8-24-2016

Taking Swarms to the Field: Decentralized Algorithms for Underwater Swarms

Sherif Ahmed Tolba
University of Connecticut, sherif.tolba@uconn.edu

Follow this and additional works at: http://digitalcommons.uconn.edu/dissertations

Recommended Citation
http://digitalcommons.uconn.edu/dissertations/1205
Modern ocean exploration and sensing approaches have been mainly based on Autonomous Underwater Vehicles (AUVs), Remotely Operated Vehicles (ROVs), and/or static Underwater Acoustic Sensor Networks (UASNs) deployments. Individual AUVs and ROVs represent a single point of failure in addition to being bulky and expensive as vehicles are usually full-featured and sophisticated. UASNs have traditionally been statically deployed. This limits their use to original deployment locations and renders them unsuitable for search tasks. Swarm Robotics (SR) are a natural, better alternative. Swarms possess superior features over a sophisticated AUV; they are smaller, cheaper, robust, reliable, and scalable by design and definition. They also have the sensing capabilities of UASNs and built-in active mobility.

Designing successful swarm missions in harsh aquatic environments is an involved task. We address this by analyzing the indispensable stages of a typical mission and carefully designing decentralized algorithms to achieve the desired per-stage goals. Important system and environmental parameters are taken into consideration to achieve the end goal: completing mission requirements while respecting time constraint, with best possible performance and minimum loss of agents. Special attention is given to target search, task identification and allocation, and mission-stage integration due to their importance. Identifying target location in an unbounded environment is challenging. Bandwidth limited and intermittent communication complicates the process further. Therefore, we develop global search algorithms that use minimal communication and utilize flocking to maintain cohesion. These algorithms have multiple advantages over traditional ones in terms of convergence time, omni-directionality, consideration of physical constraints, and being self-bounding. At the
target, tasks are autonomously identified and allocated in a completely decentralized manner. Validation of the developed techniques is done through realistic simulations and analytical comparisons.

Our main contributions are: 1) a general framework for underwater mission planning, 2) three novel global search algorithms for unbounded underwater environment, 3) an algorithm for initial self-organization, 4) an optimized same-position reorientation algorithm for use in certain mission stages, 5) three autonomous task allocation algorithms, 6) three local target search algorithms, 7) a measure of mission utility, and 8) the design of a human brain-inspired model to support learning and complete autonomy.
Taking Swarms to the Field: Decentralized Algorithms for Underwater Swarms

Sherif Ahmed Tolba

B.Sc., Mansoura University, 2007
M.Sc., University of Connecticut, 2011

A Dissertation
Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2016
Acknowledgments

First and foremost, I thank Allah, Most Gracious, Most Merciful for guiding me and giving patience and strength to make this journey. Without His generosity and mercy, I would not have been able to complete it. I would also like to thank my great advisor, Prof. Reda Ammar, for his sincere advice, guidance, patience, and kindness. He understands his students’ work- and research-style very well and allows them to excel and innovate. Certainly, I would also like to thank my associate advisors: Prof. Sanguthevar Rajasekaran and Prof. Bing Wang for their sincere advice, cooperation, and open-mindedness.

Out of my sincere love and gratitude towards them, my beloved parents, Ahmed and Mona, are the next to thank. They have always been there for me; supportive, kind, patient, generous, merciful, and the list goes on and on. I am absolutely nothing without them. Next, I would like to give my lovely wife, Amal, her due respect and thanks. She was extremely patient, supportive, encouraging, and took great care of me and of our lovely kids all the time. She put me first and sacrificed many of her ambitions to help me reach what I wanted and made it our common goal. Thanks to my kids who made my life worth living and filled it with happiness!

