows:
     \[ v_{xi} \sim \mathcal{N}(0, \sigma^2_{vx}) \] (7.9a)
     \[ v_{yi} \sim \mathcal{N}(0, \sigma^2_{vy}) \] (7.9b)
     \[ v_{zi} \sim \mathcal{N}(0, \sigma^2_{vz}) \] (7.9c)
     where $\sigma^2_{ax,y,z}$ is the acceleration variance in each direction.
5:   end if
6: end for
the height and elevation using Equation (7.13). From this point on, the initialization procedure is identical to that for the UGV.

**Soft sensor measurement model** Data from human operators was gathered using the GUI developed in Section 4.2. Recall that the GUI allowed users to enter the class of an item (person, vehicle, building, or intersection), with an associated confidence; whether the item was a friend, foe or neutral, once again with a confidence; the bearing to the item; and the distance to the item. In this situation the target was always considered to be a person, so the classification was not used, but all of the other inputs were used. Once the user entered data into the system, the likelihoods for each particle were evaluated using the soft sensor measurement model developed in Section 6.3.2. This time, however, the soft sensor could attach confidences to its classifications. Another difference was that the soft sensor did not input measurements for speed, heading, height, or climb angle (because the target was assumed be on the ground and moving slowly); therefore, these terms were dropped from the likelihood calculation.

**Soft sensor initialization** When the human user viewed the target first, the track was initialized using Algorithm 7.4. This algorithm generates the continuous states of the particles by sampling from a set of possible ranges and bearings according to the likelihood that they generated the soft sensor measurement (similar to Algorithm 6.5 in the previous chapter). Algorithm 7.5 likewise generates the particle regimes by creating a set of regimes according the the initial regime probabilities and sampling based on the classification likelihood. Finally, velocity states are added for particles in the IOU regimes.

### 7.2 Test setup

Testing of the hard sensor, soft sensor fusion system was carried out at the flying field described in Section 2.6.2. The UAV used was the same SIG Kadet Senior UAV that was used in Section 2.6.2. A new ground vehicle was used for this portion of the test, a MMP-5 mobile robot from The Machine Lab [217]. Figure 7.1 shows the robot as it was
Algorithm 7.3 Algorithm to initialize filter from a UAV camera measurement

1: for $i = 1$ to $N$ do
2:   Generate the azimuth $\varphi_i$ of the particle as follows
3:     $$\varphi_i \sim \mathcal{N}(\varphi, \sigma^2_\varphi),$$ \hspace{1cm} (7.10)
4:     where $\varphi$ is the camera azimuth measurement and $\sigma^2_\varphi$ is the variance of the measurement.
5:   Generate the elevation $\epsilon_i$ of the particle as follows
6:     $$\epsilon_i \sim \mathcal{N}(\epsilon, \sigma^2_\epsilon),$$ \hspace{1cm} (7.11)
7:     where $\epsilon$ is the camera elevation measurement and $\sigma^2_\epsilon$ is the variance of the measurement.
8:   Generate the noise corrupted UAV height:
9:     $$h_i \sim \mathcal{N}(h_{\text{UAV}}, \sigma^2_h),$$ \hspace{1cm} (7.12)
10:    where $h_{\text{UAV}}$ is the autopilot estimate of the UAV’s height and $\sigma^2_h$ is the variance in this estimate.
11:   Calculate the range to the particle as follows:
12:     $$\rho_i = \frac{h_i}{\cos(\epsilon_i)}.$$ \hspace{1cm} (7.13)
13:   Convert the particle state to Cartesian coordinates and translate it into the absolute frame using Equation (7.8).
14: end for
15: Initialize the regimes of the particles using Algorithm 7.2.
Algorithm 7.4 Algorithm to initialize filter from data entered into the GUI by a human operator

1: for distance, bearing do
2: Generate $N$ one dimensional particles uniformly between the minimum and maximum distance (or between 0 and $2\pi$ for bearing).
3: Attach a weight $w$ to each distance (or bearing) by evaluating the fuzzy membership function for the soft sensor measurement. (For example, if the distance measurement is “near” and the distance generated in the last step is $\rho_i$, then $w_i = \mu_{\text{near}}(\rho_i)$.)
4: Resample $N$ new one dimensional particles from the original set.
5: end for
6: for $i = 1$ to $N$ do
7: Generate a random elevation $\epsilon_i$:
\[
\epsilon_i \sim \mathcal{N}(0, \sigma^2_{\epsilon}),
\]
where $\sigma^2_{\epsilon}$ is the the variance of the elevation, which corresponds roughly to the variability of the terrain.
8: Create a composite particle in the measurement coordinate frame using the one dimensional particles generated in the previous steps and the elevation: $x_{im} = [\rho_i \varphi_i \epsilon_i]^T$.
9: Convert the particle state to Cartesian coordinates and translate it into the absolute frame using Equation (7.8).
10: Initialize the regimes of the particles using Algorithm 7.5.
11: end for

Algorithm 7.5 Algorithm to initialize the regimes of particles from a soft sensor measurement

1: Generate the $N$ regimes $\{r_i\}$ using the initial regime probability density $p(r_0)$.
2: Weight each regime by the likelihood of the soft sensor measurement (i.e. Friend, Foe, Neutral) given that regime.
3: Resample $N$ new regimes from the weighted set of regimes
4: for $i = 1$ to $N$ do
5: Assign regime $r_i$ to particle $x_i$.
6: if $r_i$ = “IOU” then
7: Initialize the velocities $(v_{xi}, v_{yi}, v_{zi})$ of the particle as follows:
\[
\begin{align*}
v_{xi} & \sim \mathcal{N}(0, \sigma^2_{vx}) \quad (7.15a) \\
v_{yi} & \sim \mathcal{N}(0, \sigma^2_{vy}) \quad (7.15b) \\
v_{zi} & \sim \mathcal{N}(0, \sigma^2_{vz}) \quad (7.15c)
\end{align*}
\]
where $\sigma^2_{ax,y,z}$ is the acceleration variance in each direction.
8: end if
9: end for
set up for this test. The MMP-5 carried the same single board computer, compass, and
GPS as the Tankbot (see Section 3.4). The Penn State University Applied Research Lab
Intelligent Controller (IC) program [67, 190, 191] provided vehicle control. The MMP-5
also carried a Logitech Webcam that was not present on the Tankbot.

Data was collected from the human user (soft sensor) using the GUI that was
developed in Section 4.2 for soft sensor only fusion. This time, however, instead of data
going to the fuzzy logic system, it was input to the same particle filter that ran on the
vehicles. Both of the vehicles and the laptop used by the soft sensor communicated
through an ad hoc wireless network. All of the sensor processing, filtering, and data
fusion was carried out onboard each local machine, there was no centralized fusion center.

7.3 Test results

The fusion system was tested through four experiments. The first test, which
evaluated local hard sensor data fusion, involved a single UGV. The second test involved
a single soft sensor. Next, distributed fusion was tested using a UGV and a soft sensor.
Finally, the search and rescue demonstration described at the start of the chapter was carried out using a UAV, a UGV, and a soft sensor.

### 7.3.1 UGV only

The first field test of the fusion system involved a single UGV. The UGV was placed a distance away from the target (a blue tarp) and commanded to go to a waypoint near the tarp. As it proceeded to the waypoint, the camera collected images, which were processed to find the blue tarp. Detections were fused with data from the compass and GPS to determine the location of the tarp. Figure 7.2 shows an image taken by the UGV where the tarp was successfully detected. The target location was estimated by taking the weighted mean of the particles in the filter; the covariance was also calculated in order to determine the spread of the particles. Figure 7.3 shows the north and east components of the position error. Note that even though several images of the target were taken, the errors remained relatively high (on the order of 25 meters), but the errors were within two standard deviations, thus the filter was not overconfident in the poor estimate. The reason that the errors remained so high was likely that all of the images of the target were taken from similar angles, so the range to the target was not observable. Figure 7.4 shows the total weight assigned to the constant position (CP) and Integrated Ornstein-Uhlenbeck (IOU) modes. Even though the target was stationary, IOU was the dominant mode, which was once again likely due to the fact that the position estimate was quite poor.

