INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.
VISION BASED OBSTACLE AVOIDANCE USING BIOLOGICAL MODELS

by

John M. Galbraith

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA

2002
UMI Number: 3073221

Copyright 2003 by
Galbraith, John M.

All rights reserved.
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by John Calbraith entitled Vision Based Obstacle Avoidance Using Biological Models and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Richard W. Ziolkowski, Ph.D.  
Date 12/02/02

Glen C. Berhard, Ph.D.  
Date 12/02/02

Garrett T. Kenyon, Ph.D.  
Date 12/02/02

Jerzy W. Rosenglit, Ph.D.  
Date

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director Richard W. Ziolkowski, Ph.D.  
Date 12/19/02
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]
ACKNOWLEDGMENTS

Sometimes one receives a service from somebody that is so significant that it is quite impossible to return a gift of equal value. At those times, one just has to do what he can to express what it means and hope for the opportunity to offer a similar service to somebody else who might come along one day. For me, I can only express gratitude to my advisor, Richard Ziolkowski, for rescuing this work when it was in shambles and seeing it through to completion. As you can probably tell by the weight of the document currently in your hands, this was no small commitment. I spent eight years in graduate school, but only fourteen months under Dr. Ziolkowski. Those few months were more productive for me than the entire previous experience. Thank you for the clear objectives, challenging yet achievable goals, and the countless prompt hours spent reading this document many times over. You salvaged eight years of my professional life, and I will continue to reap the benefits for the rest of my career.

I also owe thanks to my friend Garrett Kenyon for sharing his enthusiasm of being a scientist and turning me into one, and who, with the P-21 line management at Los Alamos National Laboratory: Bob Scarlett, Chris Wood, and Clayton Smith: would not let me quit this work when I wanted to and also provided the technical and financial support that made the whole thing possible.

My family has been very supportive of me during this time in my life, just like all the other times before. Thanks in particular to Amy for lots of stuff including hooking me up with Dr. Ziolkowski, and my stepfather Arvid who continues to be a professional and personal role model and encouraged me to get into this business in the first place.

Finally, I would like to thank the countless thousands of people out there on the Internet who, by sharing their source code for free, make scientific computing fun. I wrote a lot of code myself for this work, but I linked to and used a lot more than I could possibly write. Thank you to everybody who contributed to GNU/Linux, FreeBSD, Python, Emacs, FFTW, NetCDF, GCC, LAM-MPI, LaTeX, Metapost, Mesa/OpenGL, Blitz++, Video4Linux, and probably a whole lot more. That’s all great code that made this work technically possible.
# TABLE OF CONTENTS

**LIST OF FIGURES** ......................................................... 8

**LIST OF TABLES** ....................................................... 13

**ABSTRACT** ................................................................. 14

**1. INTRODUCTION AND LITERATURE REVIEW** ...................... 15

1.1. Robotics Review ...................................................... 18
1.1.1. Robot Taxonomy .................................................. 18
1.1.2. Popular Biological Sources .................................... 22
1.1.3. Common Problems in Robotics Navigation .................... 23
1.1.4. Robotics Literature Review Summary ......................... 28
1.1.5. Traditional AI Based Robots ................................... 30
1.1.6. Behavior Based Robots ......................................... 31
1.1.7. Biologically Inspired Robots .................................. 40
1.1.8. Neuromorphic Robots ........................................... 43
1.2. Neuroscience Review ............................................... 48
1.2.1. Implementation Strategy ....................................... 51
1.2.2. Biological Information Processing ........................... 53
1.3. Chapter Summary .................................................... 65

**2. MAMMALIAN EARLY VISION** ........................................ 66

2.1. V1 Spatial Primitives ............................................... 66
2.1.1. Fourier Components ............................................ 67
2.1.2. Windows and Receptive Fields ............................... 68
2.1.3. Spatial Feature Detectors ..................................... 71
2.1.4. Implementation ................................................. 74
2.2. V1 Motion Primitives and Motion Energy ....................... 76
2.2.1. Spatiotemporal Orientation ................................... 77
2.2.2. Aperture Problem .............................................. 82
# TABLE OF CONTENTS – Continued

2.2.3. Motion Energy ................................................. 84
2.2.4. Implementation ............................................... 87

2.3. Motion Energy Integration ........................................ 96
   2.3.1. General Form of Motion Energy Integration .................. 97
   2.3.2. Cascaded, Truncated Exponential Filter .................. 100
   2.3.3. Derivatives of a Gaussian Filter .......................... 104
   2.3.4. Literature Review ........................................... 104

2.4. Chapter Summary .................................................. 117

3. PROTOTYPE IMPLEMENTATION ........................................... 118
   3.1. Hardware ....................................................... 119
      3.1.1. Robot Platform ........................................... 119
      3.1.2. Digital Computer ......................................... 122
      3.1.3. Overhead Imager .......................................... 126
   3.2. Software ....................................................... 127
      3.2.1. Information Pipeline ...................................... 130
      3.2.2. Message Passing .......................................... 131
      3.2.3. Initialization ............................................ 135
   3.3. Chapter Summary ................................................ 136

4. OBSTACLE DETECTION .................................................. 138
   4.1. Theory of Operation ............................................ 138
      4.1.1. Time to Collision Features .............................. 141
      4.1.2. Translation Features .................................... 144
      4.1.3. Rotation Features ....................................... 156
      4.1.4. Velocity Features ....................................... 157
      4.1.5. Feature Summary ....................................... 165
   4.2. Feature Space Tuning ........................................... 166
   4.3. Approach Course Trials ........................................ 174
      4.3.1. Image Preprocessing ..................................... 177
      4.3.2. Spatial Features ......................................... 182
      4.3.3. Motion Energy Features ................................. 186
      4.3.4. Velocity Features ....................................... 194
      4.3.5. Translation Features .................................... 213
      4.3.6. Collision Features ...................................... 213
   4.4. Turn Course Trials .............................................. 231
      4.4.1. Velocity Features ....................................... 231
# TABLE OF CONTENTS – Continued

4.4.2. Rotation Features .................................................. 240  
4.4.3. Translation Features ................................................. 250  
4.4.4. Collision Features .................................................. 259  
4.5. Chapter Summary ..................................................... 270  

5. ROBOT AUTONOMY ....................................................... 275  
5.1. Control Features ..................................................... 275  
5.2. Strategic Navigation .................................................. 277  
5.3. Tactical Navigation ................................................... 283  
5.3.1. Avoidance Behavior ............................................... 285  
5.3.2. Behavior Integration .............................................. 291  
5.4. Chapter Summary ..................................................... 296  

6. CONCLUSIONS ........................................................... 299  
6.1. Common Navigational Problems Revisited ..................... 300  
6.2. Future Work ............................................................ 309  

APPENDIX A. OPTIC FLOW IMPLEMENTATION ......................... 313  
A.1. Theory of Differential Optical Flow ............................. 313  
A.2. Implementation Details .............................................. 315  

APPENDIX B. PARAMETER TUNING TABLES .......................... 317  

REFERENCES ............................................................... 329
LIST OF FIGURES

1.1. Natural vs. Engineered Autonomous Agents .............................................. 49
1.2. Visual Modes in Natural Systems .............................................................. 51
1.3. Rate Coding ................................................................................................. 55
1.4. Layers of Neural Hypercolumns .................................................................. 57
1.5. Receptive Field Topology ............................................................................ 58
1.6. Line Features ............................................................................................... 59
1.7. Population Coding ....................................................................................... 60
1.8. Motion Pathway of Mammalian Early Vision ............................................... 62
1.9. Natural vs. Electronic Retinas ...................................................................... 63

2.1. Synthetic Grating Stimulus ......................................................................... 70
2.2. 1D Gabor Function ..................................................................................... 70
2.3. 2D Gabor Function ..................................................................................... 71
2.4. Edge Detector .............................................................................................. 73
2.5. Visual Processing Layers ............................................................................ 75
2.6. Moving Bar Stimulus ................................................................................. 77
2.7. Motion as Spatiotemporal Orientation ......................................................... 78
2.8. 1D Motion in the Frequency Domain .......................................................... 80
2.9. 2D Motion in the Frequency Domain .......................................................... 81
2.10. Aperture Problem ....................................................................................... 83
2.11. Aperture Problem in Frequency Domain ................................................... 83
2.12. Separable Filters ....................................................................................... 85
2.13. Motion Energy Stages ................................................................................ 86
2.14. Bandpass Filter Transform ......................................................................... 89
2.15. Motion Energy in the Frequency Domain .................................................. 96
2.16. General Form of Motion Energy Integration .............................................. 99
2.17. 1D Center-Surround Impulse Response ..................................................... 101
2.18. Center-Surround Action ............................................................................. 101
2.19. Truncated Exponential IIR Filter Impulse Response ................................. 103
2.20. Derivatives of a Gaussian .......................................................................... 105
2.21. Adelson and Bergen's Temporal Filters .................................................... 106

3.1. Robot Chassis and Imaging Equipment ...................................................... 120
LIST OF FIGURES – Continued

3.2. Robot Chassis and Control Hardware ........................................ 123
3.3. Parallel Computer ........................................................................ 124
3.4. Overhead Locator Pattern ............................................................ 127
3.5. Information Pipeline ....................................................................... 132
3.6. Pipeline Timing ............................................................................ 134

4.1. Velocity Space Key ....................................................................... 140
4.2. Visualization Template of Velocity Feature Space $\Omega_V$ .............. 142
4.3. 1D Collision Geometry ................................................................. 144
4.4. Dense Expansive Flow .................................................................. 145
4.5. Optic Flow Fields .......................................................................... 147
4.6. Compensation for Velocity Corrupted by Rotation ....................... 148
4.7. Translation Weights $W_T(\Omega_V, R = 0, \Omega_T)$ ......................... 150
4.8. Translation Weights $W_T(\Omega_V, R = .35, \Omega_T)$ ....................... 151
4.9. Translation Weights $W_T(\Omega_V, R = -.35, \Omega_T)$ ..................... 152
4.10. Visualization Template of Translation Feature Space $\Omega_T$ ........... 154
4.11. Temporal Dynamics of Translation Features ............................... 155
4.12. Rotation Weights $W_R(\Omega_V, R)$, First Component ................. 158
4.13. Rotation Weights $W_R(\Omega_V, R = .35)$, Second Component ....... 159
4.14. Total Rotation Weights $W_R(\Omega_V, R = .35)$ ......................... 160
4.15. Compensation for Aperture Problem .......................................... 163
4.16. Local Velocity Weights $W_V(\Omega_M, \Omega_V)$ ................................. 165
4.17. Weighting Network ..................................................................... 167
4.18. Spatial Gabor Filters .................................................................. 173
4.19. Temporal Frequency Response $H_t(\nu_t)$ of Motion Energy Filters 175
4.20. Approach Course Overhead View ................................................. 176
4.21. Approach Course Stimulus with Stripes Obstacle ....................... 178
4.22. Approach Course Stimulus with Segments Obstacle ................... 179
4.23. Approach Course Stimulus with Circles Obstacle ....................... 180
4.24. Ghost Image ............................................................................... 181
4.25. $\xi(t; \nu)$ from approaching the Stripes Obstacle ....................... 183
4.26. $\xi(t; \nu)$ from approaching the Segments Obstacle ..................... 184
4.27. $\xi(t; \nu)$ from approaching the Circles Obstacle ............................ 185
4.28. $\xi(t; \theta_M)$ from approaching the Stripes Obstacle ................... 187
4.29. $\xi(t; \theta_M)$ from approaching the Segments Obstacle ................ 188
4.30. $\xi(t; \theta_M)$ from approaching the Circles Obstacle .................... 189
LIST OF FIGURES – Continued

4.31. $\xi_M(t; v_x)$ from approaching the Stripes Obstacle .......................... 191
4.32. $\xi_M(t; v_x)$ from approaching the Segments Obstacle ......................... 192
4.33. $\xi_M(t; v_x)$ from approaching the Circles Obstacle .......................... 193
4.34. $\xi(t; \tilde{F}_V = (5.25), v_x)$ from approaching the Stripes Obstacle ............... 196
4.35. $\xi(t; \tilde{F}_V = (25.25), v_x)$ from approaching the Stripes Obstacle .......... 197
4.36. $\xi(t; \tilde{F}_V = 100, \Omega_t)$ from approaching the Stripes Obstacle ............ 199
4.37. $\xi(t; \tilde{F}_V = 150, \Omega_t)$ from approaching the Stripes Obstacle ............ 200
4.38. $\xi(t; \tilde{F}_V = 200, \Omega_t)$ from approaching the Stripes Obstacle .......... 201
4.39. $\xi(t; \tilde{F}_V = (5.25), v_x)$ from approaching the Segments Obstacle .......... 203
4.40. $\xi(t; \tilde{F}_V = (25.25), v_x)$ from approaching the Segments Obstacle .......... 204
4.41. $\xi(t; \tilde{F}_V = 100, \Omega_t)$ from approaching the Segments Obstacle .......... 205
4.42. $\xi(t; \tilde{F}_V = 150, \Omega_t)$ from approaching the Segments Obstacle .......... 206
4.43. $\xi(t; \tilde{F}_V = 200, \Omega_t)$ from approaching the Segments Obstacle .......... 207
4.44. $\xi(t; \tilde{F}_V = (5.25), v_x)$ from approaching the Circles Obstacle .......... 208
4.45. $\xi(t; \tilde{F}_V = (25.25), v_x)$ from approaching the Circles Obstacle .......... 209
4.46. $\xi(t; \tilde{F}_V = 100, \Omega_t)$ from approaching the Circles Obstacle .......... 210
4.47. $\xi(t; \tilde{F}_V = 150, \Omega_t)$ from approaching the Circles Obstacle .......... 211
4.48. $\xi(t; \tilde{F}_V = 200, \Omega_t)$ from approaching the Circles Obstacle .......... 212
4.49. $\xi(t; \tilde{F}_T = (5.25), v_T)$ from Approaching Stripes Obstacle ............... 214
4.50. $\xi(t; \tilde{F}_T = (25.25), v_T)$ from Approaching Stripes Obstacle .......... 215
4.51. $\xi(t; \tilde{F}_T = 100, \Omega_T)$ Activation from approaching the Stripes Obstacle .......... 216
4.52. $\xi(t; \tilde{F}_T = 110, \Omega_T)$ Activation from approaching the Stripes Obstacle .......... 217
4.53. $\xi(t; \tilde{F}_T = (5.25), v_T)$ from Approaching Segments Obstacle ............... 218
4.54. $\xi(t; \tilde{F}_T = (25.25), v_T)$ from Approaching Segments Obstacle .......... 219
4.55. $\xi(t; \tilde{F}_T = 100, \Omega_T)$ Activation from approaching the Segments Obstacle .......... 220
4.56. $\xi(t; \tilde{F}_T = (5.25), v_T)$ from Approaching Circles Obstacle ............... 221
4.57. $\xi(t; \tilde{F}_T = (25.25), v_T)$ from Approaching Circles Obstacle .......... 222
4.58. $\xi(t; \tilde{F}_T = 100, \Omega_T)$ Activation from approaching the Circles Obstacle .......... 223
4.59. $\xi_D(t; D)$ from approaching the Stripes Obstacle .......................... 225
4.60. $\xi_D(t; D)$ from approaching the Segments Obstacle .......................... 226
4.61. $\xi_D(t; D)$ from approaching the Circles Obstacle .......................... 227
4.62. Time to Collision Estimates $T_C(t)$ from Stripes Obstacle ...................... 228
4.63. Time-to-Collision Estimates $T_C(t)$ from Segments Obstacle ............... 229
4.64. Time-to-Collision Estimates $T_C(t)$ from Circles Obstacle .......... 230
LIST OF FIGURES – Continued

4.65. Depth Map from Approaching the Stripes Obstacle .................................................. 232
4.66. Depth Map from Approaching the Segments Obstacle ............................................... 233
4.67. Depth Map from Approaching the Circles Obstacle ................................................... 234
4.68. Turn Course Overhead View ....................................................................................... 235
4.69. Turn Course Stimulus with Stripes Obstacle .............................................................. 236
4.70. Turn Course Stimulus with Segments Obstacle .......................................................... 237
4.71. Turn Course Stimulus with Circles Obstacle .............................................................. 238
4.72. \( \xi_{1}/t = 100; \Omega_{1} \) Snapshot from Turning Stripes Obstacle .......................... 241
4.73. \( \xi_{1}/t = 150; \Omega_{1} \) from the Turning Stripes Obstacle ........................................... 242
4.74. \( \xi_{1}/t = 200; \Omega_{1} \) from the Turning Stripes Obstacle ........................................... 243
4.75. \( \xi_{1}/t = 100; \Omega_{1} \) Snapshot from Turning Segments Obstacle .......................... 244
4.76. \( \xi_{1}/t = 150; \Omega_{1} \) from the Turning Segments Obstacle ........................................... 245
4.77. \( \xi_{1}/t = 200; \Omega_{1} \) from the Turning Segments Obstacle ........................................... 246
4.78. \( \xi_{1}/t = 100; \Omega_{1} \) Snapshot from Turning Circles Obstacle .......................... 247
4.79. \( \xi_{1}/t = 150; \Omega_{1} \) from the Turning Circles Obstacle ........................................... 248
4.80. \( \xi_{1}/t = 200; \Omega_{1} \) from the Turning Circles Obstacle ........................................... 249
4.81. \( \xi_{R}(t; R) \) Activations from Turning Stripes Obstacle .............................................. 250
4.82. \( \xi_{R}(t; R) \) Activations from Turning Segments Obstacle .............................................. 251
4.83. \( \xi_{1}/t = 130; \Omega_{1} \) Rotation Activation for Stripes Obstacle ........................................ 252
4.84. \( \xi_{T}(t; \tilde{r}_{T} = (5.35), T_{1}) \) Activation for Turning Stripes Obstacle .................. 253
4.85. \( \xi_{T}(t; \tilde{r}_{T} = (5.35), T_{1}) \) Activation for Turning Stripes Obstacle .................. 254
4.86. \( \xi_{T}(t; \tilde{r}_{T} = (5.35), T_{1}) \) Activation for Turning Segments Obstacle .............. 255
4.87. \( \xi_{T}(t; \tilde{r}_{T} = (5.35), T_{1}) \) Activation for Turning Segments Obstacle .............. 256
4.88. \( \xi_{T}(t; \tilde{r}_{T} = (5.35), T_{1}) \) Activation for Turning Circles Obstacle .................. 257
4.89. \( \xi_{T}(t; \tilde{r}_{T} = (5.35), T_{1}) \) Activation for Turning Circles Obstacle .................. 258
4.90. \( \xi_{T}(t = 130; \Omega_{1}) \) Activation from Turning Stripes Obstacle ........................................ 260
4.91. \( \xi_{T}(t = 130; \Omega_{1}) \) Activation from Turning Segments Obstacle ............................... 261
4.92. \( \xi_{T}(t = 130; \Omega_{1}) \) Activation from Turning Circles Obstacle ............................... 262
4.93. \( \xi_{T}(t = 130; \Omega_{1}) \) Translation Activation from Turning Stripes Obstacle .............. 263
4.94. \( \xi_{D}(t; D) \) from Turning Stripes Obstacle .............................................................. 264
4.95. \( \xi_{D}(t; D) \) from Turning Segments Obstacle .............................................................. 265
4.96. \( \xi_{D}(t; D) \) from Turning Circles Obstacle .............................................................. 266
4.97. Time-to-Collision Estimates \( T_{c}(t) \) from Turning Stripes Obstacle ....................... 267
4.98. Time-to-Collision Estimates \( T_{c}(t) \) from Turning Segments Obstacle ....................... 268
4.99. Time-to-Collision Estimates \( T_{c}(t) \) from Turning Circles Obstacle ....................... 269
**LIST OF FIGURES – Continued**

4.100. Depth Map from the Turning Stripes Obstacle .............................................. 271
4.101. Depth Map from the Turning Segments Obstacle .............................................. 272
4.102. Depth Map from the Turning Circles Obstacle .............................................. 273

5.1. Control Partitions $R_C$ ................................................................. 277
5.2. Experimental Arena Overhead View ................................................................. 279
5.3. Waypoint Trial Overhead View ................................................................. 282
5.4. $\xi_{w}(t; R_C)$ Activation from Waypoints Trial ................................................. 284
5.5. Turn Control $C_T(t)$ from Waypoints Trial ......................................................... 285
5.6. Horizontal Avoidance Control Weights $W_{AX}(R_C, x_T)$ .................................... 288
5.7. Avoidance Control Weights $W_{CA}(R_C, x_T)$ .................................................... 290
5.8. Overhead Snapshots of Obstacles Trial ............................................................. 293
5.9. Obstacles Trial Overhead View ................................................................. 294
5.10. $\xi_{r}(t; R_C)$ Activation from Obstacles Trial ..................................................... 297
5.11. Turn Control $C_T(t)$ from the Obstacles Trial .................................................. 298
LIST OF TABLES

3.1. Computational Requirements ................................. 125
3.2. Hardware Resources .............................................. 125
3.3. Support Software .................................................. 129
3.4. Processing Modules .............................................. 129

4.1. Parameter Space Tunings ......................................... 166

B.1. Temporal Tuning $v_1$ from Approach Trials ............... 318
B.2. Temporal Tuning $v_1$ from Turn Trials ..................... 319
B.3. Temporal Tuning $v_1$ from Closed Loop Trials .......... 320
B.4. Velocity Speed Tuning $v_c$ from Approach Trials ....... 321
B.5. Velocity Speed Tuning $v_c$ from Turn Trials ............... 322
B.6. Velocity Speed Tuning $v_c$ from Closed Loop Trials ... 324
B.7. Translation Speed Tuning $v_T$ from Approach Trials ..... 326
B.8. Translation Speed Tuning $v_T$ from Turn Trials .......... 327
B.9. Translation Speed Tuning $v_T$ from Closed Loop Trials .. 328
ABSTRACT

This dissertation combines recent theoretical models from the neuroscience community with recent advancements in parallel computing to implement a large simulation that emulates the motion pathway of a mammalian visual system. The simulation ran in real time, and was used to perform real world obstacle detection and avoidance with an autonomous, mobile robot. Data is shown from experimental trials of the robot navigating an obstacle course in which the robot had both strategic waypoint finding goals and obstacle avoidance tactical goals that were successfully integrated into a single navigational behavior.

The simulator is distinguished from many previous robotics efforts due to its size and faithfulness to neuroscience. It employs population coded representations of motion energy (similar to brain area V1) and velocity (similar to brain area MT). In addition, it implements new features engineered to close the control loop between a mammal's early vision system and a mobile robot. Several problems routinely encountered in robotics experiments are discussed. Novel solutions to these problems are proposed that take specific advantage of the population coded representations of visual features, and implemented in the simulator.

These results are applicable to future engineering efforts to build special purpose circuits, analog or digital, for the purpose of emulating biological information processing algorithms for robotics and image understanding tasks.
CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

Many engineers during the last few decades have designed and implemented methods for machine vision and robotics that resembled natural biological systems with varying degrees of accuracy. The objectives of this type of work usually include solutions to real world problems and or the pursuit of new scientific understanding of biology and neural systems. This branch of computer science and engineering is one of the oldest, yet remains one of the most poorly understood relative to younger, yet more successful, engineering fields.

The performance, from an information processing point of view, of people and animals behaving in the real world greatly surpasses the abilities of even our most sophisticated engineered systems for seemingly simple tasks. These include basic navigation and obstacle avoidance tasks. The smallest of rodents can find food, escape capture, and explore its habitat with vision much better than any artificial circuit has demonstrated to date. The remarkable performance of a rodent in such a small, power efficient, self sufficient package is tantalizing evidence that an artificial machine could share those features, under the control of an engineer, to the benefit of mankind. Artificial machines do not have to function like a biological system to be useful, but there is obvious evidence that biologically inspired systems actually work better for many engineering problems that we want to solve. Moreover, there is a complete lack of evidence that these kinds of tasks can be performed in any other way.
Although the benefits of building information processing machines modeled after such biological systems are clear, the technical details of how to complete such a task are not. This field depends on the convergence of neuroscience and engineering. Neuroscience is the field concerned with the way biological systems process information. Electrical and computer engineering is concerned with the implementation of information processing systems. There are two major problems with creating artificial systems that behave like natural ones. First, neuroscientists do not yet understand in any detail how massive networks of neurons process information in such a way that gives rise to sensory understanding, navigation, locomotion, and cognition. At least partial solutions to these hard problems are required to engineer artificial systems. The last one, cognition, appears to be unsolvable in the near term. Secondly, engineers have developed many methods for processing information, including mechanical, optical, and electronic systems. However, none of those methods resemble the neural substrate within which biological systems work. The most glaring deficiency in our engineered systems relative to a brain is the lack of massive, three dimensional interconnects. Furthermore, biological interconnects transfer information in a discrete amplitude, continuous time, impulsive format that is foreign to all of our engineered systems, save primitive, special purpose analog circuits specifically designed to emulate neural “wetware”.

The emulation of biological systems could be argued to be the most computationally demanding problem ever to be presented to an engineer. The amount of raw computation required to simulate partially even primitive neural systems taxes our largest supercomputers. This is likely to remain true for the near future, but it will be shown here that we can now simulate portions of a neural system at a suitable level of abstraction for rudimentary image understanding and real time robot navigation.
This dissertation represents another step in the quest to engineer artificial systems that share desirable properties with naturally occurring ones. The last two decades have provided us with much new data and theoretical models in the realm of neuroscience, along with greatly increased computational power in the form of digital computers. The rate of progress is likely to continue increasing in both areas for the foreseeable future.

This dissertation will demonstrate the achievement of a modern digital implementation of contemporary neuroscience models associated with the mammalian vision system. The networks that have been implemented are still much smaller than their natural counterparts. However, it is shown that even with a small network, rudimentary image understanding, in the form of motion based obstacle detection, can be performed in real time. The relationship between the engineered implementation and natural systems is carefully described. The relationship is actually much closer than many other research areas with names that sound more biologically inspired than they actually are, for instance, so called "neural networks" and "genetic algorithms".

A particular feature of real neural systems, population coding, is used throughout this dissertation. Population coding is infrequently encountered in most engineering efforts because it requires more computation and interconnections than alternative methods. Nonetheless, this dissertation will show that population coding offers unique advantages for accuracy in image understanding.

The network's results, obstacle detection, will furthermore be used with a ground based robot platform to perform real time obstacle avoidance. The simulation associated with this dissertation is the first known example of a closed loop autonomous robot configuration that uses biologically inspired vision to avoid obstacles and has its network size as large as the one developed here while maintaining faithfulness to a real neural system. For many parts of a closed loop robot controller, current neuroscience
has nothing yet to offer in the way of inspiration. Basically, this is a consequence of the fact that nobody understands yet in detail how a real brain works. In portions of the described system, new methods were engineered to be compatible with the known neuroscience, but no claim is made that they accurately model any real biological system. The distinction between biologically inspired and purely engineering solutions is always made clear in the dissertation.

The final result of this dissertation is a robot platform successfully navigating a real world obstacle course using vision. No previous work has shown a similar result associated with a biologically inspired vision algorithm in a closed robot control loop. Data is shown at every computational step including image acquisition, motion computation, obstacle detection, and motor control. Portions of the system were built on prior art, but there are many novel components to the solution as well. An additional contribution of this work is the integration of the prior art and novel contributions into a complete system, implementing an autonomous robot that operates in the real world.

1.1 Robotics Review

The general field of robotics is both old and broad. Some applications have used robots successfully for many years, yet other applications have yet to materialize. In general, this is because some robotic applications are easier to build than others. This dissertation deals with (arguably) the hardest and most flexible class of robots: visually guided, autonomous, and mobile.

1.1.1 Robot Taxonomy

Robots can be fixed or mobile. Fixed robotic technology is already widely used in industrial applications. These robots, in general, do one specific task very well. Usually,
these robots consist of an arm or some kind of manipulator mounted to some fixed platform. They may or may not work in tandem with some kind of sensor. The class of robots studied here, on the other hand, are mobile. A mobile robot may still be built for some special purpose, but the application may require the robot to move around some environment to perform the task. A mobile robot has many extra concerns in addition to its primary function. Major issues include knowing where to go, recognizing when it has reached a location, and avoiding obstacles along the way. This dissertation concerns the avoidance of obstacles. An obstacle avoiding robot is a general purpose platform, upon which specific applications may be built. The application will determine where the robot wants to go and what it does when it gets there.

Robots can operate with various degrees of integration with a human controller. Teleoperated robots have long been deployed successfully in real applications such as hazardous environment cleanup and space exploration. These robots are basically remote control vehicles sending video or other information to a distant human operator, who then decides what the machine should do next and sends back commands for the robot to execute. On the other hand, autonomous robots operate exclusively under the control of their own algorithms: there is no human in the control loop. Outside of science fiction, there has been very little success building mobile autonomous robots, despite decades of effort and vast application potential.

A mobile, autonomous robot designer has many options for detecting environmental features like obstacles. These include sonar \cite{1,2}, laser range-finding \cite{1}, bump sensing, radar, special purpose environmental signatures of various types \cite{3}, infrared \cite{1}, and vision. In the realm of obstacle detection, sonar has met with the most success to date. Sonar is a mature, accurate technology with moderate data processing requirements.
Even some biological systems, such as a bat, use sonar for obstacle and target detection. Despite the limited success of sonar applied to robotics, visual based processing is potentially much more useful. Vision carries more tactical information than 1D technologies like sonar because it has a high bandwidth, two-dimensional format. Sonar, radar, and laser range-finding are essentially one-dimensional technologies unless one applies additional enhancements such as scanning techniques or detector arrays. Those technologies also radiate power, resulting in fundamental limitations on sensor range and vehicle power requirements. Vision, on the other hand, is passive. Range is limited only by the scene and environmental factors, and power requirements are limited only by the information processing technology. Bump sensors are of dubious utility in most applications, as they operate by detecting physical contact, which is exactly counter to the obstacle avoidance task. Special signatures include light tracking and various line-following techniques based on electromagnetic, chemical, or visual aids present in an environment that has been treated specially for robot navigation. Infrared is similar to vision, but it uses a different portion of the electromagnetic spectrum. Unlike vision, it also works at night, but can only see obstacles that radiate heat. Vision is a superior strategy as it does not suffer from any of the above problems. Unfortunately, vision requires large amounts of data processing, and engineers have not yet discovered how to extract adequate information for robot navigation, irregardless of computational limitations. However, the majority of biological systems have evolved to handle the data processing requirements with tremendous success. Casual observation of humans or almost any animal will show that vision enables sophisticated, fast behavior. Clearly, it is possible to extract and apply that visual information to behavior. Vision is the untapped “holy grail” of robotic navigation.

\footnote{Of course, many whisker-equipped animals have found bump sensors to be a useful sensation in certain environments, especially dark or cramped ones.}
Autonomous robot control systems often have some kind of memory. Different types of memory lead to different control strategies. Deliberative control schemes, strongly associated with traditional artificial intelligence (AI), use permanent or long term memory to predict the outcomes of considered actions. This memory often takes the form of a world model, upon which the whole control scheme is very sensitive. Stored information is usually symbolic. Computation tends to be complex, fragile, and slow. The other extreme is a reactive controller. Reactive controllers live in the moment, storing acquired information only briefly before it ages and becomes out of date. Reactive systems tend to have simpler computations, although there may be a lot of them. These robots are fast, and tend to avoid many tricky representational and semantic issues associated with their deliberative relatives. Reactive systems directly couple perception and action without an intelligent coordinator. Both navigation strategies, deliberative and reactive, are highly developed in people and animals. We can perform deliberative tasks such as finding an office across campus, while reactively dealing with obstacles along the way. This dissertation describes a reactive obstacle avoidance behavior. Goals would be determined by the robotic application and would probably happen in a deliberative way. In this dissertation, a set of arbitrary waypoints serve as simple deliberative goals.

This dissertation is "biologically inspired". This descriptor means that the information processing flow is modeled after biological systems. This approach is guaranteed to have a robust, high performance solution because we know our own brains work this way, and we routinely perform tasks that are beyond the state of the art in robotics. Unfortunately, we do not yet have good understanding of the information processing that happens in our own brains, and we may not for some time. Even if we did understand everything about the brain, then we still might not be able to synthesize an artificial one because our current favorite implementation technology, integrated semiconductor
circuits, may not be suitable for the task. In the meantime, however, we would like to be able to use the meager knowledge that we do have, with readily available integrated circuit technology, to enhance the performance of our artificial robots. From an engineering perspective, it does not matter if the robot is biologically inspired or not, as long as it performs some task better than other approaches. This introductory review, therefore, will consider both biologically inspired and conventional approaches to robotic navigation.

1.1.2 Popular Biological Sources

A few real biological vision systems are particularly well studied and applicable to robotic systems. Three popular systems include the visual systems of locusts, honeybees, and flies.

Locust "Looming" Detectors

Locusts have a neuron called the "lobula giant motion detector" (LGMD) that is sensitive to approaching objects. [4] Robots detecting obstacles with looming detectors have been proposed with explicit LGMD models [5] and conventional geometric models [6, 7].

Flow Balancing in Honeybees

M. V. Srinivasan has been a primary contributor to understanding how honeybees use balanced optic flow to stay in the middle of corridors in addition to other navigational behaviors. [8] Many corridor following robots have been described based on a variety of similar optical flow computations. [9, 10, 11, 12, 13]
Fly Motion Pathway

The fly's visual system has been studied [14] and results applied to robot navigation [15, 16].

1.1.3 Common Problems in Robotics Navigation

Many of the visually guided robots reviewed here had to deal with similar problematic issues summarized in this section. Many of these issues were not discussed in the original references, but should have been a concern nonetheless. This dissertation encounters and addresses all of these issues. These descriptions include citations of the papers that explicitly addressed the importance of each of them. The labels used to identify the issues are mostly new with this dissertation; the original sources described the problems but did not name them. It will be convenient in this dissertation to use these labels in many different contexts. They are briefly summarized here for the first time for background information and to provide some organizational structure for the following literature review. This list will be collectively addressed again in Chapter 6, in relationship to the new results of this dissertation.

Diffuse Regions

All visual algorithms somehow operate on the contrast of an image. Many real world environments have regions with low contrast due to nonideal lighting effects, or naturally lack texture by nature. These phenomena effectively blind a robot in that region of the scene. The most common example is a wall painted a uniform color. In general, robotic imagers do not have the dynamic range to see the small bumps and globs of paint that humans can see, so the wall simply appears blank, and no motion or distance estimates can be made there. In this situation, a wall is indistinguishable from
open space. Unfortunately, diffuse walls are just as dangerous to a navigating robot as brightly colored ones. Ways to compensate for the blindness caused by diffuse regions include some kind of momentum information carried over from some previous moment when there was texture [12] or equipping the robot with some heuristic that recognizes the situation and acts in a special way [10].

Aperture Problem

The aperture problem, described in Section 2.2.2, is arguably the most significant problem in machine vision and robotics, including this dissertation. Briefly, the aperture problem is a geometrical phenomenon that introduces potentially large errors in visually derived motion estimates. These errors can cause havoc in a robot controller. Ways to handle the problem include discarding [9,7] or constraining [12] suspicious optic flow vectors using some heuristic, treating them as obstacles [7], or computing sparse optic flow only at those dynamic locations immune to the aperture problem [17].

Motion Bandwidth

Optic flow algorithms require some kind of time history to compute motion. This history must have some reasonable, unaliased, temporal structure for these algorithms to function properly. Robots can easily induce motion too fast for these algorithms by moving forward rapidly. However, problems usually occur because the robot turns too fast, introducing large rotational flows. Most optic flow algorithms break if the objects in the scene are moving more than 1.5–2 pixel frame. The easiest way to deal with this problem is to slow the robot, especially in turns. Other methods include performing some coarse to fine scale pyramid processing scheme [18].
**Rotational Flow**

Optic flow ideally is a result of robot translation, providing egomotion and obstacle information to the machine. Unfortunately, when the robot turns it induces a large rotational flow that can easily swamp the smaller translational flow that contains obstacle information. This problem has been addressed by turning very quickly relative to frame rate [10], only turning when the robot is stopped [3,7], rotating the imager on some kind of “neck” as the robot turns (gaze stabilization) [19,20], estimating the rotational flow and subtracting it from the whole flow [17], camera placement [12], or just disabling obstacle detection during rotations [5]. The gaze stabilization strategy usually involves two types of motion. There are slow camera rotations that counter the chassis rotation, and fast saccadic returns to a neutral position when the neck has turned too far. Coombs et al. called this strategy *nystagmus* [20].

**Symmetric Flow Anomaly**

The corridor following robots reviewed here work by equalizing optic flow on the right and left visual peripheries similar to honeybees. This heuristic works in corridors, but unfortunately cannot distinguish the intended situation from headings straight towards walls, or straight into corners. [9,10].

**Parallax Blind Spot**

Optic flow of purely translational motion is zero at the focus of expansion by definition. This creates a blind spot for optic flow based algorithms in the region. Unfortunately for navigation algorithms, this blind spot is directly in front of the robot. Literature that explicitly considers this failure mode does so by never actually heading directly towards a target [3,19,17], or by not considering the region at all [7].
Sensory and Behavior Integration

Any system that uses multiple sensed feature types to control a single behavior must integrate those features in some way. Likewise, basic behaviors must be integrated in some way for complex behavior to emerge. Poor control integration schemes lead to oscillations as the robot keeps changing its mind about which behavior should dominate at any given moment. Another problem occurs when the robot tries to satisfy two mutually exclusive behaviors simultaneously, heading in some averaged direction that breaks both behaviors.

Bumpy Ride

A camera mounted on a moving robot inevitably records every bump and sway the robot experiences as a rapid global shift of the scene. This noise is usually addressed by temporal filtering.

Body Clipping

Robot imagers have limited fields of view, and real robotic platforms consume some finite amount of floor space. A common problem is a corner of the platform colliding with an obstacle after the obstacle has left the camera’s field of view and is no longer generating flow patterns.

Computational Latency

Related to limited frequency response in control systems, computational algorithms in robots take some finite time to compute. Depending on hardware or software architecture, behavior is based on environmental input that is out of date to some degree. This latency contributes to a slow control loop. This problem is particularly acute in
this dissertation, because the amount of computation is higher than in any other robot in this review. In order to get high throughput, computation is done in a pipelined fashion which has the unfortunate consequence of increasing the latency to several frames. It will also be shown that the computation proposed in this dissertation contained extra problematic latencies related to temporal filtering that had to be addressed.

**Remote Control**

Many robot classes reported to date cannot support the weight or power requirements of the computer required for information processing. Exceptions are neuro-morphic efforts and large robots. In typical cases, however, the computer has to sit somewhere else and the robot is tethered to it using either a real umbilical wire, or some kind of radio transmission. [3,16] This is not an interesting research issue, but is a common practical feature.

**Location Estimation**

When experimenting with navigating robots, it is convenient to know where the robot is at any given instant. For quantitative experiments, this is a critical issue. There are several approaches described in this literature review. The easiest is just to accept that you don't know where the robot is, and just report qualitative data. [20] Slightly better information is recorded by videotaping the performance and manually transcribing the experiment later. [10] Better methods include the use of some kind of dead reckoning scheme by measuring odometric information [12,20,17], or by integrating optic flow [23]. These last two methods automatically generate quantitative data, but tend to accumulate large errors as the robot operates due to wheel slippage or other mechanical nonidealities associated with odometry. In many instances it has been found
that the best way to automatically collect accurate data is with an overhead camera. The main drawback of this method is that the robot cannot move beyond the overhead camera's field of view without corrupting the acquired data. This dissertation used a overhead camera. A Global Positioning System (GPS) receiver could potentially provide information similar to an overhead camera without the coverage problem, but unfortunately it does not have enough spatial resolution for indoor robotics experiments. One researcher in this review attached a pen to his (small) robot so it drew its path on a sheet of paper which was later scanned into a computer for analysis. Yet another approach uses magnetic field techniques to accurately locate the robot and its orientation in all six degrees of freedom. The drawback of this technique is that the device, sold by the Polhemus company, is expensive and its range is limited, even more than a typical overhead camera setup.

1.1.4 Robotics Literature Review Summary

Even in the subclass of visually guided, mobile autonomous robots, there are almost as many approaches as researchers. The literature reviewed here only includes work related closely enough to be considered competitive, precursor, or complimentary in some way. In general, robots perform some very limited set of behaviors that happen to correspond to something known from biological sources. There is no existing robot that does vision based obstacle avoidance in real world environments with any sort of generality.

Much existing literature only considers simulated environments; hence it is not considered here. Although synthetic stimuli are used in this dissertation for debugging and theoretical insight, real visual input is the only stimulus that adequately demonstrates the robustness of a technique. Most of the hardest issues in obstacle avoidance are
29

consequences of real world nonidealities like noise, jitter, limited aperture, and low contrast. Consequently, it is actually easier to just use real imagery than try to synthesize a visual world realistic enough for useful performance measurements.

Some work that may be considered competitive to the results of this dissertation for a limited subset of potential applications include those techniques that require some internal representation of the robot's environment. [1, 2] This may include anything from explicit maps provided by some external source, like a human operator who measured out the room by hand, to some kind of learned representation using genetic algorithms or neural networks to tune an environmental model. All of these techniques are limited to environments that are either known in advance, or can be learned in some kind of "training" phase before the robot actually starts performing a useful task. Although this limitation is acceptable in some situations, these strategies are not useful for the more general class of completely unknown environments, or known environments with dynamic obstacles. These techniques are mentioned here because they show deceptively good performance, but are inadequate for general applications in a fundamental way.

In general, it is difficult to compare quantitatively the performance between different approaches and research efforts. There is a sensitive tradeoff between any performance metric and prior information fed to the robot. Comparison is therefore made difficult because it is hard to quantify the prior information in a useful way: for example, how much is an explicit world map worth? [1] The prior information generally results in success or failure of the algorithm, and makes the technique completely unusable in harder or unknown environments. Computer or analog processing power is also highly variable in the literature. Any useful quantitative comparison between two published results must account for varying amounts of computational resources that might result from computer vintage, architecture, or developer skill and time invested. Mobile platforms
themselves are variable, with many confounding physical characteristics like minimum and maximum speed, turning radius, drive mechanism, and so on. Robots with these real world limitations are described as non-holonomic. Yet another factor contributing to the lack of good comparisons is the lack of standard navigational and obstacle avoidance benchmark tasks. Every researcher seems to have a unique figure of merit convenient for them, and many of them are purely qualitative. For instance, simply moving around the lab without hitting anything may be considered a success. Similar problems arise due to unrepeateable experiments, such as those requiring a human to hold and move a target. [10] These types of experiments can hardly be considered quantitative. In light of these considerable obstacles to real quantitative comparisons, a researcher is generally limited to qualitative comparisons with other work.