Thanks to my sister, Dina, and my relatives who always thought about me while I am away and wished me the best. Thanks to my friends and colleagues who have always been there for me and made my graduate life rich and delightful. Thanks to my former supervisors and coworkers at UITS who were always very supportive, kind, and sincere advisors. Finally, I would like to thank anybody who was there for me and supported me in any way and I did not mention explicitly in this acknowledgment.
Contents

1 Introduction

1.1 Motivation and Outline ........................................ 2
1.2 Planet Earth - The Water Planet .............................. 4
1.3 The Challenging Underwater Environment .................. 4
  1.3.1 Underwater Communication ................................ 4
  1.3.2 Localization .............................................. 5
  1.3.3 Currents ..................................................... 6
  1.3.4 Open Ocean ................................................ 7
1.4 A Compelling Need ............................................. 7
1.5 Problem Definition .............................................. 8
  1.5.1 Statement .................................................... 8
1.6 Swarm Robotics Overview .................................... 10
  1.6.1 Biological Inspiration .................................... 10
  1.6.2 To Bio or to Mix .......................................... 11
  1.6.3 Motion Coordination ...................................... 12
  1.6.4 Flocking ................................................... 13
  1.6.5 Emerging Behavior ........................................ 13
  1.6.6 Self-Organization ......................................... 13
| 3.6.1 | Robotic PSO (R-PSO) | 66 |
| 3.6.2 | Virtual Tether Search (VTS) | 71 |
| 3.6.3 | Constrained Spiral Flocking (CSF) | 77 |
| 3.6.4 | Simple Sweeping (SSW) | 83 |
| 3.6.5 | Swirling Divided Hexagonal Close Packing (SDHCP) | 84 |
| 3.6.6 | Simple Random Walk (SRW) | 88 |
| 3.7 | Task Identification and Allocation | 88 |
| 3.7.1 | Task Allocation Techniques | 90 |
| 3.7.2 | Response Threshold-Based Task Allocation | 94 |
| 3.7.3 | Beacon Based Task Allocation | 96 |
| 3.7.4 | Hybrid Task Allocation | 102 |
| 3.7.5 | Blind Task Allocation | 105 |
| 3.8 | Local Target Search | 105 |
| 3.8.1 | Retracted-Sequence Random Walk | 106 |
| 3.8.2 | Bubble-Chain Random Walk | 108 |
| 3.8.3 | Tethered Random Walk | 111 |
| 3.9 | Mission Integration | 112 |
| 3.9.1 | Mission Profit | 114 |
| 3.10 | Reinforcement Learning | 116 |
| 3.10.1 | Basic Concepts | 116 |
| 3.10.2 | Reinforcement Learning Methods | 122 |
| 3.11 | Mini-Brain: a Brain-Inspired Learning Model | 123 |
| 3.11.1 | Analogy to Human Brain | 124 |
| 3.11.2 | Definitions | 127 |
| 3.11.3 | MiniBrain Areas and Subareas | 130 |
3.11.4 Sensory Area (SA) .................................................. 130
3.11.5 Emotional Area (EA) ............................................. 131
3.11.6 Higher Mental Function (HMF) ................................ 134
3.11.7 Motor Function (MF) .............................................. 136
3.11.8 Relevance to Swarm Robotics ................................. 138

4 Results and Discussion ................................................. 140
4.1 Experimental Setup ................................................... 140
4.2 Performance Metrics .................................................. 141
4.3 Target Search ............................................................ 142
  4.3.1 Performance of VTS ............................................. 143
  4.3.2 Performance of CST ............................................. 147
  4.3.3 Performance of R-PSO ......................................... 153
  4.3.4 Performance of SDHCP ........................................ 158
  4.3.5 Performance of SSW ............................................ 163
  4.3.6 Performance of SRW ............................................ 167
  4.3.7 Algorithms Performance Comparison ....................... 171
4.4 Task Allocation ........................................................ 175
  4.4.1 Performance of BL-TA ......................................... 176
  4.4.2 Performance of BB-TA ......................................... 177
  4.4.3 Performance of HYB-TA ...................................... 177
  4.4.4 Performance of ERT-TA ...................................... 179
  4.4.5 Algorithms Performance Comparison ....................... 180
4.5 Case Studies ............................................................ 181
  4.5.1 Case Study I: Single Target .................................. 182
  4.5.2 Case Study II: Multiple, Spatially Co-Located, Intermixed Targets . 186
4.5.3 Case Study III: Multiple, Spatially Separated, Distinct Targets . . . . 192