### 7.3.2 Soft sensor only

The second test of the fusion system involved a single human soft sensor, who tested the system by viewing the target from three different distances and angles. The steps in the test were as follows:

1. Initially the user entered, “a person, who is definitely a friend, is an unknown distance to the north.”
2. Next, the user moved north and east of the original position and entered, “there is a person, who has a medium confidence of being a friend, not near to the west.”

3. Finally, the user moved north and west and entered, “there is a person, who is definitely a friend, near to the south.”

Figure 7.5 shows the north and east components of the position error for this test. Note that because the target is viewed from three very different distances and angles, the error becomes very low (on the order of five meters), with a decrease in uncertainty as well. Figure 7.6a shows the weights assigned to the CP and IOU modes. Note that this time, the weights were much closer to fifty percent. Figure 7.6b shows the weights for each class. Friend was the dominant classification throughout the test, as it should have been. Note that the weight for friend decreased during the time period when no measurements were collected. The reason for this is that a small number of particles change from friend to foe or neutral at each time step. This acts as a forgetting factor to prevent a miss-classification from influencing the result for a long period of time.
Fig. 7.3: Errors in the north and east direction for test with UGV only

(a) North component of error

(b) East component of error
7.3.3 Combined soft sensor and UGV

The first distributed test involved a soft sensor and a UGV. The soft sensor initially entered, “there is a person, who is definitely a friend, an unknown distance to the northeast.” This data was used to initialize the soft sensor’s tracker and was also sent to the UGV, which used it to initialize its own tracker. The UGV was then ordered to go to a waypoint that was near the target location. As the UGV took images of the target, it used this information to update its track and periodically sent data to the soft sensor, which fused the data with its own track. (The soft sensor did not collect any new data after the initial report.)

Figure 7.7 shows the north and east position errors for the soft sensor in this test. The track was first initialized using the original soft sensor report. After thirty seconds the soft sensor received its first update from the UGV. From this point on, the UGV sent updates to the soft sensor every ten seconds. Figure 7.8a shows the weight of the two motion regimes for the soft sensor. The two regimes had nearly equal weight, except for
Fig. 7.5: Errors in the north and east direction for test with soft sensor only
Fig. 7.6: Weights for motion regimes and classifications for the soft sensor only

(a) Weights of the CP and IOU regimes

(b) Weights for friend, foe and neutral classifications
at the last update time, when IOU became the dominant regime. Figure 7.8b shows the weight of each classification for the soft sensor. Friend stayed the dominant classification throughout the test.

Figure 7.9 shows the north and east position error for the UGV for this test. The UGV’s track was initialized 10 seconds into the test, when it first received a message from the soft sensor. Then at 25 seconds the target was spotted for the first time, and the UGV fused the detection with the data from the soft sensor in order to update the track. For both the soft sensor and the UGV, the position errors and the uncertainty both decreased as more data was collected by the UGV. Figure 7.10a shows the weight assigned to each motion regime for the UGV. Similar to the soft sensor, the two regimes were nearly equally likely until 50 seconds, when IOU began to dominate. Figure 7.10b shows the weight for each classification for the UGV. Here an interesting result occurred: at 55 seconds neutral became the dominant classification instead of friend. Because the UGV did not collect any information about the classification of the target, it weighted particles based only on how well their position and velocity matched its measurements. Therefore, it is likely that a number of neutral particles matched the measurement collected at 55 seconds very well, which made them more likely to be resampled than the friend particles. This lead to neutral incorrectly becoming the dominant classification.

7.3.4 Flight test

The final test of the fusion system was the collaborative search and rescue problem described at the beginning of the chapter and in the Introduction. The test started with the soft sensor entering, “there is a person, who is definitely a friend, an unknown distance to the northeast,” into the GUI. Next, the UAV flew a search pattern over the target area in order to better localize the target. Figure 7.11 is an image of the target collected by the UAV. Once the UAV detected the target from the air, it sent an update to the soft sensor and the UGV. When the UGV received this information, it proceeded to the estimated target location in order to verify the detection. As the UAV made more passes over the target and the UGV detected the target from the ground, they processed local sensor data and sent updates to each other and to the soft sensor.
Fig. 7.7: Errors in the north and east direction for the soft sensor in the combined UGV and soft sensor test.
Fig. 7.8: Weights for motion regimes and classifications for the soft sensor in the combined UGV and soft sensor test
Fig. 7.9: Errors in the north and east direction for the UGV in the combined UGV and soft sensor test
Fig. 7.10: Weights for motion regimes and classifications for the UGV in the combined UGV and soft sensor test

(a) Weights of the CP and IOU regimes

(b) Weights for friend, foe and neutral classifications

---

(a) Weights of the CP and IOU regimes

(b) Weights for friend, foe and neutral classifications

Fig. 7.10: Weights for motion regimes and classifications for the UGV in the combined UGV and soft sensor test
North and east position errors are shown for the soft sensor in Figure 7.12, the UAV in Figure 7.13, and the UGV in Figure 7.14. For all three sensing platforms, errors were relatively high (on the order of 50 meters) throughout the test. The uncertainty was also very high, which meant that the estimates stayed within two standard deviations of the actual location. The high errors (and high uncertainty) were likely due to the fact most of the measurements were taken by the UAV, which collected very noisy measurements due to pitching, rolling, and wind noise. More important than the magnitude of the errors was the fact that both vehicles and the soft sensor all collected and processed local data, which was fused with data from the partners.

Figure 7.15 shows the weight of each motion regime and each classification for the soft sensor, Figure 7.16 shows the weights for the UAV, and Figure 7.17 shows them for the UGV. Note in these figures that for all three platforms the two motion regime weights were nearly equal. Also, for all three platforms friend was the dominant classification, as it should have been.
Fig. 7.12: Errors in the north and east direction for the soft sensor in the flight test
Fig. 7.13: Errors in the north and east direction for the UAV in the flight test

(a) North component of error

(b) East component of error
Fig. 7.14: Errors in the north and east direction for the UGV in the flight test.
Fig. 7.15: Weights for motion regimes and classifications for the soft sensor in the flight test

(a) Weights of the CP and IOU regimes

(b) Weights for friend, foe and neutral classifications
Fig. 7.16: Weights for motion regimes and classifications for the UAV in the flight test
Fig. 7.17: Weights for motion regimes and classifications for the UGV in the flight test.
7.4 Conclusions

This chapter presented several tests of the DHGPF onboard unmanned vehicles and a laptop carried by a human soft sensor. The tests involved a single UGV, a single soft sensor, a UGV and a soft sensor, and a UAV, UGV, and soft sensor. The tests were meant to mimic finding a missing person using a team of humans and robots. These tests demonstrated that the DHGPF is capable of running in real time onboard relatively simple hardware. There have been several attempts to distribute a particle filter across multiple platforms, including mobile robots, but these tests were truly novel because of the addition of data from a soft sensor, which took on a fuzzy rather than statistical form. Few, if any, real world tests prior to this have demonstrated fusion of data collected by mobile robots with data collected by a human user in real time using random set theory and a particle filter.
Chapter 8

Conclusions

This research has addressed many aspects of the data fusion problem, especially as it applies to autonomous agents, whether they be human beings or robots. The first chapter established the importance of data fusion in general, and showed how it was particularly relevant to robotics problems. It then discussed the importance of research in fusion of data from both “hard” physics based sensors and “soft” human sensors.

The second chapter reviewed many of the important traditional approaches to data fusion, including data association, classification, tracking, and robotics problems. The chapter concluded with a real world application for data fusion involving unmanned aerial vehicle (UAV) surveillance. Algorithms used in this demonstration included classification, target localization, tracking, and data association. Both simulation and flight test results of the algorithms were presented.

The third chapter introduced the problem of distributed data fusion. It reviewed research in both distributed tracking and classification. The final section of the chapter discussed how distributed fusion could be utilized in a search and rescue mission involving multiple UAVs and unmanned ground vehicles (UGVs). It then presented an initial proof of concept experiment in UAV and UGV collaboration.