1.1.5 Traditional AI Based Robots

Traditional AI based approaches to robotics are based on the sense-model-plan-act paradigm. This is a serial processing stream of environmental sensing, modeling of the world based on the sensor data, plan a course of action based on the world model, and finally perform some action. Many current researchers have rejected this idea because it results in slow, fragile robots that often do not work at all. However, some researchers still report success using these methods. Only one robot is reviewed here, because there are not very many that work well, and those that do have not strongly impacted this dissertation.

Wolfram Burgard et al.: University of Bonn. Carnegie Mellon University

Burgard, et al. developed an “Interactive Museum Tour-Guide Robot”. [1] This was a massive effort, with multiple research institutions involved and a very broad technical
scope. Their robot did obstacle avoidance, among many other things, using a number of sensors: a laser range finder, sonar, and active infrared. The robot had no visual capabilities. This work is reviewed here because it is the only instance of a publically deployed system of any type, and dealt with a fully general, dynamic environment. It gave museum tours with hundreds of people milling around. There was no biological inspiration in the robot’s algorithms; they were based on artificial intelligence methods. It had a preprogrammed, adaptable map of the museum that it used in conjunction with sensor readings for both long range navigation and obstacle avoidance. The system was complex, but impressive because it successfully negotiated the hardest environment of any robot in this review, in addition to interacting with the museum visitors. Specific comparisons of Burgard et al.’s robot with the work presented in this dissertation are not particularly useful because of the map requirements and the lack of visual input. However, this robot represents the state of the art of the traditional AI paradigm that other robotic strategies have to outperform to be useful in the long run. It is interesting to notice that as full featured and high tech as this robot was, it had no visual capabilities at all.

1.1.6 Behavior Based Robots

Behavior based robotics use tight coupling between perception and action. World models are not used, avoiding a whole host of problems. Processing is very specialized to perform specific tasks well. This is the largest category of existing robots, and has demonstrated the most success from a robotics point of view.
Rodney Brooks: MIT AI Lab

Rodney Brooks and his students were active from approximately 1985 to 1991 with their “behavior-based” robotics ideas and the implementation of their “subsumption architecture”. Before Brooks, mobile robots processed information using the traditional AI chain of perception, modeling, planning, task execution, and motor control. This resulted in complex processing stages with strong interactions between the modules. Difficulties in path planning, for instance, was approached by improving an earlier stage like environmental perception. In addition, this strategy was vulnerable to catastrophic failure should any of the stages break. Brooks proposed a tighter coupling between sensors and actuators, and removed the traditional intelligent global controller that integrated all inputs and decided on action. Instead, he created much simpler processing modules that did specific tasks in a hierarchy of levels. Each module operated in parallel, and they did not communicate with each other. In the subsumption architecture, the output of a low level task could be overridden, or “subsumed”, by some other task. For example, the robot might be performing a random wander behavior, occasionally subsumed by an obstacle avoidance behavior operating in parallel. In this case, the “wandering” module never knew it was being overridden. Brooks was successful in integrating multiple goals, achieving a form of robustness to partial system failure or erroneous input. He built several mobile, autonomous robots that are described in a recently published collection of papers. [21] He never used vision.

Brooks was the first roboticist to break away from the traditional AI paradigm, and his work resulted in the first new robotic strategy. As such, Brooks and his associates influenced most of the ensuing robotics work, including this dissertation, in inspiration if not the details. The notion of basic modules operating in parallel, integrated into a single overall behavior is used in this dissertation. The advantages of incrementally
expanding and debugging a robot with small modules, first made possible by Brooks’ ideas, were critical in completing this dissertation in a timely manner. This dissertation differs from any of Brooks’ work in the details: his robots were not biologically inspired, nor did they use vision.

**Liana Lorigo: MIT AI Lab**

Rodney Brooks has been quiet on the robotics front for the last ten years or so, but his student Liana Lorigo published in 1997. The robot associated with this paper worked with multiple related features of static vision, based on the heuristic that close objects appeared lower in the visual frame than far objects. The bottom region of each frame was always assumed to be safe. Qualitative data showed general success at avoiding obstacles in tested environments including a student lounge and a JPL Mars simulator sandbox. Martin continued this idea as described below.

Lorigo’s work is closer to this dissertation than Brooks’ earlier work, so broad qualitative comparisons can be made. Similarities include vision based processing, the lack of world model, and compatibility with subsumption architectures. Lorigo’s robot maintained very little state, sometimes called a “reflective” architecture, unlike this dissertation which, based on visual motion, required short term temporal storage. Lorigo’s robot was prone to errors when the heuristic was invalid, such as in the presence of floor texture and specularities, and shadows. In this dissertation, the confusing effects of these anomalies are reduced because the motion associated with them is the same as the object upon which the texture or shadow is projected. Such features actually enhance performance of the robot in this dissertation, because shadows and any other contrast generators break diffuse regions and enable better and more dense motion estimates. Another advantage of this dissertation’s result is that obstacles do not
necessarily have to rest on the ground for accurate detection. This common situation includes desks, chairs, and bushes. Lorigo also had trouble with body clipping.

**Martin Martin: Carnegie Mellon**

Martin’s work extended Lorigo’s method of searching for obstacles by finding the lowest feature in vertical columns of an image. The extension was to perform more sophisticated image processing with genetically discovered programs. As in similar genetic programming work, he used a library of low level operations arranged in individual programs, and evolved the programs to achieve better fitness to hand tuned, known obstacle positions. This robot actually had two modes of operation. In the first mode, used to collect training data, the cameras collected imagery while a ring of sonar detectors actually detected the obstacles. Next, the collected imagery was hand labeled to generate a training set of data and known truth, and a large offline evolution was performed to find the best expression to estimate the truth based on the input imagery. Finally, the robot navigated the same environment using the camera and final evolved program, in the second, operational, mode.

Martin’s robot avoided unknown obstacles like chairs and people, although it had sensitivities to camera orientation and high contrast specularities on the floor. This problem was similar to that reported by Lorigo, as were other limitations of the previous work including the assumption that the floor is flat and that all obstacles rest directly on the floor. The training phase required for Martin’s robot limits the potential applications. As he discussed in the paper, online training is not feasible because it is too slow and requires human supervision. On the other hand, offline training requires some kind of simulated input which is also fraught with problems. The need to implement a bootstrap control scheme using sonar is a disadvantage, as is the need
to have access to the environment for the training phase. Hand labeled ground truth is only possible for small environments, and there is no evidence or reason to believe that the evolved programs are suited to any environment other than the one used for training. Finally, the evolution phase required before the robot can actually operate is prohibitively expensive for many applications. For instance, in Martin's case the evolution phase took twenty hours.

Other than dependence on vision, Martin's method had very little in common with the robot developed in this dissertation. Genetic programming is far removed from true biological inspiration. despite the subject title. This paper is reviewed here because it is brand new work, and clearly shows just how difficult visual obstacle detection is to the computer science community, even in state-of-the-art systems.

Andrew Duchon, William Warren, and Leslie Kaelbling: Brown University

Duchon, Warren, and Kaelbling's robot [10] used the outputs of an established non-biological optic flow algorithm [26] integrated in a flow "balance strategy" similar to that proposed for honeybees. [8] This method integrated optic flow on each side of the focus of expansion, and turned the robot in the direction of lesser flow. Objects close to the robot generated larger flow vectors, indicating their presence for obstacle avoidance. Data was reported from hand analysis of videotapes of the robot trials. Results are qualitative, with one of the robots moving around a laboratory environment for "25 minutes without collision".

Duchon et al. basically ignored the rotational flow problem by performing fast "body saccades", in reference to the fast large movement an eye makes called a saccade. The robot moved straight forward most of the time to avoid rotational flow, but when a course correction was required it performed a large rotation quickly; ideally, in the time
of a single frame. This robot encountered difficulties with diffuse regions, such as the
darkness under tables and other zones of low contrast. It was enabled with “emergency
reflexes”, a set of heuristics, to recognize and handle these situations. The symmetric
flow anomaly was reported and this problem was handled with another heuristic.

This dissertation shares a similar control law that basically integrates optic flow
estimates for obstacle avoidance. Many of the same problem issues are addressed,
such as diffuse regions and rotational flow. Differences abound in the details, such as
information coding, optic flow implementation, and handling of failure modes.

**Zigzagging Robots**

Sobey addressed the parallax blind spot problem by making his robot not head
directly towards the target, but rather “zig-zag” towards it. The diffuse regions
problem was handled by deliberately heading towards obstacles; if it knew where an
obstacle was, then the robot could move with confidence at least that far without hitting
anything. Hopefully, the contrast situation was more favorable from the new vantage
point. This “zigzagging” strategy was inspired by flying insect navigation.

Sobey’s robot ran on older computer hardware, and could not process motion in real
time. Sixteen frames were gathered before the robot had to stop and wait for offline
processing. Even then, flow was only computed along a single horizontal row. Goals
were either explicit locations found by dead reckoning, or searching for a light source.
Direction was determined by the “potential fields” method, where environmental objects
including targets exerted an attractive or repulsive force. Lewis [19] used the same
“zigzag” idea, but had access to a larger computer. The main differences were the optic
flow computation, which was a very crude temporal derivative, and the replacement
of Sobey’s potential fields with a voting scheme. Lewis’ robot computed in real time.
and therefore had to deal with rotational flow. He used gaze stabilization for this by rotating the camera on the robot with saccadic returns.

The zigzagging robots share similarities with this dissertation in that obstacle avoidance is performed using optic flow estimates. The control strategies are different, however, in addition to the optic flow computation. The zigzag behavior is inspired by insect vision, unlike this dissertation which works more like human vision. In human motivated processing, the parallax blind spot would be better addressed by using stereoscopic disparity rather than zigzagging behavior.

**David Coombs: NIST**

David Coombs, Martin Herman, Tsai-Hong Hong, and Marilyn Nashman built a robot that integrated vision based corridor following, obstacle avoidance, and gaze stabilization behaviors for navigation in a real office environment. The first version of this robot did not include obstacle avoidance, but the centering behavior was biologically motivated by honeybees. The same centering behavior still operated in the new version, but the obstacle avoidance was based on traditional differential optic flow and flow divergence. Centering behavior was achieved by balancing the largest optic flow component in two large receptive fields: left and right of the focus of expansion. This is similar to, but subtly different from the approach taken by Duchon, and shares the symmetric flow anomaly. The centering behavior leaves a large blind spot directly in front of the robot, that is reduced by the flow based obstacle avoidance behavior. The centering and obstacle avoidance behaviors required different fields of view, so two asymmetric cameras were used to accommodate this requirement. Both of Coombs et al.'s robots used gaze stabilization with fast saccadic returns. Because gaze stabilization was employed, no other attempt was made to compensate for rotational
flow. This caused difficulties when the robot hit bumps causing rotational transients faster than the gaze stabilization could track. The first robot required much human hand-holding to avoid obstacles, but the second version navigated the office environment without collision for “20 minutes” at “30 cm second”. Quantitative paths are shown, but data was collected with a corrupted dead reckoning scheme.

This work is functionally the closest robot in the existing literature to this dissertation. The implementation is totally different; but both robots avoid obstacles based on time to collision estimates from optic flow and use similar motor control equations. Significant differences in the implementation include the optic flow computation, gaze stabilization strategy, and number of cameras. Coombs et al. used non-biological computational strategies including finite state automata and lookup tables that helped achieve real-time performance on a smaller computer, but their approach goes against the philosophy embraced in this dissertation. Many of the same problems are dealt with (or tolerated) in this dissertation, including diffuse regions, rotational flow, computational latency, and body clipping.

**Guilio Sandini and José Santos-Victor: University of Genova**

Guilio Sandini and José Santos-Victor reported a robot that also used an optic flow balancing honeybee inspired algorithm for corridor following [11] and, later, a robot that added obstacle avoidance and docking behaviors [12]. This robot handled diffuse regions with a “sustained” behavior that assumed the optic flow was the same as it was before the contrast disappeared. The rotational flow problem was addressed by their “divergent stereo” camera placement pointing sideways like a honeybee, not forward like most predators. In this arrangement, rotational flow was minimal in the tested corridor-like environments. Obstacles were avoided in a non-biological way by assuming that
the robot was operating on a flat surface, and that anything not part of that plane was an obstacle.

Similar in relationship to this dissertation and to Coombs et al., Sandini and Santos-Victor's robots avoided obstacles using a control strategy based on integrating optic flow estimates. Differences include information coding strategies, optic flow computation, and control laws.

**Marc Ebner and Andreas Zell: Universität Würzburg, Germany**

Mark Ebner and Andreas Zell built another visually guided, corridor following robot that worked by balancing optic flow [17]. This work was similar to Coombs et al. [20] and Santos-Victor et al. [12], except that it used a nonlinear foveated mapping and the optic flow computation was sparse. Flow was computed only at special locations picked by an "interest" operator which were identified as not susceptible to the aperture problem. This interest operator was determined using genetic programming. The navigation strategy was influenced by Sobey [3] in that the robot headed towards known obstacles in order to avoid uncertain places.

Ebner and Zell's work is again different from this dissertation in the optic flow computation and control laws, although it shares the dependence on vision. The strategy of computing optic flow at sparse locations that move is not particularly amenable to hardware implementation in silicon or wetware, so this idea is not compatible with the philosophy of this dissertation.

**Hartmut Neven and Gregor Schöner: Ruhr-Universität, Bochum, Germany**

Hartmut Neven and Gregor Schöner built a robot that used optic flow to estimate the time to contact for obstacle avoidance [7]. The focus of their work was the integration
of homing and obstacle avoidance behaviors. This integration was achieved through the use of control space attractors. The control vector of the robot was found to be near to these dynamical attractors, with the attractors being able to move around an abstract space as the environment was sensed. Each behavior was represented as an attractor in this space. Integration therefore consisted of finding the strongest attractor. Control oscillations were avoided with hysteresis between the attractors. This strategy was very effective at handling noisy optic flow estimates.

This robot is unique in this review in many respects. The way this work handles unreliable flow estimates and integrates behavior appears to be the most robust strategy in the review.

1.1.7 Biologically Inspired Robots

Robots described in this section are closely related to specific biological systems.

Mark Blanchard, Claire Rind, and Paul Verschure: University of Zurich

The robot of Blanchard, Rind, and Verschure used a neurobiological model of a locust "looming" neuron to visually detect approaching obstacles. In addition, information was represented by spiking population codes in both the vision and control algorithms. The 20×20 input photodetector array used in their robot drove both excitatory and inhibitory cell types that were integrated in the computation of the "looming" detector. Their artificial implementation was closely related to that of a locust, making this work the closest thing to a real animal reviewed here.

The focus of this work was the study of their looming detector neural model. This emphasis exposed some undesirable features from the roboticists' point of view. The biggest limitation was that the robot is actually an infrared based machine, with the
visual looming feature bolted on later to detect "far" obstacles, leaving the infrared detectors intact to look for "close" obstacles. The test environment, although real, was severely constrained without presenting any really hard obstacles or diffuse regions. Rotational flow was handled using lateral inhibition in their network. This implementation presumably blinded the obstacle detector during rotation. The looming detector did not provide any bias right or left for obstacle avoidance; the robot arbitrarily turned left when the detector fired.

The work of Blanchard, Rind, and Verschure shared some important similarities with this dissertation. It is the only other working robot described in this review that used biologically motivated vision algorithms to detect obstacles. Processing occurred in neurally inspired layers, with outputs of one layer integrated in the next to extract navigationally relevant features. Information was passed between these layers using population codes. Important differences with this dissertation follow from their locust inspiration. Their motion detecting neurons were less sophisticated than motion energy detectors, corresponding to the simplicity of insect versus mammalian vision. Their population codes were actually closer to real biological systems, as they modeled the temporal dynamics with integrate and fire neurons. Here, information is represented as averaged rate codes. Their network size was considerably smaller (1800 neurons, 6000 interconnections vs. over 50,000 neurons used here) making that coding strategy more computationally tractable. Strategically, their work was motivated by the need to justify their neural model, where this dissertation is more concerned with engineering robots, using biological inspiration for performance reasons. Experimentally, the test environments were similar in that there were high contrast obstacles deliberately placed in front of a no-clutter background, and an overhead camera monitored the experiment for location estimation.
The Neurosciences Institute

Krichmar, Snook, Edelman and Sorns built a robot to study perception and behavior in a large neural model. The robot was actually just an application for their neural simulator, so detailed data on robotic performance was de-emphasized in favor of their network training algorithms and the behavior of the model elements. The network was not modeled after any particular organism, although it was very biological in nature. It had over nine thousand computational elements with almost 700,000 synaptic connections. Each “wire” represented the rate encoded output of a set of several neurons. The most interesting aspect about this work was how the robot learned, using plasticity and learning rules, a “conditioned response” based on previous experiences. After conditioning, the robot avoided targets that “tasted” bad on previous encounters.

In the example application of a robot, “taste” was defined to be the conductivity of small blocks placed around the environment, each with a high contrast visual pattern on it. The robot moved around, tasting randomly placed blocks, until it learned which ones it liked and which ones it did not based on the visual patterns painted on the blocks.

From an obstacle avoidance perspective, this machine was a routine application of sonar. Vision was a large component of this work, but it was used only to discriminate targets for tasting, not obstacle avoidance. Even within the visual target detectors, most visual difficulties were removed by painting the whole environment black, so that only the target blocks stood out. Overlap with this dissertation is light given the different goals, except the neurally inspired architecture. Krichmar et al. had results that could contribute to future robots along the lines of this dissertation, if adaptation or learning were to be considered.
Damper, French, and Scutt: University of Southampton, UK

Damper, French, and Scutt constructed a biologically inspired autonomous robot that learned to avoid obstacles based on the shadows they cast. [31] The architecture of the robot was based on a well motivated neural architecture with several spiking neuron types and plastic connections. The focus of this paper was the unsupervised adaptation of neural circuits. The robot began life with an unconditioned reflex to back away from obstacles detected with a bump sensor switch mounted on the chassis. Infrared sensor neurons fired upon nearing the obstacle. Nonetheless, the robot had to learn the relationship between infrared neuron activity and bump “pain” sensors before it developed a conditioned response based on the infrared detectors and avoided actually bumping into obstacles. A second order conditioned response was also developed based on the shadows cast by obstacles.

This work shares many similarities to Krichmar et al. in purpose, although the implementation was totally different. The relationship to this dissertation include biologically inspired networks. However, Damper et al. modeled their neurons at a lower level (spiking) and concentrated on plasticity and learning simple behaviors. In contrast, this dissertation concentrates on more sophisticated behavior with hardwired networks.

1.1.8 Neuromorphic Robots

Neuromorphic Engineering grew out of Carver Mead’s efforts at Caltech in the late 1980’s. [32,33] Although Mead himself is no longer working in the robotics field, there remains a world wide distributed effort continuing his ideas. Virtually all of the researchers involved have some association with The Computation and Neural Systems Group at Caltech, either as professors, graduates, postdocs, or students of the same.
The basic tenets of Neuromorphic Engineering are that biology points the way to the best robotic algorithms, and that subthreshold, full custom analog VLSI naturally implements biological inspired processes. The first idea is a result of frustration with traditional robotics having never produced anything approaching the performance of most interesting biological systems, and the second with the frustration that digital computers are too slow, expensive, heavy, and power hungry for biologically motivated information processing strategies. Ironically, very few neuromorphic robots have appeared in the fifteen years since Mead started the field. As promised, neuromorphic circuits have proven to be fast, small, and power efficient, but the design cycles are so tedious and slow that most work has been limited to special purpose chips with limited capabilities. Many of them are vision chips [34], although there is other work with audition and motor control [33]. Only complete robots are reviewed here, and their capabilities were quite limited. All the vision chips used in the robots reported in this review are one dimensional arrays of detectors, all of which would have to be generalized to two dimensions for most realistic applications.

Reid Harrison and Christof Koch: Caltech

Harrison’s robot used a neuromorphic visual motion sensor to control an unstable wheeled chassis [15]. This work was explicitly inspired by fly vision. Like a housefly, his chassis was unstable in an open loop configuration. Harrison forced this condition by using a pair of asymmetric drive motors. Consequently, the closed loop control system had to use vision for stabilization. A one dimensional array of motion detectors was integrated into a single output, measuring wide angle motion across the entire visual field. The integration smoothed the random distribution of environmental features indicating rotational motion present when the robot was unstable. The resulting behavior of the
robot was to move robustly straight ahead without the right uncontrollable spinning behavior that occurred when this particular chassis ran open loop.

Like most of the neuromorphic literature, Harrison’s work was necessarily dominated by custom analog VLSI development efforts. Real hardware such as this always provides solutions to a variety of tricky issues that arise in real world environments, and real custom circuitry. Such circuitry always comes with size and power advantages over software and digital implementations. Unfortunately, hardware this specialized is not particularly useful by itself in a general purpose robot. In Harrison’s work, the hardware supported the course stabilization task described, but not much else. It is unclear how these types of extremely specialized circuits can be generalized to harder tasks like obstacle avoidance.

The relationship of Harrison’s work to this dissertation is limited due to the super-specialized behavior. It is reviewed here because similar analog circuitry is an eventual contender for faster, lighter, and larger implementations of the ideas in this dissertation, that so far have been implemented only in software.

**Ralph Etienne-Cummings: Johns Hopkins University**

Etienne-Cummings’ work in general is hardware oriented and based upon focal plane information processing. His hardware published to date [35] had both analog and digital components; it broke slightly away from the hardcore, all analog neuromorphic efforts. As in this dissertation, he has also remarked on the prohibitively slow pace associated with the development of custom VLSI hardware for each robotic application. Keeping with his hardware emphasis, however, he is currently pursuing general purpose, reconfigurable analog hardware as a solution instead of the large computers as done in this dissertation.
Etienne-Cummings has published results obtained with a real robot that used his focal plane processor and could do line following while simultaneously avoiding obstacles. The robot used a pair of focal plane processors, each of which computed the "motion centroid" of its respective scene. In the absence of obstacles, the centroid outputs were integrated in an antagonistic way such that the line was out of both fields of view, and presumably in the middle, between the two. An obstacle was detected by observing the centroid in a "zone" of the visual field where the line should not be. In this situation, the robot temporarily abandoned the line following task and turned away from the obstacle, after which the line was re-acquired. Because the focal plane processors could only detect one centroid at a time, this strategy would presumably fail in the presence of more complex environments in which multiple centroids would need to be detected simultaneously. This robot was really just a demonstration of Etienne-Cummings' current focal plane chip. It is believed that he has plans for more sophisticated robotic behavior as his chip matures.

The details of Etienne-Cummings' robot and this dissertation are quite different, although they share some common high level features including vision, control by feature integration, and explicit biological inspiration. Strategically, Etienne-Cummings is interested in hardware that may be used in robotics, where this dissertation is interested in robotics that may eventually be implemented in hardware. As such, Etienne-Cummings has a real hardware system, but his control strategy is based on a fragile, application specific heuristic. This dissertation does more sophisticated information processing and can handle more general environments, but requires a large general purpose computer. Other work of Etienne-Cummings and Van der Spiegel of particular relevance to this dissertation is a hardware implementation of a modified Adelson Bergen motion energy filter. [36]
University of Zurick

Giacomo Indiveri and Paul Verschure built very simple line-following robot that used a neuromorphic vision front end for edge detection. [37] The brightest edge was detected, sampled, and then used in a very simple digital control circuit. Jörg Kramer and Indiveri also built a simple optic flow based corridor follower [13]. It used the popular honeybee inspired [8] optic flow balancing method. These papers were light on details, and both robots required special environmental conditions to operate properly: lines on the ground, or treated corridor walls. The most relevant result of these two robots to the work presented in this dissertation is the superior robustness to different ambient light conditions of the neuromorphic retina relative to a CCD detector. This desirable feature is characteristic of the popular Delbrück and Mead photoreceptor circuit. [38]

Marinus Maris built a line following robot similar to Indiveri and Verschure, except that he added a neuromorphic “attention” circuit enabling the robot to select a direction when the line forks. [39] Attention was granted to the fork that was more brightly illuminated, simulating a “power source”.  

Ralf Möller built an analog implementation of insect navigation that guided a small robot. [24] The emphasis of this work was on long range navigation, and does not consider obstacle avoidance along the way. Being a hardware implementation, algorithms were much simpler than in this dissertation. In particular, the “retina” was really just a 1D array of photodiodes that fed very simple edge detection circuits, and the environment in which the robot operated was also very simple. The results of this work support theories of insect navigation that do not overlap the goals of this dissertation. However.

---

[2]The classic idea of a robot finding a light source goes way back to Grey Walter’s 1953 “cybernetics” work with vacuum tubes. [40].
the computation shares some similarities in that there are simple vision circuits, feature integration, and explicit biological inspiration. The navigation model of Möller and the obstacle avoidance strategies of this dissertation do not exclude one another, but actually address different types of navigation. An interesting experimental technique was used to make the results quantitative: the robot was small, and a pen was mounted on it and marked its path on paper lying underneath the robot as it moved around in its environment. The paper could then be scanned so the robot paths could be treated quantitatively.

1.2 Neuroscience Review

Many researchers have looked to neuroscience to improve the performance of engineered machines for general information processing [32], visual processing [41], and robot behavior [21, 42]. Roboticists have claimed "biological inspiration" for decades, always looking for those elusive natural mechanisms that contribute to the superior performance of biological systems. The research fields of artificial intelligence, neural networks, neuromorphic engineering, expert systems, genetic algorithms, and behavior-based robotics all have some relationship to neuroscience, although the link is sometimes remote. Clearly, claiming "biological inspiration" by itself does not sufficiently describe the relationship of engineering work to real neurobiology. This section reviews the neuroscience literature of typical biological systems such as Fig. 1.1(a) in terms of algorithms, data representation, and implementation, in order to understand how those principles could be used in engineered systems such as Fig. 1.1(b).

Motivating the need for visually controlled behavior similar to biological systems in robots is easy. [42] All one has to do is observe animals and people behaving in their environments to see that they routinely perform tasks that no autonomous machine has
ever done. Unfortunately, implementing a robot by reverse engineering animals is made difficult by incomplete knowledge of how biological systems process information. What a robotics engineer would really like to have from the neuroscience community is a known functional decomposition of the physiology of all the visual processing circuits in at least one interesting animal like a cat, monkey, or human. Unfortunately, experimental neuroscience is not likely to provide such a description any time soon, due to the difficulty of simultaneously measuring more than a handful of the millions of neurons participating in the computation. However, it has been possible to experimentally measure the response of certain types of neurons in limited numbers. This data has enabled the proposal of computational models for certain mammalian visual processes suitable for implementation in a robotic system.

A task considerably simpler than discovering the physiology of visual processing circuits is a physical description of the neural “wetware” that implements them. This is possible because we can simply look at animals to discover
neural components, and apply more sophisticated techniques that show in more detail how neurons are connected.\[46\] There is an important distinction between the physical and functional decompositions, however, because many neural components perform multiple types of computation, and some computations are spread over multiple components. Although the details of these processes are still open research issues, many experiments support theories that certain neural components are primarily responsible for certain computations in visual systems.\[47\] In general, these theories are quite specific and quantitative for the retina itself, and become less so as the visual information propagates into layers deep in the brain. Components of the brain described in this chapter include areas V1 (primary visual cortex or striate cortex), MT (Medial Temporal Cortex), and MST (Medial Superior Temporal Cortex). It is convenient to abuse this morphology by assigning specific information processing functions to each area, because engineering a complex system such as a robot requires just such a functional breakdown. Analogs presented here, therefore, represent a simplified theory and do not account for many complexities that real neuroscientists have discovered. On the other hand, these analogs do represent a much closer relationship to real neuroscience than other robotic methods like artificial intelligence \[48\] and so called “neural networks” \[49\].

In biology, “vision” includes the detection of many types of features from optical input. These features are all computed separately, and integrated to influence behavior in many complex ways. A functional decomposition of these modalities is shown in Fig. 1.2. \[46\] No single animal has evolved all of these modes, although humans have all of them except polarization. Behavior is in general an extremely broad topic, but in this dissertation it specifically refers to obstacle avoidance. Within this larger context, the scope of this dissertation is the motion processing modality and how it can be used to influence obstacle avoidance behavior in a mobile robot. However, nothing in this work
precludes the integration of these additional modalities in future robots. In particular, there are compatible models of stereoscopic disparity [27] that could be used to compute depth and compliment the motion based depth computation used here. Such integration of multiple visual features could increase robustness of the application, and extend the types of environments a robot could navigate.

Figure 1.2: Visual modalities evolved in natural systems. Phototransduction is the process of converting light to a biophysical or electrical quantity usable by the information processing system. Form is the shape and location of objects. Color and polarization are characteristics of incident light. Motion is objects changing spatial location with time. Depth, often associated with two eyes, provides a visual estimate of how distant an object is.

1.2.1 Implementation Strategy

Data processing systems, both biological and engineered, maintain internal representations of the information relevant to their particular tasks. These may be short term (instantaneous) representations in the case of a reactive system, or some kind of longer term memory in the deliberative case. In animals and robots, these include environmental features used to navigate and avoid obstacles. Because these agents cannot
maintain physical replicas of the world inside of them, some other mechanism must be used to represent these features. There are two components to a representation: the format and the implementation. The distinction is important for biologically inspired robots because the format is something engineers can copy from neural systems, but the implementation is not.

Implementation

Neural systems implement information processing circuits biophysically. This neural substrate, or "wetware", operates with chemical and electrical mechanisms, and supports primitive computational neurons with dense interconnections. A large part of neuroscience is devoted to this interesting topic. Unfortunately, engineers have no technology capable of fabricating wetware, so they have to resort to purely electronic circuits that are crude by comparison, especially with respect to interconnect capability. In biology, a single neuron may accept input from 10,000 other neurons and send input to another 10,000 neurons. No engineered hardware efficiently supports such density and parallelism. On the other hand, individual electronic circuits can operate much faster than their biological counterparts because the physical processes support higher bandwidth. Not all circuits are faster, however, and most are actually slower. General purpose digital circuits (i.e., microprocessors) are particularly poor for neural-like implementations due to their serial architecture and complex switching requirements. Neuromorphic engineering (Section 1.1.8) is a response to this limitation that results in fast analog circuits, although dense interconnects are still a problem. Address Event Representation, a digital multiplexing scheme, addresses this issue in the neuromorphic context. As discussed in Section 1.1.8, neuromorphic circuits are time consuming and tedious to build, which hinders the timely development of complete robots. Another
approach, used in this dissertation, is to tackle the parallelism with multiple microprocessors. Although not ideal in terms of power, weight, and speed, they support more flexibility and faster development cycles than custom hardware. The flexibility and rapid development of the robot are critical factors to this dissertation: the power and weight of the implementation do not affect the results at all.

Format

The format of neural information describes the coding and associated semantics of the physical quantities stored in the implementation. The biophysical details of real neural systems, summarized in Section 1.2.2, describe a neural implementation. This wetware implementation is not naturally implemented in available engineering technologies. In this dissertation, biological coding methods are copied, but the implementation is designed to be compatible with standard engineering methods. Thus, the overall strategy of this dissertation is to compute neural algorithms like a biological system, implementing the computation electronically instead of biophysically. The neuroscience literature important to this work therefore deals with computation: the biophysical literature influences this work very little.

1.2.2 Biological Information Processing

This section describes a general architecture of biophysical vision systems. A robot does not need to emulate a particular animal in all respects: it is possible to choose desirable features from many organisms and combine them into some abstract artificial being. However, evolution has discovered many alternative strategies that all work well in certain situations, and choices have to be made designing an artificial system. In this dissertation, inspiration is taken primarily from mammals like cats, monkeys, and
humans. This section shows this bias, and therefore neglects other strategies considered by other roboticists, like insect vision, \[15\]. Even among cats, monkeys, and humans, detailed neuroscience requires different language because the biology is different between the animals. When required to be specific, this dissertation uses language and strategies relevant to human biology.

Neural systems are large networks of interconnected neurons. Neurons are the basic computational unit, or cell. There are many types of neurons, even within visual systems. A particular neuron only has a single logical output that is broadcast to the many destination neurons. Biophysical mechanisms \[51\] inside the neuron result in the integration of inputs to compute some value, represented at the output by discrete electrical spikes, or action potentials, that fire in some temporal pattern. Information passed between neurons is represented by some code, which defines the semantics of the temporal pattern. Neurons are organized in layers. Layer outputs, consisting of many individual neuron outputs, have their own semantics beyond the individual components. This section describes individual neuron coding, neural organization, and layer semantics called a population code.

**Rate Coding**

A rate code, illustrated in Fig. 1.3, represents a continuous output value as an asynchronous sequence of spikes, firing at a rate proportional to the encoded value. The dominant theory of neural computation is that all the information transmitted by a neuron is contained in the firing rate, and that the individual spike timings are random (Poisson distributed).\(^3\) Spikes are convenient biophysically, but from an information

\(^1\)A relatively recent development in neuroscience is the idea that temporal relationships between neuron spike trains also code significant information. The phenomenon of neurons *synchronizing* has been shown both in the laboratory \[52\] and numerical models \[53\]. The information contained in this synchrony, if any, is still controversial. The implications for biologically inspired robotics, however, are
processing point of view they are an implementation detail. If this theory is true, then engineered machines may be implemented without actual spikes, using some other coding better suited for the available technology, without losing the "magic" of biologically inspired information processing. In this dissertation, the output of a neuron is represented by a floating point number instead of a spike train. The value is logically equivalent to a biophysical neuron output, but the implementation is different.

![Figure 1.3: A rate code represents a value as a sequence of Poisson distributed, random spike timings. The average spike rate encodes the value.](image)

**Neural Organization**

Biophysical neurons are organized in layers. An example for the visual system is shown in Fig. 1.4. Each disk represents a neuron. The neurons are retinotopically organized in layers. In the figure, the organization is highlighted by connecting lines. At each retinotopic location, there is a *hypercolumn* of neurons. A hypercolumn is described in Section 3. This particular example shows three layers with the same retinotopic organization, but this does not have to be the case. As an example in potentially vast. The current work does not address synchrony, but only considers the traditional view that all the information is contained in average neuron firing rates, or rate codes. Should synchrony prove to be important, then a data format more rich than a simple scalar value will be required for artificial machines.
this dissertation, images are acquired at 100 × 100 pixel resolution, but features are computed at lower resolution to save computational expense.

An example of the connections between neurons in different layers is shown in Fig. 1.5. In this figure, the bottom layer of Fig. 1.4 is reproduced along with a single hypercolumn of the middle layer. The connections of the bottom layer that feed to the hypercolumn are shown, a feature lacking from Fig. 1.4 to avoid clutter. The bottom layer neurons that connect to the hypercolumn consist of the hypercolumn's receptive field. Notice that an adjacent middle layer hypercolumn (not shown), assuming it has the same relative interconnection topology, shares input from the lower layer.

**Information Representation and Population Coding**

Most neurons in the visual system operate as feature detectors. A feature can be a specific value associated with one of the visual modes shown in Fig. 1.2, or some higher level combination of lower level features. One of the simplest feature detector neurons in the form pathway is a line detector. A “line” is an abstract concept, however, and real neurons cannot detect abstractions directly. In order to fully specify the line, it has to be associated with a particular retinotopic location, orientation, and polarity (light to dark or vice versa). It also matters if the edge ends in the neuron’s receptive field, or exists as a continuous entity. Four different lines are shown in Fig. 1.6 to illustrate this concept. The circular outlines represent the borders of a receptive field, beyond which a particular neuron cannot see. Each line in the figure is different from the other lines in the parameter spaces of orientation, contrast, or length. Combined with a retinotopic location, the lines in Fig. 1.6 are each concrete features detectable by a neuron. Any particular neuron detects only one feature of a given type, although it could detect multiple types simultaneously. If the neuron detects its associated feature, it will fire.
Figure 1.4: Visual processing occurs in retinotopically organized layers. The disk stacks represent hypercolumns of feature detectors.
Figure 1.5: Receptive field topology. Neurons take input from lower layers. The neurons that feed another neuron make up the destination neuron’s receptive field.
rigorously, corresponding to a high floating point value in the artificial implementation. Otherwise, the biophysical neuron will remain silent, or the artificial one will output a small value.

Figure 1.6: Line features have parameters including orientation, contrast, and length.

In traditional engineering, the lines of Fig. 1.6 could be specified by measurements in three parameter spaces: orientation, contrast, and length. This is not how biophysical systems work, however, as each feature detector is sensitive to only one point in the parameter space. The parameter space is sampled by a set of related feature detectors as in Fig. 1.7. Here we have a retinotopically placed hypercolumn, with each neuron in the hypercolumn sensitive, or “tuned”, to a different point in the orientation parameter space. A hypercolumn, therefore, is a set of feature detecting neurons that sample some parameter space. Thus, the stacks of Fig. 1.4 are used only because it is hard to visualize on paper the 4D parameter space of \( (x, y) \) retinotopic location, layer, and orientation tuning. In reality, the hypercolumns may span more than one tuning dimension, as will be the case for motion processing. The collection of neuron outputs \( \{p_1, \ldots, p_N\} \) \( (N = 6 \) in this case) is called a population code. [54] Notice that \( \{p_1, \ldots, p_N\} \) have implicit orientation associations that downstream information processing mechanisms must already know, because the orientations are not otherwise indicated by the population code.
Figure 1.7: The population code \( \{p_1, \ldots, p_6\} \) represents the orientation of a line by sampling the parameter space in six places and associates feature detector outputs at each sample.

The population code is an alternative data format to representing the line with a simple radian orientation measure. They are not equivalent, however. The population code represents a quantized orientation measurement, and is more expensive to compute and store on traditional machines. The population code is capable of representing multiple features simultaneously, unlike the scalar measurement. It is well suited to the biological feature detection strategy, instead of some other method that directly computes the orientation of the line. It is a natural fit for parallel computation strategies, and computations can survive the loss of individual members of the code, or at least degrade gracefully instead of the sudden catastrophic failure that occurs if a scalar measurement fails. This dissertation is concerned with the implementation of visual processing systems using parallel computing architectures, so the high expense of serially computing population codes is mitigated. Thus, population codes are used throughout.
Motion Transforms

Feature detectors extract salient information from input, and discard everything else. A layer of neuron feature detectors therefore transform a rich population code input to a less rich, more specific, output code. In a particular visual pathway, information is transformed several times as the salient details from the raw photodetector input is filtered and converted to a format useful for influencing specific behaviors. In the form pathway example from Fig. 1.7, line detection could be an early step in detecting more complex geometrical shapes such as circles and squares, which in turn eventually lead to complex form recognition of items in an organism's real world environment. In this dissertation, motion is the visual pathway used for obstacle detection. This section describes the transforms and features encountered along the motion processing pathway in humans.

The basic morphology of the motion pathway in humans is shown in the top row of Fig. 1.8. The retina includes the eye, and processing that happens before information reaches the brain via the optic nerve. The components V1, MT, and MST of the brain, each described below, perform vision computations that include the motion pathway, among others. These neural components are described in this section. There are many other components in the visual system not mentioned here, because they are not modeled in this dissertation.

Retina

The retina is the part of the eye on which images are projected from the lens. In addition to phototransduction, cells in the retina also process visual information both spatially and temporally before sending it to the brain via the optic nerve. In this dissertation, the first computational neurons (V1 simple cells) are modeled by a linear integration
Figure 1.8: The top row shows the morphology of the mammalian early vision motion pathway. The bottom row shows the analogous engineering function performed in the artificial implementation.

of photoreceptor outputs. Because the cortical model used in this dissertation operates directly on the photodetector outputs, the biophysics of the retina does not influence this work. However, the organization of the photodetector layer is a critical factor in the performance and computational expense of the processing system.

A biophysical retina is contrasted to a typical CCD based circuit in Fig. 1.9. The large circular outline of Fig. 1.9(a) shows the extents of a retina that have been transformed to lie flat on the page. The small dots show non-uniformly distributed photoreceptor locations. In humans, there are over 100 million of these cells. The retina is *foveated*, which means there is a higher density of photoreceptors in the center of the retina, along with increased computational resources in the brain devoted to that area throughout the visual pathways. Electronic circuits, on the other hand, are typically square arrays of pixels as shown in Fig. 1.9(b). Technological constraints limit the pixel density, resulting in the square layout for maximum resolution and sensitivity. In electronic systems, it is possible to warp the resulting imagery by using an optical front end, resulting in a nonlinear mapping from the world to the image plane. This technique can simulate a fovea, even though the photodetector array is uniformly distributed.
Another way to simulate a fovea is to use a nonuniform distribution of the imager's pixels, and throw away the rest. In any case, even the densest imaging technologies available to engineers cannot approach the density of a biophysical retina, even at the edges where the resolution is the lowest. Even if we had such a technology, increased computational complexity would hinder the utility for robotic systems. Figure 1.9 makes it clear that engineers cannot compute imagery with the same acuity as the higher mammals given current technology. However, it will be shown in this dissertation that a crude CCD based imager is sufficient for basic robotic navigation tasks using visual motion.

**Figure 1.9:** Natural mammalian retinas have dense photoreceptor cells with a very dense fovea in the middle. Electronic photodetector arrays are crude in comparison, due to technology limitations. Linear receptive field examples are shown, with shading representing positive and negative weights.

Examples of "center surround" and V1 simple cell receptive fields are shown in Fig. 1.9(a) and Fig. 1.9(b), respectively. The ellipse in Fig. 1.9(a) represents the receptive field of a particular retinal neuron. The dark interior section shows an *excitatory* region, and the light section shows an *inhibitory* region. The receptive field operation
is a weighted integration, where inhibitory regions are negative weights, and excitatory regions are positive weights.

**V1 and Motion Energy**

The primary visual cortex, or V1, is thought to perform several functions, but the one relevant to the motion pathway is the computation of motion energy. Motion energy is related to image velocity, and will be the input to the MT model. The quantitative definition of motion energy is deferred to Section 2.2. Qualitatively, motion energy is a measurement of spatiotemporal (retinotopic space and time) frequency bands, non-linearly combined to eliminate phase and contrast dependencies. The computation in V1 is performed in two stages: linear *simple cells* which effectively perform a linear integration of photodetector outputs, and nonlinear *complex cells* that eliminate phase and contrast dependence.

The artificial implementation computes motion energy with the well established model proposed by Adelson and Bergen [43], which is itself inspired and supported by physiological experiments on V1 simple and complex cells [56]. Essentially, this operation is two spatiotemporal filters followed by a nonlinear combination into motion energy.