4.5.4 Case Study IV: Multiple, Spatially Separated, Intermixed Targets . . 197

5 Conclusions and Future Work 203
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Common research areas in Swarm Robotics. Highlighted areas are the ones</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>considered in this work.</td>
<td></td>
</tr>
<tr>
<td>1-2</td>
<td>CoCoRo Project: Collective Cognitive Robots; the “Lily” platform in action.</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>The image is taken from [75].</td>
<td></td>
</tr>
<tr>
<td>1-3</td>
<td>Serafina platform. The image is taken from [102].</td>
<td>16</td>
</tr>
<tr>
<td>1-4</td>
<td>SWARMS Project: ROV repairing the issues detected by an AUV in the</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>application of corrosion prevention in offshore installations. The image is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>taken from [53].</td>
<td></td>
</tr>
<tr>
<td>3-1</td>
<td>Mission planning framework.</td>
<td>27</td>
</tr>
<tr>
<td>3-2</td>
<td>Swarm navigation in the presence of time-varying flows - figure follows Figure 1 in [50].</td>
<td>33</td>
</tr>
<tr>
<td>3-3</td>
<td>Snapshot of the Complex-Environment mode of SWIM simulator.</td>
<td>40</td>
</tr>
<tr>
<td>3-4</td>
<td>Organization of packages in SWIM simulator.</td>
<td>41</td>
</tr>
<tr>
<td>3-5</td>
<td>Vehicle model used, showing the ranges and spans of the five sensors.</td>
<td>45</td>
</tr>
<tr>
<td>3-6</td>
<td>Finite State Machine used in the AUV’s controller.</td>
<td>46</td>
</tr>
<tr>
<td>3-7</td>
<td>Minimum Turn Radius (MTR) and Maximum Turn Angle (MTA).</td>
<td>48</td>
</tr>
<tr>
<td>3-8</td>
<td>Simple vehicle reorientation algorithm.</td>
<td>49</td>
</tr>
<tr>
<td>3-9</td>
<td>Concept of the optimized reorientation algorithm.</td>
<td>51</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>3-10</td>
<td>Reorienting left (right): path comparisons.</td>
<td>53</td>
</tr>
<tr>
<td>3-11</td>
<td>Finding generalized length of the reorientation path</td>
<td>54</td>
</tr>
<tr>
<td>3-12</td>
<td>Change of $\alpha$ and $\beta$ with $\theta$.</td>
<td>56</td>
</tr>
<tr>
<td>3-13</td>
<td>Full range of $\theta$ and $\alpha$.</td>
<td>56</td>
</tr>
<tr>
<td>3-14</td>
<td>Deriving the general equations for the two-armed path with straight start and end line segments.</td>
<td>57</td>
</tr>
<tr>
<td>3-15</td>
<td>Change of path length with $\beta$ for the optimal path ($l_p$) and the two-armed path ($l_{p2}$).</td>
<td>59</td>
</tr>
<tr>
<td>3-16</td>
<td>Reorienting backwards: path comparisons.</td>
<td>60</td>
</tr>
<tr>
<td>3-17</td>
<td>Reorienting in arbitrary directions.</td>
<td>61</td>
</tr>
<tr>
<td>3-18</td>
<td>Reynolds' flocking (Image taken from: <a href="http://www.red3d.com/cwr/boids/">http://www.red3d.com/cwr/boids/</a>).</td>
<td>62</td>
</tr>
<tr>
<td>3-19</td>
<td>Normalized tunable half sigmoid function with $k = 0.01$.</td>
<td>63</td>
</tr>
<tr>
<td>3-20</td>
<td>Snapshot of 100 AUVs applying Initial Self-Organization algorithm.</td>
<td>65</td>
</tr>
<tr>
<td>3-21</td>
<td>PSO update subprocess running within Update state of the FSM.</td>
<td>69</td>
</tr>
<tr>
<td>3-22</td>
<td>Theoretical basis for VTS search.</td>
<td>73</td>
</tr>
<tr>
<td>3-23</td>
<td>VTS search: angle selection probability change with radius.</td>
<td>74</td>
</tr>
<tr>
<td>3-24</td>