The past two chapters having dealt with traditional, physics based “hard” sensors, the fourth chapter introduced the problem of “soft” sensor data fusion. The chapter started with a review of the major research in soft sensor data fusion. Then, it presented a novel fuzzy-logic based method for fusing data from multiple human users and a graphical user interface (GUI) for collecting the data. A simulated search and rescue mission was designed in order to test the fuzzy logic system. The system was tested both in the lab and in the field by users entering data relevant to the search and rescue problem in order to locate a missing pilot while avoiding hostile forces.
Having dealt with both hard and soft sensors separately, the fifth chapter dealt with the problem of fusing data coming from both hard and soft sensors. The chapter began with a survey of the literature in hard sensor and soft sensor data fusion. Then, random set theory was presented as a theoretical approach to fusing hard and soft data. The chapter concluded with a discussion of a new method for fusing hard and soft data over a distributed network: the distributed, hybrid, generalized particle filter (DHGPF) algorithm. The DHGPF was based on previous research on distributed particle filtering, but because the measurement update step was modified to use generalized random set likelihoods, the filter could be used for both hard and soft data.

Having laid the theoretical basis for hard and soft fusion, the next chapter described practical applications for hard and soft sensor data fusion. Through a series of three progressively more difficult simulations, some important hard and soft sensor data fusion capabilities were demonstrated. The first simulation demonstrated fusing data from a single soft sensor and a single hard sensor in order to track a car that could be driving normally or erratically. The second simulation added the extra complication of classifying the type of vehicle to the simulation. The third simulation was designed to test the DHGPF algorithm by using multiple hard and soft sensors, with a limited field of view, to track a moving target and classify it as a friend, foe, or neutral.

The final chapter built on the work done in previous chapters by performing a field test involving hard and soft sensor data fusion. The test utilized a UAV, a UGV, and a human observer with a laptop. The test was designed to mimic searching for and rescuing a missing person using a team of robots and human searchers. This chapter made some of the most important contributions in this thesis, as it was one of the first, if not the first, demonstrations of distributed hard and soft sensor data fusion using a particle filter. Not only is this of theoretical importance, it is of practical importance because it showed that the algorithms that have been developed for hard and soft sensor data fusion in this thesis are capable of running in real time on relatively simple hardware.
8.1 Future work

Though this thesis has covered much ground, there is still a great deal of research to be done in distributed hard and soft sensor data fusion. The first area for future research is in tracking moving targets from a UAV using a camera. All of the aerial tracking results presented in the second chapter dealt with static targets, because of the difficulty of finding a flying area where it was safe to drive a vehicle across the runway. The extended Kalman filter running on the UAV, however, was designed for moving targets, so it would be a simple extension to test it on moving targets if a safe method for doing so could be devised.

The second area for future research is in multiple vehicle collaboration. Because the UGV used in the experiment in the third chapter was not equipped with a camera, it had to rely completely on the UAV to guide it to the target. In the final chapter the UGV was equipped with a camera and was able to use its own sensor data to improve its estimate of the target location, but because the controller did not use this sensor data, the vehicle still relied on the UAV to guide it to the target. If the traverse action in the Intelligent Controller were modified, the UGV could get near the target using the report from the UAV, then once it sighted the target, it could use its camera to drive right up to it.

In the area of soft sensor data fusion, the fuzzy logic system so far has only been tested using two users. It should easily scale to more than two users, but it would be interesting to test it with many users to see if there is a point where it stops scaling well. Also, the GUI that was used to collect data from the users, was fairly rudimentary; it would be much more powerful if someone with expertise in user interfaces could design a user-friendly GUI for collecting data from human users. Another limitation of the system was the fact that there is currently no way to handle uncertainty in the uncertainty, i.e. type-two uncertainty. One possible extension to the current system could be to use fuzzy Dempster-Shafer theory [218,219]; this would allow the user to say things such as, “I am very certain the target is near” or “I am somewhat certain that the target is more or less far, but there is a slight possibility that it is medium distance.” It could also be extended by using type-2 fuzzy logic [220,221]; this would allow the fuzzy membership functions
for concepts such as “near” or “far” to themselves be uncertain. Doing so would be a
very powerful extension, because different people may have different notions of what is
near versus medium or far. Another possible way to handle data from soft sensors would
be to use subjective probabilities. Using probability would allow soft data to be fused
more easily with hard data, but in general the probability distributions that describe soft
sensor data are not known. The best approach to determining the probabilities would
be to conduct a test with a large number of people in order to generate training data for
the probability model.

Another area for future work is in hard and soft sensor data fusion. Just as with
soft only fusion, the addition of a mechanism to handle type-two uncertainty in the soft
center measurements would be useful. Using fuzzy Dempster-Shafer theory to do so
would be well suited to fusion using random set theory [5]. Type-2 fuzzy logic would
likely be more difficult to handle using random set theory, but it could also be explored
as a possibility.

In all of the simulations and tests involving soft sensors, it was assumed that hu-
man observers would classify target driving styles (normal or erratic) and intent (friend,
foe, or neutral) based only on their own observations, without being influenced by other
sensors. A possible area for future research would be to see if it is possible to relax this
assumption. This would require modification to the random set measurement model to
account for the statistical correlation between reports from one human observer and the
other sensors. It would also require testing involving some human observers that have
access to data from partners when making their decisions and others who do not.

A related issue to explore would be the connection between the driving style of
a target (normal or erratic) and its intent (friend, foe, or neutral). It is plausible that
a target which is traveling erratically is more likely to be hostile than one that is not.
There is not necessarily a one-to-one correspondence, however: hostile targets very well
could drive in a straight line, and neutral targets could drive erratically (if they were lost,
for instance). If training data were available, it would be possible to statistically model
the relationship between driving style and intent. Without data, this would need to be
done heuristically. Explicitly modeling the connection between driving style and intent
could have two major benefits: First, it could yield an improvement in tracking results by better fitting the motion model to the target type. Second, it would allow hard sensors alone to make some inferences about target intent. For example, in Section 6.1.5.4 it was shown that a hard sensor alone could classify a target as erratic. If the connection between erratic and foe had been modeled, it would have been possible for the hard sensor to infer that the target was more likely to be a foe, based on how it was driving.

Another issue related to classification involves the speed at which a human can determine that a vehicle is driving erratically. In Section 6.1.5.4 the simulated human was very slow to make a determination that the vehicle was traveling erratically. Consequently, the addition of the soft sensor did not yield a performance benefit in terms of classification compared to the hard sensor alone. If, on the other hand, the simulated soft sensor responded instantly to a change in trajectory, there likely would have been a performance benefit in terms of classification speed and tracking error. One area for future work could be to vary the response time of the soft sensor in simulation to find the relationship between response time and performance. An even more interesting test would be to show a group of people an animation of a target and to ask them to press a button if the target starts driving erratically. This would yield understanding as to the types of maneuvers people consider erratic and how quickly they can respond to them. This information could then be used in the simulation to yield more realistic results.

Also, it will be important to further test the DHGPF in a real world scenarios. The simulations showed that the DHGPF is capable of scaling to entire swarms of vehicles, therefore testing it with more vehicles would be theoretically tractable, although there would be additional cost and the need for more people to act as soft sensors and safety pilots. The error in the tracking results with the DHGPF could likely be drastically improved with the addition of more sophisticated sensors, such as high resolution cameras, laser altimeters, and compasses onboard the UAVs and high resolution cameras and LIDARs or stereo cameras onboard the UGVs. Also if more sophisticated control capabilities were added, it would be possible to test such things as avoiding obstacles detected by other vehicles or avoiding hostile forces detected by a soft sensor.
Appendix A

State estimation

Estimation theory deals with determining some unknown quantity $x$, based on measurements $y$, which in some way depend on $x$. Estimation theory is used in this thesis both for tracking moving objects and for determining the position of a robot relative to features in its environment. State estimation has much broader applications in such fields as control, guidance, and navigation. There are many good books about estimation theory. A broad overview of the field of state estimation can be found in Simon [10]. Thrun et al. [11] derive results that are particularly important in the robotics domain. Ristic et al. [7] provide many results for applying particle filters to tracking problems. Finally, Mahler [5] uses random set theory to extend traditional estimation techniques to non-traditional data types. This appendix will summarize (without derivation) some of the most important results in the field of state estimation. For details please see the references.