**MT and Optic Flow**

The Medial Temporal Cortex, or Area MT, integrates motion energy input from V1 to compute velocity features. These features are similar to optic flow from the computer vision literature, except that it is population coded. The implementation described here is similar to the proposal of Grzywacz and Yuille [45].
MST and Time to Collision

The Medial Superior Temporal Cortex, or Area MST, integrates MT outputs into large flow features. There is no physiological evidence for a feature as high level as time-to-collision being computed here, but that is what the model in this dissertation does. The velocity integration performed is novel, with no established neuroscientific model involved.

1.3 Chapter Summary

This chapter started by motivating the need for a biologically inspired, vision-based, autonomous robot. It next reviewed the broad field of robotics, and defined the specific area into which this dissertation fits. Previous work most closely related to autonomous, visual, or biologically inspired robotics was reviewed. The basic neuroscience that contributes to this work was described. The next chapter will be a quantitative discussion of the basic neuroscience as it will be applied to the robotic implementation described in this dissertation.
CHAPTER 2
MAMMALIAN EARLY VISION

This chapter begins a quantitative discussion of motion processing as computed by the artificial vision implementation presented in this dissertation, and reviews the existing literature upon which the later chapters will build. The primary topics of this chapter are the motion energy model of Adelson and Bergen [43] and the velocity model of Gryzwoicz and Yuille [45].

There are several stages of feature detection inspired by the biological strategy described in Section 1.2.2. Each stage transforms its input into new features more specific to motion detection, eventually computing velocity and an estimate of time-to-collision of environmental obstacles. This chapter describes some basic ideas already existing in the literature. In Chapter 4, real world stimuli will show how the basic models fail in the robotic application, along with novel solutions to those problems.

2.1 V1 Spatial Primitives

The first processing stage of this dissertation is analogous to spatiotemporal processing performed by V1 simple cells. Current models of V1 and MT such as those reported by Adelson and Bergen [43], Watson and Ahumada [60], and Simoncelli and Heeger [44] are based on a different functional theory than that proposed by early physiologists like Hubel and Weisel [61]. The latter have traditionally dominated computer vision studies. The earlier theories propose that images are decomposed by the visual
system into primitives in V1, and that the primitives include line and edge features. This strategy has found its way into computer vision and has driven an obsession with edge detection. It turns out that many neuroscientists now believe, based on new physiological evidence, that the actual V1 image primitives are not edges, but localized bands of spatial frequency. Palmer writes a nice qualitative review of the experiments that led to and support this theory. Basically, this theory proposes that the image primitives represented in V1 are Fourier components of image features, a concept very familiar to engineers.

2.1.1 Fourier Components

In image processing, the basis function of a 2D Fourier transform is a complex sinusoidal grating

\[ I(x, y) = e^{j2\pi(x\xi + y\eta)} \]  \hspace{1cm} (2.1)

where \( \xi \) and \( \eta \) are the spatial frequencies in the \( x \) and \( y \) dimensions measured in cycles per pixel. In a computational vision context, it is convenient to recast the spatial frequency variables of Eq. (2.1) in terms of an orientation angle \( \theta_M \) and the spatial frequency magnitude \( \psi \):

\[ \xi = \psi \cos(\theta_M) \]  \hspace{1cm} (2.2)

\[ \eta = \psi \sin(\theta_M) \]

Because orientation and spatial frequency are more intuitive and are commonly found in the neuroscience literature, they will be two important parameter dimensions examined in this dissertation. The subscripts \( s \) indicate spatial parameters, in contrast to the temporal parameters that will appear later. With the substitution of Eq. (2.2), Eq. (2.1)
becomes
\[ I_b(x, y; \theta_M, \psi_x) = e^{j2\pi\psi_x \cos\theta_M x + \sin\theta_M y} \]  
(2.3)

The Fourier transform of Eq. (2.3) is
\[ \mathcal{F}\{ I_b(x, y; \theta_M, \psi_x) \} = \delta(f_x - \psi_x, \theta - \theta_M) \]  
(2.4)

where \( \delta(f_x, \theta) \) is the Dirac delta function, and \((f_x, \theta)\) is a polar coordinate in the spatial frequency domain. Eq. (2.3) defines a good basis function for image decomposition because the simple frequency domain representation can be easily scaled, shifted, and integrated into an arbitrary image using basic linear systems theory.

2.1.2 Windows and Receptive Fields

Although mathematically convenient, Eq. (2.3) is not directly suitable for an image processing primitive because it is not physically realizable, and does not account for the biological notion of a receptive field. Physical cameras and retinas have real output, and finite support in \( x \) and \( y \). In this dissertation, the visual field is always a regularly spaced square array with size \( R_y \times R_x \) square pixels. Real output requires an even frequency domain representation along the line \( \theta = \theta_M \), and the finite support effectively multiplies \( I_b(x, y; \theta_M, \psi_x) \) with a rectangular window:

\[
I_R(x, y; \theta_M, \psi_x) = \begin{cases} 
I_b(x, y; \theta_M, \psi_x) & : |x| \leq \frac{R_x}{2}, |y| \leq \frac{R_y}{2} \\
0 & : |x| > \frac{R_x}{2}, |y| > \frac{R_y}{2}
\end{cases}
\]  
(2.5)

where the \( b \) subscript has been changed to \( R \) to show that the new grating is physically realizable because it has been truncated with a two-dimensional (2D) rectangular window. Figure 2.1(a) shows a plot of \( I_R(x, y; \theta_M, \psi_x) \) with a gain of 1 and a particular \( \theta_M \).
The effect of this window is seen in the frequency domain plot of Fig. 2.1(b) as rippled smearing around the main two impulses, parallel to the coordinate axes. The ripple pattern, a result of a 2D sync function, is the Fourier transform of the rectangular window. This phenomenon is common in signal processing, and various other windows have been proposed, all with better frequency domain characteristics at the expense of bandwidth. [63] These alternate windows are all qualitatively similar in that they damp the signal near the edges of the extent. One possible window function is the Gaussian

$$g(x, x_M) = \frac{1}{\sqrt{2\pi \sigma_x}} e^{-\frac{(x - x_M)^2}{2\sigma_x^2}}$$

with width $\sigma_x$ and center $x_M$, shown in Fig. 2.2 as the dashed envelope. A Gabor function is a Gaussian modulated sinusoid

$$I_G(x; x_M, v_z) = \frac{1}{\sqrt{2\pi \sigma_x}} e^{-\frac{(x - x_M)^2}{2\sigma_x^2}} \cos(2\pi v_z x)$$

which is also plotted in Fig. 2.2 as the solid line. Gabor functions are commonly found in image processing, compression algorithms (wavelets are related to Gabor functions), and computational vision.

Figure 2.3 shows a 2D Gabor function. In the spatial domain, it is shown that the Gabor function has finite support in $x$ and $y$, in addition to being localized to a small region of the visual field around some location $x_M$. In the frequency domain, the sinusoidal impulses have been spread by the Gaussian window frequency response, which is also Gaussian with bandwidth $1/\sigma_x$. The envelope of the 2D Gabor function, a 2D Gaussian

$$g(x, y; x_M, y_M) = \frac{1}{2\pi \sigma_x^2} e^{-\frac{(x - x_M)^2 + (y - y_M)^2}{2\sigma_x^2}}$$

(2.8)
Figure 2.1: Plot of a grating function $I_R(x,y)$ with $\theta_M = \pi \delta$ rad and $\psi_\theta = 10$ pixel$^{-1}$. The image has $(R_y \times R_z) = (100, 100)$ pixels.

Figure 2.2: Plot of a 1D Gabor function with $\sigma_y = 15$ pixel$^{-1}$ and $x_M = 0$ pixel.
implements a receptive field at retinotopic location $\xi_M = (x_M, y_M)$ with width $\sigma_x$. Real receptive fields are not always circular as in the Gabor case, but a Gabor function is a good model of V1 simple cell receptive fields in the motion pathway. Biophysical systems implement receptive fields with a neural interconnection topology. The window function $g(x, y; \xi_M)$ and a similarly shaped biophysical receptive field are logically similar information processing operations, but they are implemented in different ways.

![Spatial Gabor Function](a) (b) Frequency Response

**Figure 2.3:** Plot of a 2D Gabor function with $\theta_M = \pi/8$ rad, $\nu_x = 1/10$ pixel$^{-1}$, and $\sigma_x = 10$ pixel.

### 2.1.3 Spatial Feature Detectors

Gabor functions can also act as impulse responses of finite impulse response (FIR) bandpass filters. Figure 2.3 was developed as a primitive image component that might be detected by V1 simple cells. In an engineering context, the same function also works as a spatial feature detector if another image is correlated with it. Thus, a narrow band
of spatial frequencies. Fig. 2.3(b) can be detected using a 2D spatial filter with a Gabor impulse response.

A correlation effectively shifts the kernel function

\[ I_G(x, y; \Omega_M) = g(x, y; \vec{r}_M) h(x, y; \Theta_M, \phi) \]

where \( \Omega_M = \{ \vec{f}_M, \theta_M, \phi \} \) (\( \psi_r \) is introduced later) includes the spatial tuning of the filter. \( R_x R_y \) times and computes a weighted integration at each pixel. In most image processing approaches, the kernel function is stored once and the correlation is computed pixel by pixel, taking the shift \( \vec{r}_M \) into account at each location. This filter implementation is not suited to biophysical processing because such systems cannot continually rewire connections, as is effectively done in most software correlation implementations. Instead, real neural systems store many copies of the kernel function, each one hardwired with a built-in \( \vec{r}_M \) shift. Stated from a neuroscience point of view, a neuron’s receptive field has a retinotopic location at \( \vec{r}_M \). The neurons that implement the kernel functions all operate in parallel, with fixed connections to the retina. The \( \vec{r}_M \) offset, therefore, is an important parameter associated with a particular neuron. The other parameter that specifies the receptive field of a particular neuron is the width \( \sigma_s \).

The notion of using Fourier components as V1 spatial primitives is not inconsistent with early physiological data that supported line or edge detecting neurons. If the Fourier based theory is true, then Hubel and Weisel probably found a neuron that fired in the presence of an edge because they happened across a V1 simple cell with a Gabor like receptive field organization that corresponded to a harmonic of their edge stimulus. For instance, Fig. 2.4 shows an edge stimulus and its frequency response. Viewed in the frequency domain, it is clear that the edge stimulus is a superposition of multiple
Gabor features by basic linear systems theory. The Gabor filter of Fig. 2.3(b) covers the fundamental of the edge stimulus, and would fire rigorously when stimulated by the edge. The Fourier theory then does not preclude the presence of edge detecting neurons somewhere in the visual system, but they probably do not exist in V1 as originally hypothesized. Instead, they would be somewhere downstream in the visual processing flow implemented as an integration of V1 simple cells, essentially performing a Fourier synthesis operation. Regardless of biological truth, Fourier decomposition is useful in an engineering context because arbitrary features can be detected using such an integration, not just simple edges. In particular, a motion energy computation (Section 2.2) depends on just such an integration.

\[ f(x) \]

Figure 2.4: This edge detector has the same configuration as Fig. 2.3, except it has edges instead of being a Gabor function.
2.1.4 Implementation

Now that Gabor filters have been motivated with biological arguments, an efficient engineering solution may be designed for their computation in an artificial vision processing system. The computational results should resemble biophysical outputs, but the artificial implementation needs to be fast and well suited for the purposes of this dissertation. Receptive fields are not placed at every pixel of the input stimulus $S(x, y)$ because that would be unnecessarily expensive. Instead, receptive fields are placed in a sparse manner as shown in Fig. 2.5. These $m$ locations are members of

$$M = \{\mathbf{x}_M : 1 \leq M \leq m\} \quad (2.10)$$

where $\mathbf{x}_M = (x_M, y_M)$ are the retinotopic spatial coordinates of receptive field location $M$. In this dissertation, only uniformly distributed receptive field organizations are considered, although there may be opportunities for optimization in future robots by generalizing the distribution as real organisms do with foveation.

V1 simple cell responses are computed by a linear correlation of $S(x, y)$ with Gabor receptive fields $I_G(x, y; \Omega_M)$. As such, there are two ways to perform this operation on a serial computer, neither of which is the parallel method used by biophysical systems. The first is a 2D spatial correlation

$$\xi_G(t; \Omega_M) = \sum_{y=-\frac{\mu_y}{2}}^{\frac{\mu_y}{2}} \sum_{x=-\frac{\mu_x}{2}}^{\frac{\mu_x}{2}} S(x, y, t) I_G(x - x_M, y - y_M; 0, 0, \theta_M, \psi) \quad (2.11)$$

and the second uses a frequency transform

$$\xi_G(t; \Omega_M) = \mathcal{F}^{-1}\{\mathcal{F}\{S(x, y, t)\}\mathcal{F}\{I_G(-x, -y; 0, 0, \theta_M, \psi)\}\} \quad (2.12)$$
Figure 2.5: V1 simple cell receptive fields, and downstream processing, are less dense than the input stimulus $S(x, y)$. In this figure, the bottom layer represents $S(x, y)$, the middle layer represents the V1 simple cells, and the top layer is a downstream computation that uses V1 simple cell outputs. Connections, because they are dense, are not shown for clarity (see Fig. 1.5).
that has been subsampled at the receptive field locations $f_M$. Normally, an engineer would use Eq. (2.12) because the availability of the Fast Fourier Transform (FFT) results in an operation of order $2(R_x \log_2 R_x + R_y \log_2 R_y) - R_x R_y$ instead of $R_x^2 R_y^2$. However, two special considerations exist for this application. First, the computational expense of Eq. (2.11) is not $R_x^2 R_y^2$ in this case, but $R_x R_y C$, where $m \leq R_x R_y$. These savings are the result of the sparse sampling in $M$. Second, the exponential tails of the Gabor envelope $g(x, y, f_M)$ are small for potentially large regions of $I_G(x, y; \Omega_M)$ and, therefore, do not necessarily have to be computed. These properties have two unfortunate consequences: the total computation size is dependent on the receptive field width $\sigma_z$, and the truncation of $I_G(x, y; \Omega_M)$ results in an extra extent parameter to consider and the addition of noise correlated with $I_G(x, y; \Omega_M)$. Nevertheless, these two optimizations result in faster computations with Eq. (2.11) for some networks than with Eq. (2.12), particularly small ones. However, many of the experimental results reported in this dissertation were gathered using the largest possible network that could be run on the available computer platform. As Eq. (2.12) works best for those large network configurations, yet still supports smaller simulations, and does not introduce new parameters and noise sources, it was used here exclusively.

2.2 V1 Motion Primitives and Motion Energy

V1 complex cells perform a nonlinear integration of V1 simple cell responses. The outputs of these cells represent *motion energy*. This section describes motion energy and how it is computed by the artificial implementation.
2.2.1 Spatiotemporal Orientation

Motion is a visual phenomenon that necessarily occurs in both space and time. Figure 2.6(a) shows a 2D image of a 10 pixel bar with an arrow suggesting motion to the right. Figure 2.6(b) shows another representation of the moving bar, this time plotting the 2D image as a function of time. A spatiotemporal function such as this one is similar to a spatial image, but has dependence on both time and space.

![Figure 2.6](image)

**Figure 2.6:** A 10 pixel bar is moving right at a rate of $v_x = 70$ pixel/s. The specific numbers are arbitrary examples. Figure is after a qualitative plot from Adelson and Bergen.

Figure 2.7(a) shows the $x$ and $t$ dimensions of Fig. 2.6(b). In this contrived example there is no independent variation in $y$; consequently, a 2D spatiotemporal plot $(x$ and $t$) effectively conveys the same information shown in Fig. 2.6(b). In real imagery, and the implementation reported here, there generally is an independently varying $y$ dimension that must be considered.

Frequency based optical flow methods [28] and motion energy models [43, 44] operate by detecting spatiotemporal orientation. Figure 2.3 shows that a 2D Gabor function is
Motion is a spatiotemporal orientation that can be detected with an oriented spatiotemporal filter. A type of bandpass filter tuned for spatial frequency and orientation. In that context the Gabor function was described in terms of two spatial dimensions. However, the same theory applies to spatiotemporal Gabor functions. Figure 2.7(b) illustrates how an oriented spatiotemporal filter can be tuned to detect a particular motion pattern.

A frequency domain representation of spatiotemporal orientation is shown in Fig. 2.8. In this particular figure, a 1D cosine wave is translating in time. Two plots are shown: Fig. 2.8(a) shows the actual transform of Fig. 2.7(b), and Fig. 2.8(b) shows the more attractive passbands of a Gabor filter tuned to a slightly different case of the same speed, but higher spatial and temporal frequencies. The real passbands have a Gaussian envelope, but the plot only shows a simplified representation that would result from thresholding them. The circular shape is therefore accurate, but the diameter is
arbitrary. This representation is less intuitive than Fig. 2.7, but it provides the relationship between spatiotemporal orientation and the spatial and temporal bandpass filters that will be used in the implementation. The white circles show a Gabor passband for a stationary, four pixel \( \psi_z = 1.4 \) grating. There is nothing special about four pixels, other than it makes an attractive plot in the frequency domain. Because the passband is located on the \( f_s \) axis, \( f_t = 0 \) and the filter detects gratings that are not changing with time. As the grating begins to translate, it appears to oscillate in time under the aperture of a particular receptive field at \( F_M \). This oscillation has some frequency related to the speed of translation and spatial frequency of the grating. The dashed arrows show the movement of the spatiotemporal energy as the \( \psi_z = 1.4 \) grating accelerates from rest to a speed of 70 pixel/s. At that speed, the four pixel grating passes the receptive field 70 \( \psi_z = 17.5 \) times in a second, so the temporal energy is centered at \( \pm f_t = 17.5 \). The spatiotemporal Gabor passband for this situation is shown by the black circles. The line connecting the black circles shows the constant speed of 70 pixel/s. Any spatiotemporal energy on that line corresponds to the same speed of some specific spatial frequency. A general velocity detector sensitive to a wide bandwidth of spatial frequency stimuli has been built by tiling that constant velocity line with an array of spatiotemporal filters and integrating their outputs.

Real imagery contains an independent spatial dimension \( y \). With two spatial dimensions, speed becomes a velocity feature with both speed tuning \( v_s \) (pixel/s) and direction \( \theta_v \) (rad). In this case, the frequency domain representation is shown in Fig. 2.9. The black dots represent the energy of a static grating accelerating to the grey dots, which represent a translating grating. The planes represent constant image velocity analogous to the lines of Fig. 2.8. At any velocity, the projection of stimulus energy onto the \( (f_r, f_y) \) plane is just the spatial frequency representation of the stimulus.
Figure 2.8: The frequency domain representation of a 1D translating grating is indicated by black points, and of a stationary grating of same spatial frequency by white points. Lines passing through origin show constant speed.
Figure 2.9: Spatiotemporal frequency representation of static and moving 2D image gratings. The two planes of constant velocity intersect on the $f_x$ axis. The black points represent the energy of a static grating. The grey points represent the same grating, but in motion. The arrows show the difference in velocities of the static and moving gratings. Figure after Watson and Ahumada. [60]
2.2.2 Aperture Problem

With two spatial dimensions, a phenomenon called the *Aperture Problem* occurs. The aperture problem is an ambiguity of the velocity that results from features larger than a neuron's receptive field as shown in Fig. 2.10(a). The circular outline represents a receptive field, or aperture, through which a moving bar is seen. The arrows suggest three different possible directions of motion of a specific point on the edge of the bar. Motion along any of those directions results in the same spatiotemporal pattern in the aperture. It is not mathematically possible to resolve this ambiguity within a single receptive field, unless there are frequency components at different orientations, as shown in Fig. 2.10(b). In the frequency domain, the aperture problem is shown in Fig. 2.11. A thresholded Gabor passband is shown by the black circles, along with two planes that both intersect the passband. The line that connects the two passbands passes through the origin, and is the intersection of an infinite number of planes, including the two shown. Because each plane corresponds to a constant velocity, there is a velocity ambiguity if all the motion energy lies on the line. Other spatial frequency components with the same orientation as the one shown will all lie on the same line, providing no relief from the aperture problem. Good solutions to the aperture problem are still open research issues. It is likely that all solutions to the aperture problem will require additional information, like motion patterns from adjacent receptive fields, or special application specific knowledge. This dissertation uses specific knowledge with respect to downstream computations to help resolve the aperture problem. The specific techniques will be shown in Chapter 4.
Figure 2.10: The Aperture Problem results when identical spatiotemporal patterns are produced by different motion patterns. The three arrows on the left suggest possible trajectories of a particular point that all result in the same motion patterns, and thus ambiguous velocity estimates.

Figure 2.11: Aperture Problem in the frequency domain. The black dots represent a thresholded Gabor passband. The planes show constant velocity. The phenomenon of many different planes intersecting the passbands results in the aperture problem.
2.2.3 Motion Energy

Oriented spatiotemporal filters like the 2D Gabor function in Fig. 2.7(b) cannot be separated into independent spatial and temporal filters. This raises an issue for implementations, both biophysical and engineered. A temporal filter implementation is likely to be significantly different than a spatial filter, making it difficult to combine them into a nonseparable spatiotemporal filter. Basic differences include causality and continuous time operation in a temporal filter, while the spatial filter has no causality considerations and is sampled at discrete photoreceptor positions. Realistic implementations require spatial and temporal filtering to be performed separately in series. Unfortunately, independent filters result in separable filter responses. Such a filter is shown in Fig. 2.12, with the same \( v_x = 70 \) pixel/s constant velocity line from Fig. 2.8 superimposed. The spatiotemporal filter indeed detects motion energy along \( v_x \), but unfortunately it also sees motion energy corresponding to \(-v_x\). In fact, the separable filter cannot distinguish between \( \pm v_x \).

Separable filters have other problems illustrated in Fig. 2.13. There are four columns of time domain traces. Above each column is shown a spatial impulse, moving in the direction indicated by the dotted arrow. The first two columns represent the responses to a dark impulse on a light background, and the second two columns indicate the response to a light impulse on a dark background. Processing these spatial impulses with spatiotemporally oriented Gabor filters results in the 1D temporal Gabor function responses \( \xi(t; \Omega_M) \), shown in the top row traces of the figure. The Gaussian envelopes are shown in light gray. One of three problems shown here is that the instantaneous value of the linear filters depends on the phase of the filter or, equivalently, the location of the stimulus within the receptive field. Secondly, the response of the linear filters are polarity dependent. Finally, the same separable filter responds to motion in both

---

---
Figure 2.12: Representation of separable filters to illustrate that they cannot have orientation. The solid line in the right subfigure corresponds to 140 pixels per second, and the dashed line is -140 pixels per second.
Figure 2.13: The four idealized stages of Adelson and Bergen's motion energy detector. Figure after Adelson and Bergen. [43]

directions as discussed above using frequency domain arguments. In fact, the linear filter cannot distinguish between a light impulse moving left and a dark impulse moving right.

The second row traces $\xi_L(t; \Omega_M)$ show the responses of an oriented linear filter to the same stimulus. This set of responses cannot be generated with a separable filter. The directional ambiguity is resolved, but the phase and polarity dependencies are not.

The third row $\xi_E(t; \Omega_M)$ is the result of a nonlinear combination of two oriented linear filters. This will be detailed below, but basically it is the result of two linear Gabor functions, out of phase by $\pi/4$ radians, summed and squared:

$$
\xi_E(t; \Omega_M) = \left\{ S(x,t) \otimes g(x,t; \Omega_M) \sin(2\pi \nu_x x - 2\pi \nu_t t) \right\}^2 - \\
\left\{ S(x,t) \otimes g(x,t; \Omega_M) \cos(2\pi \nu_x x - 2\pi \nu_t t) \right\}^2
$$
\[ S(x,t) = y(x,t; \Omega_M). \] (2.13)

The term "motion energy", from Adelson and Bergen [43], is used to indicate that the measure is independent of frequency, phase, and contrast polarity. Opponent energy is just the oriented energy response of one velocity \( \xi^\tau(t; \Omega_M) \) minus the oriented energy response of a filter tuned for the opposite direction \( \xi^\tau(t; \Omega_M) \):

\[ \xi(t; \Omega_M) = \xi^\tau(t; \Omega_M) - \xi^\tau(t; \Omega_M). \] (2.14)

It will be shown below that the oriented energy responses \( \xi(t; \Omega_M) \) are corrupted by undesired mathematical terms, so Fig. 2.13 is never realized as cleanly as shown. However, the terms cancel out in the final opponent motion energy \( \xi(t; \Omega_M) \) to provide the same results as shown in Fig. 2.13.

### 2.2.4 Implementation

It is argued above that a motion energy detector has to use separable spatial and temporal filters for real implementations, but that separable filters have some problems of their own that have to be resolved. The nonlinear combination of quadrature bandpass filter pairs is the approach taken here.

#### General Bandpass Filters

Gabor filters are but a single class of oriented functions that can be used to detect spatiotemporal orientation. There is no physiological evidence that such a restricted class of filters is necessary for practical visual processing. Indeed, physically realizable, efficient hardware implementations of these models may have a different response than the ideal Gabor functions discussed so far. A further complication is that the temporal
part of the filter needs to be causal, which a Gabor filter is not. Adelson and Bergen
originally described a class of motion energy detectors with proposals for specific spa­
tial and temporal filters that could be used. 43 Simoncelli and Heeger alternately used
directional derivatives of a Gaussian. 44 The basic idea of motion energy is valid for
many filter types, and this creates a potential divergence between the way biological
systems implement motion energy detectors and the way that engineered robots might.
The following derivation allows the issue to be resolved using technological arguments,
without introducing new algorithmic constraints. In the artificial implementation de­
scribed in this dissertation, it is a trivial matter to change the quantitative details of the
spatial and temporal filters. In other implementations using special purpose hardware,
there may be important technological reasons to choose a specific filter class different
than, but compatible with, the ones used here.

In order to deal with a more general class of spatiotemporal filters, consider the class
of bandpass filters described in the frequency domain by Fig. 2.14. Some arbitrary low
pass filter response $A(f)$ is transformed into a pair of bandpass filters, both with center
frequency $\omega$. The corresponding spatial or time domain representation is

$$h'(x) = a(x) \cos(2\pi \omega x),$$

$$h''(x) = a(x) \sin(2\pi \omega x)$$  \hspace{1cm} (2.15)

where $a(x)$ is the inverse Fourier transform of $A(f)$:

$$a(x) = \int_{-\infty}^{\infty} A(f) e^{2\pi i f x} df.$$  \hspace{1cm} (2.16)
Figure 2.14: The low pass filter $A(f)$ is transformed into a pair of quadrature bandpass filters $H'(f)$ and $H''(f)$ with center frequency $\nu$.

The frequency response of the bandpass filters is

$$H'(f) = \frac{1}{2} \left[ A(f - \nu)e^{j\kappa} - A(f - \nu)e^{-j\kappa} \right]$$

where $\kappa = 0$ for $H'(f)$ and $\kappa = \pi$ for $H''(f)$. The 1D Gabor filter is recovered when

$$A(f) = e^{-\frac{\omega^2 f^2}{2}}.$$  \hspace{1cm} (2.18)

However, the following derivation of motion energy does not depend on any particular form for $A(f)$.

The rest of this section derives the motion energy features $\xi_M(t; \Omega_M)$ in quantitative detail. It is a somewhat tedious derivation, so some readers may wish to skip forward.
to Section 2.3 armed with the qualitative understanding of Eq. (2.14) and Fig. 2.13. However, the following derivation of motion energy is a novel contribution in that it uses the general form of the spatial and temporal filters seen in Fig. 2.14.

**First Stage Separable Filters**

The first stage of the Adelson Bergen motion energy model consists of separable spatial and temporal responses

\[ \xi_s(t: \Omega_M) = S(x, y, t) \ast h_s(x, y; \theta_M, \psi_s) \ast h_t(t; \psi_t) \]  

(2.19)

where \( S(x, y, t) \) is an input stimulus, \( h_s(x, y; \theta_M, \psi_s) \) is a spatial bandpass filter, \( h_t(t; \psi_t) \) is a temporal bandpass filter. \( \ast \) represents a 2D correlation in space

\[ S(x, y, t) \ast h_s(x, y; \theta_M, \psi_s) = \int \int S(x - x', y - y')h_s(x, y; \theta_M, \psi_s)dx dy. \]  

(2.20)

\( \ast \) represents convolution in time

\[ S(x, y, t) \ast h_t(t; \psi_t) = \int \int S(x, y, t')h_t(t; \psi_t)dx dy. \]  

(2.21)

and \( \Omega_M \) is the full motion energy tuning \( \{ \Omega_M, \theta_M, \psi_s, \psi_t \} \). Both \( h_s(x, y; \theta_M, \psi_s) \) and \( h_t(t; \psi_t) \) are of the form of Eq. (2.15) with frequency tunings \( \psi_s \) and \( \psi_t \). The spatial filter is a 2D version with associated orientation parameter \( \theta_M \). In the frequency domain,

\[ \Xi_s(f_x, f_y, f_t; \Omega_M) = \mathcal{F}\{S(x, y, t)\}H_s(-f_x, -f_y; \Omega_M, \psi_s)H_t(f_t; \psi_t). \]  

(2.22)
Since the spatial filters under consideration are separable, the remainder of this derivation will, for simplicity, be conducted using only a single spatial dimension labeled as $x$. Consider the response of this filter at a single frequency by using the spatiotemporal grating stimulus

$$S(x,t) = \cos(2\pi \eta_x x - 2\pi \eta_t t).$$

(2.23)

where $\eta_x$ and $\eta_t$ are spatial and temporal frequencies, respectively. In this case, Eq. (2.22) becomes

$$\Xi_{\eta_x}(f_x, f_t) = \frac{1}{2} \delta(f_x - \eta_x) \delta(f_t - \eta_t) - \delta(f_x - \eta_x) \delta(f_t - \eta_t) H_x(f_x) H_t(f_t).$$

(2.24)

The frequency domain is a convenient way to design $A(f)$, but the implementation must operate in the spatiotemporal domain. The motion energy result $\xi_M(t; \Omega_M)$ must be independent of oscillations from $\eta_x$ and $\eta_t$ for downstream integration of motion energy features, also best shown in the spatiotemporal domain. Returning to the spatiotemporal domain by taking the inverse Fourier transform.

$$\xi_M(t; \Omega_M) = \mathcal{F}^{-1}\{\Xi_{\eta_x}(f_x, f_t; \Omega_M)\} \bigg|_{x_M}$$

$$= \frac{1}{2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(f_x - \eta_x) \delta(f_t - \eta_t) H_x(f_x) H_t(f_t)$$

$$\times \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(f_x - \eta_x) \delta(f_t - \eta_t) H_x(f_x) H_t(f_t)$$

$$= \frac{1}{2} H_x(-\eta_x) H_t(-\eta_t) e^{-j2\pi \eta_x x_M} e^{-j2\pi \eta_t t_M} - \frac{1}{2} H_x(\eta_x) H_t(\eta_t) e^{j2\pi \eta_x x_M} e^{j2\pi \eta_t t_M}$$

where the evaluation at $x_M$ accounts for the receptive field location implicit in $\Omega_M$.

At this point, $H_x(f_x)$ represents an abstract filter with some phase parameter $\kappa_x$, and $H_t(f_t)$ has a similar interpretation with the phase parameter $\kappa_t$. The motion energy
model of Adelson Bergen uses four features corresponding to all combinations of in
phase and quadrature versions of $H_x(f_x; \Omega_M)$ and $H_t(f_t; \Omega_t)$: $\xi_S(t; \Omega_M)$, $\xi_S(t; \Omega_M)$, $\xi_S(t; \Omega_M)$. To show this, $\kappa_x$ and $\kappa_t$ are included in the notation of $\xi_S$, with expanded $H_x(f_x)$ and $H_t(f_t)$:

$$
\xi_S(t; \Omega_M; \kappa_x, \kappa_t) = \frac{1}{8} \cdot A_x(-\eta_x - \nu_x) e^{-j\kappa_t^2} - A_x(-\eta_x - \nu_x) e^{-j\kappa_t^2} e^{-j2\pi\eta_x t_M} \\
\cdot A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} - A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M} \\
\frac{1}{8} \cdot A_x(\eta_x - \nu_x) e^{j\kappa_t^2} - A_x(\eta_x - \nu_x) e^{j\kappa_t^2} e^{j2\pi\eta_x t_M} \\
\cdot A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} - A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M} \\
\frac{1}{8} \cdot A_x(\eta_x - \nu_x) e^{j\kappa_t^2} e^{-j2\pi\eta_x t_M} - A_x(\eta_x - \nu_x) e^{j\kappa_t^2} e^{-j2\pi\eta_x t_M} \\
\cdot A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M} - A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M}.
$$

(2.25)

The filters $A_x(f_x)$ and $A_t(f_t)$ are even because they are Fourier transforms of real
impulse responses. Therefore,

$$
\xi_S(t; \Omega_M, \kappa_x, \kappa_t) = \frac{1}{8} \cdot A_x(\eta_x - \nu_x) e^{-j\kappa_t^2} - A_x(\eta_x - \nu_x) e^{-j\kappa_t^2} e^{-j2\pi\eta_x t_M} \\
\cdot A_t(\eta_t - \nu_t) e^{j\kappa_t^2} - A_t(\eta_t - \nu_t) e^{j\kappa_t^2} e^{-j2\pi\eta_t t_M} \\
\frac{1}{8} \cdot A_x(\eta_x - \nu_x) e^{j\kappa_t^2} - A_x(\eta_x - \nu_x) e^{j\kappa_t^2} e^{j2\pi\eta_x t_M} \\
\cdot A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} - A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M} \\
\frac{1}{8} \cdot A_x(\eta_x - \nu_x) e^{j\kappa_t^2} e^{-j2\pi\eta_x t_M} - A_x(\eta_x - \nu_x) e^{j\kappa_t^2} e^{-j2\pi\eta_x t_M} \\
\cdot A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M} - A_t(\eta_t - \nu_t) e^{-j\kappa_t^2} e^{-j2\pi\eta_t t_M}.
$$

(2.26)

In order to build realizable filters, $\kappa_x$ and $\kappa_t$ must be specified. Using the only
possible values of $\kappa_x$ and $\kappa_t$: 0 or $-\pi$, there are four possible combinations:

$$
\xi_{S1}(t; \Omega_M) = \xi_S(t; \Omega_M; \kappa_x = 0, \kappa_t = 0) \\
= \frac{1}{8} \cdot A_x(\eta_x - \nu_x) - A_x(\eta_x - \nu_x) e^{-j2\pi\eta_x t_M} - A_t(\eta_t - \nu_t) - A_t(\eta_t - \nu_t) e^{-j2\pi\eta_t t_M} \\
\frac{1}{4} \cdot A_x(\eta_x - \nu_x) - A_x(\eta_x - \nu_x) e^{-j2\pi\eta_x t_M} - A_t(\eta_t - \nu_t) - A_t(\eta_t - \nu_t) e^{-j2\pi\eta_t t_M}.
$$
\[
\begin{align*}
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi ') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi ') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma - \eta t) \varphi - (\gamma - \eta t) \varphi') & ((\gamma - \eta t)^2 \varphi - (\gamma - \eta t)^2 \varphi') \frac{1}{\Gamma} = \\
((\gamma + \eta t) \varphi' - (\gamma + \eta t) \varphi') & ((\gamma + \eta t)^2 \varphi' - (\gamma + \eta t)^2 \varphi') \frac{1}{\Gamma} = \end{align*}
\]
One finds that \( \xi_{S1}(t; \Omega_M), \xi_{S2}(t; \Omega_M), \xi_{S3}(t; \Omega_M), \) and \( \xi_{S4}(t; \Omega_M) \) represent all the filtering that is actually required for computing motion energy. The rest of the steps consist only of linear and nonlinear combinations of the separable responses. The first are linear combinations, i.e., additions and subtractions, to form the oriented linear responses

\[
\begin{align*}
\xi_{L1}(t; \Omega_M) &= \xi_{S1}(t; \Omega_M) - \xi_{S4}(t; \Omega_M) \\
&= \frac{1}{2} \left[ A_r(\eta_x - \nu_x)A_r(\eta_y - \nu_y) - A_r(\eta_x + \nu_x)A_r(\eta_y - \nu_y) \right] \\
&\quad \cos(2\pi \eta_x x_M - 2\pi \eta_y t). \\
\xi_{L2}(t; \Omega_M) &= \xi_{S2}(t; \Omega_M) - \xi_{S3}(t; \Omega_M) \\
&= \frac{1}{2} \left[ A_r(\eta_x - \nu_x)A_r(\eta_y + \nu_y) - A_r(\eta_x + \nu_x)A_r(\eta_y + \nu_y) \right] \\
&\quad \sin(2\pi \eta_x x_M - 2\pi \eta_y t). \\
\xi_{L3}(t; \Omega_M) &= \xi_{S1}(t; \Omega_M) + \xi_{S4}(t; \Omega_M) \\
&= \frac{1}{2} \left[ A_r(\eta_x - \nu_x)A_r(\eta_y - \nu_y) + A_r(\eta_x + \nu_x)A_r(\eta_y + \nu_y) \right] \\
&\quad \cos(2\pi \eta_x x_M - 2\pi \eta_y t). \\
\xi_{L4}(t; \Omega_M) &= \xi_{S2}(t; \Omega_M) + \xi_{S3}(t; \Omega_M) \\
&= -\frac{1}{2} \left[ A_r(\eta_x - \nu_x)A_r(\eta_y + \nu_y) - A_r(\eta_x + \nu_x)A_r(\eta_y - \nu_y) \right] \\
&\quad \sin(2\pi \eta_x x_M - 2\pi \eta_y t). 
\end{align*}
\]

Oriented motion energy is computed by squaring the oriented linear responses and using the identity \( \cos^2(x) - \sin^2(x) = 1 \):

\[
\xi_{O1}(t; \Omega_M) = \xi_{L1}^2(t; \Omega_M) - \xi_{L2}^2(t; \Omega_M)
\]
Finally, the opponent motion energy feature is computed as

\[ \xi_{O2}(t; \Omega_M) = \xi_{L2}(t; \Omega_M) - \xi_{L4}(t; \Omega_M) \]

\[ = \frac{1}{4} \{ A_x^2(\eta_x - \zeta_x) A_x^2(\eta_x + \zeta_x) - A_x^2(\eta_x + \zeta_x) A_x^2(\eta_x - \zeta_x) \} \]

\[ - \frac{1}{4} \{ A_x(\eta_x - \zeta_x) A_x(\eta_x + \zeta_x) A_x(\eta_x - \zeta_x) A_x(\eta_x + \zeta_x) \} - \cos^2(2\pi\eta_x t_M - 2\pi\eta_t) - \sin^2(2\pi\eta_x t_M - 2\pi\eta_t) \}. \]

The first term in Eq. (2.27) is sensitive to leftward motion and the second term is sensitive to rightward motion. Because the signs of the two terms are different, the polarity of \( \xi_M(t; \Omega_M) \) distinguishes right from left. Also, Eq. (2.27) is independent of \( x, t, \kappa_x, \) and \( \kappa_t \). This independence removes all oscillatory behavior, resulting in a simple demodulated magnitude estimate of motion energy based only on the closeness of the stimulus frequency to the filter passband. Downstream processing is simplified as a result, easing computations that involve multiple motion energy outputs, each tuned to different frequencies. Otherwise, downstream results would contain undesired oscillations. Figure 2.15 shows the spatiotemporal frequency response of \( \xi_M(t; \Omega_M) \).
Figure 2.15: Frequency domain representation of motion energy feature detector $\xi_M(t; \Omega_M)$. Black passbands represent a positive response, indicating rightward motion. White passbands represent a negative response and leftward motion.

2.3 Motion Energy Integration

The next set of features relevant for robotic navigation based on visual motion is the computation of image velocity. Traditionally, computer scientists have computed a similar measure, called optic flow, using differential techniques and the assumption that image luminance is conserved. [28] Neuroscientists, on the other hand, tend to believe that optic flow in biological systems is computed using neurobiologically plausible mechanisms in area MT of the mammalian brain. As discussed previously in this dissertation, the primary visual cortex (V1) contains such neural mechanisms that are based on spatiotemporal frequency selectivity. The integration of V1 complex cell afferent $\xi_M(t; \Omega_M)$ into optic flow has been studied by many authors reviewed in this section. [43, 45, 46, 47, 48, 49, 50] All previous work on motion energy integration
reviewed here was done with the purpose of explaining physiological or physio-
physiological experimental results from neuroscience laboratories. Our concern here is robotic
vision, which turns out to stress these algorithms that have never before been com-
puted (other than, perhaps, by real animals) in large scale implementations, with real
imagery, fast enough for closed loop control of a mobile robot. The cited research is
usually performed with very basic, nonrealistic stimuli such as sinusoidal gratings and
combinations of those, random dot patterns, and edge patterns. Not coincidentally,
these are the same stimuli that experimental neuroscientists use when studying real
biological vision systems. Applied without modification to real world imagery, these al-
gorithms result in comic robotic performance. In this dissertation, these weaknesses are
exposed and a related algorithm, modified from these based on experimental evidence,
is used to increase robustness for this application.

Motion energy features $\xi_M(t; \Omega_M)$ are sensitive to spatiotemporal frequency, not ve-
locity. In general, motion energy is computable with biologically plausible mechanisms
but confounds stimulus contrast, spatial frequency, and temporal frequency. Integration
of the motion energy features $\xi_M(t; \Omega_M)$ resolves these confounds into reliable measures
of velocity taking into account the aperture problem and image contrast.

2.3.1 General Form of Motion Energy Integration

Most of the literature on this topic conforms to a basic overall strategy inspired by
primate visual systems diagrammed in Fig. 2.16. In the retina itself, “bipolar” cells per-
form a spatial prefiltering operation $P(x, y)$ on raw luminance information $I(x, y, t)$ that
equalizes regions of different illumination but low contrast: basically a high pass filter.
In this dissertation, the output of this prefilter is specified as the stimulus $S(x, y, t)$. In
the primary visual cortex (V1), there is an approximately linear spatiotemporal filtering operation \( H(x, y, t; \Omega_M) \) performed by V1 simple cells computing separable linear features \( \xi_s(t; \Omega_M) \), nonlinearly combined into phase and contrast polarity independent motion energy features \( \xi_M(t; \Omega_M) \) by V1 complex cells. This operation may or may not include a normalization term. Some models include a rectification of their linear features \( \xi_s(t; \Omega_M) \) to account for the fact that real firing rates cannot be negative, as could happen when the impulse response of \( H(x, y, t; \Omega_M) \) has negative, or inhibitory, regions. In any case, the linear filter \( H(x, y, t; \Omega_M) \) detects spatiotemporal orientation as all these methods require. The specific orientation detected is specified by the tuning parameters \( \Omega_M \).