<td>Two concatenated logarithmic spirals used by CSF search algorithm as the main path of motion.</td>
<td>81</td>
</tr>
<tr>
<td>3-25</td>
<td>Simple Sweeping algorithm illustration.</td>
<td>84</td>
</tr>
<tr>
<td>3-26</td>
<td>Divided Hexagonal Close Packing algorithm illustration.</td>
<td>86</td>
</tr>
<tr>
<td>3-27</td>
<td>Swirling Divided Hexagonal Close Packing algorithm illustration.</td>
<td>87</td>
</tr>
<tr>
<td>3-28</td>
<td>Retracted-Sequence Random Walk algorithm (RS-RW) illustration.</td>
<td>107</td>
</tr>
<tr>
<td>3-29</td>
<td>Bubble-Chain Random Walk algorithm (BC-RW) illustration.</td>
<td>109</td>
</tr>
<tr>
<td>3-30</td>
<td>Tethered-Sequence Random Walk algorithm (TS-RW) illustration.</td>
<td>112</td>
</tr>
<tr>
<td>3-31</td>
<td>Agent-environment interaction. Taken from [88].</td>
<td>117</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>3-32</td>
<td>Functions of human brain. © 2016. Nucleus Medical Media. All rights reserved. <a href="http://www.nucleusinc.com">www.nucleusinc.com</a></td>
<td></td>
</tr>
<tr>
<td>3-33</td>
<td>High-level components of MiniBrain, their connections, and interactions.</td>
<td></td>
</tr>
<tr>
<td>3-34</td>
<td>Types of triggers used in the current implementation of MiniBrain.</td>
<td></td>
</tr>
<tr>
<td>3-35</td>
<td>A simplified version of <em>The Feeling Wheel</em> used in the design of MiniBrain.</td>
<td></td>
</tr>
<tr>
<td>3-36</td>
<td>Relationships between triggers, emotion, action, situation, and behavior.</td>
<td></td>
</tr>
<tr>
<td>3-37</td>
<td>Structural and functional representation of MiniBrain: (top) brain-to-brain areas relationship, (bottom) brain areas-to-subareas relationship.</td>
<td></td>
</tr>
<tr>
<td>3-38</td>
<td>Structure and function of the sensory area (SA).</td>
<td></td>
</tr>
<tr>
<td>3-39</td>
<td>Subareas, neural networks, and emotion-emotion interactions inside the emotional area of the MiniBrain.</td>
<td></td>
</tr>
<tr>
<td>3-40</td>
<td>General structure of an emotional sub-area.</td>
<td></td>
</tr>
<tr>
<td>3-41</td>
<td>Function of emotional area.</td>
<td></td>
</tr>
<tr>
<td>3-42</td>
<td>Structure and functions of the Higher Mental Function.</td>
<td></td>
</tr>
<tr>
<td>3-43</td>
<td>Structure and functions of the Motor Function area.</td>
<td></td>
</tr>
<tr>
<td>3-44</td>
<td>Motor function parameters used by Motor Function area.</td>
<td></td>
</tr>
<tr>
<td>4-1</td>
<td>VTS – Number of AUVs that reached target.</td>
<td></td>
</tr>
<tr>
<td>4-2</td>
<td>VTS – Percentages of AUVs that reached target for different swarm sizes.</td>
<td></td>
</tr>
<tr>
<td>4-3</td>
<td>VTS – Average distance to target.</td>
<td></td>
</tr>
<tr>
<td>4-4</td>
<td>VTS – Average spread.</td>
<td></td>
</tr>
<tr>
<td>4-5</td>
<td>VTS – Number of lost AUVs.</td>
<td></td>
</tr>
<tr>
<td>4-6</td>
<td>VTS – Overall distance traveled by all AUVs.</td>
<td></td>
</tr>
<tr>
<td>4-7</td>
<td>CSF – Number of AUVs that reached target.</td>
<td></td>
</tr>
<tr>
<td>4-8</td>
<td>CSF – Number of lost AUVs.</td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4-9</td>
<td>CSF – Percentages of surviving, lost, and AUVs that reached target for different swarm sizes.</td>
<td></td>
</tr>
<tr>
<td>4-10</td>
<td>CSF – Average distance to target.</td>
<td></td>
</tr>
<tr>
<td>4-11</td>
<td>CSF – Average spread.</td>
<td></td>
</tr>
<tr>
<td>4-12</td>