A.1 Bayes filter

All estimators in this section are based on the recursive Bayes filter. The Bayes filter is a very general estimator for nonlinear systems. The complete algorithm for the filter is given in Algorithm A.1. The filter assumes the state $x$ and measurements $y$ are given by a known nonlinear model (Equation (A.1)). It is also assumed that some initial probability distribution function (pdf) of the state $p(x_0)$ is known. (If it is not known, a uniform distribution can be used.) Based on this information, the filter recursively calculates two quantities: an a priori pdf and an a posteriori pdf. The a priori pdf, $p(x_k|Y_{k-1})$, (given by Equation (A.3)) represents an estimate of the value of the state at some time $k$, given all measurements, $Y_{k-1} = \{y_1, y_2, \ldots, y_{k-1}\}$, up till (but not including) time $k$. The a posteriori pdf, $p(x_k|Y_k)$, (given by Equation (A.4)) represents an estimate
of the state given all measurements including \( y_k \), the measurement at time \( k \). The Bayes filter is deceptively simple \([5]\). Although there are only two computations that must be carried out at each time step, these computations usually do not have closed form solutions. One exception is the case where the system equations are linear and the noise terms are Gaussian, in which case the well known Kalman filter is the closed form solution. In most other cases, some approximation must be employed.

Algorithm A.1 The recursive Bayes filter \([10]\)

1: The system is governed by the following equations:

\[
x_{k+1} = f_k(x_k, w_k) \quad (A.1a)
\]

\[
y_k = h_k(x_k, v_k), \quad (A.1b)
\]

where \( \{w_k\} \) and \( \{v_k\} \) are independent white noise with known pdf’s.

2: Initialize the estimator using the pdf of the initial state:

\[
p(x_0|Y_0) = p(x_0) \quad (A.2)
\]

3: for \( k = 1, 2, \ldots \) do

4: Calculate the a priori pdf:

\[
p(x_k|Y_{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|Y_{k-1}) \, dx_{k-1} \quad (A.3)
\]

5: Calculate the a posteriori pdf:

\[
p(x_k|Y_k) = \frac{p(y_k|x_k)p(x_k|Y_{k-1})}{\int p(y_k|x_k)p(x_k|Y_{k-1}) \, dx_k} \quad (A.4)
\]

6: end for

A.2 Linear Kalman filters

For systems described by linear equations with additive Gaussian noise, the Bayes filter reduces to the well known Kalman filter \([5]\). The algorithm for Kalman filter is given as Algorithm A.2. Many notations are used in the literature to denote a priori and a posteriori estimates. For simplicity, this section will adopt the notation used by Simon \([10]\), where \( \hat{x}_k^- \) denotes the a priori state estimate (the estimate at time \( k \) prior to
receiving a measurement), $P_k^-$ denotes the \textit{a priori} covariance matrix, $\hat{x}_k^+$ denotes the \textit{a posteriori} state estimate (the estimate after receiving a measurement), and $P_k^+$ denotes the \textit{a posteriori} covariance matrix. Using this notation, in the case of a linear Gaussian system, Equation (A.3) reduces to

$$p(x_k|Y_{k-1}) = \mathcal{N}(\hat{x}_k^-, P_k^-),$$

and Equation (A.4) reduces to

$$p(x_k|Y_k) = \mathcal{N}(\hat{x}_k^+, P_k^+),$$

where $\mathcal{N}(x, P)$ is the normal distribution with mean $x$ and covariance $P$. Note that in Equation (A.12) the needed quantities are computed by simple matrix operations, thus avoiding the numerically costly integrations of the Bayes filter.

In some cases it is useful to use a different form of the Kalman filter, known as the information filter (Algorithm A.3). The information filter utilizes the canonical parametrization of a Gaussian random variable, which is given in terms of two quantities: the information matrix

$$\mathcal{I} = P^{-1}$$

and the information vector

$$\xi = \mathcal{I}x.$$

The information filter is mathematically equivalent to the Kalman filter, but it may be more numerically efficient in certain cases. Namely, if there are many more measurements than states, then the measurement update can be done more efficiently using the information filter (at the expense of an inefficient state update). The other advantage is if many measurements from different sources are fused at a central location, then the information term can be updated by adding in each measurement individually. For instance, if there were two measurements at time $k$, $y_{k,1}$ and $y_{k,2}$, Equation (A.15d) could
be modified as follows:

\[ \xi_k^+ = H_k^T R_{k,1}^{-1} y_{k,1} + H_k^T R_{k,2}^{-1} y_{k,2} + \xi_k^- \]  \hspace{1cm} (A.9)

The equation for updating the information matrix could be modified similarly. This form of the measurement update is especially useful in distributed data fusion systems.

### A.3 Extended Kalman filter

In practice, all real systems are nonlinear. The Bayes filter provides an optimal solution to estimating the state of a nonlinear system, but it is computationally intractable in all but the simplest cases (i.e. linear systems). One method for overcoming this limitation is the extended Kalman filter (EKF), given by Algorithm A.4. The EKF works by approximating the state and measurement equations as linear about the current estimate of the state. It does this by computing the partial derivative (Jacobian) matrices of the state equation (Equation (A.18)) and the measurement equation (Equation (A.20)). These Jacobians are then used analogously to their linear counterparts in the Kalman filter.

### A.4 Particle filter

In cases where nonlinearities are severe and noise is highly non-Gaussian, the particle filter (Algorithm A.5) provides a computationally tractable approximation to the Bayes filter. The particle filter approximates the probability distributions in the Bayes filter by a group of particles. There should be many particles in regions of the state space where the probability is high and few particles in regions where it is low. The number of particles, \( N \), must be selected by the user. More particles will yield a more accurate approximation, at the expense of higher computation. One important step of the algorithm that deserves more attention is line 7, the resampling step. A simple resampling procedure is given in Algorithm A.6. This method is simple but it suffers from some drawbacks: first, it is computationally inefficient; and second, it can lead to a phenomenon known as sample impoverishment, where all of the samples are
Algorithm A.2 The Kalman filter [10]

1: The system is governed by the following equations:

\[ x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1} \]  
(A.10a)

\[ y_k = H_kx_k + v_k \]  
(A.10b)

\[ w_k \sim \mathcal{N}(0, Q_k) \]  
(A.10c)

\[ v_k \sim \mathcal{N}(0, R_k) \]  
(A.10d)

\[ E(w_k w^T_j) = Q_k \delta_{k-j} \]  
(A.10e)

\[ E(v_k v^T_j) = R_k \delta_{k-j} \]  
(A.10f)

\[ E(w_k v^T_k) = 0 \]  
(A.10g)

2: Initialize the estimator as follows:

\[ \hat{x}_0^+ = E(x_0) \]  
(A.11a)

\[ P_0^+ = E \left[ (x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T \right] \]  
(A.11b)

3: for \( k = 1, 2, \ldots \) do

4: Update the state estimate as follows:

\[ P^-_k = F_{k-1}P^+_k F_{k-1}^T + Q_{k-1} \]  
(A.12a)

\[ K_k = P^-_k H_k^T (H_k P^-_k H_k^T + R_k)^{-1} \]  
(A.12b)

\[ \hat{x}^-_k = F_{k-1} \hat{x}^+_k - G_{k-1}u_{k-1} \]  
(A.12c)

\[ \hat{x}^+_k = \hat{x}^-_k + K_k(y_k - H_k \hat{x}^-_k) \]  
(A.12d)

\[ P^-_k = (I - K_k H_k) P^-_k \]  
(A.12e)

5: end for
Algorithm A.3 Information form of Kalman filter [10,11]

1: The system is governed by the following equations:

\[
x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1} \tag{A.13a}
\]
\[
y_k = H_k x_k + v_k \tag{A.13b}
\]
\[
w_k \sim \mathcal{N}(0, Q_k) \tag{A.13c}
\]
\[
v_k \sim \mathcal{N}(0, R_k) \tag{A.13d}
\]
\[
E(w_k w_j^T) = Q_k \delta_{k-j} \tag{A.13e}
\]
\[
E(v_k v_j^T) = R_k \delta_{k-j} \tag{A.13f}
\]
\[
E(w_k v_k^T) = 0 \tag{A.13g}
\]

2: Initialize the estimator as follows:

\[
\hat{x}_0^+ = E(x_0) \tag{A.14a}
\]
\[
\mathcal{I}_0^+ = \left\{ E \left[ (x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T \right] \right\}^{-1} \tag{A.14b}
\]
\[
\xi_0^+ = \mathcal{I}_0^+ \hat{x}_0^+ \tag{A.14c}
\]