In area MT, motion energy afferents \( \xi_M(t; \Omega_M) \) are integrated into spatiotemporal frequency independent velocity features \( \xi_V(t; \Omega_V) \) tuned to retinotopical location \( \vec{r}_V \), orientation \( \theta_V \), and speed \( v_s \), gathered into a tuning vector \( \Omega_V = (\vec{r}_V, \theta_V, v_s) \). This integration also may or may not include nonlinearities and normalization. In Fig. 2.16 various spatiotemporal functions are indicated as functions of \( x, y \) and \( t \) and population codes are represented as \( \xi_*(t; \Omega_*) \) where \( * \) is replaced with specific feature and tuning vector types. All population codes are functions of time and tuned to some retinotopic location \( \vec{r}_M \) or \( \vec{r}_V \). Separable features \( \xi_s(t; \Omega_M) \) and motion energy features \( \xi_M(t; \Omega_M) \) are tuned to retinotopic location and 3D spatiotemporal passbands \( \Omega_M = (\vec{r}_M, v_s, \theta_M, \psi_s, \sigma, \sigma) \). Usually, there are multiple \( \xi_s(t; \Omega_M) \) features for a particular tuning. Velocity outputs \( \xi_V(t; \Omega_V) \) are tuned to retinotopic location, orientation, and speed \( \Omega_V = (\vec{r}_V, \theta_V, v_s) \) where \( (\theta_V, v_s) \) is velocity orientation and speed, respectively.

The rest of this section describes specific operations commonly found either in MT models themselves or their underlying implementations. Because the application of
Figure 2.16: General form of motion energy integration assumed by the literature cited in this review.
this dissertation is in real time robotic control, every operation in the information processing flow must have concrete implementations that are not always addressed when considering abstract models of visual processing. In many cases, existing models have to be modified for use in this robotics application for real world implementation issues such as causal temporal filters and fast computation.

**Center-Surround Spatial Filter**

Some of the methods reviewed in this section simulate the center-surround highpass filter \( P(x, y) \) thought to be done by bipolar cells in the primate retina. Computationally, some authors find that their spatial filters, unless odd functions, result in an undesirable DC offset that can be overcome with this method. The typical model of a center-surround prefilter is a Difference-of-Gaussians

\[
P(x, y; \sigma_1, \sigma_2) = \frac{1}{2\pi\sigma_2^4} \frac{e^{-x^2/2\sigma_2^2}}{\sigma_2^2} - \frac{1}{2\pi\sigma_1^4} \frac{e^{-x^2/2\sigma_1^2}}{\sigma_1^2}
\]

where \( \sigma_2 > \sigma_1 \). This function, shown in 1D in Figure 2.17, has a positive peak at the center of its receptive field, and a broad inhibitory region around it. Diffuse image features convolve with this filter to zero, resulting in a high pass response. An example of a real world image convolved with \( P(x, y; \sigma_1 = 1, \sigma_2 = 3.5) \) is illustrated in Fig. 2.18. The only other prefilter found in this review was used by Simoncelli and Heeger [44], which will be described in context below.

**2.3.2 Cascaded, Truncated Exponential Filter**

Realistic temporal filters must be causal. Gabor filters, while theoretically convenient, are not causal. Many published velocity models used temporal Gabor filters
Figure 2.17: 1D center-surround impulse response.

(a) $H(x; \sigma_1 = 1, \sigma_2 = 3.5)$

(b) Components

Figure 2.18: The input image $I(x, y, t_0)$ has been filtered with a center-surround 2D Difference-of-Gaussians filter with $\sigma_1 = 1$ and $\sigma_2 = 3.5$.

(a) Raw image slice $I(x, y, t_0)$

(b) Processed image $S(x, y, t_0)$
for their convenience. However, they must be replaced in real implementations. It is possible to approximate an FIR Gabor filter by delaying input for a few frames and truncating the impulse response to achieve causality. Unfortunately, the delay results in undesirable latency for a real-time system. An attractive alternative is based on a cascade of truncated exponential filters. These filters are used in this dissertation because they are causal and have a recursive (IIR) implementation. Recursive filters are substantially cheaper to compute than similar FIR filters, and from a design point of view, are similar to continuous time analog filters. Thus, an IIR design will most likely be required for any real-time motion energy implementation, especially using special purpose analog hardware.

The basic truncated exponential impulse response is

$$a_1(t; \alpha t) = \begin{cases} \alpha t^{\alpha t} & : t \geq 0 \\ 0 & : t < 0 \end{cases}$$

(2.29)

with frequency response

$$A_1(f_t) = \frac{\alpha t}{\alpha t - j2\pi f_t}. \quad (2.30)$$

The truncation is represented by the fact that $a_1(t; \alpha t) = 0$ for $t < 0$. Better frequency characteristics (i.e., a steeper response) can be achieved by cascading several of them together, which results in an impulse response

$$a_n(t; \alpha t) \equiv a_{1^n} = \begin{cases} \frac{t^{\frac{n-1}{n}} \alpha t^{\frac{n}{n-1}} - \alpha t}{n-1} & : t \geq 0 \\ 0 & : t < 0 \end{cases}$$

(2.31)
where $a_i^n(t)$ means $a_i(t)$ convolved with itself $n$ times. The cascaded exponential has frequency response

$$A_n(f) = \frac{\alpha_i^n}{\sqrt{(\alpha_i^2 + (2\pi f)^2)^n}}. \quad (2.32)$$

As a consequence of the Central Limit Theorem, $a_n(t)$ approaches a Gaussian as $n$ becomes large. This is a satisfying convergence to a Gabor filter, upon which the basic motion energy theory was based in Section 2.1, in addition to other velocity models reviewed in this section.

Equation 2.32 is a low pass filter, but bandpass filters are required for computing motion energy $\xi_M(t; \Omega_M)$. Consistent with the previous notation introduced in Eq. (2.15), $a_n(t; \alpha_i)$ can be used as a frequency envelope for a bandpass filter

$$f_n(t) = \begin{cases} \frac{\alpha_i^n}{i^n} e^{-\alpha_i^2 j2\pi \alpha_i t} & : \ t \geq 0 \\ 0 & : \ t < 0 \end{cases} \quad (2.33)$$
2.3.3 Derivatives of a Gaussian Filter

Many authors use spatial filters based on derivatives of a Gaussian as an alternative to other bandpass filters like the Gabor:

\[ g_0(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \]
\[ g_1(x) = \frac{\partial g_0(x)}{\partial x} = -\frac{x}{\sigma^2} g_0(x) \]
\[ g_2(x) = -\frac{\partial^2 g_0(x)}{\partial x^2} = -\left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) g_0(x) \]
\[ g_3(x) = -\frac{\partial^3 g_0(x)}{\partial x^3} = -\left(\frac{2x}{\sigma^4} - \frac{x^3}{\sigma^6} - \frac{x}{\sigma^2}\right) g_0(x). \] (2.34)

These functions are plotted in Fig. 2.20. Computationally, derivatives of a Gaussian are convenient filters because they are **steerable**, meaning that arbitrary orientation tunings can be interpolated from a minimal set of basis filters\(^1\).

In this case, tuned spatial filters of any orientation can be simulated from just two actual 2D convolutions: one with vertical orientation, the other with horizontal orientation.

2.3.4 Literature Review

Adelson and Bergen: 1986

Adelson and Bergen described concrete filters for their energy model and a simple way to integrate these features into velocity.\(^43,73\) Adelson and Bergen did not use a center-surround prefilter \(P(x, y)\). Their spatial filters were second and third derivatives of a Gaussian (Fig. 2.20) and their temporal filter's impulse response was

\[ f(t; k, n) = (kt)^n e^{-kt} \left[ \frac{1}{n!} - \frac{(kt)^2}{(n - 2)!} \right] \] (2.35)

\(^1\)Gabor functions are steerable only if approximated by polynomials.
with the even and odd versions corresponding to $n = (3, 5)$, respectively. Figure 2.21 plots these functions. Adelson and Bergen claimed these filters approximate psychophysically observed curves. That may be true, but these filters are computationally inconvenient and do not support the theoretical and practical justification of Gabor and truncated exponential filters. Here, we simply observe the qualitative similarity to those other functions (Fig. 2.19) and do not further consider these filters.

As detailed in Section 2.1, Adelson and Bergen’s model computed motion energy features corresponding to V1 complex cell outputs

$$
    \xi_M(t; \Omega_M) = \left\{ [\xi_{S1}(t; \Omega_M) - \xi_{S4}(t; \Omega_M)]^2 + [\xi_{S2}(t; \Omega_M) - \xi_{S3}(t; \Omega_M)]^2 \right\} - \\
    \left\{ [\xi_{S1}(t; \Omega_M) - \xi_{S4}(t; \Omega_M)]^2 + [\xi_{S2}(t; \Omega_M) - \xi_{S3}(t; \Omega_M)]^2 \right\} 
$$  (2.36)
Adelson and Bergen's temporal filters. Solid line plots $f_1(t) = f(t; k = 1, n = 3)$, dashed line plots $f_2(t) = f(t; k = 1, n = 5)$.

where the linear simple cell outputs $\xi_{S_n}(t; \Omega_M), 1 \leq n \leq 4$, correspond to the four combinations of even and odd spatial and temporal filters convolved with the input stimulus $S(x, y, t)$.

Adelson and Bergen's model was a 2D spatiotemporal model: as such, it only included one spatial dimension and the concept of velocity reduced to speed. In this case, there was no aperture problem as there was no spatial orientation. The only issue the authors considered is contrast normalization, which they performed by normalizing motion energy outputs by the output of a static motion energy feature $\xi_{M0}(t; x_M, v_x) = \xi_M(t; x_M, v_x, v_z = 0)$:

$$\xi_V(t; x_M, v_x, v_z = v_z, v_v) = \frac{\xi_M(t; x_M, v_x, v_z)}{\xi_{M0}(t; x_M, v_x)}$$  (2.37)
This algorithm is too simple to be applied to robotic navigation. It only handles 1D spatial imagery, and breaks when \( \xi_{M0}(t; x_M, \psi) \) is small. This sort of fragility is catastrophic using real imagery. It also does not integrate multiple spatiotemporal tunings as \( \xi_{M1}(t; x_M, \psi, \psi) \) is still a function of \( \psi \), which is a necessary step for robust performance with real imagery.

**Heeger: 1987**

Heeger [18] used a center-surround filter \( P(x, y) \) to equalize bright, low contrast images. He was not quantitatively specific about the filter, but his qualitative description sounds like a Difference-of-Gaussians similar to Eq. (2.28). Next, his algorithm computed even and odd linear features \( \xi_s(t; \Omega_M), \xi_o(t; \Omega_M) \) from simple quadrature 3D spatiotemporal Gabor filters:

\[
H_s(x, y, t; \Omega_M) = \frac{1}{(2\pi)^{3/2} \sigma_x^2 \sigma_y^2 \sigma_t^2} \exp\left(-\frac{1}{2} \frac{x^2}{\sigma_x^2} - \frac{y^2}{\sigma_y^2} - \frac{t^2}{\sigma_t^2}\right) \times \\
\cos[2\pi \psi, (x \cos \theta_M - y \sin \theta_M) + 2\pi \psi, t]
\]

\[
H_o(x, y, t; \Omega_M) = \frac{1}{(2\pi)^{3/2} \sigma_x^2 \sigma_y^2 \sigma_t^2} \exp\left(-\frac{1}{2} \frac{x^2}{\sigma_x^2} - \frac{y^2}{\sigma_y^2} - \frac{t^2}{\sigma_t^2}\right) \times \\
\sin[2\pi \psi, (x \cos \theta_M - y \sin \theta_M) + 2\pi \psi, t].
\]

Motion energy features were nonlinear combinations of

\[
\xi_s(t; \Omega_M) = S(x, y, t) \odot H_s(x, y, t; \Omega_M);
\]

\[
\xi_o(t; \Omega_M) = \left\{ [S(x, y, t) \odot H_s(x, y, t; \Omega_M)]^2 - [S(x, y, t) \odot H_o(x, y, t; \Omega_M)]^2 \right\}^{1/2}
\]

\[
\odot \rho(x, y, t)
\]
where $p(x, y, t)$ was a 3D spatiotemporal Gaussian with four pixel spatial width and one frame temporal width. This pooling was required for theoretical reasons and had the additional effect of making the MT receptive fields about four times larger than the receptive fields of the V1 models. Velocity was computed as a least squares minimization of

$$E(t; \tilde{F}_V, \theta_V, r_z) = \sum_{\theta_M, \Omega_M} \left[ \frac{\xi'_M(t; \Omega_M)}{\xi_M(t; \Omega_M)} \right]^2$$

with respect to velocity $(\theta_V, r_z)$, where $\xi'_M(\theta_V, r_z; \Omega_M)$ is a constant predicted $\Omega_M$ tuned motion energy output for stimulus velocity $(\theta_V, r_z)$ and

$$\xi_M(t; \tilde{F}_M, \theta_M) = \sum_{c, c'} \xi_M(t; \Omega_M).$$

$$\xi'_M(\theta_V, r_z; \theta_M) = \sum_{c, c'} \xi'_M(\theta_V, r_z; \Omega_M).$$

The divisive terms $\xi_M(t; \tilde{F}_M, \theta_M)$ and $\xi'_M(\theta_V, r_z; \theta_M)$ normalized the motion energy features over $c_z$ and $c_t$, but not $\theta_M$. This had the effect of normalizing orientation channels separately. The predicted values $\xi'_M(\theta_V, r_z; \Omega_M)$ were computed explicitly based on theoretical arguments, but those arguments assumed a flat image power spectrum. This assumption was a weak point in Heeger’s algorithm which he partially addressed with the $\theta_M$ specific normalization terms. In addition, the explicit calculation of $\xi'_M(\theta_V, r_z; \Omega_M)$ was made possible by the simplicity of a temporal Gabor filter in the motion energy stage. In real implementations where one has to worry about causality of the temporal filters, Heeger’s formulas are no longer valid. In this case, some kind of iterative or learning algorithm is required.
Alternatively to maximizing Eq. (2.39), a population coded output was computed as

\[
\xi_V(t; \Omega_V) = \sum_{a_M, \nu, \lambda} \xi_M^2(t; \Omega_M) - \left[ \frac{\xi_M(t; \mathcal{P}_M, \theta_M)}{\xi_M(\theta_M, \nu, \lambda; \Omega_M)} - \xi_M(t; \Omega_M) \right]^2 \quad (2.39)
\]

Equation 2.39 implemented an intersection of constraints calculation, finding those candidate velocities most likely given all the motion energy information.

Heeger's network consisted of twelve motion energy features \( \xi_M(t; \Omega_M) \) at each spatial frequency \( \nu \): four orientations and three temporal frequency tunings. His implementation allowed only octave spacing in \( \nu \), but that is not a major concern because his results hold for alternative implementations as well. His twelve velocity tunings included eight directions, and four stationary tunings of different spatial orientation. There is no discussion on how this particular configuration was chosen.

**Grzywacz and Yuille: 1990**

Grzywacz and Yuille [45] did not use a prefilter \( P(x, y) \), but otherwise their method is similar to Heeger [18]. Their linear filters were basically the same as Eq. (2.38) except written in complex form

\[
H(x, y, t; \Omega_M) = \frac{1}{(2\pi)^{\frac{3}{2}} \sigma_x \sigma_y} e^{-\frac{x^2 + y^2}{2\sigma_x^2}} e^{-\frac{t^2}{2\sigma_t^2}} e^{-j2\pi \nu \left[ x \cos \theta_M + y \sin \theta_M + 2\pi \nu t \right]} \quad (2.40)
\]

with a similar nonlinear combination into motion energy features

\[
\xi_M(t; \Omega_M) = |H(x, y, t) \odot S(x, y, t)|^2 \quad (2.41)
\]
but without any spatial pooling or other filtering of the V1 complex cell outputs. They proposed two different methods of integrating their motion energy features into velocity features $\xi_V(t; \Omega_V)$. The biologically plausible one was basically a template match looking for a plane in spatiotemporal frequency space

$$\xi_V(t; \Omega_V) = \sum_{\eta_M, \xi_M} W(\Omega_M, \Omega_V) \xi_M(t; \Omega_M)$$ (2.42)

where the weights were computed as

$$W(\Omega_M, \Omega_V) = e^{-\frac{1}{2} \frac{1}{\sigma^2_{\eta M}} \eta_M^2}, e^{-\frac{1}{2} \frac{1}{\sigma^2_{\xi M}} \xi_M^2}, e^{-\frac{1}{2} \frac{1}{\sigma^2_{\eta V}} \eta_V^2}, e^{-\frac{1}{2} \frac{1}{\sigma^2_{\xi V}} \xi_V^2}.$$ (2.43)

The first exponential term measured the distance between the spatiotemporal frequency plane in $\Omega_V$ and motion energy tuning $\Omega_M$. The second exponential term biased with weight $k$ those candidate velocities to normal flow in order to resolve ambiguities related to the aperture problem.

Grzywacz and Yuille did not discuss specific methods of choosing $\Omega_M$ and $\Omega_V$, although they did propose increasing the spatiotemporal bandwidth of $\Omega_M$ with distance from the origin to reduce sensitivity to noise.

**Smith and Grzywacz: 1993**

Smith and Grzywacz [64] extended the model of Gryzwacz and Yuille [45] to account for transparent motion. Two changes were a normalization term and a dynamic integration region in the velocity feature computation

$$\xi_V(t; \Omega_V, \Theta_M^t) = \sum_{\eta_M, \xi_M \in \Theta_M^t} W(\Omega_M, \Omega_V) \xi_M(t; \Omega_M) \frac{\xi_M(t; \Omega_M)}{\sum_{\xi_M \in \Theta_M^t} \xi_M(t; \Omega_M, \Theta_M^t)}$$ (2.44)
where \( \Theta_g \) is a region of orientations where the gradient \( \partial \xi_M(t; \Omega_M) / \partial \theta_M \) is small, and the new summation is over some local spatial neighborhood \( N \) which normalizes the motion energy features with respect to stimulus contrast and intensity. The neighborhood \( \Theta_g \) performs segmentation in orientation that accounts for multiple velocities at receptive field location \( \vec{x}_V \). In the original model of Grzywacz and Yuille, a winner take all step chose a single velocity for location \( \vec{x}_V \). This segmentation makes it possible to have more than one velocity. In this dissertation, however, the whole population code \( \xi_V(t; \Omega_V) \) is retained for further processing. The dependence on \( \Theta_g(t) \) on the velocity features \( \xi_V(t; \Omega_V, \Theta_g(t)) \) is awkward in this context, especially since the number, placement, and size of regions can change with time. This is not compatible with the parallel, fixed network topology proposed in this dissertation.

**Sereno: 1993**

Sereno did not specify how she computed her motion energy features \( \xi_M(t; \Omega_M) \), nor if any preprocessing was necessary using her algorithm. It is possible that she used synthetically generated \( \xi_M(t; \Omega_M) \) to study her network without actually using imagery at all. However, it is clear that her motion energy features only indicated motion orthogonal to their spatial orientation, which is also a limitation of all the other motion energy feature computations described in this review due to the aperture problem. Sereno integrated normalized motion energy features into velocity features in a purely linear fashion

\[
\xi_V(t; \Omega_V) = \sum_{\mu_M, \xi, \omega} W(\Omega_M, \Omega_V) \frac{\xi_M(t; \Omega_M)}{\sum_{\mu_M, \xi, \omega} \xi_M^2(t; \Omega_M)} 
\]  

(2.45)
with weights $W_1, \Omega_M, \Omega_V$ determined by a supervised training rule from the neural network literature [49]. Sereno's parameter tunings $\Omega_M$ and $\Omega_V$ did not separate spatial and temporal frequency; those parameters were collapsed into a single parameter, speed (pixel s). The MT receptive fields were ten times as big as the V1 receptive fields in her model, so the velocity computation integrates motion energy over a spatial neighborhood $N$ in addition to other parameters.

Sereno derived her parameter tunings from physiological evidence. Direction tunings were uniformly distributed at $15^\circ$ intervals with overlapping bandwidths of $60^\circ$. Overlapping speed tunings were spaced at octave intervals with three octave bandwidths between 2 and 128 pixel s.

Sereno's model is incomplete in that it only shows how to integrate motion energy features into velocity features, but does not show how to compute the motion energy features to begin with. Therefore, one has to wonder if her results are applicable when some particular method of computing motion energy is used. Also, the fact that her model operated on motion energy units already tuned to speed instead of spatiotemporal frequency side-stepped the critical issue of how that speed is computed. The supervised neural network algorithm also creates a problem, in that a training set must be generated. This issue is common with neural network algorithms in general, and it is not clear how a roboticist would generate such a training set from real imagery without painstaking manual efforts. Real biological systems adapt neural connections in an unsupervised way, so nature never required such a training set.

**Qian, Andersen and Adelson: 1994**

Qian, Andersen and Adelson [67] were concerned with modeling physiological and psychophysical results of experiments with motion transparency. Their motion energy
features $\xi_M(t; \Omega_M)$ were computed with Adelson and Bergen's algorithm [43] using 3D spatiotemporal Gabor filters. To account for motion transparency, their motion energy features used several possible methods of suppression or inhibition, but only within spatial frequency tunings. Orientation channels did not interact, so motion energy in one direction did not inhibit motion energy in another. In their paper, the MT model did not actually integrate motion energy into velocity features. Instead, it was concerned with MT "subunits" which performed subtractive or divisive normalization. In this dissertation, the Adelson Bergen motion energy features already include this stage. The contribution of this paper to this dissertation was evidence for a divisive normalization method for motion energy, but not motion energy integration.

These authors also extended the motion energy framework for stereoscopic disparity tuning in addition to spatiotemporal frequency. Disparity is not in the scope of this dissertation, so the stereoscopic aspect of Qian, Andersen, and Adelson's work is not considered here.

**Nowlan and Sejnowski: 1995**

Nowlan and Sejnowski's algorithm [68, 74] used a center-surround prefilter derived from a Difference-of-Gaussian (Eq. (2.28)) $P(x, y; \sigma_1 = 2, \sigma_2 = 7)$. Next, there was an Adelson Bergen motion energy computation at four orientations and nine spatial and temporal frequency combinations. Spatial filter center frequencies were distributed with octave spacing, and the temporal filters were distributed with 1.5 octave spacing. Their spatial filters were Gabor functions, and their temporal filters were Adelson and Bergen's suggestion of Eq. (2.35). These 36 motion energy features were computed at 49x49 retinotopic locations. They used a normalizing "softmax" function, unique in
this review:

\[ \hat{\xi}_M(t; \Omega_M) = \frac{e^{\alpha_1 \xi_M(t; \Omega_M)}}{\sum_{c_i, \Omega_m, \Omega_m} e^{\alpha_1 \xi_M(t; \Omega_M)}}. \] (2.46)

The effect of this normalization is to increase the largest motion energy estimates and inhibit the rest while making the total neural activity \( \sum \hat{\xi}_M(t; \Omega_M) \) constant. The gain parameter \( \alpha_1 \) determines how much inhibition to use. In the limit as \( \alpha_1 \to \infty \), this operation becomes a winner-take-all computation. The normalization was taken over all the motion energy features at location \( \tilde{\Omega}_M \), without distinguishing spatial frequency or orientation as some other authors did.

Velocity features were computed in a similar fashion to Grzywacz and Yuille [45]:

\[ \hat{\xi}_1(t; \Omega_1) = \sum_{\# \Omega_1 \times \# \tilde{\Omega}_M \times \tilde{\Omega}_M} W(\Omega_M, \Omega_1) \hat{\xi}_M(t; \Omega_M) \] (2.47)

except there is spatial pooling over a \( 9 \times 9 \) region \( \tilde{\Omega}_M \) and the weights \( W(\Omega_M, \Omega_1) \) were determined by an optimization procedure, not an explicit formulation. Velocity samplings \( \Omega_1 \) were distributed in eight directions (\( \Omega_M \) only had four) and four octave spaced speed tunings (0.25, 0.5, 1, 2 pixel frame). An additional velocity tuning corresponding to zero completed the set of 33 velocity tunings at each retinotopic location \( \tilde{\Omega}_1 \).

In this model, velocity features were also normalized

\[ \hat{\xi}_1(t; \Omega_1) = \frac{e^{\alpha_2 \hat{\xi}_1(t; \Omega_1)}}{\sum_{c_i, \Omega_1} e^{\alpha_2 \hat{\xi}_1(t; \Omega_1)}}. \] (2.48)

A unique contribution of Nowlan and Sejnowski was a third integration stage, which no other authors used and is not included in Fig. 2.16. The set of velocity estimates \( \hat{\xi}_1(t; \Omega_1) \) always had errors and ambiguities as a result of the aperture problem, motion transparency, and occlusion. To account for this, a second “selection” network operated
in parallel that was trained as part of the optimization procedure that computed a confidence measure $D(t; \Omega_1)$ for each $\xi_{\theta}(t; \Omega_1)$. A final, top level set of velocity estimates was spatially integrated with these confidence measures

$$\xi_{1:2}(t; \theta_1, r_1) = \sum_{z_i} D(t; \Omega_1) \xi_{\theta}(t; \Omega_1). \quad (2.49)$$

Notice that their entire network collapsed into a single retinotopic location at this stage. The receptive field of $\xi_{1:2}(t; \theta_1, r_1)$ is large (the whole image) and is computed using only those velocity features $\xi_{\theta}(t; \Omega_1)$ which the selection network identified as accurate at time $t$. The selection network outputs $D(t; \Omega_1)$ are not required to be spatially correlated: multiple segments of a large object could be discovered by the selection network, even considering occlusion and transparency. The integration of the whole network into a single retinotopic location is but a minor limitation: an extension to multiple receptive fields is straightforward.

**Simoncelli and Heeger: 1998**

Simoncelli and Heeger [44] used a “local contrast” prefilter

$$S(x, y, t) = \frac{I(x, y, t) - \tilde{I}(t)}{I(t)} \quad (2.50)$$

where $\tilde{I}(t)$ was the average luminance in some local spatial region $M$

$$\tilde{I}(t) = \frac{1}{\text{size}(M)} \sum_{(x, y) \in M} I(x, y, t). \quad (2.51)$$

Their linear filters $H(x, y, t; \Omega_M)$ were directional third derivatives of a Gaussian, Eq. 2.34, tuned in 28 spatiotemporal orientations at each receptive field location $\mathcal{R}_M$. Linear
features were computed in a slightly different way from the other methods in this review: the spatial filters were computed via correlation and the temporal filters using the convolution:

\[
\xi_S(t; \Omega_M) = \int \int H(x, y, t; \Omega_M) S(x, y, t - T) dx dy dT - \alpha_1
\]

where \( \alpha_1 \) was an additive constant representing a spontaneous firing rate. They defined a half squaring nonlinearity

\[
[\xi(t)]^2 = \max[0, \xi(t)]^2.
\]

The linear features were operated on by this nonlinearity and normalized

\[
\hat{\xi}_S(t; \Omega_M) = \frac{[\xi_S(t; \Omega_M)]^2}{\sum_{\ell', \ell''} [\xi_S(t; \ell', \ell'', \theta_M, \psi_M, \psi_M, \gamma_M)]^2 - \sigma_1^2}
\]

where \( \sigma_1 \) is a "semi-saturation" constant that accounted for physiological data, and computationally prevented the denominator from approaching zero in low contrast situations. Motion energy features were computed by pooling units of the same spatiotemporal orientation and phase over a local spatial region \( N_1 \):

\[
\xi_M(t; \Omega_M) = \sum_{F \in N_1} \hat{\xi}_S(t; F, \theta_M, \psi_M, \psi_M, \gamma_M) W_S(F, \theta_M, \psi_M, \psi_M, \gamma_M).
\]

Simoncelli and Heeger did not clearly specify the weights \( W_S(\Omega_M) \) other than as a pair of constraints. Velocity features were computed as

\[
\xi_V(t; \Omega_V) = \sum_{\theta_M, \psi_M, \gamma_M} W_M(F_M, \theta_M, \psi_M, \gamma_M, \Omega_V; \xi_M(t; F_M, \theta_M, \psi_M, \gamma_M)) - \alpha_2
\]
where $a_2$ is another spontaneous firing rate, and the spatial region $N_2$ is another local neighborhood in the vicinity of $F_M$. Simoncelli and Heeger did not describe these neighborhoods, nor if there was a relationship between $N_1$ and $N_2$. The weights $W_M(\Omega_M, \Omega_V)$ corresponded to an annular ring in spatiotemporal frequency space that implemented an intersection of constraints computation similarly to other methods in this review. However, each weight was modified by subtracting the mean of all the weights. This had the effect of inhibition from motion energy features $\xi_M(t; \Omega_M)$ that were far from the optimal plane. Also, the motion energy tunings $\Omega_M$ did not necessarily correspond to the tunings where motion energy was integrated. Due to the particular choice of derivatives of a Gaussian filters, motion energy features were interpolated at these new locations. The final velocity features were rectified with the half squaring nonlinearity and normalized:

$$\hat{\xi}_V(t; \Omega_V) = \frac{[\xi_V(t; \Omega_V)]^2}{\sum_{F_i \in N_2} [\xi_V(t; F_i, \theta_i, v_s)]^2 + \sigma_2}$$

where $\sigma_2$ is another semi-saturation constant.

### 2.4 Chapter Summary

In this chapter, the basic neuroscience of a mammalian early vision system has been described from an engineering perspective. The quantitative analysis introduced a set of notations used throughout the rest of this dissertation. Additionally, it provided enough background to describe the computational requirements of the simulator in the next chapter.
CHAPTER 3

PROTOTYPE IMPLEMENTATION

The information processing algorithms studied in this dissertation are closely modeled after biological processes. Real biological systems are implemented by nature in a fundamentally different "technology" than any artificial process engineers have yet created. Engineered implementations of biologically inspired algorithms tend to be awkward, as the algorithms cannot be expressed elegantly in the available analog or digital circuit technologies. Real biological systems have tremendous capability for dense interconnectivity and high parallelism of computational neurons. The emulation of biological systems must be implemented in technologies available to an engineer, despite the fact that they support limited interconnection topologies and parallelism compared to real neural systems. In addition, real biological networks code information using electrical potentials organized in impulse trains. The impulses themselves are discrete in amplitude, a feature well suited for digital computing, but the timing between the impulses is continuous, better synthesized in an analog framework. Creating application specific analog computers is expensive and time intensive. Although analog circuitry can potentially operate faster and with denser interconnect than a digital system performing the same task, the tedious development process makes analog technology unsuitable for the rapid prototyping and flexible experimentation required of this dissertation. Digital computers are the only engineering technology available at this time with adequate flexibility. Fortunately, digital computers can perform basic mathematical operations
faster than their neuron counterparts, and are capable of multiplexing interconnects to emulate the massive connectivity found in real biology. This chapter documents the hardware and software architecture used to demonstrate the concepts proposed in this dissertation with a real world prototype. Despite the poor match of implementing biologically inspired algorithms on a digital computer, this implementation achieved real time performance (≈ 11 frames per second) on a computationally intensive biological simulation.

Importantly, experienced gained with a general purpose digital system such as this one provides crucial insight into how a special purpose implementation, either analog or digital, could be built. Eventually, analog technology will probably be the most elegant way to implement algorithms such as these. In order for that to happen, engineers can use digital systems to learn how to design new computers that function in a biologically inspired way. The first analog designs will most likely emerge out of digital precursors such as this one.

3.1 Hardware

The prototype system consists of three physical entities: a robot chassis, a digital computer, and an overhead camera. This section describes these components.

3.1.1 Robot Platform

To experiment with robotic algorithms, one needs a robot. The one used here is pictured in Fig. 3.1. This chassis is a remote control model tank. A tracked platform such as this one simplifies locomotion compared to a walking design, or a robot that operates in some 3D space such as a swimming or flying robot. Although the ideas proposed in this dissertation are suitable for use with those more sophisticated robotic
platforms, there are mechanical and information processing factors associated with their use that lie beyond the scope of this dissertation. The floor based, tracked chassis was adequate for demonstrating this dissertation without these additional complications. The original tank model was stripped of its cosmetic accessories to make room for an aluminum platform upon which image acquisition and transmission hardware was mounted. All the computation was performed off-board. The only image related tasks performed on the robot chassis were the acquiring of images and their transmission, via an off the shelf NTSC radio, to framegrabber hardware (or a “TV card”) installed in the digital computer. The framegrabber hardware subsampled the NTSC image at the resolution required by the application, $100 \times 100$ square pixels. Consequently, the processor resources did not have to perform the otherwise expensive decimation task.

The CCD camera used on the robot was a small, single board device that measured less than one square inch in area. It was mounted on an aluminum post at the rear of the chassis. The post mounting hardware allowed for coarse adjustment of the vertical and horizontal view angles of the camera. The mounting was used to locate the focus of expansion at the approximate center of the acquired imagery. There was no active

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.1.png}
\caption{Remote control model tank chassis with video hardware mounted on top. The circular patterns are used to locate the robot with an overhead camera.}
\end{figure}
control of the camera orientation; once set, it was rigidly attached to the chassis so that it always pointed straight forward. The camera was mounted at the back of the chassis to reduce problems with body clipping. Initially, the camera was mounted at the front of the chassis. This caused problems because the tail of the robot would collide with obstacles after the camera stopped seeing them. After the camera passed an obstacle, it was common for the obstacle to be forgotten before the rear of the robot chassis cleared it, often resulting in a collision. With the camera mounted at the back, it was the last thing that had to clear the obstacles, so the body clipping problem due to rearward blindness was reduced.

Image transmission was performed with an inexpensive NTSC transmitter/receiver pair sold through a "security products" catalog. The robot also had a coaxial cable output for video, but the wireless system prevented the hopeless tangles that the robot would have experienced had a physical wire been used. The CCD camera output an NTSC video signal compatible with the transmitter, and the framegrabber card in the computer accepted the NTSC output from the wireless receiver. The wireless image transmission system was not of high quality nor particularly reliable. In particular, image frames were commonly dropped or distorted when the chassis motors were operating, probably due to RF emissions. Acquired imagery was also jittery due to the jerky motion of the robot. Low pass temporal filtering was used to clean up both the jitter and frames dropped by the image transmission process. Both the camera and NTSC transmitter were powered with a rechargeable battery, also mounted on the robot's platform. One additional component contained a voltage regulator because the camera and transmitter's supply requirements were incompatible. This component also worked as a distribution point for power and video signals.
The robot chassis also had a control system located under the aluminum platform with the imaging equipment, shown in Fig. 3.2. The control system included two drive motors, a radio receiver, electronics to convert received servo commands into motor control signals, and two rechargeable batteries: one for the receiver, and another for the drive motors. These components were part of the original toy application, but the receiver was modified by the supplier to use a remote control airplane radio instead of the standard type of radio typically used for ground applications. The radio modification was necessary because remote control airplane transmitters are more sophisticated. A required feature was that the radio be able to accept commands from the digital computer. Remote control airplane radios have a "trainer" port on the back, typically used in concert with a second airplane radio for a student. The first radio, operated by an experienced remote control airplane pilot, can override the student's radio at any time to protect the model airplane. Although the interface to this trainer port is proprietary, the supplier provided enough information for the electronic protocol to be reverse engineered with an oscilloscope and logic analyzer. This reverse engineering was performed, and an FPGA (field programmable gate array) was designed to accept commands from the computer's parallel port and convert them into servo commands accepted by the airplane radio's trainer port. The airplane radio used, therefore, thought that it was sending commands to an airplane from a student pilot; the reality was different. The computer controlled system worked satisfactorily.

3.1.2 Digital Computer

Table 3.1 shows a rough estimate of the computational requirements of the neural simulation used for this dissertation. Much of the notation in the table has not been described yet; it will appear in the following two chapters. For now, it is useful just
Figure 3.2: The tank chassis had control hardware mounted inside. A remote control airplane radio and custom interface chip enabled automatic control from a computer.

to report a rough idea of the computational requirements of the simulation. The estimates add up to about 770 million floating point operations per frame. For the network configuration associated with Table 3.1, and the closed loop experiments of Chapter 5, the overall simulation ran at about 11 frames per second, for a total computational throughput of about 8.5 billion floating point operations per second. The floating point coprocessor of a 933MHz Pentium-III has a maximum theoretical throughput per second of 933 million floating point operations (MFLOPS). It is clear that a single processor workstation was not adequate to run this simulation in real time. A cluster of sixteen processors, shown in Fig. 3.3, was available with features summarized in Table 3.2. The Pentium-III processor at 933MHz has a maximum theoretical floating point throughput of 933 MFLOPS using the x87 coprocessor, and a maximum rate of 3732 MFLOPS using the SSE floating point extensions. The simulation described here used a combination of
Figure 3.3: Sixteen processor computing cluster used for this dissertation

the two processor features: the FFT routines did not take advantage of the SSE extensions, but some of the multiplications and temporal filtering associated with computing $\xi_M(t; \Omega_M)$ and the multiply-accumulate operations associated with computing $\xi_V(t; \Omega_V)$ did. It was determined that for this application, the maximum theoretical throughput rating of the SSE extensions was unrealistic considering the motion energy storage requirements for $\xi_M(t; \Omega_M)$ were larger than the processor cache. Thus, memory bandwidth was the real bottleneck. In reality, the SSE version of the multiply-accumulate operations was only about twice as fast as using the x87 coprocessor version. Due to ethernet limitations, the best performance was actually achieved using only 14 of the available 16 processors. In a very rough estimate, therefore, the simulation achieved with 14 processors about half the maximum floating point throughput of the hardware.
### Table 3.1: Estimate of real-time computational requirements.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cost Symbol</th>
<th>FLOPS per frame</th>
<th># of Features</th>
<th>Total FLOPS</th>
<th>Interconnect Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus FFT</td>
<td>$F[s(t,x,y)]$</td>
<td>4480</td>
<td>1</td>
<td>4480</td>
<td>16384</td>
</tr>
<tr>
<td>Motion Energy</td>
<td>$\xi(t;\Omega_M)$</td>
<td>26,750</td>
<td>27,648</td>
<td>739,584,000</td>
<td>27,648</td>
</tr>
<tr>
<td>Velocity</td>
<td>$\xi_v(t;\Omega_v)$</td>
<td>$\approx 1200$</td>
<td>24,576</td>
<td>$\approx 29,491,200$</td>
<td>24,576</td>
</tr>
<tr>
<td>Translation</td>
<td>$\xi_T(t;\mathcal{F},\mathcal{V})$</td>
<td>$\approx 200$</td>
<td>288</td>
<td>$\approx 691,200$</td>
<td>288</td>
</tr>
<tr>
<td>Collision</td>
<td>$\xi_D(t;\mathcal{F},\mathcal{D})$</td>
<td>0</td>
<td>277</td>
<td>0</td>
<td>277</td>
</tr>
<tr>
<td>Control</td>
<td>$\xi_C(t;\mathcal{R}_C)$</td>
<td>$\approx 900$</td>
<td>42</td>
<td>$\approx 37,800$</td>
<td>0</td>
</tr>
</tbody>
</table>

1 FLOP = 1 floating point operation per frame. The FLOPS column is the computational expense per feature tuning, and the "Total FLOPS" column shows the total expense for all the features of that type. Bandwidth is the number of floating point values output per frame that usually travelled over the ethernet to the next process. The FLOP estimates are very rough.

### Table 3.2: Hardware Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Details</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>16·933MHz Pentium III</td>
<td>Dual CPU motherboards</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>1 GB per processor pair</td>
<td></td>
</tr>
<tr>
<td>Interconnect</td>
<td>100 Mbit ethernet</td>
<td>Between processor pairs</td>
<td></td>
</tr>
<tr>
<td>Video input</td>
<td>wireless NTSC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overhead input</td>
<td>NTSC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Hardware Resources
3.1.3 Overhead Imager

The location of the robot during the experimental trials was measured using an overhead camera mounted on the ceiling above the experimental arena. Measuring the location of a robot for these types of experiments is surprisingly hard, as described briefly in Section 1.1.3. In the case of this dissertation, the robot's orientation was needed in addition to the location. The method of location estimation described below works well, but does not represent a good technique for real robotic applications because the existence of an overhead camera is not a likely asset. However, the Global Positioning System (GPS) is commonly available, and provides similar information on a larger scale. The location estimation is in no way biologically inspired. The overhead camera data was used for data recording and strategic navigation, and was not related to the biologically inspired obstacle avoidance technique.

The circular patterns seen in Fig. 3.1 and illustrated more clearly in Fig. 3.4 were shapes of colored red paper pasted on a black background, also of paper. At each computational frame, the overhead image was processed to detect those red regions. Under the assumption that no other red patterns were located in the experimental arena, a binary image was created that was equal to zero, except where red was found. A pixel was determined to be "red" if the red RGB component minus the green RGB component was greater than a threshold. The green subtraction was necessary because some colors, like white, had a strong red component even though they were not actually red. The binary image constructed this way was then scanned pixel by pixel, and groups of adjacent nonzero pixels were found. Groups were weighted according to the total number of red pixels found in them, and all but the largest four groups were thrown away. In theory, there should never have been more than four, but noise, nonuniform lighting patterns, and the use of an unreliable overhead camera often resulted in random
Figure 3.4: Pattern mounted on the robot chassis for estimation of location and orientation by overhead imagery.

individual pixels being close to red. Four points were then determined by finding the average location of all the pixels in each group. With the four \((x, y)\) pairs, the center point was identified by finding the point contained by a triangle defined the other three. The point located on the tail of the robot was computed next by finding the remaining point farthest away from the center. The center point location was reported as the location of the robot \(r_L\), and the center point and the tail point were used together to trigonometrically compute the robot's orientation \(\theta_L\).

3.2 Software

The use of a parallel computer requires that the software be designed for it. Biological systems are inherently parallel with many special purpose processing neurons and extensive interconnection networks between them. The parallelism is a welcome compatibility with artificial implementations. Software is still largely serial on a particular digital processor in contrast to real biological systems which process information with every neuron dedicated to a single purpose and operating in parallel with every other neuron.
Parallelism was exploited at two levels for this simulation. At the higher level, a division was made along the lines of functionality. The simulation included models of real anatomical units like the mammalian brain areas V1 and MT, plus other modules engineered to handle input, output, and the high level processing of motion features that the neuroscience community has not yet described. These new features, described in the following chapters, are called "Rotation", "Translation", "Collision", and "Control". These high level modules were each associated with a separate code base; in other words, heterogeneous multiprocessing was used. Some of the modules, "Motion" and "Velocity" in particular, were particularly expensive to compute. They were distributed across processors as well, using homogeneous multiprocessing.

This section is specific about publicly available software used to support the simulation. Some readers may be familiar with the tools and might find it interesting, but such experience is not required to understand this section. This section is the only part of this dissertation to make these specific references. No tool mentioned here is required for a simulation of this type, but they all worked well. With the exception of the Intel C++ compiler, all the tools are either "Free" or "Open Source" and the source code is available to everybody over the Internet at the time of this writing. The Intel C++ compiler was used because at the time of this implementation, it supported the advanced features of the C++ specification better than any open source alternative. It also supported the Intel SSE floating point extensions more conveniently. The compiler’s SSE support, in particular, substantially improved computational throughput.

Most of the data processing and message passing in the simulation was in the format of multidimensional arrays. The arrays came about by sampling the multidimensional feature spaces, such as $\Omega_M$, $\Omega_V$, and $\Omega_T$, which are described later in the dissertation, and associating a rate code with each sampled location. These multidimensional arrays
<table>
<thead>
<tr>
<th>Resource</th>
<th>Specification</th>
<th>Description</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communications</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td>MPI</td>
<td></td>
<td>LAM 6.5.6</td>
</tr>
<tr>
<td>Language</td>
<td>C--</td>
<td></td>
<td>Intel C-- 6.0</td>
</tr>
<tr>
<td>Language</td>
<td>Python</td>
<td></td>
<td>gec 3.2</td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td>SSE assembly</td>
<td>Pentium III floating point extension</td>
</tr>
<tr>
<td>FFT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>FFTW</td>
<td></td>
<td>C-- and Python</td>
</tr>
<tr>
<td>Array Class</td>
<td>Blitz--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array Class</td>
<td>Numerical Python</td>
<td></td>
<td>Python</td>
</tr>
<tr>
<td>Operating System</td>
<td>Red Hat Linux 7.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>Linux 7.4.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Logging</td>
<td>NetCDF</td>
<td></td>
<td>Python and C</td>
</tr>
<tr>
<td>Data Format</td>
<td>single precision</td>
<td></td>
<td>floating point</td>
</tr>
</tbody>
</table>

Table 3.3: Commonly available support libraries and software tools used for this dissertation.

<table>
<thead>
<tr>
<th>Module</th>
<th>Language</th>
<th>Processes</th>
<th>Threads</th>
<th>Messages Sent</th>
<th>Messages Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initializer</td>
<td>Python</td>
<td>1, 1</td>
<td></td>
<td>$\Omega_M, \Omega_V, \Omega_T, D$</td>
<td></td>
</tr>
<tr>
<td>Logger</td>
<td>Python</td>
<td>1, 1</td>
<td></td>
<td></td>
<td>Everything</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Python</td>
<td>1, 1</td>
<td></td>
<td>$\mathcal{S}(t, x, y), \mathcal{F}(\mathcal{S}(t, x, y))$</td>
<td></td>
</tr>
<tr>
<td>Motion</td>
<td>C, SSE</td>
<td>6, 12</td>
<td></td>
<td>$\xi_M(t; \Omega_M), \xi_G(t; \Omega_M), \xi_S(t; \Omega_M), \xi_D(t; \Omega_M), \mathcal{H}(\Omega_M)$</td>
<td>$\mathcal{F}(\mathcal{S}(t, x, y))$</td>
</tr>
<tr>
<td>Velocity</td>
<td>C, SSE</td>
<td>6, 12</td>
<td></td>
<td>$\xi_V(t; \Omega_V), W_V(\Omega_M, \Omega_V)$</td>
<td>$\xi_M(t; \Omega_M), W_T(\Omega_V, \Omega_T)$</td>
</tr>
<tr>
<td>Rotation</td>
<td>C--</td>
<td>1, 1</td>
<td></td>
<td>$\xi_R(t; \Omega), \xi_R(t; \Omega), \mathcal{F}(\mathcal{S}(t, x, y))$</td>
<td>$\xi_V(t; \Omega_V)$</td>
</tr>
<tr>
<td>Translation</td>
<td>C, SSE</td>
<td>1, 2</td>
<td></td>
<td>$\xi_T(t; \Omega_T), \mathcal{F}(\mathcal{S}(t, x, y))$</td>
<td>$\xi_V(t; \Omega_V), \xi_R(t; \Omega_R)$</td>
</tr>
<tr>
<td>Collision</td>
<td>C--</td>
<td>1, 1</td>
<td></td>
<td>$\xi_D(t; \mathcal{F}, D)$</td>
<td>$\xi_V(t; \Omega_V)$</td>
</tr>
<tr>
<td>Overhead</td>
<td>C--</td>
<td>1, 1</td>
<td></td>
<td>$\mathcal{F}_L, \theta_L$</td>
<td>$\xi_D(t; \mathcal{F}, D)$</td>
</tr>
<tr>
<td>Control</td>
<td>Python</td>
<td>1, 1</td>
<td></td>
<td>$\xi_C(t; R_C), \xi_W(t; R_C), \xi_A(t; R_C), W_C(t; \mathcal{F}, D), R_C$</td>
<td>$\xi_D(t; \mathcal{F}, D)$</td>
</tr>
</tbody>
</table>

Table 3.4: Complete module list with implementation language and number of associated processes.
were implemented using the Blitz++ library in the C++ modules, and the Numerical Python extensions in the Python modules.

A data logging process operated in parallel with the simulation. It received messages from the rest of the simulation as if it were a computational module; but instead of performing computations, it wrote them to disk using the NetCDF protocol and library. The NetCDF software natively supports multidimensional arrays. The logger module could store every message passed between modules, although certain large message types were usually skipped to reduce the size of the logfiles. This prevented the logger from running so slow that it effected the throughput of the overall simulation. The plots in this dissertation were created with Python scripts that extracted data from these NetCDF logfiles using a Python binding of NetCDF.

3.2.1 Information Pipeline

In order to take advantage of multiprocessing, the computational tasks of the simulation were separated into the modules listed in Table 3.4. In general, each computational thread ran on a dedicated processor, although the less expensive threads shared processors. This software architecture used the information pipeline shown in Fig. 3.5. The simulation was discretized in time. Each timestep was called a frame. as one new frame was acquired from the imaging hardware of both the robot itself and the overhead system at each iteration. As each frame was being acquired, the “Motion” module, for instance, was computing the motion energy activations associated with the image it had received on the previous frame. As it took an additional frame to send the image message, the motion energy features $\xi_M(t_n; \Omega_M)$ accounted only for input up to $t = t_{n-2}$. This extra transmission delay is explained in the next section. The pipeline architecture incurred this latency at every module, so the “Control” module was actually driving
the motor plant with information ten frames, ≈ 1s, old. The advantage of the pipelined architecture was high throughput at the expense of this ten frame latency. This is a routine engineering tradeoff that has been addressed many times before in both computer hardware and software. The latency incurred is a function of software complexity: in this case, the number of serially configured modules in the pipeline. A convenient characteristic about this cost is that it becomes less significant as the frame rate increases with more hardware. In the current configuration that runs at ≈ 11 frames per second, the latency is about 1 second of real time. Faster hardware executing the same code would increase the frame rate; so the real time latency would decrease. If adequate hardware was available to compute at the full NTSC frame rate of ≈ 30 frames per second, the real time latency would only be ≈ .3 second. It will be shown in Chapter 4 that temporal filtering, and temporal filtering associated with motion energy computations in particular, introduced another latency without this convenient scaling property. In the simulation described here, the latencies associated with the motion energy temporal filters were more significant, and fundamental to the entire technique, than the implementation induced latencies of the processing pipeline. Therefore, the pipelined architecture shown in Fig. 3.5 was useful for optimizing the computational throughput with almost no effective impact on latency.

3.2.2 Message Passing

Communication between processes in the simulation was the primary software design consideration as it had the highest impact on performance. The Message Passing Interface (MPI), a common supercomputing message passing API [75], was used. It worked well for this purpose, was well documented, and high quality implementations existed for the Linux platform summarized in Table 3.2. The modules listed in Table 3.4 were
Figure 3.5: The complete information processing pipeline of the dissertation. The modules and notations are described in Chapters 4 and 5. The numbers after each colon represent the number of threads used for that module. The initializer sends data to all other modules when the simulation starts. It never sends data at any other time, and never receives data. The logger receives copies of all messages and writes them to disk in NetCDF format. The "Real World Feedback" label is used to indicate that the feedback loop is closed by the motor commands controlling the camera's point of view via real world physics. All other connections are electronic.
natural divisions considering the communication patterns. In all cases, the communication requirements between the selected modules were minimized by their choices. In some cases, a particular message type, like the velocity features \( \xi_{v1}(t; \Omega_1) \), were sent to multiple destinations. Therefore, it was efficient to implement the MPI communications at those points in the flow diagram.

Due to the overall communications bandwidth requirements, it was crucial to overlap communications with computations to make efficient use of the processors and network hardware. This overlap required using asynchronous communication between processes, a basic feature of MPI. Using asynchronous message passing, a separate thread, not listed in Table 3.4, kept the ethernet hardware supplied with data, while the other threads simultaneously consumed processor resources. Each thread running in the same process as another thread shared memory. This memory sharing saved communication, because a message only needed to be sent once to a processor pair for two threads to use. This is why the “Processes: Threads” column of Table 3.4 makes the distinction between processes and threads. The result of all these considerations was near saturation of both the processor and network resources, instead of each resource wasting time waiting for the other. A hypothetical communication pattern is shown in Fig. 3.6. The boxes represent one frame’s worth of computation associated with the module listed on the left. Each computation took some amount of time, represented along the abscissa. At the beginning of each frame \( t_n \), each module posted an asynchronous send of the results computed during the previous frame \( t_{n-1} \), represented by the arrow connected to the lower left corner of each computation representation. The arrow represents an MPI message that also takes a certain amount of time to complete. During each frame \( t_n \), the MPI message holding the \( t_{n-1} \) result was sent simultaneously with the computation of the new result. The computations shown in the figure are notated with the most
Figure 3.6: Pipeline timing.

recent stimulus image \( S(t) \) that had reached the module at \( t_n \). Notice that the last module in the pipeline, the "Control" module, was computing with information ten frames old. The "Motion" and "Velocity" modules were by far the most expensive modules; this characteristic is shown by computations consuming the most time on the plot. The best performance was achieved by assigning twelve processors to twelve threads associated with the "Motion" module, plus twelve threads associated with the "Velocity" module. All other computations were performed on another two processors, for a total of fourteen processors used. An additional two processors were available, but the extra communication required to supply those two extra processors with data was too much for the network hardware, so the total throughput was actually degraded by their use.
3.2.3 Initialization

Message routing patterns in the simulation were complex, with each module of Table 3.4 potentially generating many message types and receiving different message types from other modules. It was important for performance reasons that only the minimally required messages were sent during the simulation. Complicating the situation was that during development, the message patterns often changed, and partial simulations were often performed. It was impossible for modules to know at compile time which message types were going to be produced by other modules, and where its own messages would be sent. In addition, it was common for the network configuration to change from trial to trial, including arbitrary changes in the feature spaces $\Omega_M, \Omega_V$, and $\Omega_T$. This in turn impacted the size and format of the messages. For all these reasons, a dynamic message routing protocol was used during the simulation’s initialization phase.

As each MPI process started, it would “Publish” new message types generated by the module, and “Subscribe” to message types offered by other modules. A partial list of the message types that each module published and subscribed to are listed in Table 3.4. The most common message type was a persistent, asynchronous message that was exchanged at every frame. In Table 3.4, these message types are identified by being functions of time $t$. Other message types, used only at initialization, specified things like the network configuration in terms of the parameter spaces $\Omega_M, \Omega_V$, and $\Omega_T$, and additional parameters that specified everything including numerical thresholds and gains, logfile names, and stimulus types and parameters. Most of these message types were generated by the Initializer module. This allowed global changes to be made quickly in one place while the whole simulation would immediately reflect the modifications. Other message types included module states, usually represented as arrays of weights, like $W_v(\Omega_M, \Omega_v)$, $W_T(\Omega_V, R, \Omega_T)$, and $W_{CA}(R_C, \theta_T, D)$. Usually
these messages were only received by the “Logger” module; but in the case of the “Velocity” module, they were used to eliminate the wasted computation of particular features that were never used downstream. The “Velocity” module used the temporal filter state of the “Motion” module to compute its own weights \( W_k(\Omega_M, \Omega_V) \). In most cases, each module had to subscribe to some messages and read their values before it could initialize new messages and publish them. The modules could subscribe to and publish messages in arbitrary order, and as many times as needed before indicating it was done with the initialization phase. When all processes were done initializing, the simulation would start and only the prepared messages were sent from that time onward. Each message type was only sent to other modules that had previously subscribed. Usually, there were some message types that had been published but to which no module had subscribed. Those messages were prepared by the originating module (usually at small expense), but not sent anywhere during that trial.

### 3.3 Chapter Summary

Many engineers have often claimed that digital computers are fundamentally too slow for the real time simulation of biological processes used for real world image understanding and visually guided robots. These same engineers usually study special purpose analog circuit implementations of similar algorithms in order to achieve the required computational throughput. While analog circuits have proven to be fast, small, and power efficient, the tedious design process and technology limitations have so far prevented the implementation of large systems like the one described in this dissertation. Computers are always getting faster; their capabilities generally increase at a faster pace than analog circuit design. The computer described here is still, in fact, slower than a hypothetical analog circuit would be. However, the computer is fast enough, using
the described software architecture, to demonstrate the principle of real time, vision
based, robotic control using biological models. The flexibility of the software system, in
contrast to analog circuits, enabled the rapid implementation of a system of adequate
complexity to prove this principle: a result not yet achieved by the analog circuit com­

munity. The traditional criticisms of digital computers still apply: they are too large,
heavy, and power hungry to be carried on a robot. These problems have been addressed
for this prototype implementation largely by the wireless video and control tethers to
an offboard computer. More importantly, biological processes cannot be expressed well
on computers. The following two chapters will show that elegance notwithstanding,
the software system described in this chapter was suitable for simulating biological pro­
cesses well enough for obstacle detection, and fast enough for real time avoidance of
those obstacles.
CHAPTER 4
OBSTACLE DETECTION

Chapter 2 told a simplified story of a basic synthetic vision system, built upon prior neuroscientific experiments that tested one feature at a time using carefully controlled stimuli. Implementing a large network of previously unrelated neuroscientific models that interacted on a robotic platform, operating in the real world, introduced new problems and exacerbated the old. Exposure of these issues and the proposal of corresponding solutions described in this chapter are important contributions of this dissertation. Additionally, a novel, complete network able to compute the time to collision to real world environmental obstacles is presented. Experimental results are shown that demonstrate the features and operation of the new network operating on real imagery. This chapter deals with the computational issues of obstacle detection. Chapter 5 will build upon the obstacle detection network described here, and describe the control issues of the robot itself.

4.1 Theory of Operation

This section describes the basic mathematics of the new obstacle detection network. Some additional theory is introduced later with the experimental results in Section 4.3 because the data reveals why certain enhancements are necessary. At the end of this section is a graphic that summarizes the method.
Previous robotic vision work computed time to collision from optic flow, which is image velocity coded as an array of vector values distributed over the visual field. \[28, 76, 77, 20, 78, 79, 80, 81, 29, 82\]. Each 2D optic flow vector in the array corresponded to the estimated speed and orientation of the visual feature at a particular retinotopic location in units of pixels frame. Optic flow is a vector coded version of velocity information similar to what neuroscientists believe is represented by mammals in brain area MT as a population code. The velocity features computed here are inspired by the MT population coded version, not the traditional optic flow used in previous work. Accurate and dense optic flow has proven to be notoriously hard to compute using real world imagery due to aperture effects and diffuse regions. \[76\] Another reason optic flow computations tend to be sparse and unreliable is that the representation is 2D (consisting of speed and orientation), but the basic mathematics only provide a single constraint based on equalizing spatial and temporal derivatives. Many techniques have been proposed to generate a second constraint based on various heuristics. \[28, 76\] Two popular methods, proposed by Horn and Schunk \[83\], and Lukas and Kanade \[84\], both operate by applying a regularizing constraint on the spatial derivative of the flow mathematics. Regularization tends to introduce errors by smoothing the optic flow. Since visual occlusions generate discontinuities in the flow patterns, the smoothing causes the most useful information potentially derived from motion to be lost. Regularization does not solve the aperture problem either, although the smoothing effect often performs remarkably well in this regard for many stimuli. A population coded velocity feature, as proposed here, does not require fabricating a second flow constraint, and allows a novel, delayed resolution of aperture effects that will be described in this chapter. However,
Figure 4.1: This velocity representation shows the parameter space of $c_s \theta_V$ at an unspecified retinotopic location $\mathcal{F}_V$. The radial axis indicates velocity speed $c_s$ with slow speeds corresponding to small radii near the origin of the polar plot and fast speeds notated near the edge of the plot. Orientation $\theta_V$ is indicated by the angle of a notation on the plot. Yellow shading on a blue background represents the level of activation of a particular feature with tuning corresponding to its polar location. The red circle terminated arrow shows an optic flow vector, which exists in the same feature space.

the concept of optic flow is still useful for deriving the population coded approach proposed here. In addition, the results in this chapter are compared to traditional optic flow. The algorithm used to compute optic flow is described in Appendix A.

Figure 4.1 introduces a visualization technique used throughout this chapter to represent a feature space slice. This particular polar plot shows a portion of the velocity feature space $\Omega_V$ introduced in Section 2.3 at some unspecified $\mathcal{F}_V$. The feature space is partitioned into regions that each have some associated velocity orientation $\theta_V$, indicated by the angle of the partition on the plot, and velocity speed $c_s$ indicated by the radial distance from the center of the plot. Some of the partitions are shaded yellow in proportion to feature activations $\xi_V(t; c_s, \theta_V)$ at some time $t$. The blue partitions are inactive.

Optic flow vectors exist in the same feature space as the velocity features $\xi_V(t; \Omega_V)$. An example of an optic flow vector is shown by the circle terminated arrow in Fig. 4.1.
Although the optic flow exists in the same feature space as the population coded \( \xi_t(t; \Omega_t) \), its representation explicitly codes orientation and magnitude in a vector format. In contrast, the population coded representation represents the tunings \( \Omega_t \) implicitly: the tuning cannot be determined by the activation level \( \xi_t(t; \Omega_t) \).

Figure 4.2 generalizes Fig. 4.1 to include the retinotopic distribution of \( \Omega_t \). The figure consists of 100 polar subfigures similar to Fig. 4.1, distributed at the receptive field centers \( \mathcal{R}_t \). To maximize the available page space for figure detail, axis markings have been removed. Implicitly, the \( x \) and \( y \) coordinates both span -50 to 50 pixels in all similar figures. Figure 4.2 shows that the velocity speed distributions were not always uniform across the visual field. In fact, faster velocity speeds were more useful on the image periphery than in the middle. The velocity speed unit, pixels/frame, is not compatible with the receptive field location unit (pixel/pixel), therefore no relationship exists between the axis units and the radial dimension in the subplots. In fact, the velocity speeds themselves are not indicated on the plot; only information about relative speeds is available. Specific values of parameter space tunings are tabulated in Appendix B. The derivation of those tunings is described below in Section 4.2.

### 4.1.1 Time to Collision Features

Motion parallax can be used to compute relative depth to environmental objects. [85] Although the cited reference is a two dimensional model that includes both translational and rotational effects, we start here with a simpler 1D model, accounting only for translational motion. This dissertation handles rotational and 2D effects in a different and novel way, described below. Figure 4.3 shows the basic 1D situation from an overhead point of view. The small point shows the aperture of a pinhole camera model with focal length \( f \). The pinhole is \( Z \) cm away along the line of sight indicated by
Figure 4.2: This figure shows a blank template used in this chapter to show feature activations in the velocity feature space $\Omega_V$. The polar subfigures are retinotopically distributed over the visual field to represent $F_V$. Each cell in the subfigures will be shaded yellow to represent velocity feature activations at some instant in time. The axis labels have been removed to maximize the size of the plot: they are implicitly $-50 \leq x \leq 50$ pixels on the ordinate and $-50 \leq y \leq 50$ pixels on the abscissa.
the heavy arrow, from the plane of some environmental feature located by the large point. X cm away from the intersection between the obstacle’s plane and the camera’s line of sight. The environmental feature projects onto the camera’s image plane at \( x \) pixels. In this dissertation, the camera was mounted rigidly to the moving platform and the platform translated along the line of sight, shown by the solid arrow. Because the triangles are similar,
\[
x = f \frac{X}{Z}.
\]

Notice that \( f \) accounts for both the focal length of the camera, and the unit conversion factor between physical distance (cm) and image coordinates (pixel). The vision system does not have enough information to estimate \( Z \) directly as \( X \) is unknown. However, measurements based on optic flow, and the population coded velocity estimates \( \xi_V(t; \Omega_V) \) used here, do provide an estimate for the feature velocity
\[
v_T = \frac{dx}{dt} \tag{4.2}
\]

which in one dimension is a simple speed measurement in pixels/frame. This value \( v_T \) is called translation speed in this dissertation. Considering the temporal dynamics of Eq. (4.1).
\[
\frac{dx}{dt} = -f X \frac{dZ}{Z} \frac{dt}{dt} = -f \frac{X}{Z} \frac{dZ}{dt} = -\frac{x}{Z} \frac{dZ}{dt}. \tag{4.3}
\]

Using only visual cues, neither the distance to the obstacle plane \( Z \) nor the robot’s velocity \( dZ/dt \) is known. However, their ratio can be related to known quantities.
Substituting Eq. (4.2) and rearranging,

\[- \frac{Z}{dZ/dt} = \frac{r}{v_T} = D. \tag{4.6}\]

This new quantity $D$ is called the *time to collision* and has units of frames. Usually, the value is positive because $dZ/dt$ is negative when the robot is approaching the obstacle plane ($Z$ is decreasing). The vision system cannot actually estimate $Z$ using this method, but it can estimate when the camera will collide with the object in a perpendicular plane $Z$ pixels away. In this dissertation, time to collision is encoded as a population of retinotopically distributed $D$ tuned detectors $\xi_D(t; \bar{r}_T, D)$, computed as a simple conversion of translation features described below.

**Figure 4.3:** In one dimension, an obstacle represented by the large dot lies in a plane perpendicular to the robot’s direction of motion along $Z$. The point projects onto an imaging plane at $x$ pixels.

### 4.1.2 Translation Features

The translation features introduced here, $\xi_T(t; \Omega_T), \Omega_T = \{\bar{r}_T, v_T\}$, converted the 2D velocity estimates $\xi_1(t; \Omega_1)$ to the 1D translational speed $v_T$ at the retinotopic location $\bar{r}_T$ required for the computation of time to collision, Eq. (4.6). This subsection begins
with the situation for pure translation, and then adds the complication of rotational flows that occurred when the robot turned. In the absence of rotation and dynamic environmental features, all optic flow patterns seen by the vision system were related to the forward translational egomotion of the robot along the line of sight in Fig. 4.3. Therefore, optic flow lines should emanate radially from the focus of expansion as in Fig. 4.4. [59] In the case of a rigidly mounted camera, the focus of expansion was a known constant, located at $F = (0.0)$ in Fig. 4.4.

![Figure 4.4](image)

**Figure 4.4:** Dense, idealized optic flow patterns of an equidistant approaching scene resulting from pure translation with the focus of expansion at $F = (0.0)$. Each arrow represents the ideal speed (indicated by length) and velocity orientation of a visual feature at specific points in a hypothetical scene.

It was most useful to mount the camera on the robot such that the focus of expansion was located exactly in the middle of the captured imagery. In this dissertation, this corresponded to $F = (0.0)$ in all the stimulus figures. In the case of a ground based robot, it may be beneficial to adjust the focus of expansion in the vertical dimension towards the ground, because many surfaces like smooth floors may not have enough contrast to generate useful motion cues, unlike other environmental features above the
focus of expansion. In this dissertation, the lowest portion of the scene was actually consumed by the robot chassis itself, which never provided any useful information. This point is mentioned here, but was not considered in detail.

With a known focus of expansion and the assumption that there are no rotational motions caused by course changes or dynamic obstacles, translation speed $v_T$ can be estimated by the 2D velocity features with orientation tuning $\theta_T$ corresponding to the orientation of radial flow directed away from the focus of expansion. However, rotation artifacts resulting from the robot turning also contributed to the computation, so they will be described first.

Rotational optic flow patterns occurred when the robot turned to change course. [59] A representation of this flow component is shown in Fig 4.5(b). When the robot turned to avoid obstacles, it was usually translating and turning simultaneously, resulting in the combined expansion and rotational flow in Fig. 4.5(c). In order to keep track of obstacles, the vision system could not lose the translation patterns of Fig. 4.5(a) in the overall combined flow pattern. Ideally, one could simply subtract the constant rotational component from all the combined components, exposing the expansion components with length $v_T(\vec{F}_T)$. Unfortunately, real world motion estimates are horrendously noisy and sparse due to diffuse regions, the aperture problem, image noise, and in the case of optic flow computation, regularization heuristics. Proposed here is a population coded alternative method that had advantages in the presence of the aperture problem and visual occlusions.

Figure 4.6 shows the combined optic flow situation at a single receptive field location $\vec{F}_T$. The translational component is shown by the vector $\vec{T} = v_T \cdot \theta_T$, the rotational component is shown by the vector $\vec{R} = (R, 0)$, and an error component $\vec{e}$ is shown. When $\vec{T}$ and $\vec{R}$ represented the true translational and rotational flows, respectively,
Figure 4.5: Idealized optic flow fields induced by pure translation forward, pure rotation to the left, and translation and rotation combined. In reality, it is hard to compute optic flow with this kind of density and accuracy due to noise, the aperture problem [59], and diffuse regions in the imagery.
Figure 4.6: An expansive flow component \( \vec{T} \) combined with a rotational flow component \( \vec{R} \) defined an ideal velocity hypothesis \( \vec{C} \). An activated velocity feature in the neighborhood of \( \vec{C} \) defined by \( \sigma_C \) combined with rotational estimates of \( \vec{R} \) was evidence that \( \vec{T} \) was the true translation.

then the expected combined flow was

\[
\vec{C} = \vec{T} - \vec{R} \tag{4.7}
\]

\[
= \begin{bmatrix}
 v_T \cos \theta_T \\
 v_T \sin \theta_T 
\end{bmatrix} - \begin{bmatrix}
 R \\
 0 
\end{bmatrix} \tag{4.8}
\]

\[
= \begin{bmatrix}
 v_T \cos \theta_T - R \\
 v_T \sin \theta_T 
\end{bmatrix} \tag{4.9}
\]

When \( \vec{T} \) and \( \vec{R} \) were unknown, the vector \( \vec{C} \) represented a hypothesis that the true translational flow component was \( \vec{T} \), and the true rotation flow component was \( \vec{R} \). The velocity detectors \( \xi_V(t; \Omega_V) \) were sensitive to the combined flow \( \vec{C} \) plus the error \( \vec{\epsilon} \), \( \vec{v}_x \theta_V = \vec{T} - \vec{R} - \vec{\epsilon} \). Usually, the error component \( \vec{\epsilon} \) was significant for real world imagery. Therefore, the translation detectors \( \xi_T(t; \Omega_T) \) were designed to be sensitive to a region \( \sigma_C \) around \( \vec{C} \).

The job of the translation features \( \xi_T(t; \Omega_T) \) was to reduce the velocity features \( \xi_V(t; \Omega_V) \) to the expansive component of optic flow \( \vec{T} \), accounting for any possible rotation \( \vec{R} \). A particular \( \Omega_V \) tuning was optimal in many different flow situations, each corresponding to different rotational possibilities \( \vec{R} \). The translation detectors were
provided a rotation estimate code $\xi_R(t; R)$, and considered each $\xi(t; \Omega_V)$ in relation to their own tuning $\Omega_T$ and the whole set of $R$:

\[
\xi_T(t; \Omega_T) = \sum_{\Omega_V, R} \xi_V(t; \Omega_V) \xi_R(t; R) W_T(\Omega_V, R, \Omega_T)
\]

(4.10)

with each velocity feature tuning $\Omega_V$ corresponding to a hypothesis $\vec{C}$. The weights $W_T(\Omega_V, R, \Omega_T)$ determined the combinations of $\Omega_V, R$, and $\Omega_T$ that contributed to $\xi_T(t; \Omega_T)$:

\[
W_T(\Omega_V, R, \Omega_T) = e^{-\frac{\|\vec{r} - \vec{C} \|^2}{\sigma^2}}.
\]

(4.11)

Velocity weights computed with Eq. (4.11) are shown in Figs. 4.7 through 4.9 for $R=0.35$, and -0.35 pixels frame, respectively. Each polar subplot shows which velocity features $\xi_V(t; \Omega_V)$ were weighted to influence the translation feature $\xi_T(t; \Omega_T)$ at the same retinotopic location (i.e., $\vec{x}_V = \vec{x}_T$). It is possible that non-local velocity features ($\vec{x}_V \neq \vec{x}_T$) also could have contributed to the translation estimates, but such situations are not represented in the plots, nor were they used in the experiments. Specific tuned values of $\nu_T$ were a function of $\vec{x}_T$ and are therefore hard to compactly specify here, but they are tabulated in Appendix B corresponding to $D = 87$ frames. Figure 4.7 shows the case for $R = 0$, when there was no rotational flow and the translation weights corresponded to purely expansive velocity. Figure 4.8 shows the situation for rightward rotation $R = 0.35$, and Fig. 4.9 shows the situation for leftward rotation $R = -0.35$. These plots show why faster velocity tunings were useful for receptive field locations far from the focus of expansion. For visual clarity, fewer velocity speed tunings $v_\nu$ are shown (8) than actually used in the following simulations (16).
Figure 4.7: Translation weights that connect the velocity features $\xi_V(t; \Omega_V)$ to translation features $\xi_T(t; \Omega_T)$ when $R = 0$ and $\vec{x}_V = \vec{x}_T$. The feature space shown in the plot is $\Omega_V$. The circle terminated arrows indicate the ideal optic flow vector corresponding to $\vec{v_T}$. This figure can be compared to Fig. 4.5(a) to see the relationship between the optic flow and $\xi_T(t; \Omega_T)$ for a purely translational situation.
Figure 4.8: Same format as Fig. 4.7, but with $R = .35$ pixel frame. The arrow has been extended to show the nonzero rotational component. The ideal optic flow vector is the sum of the translational and rotational components. This plot can be compared to Fig. 4.5(c).
Figure 4.9: Same format as Fig. 4.8, but with $R = -0.35$ pixel/frame.
Figure 4.10 shows a blank visualization of the translation features $\xi_T(t; \Omega_T)$ over $\Omega_T$. The visualization is similar to Fig. 4.2, except there is only one orientation at each receptive field location $\vec{x}_T = (x_T, y_T)$. The orientation is implicitly a function of $\vec{x}_T$. $\theta_T = \tan^{-1}(y_T, x_T)$, and not an explicit tuning dimension in $\Omega_T$. Figure 4.10 is also similar to Fig. 4.4, except that it accommodates a population code instead of a vector code.

Figure 4.11 shows the expected temporal dynamics of a particular column of translation features $\xi_T(t; v_T)$ for the common situation of approaching an obstacle. As the obstacle nears, the corresponding time to collision $D$ decreases with time. For any particular column of translation detectors at $\vec{x}_T$, Eq. (4.6) shows that the translation speed $v_T$ must increase. Therefore, translation feature detectors tuned to low speeds $v_T$ fire early in time, deactivating as the obstacle approaches and translation features tuned to higher speeds activate. Because the translation features $\xi_T(t; \Omega_T)$ are related to the velocity features $\xi_v(t; \Omega_v)$, different only to account for the aperture problem and rotation, similar temporal dynamics occur in the velocity features. In the absence of rotation and the aperture problem (or when normal flow equals the true flow), the translation and velocity features have very similar dynamics. In fact, the data shown in Fig. 4.11 is actually a preview of real velocity feature activations that will be shown later in the chapter for such a situation, in Fig. 4.34.

The translation features $\xi_T(t; \Omega_T)$, were closely related to the time to collision features $\xi_D(t; \vec{x}_T, D)$ introduced in Section 4.1.1:

$$\xi_D(t; \vec{x}_T, D) = \xi_T(t; \Omega_T)$$

(4.12)
Figure 4.10: The translation features $\xi_T(t; \Omega_T). \Omega_T = \{ \vec{x}_T, \nu_T \}$ are tuned to retinotopic locations $\vec{x}_T$, shown by the spatial distribution of the "slices". and translation speed $\nu_T$, indicated by the radial dimension of the "slice". The slices suggest a feature similar to velocity that has been reduced to a single orientation $\theta_T$, implicitly related to $\vec{x}_T$ and the focus of expansion at $\vec{x} = (0, 0)$. The width of the slice has no meaning, it is nonzero only to have space for shading and to be otherwise similar to Fig. 4.2.
Figure 4.11: When approaching an obstacle, the dynamics of a population coded translation feature column $\xi_T(t; v_T)$ will show a progression of feature activations similar to the one shown here. When the obstacle is distant early in time, it activates translation features tuned to slow speeds $v_T$. As time progresses and the obstacle is approached, it appears to move faster and corresponding feature detectors fire.
when \( D = \| x_L \| v_T \), which is the same relationship described by Eq. (4.6), generalized for two spatial dimensions.

### 4.1.3 Rotation Features

The computation of the translation features \( \xi_T(t; \Omega_T) \) required the rotational estimates \( \xi_R(t; R) \) as input, the computation of which is described in this subsection. As noticed by a previous researcher, the true rotation is constant everywhere in the scene, regardless of depth, for distant environmental features and retinotopic pixels near the focus of expansion where expansive flow is inherently small. Specializing those ideas to the case when the focus of expansion is known, rotation can be estimated as an integration of the velocity features \( \xi_V(t; \Omega_V) \):

\[
\xi_R(t; R) = \sum_{\Omega_V} W_R(\Omega_V, R) \xi_V(t; \Omega_V)
\]

where the weights

\[
W_R(\Omega_V, R) = \begin{cases} 
1 - \left( \frac{v_R}{\sigma_R} \right)^2 & \text{sgn}(\cos(\theta_V)) = \text{sgn}(R) \\
-\frac{1}{2} \left( \frac{R \cdot \cdot \cdot - \cdot \cdot \cdot - \cdot \cdot R}{\sigma_R^2} \right)^2 & |\cos(\theta_V - \theta_R)| < .9 \\
0 & \text{otherwise}.
\end{cases}
\]

In Eq. (4.14), \( v \) and \( \theta_V \) were components of \( \Omega_V \), knowledge of the focus of expansion was used to compute the implicit translation orientation \( \theta_T \), and the system was relatively insensitive to the bandwidths \( \sigma_\theta \) and \( \sigma_R \) which were arbitrarily chosen to be \( \sigma_R = .15v \), and \( \sigma_\theta = \pi/2 \). The orientation of rotation was always \( \theta_R = 0 \) in this application, because the robot chassis was ground based and could therefore only generate horizontal rotation patterns. Flying or swimming robots that change elevation would require using a larger
set of $\theta_R$. In Eq. (4.14), the first exponential term eliminated those velocity tunings that would be expected by pure translation with bandwidth $\sigma_v$. Activation of these feature detectors might be due to rotation, but might also be due to translation, so they did not make good inputs to the rotation detectors. This term is illustrated in Fig. 4.12. The second exponential term is the tuning for $R$ itself, with bandwidth $\sigma_R$. The thresholds select for orientation. Figure 4.13 shows the $R$ tuned terms tuned for a particular rightward $R = .35$. Figure 4.14 shows the whole rotational weight network. Such a network ideally discovered and weighted distant environmental features and motion patterns near the focus of expansion where motion due to translation was small, and rotational motion estimates were most accurate.

### 4.1.4 Velocity Features

The velocity features $\xi_V(t; \Omega_V)$ were required to compute the rotation features $\xi_R(t; R)$ and the translation features $\xi_T(t; \Omega_T)$ as described above. These velocity features were computed with a modified version of the integration formula proposed by Grzywacz and Yuille [45] and reviewed in Section 2.3.4. The two modifications replaced their noncausal temporal Gabor filter with the cascaded exponential IIR temporal filter required for the real-world, real-time implementation, and eliminated the normal flow bias term. The bias term in the original work caused the velocity estimates to default to normal flow in the presence of the aperture problem. As Grzywacz and Yuille proposed a subsequent "winner-take-all" computation, this bias introduced a reasonable heuristic in that context. In this dissertation, however, there was no need to resolve the aperture problem at the velocity computation stage. There was no "winner-take-all" operation performed, and the population code was retained without modification for further computation of the downstream rotation and translation features. By delaying
Figure 4.12: The yellow shading represents inverse connection strengths between the velocity features $\xi_V(t; \Omega_V)$ and the rotation features $\xi_R(t; R)$. The first term of the rotation weight equation Eq. (4.14) is shown here. It discounts those velocities in which rotational information is confounded with translational information. The visualized feature space is $\Omega_V$. 
Figure 4.13: The interconnections between $\xi_V(t; \Omega_V)$ and $\xi_R(t; R)$ are sparse: only the velocity features corresponding to $\theta_V = 0$ and $v_r \approx R$ are connected. The shading represents the second term of Eq. (4.14).
Figure 4.14: The full rotation weight network $W_R(\Omega_u, R = .35)$. The shading represents Eq. (4.14), which is also the product of Figs. 4.12 and 4.13.
the resolution of the aperture problem, new errors were not introduced as a result of the common real-world case when Grzywacz and Yuille's heuristic was invalid. This delayed resolution of aperture effects can be seen by comparing Figures 4.7 and 4.14. Both weighting networks accepted $\xi(t; \Omega_V)$ as input, but connected to different sets of elements in the population code. This would not have been possible if the aperture problem was already "resolved" in the velocity computation stage.

As in the other integration models proposed in Section 2.3.4, a weighted integration of the motion energy features $\xi_M(t; \Omega_M)$ was used to compute the velocity features $\xi_V(t; \Omega_V)$:

$$\xi_V(t; \Omega_V) = \sum_{\Omega_M} W_V(\Omega_M, \Omega_V) \xi_M(t; \Omega_M).$$  \hspace{1cm} (4.15)

The velocity weights $W_V(\Omega_M, \Omega_V)$ distinguish linear integration models. This dissertation proposes a novel application of motion energy integration with unique requirements. Although the model used here was inspired by Grzywacz and Yuille, these new requirements were best satisfied by the new model presented here.

As in the former model, each velocity detector $\xi_V(t; \Omega_V)$ correlated the motion energy inputs $\xi_M(t; \Omega_M)$ at time $t$ with a template of what they would have been if the velocity tuning $\Omega_V$ was the true velocity. The template approach was attractive because it was relatively insensitive to the specific tuning parameter choices $\Omega_M$ and $\Omega_V$. If some other model was used that required a specific set of motion energy tunings $\Omega_M$ to compute a particular velocity feature $\xi_V(t; \Omega_V)$, then the velocity feature in question would have been primarily sensitive to those $\Omega_M$ tunings, and could not have taken advantage of other nearby $\Omega_M$ tunings that happened to be available and relevant. This inability to share motion energy inputs would have required a larger overall set of motion energy features, perhaps with intractable computational consequences, and would have reduced the advantages of initially using an information rich population code. The template
model avoided these problems by accepting any set of $\Omega_M$ tunings, and weighting the relevant ones in the context of the velocity tuning $\Omega_V$.

Two features of the robot application that distinguished the motion energy integration performed here and that of Grzywacz and Yuille are the use of a different temporal filter and the need to retain the velocity population code uncompensated for the aperture problem. The velocity integration features of rotation and translation required different methods of aperture problem compensation, so it was inappropriate to perform that step here. In fact, the velocity features represented a set of hypotheses, many of which could have been true simultaneously due to the confounding effects of the aperture problem. Figure 4.15 shows how multiple hypotheses were derived using the notation introduced in Chapter 2. The flow vector labeled $\nu_1, \nu_2$ is the normal flow corresponding to a linear feature that is susceptible to the aperture problem, indicated by the crosshatched region in the figure. A particular velocity hypothesis is shown as a second vector labeled $\hat{\nu}$. In this case, it is possible that the true optic flow has magnitude $\nu_1$ and orientation $\theta_1$, even though the normal flow detected by $\xi_M(t; \Omega_M)$ has magnitude $\nu_1, \nu_2$ and orientation $\theta_M$. The motion energy features $\xi_M(t; \theta_M)$ were always susceptible to the aperture problem, so the motion energy feature corresponding to normal flow always had the largest activation in the orientation dimension. The velocity integration described here considered both the situations when the normal flow was the true optic flow, and the similar situations when the optic flow was different from the normal flow as a result of the aperture problem. The velocity feature corresponding to $\theta_M$ and $\nu_2$, therefore, looked at the motion energy features corresponding to both the normal flow $\nu_1, \nu_2 = \nu_2, \theta_M = \theta_V$, and related situations when

$$\frac{\nu_1}{\nu_2} = \nu_2 \cos(\theta_M - \theta_V). \quad (4.16)$$
Figure 4.15: Geometric arguments for compensating the normal flow estimate \( \nu \), for the aperture problem to estimate the true velocity \( \nu_\nu \).

The velocity weights \( W_\nu(\theta_\nu, \theta_M) \) equaled the expected motion energy activations \( \xi_M(\nu; \Omega_M) \) when \( \Omega_\nu \) was the true velocity, including cases when the motion energy features were corrupted by the aperture problem:

\[
W_\nu(\Omega_\nu, \Omega_M) = \frac{H_s^2(f_s = \frac{\nu}{\nu_\nu \cos(\theta_M - \theta_\nu)}, \theta = \theta_M) H_t^2(f_t = \nu \cos(\theta_\nu - \theta_M) \nu_\nu)}{\sum_{\Omega_M} W_\nu(\Omega_\nu, \Omega_M)} \quad (4.17)
\]

where \( H_s(f_s, \theta) \) is the spatial frequency response function of the motion energy stage described in the polar frequency space \((f_s, \theta)\) introduced in Section 2.1.1, and \( H_t(f_t) \) is the corresponding temporal frequency response function. Equation 4.17 is the product of two squared filter magnitudes. Each filter magnitude was evaluated assuming \( \Omega_\nu \) was the true velocity and the stimulus frequency of the other domain was perfectly tuned to \( \Omega_M \). Therefore, the \( H_s(f_s, \theta) \) factor was evaluated assuming \( \nu \) was the true temporal frequency of the stimulus, and \( \Omega_\nu = \{\nu, \theta_\nu\} \) was the true velocity. In this case, the spatial frequency corresponding to normal flow was \( f_s = \nu / [\nu \cos(\theta_\nu - \theta_M)] \) and, because the motion energy detectors always computed normal flow: \( \theta = \theta_M \). Similarly, \( H_t(f_t) \) was evaluated assuming \( \nu \) was the true spatial frequency: \( f_t = \nu \cos(\theta_\nu - \theta_M) \nu_\nu \).
Both responses were squared in Eq. (4.17) as a result of Eq. (2.27), the stage of the motion energy computation that eliminated the temporal phase dependence using the identity \(\cos^2(x) + \sin^2(x) = 1\). Finally, the weights were normalized over \(\Omega_M\).