3: for \( k = 1, 2, \ldots \) do

4: Update the state estimate as follows:

\[
\mathcal{I}_k^- = (F_{k-1}P_{k-1}^+ F_{k-1}^T + Q_{k-1})^{-1} \tag{A.15a}
\]
\[
\xi_k^- = \mathcal{I}_k^- \left( F_{k-1} (\mathcal{I}_{k-1}^+)^{-1} \xi_{k-1}^+ + G_{k-1}u_{k-1} \right) \tag{A.15b}
\]
\[
\mathcal{I}_k^+ = H_k^T R_k^{-1} H_k + \mathcal{I}_k^- \tag{A.15c}
\]
\[
\xi_k^+ = H_k^T R_k^{-1} y_k + \xi_k^- \tag{A.15d}
\]

5: end for
Algorithm A.4 Extended Kalman filter [10]

1: The system is governed by the following equations:
   \[ x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \]  
   \[ y_k = h_k(x_k, v_k) \]  
   \[ w_k \sim \mathcal{N}(0, Q_k) \]  
   \[ v_k \sim \mathcal{N}(0, R_k) \]

2: Initialize the estimator as follows:
   \[ \hat{x}_0^+ = E(x_0) \]  
   \[ P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T] \]

3: for \( k = 1, 2, \ldots \) do
4: Compute the following partial derivatives:
   \[ F_{k-1} = \left. \frac{\partial f_{k-1}}{\partial x} \right|_{\hat{x}_{k-1}^+} \]  
   \[ L_{k-1} = \left. \frac{\partial f_{k-1}}{\partial w} \right|_{\hat{x}_{k-1}^+} \]

5: Update the state estimate as follows:
   \[ P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + L_{k-1} Q_{k-1} L_{k-1}^T \]  
   \[ \hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) \]

6: Compute the following partial derivatives:
   \[ H_k = \left. \frac{\partial h_k}{\partial x} \right|_{\hat{x}_k^-} \]  
   \[ M_k = \left. \frac{\partial h_k}{\partial v} \right|_{\hat{x}_k^-} \]

7: Correct the state estimate based upon the observation:
   \[ K_k = P_k^- H_k^T (H_k P_k^- H_k^T + M_k R_k M_k^T)^{-1} \]  
   \[ \hat{x}_k^+ = \hat{x}_k^- + K_k [y_k - h_k(\hat{x}_k^-, 0)] \]  
   \[ P_k^+ = (I - K_k H_k) P_k^- \]

8: end for
concentrated at a few points. To overcome these difficulties, many variations of the particle filter have been proposed, which are more efficient or more robust to sample impoverishment. Details on the many variations on particle filtering can be found in Ristic et al. [7] or Simon [10].
Algorithm A.5 The particle filter [10]

1: The system is governed by the following equations:

\[ x_{k+1} = f_k(x_k, w_k) \]  \hspace{1cm} (A.22a)
\[ y_k = h_k(x_k, v_k), \]  \hspace{1cm} (A.22b)

where \( \{w_k\} \) and \( \{v_k\} \) are independent white noise with known pdf’s.

2: Randomly generate \( N \) particles by sampling from the pdf of the initial state, \( p(x_0) \).

These particles are denoted by \( \{x_{0,i}^k\} \).

3: for \( k = 1, 2, \ldots \) do

4: Obtain the a priori particles \( x_{k,i}^- \) using the following equation:

\[ x_{k,i}^- = f_{k-1}(x_{k-1,i}^+, w_{k-1,i}^i), \]  \hspace{1cm} (A.23)

where each noise vector, \( w_{k-1,i}^i \) is generated randomly based on the known pdf of \( w_{k-1} \).

5: Compute the relative likelihood \( q_i \) of each particle \( x_{k,i}^- \) conditioned on the measurement \( y_k \). This is done by evaluating the pdf \( p(y_k|x_{k,i}^-) \), which is based on the measurement equation and the pdf of the measurement noise.

6: Scale the relative likelihoods so that they sum to one as follows:

\[ q_i = \frac{q_i}{\sum_{j=1}^N q_j} \]  \hspace{1cm} (A.24)

7: Generate a set of a posteriori particles \( x_{k,i}^+ \) by resampling in proportion to the relative likelihoods \( q_i \).

8: end for

Algorithm A.6 Resampling algorithm [7]

1: Initialize the set of cumulative sum of weights (CSW): \( c_1 = q_k^i \)

2: for \( i = 2 \) to \( N \) do

3: Update the CSW: \( c_i = c_{i-1} + q_k^i \)

4: end for

5: Start at the bottom of the CSW: \( i = 1 \)

6: Generate a random number \( r_1 \) that is uniformly distributed on \([0, N^{-1}]\).

7: for \( j = 1 \) to \( N \) do

8: Move along the CSW: \( r_j = r_1 + N^{-1}(j - 1) \)

9: while \( r_j > c_i \) do

10: \( i = i + 1 \)

11: end while

12: Assign sample: \( x_{k}^{j*} = x_{k}^i \)

13: Assign weight: \( w_k^j = N^{-1} \)

14: end for
Fuzzy logic

Fuzzy logic is an extension of classical two-valued logic which allows elements to partially belong to a set. The term has also come to be applied to all technologies which utilize fuzzy set theory. It was first proposed by Lofti Zadeh in 1965 [28]. Since then many results based on Zadeh's fuzzy sets have been devised. This appendix will focus on the important results for this thesis. Much more detail can be found in a reference text on fuzzy logic, such as Yen and Langari [29].

B.1 Fuzzy sets

At the heart of fuzzy logic is the concept of a fuzzy set [28,29]. A fuzzy set allows elements to partially belong to a set (in other words, it has smooth boundaries, rather than sharp boundaries). A fuzzy set $A$ is defined by a fuzzy membership function $\mu_A(x)$, which describes the degree to which an element $x$ belongs to the set. If $\mu_A(x) = 1$, then $x$ definitely belongs to $A$, but if $\mu_A(x) = 0$, $x$ definitely does not belong to $A$. For instance, if tall where a fuzzy membership function then it would be reasonable to define $\mu_{\text{tall}}(5\text{ft}) = 0$, $\mu_{\text{tall}}(6\text{ft}) = 0.5$, and $\mu_{\text{tall}}(7\text{ft}) = 1$.

For discrete fuzzy sets, it is possible to enumerate the fuzzy membership function values element-by-element, but for continuous distributions this is not possible. Instead, membership functions for continuous distributions are described mathematically. Any function that maps the domain $X$ to the interval $[0, 1]$ can serve as a fuzzy membership function, but there are several standard membership functions which have found wide acceptance. Four of these are important for this thesis: triangular, trapezoidal, Gaussian, and sigmoidal.

**Triangular membership function** The triangular membership function is one of the most common membership functions, due to its simplicity. It is specified by three
parameters \{a, b, c\}, which specify the lower cutoff, peak value, and upper cutoff of the membership function, respectively. The equation for the membership function is:

\[
\text{triangle}(x : a, b, c) = \begin{cases} 
0 & x < a \\
(x - a)/(b - a) & a \leq x < b \\
(c - x)/(c - b) & b \leq x < c \\
0 & x > c
\end{cases}.
\] (B.1)

Figure B.1a shows the membership function \text{triangle}(x, 10, 15, 25).

**Trapezoidal membership function** The trapezoidal membership function is specified by four parameters \{a, b, c, d\}. It allows multiple points to have a membership function value of one. The equation for the trapezoidal membership function is:

\[
\text{trapezoid}(x : a, b, c, d) = \begin{cases} 
0 & x < a \\
(x - a)/(b - a) & a \leq x < b \\
1 & b \leq x < c \\
(d - x)/(d - c) & c \leq x < d \\
0 & x > d
\end{cases}.
\] (B.2)

The triangular membership function is a special case of the trapezoidal membership function with \(b = c\). Figure B.1b shows the membership function \text{trapezoid}(x, 10, 15, 25, 40).

Due to their simplicity, both triangular and trapezoidal membership functions have found extensive use in practice [29].

**Gaussian membership function** A Gaussian membership function is given by two parameters \{m, \sigma\}, which specify the center and spread of the membership function. The equation for the Gaussian membership function is:

\[
\text{Gaussian}(x : m, \sigma) = \exp\left( -\frac{(x - m)^2}{\sigma^2} \right).
\] (B.3)
The Gaussian membership function has the same shape as the Gaussian probability distribution, but it is not a probability distribution because it does not integrate to one. It is useful because it is differentiable and because it can be used similarly to a Gaussian probability distribution. Figure B.1c shows the membership function Gaussian($x, 30, 20$).

**Sigmoidal membership function** The sigmoidal membership function is useful on the boundaries of a given domain. For instance, a man’s height could be regarded as tall if it is above six feet. It is controlled by two parameters $\{a, c\}$ which control the slope and break-even point (i.e. where $\mu = 0.5$) of the membership function. The equation for the sigmoidal membership function is:

$$\text{sigm}(x : a, c) = \frac{1}{1 + e^{-a(x-c)}}. \quad (B.4)$$

The sigmoidal membership function is similar to the sigmoidal activation function in neural networks, and is useful for hybrid neuro-fuzzy systems. Figure B.1d shows a plot of the function sigm($x : 0.1, 30$).

### B.2 Fuzzy logic operations

Just as two valued logic defines the logical operations conjunction (and), disjunction (or), and complement (not), similar operations can be defined for fuzzy logic. Fuzzy complement is simple to define:

$$\mu_{\neg A}(x) = 1 - \mu_A(x). \quad (B.5)$$

Conjunction and disjunction are more complicated because there are many possible choices. A fuzzy conjunction operator is defined by a triangular norm or t-norm, $t(x, y)$. Likewise, a fuzzy disjunction operator is defined by a fuzzy t-conorm (or s-norm), $s(x, y)$. The two operators form a dual pair if they satisfy:

$$1 - t(x, y) = s(1 - x, 1 - y), \quad (B.6)$$
(a) Triangular membership function  
(b) Trapezoidal membership function  
(c) Gaussian membership function  
(d) Sigmoidal membership function  

Fig. B.1: Fuzzy membership functions
which is equivalent to De Morgan’s Law for traditional logic:

\[-(x \land y) = \neg x \lor \neg y.\] (B.7)

Fuzzy conjunction and disjunction operators are used to define intersection and union of fuzzy sets as follows:

\[
\mu_{(A \cap B)}(x) = t(\mu_A(x),\mu_B(x)) \quad \text{(B.8)}
\]

\[
\mu_{(A \cup B)}(x) = s(\mu_A(x),\mu_B(x)). \quad \text{(B.9)}
\]

There are four axioms, which a t-norm must satisfy [29]:

- \(t(0, 0) = 0\), \(t(x, 1) = t(1, x) = x\)
- \(t(x, y) \leq t(z, w)\) if \(x \leq z\) and \(y \leq w\) (monotonicity)
- \(t(x, y) = t(y, x)\) (commutativity)
- \(t(x, t(y, z)) = t(t(x, y), z)\) (associativity)

Likewise, a t-conorm must satisfy the following [29]:

- \(s(1, 1) = 1\), \(s(x, 0) = s(0, x) = x\)
- \(s(x, y) \leq s(z, w)\) if \(x \leq z\) and \(y \leq w\) (monotonicity)
- \(s(x, y) = s(y, x)\) (commutativity)
- \(s(x, s(y, z)) = s(s(x, y), z)\) (associativity)

There are many possible pairs of fuzzy conjunction and disjunction operators. Three important pairs are max and min, algebraic sum and product, and Hamacher sum and product. Max and min (also known as the Zadeh logic) is the most commonly used pair of operators:

\[
t(x, y) = \min\{x, y\} \quad \text{(B.10)}
\]

\[
s(x, y) = \max\{x, y\}. \quad \text{(B.11)}
\]
Another common choice is the algebraic product and algebraic sum:

\[ t(x, y) = x \cdot y \]  \hspace{1cm} (B.12)

\[ s(x, y) = x + y - x \cdot y. \]  \hspace{1cm} (B.13)

A less common, but still useful choice is the Hamacher product and Hamacher sum:

\[ t(x, y) = \frac{x \cdot y}{x + y - x \cdot y} \]  \hspace{1cm} (B.14)

\[ s(x, y) = \frac{x + y - 2xy}{1 - x \cdot y}. \]  \hspace{1cm} (B.15)

**B.3 Fuzzy mapping rules**

A fuzzy mapping rule is a generalization of a traditional functional mapping. A fuzzy mapping rule is usually expressed in terms of an if-then rule. There are four possible forms for such a mapping [29]:

1. Crisp constant: the output is a numerical constant. For instance, “IF \( x \) is Small THEN \( y = 5 \),” where \( x \) is the input, \( y \) is the output, and Small is a fuzzy set.

2. Fuzzy constant: the output is itself a fuzzy set. For instance, “IF \( x \) is Small THEN \( y \) is Medium.”

3. Linear model: the output is a linear function of the input. For instance, “IF \( x \) is Small THEN \( y = 4x \).”

4. Nonlinear model: the output is a nonlinear function of the input. For instance, “IF \( x \) is Small then \( y = f(x) \).”

It is also common to make rules with multiple inputs by joining them with AND, OR, and NOT. For instance, one could construct a rule of the form: “IF \( x_1 \) is Small AND \( x_2 \) is Medium THEN \( y \) is Large.” In practice, there will be several fuzzy rules that define a mapping between the input space and the output space. In this case, the rules
will form a fuzzy partition of the input space, with each rule focusing on a certain part of the space. This is analogous to piecewise linear function approximation, except that the transitions between various regions are smooth rather than abrupt.
Appendix C

Random set theory

C.1 Introduction to random set theory

In traditional measurement modeling, a measurement is characterized by a random vector $z$, which is an element of some measurement space $Z$. From this representation, a likelihood function $f(z|x)$ can be defined, which characterizes the probability that $z$ would take on the observed value given that the state of the entity being observed is $x$. From this likelihood, the state estimate can be updated using the Bayes filter described in Appendix A.

This data modeling approach becomes more difficult when measurements take on a form such as, “the target is near building 7.” Because this statement is fuzzy, it is unnatural to model it as a random variable. Random set theory addresses this problem. Rather than dealing with random vectors $z \in Z$, random set theory deals with random subsets of the measurement space $\Theta \subseteq Z$. For instance, the set $\Theta$ could be an interval, which represents all measurements which can be categorized as “near.” These random sets satisfy all the normal axioms of set theory [5]. From a random set observation $\Theta$ a generalized likelihood $f(\Theta|x)$ can be constructed, which can be used analogously to a traditional likelihood.

C.2 Random set uncertainty representations

The first step in using random sets for data fusion is to model the uncertainty in the data as a random set. For purely probabilistic data, this is straightforward, because a random variable is a random set with a single element. Fuzzy sets and Dempster-Shafer uncertainty representations, which are more involved, are described below.
C.2.1 Fuzzy sets

The random set representation of a fuzzy set starts with a fuzzy membership function (see Appendix B) \( \mu(u) \) (where \( u \) is in the universe of discourse \( \Omega \)), and a random number \( A \), which is uniformly distributed on \([0, 1]\). Then the synchronous random set representation [5] of \( \mu \) is

\[
\Sigma_A(\mu) \triangleq \{u | A \leq \mu(u)\}. \tag{C.1}
\]

For any \( \alpha \in [0, 1] \) the set \( \Sigma_\alpha(\mu) \subseteq U \) is the alpha-cut of the fuzzy membership function \( \mu \).

Because \( A \) is uniform,

\[
\Pr(u \in \Sigma_A(\mu)) = Pr(A \leq \mu(u)) = \mu(u). \tag{C.2}
\]

This means that the random set contains all of the information contained in the fuzzy membership function, while also attaching random content using the random variable \( A \).

Intersection and union of random sets is analogous to fuzzy conjunction and disjunction of fuzzy membership functions. In fact, if min is used for conjunction and max for disjunction, then

\[
\Sigma_A(f) \cap \Sigma_A(g) = \Sigma_A(f \land g) \tag{C.3}
\]

\[
\Sigma_A(f) \cup \Sigma_A(g) = \Sigma_A(f \lor g). \tag{C.4}
\]

However, the complement is not the same, \( \Sigma_A(f^c) \neq \Sigma_A(f^c) \).