Ideally, a stimulus with velocity \(\Omega_V\) produced a motion energy response \(\xi_M(t; \Omega_M) = W_t(\Omega_V; \Omega_M)\), resulting in the maximum response of \(\xi_t(t; \Omega_V)\) from Eq. (4.15). The divisive normalization of Eq. (4.17) compensated for the inevitable variations in the number and suitability of motion energy features \(\xi_M(t; \Omega_M)\) that supported \(\xi_t(t; \Omega_V)\) and their corresponding weights \(W_t(\Omega_V; \Omega_M)\). Without it, some velocity features would have had a bias dependent on the peculiarities of the specific motion energy tunings \(\Omega_M\). Unfortunately, Eq. (4.17) was only an approximation for any real stimulus that did not have a perfectly flat spatial frequency spectrum. Any other stimulus resulted in motion energy activations \(\xi_M(t; \Omega_M)\) confounded with the stimulus contrast at spatial frequency \(\nu_M\).

In practice, the spatial filter \(H_s(f_x, \theta)\) was evaluated in the Cartesian frequency domain \((f_x, f_y)\). The corresponding arguments from Eq. (4.17) became

\[

t_x = \frac{\nu_v \cos(\theta_M)}{\nu_x \cos(\theta_V - \theta_M)}, \quad f_x = \frac{\nu_v \sin(\theta_M)}{\nu_x \cos(\theta_V - \theta_M)},
\]

and the evaluation was therefore

\[
H_s(f_x, f_y) = e^{-\frac{\pi^2 \nu_x^2}{\nu^2} \left[ (f_x - \nu_v \cos(\theta_M))^2 + (f_y - \nu_v \sin(\theta_M))^2 \right]}.
\]

Figure 4.16 shows a graphical representation of the weights of a particular velocity detector computed with Eq. (4.17). Color intensity represents the weight of a particular
\( \xi_M(t; \Omega_M) \) feature. The radius and orientation of the polar plot organize the spatial tunings \( \psi_s \) and \( \theta_M \) of the motion energy network, and the color itself shows the temporal tuning \( \psi_t \). Only the largest weight at a particular \( (\psi_s, \theta_M) \) tuning is seen; other temporal tunings also had nonzero connection strengths, but they were weaker than the ones shown.

![Polar plot diagram]

**Figure 4.16:** The velocity weights \( W_V(\Omega_M, \Omega_V) \) represent interconnection strengths between a motion energy feature \( \xi_M(t; \Omega_M) \) and a velocity feature \( \xi_V(t; \Omega_V) \). Inverse spatial frequency \( \psi^{-1}_s \) and motion orientation \( \theta_M \) are represented the polar axes, and the inverse temporal tuning \( \psi^{-1}_t \) is represented by the color. In this example, \( \psi_s = .471 \) pixel/frame, \( \theta_M = \pi/4 \) rad.

### 4.1.5 Feature Summary

Figure 4.17 shows a graphical summary of this section using the population code notations described there. In the first row, the input stimulus is processed by the motion energy filter to produce \( \xi_M(t; \Omega_M) \). The green background color represents the feature space \( \Omega_M \) to distinguish it from the other feature spaces \( \Omega_V \) (blue) and \( \Omega_T \) (red).

A single motion energy filter drives one motion energy detector; an example is shown in purple (corresponding to \( \psi_s = 1/16, \theta_M = \pi/4, \psi_t = 1/15 \)). Every other motion energy feature detector has a different motion energy filter associated with it. The second row shows how a single motion energy feature is operated on by the velocity weight.
Table 4.1: This table summarizes the parameter space tunings used in this chapter. Some of the tunings were arbitrarily determined from empirical experience, and others were set in relationship to other variables.

network to generate a velocity feature $\xi_T(t; \Omega_T)$. The third row shows the velocity feature operated on by both the translation weight and rotation weight networks to produce the translation feature $\xi_T(t; \Omega_T)$. The arrows show general information flow, but the quantitative details are not indicated. All three feature spaces $\Omega_M, \Omega_V,$ and $\Omega_T$ also include retinotopic locations $\tilde{x}_M, \tilde{x}_V,$ and $\tilde{x}_T,$ respectively, that are not shown in this figure. The computations shown here are therefore repeated at all retinotopic locations in the real simulation.

4.2 Feature Space Tuning

In Section 4.1, specific tuning choices were only specified abstractly. Discussed here is how those tunings were actually selected. Table 4.1 summarizes this section. Appendix B contains extended tables of parameter spaces $\Omega_M, \Omega_V,$ and $\Omega_T$ that are too complex to compactly show here.
Figure 4.17: This figure is a summary of Section 4.1 using the introduced notations. The three rows show the conversion of the input stimulus $S(x, y, t)$ to motion energy $\xi_M(t; \Omega_M)$, motion energy $\xi_M(t; \Omega_M)$ to velocity $\xi_V(t; \Omega_V)$, and velocity $\xi_V(t; \Omega_V)$ to translation $\xi_T(t; \Omega_T)$. 
Tuning the network started with the back end, because it minimized having to make arbitrary choices. In addition, time to collision was the most important parameter to control for system integration, not the specific feature tunings $\Omega_M$, $\Omega_U$, or $\Omega_T$. Choosing $D < 50$ was useless because the camera was mounted at the rear of the robot chassis and when the camera was 50 frames away from an obstacle, the front of the chassis was already colliding with it. Practically, $D < 75$ did not allow any time for course corrections, so $D = 75$ was chosen as the smallest time to collision tuning. The trials in this chapter were 200 frames long, so the largest tuning was chosen as $D = 200$. The system had a hard time discriminating $D > 200$ anyway because the temporal characteristics of those far away obstacles were sufficiently similar that the temporal motion energy filters $H_t(f_t)$ could not distinguish them. Humans have this same limitation; for instance, we cannot estimate the distance to a far away mountain using only motion cues when we hike towards it. Better discrimination occurred for small time to collision tunings because the higher frequency temporal filters operated with narrower bandwidths. These details will be shown in Section 4.3, but for now the conclusion was to use exponentially spaced time to collision tunings between $D = 75$ and $D = 200$:

$$D \in \{75, 87, 103, 119, 135, 151, 167, 199\} \text{.}$$  \hspace{1cm} (4.21)$$

Narrower spacing of the time to collision tunings turned out to be not useful, because the system did not have the resolution to distinguish them.

The depth tunings $D$ were identical at all retinotopic locations, a property shared only with $\gamma$, $\theta_M$, and $\theta_L$. These other parameters were constant with respect to retinotopic location only to simplify the implementation. Ideally, $\gamma$, $\theta_M$, and $\theta_L$ would vary
with $\mathcal{F}_M$ and $\mathcal{F}_V$ to better exploit the statistical characteristics of natural imagery. The implementation supported retinotopically varying tuning for $v_T$, $v_z$, and $v_\psi$.

The retinotopic tunings of $\mathcal{F}_M$, $\mathcal{F}_V$, and $\mathcal{F}_T$ were chosen to equal each other, due to the lack of any compelling reason to have it otherwise. The implementation supported different choices, but simply setting them equal minimized the total amount of computation, as each feature detector accepted input only from its identical location. It may have been useful to compute some features more densely than others, but this topic has not yet been studied. Because all three sets of locations were identical, the remaining constraint was how many locations could be computed in real time on the available computer hardware. Empirically, it was discovered that a $6 \times 8$ array was tractable, so that is what was used for the closed loop experiments of Chapter 5. For the results of this chapter, real time performance was not required, so a larger $9 \times 10$ set was used. A uniform distribution was chosen, because the translation detectors $\xi_t(t; \Omega_t)$ were more accurate on the boundaries of the visual field, but the rotation features $\xi_R(t; R)$ required information from the center near the focus of expansion. The extreme lower portion of the visual field was not included in the computations, because the front part of the robot chassis was always seen there, never providing any useful visual cues. There were undoubtably opportunities for overall performance enhancement or computational reduction by choosing the set of $\mathcal{F}_T$ more carefully after studying which retinotopic locations contribute most to the overall network activations. However, a simple regular distribution of $\mathcal{F}_T$ provided adequate performance for the experiments demonstrated in this dissertation. All the experiments shown in this chapter used $x_M, x_V, x_T \in \{-45, -35, -25, -15, -5, 5, 15, 25, 35, 45\}$ pixels, and $y_M, y_V, y_T \in \{-35, -25, -15, -5, 5, 15, 25, 35, 45\}$ pixels.
The translation speeds $v_T$ were determined by the time to collision tunings $D$ and the translation detector’s receptive field location $f_T$:

$$v_T(f_T, D) = \frac{\|f_T\|}{D}. \quad (4.22)$$

Hence, there were exactly the same number of translation detectors $\xi_T(t; \Omega_T)$ as time to collision detectors $\xi_D(t; f_T, D)$. Equation 4.22 also illustrates why the velocity speed tunings $v_s$, examined below, were functions of the receptive field locations, and there were smaller velocity speed tunings near the focus of expansion than on the peripheries of the scene.

The rotation features $\xi_R(t; R)$ were relatively cheap to compute as they were global features with only one tuning dimension $R$. However, sampling the feature space more closely than the velocity speed $v_s$, feature space did not provide more information, as the set of $v_s$ determined the maximum resolution of the rotation detectors. In fact, it was found that the velocity tuning curves were so broad that only widely spaced rotation estimates could be made. It was found empirically that a spacing of 0.2 pixels/frames contained as much information, with less potential confusion, than larger sets of $R$.

Velocity orientation $\theta_V$ was sampled in 32 uniformly distributed orientations, $\theta_V \in \{n\pi/16.0 \leq n < 32\}$. Finer orientations for $\theta_V$ were used than those for $\theta_M$ because it was found that better performance resulted if the translation detectors had input from velocity tunings with $\theta_V$ close to $\theta_T$. The velocity features $\xi_V(t; \Omega_V)$ could adequately interpolate from a reduced set of motion energy orientations $\theta_M$. Due to implementation constraints, velocity orientation tunings $\theta_V$ had to be identical at every retinotopic location $f_V$, even though some orientations were more useful than others at
any particular \( \mathcal{F}_1 \). In order to support the whole visual field and the implementation constraint, velocity orientation tunings were therefore uniformly distributed between 0 and \( 2\pi \).

Each translation feature \( \xi_T(t; \Omega_T) \) and rotation feature \( \xi_R(t; R) \) had one or more optimal velocity speed \( \nu \), for their tunings. The \( \nu_T \) tunings at every \( \mathcal{F}_1 \) were added to each \( R \), in addition to including each \( R \) by itself. The set of \( \nu \) tunings was chosen by gathering each of these optimal velocities in a set, and then thinning the set to reduce the required computation. The thinning worked by sorting the set, and repeatedly discovering subsets of three consecutive velocity tunings that spanned the smallest range of velocity speed and discarding the middle value of the subset. This was implemented with the following Python code, with \( \text{tunings} \) being a list of all possible \( \nu \), and \( \text{size} \) being the desired compute size:

```python
def thin_to_size(tunings, size):
    tunings = sort(tunings)
    while len(tunings) > size:
        min_spacing = 1e10
        for t in range(1, len(tunings) - 1):
            if tunings[t+1] - tunings[t-1] < min_spacing:
                min_spacing = tunings[t+1] - tunings[t-1]
                axe = t
        del tunings[axe]
    return tunings
```

Motion orientation \( \theta_M \) had eight samples, uniformly distributed between 0 and \( \pi \). Motion energy coded opponent energy (see Section 2.2.3), so orientations greater than \( \pi \) were represented by negative activation of the units tuned to \( \theta_M - \pi \). More rotation tunings did not appreciably increase the quality of the velocity estimation. Fewer samples were also tried informally to save computation, but this tended to negatively affect the velocity estimators for velocity orientation tunings \( \theta_T \) far from any motion orientation tuning \( \theta_M \).
Spatial frequency tunings were chosen to be logarithmically spaced between 1.4 and 11.5 pixels. For the stimuli used in these experiments, most features were between 4 and 15 pixels wide. Logarithmic spacing was used because the smaller features (usually further away) tended to grow less rapidly than large, close features. The logarithmic spacing was useful for distinguishing finer gradations in the distance, but saved on computation for closer features. There were fifteen total \( \psi_s \) samples at \( \psi_s \in \{1.4, 1.45, 1.49, 1.53, 1.57, 1.6, 1.64, 1.69, 1.74, 1.77, 1.8, 1.83, 1.89, 1.93, 1.99, 2.01, 2.09, 2.17, 2.27, 2.37, 2.47\} \). Tighter tunings than those used tended to overlap each other considering the bandwidth of the spatial filter's frequency response \( H(f_{\theta}, \theta) \) and therefore did not provide increased motion energy resolution. A subset of the even Gabor functions used are shown in Fig. 4.18. The rows correspond to different spatial frequency tunings \( \psi_s \) and the columns show the orientation tunings \( \theta_M \). All the Gaussian envelopes are sized by the spatial frequency, so \( \sigma = \psi_s^{-1} \). In the real trials shown in this section, fifteen spatial frequency tunings were used even though only eight are shown. In the next chapter, when the vision system had to run in real time, a smaller set of twelve tunings was used to improve the frame rate. Here, a larger set was used to show better potential performance even though the available computer was not able to achieve the performance in real time.

Temporal frequency tunings \( \psi_t \) were chosen to correspond to the velocity speed tunings \( v_s (\vec{f}_M) \) and spatial frequency tunings \( \psi_s \):

\[
\psi_t (\vec{f}_M, \psi_s) = \frac{v_s (\vec{f}_M)}{\psi_s}.
\] (4.23)

Equation 4.23 generated too many temporal tunings to be computationally tractable when all velocity tunings \( v_s \) and spatial frequency tunings \( \psi_s \) were included. Therefore,
Figure 4.18: A subset of the Gabor filters used to compute $\xi_G(t; \Omega_M)$ is shown, corresponding to eight spatial frequency tunings $\psi$, and eight motion orientations $\theta_M$. 
the generated set was trimmed in a similar fashion to the set of $\epsilon$, except the distance measurement was logarithmic instead of linear:

```python
def thin_to_size_log(tunings, size):
    tunings = sort(log(tunings))
    while len(tunings) > size:
        min_spacing = 1e10
        for t in range(1,len(tunings)-1):
            if tunings[t+1]-tunings[t-1] < min_spacing:
                min_spacing = tunings[t+1]-tunings[t-1]
                axe = t
        del tunings[axe]
    return exp(tunings)
```

It turned out that the temporal filter bandwidths were so broad that only six temporal tunings $\omega_i$ provided adequate coverage, shown in Fig. 4.19. Tighter tunings essentially overlapped in their frequency responses, providing no improvement in motion energy discrimination. In addition, candidate tunings greater than $1/10$ and less than $1/40$ frame$^{-1}$ were discarded because it was discovered empirically that those tunings hardly ever activated anyway using these stimuli.

### 4.3 Approach Course Trials

The first set of experiments described here demonstrate static obstacle detection without the complications of rotational flow. Figure 4.20 shows the experimental arena used for this section. The axes are calibrated in cm units on the floor where the robot operated. Three paths corresponding to three different trials are shown, although the experiments' paths were very similar. In each trial, the robot moved straight forward at approximately $0.7$ cm/frame towards an obstacle on the right. Each trial used a different obstacle pattern designated by the names Stripes, Segments, and Circles. In the overhead image, the Stripes obstacle is shown, but the other two obstacles were placed in the exact same configuration in the other trials.
Figure 4.19: Temporal motion energy frequency responses $H_t(\omega, \bar{x}_M = (5, 25))$. The poor low frequency responses are characteristic of the cascaded exponential IIR filters.
Figure 4.20: Three Approach trials demonstrated the vision system’s ability to detect collision times to three different types of obstacles. Three similar paths are shown. Robot locations at $t = 110$ are shown by the solid circle, open circle, and cross for the Stripes, Segments, and Circles trials, respectively.
Figures 4.21, 4.22, and 4.23 each show four input frames from the Stripes, Segments, and Circles trials, respectively. Of the three obstacle patterns, Stripes was the least realistic relative to the real world, but provided the best dataset for developing intuition and debugging the implementation. The motion energy approach to motion processing was particularly well suited for this type of regular grating type pattern due to the similarity with a Gabor function used in the computation of $\xi_{\Omega}(t; \Omega_M)$. It was also the most troublesome pattern for the traditional gradient and regularization based optic flow algorithms such as Lukas-Kanade, due to extreme susceptibility to the aperture problem. The Segments pattern was chosen to study how the system handles aperture effects at distinct orientations, and to try a more realistic real-world pattern that is not quite so well matched to a Gabor spatial filter. The Circles pattern was chosen to minimize aperture effects as there are no straight edges, although some effects were unavoidable when the circles got close enough to the imager. The Circles pattern also had diffuse regions when it was close to the imager.

### 4.3.1 Image Preprocessing

The first problem encountered in the Approach trials was the "bumpy ride" phenomenon introduced in Section 1.1.3. Even on smooth floors, the robot experienced small bumps and oscillations as it moved due to the rigid camera mount. These small perturbations were faithfully reproduced as high frequency jitter in the captured imagery. Smooth input was derived by preprocessing the input image with a second order, low pass, Butterworth IIR temporal filter.

The filter was beneficial as long as the image motion was small. However, the filter created a "ghost" image when the motion was too fast, particularly when the robot made fast turns. Figure 4.24 shows an example of such an image.
Figure 4.21: Approach course stimulus using the Stripes obstacle. Notice the temporal progression of high to low spatial frequencies of the obstacle as seen by the robot.
Figure 4.22: Approach course stimulus using the Segments obstacle.
Figure 4.23: Approach course stimulus using the Circles obstacle. Notice the diffuse regions that appeared on the obstacle as the robot approached.
Figure 4.24: Ghost images resulted from the temporal filtering when the robot rotated fast. The left image shows crisp features, particularly on the left side. The right image shows a similar scene, but ghost artifacts are obvious on the left.
4.3.2 Spatial Features

This subsection examines the spatial Gabor responses $\xi_G(t; \Omega_M)$ from the Approach course, on the three trials using the different obstacle patterns Stripes, Segments, and Circles. The responses to the Stripes obstacle were the simplest due to the similarity between the stimulus and Gabor impulse responses, shown in Fig. 4.25 as a function of time and spatial frequency $\nu_s$ for fixed orientation $\theta_M = 0$. As the robot approached the Stripes obstacle, the vertical stripe features appeared larger as seen in Fig. 4.21. This phenomenon is represented as a temporal progression of Gabor activations as those detectors tuned to low spatial frequencies fired later in the trial than those with high spatial frequencies. The Segments pattern did not have as much temporal distinction between spatial frequencies, as shown in Fig. 4.26. In this case, the activation was stimulated by the vertically oriented features that entered the receptive field at $\mathcal{F}_M = (5.25)$ around $t = 120$. Only a slight temporal progression is seen because the stimulus had wider spatial bandwidth that caused all the Gabor detectors to activate approximately simultaneously when the pattern entered the receptive field. However, the temporal oscillation frequencies of the Gabor features $\xi_G(t; \Omega_M)$ are distinct, with larger spatial frequencies corresponding to higher temporal frequencies, consistent with the hypothesis $c_s = c_T \cdot c_s$, where $c_s$ is constant over all spatial frequencies. The behavior of the Gabor detectors during the Circles trial was similar to the Segments trial, except there was no contrast under the receptive field for much of the trial (Fig. 4.27).

In addition to the spatial frequency variation, the three obstacle patterns also had different spatial orientation characteristics. Figure 4.28 shows $\xi_G(t; \Omega_M)$ at $\mathcal{F}_M = (25.25)$ as a function of orientation $\theta_M$ and fixed spatial frequency $c_s = 1.10.1$. Only one orientation tuning ($\theta_M = 0$) responded to the stimulus, as there was only one orientation present in the obstacle pattern. For the Stripes stimulus, all the right hand side Gabor
Figure 4.25: As the robot approached the Stripes obstacle, the spatial features got larger and the corresponding feature detectors $\xi_G(t; \nu_\alpha)$ fired. These plots show the even (solid) and odd (dashed) Gabor responses from approaching the obstacle at $\vec{x}_M = (5.25), \theta_M = 0$. The spatial frequency tunings $\nu_\alpha$ are indicated on the $y$ axis labels.
Figure 4.26: Gabor responses $\xi_G(t; \psi_\ast)$ to the Segments obstacle on the Approach course. A vertically oriented spatial pattern entered the receptive field centered at $\bar{x}_M = (5.25)$ at approximately $t = 120$. The temporal oscillations are subtly faster for larger $\psi_\ast$ tuning.
Figure 4.27: Gabor responses $\xi_G(t; \psi)$ to the Circles obstacle on the Approach course. The activations looked similar to the Segments case, except the vertically oriented pattern crossed the receptive field early in the trial.
features looked similar, regardless of their retinotopic location. Figure 4.29 shows the same plot as Fig. 4.28, but for the Segments stimulus. The Segments stimulus had a variety of orientations at \( \vec{f} = (25, 25) \) as a function of time, so many of the orientation tunings responded at some point during the trial. The Circles pattern activated the rotation tuned Gabor detectors more continuously than either of the straight edged trials, and the increasing temporal frequencies corresponding to approaching an obstacle can be seen in the plots of Fig. 4.30.

4.3.3 Motion Energy Features

The Gabor features \( \xi_G(t; \Omega_M) \) described in Section 4.3.2 were selective for spatial patterns only. The motion energy features \( \xi_M(t; \Omega_M) \) were also sensitive to temporal frequency, and insensitive to the temporal oscillations present in the spatial responses \( \xi_G(t; \Omega_M) \). Figure 4.31 shows motion energy computed from the Gabor inputs of Fig. 4.25. Four temporal bands are shown in each subplot, corresponding to the most active temporal tunings for the stimulus considered. Ideally, all the temporal channels corresponding to a particular spatial frequency tuning would have activated immediately and simultaneously at appropriate magnitudes. Unfortunately, as seen in the figure relative to Fig. 4.25, the temporal filter introduced a temporal latency that was a function of the temporal tuning \( \nu_t \). The faster channels (higher temporal frequency \( \nu_t \)) always activated first, and it is not clear at the onset of the activations which temporal channel eventually dominated. This problem results in broad velocity tuning curves for \( \xi_M(t; \Omega_M) \), which will be shown in the next section. An intuitive relationship between the spatial features \( \xi_G(t; \Omega_M) \) of Fig. 4.25 can be observed by comparison with Fig. 4.31. The periods of temporal oscillation in Fig. 4.25 are similar to the dominant temporal channel tunings \( \nu_t^{-1} \) in Fig. 4.31. Normal flow magnitude is computed from \( \nu_t/\nu_x \), so
Figure 4.28: Gabor responses $\xi_G(t;\Omega_M)$ to the Stripes pattern on the Approach course, as a function of orientation $\theta_M$. The pattern only had one orientation (vertical).
\[ \xi_G(t; \Omega_M = 25.25); \phi_s = 1.10.1 \]

Dataset: approach_segments.nc

**Figure 4.29:** Gabor response \( \xi_G(t; \Omega_M) \) to the Segments obstacle on the Approach course, as a function of orientation \( \theta_M \).
Figure 4.30: Gabor response to Circles obstacle on the Approach course, as a function of orientation $\theta_M$. 
velocity contributions from Fig. 4.31 can be anticipated by scaling each motion energy curve from \( c \) to a velocity curve tuned to \( v \). Similar plots corresponding to the Segments and Circles trials are seen in Fig. 4.32 and Fig. 4.33. These two plots are less visually appealing, but it will be shown in the next section that the data integrated into velocity tuned features nonetheless.
Figure 4.31: The motion energy responses $\xi_M(t; u_\alpha)$ from approaching the Stripes obstacle at $F_M = (5, 25)$ and $\theta_M = 0$. 
Figure 4.32: The motion energy responses from approaching the Segments obstacle at $\vec{x}_{MF} = (5, 25)$ and $\theta_{MF} = 0$.
Figure 4.33: The motion energy responses from approaching the Circles obstacle at $x_M = (5.25)$ and $\theta_M = 0$. 

Dataset: approach_circles.nc
4.3.4 Velocity Features

At the receptive field location $x_1 = (5, 25)$, the true optic flow of the Stripes trial was close the normal flow due to the special arrangement of the stimulus relative to $x_1$. In this case, a velocity integration technique that reverted to the normal flow in the presence of the aperture problem, as was the case here, would by chance correspond to the true optic flow. Although the integration technique proposed here took no special steps to revert to normal flow, it handled with relative ease the case shown in Fig. 4.34. The actual problem here with the raw velocity features $\xi_V(t; \Omega_V)$ is that the tuning for $v_r$ was very broad and lacked discriminatory quality. Although the general trend of increasing velocity with time consistent with approaching an obstacle was apparent considering $\xi_V(t; \Omega_V)$, it was hard at any particular instant for downstream algorithms to distinguish the velocity activations. In order to tighten the velocity tunings, subtle differences in $\xi_V(t; \Omega_V)$ activations were amplified by the expansive inhibition formula

$$\xi_{V,I}(t; \Omega_V) = \frac{\alpha_V \xi_V(t; \Omega_V)}{\sum_{r} \alpha_V \xi_V(t; \Omega_V)}$$

with $\alpha_V = 5$ controlling the amount of inhibition. [74] The inhibited velocity features $\xi_{V,I}(t; \Omega_V)$ are also shown in Fig. 4.34 and show a clear temporal progression of velocities consistent with approaching the obstacle. Although this inhibition technique worked well for this implementation, it demanded high dynamic range in the number representation to accommodate the exponential function. The figure shows an order of magnitude drop in activation strength in this case, but the intermediate calculations required additional dynamic range not obvious from the figure. The single precision floating point representation used in this implementation could handle the dynamic...
range requirements of the expansive inhibition, but special purpose implementations
with more modest dynamic range capabilities may require a different approach.

At $\mathbf{r}_M = (25.25)$, the motion energy activation $\xi_M(t; \Omega_M)$ for the Stripes trial was
qualitatively the same as at $\mathbf{r}_M = (5.25)$ because the stimulus pattern did not vary
in $y$. However, the true optic flow at $\mathbf{r}_V = \mathbf{r}_M$ was quite different, because in the
case of pure translation associated with the approach trials the flow was expansive,
directed away from the focus of expansion at $\mathbf{r} = (0, 0)$. Any technique that reverted
to normal flow in the presence of an aperture effect such as this one would compute
the wrong flow. Although the velocity detectors used here could not determine which
velocity feature corresponded to the true flow, all velocity tunings $\Omega_V$ that might have
been the correct flow activated. At $\mathbf{r}_V = (25.25)$ for pure translational robot motion,
the true flow corresponded to $\theta_V = \pi/4$. These velocity feature activations are shown
in Fig. 4.35 and look similar to the normal flow case of Fig. 4.34, except the velocity
tunings themselves corresponded to higher $c$. Compare the $y$ axis labels in Fig. 4.34
with Fig. 4.35 to observe this trend. The faster velocities are a consequence of the
receptive field being at a farther retinotopic distance from the focus of expansion and
Eq. (4.22).

Although it was convenient that the correct velocity tunings activated at both $\mathbf{r}_V = (5.25)$
and $\mathbf{r}_V = (25.25)$, it cannot be forgotten that other velocity tunings, that
was each $\Omega_V$ that was consistent with the underlying motion energy features in the
presence of the aperture problem, also activated. The situation at $t = 100$ is illustrated
in Fig. 4.36, which shows the effect of the aperture problem. On the right, every
rightward orientation over the striped obstacle pattern had an active velocity feature.
The linear appearance of the activation patterns is a result of velocity tunings close
to the normal flow activating for slow speeds, and orientations far from normal flow.
Figure 4.34: The velocity activations $\xi_k(t; r_*)$ from approaching the Stripes obstacle at $\vec{r}_k = (5, 25), \theta_k = 0$ are shown by the solid lines. The corresponding inhibited velocity features are shown by the dashed lines.
Figure 4.35: The velocity activations $\xi_V(t; v_1)$ and $\xi_V(t; v_2)$ from approaching the Stripes obstacle at $\vec{x}_V = (25, 25), \theta_V = \pi / 4$. 
activating for higher speeds, as expected from Fig. 4.15. The upper left quadrant had fewer aperture effects and tended towards lower speeds, consistent with the background being distant. The lower left quadrant had diffuse regions, so only a few detectors activated. Figure 4.36 also shows the theoretically correct optic flow computed with knowledge of the robot's ground velocity and distance to the obstacle (green arrows with circular termination) and the optic flow estimate computed by the Lukas-Kanade algorithm described in Appendix A (red arrows with pointed termination). Ideally, the true flow vectors would terminate on an active velocity tuning, and the Lukas-Kanade vector would correspond exactly to the true flow vector. The Lukas-Kanade optic flow algorithm provided a confidence estimate that was not used or displayed here. If the confidence estimate had been used to gate suspicious flow estimates, the effect would have been to make the flow estimates sparse, and to provide extra confidence in the remaining estimates.

Figure 4.36 is the first of three similar figures (Figs 4.36-4.38) in a series that show the temporal dynamics of the inhibited velocity activations \( \xi_{V/}(t; \Omega_V) \). The image series shows snapshots of the activations over the entire velocity space \( \Omega_V \) at three instants in time, \( t \in \{100, 150, 200\} \). The graphics show that at any particular receptive field location \( f_V \), velocity features tuned to higher speeds \( v \) tended to activate later in the trial. This is the same phenomenon shown in Fig. 4.34, except that Fig. 4.34 shows the inhibited velocity activations for all times for a single column of feature detectors, and Figs. 4.36-4.38 show all the columns at three separate instances in time.

Figures 4.39 through 4.48 show similar figures for the Segments and Circles trials. The plots demonstrate velocity activations \( \xi_{V/}(t; \Omega_V) \) at orientations different than the underlying motion energy activations \( \xi_M(t; \Omega_M) \) shown in Figs. 4.28-4.30. This is
Figure 4.36: The inhibited velocity activations $\xi_{t}(t;\Omega_{t})$ at $t = 100$ from approaching the Stripes obstacle over the full $\Omega_{t}$ parameter space. The true optic flow computed with information collected from the overhead camera is indicated by the green circle terminated arrow. Optic flow estimated by the Lukas-Kanade algorithm is shown by the red triangle terminated arrows.
Figure 4.37: The inhibited velocity activations $\xi_{11}(t; \Omega_1)$ at $t = 150$ from approaching the Stripes obstacle.
Figure 4.38: The inhibited velocity activations $\xi_{V}(t; \Omega_{V})$ at $t = 200$ from approaching the Stripes obstacle.
an important result with respect to how this dissertation handles the aperture problem. Additionally, there are situations shown such as in Fig. 4.41 at \( \tilde{x}_v = (45, 5) \) and \( \tilde{x}_v = (45, -5) \) where the velocity detectors correctly handled an occlusion. Although the receptive field centers were adjacent, feature detectors tuned to distinct velocity orientations \( \theta_v \) activated. This is the correct response, because the occlusion of the obstacle against the background resulted in a discontinuous flow pattern. Regularization often used in optic flow estimation tends to smooth such discontinuities incorrectly.
Figure 4.39: The velocity activations $\xi_v(t; v_s)$ and $\xi_{v/y}(t; v_s)$ from approaching the Segments obstacle at $\vec{r}_V = (5, 25), \theta_V = 0$. 

Dataset: `approach_segments.nc`
Figure 4.40: The velocity activations $\xi_V(t; r_s)$ and $\xi_{V'}(t; r_s)$ from approaching the Segments obstacle at $\vec{x}_V = (25, 25), \theta_V = \pi/4$. 

Dataset: approach_segments.nc
Figure 4.41: The inhibited velocity activations $\xi_{V_I}(t; \Omega_V)$ at $t = 100$ from approaching the Segments obstacle.
Figure 4.42: The inhibited velocity activations $\xi_{\gamma t}(t: \Omega_{\gamma})$ at $t = 150$ from approaching the Segments obstacle.
Figure 4.43: The inhibited velocity activations $\xi_{V}(t; \Omega_{V})$ at $t = 200$ from approaching the Segments obstacle.
Figure 4.44: The velocity activations $\xi_1(t; x_k)$ and $\xi_1(t; x_k)$ from approaching the Circles obstacle at $x_k = (5.25), \theta_k = 0$. 

Dataset: approach.circles.nc
Figure 4.45: The velocity activations $\xi_V(t; \nu_s)$ and $\xi_{VI}(t; \nu_s)$ from approaching the Circles obstacle at $\mathcal{F}_V = (25.25), \theta_V = \pi/4$. 

\[ \xi_V(t; \nu_s) = (25.25), \theta_V = \pi/4 \] 

Dataset: approach_circles.nc
Figure 4.46: The inhibited velocity activations $\xi_{V}(t; \Omega_V)$ at $t = 100$ from approaching the Circles obstacle.
Figure 4.47: The inhibited velocity activations $\xi_{\Omega}(t = 150; \Omega_{\cdot})$ at $t = 150$ from approaching the Circles obstacle.
Figure 4.48: The inhibited velocity activations $\xi_{i1}(t; \Omega_{t})$ at $t = 200$ from approaching the Circles obstacle.
4.3.5 Translation Features

The translation features $\xi_T(t; \Omega_T)$ reduced from velocity $\xi_{VT}(t; \Omega_V)$ for the Stripes obstacle are shown for $x_T = (5, 25)$ in Fig. 4.49. A $t = 110$ snapshot is shown over $\Omega_T$ in Fig. 4.51. In all the Approach trials, only one rotation tuning was used ($R = 0$) and $\xi_R(t; R = 0) = 1$ for all $t$. The added complexity of multiple rotation tunings is added for the Turn trials of the next section. The ball terminated arrow shows the correct value computed knowing the true time to collision based on overhead imagery. The pointed red arrow shows the Lukas-Kanade optic flow estimate, projected onto $v_T$. All the information shown: $\xi_T(t; \Omega_T)$, the “true” translation vector, and the Lukas-Kanade projection, are identically oriented as $\theta_T$. The orientations are tweaked on the visualization for clarity, so the arrows do not overlap each other or the $\xi_T(t; \Omega_T)$ representation. Figure 4.52 shows the velocity features $\xi_{VT}(t; \Omega_V)$ that activated the translation features shown in Fig. 4.51: plotted in the $\Omega_V$ space is $\xi_{VT}(t = 110; \Omega_V)\Omega_T(\Omega_T, \Omega_V)$ for $x_T = x_V$ and $v_T$ corresponding to $D = 87$. Also shown in green is the true optic flow vector.

Figures 4.53-4.58 show the translation features of the Segments and Circles trials.

4.3.6 Collision Features

Figures 4.59-4.61 show the collision features $\xi_D(t; x_D, D)$ computed from the translation features $\xi_T(t; \Omega_T)$ corresponding to the Stripes, Segments, and Circles trials, respectively. The obstacle was oriented perpendicular to the robot’s path. so ideally the curves would have been identical at both receptive field locations $x_T = (5, 25)$ (solid line) and $x_T = (25, 25)$ (dashed line) shown. The collision features $\xi_D(t; x_D, D)$ are the culmination of the data shown in this section in detail for those two retinotopic locations.
Figure 4.49: Translation activations $\xi_T(t; \vec{x}_T)$ from the Stripes trial at $\vec{x}_T = (5.25)$. 
\[ \xi_T(t; \vec{x}_T = (25.25)) \]

**Dataset:** approach_stripes.nc

**Figure 4.50:** The translation activations \( \xi_T(t; \nu_T) \) from approaching the Stripes obstacle at \( \vec{x}_T = (25.25) \).
Figure 4.51: The translation activations $\xi_T(t; \Omega_T)$ at $t = 100$ from approaching the Stripes obstacle. The true optic flow and the Lukas-Kanade estimate projected onto the true optic flow are shown by the circle and triangle terminated arrows, respectively.
Translation Activation. $t = 110. R = 0.00. D = 130$

Dataset: approach_stripes.nc

Figure 4.52: The inhibited velocity activations $\xi_{V}(t; \Omega_{V})$ weighted by the translation weight network $W_{T}(\Omega_{T}, \Omega_{V})$. The green arrows show the true optic flow.
Figure 4.53: The translation activations of \( \xi_T(t; \Omega_T) \) from approaching the Segments obstacle as a function of \( t_T \) at \( \tilde{x}_T = (5.25) \).
Figure 4.54: The translation activations $\xi_T(t; \Omega_T)$ from approaching the Segments obstacle as a function of $v_T$ at $\vec{x}_T = (25, 25)$. 
Figure 4.55: The translation activations $\xi_T(t; \Omega_T)$ at $t = 100$ from approaching the Segments obstacle. The true optic flow and the Lukas-Kanade estimate projected onto the true optic flow are shown by the circle and triangle terminated arrows, respectively.
Figure 4.56: The translation activations $\xi_{T}(t; \Omega_{T})$ from approaching the Circles obstacle as a function of $t_T$ at $\bar{x}_T = (5, 25)$. 
Figure 4.57: The translation activations \( \xi_T(t; \Omega_T) \) from approaching the Circles obstacle as a function of \( r_T \) at \( \bar{x}_T = (25, 25) \).
Figure 4.58: The translation activations $\xi_T(t; \Omega_T)$ at $t = 100$ from approaching the Circles obstacle. The true optic flow and the Lukas-Kanade estimate projected onto the true optic flow are shown by the circle and triangle terminated arrows, respectively.
and were the outputs of the obstacle detection system used in the obstacle avoidance behavior described in the next chapter.

A compact representation of the obstacle detection results for the three trials is shown in Figs 4.62-4.64. Each blue line shows the time to collision as functions of time and $\bar{x}_T$:

$$T_C(t; \bar{x}_T) = \frac{\sum D \xi_D(t; \bar{x}_T, D)}{\sum \xi_D(t; \bar{x}_T, D)} \quad \text{(4.25)}$$

This equation reduces the population coded collision features $\xi_D(t; \bar{x}, D)$ to a single scalar estimate of depth. This reduction was never used by the implementation for obstacle avoidance, but it was a useful visualization for intuitive appeal and debugging purposes. For adequate clarity, only data from the upper right quadrant is shown. Also shown on the plot is the true time to collision computed from overhead imagery (straight dashed red line) and the depth estimates computed from the Lukas-Kanade data (noisy dot-dashed green line). The Lukas-Kanade derived depth estimates are obviously unreliable, and do not even reveal apparent trends in most cases.

Another visualization of Eq. (4.25) is shown in Figs 4.65 through 4.67 for the three obstacle types Stripes, Segments, and Circles, respectively. Each plot uses the intensity of red shading to show times to collision at four instances in time, $t \in \{50, 100, 150, 190\}$. Brighter annotations represent smaller $T_C(t; \bar{x}_T)$. These figures are essentially depth maps, showing both the distance and location of the obstacles in the scene. Generally, the shading over the obstacles is brighter at later times, corresponding to the decreasing times to collision as the robot approached the obstacles. Also apparent is the poor performance around the focus of expansion, shown by generally bright notations regardless of depth, as well as in the diffuse regions that occurred primarily when the robot approached the Circles obstacle, resulting in no shading notations regardless of
Figure 4.59: The collision activations $\xi_p(t; D)$ from approaching the Stripes obstacle as a function of $D$ at $\vec{r}_T = (5.25)$ (solid line) and $\vec{r}_T = (25.25)$ (dashed line).
Figure 4.60: The collision activations $\xi_D(t; D)$ from approaching the Segments obstacle as a function of $D$ at $\bar{x}_T = (5.25)$ (solid line) and $\bar{x}_T = (25.25)$ (dashed line).
Figure 4.61: The collision activations $\xi_D(t; \vec{r}_T, D)$ from approaching the Circles obstacle as a function of $D$ at $\vec{r}_T = (5.25)$ (solid line) and $\vec{r}_T = (25.25)$ (dashed line).
Figure 4.62: This 5×5 grid shows the time to collision $T_C(t; \hat{x}_T)$ estimates for all the receptive fields in the upper right quadrant as the blue line from approaching the Stripes obstacle. The true time to collision is the red line, and the (poor) optic flow based estimate is green.
Figure 4.63: This figure is the same format as Fig. 4.62 from approaching the Segments obstacle.
Figure 4.64: This figure is the same format as Fig. 4.62 from approaching the Circles obstacle. Qualitatively, this data was better than expected considering the poor results demonstrated so far from approaching the Circles obstacle.
depth. The background features on the left generally did not register low $T_c(t; \vec{x}_F)$, corresponding to the extra distance between the near obstacle on the right and the background panel.

### 4.4 Turn Course Trials

In this section, obstacles were detected as the robot turned as if avoiding them to test the system in the presence of rotational flow. Figure 4.68 shows an overhead view of the course used for three trials. The trials used the same obstacle patterns as the Approach course above: Stripes, Segments, and Circles. Figures 4.69-4.71 show the stimulus patterns that were seen by the robot. The robot’s egomotion induced a rotational flow component in addition to the expansive component induced by approaching the obstacle. This can be seen, for instance, by comparing Fig. 4.21 with Fig. 4.69. The edge of the obstacle stayed around $x = 0$ during the Approach trials, but moved steadily to the right in the Turn trials. The rotation component was estimated to be about 0.2 pixel/frame in these experiments by manually examining the captured imagery and counting the number of frames it took for a visual feature near the focus of expansion to move one pixel. The obstacle detection system itself did not have access to this value; it relied on the rotational estimates $\xi_R(t; R)$. In the data shown in this section, the true time to collision was computed relative to the obstacle edge at $y=75$ cm in Fig. 4.68.