C.2.2 Dempster-Shafer events

Recall from Section 3.2.2 that the Dempster-Shafer theory defined the basic mass assignment \( m(U) \) to various subsets \( U \) of the universe of discourse \( \Omega \). These subsets did not need to be mutually exclusive or exhaustive, but the sum off all the basic mass assignments had to be one. Equation (3.11) defined three quantities: belief, plausibility, and commonality from the basic mass assignment.
In order to represent a Dempster-Shafer event as a random set, start with a random subset $\Sigma_m$ of the universe of discourse $\Omega$. Then define the probability distribution of $\Sigma_m$ as follows [5]:

$$\Pr(\Sigma_m = U) = m(U),$$

(C.5)

where $U$ is a focal set of the Dempster-Shafer basic mass assignment $m(\cdot)$. Then $\Sigma_m$ is a random set representation of the basic mass assignment $m$.

From the random set representation, belief, plausibility, and commonality can be redefined as follows:

$$\text{Bel}_m(U) = \Pr(\Sigma_m \subseteq U)$$

(C.6)

$$\text{Pl}_m(U) = \Pr(\Sigma_m \cap U \neq \emptyset)$$

(C.7)

$$Q_m(U) = \Pr(\Sigma_m \supseteq U).$$

(C.8)

Dempster’s combination rule can also be expressed probabilistically as follows:

$$(m_1 * m_2)(U) = \Pr(\Sigma_{m_1} \cap \Sigma_{m_2} = U | \Sigma_{m_1} \cap \Sigma_{m_2} \neq \emptyset),$$

(C.9)

when it is assumed that $\Sigma_{m_1}$ and $\Sigma_{m_2}$ are independent and $\Pr(\Sigma_{m_1} \cap \Sigma_{m_2} \neq \emptyset) \neq 0$.

### C.2.3 Fuzzy Dempster-Shafer Theory

Fuzzy Dempster-Shafer theory is a generalization of Dempster-Shafer theory where the focal sets are allowed to be fuzzy sets [5]. In this case, there is a set of focal fuzzy sets $U_i$ with fuzzy membership functions $\mu_i$. For each of these fuzzy sets there is a basic mass assignment $m(\mu_i)$, which obeys all of the normal axioms of a basic mass assignment. Fuzzy Dempster’s combination is now defined as

$$(m_1 * m_2)(\mu) \triangleq \alpha^{-1} \sum_{\mu_1 \mu_2 = \mu} m_1(\mu_1) \cdot m_2(\mu_2),$$

(C.10)
where the agreement is defined as

\[
\alpha = \alpha_{DS}(m_1, m_2) \triangleq \sum_{\mu_1 \mu_2 \neq 0} m_1(\mu_1) \cdot m_2(\mu_2).
\] (C.11)

In the case where all the fuzzy focal sets are crisp, this reduces to ordinary Dempster’s combination. The random set representation of a fuzzy Dempster-Shafer event is more difficult to construct because it makes use of generalized fuzzy sets. For details on how to construct the random set representation see Mahler [5].

### C.3 Random set likelihoods

Once the uncertainty in a measurement is modeled as a random set, the next task is to use this random set representation in a generalization of the Bayes filter of Section A.1. This task is simplified if the measurement is of a type which Mahler [5] refers to as unambiguously generated ambiguous. This simply means that there is a precise equation of the form

\[ z = \eta(x), \] (C.12)

which associates measurements \( z \) to states \( x \). Any uncertainty in such a measurement is due to noise or ambiguity in the sensor itself. In such a case, the measurement is modeled as a random subset \( \Theta \) of the measurement space \( \mathcal{Z} \). The ambiguous observation is in agreement with this model if

\[ \eta(x) \in \Theta. \] (C.13)

From this measurement, it is possible to generate a generalized likelihood of the form

\[ f(\Theta | x) \triangleq \Pr(\eta(x) \in \Theta). \] (C.14)

This likelihood can be used analogously to the likelihood \( f(z | x) \) in the traditional Bayes filter (Algorithm A.1).
C.3.1 Statistical

In the special case that the sensor uncertainty is entirely statistical, the random set $\Theta$ is a set with a single element, the random variable $z$. In this case the generalized likelihood is simply a traditional likelihood:

$$f(\Theta_z|x) = f(z|x). \tag{C.15}$$

C.3.2 Fuzzy

In the case that the uncertainty is a fuzzy set with membership function $\mu(z)$, the generalized measurement is the random set $\Theta_\mu \triangleq \Sigma_A(\mu)$, where $\Sigma_A(\mu)$ was defined in Equation (C.1). The likelihood for this measurement is

$$f(\Theta_\mu|x) = \Pr(\eta(x) \in \Sigma_A(\mu)) = \mu(\eta(x)). \tag{C.16}$$

C.3.3 Dempster-Shafer

In the case that the uncertainty is described by a Dempster-Shafer basic mass assignment $m(O)$, where $O \subseteq Z$ is a subset of the measurement space, the generalized measurement is the random set $\Theta_m \triangleq \Sigma_m$, where $\Sigma_m$ was defined in Equation (C.5). The likelihood for this measurement is

$$f(\Theta_m|x) = \Pr(\eta(x) \in \Sigma_m) = \sum_{O \ni \eta(x)} m(O). \tag{C.17}$$

C.3.4 Fuzzy Dempster-Shafer

In the case where the measurement $\Theta_m$ is given as a fuzzy Dempster-Shafer mass assignment the likelihood is given as follows:

$$f(\Theta_m|x) = \sum_{\mu} m(\mu) \cdot \mu(\eta(x)). \tag{C.18}$$

For details on how this is derived see Mahler [5].
C.4 Bayesian data fusion

The purpose of modeling measurements as random sets then finding generalized
likelihoods \( f(\Theta|x) \) for these measurements, is ultimately to use these measurements
to create state estimates. In other words, the likelihood is used to create a posterior
distribution using Bayes’ rule:

\[
f(x|\Theta) \propto f(\Theta|x) \cdot f_0(x). \tag{C.19}
\]

In the case that there are two measurements \( \Theta_1 \) and \( \Theta_2 \), which should be fused, the
goal is to construct the posterior \( f(x|\Theta_1, \Theta_2) \) conditioned on both measurements. This
process can be simplified if there exists an operator \( \otimes \) which fuses \( \Theta_1 \) and \( \Theta_2 \) into a
composite measurement \( \Theta_1 \otimes \Theta_2 \). If the operator obeys the following:

\[
f(x|\Theta_1, \Theta_2) = f(x|\Theta_1 \otimes \Theta_2), \tag{C.20}
\]

then it is a Bayes combination operator \([5]\). In this case, fusion using the operator
yields the same results as fusion using Bayes’ rule. Examples of such operators include
Dempster’s combination and copula fuzzy conjunction.

C.4.1 Dempster’s Combination

In the case where measurements are given by Dempster-Shafer basic mass assign-
ments \( m_1 \) and \( m_2 \) it is possible to show that \([5]\):

\[
f(m_1 \ast m_2|x) = \alpha^{-1} f(m_1|x)f(m_2|x), \tag{C.21}
\]

where \( \alpha = \alpha_{DS}(m_1, m_2) \) was defined in equation (3.14). In the case where \( m_1 \) and \( m_2 \) are
independent (i.e., their random set representations are independent), this can be further
simplified because \( f(m_1, m_2|x) = f(m_1|x)f(m_2|x) \). By using Bayes’ rule it can further be shown that:

\[
f(x|m_1 \ast m_2) = f(x|m_1, m_2). \tag{C.22}
\]
This means that Dempster’s combination is a Bayes combination operator as long as the
two items to be fused are independent. Similarly, it can be shown that fuzzy Dempster’s
combination is also a Bayes combination operator.

C.4.2 Copula fuzzy conjunctions

If $A_1$ and $A_2$ are two uniformly distributed random numbers on $[0, 1]$ their copula
is defined as [5, 200, 222]:

$$a_1 \land_{A_1, A_2} a_2 \triangleq \Pr(A_1 \leq a_1, A_2 \leq a_2) \quad (C.23)$$

for $0 \leq a_1, a_2 \leq 1$. The statistical dependencies between the two random variables is
completely characterized by their copula.