#### 4.4.1 Velocity Features

This section starts by showing the velocity features because plots of motion energy $\xi_M(t; \Omega_M)$ for the Turn trials had no intuitive appeal. In general, the formats of the plots are similar to those in Section 4.3. Snapshots of $\xi_M(t; \Omega_M)$ for $t \in \{100, 150, 200\}$ from the turning Stripes trial are shown in Figs. 4.72-4.74. The data look similar to
Figure 4.65: A depth map is shown from approaching the Stripes obstacle by plotting $T_c(t, \beta_T)$ at $t \in \{50, 100, 150, 190\}$ with short times to collision indicated by the intensity of red shaded annotations.
Figure 4.66: A depth map is shown from approaching the Segments obstacle by plotting $T_C(t; \tilde{x}_T)$ at $t \in \{50, 100, 150, 190\}$ with short times to collision indicated by the intensity of red shaded annotations.
Figure 4.67: A depth map is shown from approaching the Circles obstacle by plotting $T_C(t; \mathcal{R})$ at $t \in \{50, 100, 150, 190\}$ with short times to collision indicated by the intensity of red shaded annotations.
Figure 4.68: The overhead view of the Turn course used in this section. Three similar paths are shown. Robot locations at $t = 130$ are shown by the solid circle, open circle, and cross from the turning Stripes. Segments, and Circles trials, respectively.
Figure 4.69: Turn course stimulus using the Stripes obstacle.
Figure 4.70: Turn course stimulus using the Segments obstacle.
Figure 4.71: Turn course stimulus using the Circles obstacle.
Figs. 4.36-4.38, except that there were more velocity speed tunings $c$, and there was a rotation offset of about 0.2 pixels per frame in the horizontal direction due to the rotational flow. This offset is shown explicitly as a new horizontal segment associated with the true flow vector.

In Figs. 4.72-4.74 the Lukas-Kanade optic flow estimates shown by the red triangle terminated arrows generally found magnitudes and orientations consistent with the local visual pattern. Unfortunately, the vectors were usually different from the true flow as a result of the aperture problem. The population coded version generally found all of the consistent flow patterns. Neither estimate provided optic flow estimates that were both correct and unique in the presence of the aperture problem. However, the population coded estimates at least included the correct flow pattern when the Lukas-Kanade estimate arbitrarily selected a single vector from the many possibilities. Downstream processing of the Lukas-Kanade estimate was already generally crippled. In contrast, the algorithms based on the population codes $\xi_{1}(t; \Omega_{1})$ had the opportunity to select the correct estimates with additional information provided by their own private contexts.

The velocity snapshots of the Segments trial shown in Figs. 4.75-4.77 show more of the aperture problem, except many orientations were involved. The above analysis paragraph is in general true for this case as well, except the Lukas-Kanade algorithm tended to do better due to the presence of multiple orientations and visible feature terminations: both characteristics tended to mitigate the aperture problem somewhat. Examples of extremely erroneous Lukas-Kanade estimates were still available, however, over particular receptive fields where the aperture problem still dominated.

The activation patterns of $\xi_{1}(t; \Omega_{1})$ for the Circles obstacle in Figs. 4.78-4.80 are surprisingly linear considering the obstacle, although some subtle curvature is apparent in Fig. 4.78. The stimulus was designed with the expectation for fewer aperture effects.
but it turned out that at this scale, the circular edges were straight enough that the
motion patterns were still ambiguous. Not surprisingly considering the variety of ori­
entations in the stimulus. The Lukas-Kanade algorithm performed best on the Circles
pattern.

4.4.2 Rotation Features

The Turn trials were more complicated than the Approach trials, and this is the
first time in this dissertation that rotation was a factor. Figure 4.81 shows the rotation
features $\xi_R(t; R)$ computed in the Stripes trial. Because the scale of the rotation features
$\xi_R(t; R)$ was dependent on the $\xi_1(t; \Omega_1)$ activation levels, and because there was always
a true rotation value at any $t$ (maybe corresponding to $R = 0$). A normalized version
was computed:

\[
\xi_{R1}(t; R) = \frac{e^{\alpha R \xi_R(t; R)}}{\sum_R e^{\alpha R \xi_R(t; R)}}.
\]

A small offset was added to $\xi_R(t; R = 0)$ to avoid numerical problems in low contrast
situations (and simulator startup) and to set a default rotation estimate. Raw rotation
estimates $\xi_R(t; R)$ are shown in Figs. 4.81 and 4.82 for the Stripes and Segments trials,
respectively. The normalized versions were $\xi_{R1}(t; R = 0.2) = 1$ and $\xi_{R1}(t; R \neq 0.2) = 0$
in both of these trials.

A $t = 130$ snapshot of the rotation weight activation $\xi_{11}(t; \Omega_1) W_R(R = 0.2; \Omega_1)$ is
shown in Fig. 4.83. It was found that the tuning curves of $\xi_R(t; R)$ with respect to $R$ were
very broad, causing many closely spaced $\xi_R(t; R)$ features to activate simultaneously. In
turn, this caused the translation estimates $\xi_T(t; \Omega_T)$ to have very poor discriminatory
power, because it never had a good estimate of $R$. Therefore, the set of $R$ was reduced to
Figure 4.72: The inhibited velocity activations $\xi_{V I}(t = 100; \Omega_V)$ at $t = 100$ from the turning Stripes trial over the full $\Omega_V$ parameter space. As in the similar data from the Approach trials, the aperture problem is apparent on the right. The upper left quadrant had fewer aperture effects and tended towards lower speeds, consistent with the background being distant. The lower left quadrant had diffuse regions, so few detectors activated.
Figure 4.73: The inhibited velocity activations $\xi_{i\gamma}(t; \Omega)$ at $t = 150$ from the turning Stripes trial.
Figure 4.74: The inhibited velocity activations $\xi_{t/i}(t; \Omega_t)$ at $t = 200$ from the turning Stripes trial.
Figure 4.75: Inhibited velocity activations $\xi_{V}(t; \Omega_{V})$ at $t = 100$ from the turning Segments obstacle.
Figure 4.76: Inhibited velocity activations $\xi_{\gamma}(t; \Omega; \cdot)$ at $t = 150$ from the turning Segments obstacle.
Figure 4.77: Inhibited velocity activations \( \xi_{V/T}(t; \Omega_V) \) at \( t = 200 \) from the turning Segments obstacle.
Figure 4.78: Inhibited velocity $\xi_f(t; \Omega_1)$ activation at $t = 100$ for the turning Circles obstacle.
Figure 4.79: Inhibited velocity $\xi_{V I}(t; \Omega_V)$ activation at $t = 100$ for the turning Circles obstacle.
Figure 4.80: Inhibited velocity $\xi_{V}(t; \Omega_{V})$ activation at $t = 100$ for the turning Circles obstacle.
only three components: \( R \in \{-0.2, 0, 0.2\} \). Accuracy in depth estimation was therefore traded for tighter tuning curves of \( \xi_I(t; \Omega_I) \).

![Graph](image)

**Figure 4.81:** The rotation features activations \( \xi_R(t; R) \) from the turning Stripes trial. A bias of .5 was added to \( \xi_R(t; R = 0) \) to stabilize the division of \( \xi_R(t; R) \) and provide a default value in low contrast situations.

### 4.4.3 Translation Features

The translation features \( \xi_T(t; \bar{x}_T = (5, 3.5), v_T) \) and \( \xi_T(t; \bar{x}_T = (3.5, 3.5), v_T) \) are shown as a function of time for the Stripes trial in Figs. 4.84 and 4.85. The general trend of increasing translation speed as the robot approached the obstacle was apparent, showing that the confusing rotational flow effects were reduced: compare to Fig. 4.49 that did not have any rotational flows included. The translational estimates in Fig. 4.85 also show the correct trend, accounting for both the aperture problem and rotational flows simultaneously. Notice that the translational speed tunings \( v_T \) were larger in Fig. 4.85 than in Fig. 4.84 even though the stimulus pattern under the receptive field apertures were similar: this happened because the receptive field at \( \bar{x}_T = (3.5, 3.5) \) was further from the focus of expansion than the receptive field at \( \bar{x}_T = (5, 3.5) \).
Figure 4.82: The rotation feature activations $\xi_R(t; R)$ from the turning Segments trial. A bias of 1 was added to $\xi_R(t; R = 0)$ to stabilize the division of $\xi_R(t; R)$ and provide a default value in low contrast situations.

The turning Segments trial also showed an advancing $v_T$ trend in Fig. 4.86 similar to Fig. 4.84 despite the additional complication of separate visual features with different associated spacings and orientations passing through the receptive field (to see those, refer to Fig. 4.70). An unfortunate arrangement of visual features produced unsinusoidal behavior at $x_T = (35, 35)$ for the Segments stimulus at $t = 130$ and $t = 170$, although the advancing $v_T$ trend is still visible.

The performance of the translation detectors was the worst for the Circles obstacle, shown in Figs. 4.88 and 4.89. The translation detectors were hindered at $x_T = (5, 35)$ for $t > 175$ due to diffuse regions.

Translation snapshots for the three trials at $t = 130$ in the upper right quadrant of the visual field are shown in Figs. 4.90-4.92. Many of the activation vectors located at a particular $x_T$ were indistinguishable in terms of activation level at $t = 130$. This is a result of the tuning curves associated with $\xi_T(t; \Omega_T)$ being too broad in $v_T$. This broad tuning behavior was present in all of the features in the vision system, tracing all the
Figure 4.83: The inhibited velocity features $\xi_1(t; \Omega_1)$ that contributed to $\xi_R(t = 130; R = 0.20)$ during the Stripes trial. Because the rotation flow corresponding to $R = 0.2$ and translation are easily confused in the right half of the visual field, the active velocity features are only found near the focus of expansion ($x = 5$).
Figure 4.84: The translation feature activations $\xi_T(t; \Omega_T)$ from the turning Stripes trial as a function of $v_T$ at $\bar{r}_T = (5.35)$. The feature detectors activated for long time periods because the tuning in $v_T$ was broad, reducing discriminatory power.
Figure 4.85: The translation activations $\xi_T(t; \Omega_T)$ from the turning Stripes trial as a function of $\Omega_T$ at $\bar{x}_T = (35, 35)$. These features were computed despite considerable aperture effects in the stimulus and rotational flows.
Figure 4.86: The translation activations $\xi_T(t; \Omega_T)$ from the turning Segments trial as a function of $v_T$ at $\tilde{f}_T = (5.35)$. 

Dataset: turn_segments.nc
Figure 4.87: The translation activations $\xi_T(t; \Omega_T)$ from the turning Segments trial as a function of $v_T$ at $x_T = (35, 35)$. The sudden deactivation around $t = 170$ was the result of a stimulus feature leaving the receptive field at $x_T$ and another feature of a different orientation entering shortly after.
Figure 4.88: The translation activations $\xi_T(t; \Omega_T)$ for the Circles trial as a function of $r_T$ at $\tilde{F}_T = (5.35)$. 
Figure 4.89: The translation activations $\xi_T(t; \Omega_T)$ for the Circles trial as a function of $v_T$ at $\bar{r}_T = (35, 35)$. 

Dataset: turn.circles.nc
way back to the temporal motion energy filters \( H_t(t; \Omega_M) \). The weights contributing to translation features corresponding to \( D = 133 \) frames. \( R = 0.2 \) at \( t = 130 \) are shown in Fig. 4.93.

### 4.4.4 Collision Features

The translation features \( \xi_D(t; \vec{x}_T = (5.25), D) \) are shown as a function of time in Figs. 4.94-4.96. Data from \( \vec{x}_T = (5.35) \) and \( \vec{x}_T = (35.35) \) are shown in the same plots together, because the visual features under these receptive fields always corresponded to the same obstacle, and the depth, therefore, should have been approximately equal. Considering that the relative magnitudes of activation are what matter for depth discrimination, differences in scale can be discounted.

As in Section 4.3, the results of this section are summarized in Fig. 4.97 through Fig. 4.99 that show a reduction of the collision features \( \xi_D(t; \vec{x}_D, D) \) using Eq. (4.25). The Stripes obstacle resulted in the best depth estimates, with smooth decreasing trends in \( T_t(t; \vec{x}_T) \) corresponding to approaching the obstacle. A few receptive field locations produced particularly poor depth estimates: \( \vec{x}_T \in \{ (10.10), (25.25), (35.25) \} \). For navigation, these poor estimates will be discounted because they are generally inconsistent with their retinotopic neighbors. The reduction process also tended to amplify these anomalies, because one erroneous detector could make it appear that the other detectors at the same receptive field location were also erroneous, even though that may not have been the case. The system had a harder time with the Segments obstacle, although the data is sufficient for obstacle avoidance. The Circles obstacle generally defeated the depth detection system. In all cases, the depth estimates computed from the Lukas-Kanade algorithm were worthless for depth detection and obstacle avoidance.
Figure 4.90: Translation activation $\xi_T(t; \Omega_T)$ at $t = 130$ for the turning Stripes trial. The true optic flow and the Lukas-Kanade estimate projected onto the true optic flow are shown by the circle and triangle terminated arrows, respectively.
Figure 4.91: Translation activation $\xi_T(t; \Omega_T)$ at $t = 130$ for the Segments trial. The true optic flow and the Lukas-Kanade estimate projected onto the true optic flow are shown by the circle and triangle terminated arrows, respectively.
Figure 4.92: Translation activation $\xi_T(t; \Omega_T)$ at $t = 130$ for the Circles trial. The true optic flow and the Lukas-Kanade estimate projected onto the true optic flow are shown by the circle and triangle terminated arrows, respectively.
Translation Activation, $t = 130$, $R = 2.00$, $D = 133$

Dataset: turn_stripes.nc

Figure 4.93: Stripes translation activation over $\Omega_Y$ and $\Omega_T$ for $R = 0.2$: $W_T(\Omega_T, \Omega_Y, R)\xi_Y(t; \Omega_Y)$. 
Figure 4.94: Collision activation of $\xi_D(t; \bar{x}_T, D)$ from the turning Stripes trial as a function of $D$ at $\bar{x}_T = (5.35)$ (solid line) and $\bar{x}_T = (35.35)$ (dashed line).
Figure 4.95: Collision activation of $\xi_D(t; \bar{x}_T, D)$ from the turning Segments trial as a function of $D$ at $\bar{x}_T = (5.35)$ (solid line) and $\bar{x}_T = (35.35)$ (dashed line).
Figure 4.96: Collision activation of $\xi_D(t; \bar{x}_T, D)$ from the turning Circles trial as a function of $D$ at $\bar{x}_T = (5, 35)$ (solid line) and $\bar{x}_T = (35, 35)$ (dashed line).
Figure 4.97: The time to collision $T_c(t; x_T)$ estimates from the turning Stripes trial for all the receptive fields in the upper right quadrant as the blue line. The true time to collision is the straight red line, and the (poor) optic flow based estimate is green.
Figure 4.98: The time to collision estimates $T_{C}(t; \bar{x})$ from the turning Segments trial in the same format as Fig. 4.97.
Figure 4.99: The time to collision estimates $T_C(t, \bar{x}_T)$ from the turning Circles trial in the same format as Fig. 4.97.
Also similar to Section 4.3, the time to collision estimates are visualized in Figs 4.100 through 4.102 for the three obstacle types Stripes, Segments, and Circles, respectively. Each plot again uses the intensity of red shading to show times to collision at four instances in time, \( t \in \{50, 100, 150, 190\} \). The first panel in each figure, corresponding to \( t = 50 \), shows the tail end of a troublesome transient that occurred when the obstacle patterns initially appeared. This phenomenon, due to the motion energy filters tuned to high temporal frequencies \( \nu \); activating when features appeared, resulted in erroneous depth estimates early in the trials.

4.5 Chapter Summary

In this chapter it was shown than one can use established neuroscientific models of mammalian brain areas V1 and MT to locate obstacles in real world imagery captured by a mobile robot. It was shown that population coding offers certain advantages over the traditional scalar methods of coding optic flow in the presence of the aperture problem. Furthermore, population coding offers a way to compensate for wide field rotation patterns caused by a robot's egomotion. Population codes offer a way to delay the resolution of common machine vision problems, for instance, the aperture problem and rotational flow, to downstream computations where private considerations offer additional information that can be used. In this chapter, this population coding feature was used to retain all the possible velocity features in the presence of the aperture problem. These features were then resolved feature by feature using special considerations in the rotation and translation computations. The result was successful obstacle detection in the form of a population coded depth map. In contrast, the corresponding obstacle detections derived from traditional optic flow were clearly unusable; a quantitative
Figure 4.100: A depth map is shown from the turning Stripes obstacle by plotting $T_C(t; \bar{x}_T)$ at $t \in \{50, 100, 150, 190\}$ with short times to collision indicated by the intensity of red shaded annotations.
Figure 4.101: A depth map is shown from the turning Segments obstacle by plotting $T_C(t; \vec{x}_T)$ at $t \in \{50, 100, 150, 190\}$ with short times to collision indicated by the intensity of red shaded annotations.
Figure 4.102: A depth map is shown from the turning Circles obstacle by plotting $T_C(t; \bar{x}_T)$ at $t \in \{50, 100, 150, 190\}$ with short times to collision indicated by the intensity of red shaded annotations.
measurement was not even required. The next chapter will use the obstacle detection result described here, \( \xi_D(t; \tilde{x}_I, D) \), as input to an autonomous robot controller.
CHAPTER 5

ROBOT AUTONOMY

Chapter 4 demonstrated a novel, biologically inspired depth detection system based on motion processing. Although the experiments shown there used real imagery captured by a mobile robot, they were performed in open loop configurations so the obstacle detection results did not influence the succeeding imagery. This chapter takes the obvious step of using a closed loop configuration, where the imagery was captured by the robot, obstacles were detected in real time, and the results were used to immediately influence the robot’s navigational course to avoid the detected obstacles. For the first time in this dissertation, autonomous behavior is demonstrated.

5.1 Control Features

The control system accepted a population coded collision feature input $\xi_D(t; \tilde{x}_T, D)$ described in Chapter 4 and produced a new population coded control vector $\xi_C(t; R_C)$, where $R_C$ was the robot’s turn rate and sole component of the feature space, by doing an integration of the input code. The feature space of the control code, $R_C$, represented the possible commands that the robot chassis accepted. For the floor based, tracked robot used here this was the ground speed $S_C$ and turn rate $R_C$. Robots that operate in a three dimensional environment, such as flying or swimming robots, would also require at least an elevation control, and perhaps other parameters as required by the platform. A simplification used here was to keep the speed $S_C$ constant so the robot
always moved forward at the same rate. Therefore, a one dimensional feature space over $R_c$ was adequate to represent a complete control solution. The real effect of the specific value of turn rate parameter $R_c$ was completely dependent on the control hardware of the robot chassis. The software used $-1 \leq R_c \leq 1$ for convenience, but the value was arbitrarily scaled by the robot chassis' electronics to a real turning rate in rad/s units. In addition, the control electronics used here were of poor quality and the real turning effect of the $R_c$ commands was nonlinear in $R_c$, dependent on the battery state, and prevented a good characterization of the robot platform's locomotion system. Nevertheless, an $R_c$ configuration in eight partitions (42 were actually used) is shown in Fig. 5.1. It shows the boundaries of the partitions as tick marks on the curved $R_c$ axis. As a convention, $R_c = 0$ meant to maintain the current heading, $-1 \leq R_c < 0$ meant to turn left at rate $|R_c|$, and $0 < R_c \leq 1$ meant to turn right at rate $R_c$. Even though the partitions of the $R_c$ feature space were discrete, the represented course was not quantized. In the figure, both control features tuned to $R_{c1}$ and $R_{c2}$ activated, as the course to the waypoint corresponded most closely with the control feature $\xi_c(t; R_c = R_{c2})$ with a bias towards the $R_{c1}$ partition.

The control code $\xi_c(t; R_c)$, the computation of which is described later in this chapter, represented the control configuration desired by the robot’s navigational algorithm. However, the control code $\xi_c(t; R_c)$ was also the last set of population coded features that the robot’s algorithms computed before sending a command to the robot platform’s hardware interface. The servo controllers on the off-the-shelf robot chassis used here knew nothing of population codes, so $\xi_c(t; R_c)$ was reduced to a single scalar value compatible with the robot’s mechanism:

$$C_{f}(t) = \frac{\sum_{R_c} \xi_c(t; R_c)R_c}{\sum_{R_c} \xi_c(t; R_c)}.$$  \hspace{1cm} (5.1)
Thus, as far as the robot’s hardware was concerned, the scalar value $C_f(t)$ represented the result of the entire information processing chain of this dissertation. The scalar reduction of Eq. (5.1) did not handle well the case where $\xi_c(t; \theta_c)$ was multi-mode. The multi-mode situation arose when the control model activated multiple control features $\xi_c(t; \theta_c)$ tuned to widely separated $\theta_c$ tunings due to the lack of a unique control solution. This occurred if the $\xi_c(t; \theta_c)$ controller outputs were not of high quality since Eq. (5.1) could not fix them. The robot subsequently received a command that was the average of the activated control feature tunings, usually causing a collision or a situation where the robot left the experimental arena.

5.2 Strategic Navigation

Obstacle avoidance is a secondary (yet critical) component of robot navigation. Even in the absence of obstacles, the robot must have some overall strategic goal for navigation or it would never have any reason to move anywhere at all. Some robots in the literature have had a strategic goal to simply keep moving, or “wandering”, without concern for the final destination. [9, 20, 10] Although this can be an interesting result due to the primitive state of the art in autonomous robotic navigation, most useful applications of autonomous robots will require the ability to control where the robot is
eventually going. Unfortunately, integrating wandering behavior with obstacle avoidance is actually an easier problem to solve than with a more specific strategic goal. Another goal might be to visually (or non-Visually) track and follow a moving target. One robot even played “tag” [10]. Although this strategic goal could be useful in many real applications, the added difficulty of target recognition adds complexity to the experiment, and the target recognition performance may be confounded with navigation performance. The strategic goal considered here was that of waypoint finding [7], a generally useful strategic goal which in real applications could be easily implemented using the Global Positioning System (GPS). Basically, the robot’s navigation system was provided a continuous estimate of where it was relative to some preprogrammed waypoint. In the absence of obstacles, it was a trivial matter to trigonomically orient the robot towards a preset waypoint and detect when the waypoint was achieved. This is similar to “docking” goals [12].

The waypoint finding behavior implemented for these robot experiments was guided by an overhead camera. Although an overhead camera is not a realistic asset for real robotic applications, GPS is commonly available and provides very similar information. An overhead camera was used here in lieu of GPS only because the laboratory arena was too small relative to the spatial resolution of GPS for accurate waypoint finding and performance measurement. The details of the overhead camera implementation were described in Section 3.1.3. Figure 5.2 shows the view of the experimental arena as seen by the overhead camera. There were four preset waypoints indicated in the figure by cross symbols arranged in a square pattern. The number and placement of the waypoints was arbitrary and programmed in advance of the experiment. Waypoints could easily have been computed and changed on the fly as the robot operated, but there was no need for that here.
Figure 5.2: The experimental arena was defined by the overhead camera view shown here. Four waypoints are shown as crosses, with circles around them showing the regions that the robot had to enter to "achieve" the waypoint. The robot chassis is also shown on the left, arranged as at the beginning of all trials shown in this chapter. The four circular patterns on the robot chassis were used to compute where the robot was at any given instant along with its orientation. The specific locations reported here correspond to the center circle on the chassis, and the same center circle is the point on the chassis that had to enter the waypoint regions before the next waypoint was selected.
In anticipation of a second type of behavior, obstacle avoidance, the waypoint finding behavior used a private control code $x_{cw}(t; R_C)$ to represent the output of the strategic controller by itself. The interpretation of $x_{cw}(t; R_C)$ was identical to $x_c(t; R_C)$, and in the absence of any other behavior they were almost identical, with the exception of a normalization procedure described below. The overall control vector $x_c(t; R_C)$ was an integration of all implemented behaviors, so a separate control signal was convenient for each navigation behavior. The integration of multiple behaviors is described in the next section. As the location of the robot was known from the overhead camera, there was a simple trigonometric relationship to compute a course correction $\theta_w$ from Fig. 5.1 to the next waypoint. In the hypothetical case of the figure, $R_{C1}$ and $R_{C2}$ were the two control partitions tuned for $\theta_w$. Thus, both $x_{cw}(t; R_{C1})$ and $x_{cw}(t; R_{C2})$ were activated where $x_{cw}(t; R_C)$ were the waypoint control features defined by the relation

$$x_{cw}(t; R_C) = e^{-\frac{(R_c - \sigma_R)^2}{2\sigma_R^2}}$$

where $\theta_w$ has been normalized by $\pi$ to be compatible with $R_c$, and $\sigma_R$ represents the width of the partition. A more sophisticated model might have used a non-uniform partition width $\sigma_R(R_C)$ along with an accurate model of the robot’s motor controller, but that was not done here.

In the absence of any obstacles, the robot was programmed to travel between the waypoints in a square pattern for the Waypoints trial. The experimental results are shown in Fig. 5.3. The figure has the same format as Fig. 5.2, except the path of ten circuits around the arena is shown. The thick vertical lines are lengths of colored tape stuck to the floor that indicate the future placement of obstacles. Notice that the paths of Fig. 5.3 intersect those markings, particularly when considering the width
of the robot chassis and the potential for body clipping. The wide variation in paths from circuit to circuit is a result of low control loop gain. The intrinsic unpredictability of this particular robot chassis design and low frame rate of the computations ($\approx 11$ frames/second) made higher loop gains unstable.

A subset of the control feature activations $\xi_C(t; R_C)$ and $\xi_{CW}(t; R_C)$ is shown in Fig. 5.4 corresponding to a left turn from the Waypoints trial. The overhead path results are shown in Fig. 5.3. A total of 42 uniformly spaced control features were computed for a resolution of 0.05 in $R_C$. Only the six most active control features during the interval $800 \leq t \leq 900$ are shown in the figure. The remaining features not shown were mostly inactive. During the straightaway shown between $t = 800$ and $t = 840$, the two control features tuned to $R_C = -10$ and $R_C = -15$ were most active, encouraging the robot to make a slight leftward course correction towards the waypoint. Low loop gain prevented the minor correction, or else $\xi_C(t; R_C = 0)$ would have been most active during the straightaway after the minor corrections had taken place. When the robot achieved the waypoint at $t = 840$, however, a large correction was required to reorient itself towards the next waypoint. As the robot turned left, the heading approached the new strategically desired course and eventually the control features tuned to lower turn rates activated, diminishing the scalar control signal $C_T$. Eventually, at $t = 870$, the course was very close to correct to achieve the next waypoint and the robot resumed a straight course. In the absence of obstacles, the waypoint control features $\xi_{CW}(t; R_C)$ were very close to the total control features $\xi_C(t; R_C)$, differing a small amount only due to a normalization process with more than one active control feature. Both the waypoint control features $\xi_{CW}(t; R_C)$ and total control features $\xi_C(t; R_C)$ are shown in the figure. The scalar reduction $C_T$ of Fig. 5.4 is shown in Fig. 5.5. A whole circuit around the arena is represented in Fig. 5.5, as the timescale is longer than that
Figure 5.3: Two representations of the Waypoints trial path. The top figure shows five distinct circuits marked at $\approx 1$s intervals. The bottom figure shows all ten circuits.
in Fig. 5.4. Four sharp left turns are clearly represented by periods of very negative $C_T(t)$. The unusual behavior at $t = 1100$ of a small left turn and rightward correction immediately after was the result of the obstacle avoidance mechanism, described in the next section, that detected an approaching feature on the wall beyond the robot’s arena. This anomaly commonly occurred on the upper left waypoint of Fig. 5.3 due to the visual appearance of an appliance next to the arena, and the figure shows the messy left turns that occasionally resulted.

5.3 Tactical Navigation

Tactical navigation describes the short term navigational behavior used to handle routine hazard management tasks. In general, robotic applications are not directly concerned with this aspect of navigation as long as the robot is able to reach strategic goals safely and efficiently. Tactical navigation involves the details of how this safety and efficiency are achieved in the presence of environmental hazards and obstacles. This dissertation did not identify and take advantage of potentially navigable hazards like stairways, vegetation, or aquatic features that may or may not be navigable by a particular robotic platform. This section is instead concerned with another large part of tactical navigation, the obstacle avoidance task. The depth detection system described in Chapter 4 discovered obstacles in the robot’s field of view. The tactical navigation part of the control algorithm steered the robot away from those detected obstacles with minor course perturbations while on route to a strategic goal. In the experiment described here, no obstacles were navigable, so a collision with either obstacle represented a catastrophic failure, and there was always an obstacle free path available to the strategic goal, although not a direct path.
Figure 5.4: The control activations $\xi_c(t; R_C)$ for the Waypoints trial. A sharp left turn occurred between $842 \leq t \leq 870$. The rest of the time, the robot headed straight forward.
5.3.1 Avoidance Behavior

Obstacle location information was contained in the collision features \( \xi_D(t: \vec{x}_T, D) \), described in Chapter 4. These features were linearly integrated into the avoidance control features

\[
\xi_{CA}(t: R_C) = \sum_{\vec{x}_T, D} W_{CA}(R_C, \vec{x}_T, D) \xi_D(t: \vec{x}_T, D). \tag{5.3}
\]

The obstacle avoidance weights \( W_{CA}(R_C, \vec{x}_T, D) \) determined the tactical navigation behavior of the robot. In real animals, cognitive skills allow for much more sophisticated behavior than the model of Eq. (5.3). As that level of reasoning is well beyond the state of the art in engineering, a relatively simple set of heuristics was used here that resulted in rudimentary obstacle avoidance. Most robot designs use some set of heuristics, and...
the use of them is always associated with a restriction of the environments in which
the robot can reliably operate. The same is true here, although the heuristics used were
more general than most others found in the literature (see Section 1.1). The heuristics
were used to derive the obstacle avoidance weights $W_A(R_C, \bar{x}_T, D)$ in four separable
terms:

$$W_A(R_C, \bar{x}_T, D) = W_{A_Y}(y_T)W_{A_X}(R_C, x_T)W_{A_D}(D)W_{AFOE}(\bar{x}_T). \quad (5.4)$$

The terms account for obstacle height, horizontal location, time-to-collision, and relationship to the focus of expansion, respectively. The purpose of the depth term $W_{A_D}(D)$ is particularly straightforward: it emphasized image regions with small associated times-
to-collision $D$:

$$W_{A_D}(D) = 1 - \frac{D - \min(D)}{\max(D) - \min(D)} \quad (5.5)$$

where $\min(D)$ and $\max(D)$ were respectively the smallest and largest time-to-collision
tunings in the feature space of $\xi_D(t; \bar{x}_T, D)$, respectively. More sophisticated models
than this linear one are certainly possible. For instance, the system could be designed
to detect obstacles so far away that they are known to be harmless in the immediate
future. The prototype system used here did not demonstrate the capability to detect
obstacles until they were close enough to require avoidance. Consequently, the linear
model of Eq. (5.5) was adequate.

The horizontal term $W_{A_X}(R_C, x_T)$ determined the direction $R_C$ the robot took to
avoid an obstacle detected in the visual field at $x_T$. The heuristic used here is that
hazards on the extreme sides of the scene (large $|x_T|$) required only small course ad-
justments if any at all, and hazards closer to the center ($x_T$ near zero) required more
urgent attention in the form of higher turning rates. If the obstacle was on the left side
of the view, then the robot turned right. Likewise, it turned left if the obstacle was
on the right side. The confusing situation of the obstacle being discovered exactly in
the middle was not addressed, although it would be easy to pick an arbitrary direction
in this case. The control vector $\xi_{\text{CA}}(t; R_c)$ coded sharper turns as $|R_c|$ became larger
according to the weights

$$W_{\text{AX}}(R_c; x_T) = \begin{cases} 
\frac{(\frac{R_c}{\sigma_{\text{AX}}} x_T - 1)^2}{2\sigma_{\text{AX}}^2} & : R_c \leq 0 \\
\frac{(\frac{R_c}{\sigma_{\text{AX}}} x_T - 1)^2}{2\sigma_{\text{AX}}^2} & : R_c > 0
\end{cases}$$

(5.6)

where $\sigma_{\text{AX}} = 8$ pixels was the receptive field width in $x_T$; $R_c$ was the width in pixels
of the input image, as described in Chapter 2; and $\alpha_{C1}$ and $\alpha_{C2}$ were arbitrary tuning
constants used to scale $R_c$ into compatibility with the robot chassis controller hard­
ware. Examples of control feature receptive fields tuned to $R_c \in \{.7, .3, .5\}$ in the
$x_T$ parameter space are shown in Fig. 5.6. These curves are simply plots of Eq. (5.6)
with $\alpha_{C1} = \alpha_{C2} = 1$. Also, obstacles close to the focus of expansion activated stronger
turning commands than obstacles associated with large $x_T$. Choosing $\alpha_{C1}$ and $\alpha_{C2}$ was
arbitrary and problematic due to the poor quality of the controller hardware. In reality,
good choices of $\alpha_{C1}$ and $\alpha_{C2}$ turned out to be dependent on the battery state of the
chassis, and possibly some other unknown factors as well. This sensitivity made the
control scheme generally fragile. For constant values of $\alpha_{C1}, \alpha_{C2}$, and $C_T$, fresh bat­
teries often spun the robot out of control, while batteries near discharge did not turn
the robot at all. There was therefore a sweet spot of battery charge in which successful
experimental trials could be conducted. To solve these problems, it might be possible
to adapt $\alpha_{C1}$ and $\alpha_{C2}$ on the fly based on feedback from the rotation features $\xi_R(t; R)$,
although an easier and more robust solution would be to simply use a better robot
platform.
Figure 5.6: Each curve $W_{AX}(R_C, x_T)$ plots the “receptive field” of a $R_C$ tuned control feature $\xi_{G\cdot A}(t; R_C)$ in the parameter space $x_T$. For instance, the $R_C = .7$ feature activates to encourage a hard right turn when close obstacles are detected around $x_T = -15$. 
The vertical component $W_{gy}(y_T)$ biased potential hazards based on height. On a ground-based robot, obstacles low in the field of view were always avoided as they were presumably resting on or secured to the ground. On the other hand, obstacles higher than the robot platform such as tabletops or ceiling features might be problematic only if they were part of a larger hazard that extended to ground level. Otherwise, the robot would in theory be able to pass safely beneath them (this situation was not tested). Thus, obstacles were biased with less weight assigned to high hazards. The simplest model to achieve this behavior is linear with a threshold $y_0$:

$$W_{gy}(y_T) = \begin{cases} 1 & y_T \leq y_0 \\ 1 - \frac{y_T - y_0}{R_y} & y_T > y_0 \end{cases}$$ (5.7)

where $R_y$ was the height in pixels of the input image. In reality, unusual obstacles such as tables were not studied in this dissertation, so the $y_T$ bias was disabled by using $y_0 > R_y/2$.

The final term of the Eq. (5.4) deemphasized the image region around the focus of expansion:

$$W_{AFOE}(\vec{x}_T) = \begin{cases} 0 & |x_T| < 10, |y_T| < 20 \\ 1 & \text{elsewhere} \end{cases}$$ (5.8)

Motion parallax was so small in this region, disappearing entirely at $\vec{x}_T = (0, 0)$, that the depth estimates $\xi_D(t; \vec{x}_T, D)$ were usually unreliable. Thus, the constant 0.1 values were sufficient for this application.

Figure 5.7 shows the control weights for a subset of $\xi_{CA}(t; R_C)$. There was only partial coverage across the visual field in order to reduce computation requirements and achieve real-time performance.
Figure 5.7: Avoidance control weights $W_{CA}(R_c, x_T, D = 75)$. The dots show the receptive field centers $x_T$, and the shading intensity represents connection strength to the control features identified in the captions.
5.3.2 Behavior Integration

When an obstacle was detected, the robot turned away from the obstacle by activating control features $\xi_{CA}(t; R_C)$ corresponding to an appropriate $R_C$ tuning. When there were no detected obstacles, the activations $\xi_{CA}(t; R_C)$ were small relative to the strategic controller activations $\xi_{CW}(t; R_C)$. The waypoint and avoidance behaviors $\xi_{CW}(t; R_C)$ and $\xi_{CA}(t; R_C)$ were integrated to form a single control vector $\xi_C(t; R_C)$. Similar to the "subsumption architecture" [21], the robot by default headed towards the next waypoint. When an obstacle was detected, the obstacle avoidance behavior activated at the expense of the waypoint behavior, until the obstacle was safely avoided and the waypoint behavior resumed control. The control vector $\xi_C(t; R_C)$ was normalized using Nowlan and Sejnowski's softmax function [74] with parameter $\alpha_c$ and a new mixing constant $\lambda_c$:

$$\xi_C(t; R_C) = \frac{e^{\alpha_c \cdot \xi_{CW}(t; R_C) + \lambda_c \cdot \xi_{CA}(t; R_C)}}{\sum_{R_C} e^{\alpha_c \cdot \xi_{CW}(t; R_C) + \lambda_c \cdot \xi_{CA}(t; R_C)}}.$$  \hspace{1cm} (5.9)

The expansive coefficient $\alpha_c$ resulted in mutual inhibition between the control features for narrower tuning curves in $R_C$. The mixing coefficient $\lambda_c$ adjusted the sensitivity of the control algorithm to obstacles. A good setting for $\lambda_c$ resulted in a course correction with enough time to get around an obstacle, but not so early as to wastefully avoid non-dangerous hazards. Here, $\lambda_c$ was chosen with an ad-hoc experimental method. The coefficient needed to be large enough for the avoidance controller to dominate Eq. (5.9) when obstacles were actually present, but not so large that noise and small errors would disrupt the strategic controller at other times. The mixing coefficient also normalized the tactical controller activations to account for the sizes of its receptive fields, which varied with the density of depth estimates $\xi_D(t; F_T, D)$. The explicit normalization in Eq. (5.9) assures that there was always at least one active control feature.
Figure 5.8 shows a series of overhead images of the robot making one complete circuit around the arena with two obstacles for the Obstacles trial. Figure 5.9 shows a similar overhead image with the paths of eighteen circuits around the arena traced. The obstacle at the bottom of the figure required a left course deviation for avoidance, while the obstacle on the top required a rightward correction. The right turn on top required a stronger correction than the left turn due to the placement of the obstacles relative to the waypoints.

Body clipping was a significant problem in the Obstacles trial. It was a routine situation for the robot to make an avoidance correction that turned the robot enough for the obstacle to leave the field of view. When that happened, the depth detectors deactivated and the strategic navigation behavior resumed control, often re-orienting the robot back towards the obstacle before the it had enough time to progress forward at the perturbed course to actually pass the obstacle. Humans and animals have at least two advantages over this engineered system in this regard. First, they can continue to look at the obstacle while traveling in a different direction because they have necks and eyesockets, and brains capable of handling the resulting complexity of information processing. Second, they have a short term memory and an abstract sense of where the hazard is that they have seen before, even if they cannot see it at any particular instant. Neither capability is possible to emulate in an artificial system given the current state of the art in engineering. Instead, a very primitive memory was provided to the robot by low pass filtering both the velocity features $\xi_1(t; \Omega_1)$ and avoidance control features $\xi_{C_A}(t; R_C)$. Both filters were simple first order IIR configurations. Low pass filtering $\xi_1(t; \Omega_1)$ caused the depth detection system to "remember" obstacles after they had left the field of view. They were remembered at the retinotopic location last seen. Filtering the avoidance control features $\xi_{C_A}(t; R_C)$ smoothed the transitions between $\xi_{C_A}(t; R_C)$
Figure 5.8: Series of successive overhead snapshots of the experimental arena during one complete circuit of the Obstacles trial.
Figure 5.9: Two representations of the Obstacles trial path. The top figure shows five distinct circuits marked at \( \approx 1s \) intervals. The bottom figure shows all eighteen circuits.
dominance and \( \xi_{CW}(t; R_C) \). This gave the robot a little bit more time to move past obstacles before being completely reoriented by \( \xi_{CW}(t; R_C) \).

Figure 5.10 shows all three control feature activations \( \xi_{CA}(t; R_C), \xi_{CW}(t; R_C) \) and \( \xi_{C}(t; R_C) \) during the avoidance of the bottom obstacle on the first circuit of the Obstacles trial. The reduced version \( C_T \) is shown in Fig. 5.11 over a longer time interval. Early in the the period of Fig. 5.10, the robot was making a hard left turn after achieving the lower left waypoint. The control features \( \xi_{CW}(t; R_C = -0.8) \) and \( \xi_{CW}(t; R_C = -0.75) \) were active during this time, and dominated the behavior of the overall control features \( \xi_{C}(t; R_C) \). Figure 5.10 shows the two features to be qualitatively similar for \( 50 < t < 60 \). They are not identical due to the normalization of Eq. (5.9) and the fact that single subfeature sets \( \xi_{CW}(t; R_C) \) and \( \xi_{CA}(t; R_C) \) usually had multiple tunings active simultaneously. Between \( 65 < t < 85 \), the robot was heading towards the lower right waypoint, still guided by \( \xi_{CW}(t; R_C) \). As in the Waypoints trial, low loop gain prevented a perfect alignment with the waypoint, and that is why \( \xi_{CW}(t; R_C = 0) \) never activated. The obstacle was detected starting at \( t = 85 \), and four tunings of \( \xi_{CA}(t; R_C) \) were quickly activated as a result. The normalization of Eq. (5.9) found the most active feature, \( \xi_{CA}(t; R_C = -0.45) \), and correspondingly activated \( \xi_{C}(t; R_C = -0.45) \) to the exclusion of all others tunings. The right turn tuned strategic control features \( \xi_{CW}(t; R_C \geq 0.3) \) were trying to reorient the robot towards the waypoint between \( 100 \leq t \leq 120 \) with increasing urgency, but the tactical navigation behavior \( \xi_{CA}(t; R_C) \) completely suppressed those commands during that interval. Finally, at \( t = 132 \), the obstacle disappeared and the tactical controller deactivated and the strategic course was resumed. Comparing Fig. 5.11 with the Waypoints case of Fig. 5.5, the obstacle avoidance commands around \( t = 100 \) (left) and \( t = 350 \) (right) are immediately followed by strong, reverse signed, corrective action by the strategic controller. In the Waypoints
trial, no rightward commands were issued at all. However, a right turn was necessary after the lower obstacle had been avoided in the Waypoints case.

5.4 Chapter Summary

In this chapter it was shown how population coded time-to-collision features \( \xi_D(t; \vec{x}_T, D) \) were used as sensory input into a population coded control model \( \xi_C(t; R_C) \). The only other known robot controller to date that has used population codes was described by Blanchard, Rind, and Verschure. [5] As reviewed in Section 1.1, their work did not otherwise share many similarities with the one implemented here. There was some previous work that used time-to-collision for obstacle avoidance, but without population coding. [20, 55, 7] The avoidance controller component \( W_{AX}(R_I, \vec{x}_T) \) shared similarities with the model of Coombs and Roberts [9], but their system was much less complex than this one, and did not employ population coding either. The method of using multiple independent controllers of different priorities used here is similar to Brooks’ subsumption architecture [21], but Brooks’ robots had significantly different implementations than this one, and did not use vision or population codes.