Now by using the random set representations of two fuzzy sets, it can be shown
that:

$$(\mu_1 \land_{A_1, A_2} \mu_2)(u) \triangleq \Pr(u \in \Sigma_{A_1}(\mu_1) \cap \Sigma_{A_2}(\mu_2)) \quad (C.24)$$

$$= \Pr(A_1 \leq \mu_1(u), A_2 \leq \mu_2(u)) \quad (C.25)$$

$$= \mu_1(u) \land_{A_1, A_2} \mu_2(u). \quad (C.26)$$

Additionally:

$$0 \land_{A_1, A_2} a_2 = 0 \quad (C.27)$$

$$a_1 \land_{A_1, A_2} 0 = 0 \quad (C.28)$$

$$1 \land_{A_1, A_2} a_2 = a_2 \quad (C.29)$$

$$a_1 \land_{A_1, A_2} 1 = a_1. \quad (C.30)$$

If the operator is also commutative and associative, then it follows all of the axioms
of a fuzzy conjunction (t-norm) found in Section B.2. The corresponding disjunction
operator is given by
\[ a_1 \lor_{A_1, A_2} a_2 \triangleq 1 - (1 - a_1) \land_{A_1, A_2} (1 - a_2). \]  
(C.31)

In the case where \( A_1 = A_2 \), the random set representations of two fuzzy membership functions are completely correlated. This means that
\[ a_1 \land_{A_1, A_2} a_2 = \Pr(A_1 \leq \min\{a_1, a_2\}) = \min\{a_1, a_2\}. \]  
(C.32)

In other words, the copula fuzzy conjunction reduces to a simple min. The other extreme case occurs when the random set representations are independent. In this case
\[ a_1 \land_{A_1, A_2} a_2 = \Pr(A_1 \leq a_1, A_1 \leq a_2) = \Pr(A_1 \leq a_1)\Pr(A_1 \leq a_2) = a_1 \cdot a_2. \]  
(C.33)

This means that the copula fuzzy conjunction reduces to the product conjunction for independent data. Another useful copula fuzzy conjunction is the Hamacher conjunction defined in equation (B.14). It models the fusion of two random variables that are nearly statistically independent [5].

Given two fuzzy membership functions, their joint likelihood can be found using a copula fuzzy conjunction as follows:
\[ f(\mu_1, \mu_2|x) \triangleq \Pr(\eta(x) \in \Sigma_{A_1}(\mu_1), \eta(x) \in \Sigma_{A_2}(\mu_2)) \]  
(C.34)
\[ = f(\mu_1 \land_{A_1, A_2} \mu_2|x). \]  
(C.35)

This means that by Bayes’ rule:
\[ f(x|\mu_1 \land_{A_1, A_2} \mu_2) = f(x|\mu_1, \mu_2); \]  
(C.36)
therefore, copula fuzzy conjunction is a Bayes combination operator (for any known statistical dependence).
C.5 Bayes-invariant conversions

It is often useful to convert measurements in one measurement space $Z$ to another space $Z'$; for instance, measurements may be given as Dempster-Shafer basic mass assignments, but a tracker requires probabilistic statements. Ordinarily, this would not be possible because converting from one space to another would result in a huge loss of information. This is due to the fact that the semantics of Dempster-Shafer theory and probability are different. However, in the case that measurements are to be used in state estimation, a weaker form of equivalency is possible. If there is a function $\phi : Z \rightarrow Z'$ which converts between measurement spaces and it can be shown that

$$f(x|\zeta) = f(x|\phi(\zeta)), \quad (C.37)$$

then the conversion is said to be a Bayes-invariant conversion [5]. Such conversions may, in fact, represent information loss, but none of the lost information is relevant for state estimation. Below, several Bayes-invariant conversions are described. Composite conversions can be formed by combining multiple basic conversions.

C.5.1 Fuzzy to fuzzy Dempster-Shafer

It is relatively simple to convert from a fuzzy membership function to a fuzzy basic mass assignment (BMA). Given a fuzzy membership function $\mu(z)$ on the measurement space $Z$, define the fuzzy BMA $m(\mu')$ on membership functions $\mu'$ of $Z$ as follows:

$$m_{\mu}(\mu') \triangleq \begin{cases} 
1 & \text{if } \mu' = \mu \\
0 & \text{if } \text{otherwise}
\end{cases} \quad (C.38)$$

Given this conversion it is possible to show that [5]

$$f(\Theta_\mu|x) = f(\Theta_m|x) \quad (C.39)$$

$$m_{\mu\mu'} = m_\mu * m_{\mu'} \quad (C.40)$$
In other words, the likelihood of the fuzzy measurement is identical to the likelihood of the fuzzy Dempster-Shafer measurement. Also, fusion then conversion is identical to conversion then fusion. Therefore, the conversion is a Bayes-invariant conversion.

**C.5.2 Fuzzy Dempster-Shafer to fuzzy**

Given a fuzzy basic mass assignment $m(g)$ on fuzzy membership functions $g$ of $\mathcal{Z}$, define the fuzzy membership function $\mu_m(z)$ on $\mathcal{Z}$ as follows

$$
\mu_m(\zeta) \triangleq \sum_g m(g) \cdot g(\zeta).
$$

(C.41)

Then it can be shown that [5]

$$
f(\Theta_m|x) = f(\Theta_\mu|x) \quad (C.42)
$$

$$
\mu_{m \ast m'} = \alpha^{-1} \cdot \mu_m \cdot \mu_{m'}.
$$

(C.43)

This means that the conversion is indeed a Bayes-invariant conversion.

**C.5.3 Fuzzy Dempster-Shafer to probabilistic**

If $m(\mu)$ is a fuzzy BMA and $\int \mu(z)dz < \infty$ whenever $m(\mu) > 0$ then it is possible to define a probability density function $\varphi(z)$ on $\mathcal{Z}$ as follows:

$$
\varphi_m(z) \triangleq \frac{\sum_\mu m(\mu) \cdot \mu(z)}{\sum_\mu m(\mu) \int \mu(w)dw}.
$$

(C.44)

This is a generalization of the Voorbraak transform of Dempster-Shafer theory to fuzzy BMAs [5].

Define the parallel combination of two probability density functions $\varphi$ and $\varphi'$ as follows:

$$
(\varphi \ast \varphi')(z) \triangleq \frac{\varphi(z) \cdot \varphi'(z)}{\int \varphi(w) \cdot \varphi'(w)dw}.
$$

(C.45)
Finally, define the following likelihood:

\[ f(\varphi|x) \triangleq \frac{\varphi(\eta(x))}{\sup_z \varphi(z)}. \quad (C.46) \]

Given these definitions, it can be shown that

\[ f(\varphi_m|x) = K \cdot f(\Theta_m|x) \quad (C.47) \]
\[ \varphi_{m*m'}(z) = (\varphi_m * \varphi_{m'})(z), \quad (C.48) \]

where \( K \) is a constant independent of \( x \). This means that the conversion is a Bayes-invariant conversion.

C.5.4 Probabilistic to fuzzy

Because any valid probability distribution is a valid fuzzy membership function (but not the other way around), it is relatively straightforward to transform a probability distribution \( \varphi(z) \) on \( Z \) to a fuzzy membership function. The transformation is given as

\[ \mu_\varphi(z) \triangleq \frac{\varphi(z)}{\sup_w \varphi(w)}. \quad (C.49) \]

It can then be shown that [5]

\[ f(\mu_\varphi|x) = f(\varphi|x) \quad (C.50) \]
\[ \mu_{\varphi * \varphi'} = K \cdot \mu_\varphi \cdot \mu_{\varphi'}, \quad (C.51) \]

where \( K \) is a constant independent of \( x \). This means that the conversion is a Bayes-invariant conversion.
Bibliography


realizations,” in Recent Developments in Fuzzy Set and Possibility Theory (R. R. 


reasoning,” in Conditional Logic in Expert Systems (I. R. Goodman, M. M. Gupta, 

really bad data,” Signal Processing, Sensor Fusion, and Target Recognition VIII, 


Vita

Gregory L. Sinsley

Gregory Sinsley was born and raised in Scottdale, Pennsylvania. He received his B.S. in Electrical Engineering with Computer Emphasis from Grove City College in Northwestern Pennsylvania in May of 2005. In August 2005 he enrolled in the Ph.D. program in Electrical Engineering at the Pennsylvania State University. He worked for two semesters as a teaching assistant, teaching Introduction to Microcontrollers. In June 2006 he began working as a Walker Graduate Assistant at the Applied Research Laboratory, Applied Intelligent Systems Lab. In August of 2006 he married his wife Alissa. He completed his Ph.D. in Electrical Engineering with minors in Computational Science and Aerospace Engineering in August of 2012. His dissertation focused on fusing sensor data with robots with data from “soft” human sensors. He has accepted a job as an Electrical Engineer with Applied Technology, Inc.