The avoidance control model proposed and demonstrated here is novel in its ability to handle more complex environments than previous work using only vision derived input. Some of the pieces of this control model have been studied in different contexts as listed above, but new ideas have been added in this dissertation and the overall behavior of the model is more sophisticated than a simple collection of cited parts. The strategic controller was simple, but adequately simulated application specific strategic goals. The integration of the two models successfully achieved both goals. The experimental results are novel in the size and speed of the simulation and complexity of the real world arena.
Figure 5.10: Control activation $\xi_C(t; R_C)$ from Obstacles trial.
Figure 5.11: Raw turn command $C_T(t)$ for the first circuit of Fig. 5.9

The obstacles were successfully avoided using vision alone with a significant level of autonomy demonstrated by the robot.
CHAPTER 6
CONCLUSIONS

This dissertation showed that contemporary neuroscientific models related to mammalian early vision can be implemented in real time, using established engineering methods. The models used were based primarily on the motion energy model of Adelson and Bergen, and the motion energy integration technique of Grzywacz and Yuille. Important features of these models retained throughout the implementation included the use of population coded representations of information. Certain modifications, including normalization and mutual inhibition, were used to solve real world problems that were never exposed previously in the original theoretical work. The prototype implementation demonstrated adequate computational throughput for real time analysis of video imagery to estimate a mobile robot's time to collision to environmental obstacles. For portions of the task that remain beyond the current neuroscientific understanding, like how velocity features are integrated into higher level functions such as depth detection in real mammals, techniques compatible with the known neuroscience were engineered. No claim was made that the new models are neuroscientifically accurate, but it was shown that they worked well in concert with established neuroscientific models to solve a hard engineering problem.

Furthermore, it was shown that real time control of an autonomous robot was possible based on these models and using the prototype implementation. Relative to previous
robot models and implementations found in the literature, the amount of environmental tuning and the use of heuristics were minimized. The claim is made, although not demonstrated, that these techniques are more general than previous results, enabling the robot to safely handle a wider variety of environmental situations.

6.1 Common Navigational Problems Revisited

In Section 1.1.3 several problems common to previously proposed visually navigating robots were introduced. Here, the issues are revisited in the context of this dissertation’s results.

Diffuse Regions

Diffuse regions were commonly encountered in the robotics experiments described here. In the trials using the Circles obstacle, diffuse regions were caused by blank features on the obstacle itself. In other trials, the visual stimuli occasionally confused the obstacle detection system, resulting in poor, missing, or erroneous collision feature estimates. The result was the same as with real diffuse regions: no individual feature estimate of any type could be trusted by itself. In many cases, diffuse textures on distant objects were an advantage because they were never confused with close, dangerous objects. In real applications, however, this situation would represent a lucky coincidence, as opposed to behavior that was robust by design.

No attempt was made to guess or interpolate collision or earlier feature estimates in the diffuse regions. However, the controller integrated large receptive fields of collision features, averaging them. Neighboring collision features in nondiffuse regions were often more accurate than the optic flow derived versions, enhancing the performance of the large field integration. The averaging was similar to previous work that integrated wide
field optic flow to do corridor following modeled after honeybee behavior. [20] Unlike the honeybee inspired work, however, this dissertation computed the time to collision of obstacles as an explicit data product. The resulting depth map is a rich piece of information that enables more sophisticated behavior than the simple flow balancing heuristic used in the honeybee inspired robots. The depth map was used here to do simple obstacle avoidance, already an arguably harder task than corridor following, but the general purpose utility of the feature encourages a much wider set of future applications.

The temporal filtering associated with the motion energy, velocity, and control features provided additional relief from diffuse regions. When a visual region became diffuse after an obstacle passed through it, the temporal filters provided a mechanism that “remembered” the collision features for a short time afterward. This technique shared similarities with Santos-Victor, et al.’s “sustaining” technique. [12] The filtering action could be considered a general purpose heuristic: in general, real obstacles do not instantly appear and disappear unless they are moving very fast relative to a robot’s visual bandwidth. They usually have to enter the scene and gradually progress towards the robot, resulting in smooth collision estimates. Temporal filtering forced this smoothness, improving the system’s performance in the presence of diffuse regions.

Using these techniques, the robot in this dissertation successfully avoided obstacles in the presence of diffuse features. Special heuristics such as those used by Dunchon [10] were not required. This result also distinguished itself from previous work that described the problem, but did not attempt to solve it. [20]
Aperture Problem

The aperture problem was also commonly encountered in these robotics experiments. All the stimuli used were susceptible to the aperture problem, even the Circles pattern which was designed to be immune. Population coding provided a novel mechanism to delay the resolution of these aperture effects, eventually resulting in better accuracy than with optic flow.

All velocity features compatible with a particular motion pattern were activated simultaneously, because the true velocity was never known due to the aperture problem. This was only possible with a population code: optic flow formatted velocity could only represent a single vector at a time. The simultaneous propagation of multiple velocity features to downstream computations enabled the use of additional information available only to those specialized computations, to decide which of the activated features were likely to be accurate, and which were errors generated by the aperture. The downstream computations used here were those of Rotation and Translation. Special considerations, private to each module, were used to decide which velocity features to use in the computation of Rotation and Translation features. The two feature types in fact integrated different components of the velocity population code, resulting in enhanced accuracy in both modules. The only previous work to describe a distinction for aperture effects in downstream computations was based on optic flow. [12] Their distinction was not based on feature detection, but the behaviors of centering, obstacle avoidance, and docking.

The delayed resolution also made available the possibility to add new velocity based computations in the future without impacting the existing computations. This was not possible when using vector coded optic flow, as the velocity had to be completely resolved in the optic flow computation. Not only did this have the effect of generating
errors due to the lack of information relative to the population coded case, but all downstream computations would have been required to operate on identical flow estimates. A designer could not therefore change the optic flow implementation to accommodate new features without considering the negative impacts on existing ones. This would be a problem for some proposals [7] that, for instance, simply discarded flow vectors that were obviously in error.

**Bumpy Ride**

The robotic chassis used in these experiments experienced a bumpy ride, even on the smooth floor that was used. A second order, low pass temporal filter applied to the stimulus image was sufficient to eliminate the associated high frequency jitter. However, the filter introduced new undesired “ghosting” effects with large motions, rotation in particular.\(^1\) The ghosting occurred on large turns driven by the strategic controller. As the strategic behavior did not depend on vision, the ghost imaging did not occur at the critical times when it would have effected the vision based tactical controller. The bumpy ride phenomenon is not discussed much in the literature: it was only mentioned by only one other researcher who compensated with a temporal filtering process similar to this one. [3] This previous robot never turned during image acquisition, so presumably “ghosting” did not occur.

**Motion Bandwidth**

As in most robotics experiments, a reliable and easy way to reduce problems with motion patterns occurring outside the bandwidth of the system was to slow the robot.

---

\(^1\)One of the cameras used to capture video for this work had a “Super Steady Shot” feature. Interestingly, similar “ghosting” effects could be observed by sweeping the camera quickly. Most likely, the engineers that designed the camera used the same low pass filter technique to compensate for a user’s shaky hands.
especially in turns. This also reduced the ghost artifacts resulting from the front end temporal filter, introduced to counter the bumpy ride phenomenon. The high computational throughput of the prototype implementation enabled a high frame rate, effectively increasing the motion bandwidth along with the temporal sampling rate. New implementations of the same techniques built using fast, special purpose hardware would increase the frame rate, and associated motion bandwidth, even further.

The wide variety of spatial Gabor impulse responses associated with the motion energy computation also improved the motion bandwidth. Similar to a previous motion estimation technique using “pyramids” the variety of spatial scales enabled the detection of visual features moving a small amount or a large amount during a frame. Combined with extended state maintained by the temporal motion energy filters relative to most optic flow implementations, the system estimated a wide range of motion patterns, both slow and fast.

Rotational Flow

A novel method for rotational flow compensation enabled by the population coded representation of velocity was used. It did not require extra-visual input, although such information could be integrated in the future to further enhance performance. The technique allowed the robot to continue moving forward in the presence of obstacles, simultaneously estimating their time to collision while turning. In contrast, previous work required stopping forward translation or obstacle detection during turns. The technique used here also eliminated the need for mechanical gaze stabilization and saccades, or multiple cameras, enabling the use of a single, fixed position, imaging device. The technique shared some features with that of Perrone, although he was more concerned with detecting the focus of expansion: this dissertation assumed
it was a known constant. Perrone never implemented his work on a robot to test his model. Neven also subtracted a global rotation estimate, but the technique was more simple than the one used here, and was based on optic flow. [7].

**Symmetric Flow Anomaly**

Previous work reported the symmetric flow anomaly occurring with honeybee inspired, flow balancing robots being confused by corners and walls. [9, 10] Corners were not encountered in this dissertation, but the symmetric flow anomaly did occur when the robot headed towards an obstacle centered in the field of view. In this case, the tactical controller generated an unsuitable multimodal control output that signaled the robot to turn right and left at the same time. Although not demonstrated, it was argued that this situation is easy to detect, and the detection could be used to add a rightward or leftward bias to the tactical controller. This dissertation was not the first publication to make this unproven claim. [9]

**Parallax Blind Spot**

The parallax blind spot, instead of being a problem, actually improved the rotation estimates used to compensate for rotational flow. Because there was no expansive flow near the FOE by definition, it offered a reliable region to visually estimate rotation, whenever the FOE was free of the diffuse regions problem.

Like previous work [20, 7], however, the collision estimates near the parallax blind spot were unreliable and not considered by the controller. In general, the system was robust and large enough to successfully operate without the inclusion of the FOE region by the controller, and without the need for multiple cameras [20] or tortuous, complex paths. [3, 19, 17]
Sensory and Behavior Integration

Obstacle time to collision information drove the tactical obstacle avoidance behavior, which was integrated with the waypoint based strategic navigation behavior. Previous work has integrated similar behaviors both with [7] and without [21] vision, with widely differing implementations. The technique used here did not preclude the addition of even more navigational behaviors integrated in a similar way.

Body Clipping

The original body clipping phenomenon described in Chapter 1 and reported (but not addressed) in previous work [20, 22] was solved primarily by simply mounting the imager at the rear of the robot, so the whole chassis was in view of the imager. In this configuration, it was harder, although not impossible, for obstacles to “sneak” up to the chassis while remaining out of view. Previous literature did not label this issue as “body clipping” or anything else. Previous authors also did not make the distinction between body clipping and diffuse regions as done here.

A different type of body clipping was found to be a problem in these experiments. The robot occasionally turned to avoid an obstacle and lost track of the it as the obstacle left the field of view. In this case, the system could not “remember” where the obstacle was, so it prematurely reverted to waypoint finding behavior. The obstacle could then suddenly reappear as the robot re-turned towards the waypoint, as the chassis had not moved forward far enough for avoidance during the tactical avoidance move. In this situation, it was usually too late to perform a second tactical move and a collision with the obstacle occurred. This problem was addressed by giving the robot some primitive “memory” of the obstacle by low pass filtering the control signals with a simple first order, IIR filter. Slower and smoother control signals encouraged more forward motion.
and less rapid turning, and reduced the problem with this variant of body clipping. A related approach, using a “sustaining” behavior, was previously proposed by Neven [7], in a significantly different implementation than used here.

A more sophisticated technique to handle both variants of body clipping could improve the robustness of this and other robot implementations. The filters used here required experimental tuning, which undoubtedly had negative implications for different, untested environments. Previous authors suggested, but did not implement, the use of an environmental maps. [20, 22] The mapping problem has been addressed by the AI community for quite some time without tremendous success for general purpose machines. The body clipping problem is hard to solve in a robust manner, and a biologically inspired solution may require much improved understanding of cognition and general brain function.

**Computational Latency**

The software architecture of the prototype implementation produced a ten frame latency due to its pipelined design, required for high throughput. This latency had minimal effect on the collision feature estimates because the latencies associated with the motion energy stage were much more significant. The only other researchers to address this problem computed layers asynchronously so the faster computations could proceed at their naturally higher rates, but this technique raised new issues. [20]

The latencies introduced by the motion energy temporal filters were significant, and represented the biggest problem with the overall strategy studied in this dissertation. In order to reduce the latencies to a point where the robot had enough time left to avoid detected obstacles, the temporal bandwidths of the motion energy filters were increased. The wide temporal bandwidths severely degraded the discriminatory power
of the motion energy features. The result was too many of the motion energy features simultaneously activating, resulting in wide velocity, translation, and rotation tunings. The tunings were narrowed with lateral inhibition in the velocity computations, which partially overcame the problem.

The motion energy latency is fundamental to the computation of motion energy due to its temporal filtering requirements. An important characteristic of this is problem is that it would occur in any implementation: unlike the software architecture induced latency, it is not related to the discrete time system's sample rate. Even analog filters have latency associated with them; it is a function of their frequency responses. A potential way to alleviate this problem, not yet demonstrated or proven, might be to increase the spatial resolution of the stimulus image. All else being equal, visual features in the scene would then move faster across pixels, even though they would be moving at the same physical rate in degrees per second across the retina or CCD's imaging area. The increased temporal speeds would be good for several things. First, the temporal filters could be tuned to higher frequencies, directly decreasing latency. More importantly, both spatial and temporal bandwidths could be reduced, improving the discriminatory capability of the motion energy detectors. Increased frequencies could take advantage of higher speed digital implementations. Without increasing the image resolution, there is currently not much point in increasing the temporal sample rate due to the motion energy filter latencies. Finally, analog implementations would be simplified because high frequency filters require smaller integrated components like capacitors and resistors.

A final observation is that the computation latency problem may be related to a current problem in neuroscience. Neuroscientists find cells in early vision systems that have broad spatiotemporal tunings in V1, and broad velocity tunings in MT, similar
to the engineered results shown here. However, real humans and animals demonstrate accuracy detecting visual features far beyond what the broad tuning curves suggest. The biological mechanism that computes narrowly tuned features from broadly tuned motion energy and velocity inputs has not yet been found. The similarities with this dissertation suggest that evolution may also have had a problem with achieving narrow temporal bandwidths. If this was true, then natural systems may have already solved the problems described here: we just have to figure out what the solutions are in the neuroscience laboratory.

6.2 Future Work

The first and easiest problem to fix is the robot chassis. There are several commercial products available, popular for use in experimental robotics that undoubtedly work better than the toy used here. The biggest problem with the tank chassis was the coarse, unpredictable control capabilities of the motor control circuit. This problem led to unrepeatable, fragile experiments and it complicated the control equations with the arbitrary factors $\alpha_{C1}$ and $\alpha_{C2}$. With a constant speed control setting, the robot would often move with widely changing ground speeds. The effect of turning controls were even more problematic, because they actually changed during the trials as obstacles were encountered. In theory, a robust control algorithm could adapt on the fly to counter these problems; in fact, that is why the rotation features $\xi_R(t; R)$ were required. If the algorithm could have trusted the turn signal $C_T$ to be accurate, then it could have been used instead of estimating $\xi_R(t; R)$. It would be convenient, however, to start with a more reliable chassis so more pressing issues could be addressed first as the overall strategy matured. A better technique would be to spoil the control characteristics of
a robust chassis as part of quantitative testing instead of dealing with wildly unknown factors as was done here.

A better understanding is needed of the transient nature of motion energy filters and their integration into velocity features. The theory in the original neuroscience sources, and the related empirical data gathered from laboratory experiments, was concerned primarily with the steady state responses of neurons in the cat's visual cortex. It is routine procedure in neuroscience to truncate the initial portion of neuron activations when reporting and analyzing data. It turns out that in the robotic application, these transient effects that are often ignored by neuroscientists dominated the behavior of the motion energy responses at critical times. The problem manifested itself when motion energy filters tuned to high temporal frequencies activated quickly relative to the slower ones as they responded to high frequency transients. These broadband transients occurred, for instance, when a stimulus pattern first appeared in a receptive field. Regardless of the feature's motion characteristics, the sudden appearance of the feature created high frequency temporal transients that the motion energy detectors, as implemented here, could not distinguish from fast motion. The result was underestimated collision features \( \xi_D(t; \tilde{x}_T, D) \), and the related overreaction by the tactical controller \( \xi_C_A(t; R_C) \). This issue is related to the discussion above in the context of computational latency, and may have a similar solution.

An unaddressed issue was the dynamic range requirements of the vision system. These were related to normalization factors and thresholds used in the current implementation. The results reported here all used maximum contrast stimulus: usually black paper pasted on pure white Plexiglas sheets. The contrast of natural obstacles will never be constant like those. In the existing implementation, certain thresholds were used to determine when an activation was noise, or accurately reflected the state
of nature. These thresholds, and the activation levels of the motion energy model, were proportional to the stimulus contrast (but not the contrast polarity). Allowing the stimulus contrast to vary will introduce new complications that must be addressed for truly robust robot behavior. In addition, the dynamic range requirements of the system will strongly impact new implementations. The prototype used 32 bit floats, a luxury probably not available in special purpose circuits, especially analog technologies. Techniques are therefore needed to constrain the dynamic range requirements, and identify what those constraints are.

The feature space tunings $\Omega_M, \Omega_V$, and $\Omega_T$ were set in this dissertation primarily based on computational feasibility and hand tuning. The system needed to operate in real time, so the network size was trimmed to get the frame rate above an arbitrary ten frame per second threshold, and then empirically tuned to navigate the obstacle course given the computation constraints. A quantitative study would show how changing the quantity and specific values of the parameter tunings affects specific performance metrics, which also need to be defined.

The notion of “motion bandwidth” needs to be developed further. It is related to the Nyquist sampling theorem, except the theoretical limit would be velocity speed instead of a sampling rate. The temporal sampling rate would be a factor, as would the Gabor spatial frequency tunings $\psi_s$. A satisfactory result would impose constraints on the motion energy tunings $\Omega_M$ for some maximum image speed. The maximum image speed would also constrain the allowable speeds and rotation rates of the robot itself to guarantee not exceeding the motion bandwidth. In the current implementation, the motion bandwidth was qualitatively controlled by limiting the speed of the robot and choosing a large set of $\Omega_M$. A theoretical result would enable much better selectivity in picking $\Omega_M$, with implications for increased robustness by quantitatively managing
the motion bandwidth much like a signal processing engineer controls the sample rate of a digital system.

A biological vision simulation of this size controlling, in real time, an autonomous robot operating in the real world is a novel and exciting result. Now that basic closed loop functionality has been achieved, it is easier to make incremental improvements to particular parts of the system and test the effects of the changes. Before a successful closed loop experiment was performed, it was not always clear if the basic ideas would work at all. Now that the concept has been proven, there is a baseline functionality to compare new results against, which should accelerate the pace at which new research could be done.
Chapter 4 compared a population coded velocity representation with a traditional optic flow approach. The optic flow data was computed with the Lukas-Kanade algorithm described in this section. The development and implementation used here closely follows the implementation described by Barron. et al. [76] and Fleet and Langley [71].

A.1 Theory of Differential Optical Flow

Many techniques have been proposed to estimate the 2D motion field associated with points in a 3D environment projected onto a 2D spatiotemporal data stream like video. These optic flow algorithms are broadly classified into categories including differential, frequency based, and correlation based techniques [28]. The population coded technique described in the main body of this dissertation is frequency based. Correlation based techniques have their own set of issues and are not discussed here. The most popular optic flow techniques by far are the differential techniques, including algorithms originally proposed by Horn and Schunk [83] and Lukas and Kanade [84]. These algorithms are based on the conservation of image intensity in local spatial regions and short time intervals described by the optical flow constraint equation

\[ \nabla S(\vec{x}, t) \cdot \vec{v} - \frac{\partial S(\vec{x}, t)}{\partial t} = 0 \]  

(A.1)
where $\nabla S(\vec{x}, t)$ is the spatial gradient of the spatiotemporal data stream $S(\vec{x}, t)$ and $\vec{r}$ is the optical flow estimate at $\vec{x}_1$. In the context of this dissertation, the velocity variable in Eq. (A.1) corresponds to the velocity tuning space parameters in $\Omega_\gamma : \vec{r} = \nu_\gamma \theta_\gamma$. The spatial location is $\vec{x}_1$ in both representations.

Unfortunately, $\vec{r}$ is a two dimensional vector and Eq. (A.1) only provides one constraint. Differential optic flow estimation algorithms are distinguished by how a second constraint is generated to solve for $\vec{r}$. A popular technique that performed well in Barron, et al. 's comparison, originally proposed by Lukas and Kanade, is to assume that $\vec{r}$ is constant in a local region $X$ around $\vec{x}_1$. This overconstrained problem is solved by minimizing

$$\sum_{\vec{x} \in X} W(\vec{x}) \left[ \nabla S(\vec{x}, t) \cdot \vec{r} - \frac{\partial S(\vec{x}, t)}{\partial t} \right]^2$$

(A.2)

that often includes a spatial bias $W(\vec{x})$ to emphasize constraints in $X$ nearer to $\vec{x}_1$. Following Fleet and Langley [71], the solution of Eq. (A.2) is

$$A^TW_\gamma \vec{r} = A^TW_\gamma \vec{b}$$

(A.3)

where, for $n$ points $\vec{x}_n \in X$ at instant $t$,

$$A = [\nabla S(\vec{x}_1, t), \ldots, \nabla S(\vec{x}_n, t)]$$

(A.4)

$$W = \text{diag}[W(\vec{x}_1), \ldots, W(\vec{x}_n)]$$

(A.5)

$$\vec{b} = -\left[ \frac{\partial S(\vec{x}_1, t)}{\partial t}, \ldots, \frac{\partial S(\vec{x}_n, t)}{\partial t} \right]^T.$$ 

(A.6)

Solving for $\vec{r}$,

$$\vec{r} = [A^TW_\gamma A]^{-1}A^TW_\gamma \vec{b}$$

(A.7)
\[ A^T W A = \begin{bmatrix} \sum_{\vec{x} \in X} W(\vec{r}) \left( \frac{\partial S(\vec{r}, t)}{\partial x} \right)^2 & \sum_{\vec{x} \in X} W(\vec{r}) \frac{\partial S(\vec{r}, t)}{\partial x} \frac{\partial S(\vec{r}, t)}{\partial y} \\ \sum_{\vec{x} \in X} W(\vec{r}) \frac{\partial S(\vec{r}, t)}{\partial x} \frac{\partial S(\vec{r}, t)}{\partial y} & \sum_{\vec{x} \in X} W(\vec{r}) \left( \frac{\partial S(\vec{r}, t)}{\partial y} \right)^2 \end{bmatrix} \]  

(A.8)

and

\[ A^T W \tilde{b} = \begin{bmatrix} \sum_{\vec{x} \in X} W(\vec{r}) \frac{\partial S(\vec{r}, t)}{\partial x} \frac{\partial S(\vec{r}, t)}{\partial y} \\ \sum_{\vec{x} \in X} W(\vec{r}) \frac{\partial S(\vec{r}, t)}{\partial x} \frac{\partial S(\vec{r}, t)}{\partial y} \end{bmatrix} \].  

(A.9)

### A.2 Implementation Details

In order to reduce the aliasing and quantization effects associated with limited motion bandwidth, the input \( S(x, y, t) \) was preprocessed with a low pass spatiotemporal filter. Spatially, the filter had a Gaussian impulse response

\[ h_f(x, y) = \frac{1}{2\pi \sigma_f^2} e^{-\frac{x^2 + y^2}{2 \sigma_f^2}} \]  

(A.10)

with width \( \sigma_f = 1 \) pixel implemented in the frequency domain with a 2D FFT based algorithm. The temporal filter is of the cascaded exponential type described in Section 2.3.2 with \( \alpha_t = .5, n = 3 \):

\[ a_3(t) = \begin{cases} \frac{5}{2} t^{-3} : & t \geq 0 \\ 0 : & t < 0 \end{cases} \]  

(A.11)

From Fleet and Langley \cite{19}, the temporal derivative of Eq. (A.11) for \( n = 1 \) is

\[ \frac{da_3(t)}{dt} = .5a_3(t) - a_3(t) \]  

(A.12)
which provided a convenient and fast way to compute \( \frac{\partial S(\vec{x}, t)}{\partial u} \) because \( a_3(t) \) is computed in the process of computing \( a_4(t) \).

The spatial gradients were computed as

\[
\begin{align*}
\frac{\partial S(\vec{x}, t)}{\partial x} &= \frac{I(x - 2, y, t) - 8I(x - 1, y, t) + 8I(x + 1, y, t) + I(x + 2, y, t)}{12} \\
\frac{\partial S(\vec{x}, t)}{\partial y} &= \frac{I(x, y - 2, t) - 8I(x, y - 1, t) + 8I(x, y + 1, t) + I(x, y + 2, t)}{12}
\end{align*}
\]

Once obtained, these quantities were used to compute Eq. (A.8) and Eq. (A.9) and, hence, Eq. (A.7).
APPENDIX B
PARAMETER TUNING TABLES

Parameters that varied as a function of retinotopic location were not shown in the graphics and tables in the main text, particularly in Chapter 4. This appendix tabulates those specific parameter choices for the temporal frequency tuning $\nu_\tau$, velocity speed tuning $\nu_\nu$, and translation speed $\nu_T$. The two trial sets of Chapter 4, Approach and Turn, and the closed loop experiments of Chapter 5 each used different tuning sets. The Approach sets were chosen to be as simple as possible, the Turn sets were chosen for high accuracy, and the closed loop sets were chosen for adequate accuracy with real-time performance. The temporal frequency tuning sets of $\nu_\tau$ are tabulated as the inverse $1/\nu_\tau$ for compactness, easier understanding, and because the software implementation represented the values this way.
### Table B.1: Temporal tuning set of $c_t^1$ used in the Approach trials of Chapter 4

<table>
<thead>
<tr>
<th>$t_m$</th>
<th>$f_m$</th>
<th>15</th>
<th>25</th>
<th>35</th>
<th>45</th>
<th>15</th>
<th>25</th>
<th>35</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>10.11*18</td>
<td>10.12*17</td>
<td>10.12*15</td>
<td>10.11*18</td>
<td>11.15*19</td>
<td>11.15*19</td>
<td>10.11*18</td>
<td>10.12*17</td>
<td>10.12*17</td>
</tr>
<tr>
<td>15</td>
<td>21.11*18</td>
<td>16.11*17</td>
<td>10.11*18</td>
<td>11.31*26</td>
<td>18.21*27</td>
<td>18.21*27</td>
<td>11.15*20</td>
<td>10.11*18</td>
<td>10.11*17</td>
</tr>
<tr>
<td>15</td>
<td>10.12*16</td>
<td>10.12*15</td>
<td>11.15*19</td>
<td>18.21*27</td>
<td>12.60*106</td>
<td>12.60*106</td>
<td>18.21*27</td>
<td>11.15*19</td>
<td>11.15*19</td>
</tr>
<tr>
<td>15</td>
<td>10.12*16</td>
<td>10.12*15</td>
<td>11.15*19</td>
<td>18.21*27</td>
<td>12.60*106</td>
<td>12.60*106</td>
<td>18.21*27</td>
<td>11.15*19</td>
<td>11.15*19</td>
</tr>
<tr>
<td>15</td>
<td>10.11*17</td>
<td>10.13*17</td>
<td>10.14*18</td>
<td>11.15*20</td>
<td>18.21*27</td>
<td>18.21*27</td>
<td>11.15*20</td>
<td>10.11*18</td>
<td>10.11*17</td>
</tr>
<tr>
<td>15</td>
<td>10.11*18</td>
<td>10.12*17</td>
<td>10.12*15</td>
<td>10.11*18</td>
<td>11.15*19</td>
<td>11.15*19</td>
<td>10.11*18</td>
<td>10.12*15</td>
<td>10.12*17</td>
</tr>
<tr>
<td>15</td>
<td>10.12*11</td>
<td>10.13*16</td>
<td>10.11*18</td>
<td>10.11*15</td>
<td>10.12*16</td>
<td>10.12*16</td>
<td>10.11*15</td>
<td>10.11*18</td>
<td>10.11*16</td>
</tr>
<tr>
<td>M</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
</tbody>
</table>

Table B.2: Temporal tuning set of \( \psi \) used in the Turn trials of Chapter 4.
Table B.3: Temporal tuning set of $c_l$ used in the closed loop trials of Chapter 5

<table>
<thead>
<tr>
<th>YM</th>
<th>25</th>
<th>35</th>
<th>-15</th>
<th>15</th>
<th>25</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>30.49</td>
<td>27.39</td>
<td>30.39</td>
<td>27.39</td>
<td>30.39</td>
<td>27.39</td>
</tr>
<tr>
<td>10.13.15.20</td>
<td>10.12.15.21</td>
<td>10.12.16.19</td>
<td>10.13.15.20</td>
<td>10.13.15.20</td>
<td>10.12.15.21</td>
<td>10.13.15.20</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>35</td>
<td>25</td>
<td>15</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>0.250,31</td>
<td>0.250,30</td>
<td>0.220,26</td>
<td>0.190,21</td>
<td>0.180,21</td>
<td>0.180,21</td>
</tr>
<tr>
<td>1.5</td>
<td>0.260,32</td>
<td>0.260,31</td>
<td>0.230,27</td>
<td>0.210,22</td>
<td>0.210,22</td>
<td>0.210,22</td>
</tr>
<tr>
<td>2</td>
<td>0.270,33</td>
<td>0.270,32</td>
<td>0.240,28</td>
<td>0.220,23</td>
<td>0.220,23</td>
<td>0.220,23</td>
</tr>
<tr>
<td>2.5</td>
<td>0.280,34</td>
<td>0.280,33</td>
<td>0.250,29</td>
<td>0.230,24</td>
<td>0.230,24</td>
<td>0.230,24</td>
</tr>
<tr>
<td>3</td>
<td>0.290,35</td>
<td>0.290,34</td>
<td>0.260,30</td>
<td>0.240,25</td>
<td>0.240,25</td>
<td>0.240,25</td>
</tr>
<tr>
<td>3.5</td>
<td>0.300,36</td>
<td>0.300,35</td>
<td>0.270,31</td>
<td>0.250,26</td>
<td>0.250,26</td>
<td>0.250,26</td>
</tr>
<tr>
<td>4</td>
<td>0.310,37</td>
<td>0.310,36</td>
<td>0.280,32</td>
<td>0.260,27</td>
<td>0.260,27</td>
<td>0.260,27</td>
</tr>
<tr>
<td>4.5</td>
<td>0.320,38</td>
<td>0.320,37</td>
<td>0.290,33</td>
<td>0.270,28</td>
<td>0.270,28</td>
<td>0.270,28</td>
</tr>
</tbody>
</table>

Table B.4: Velocity speed tuning set of $v$, used in the Approach trials of Chapter 4
<table>
<thead>
<tr>
<th>$\eta$</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
<th>1000000</th>
<th>10000000</th>
<th>100000000</th>
<th>1000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.100</td>
<td>0.183</td>
<td>0.190</td>
<td>0.190</td>
<td>0.183</td>
<td>0.190</td>
<td>0.183</td>
<td>0.190</td>
<td>0.183</td>
</tr>
<tr>
<td>0.200</td>
<td>0.367</td>
<td>0.367</td>
<td>0.367</td>
<td>0.367</td>
<td>0.367</td>
<td>0.367</td>
<td>0.367</td>
<td>0.367</td>
</tr>
<tr>
<td>0.300</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
<td>0.550</td>
</tr>
<tr>
<td>0.400</td>
<td>0.733</td>
<td>0.733</td>
<td>0.733</td>
<td>0.733</td>
<td>0.733</td>
<td>0.733</td>
<td>0.733</td>
<td>0.733</td>
</tr>
<tr>
<td>0.500</td>
<td>0.916</td>
<td>0.916</td>
<td>0.916</td>
<td>0.916</td>
<td>0.916</td>
<td>0.916</td>
<td>0.916</td>
<td>0.916</td>
</tr>
</tbody>
</table>

Table B.5: Velocity speed tuning set of $\eta$, used in the Turn trials of Chapter 4

continues on next page
Table B.6: Velocity speed tuning set of $v_s$ used in the closed loop trials of Chapter 5

<table>
<thead>
<tr>
<th>$v_s$</th>
<th>-35</th>
<th>-25</th>
<th>-15</th>
<th>5</th>
<th>15</th>
<th>25</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11018</td>
<td>0.11018</td>
<td>0.11018</td>
<td>0.11014</td>
<td>0.11011</td>
<td>0.11011</td>
<td>0.11018</td>
<td>0.11018</td>
</tr>
<tr>
<td>0.21026</td>
<td>0.21026</td>
<td>0.21026</td>
<td>0.21023</td>
<td>0.21020</td>
<td>0.21020</td>
<td>0.21026</td>
<td>0.21026</td>
</tr>
<tr>
<td>0.29033</td>
<td>0.29033</td>
<td>0.29033</td>
<td>0.29030</td>
<td>0.29027</td>
<td>0.29027</td>
<td>0.29033</td>
<td>0.29033</td>
</tr>
<tr>
<td>0.38041</td>
<td>0.38041</td>
<td>0.38041</td>
<td>0.38038</td>
<td>0.38035</td>
<td>0.38035</td>
<td>0.38041</td>
<td>0.38041</td>
</tr>
<tr>
<td>0.51057</td>
<td>0.51057</td>
<td>0.51057</td>
<td>0.51054</td>
<td>0.51051</td>
<td>0.51051</td>
<td>0.51057</td>
<td>0.51057</td>
</tr>
<tr>
<td>0.64066</td>
<td>0.64066</td>
<td>0.64066</td>
<td>0.64064</td>
<td>0.64061</td>
<td>0.64061</td>
<td>0.64066</td>
<td>0.64066</td>
</tr>
<tr>
<td>0.73079</td>
<td>0.73079</td>
<td>0.73079</td>
<td>0.73076</td>
<td>0.73073</td>
<td>0.73073</td>
<td>0.73079</td>
<td>0.73079</td>
</tr>
<tr>
<td>0.86093</td>
<td>0.86093</td>
<td>0.86093</td>
<td>0.86090</td>
<td>0.86087</td>
<td>0.86087</td>
<td>0.86093</td>
<td>0.86093</td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>86.0716</th>
<th>86.0718</th>
<th>86.0720</th>
<th>86.0722</th>
<th>86.0724</th>
<th>86.0726</th>
<th>86.0728</th>
<th>86.0730</th>
</tr>
</thead>
<tbody>
<tr>
<td>06.0716</td>
<td>06.0718</td>
<td>06.0720</td>
<td>06.0722</td>
<td>06.0724</td>
<td>06.0726</td>
<td>06.0728</td>
<td>06.0730</td>
</tr>
<tr>
<td>06.0712</td>
<td>06.0714</td>
<td>06.0716</td>
<td>06.0718</td>
<td>06.0720</td>
<td>06.0722</td>
<td>06.0724</td>
<td>06.0726</td>
</tr>
<tr>
<td>06.0708</td>
<td>06.0710</td>
<td>06.0712</td>
<td>06.0714</td>
<td>06.0716</td>
<td>06.0718</td>
<td>06.0720</td>
<td>06.0722</td>
</tr>
<tr>
<td>05.0712</td>
<td>05.0714</td>
<td>05.0716</td>
<td>05.0718</td>
<td>05.0720</td>
<td>05.0722</td>
<td>05.0724</td>
<td>05.0726</td>
</tr>
<tr>
<td>05.0708</td>
<td>05.0710</td>
<td>05.0712</td>
<td>05.0714</td>
<td>05.0716</td>
<td>05.0718</td>
<td>05.0720</td>
<td>05.0722</td>
</tr>
<tr>
<td>05.0712</td>
<td>05.0714</td>
<td>05.0716</td>
<td>05.0718</td>
<td>05.0720</td>
<td>05.0722</td>
<td>05.0724</td>
<td>05.0726</td>
</tr>
<tr>
<td>04.0712</td>
<td>04.0714</td>
<td>04.0716</td>
<td>04.0718</td>
<td>04.0720</td>
<td>04.0722</td>
<td>04.0724</td>
<td>04.0726</td>
</tr>
</tbody>
</table>

**Note:** The table represents data with columns 5F and 6F. The data seems to be formatted in a way that suggests it might be part of a larger spreadsheet or database. The specific context or the purpose of this data is not clear from the image alone.
Table B.7: Translation speed tuning set of \( \gamma \) used in the "Approach" trials of Chapter 4

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
</tr>
<tr>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Note: The table represents the translation speed tuning set of \( \gamma \) used in the "Approach" trials of Chapter 4.
Table B.8: Translation speed tuning set of $v_t$ used in the Turn trials of Chapter 4

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20034</td>
<td>0.25030</td>
<td>0.22026</td>
<td>0.19021</td>
<td>0.18021</td>
<td>0.19021</td>
<td>0.20026</td>
<td>0.25030</td>
<td>0.29034</td>
<td></td>
</tr>
<tr>
<td>0.30034</td>
<td>0.35037</td>
<td>0.32032</td>
<td>0.29027</td>
<td>0.28027</td>
<td>0.29027</td>
<td>0.30032</td>
<td>0.35037</td>
<td>0.40034</td>
<td></td>
</tr>
<tr>
<td>0.40034</td>
<td>0.45037</td>
<td>0.42032</td>
<td>0.39027</td>
<td>0.38027</td>
<td>0.39027</td>
<td>0.40032</td>
<td>0.45037</td>
<td>0.50040</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.50040</td>
<td>0.55043</td>
<td>0.52038</td>
<td>0.49033</td>
<td>0.46032</td>
<td>0.43031</td>
<td>0.50040</td>
<td>0.55043</td>
<td>0.60045</td>
</tr>
<tr>
<td>0.60045</td>
<td>0.65048</td>
<td>0.62042</td>
<td>0.59037</td>
<td>0.56036</td>
<td>0.53035</td>
<td>0.60045</td>
<td>0.65048</td>
<td>0.70050</td>
<td></td>
</tr>
<tr>
<td>0.70050</td>
<td>0.75053</td>
<td>0.72047</td>
<td>0.69042</td>
<td>0.66041</td>
<td>0.63040</td>
<td>0.70050</td>
<td>0.75053</td>
<td>0.80055</td>
<td></td>
</tr>
<tr>
<td>0.80055</td>
<td>0.85058</td>
<td>0.82052</td>
<td>0.79047</td>
<td>0.76044</td>
<td>0.73042</td>
<td>0.80055</td>
<td>0.85058</td>
<td>0.90058</td>
<td></td>
</tr>
<tr>
<td>0.90058</td>
<td>0.95061</td>
<td>0.92055</td>
<td>0.89050</td>
<td>0.86046</td>
<td>0.83042</td>
<td>0.90055</td>
<td>0.95061</td>
<td>1.00063</td>
<td></td>
</tr>
</tbody>
</table>

Note: The table represents the translation speed tuning set of $v_t$ used in the Turn trials of Chapter 4.
Table B.9: Translation speed tuning set of $v_f$ used in the closed loop trials of Chapter 5

<table>
<thead>
<tr>
<th>$y_f$</th>
<th>35</th>
<th>25</th>
<th>15</th>
<th>5</th>
<th>5</th>
<th>25</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>0.260.29</td>
<td>0.200.22</td>
<td>0.110.16</td>
<td>0.110.12</td>
<td>0.110.12</td>
<td>0.140.16</td>
<td>0.200.22</td>
</tr>
<tr>
<td>0.210.27</td>
<td>0.170.19</td>
<td>0.110.12</td>
<td>0.050.05</td>
<td>0.050.05</td>
<td>0.110.12</td>
<td>0.170.19</td>
<td>0.210.27</td>
</tr>
<tr>
<td>0.300.35</td>
<td>0.220.25</td>
<td>0.140.16</td>
<td>0.060.07</td>
<td>0.060.07</td>
<td>0.140.16</td>
<td>0.220.25</td>
<td>0.300.35</td>
</tr>
<tr>
<td>0.420.47</td>
<td>0.300.34</td>
<td>0.190.21</td>
<td>0.080.09</td>
<td>0.080.09</td>
<td>0.190.21</td>
<td>0.300.34</td>
<td>0.420.47</td>
</tr>
<tr>
<td>0.500.35</td>
<td>0.220.25</td>
<td>0.110.12</td>
<td>0.060.07</td>
<td>0.060.07</td>
<td>0.110.12</td>
<td>0.170.19</td>
<td>0.240.27</td>
</tr>
<tr>
<td>0.120.47</td>
<td>0.300.34</td>
<td>0.190.21</td>
<td>0.080.09</td>
<td>0.080.09</td>
<td>0.190.21</td>
<td>0.300.34</td>
<td>0.120.47</td>
</tr>
<tr>
<td>0.260.29</td>
<td>0.200.22</td>
<td>0.110.16</td>
<td>0.110.12</td>
<td>0.140.16</td>
<td>0.200.22</td>
<td>0.260.29</td>
<td></td>
</tr>
<tr>
<td>0.330.38</td>
<td>0.250.29</td>
<td>0.180.21</td>
<td>0.110.12</td>
<td>0.140.16</td>
<td>0.200.22</td>
<td>0.260.29</td>
<td></td>
</tr>
<tr>
<td>0.450.51</td>
<td>0.310.39</td>
<td>0.250.28</td>
<td>0.190.21</td>
<td>0.190.21</td>
<td>0.250.28</td>
<td>0.340.39</td>
<td>0.450.51</td>
</tr>
<tr>
<td>0.500.44</td>
<td>0.210.27</td>
<td>0.200.22</td>
<td>0.170.19</td>
<td>0.170.19</td>
<td>0.200.22</td>
<td>0.210.27</td>
<td>0.500.44</td>
</tr>
<tr>
<td>0.370.43</td>
<td>0.300.35</td>
<td>0.250.29</td>
<td>0.220.25</td>
<td>0.220.25</td>
<td>0.250.29</td>
<td>0.300.35</td>
<td>0.370.43</td>
</tr>
<tr>
<td>0.510.57</td>
<td>0.120.47</td>
<td>0.340.39</td>
<td>0.300.34</td>
<td>0.300.34</td>
<td>0.340.39</td>
<td>0.420.47</td>
<td>0.510.57</td>
</tr>
<tr>
<td>0.330.37</td>
<td>0.290.32</td>
<td>0.260.29</td>
<td>0.140.17</td>
<td>0.140.17</td>
<td>0.210.27</td>
<td>0.260.29</td>
<td>0.330.37</td>
</tr>
<tr>
<td>0.420.49</td>
<td>0.370.38</td>
<td>0.330.38</td>
<td>0.300.35</td>
<td>0.300.35</td>
<td>0.330.38</td>
<td>0.370.38</td>
<td>0.420.49</td>
</tr>
<tr>
<td>0.580.66</td>
<td>0.510.57</td>
<td>0.150.51</td>
<td>0.120.47</td>
<td>0.120.47</td>
<td>0.150.51</td>
<td>0.510.57</td>
<td>0.580.66</td>
</tr>
</tbody>
</table>
REFERENCES


332