<td>CSF – Number of sent messages.</td>
<td></td>
</tr>
<tr>
<td>4-13</td>
<td>CSF – Number of received messages.</td>
<td></td>
</tr>
<tr>
<td>4-14</td>
<td>CSF – Overall distance traveled by all AUVs.</td>
<td></td>
</tr>
<tr>
<td>4-15</td>
<td>R-PSO – Number of AUVs that reached target.</td>
<td></td>
</tr>
<tr>
<td>4-16</td>
<td>R-PSO – Number of lost AUVs.</td>
<td></td>
</tr>
<tr>
<td>4-17</td>
<td>R-PSO – Percentages of surviving, lost, and AUVs that reached target for different swarm sizes.</td>
<td></td>
</tr>
<tr>
<td>4-18</td>
<td>R-PSO – Average distance to target.</td>
<td></td>
</tr>
<tr>
<td>4-19</td>
<td>R-PSO – Average spread.</td>
<td></td>
</tr>
<tr>
<td>4-20</td>
<td>R-PSO – Number of sent messages.</td>
<td></td>
</tr>
<tr>
<td>4-21</td>
<td>R-PSO – Number of received messages.</td>
<td></td>
</tr>
<tr>
<td>4-22</td>
<td>R-PSO – Overall distance traveled by all AUVs.</td>
<td></td>
</tr>
<tr>
<td>4-23</td>
<td>SDHCP – Number of AUVs that reached target.</td>
<td></td>
</tr>
<tr>
<td>4-24</td>
<td>SDHCP – Number of lost AUVs.</td>
<td></td>
</tr>
<tr>
<td>4-25</td>
<td>SDHCP – Percentages of surviving, lost, and AUVs that reached target for different swarm sizes.</td>
<td></td>
</tr>
<tr>
<td>4-26</td>
<td>SDHCP – Average distance to target.</td>
<td></td>
</tr>
<tr>
<td>4-27</td>
<td>SDHCP – Average spread.</td>
<td></td>
</tr>
<tr>
<td>4-28</td>
<td>SDHCP – Number of sent messages.</td>
<td></td>
</tr>
<tr>
<td>4-29</td>
<td>SDHCP – Number of received messages.</td>
<td></td>
</tr>
<tr>
<td>4-30</td>
<td>SDHCP – Overall distance traveled by all AUVs.</td>
<td></td>
</tr>
</tbody>
</table>
4-31 SSW – Number of AUVs that reached target. ................................. 164
4-32 SSW – Percentages of AUVs that reached target for different swarm sizes. 165
4-33 SSW – Average distance to target. .............................................. 165
4-34 SSW – Average spread. ............................................................. 166
4-35 SSW – Overall distance traveled by all AUVs. .............................. 167
4-36 SRW – Number of AUVs that reached target. .............................. 167
4-37 SRW – Number of lost AUVs. .................................................... 168
4-38 SRW – Percentages of surviving, lost, and AUVs that reached target for different swarm sizes. ............................. 168
4-39 SRW – Average distance to target. .............................................. 169
4-40 SRW – Average spread. ............................................................. 170
4-41 SRW – Overall distance traveled by all AUVs. .............................. 170
4-42 Comparison of number of lost AUVs for different algorithms. ............. 172
4-43 Comparison of number of AUVs that reached target for different algorithms. 173
4-44 Comparison of overall distances traveled by the whole swarm for different algorithms. ......................................................... 174
4-45 Configuration of targets used in the simulation. .............................. 175
4-46 BL-TA - Percentage of processed targets over time. ......................... 176
4-47 BL-TA - Average number of per-target processings. ....................... 177
4-48 BB-TA - Percentage of processed targets over time. ....................... 178
4-49 BB-TA - Average number of per-target processings. ....................... 178
4-50 HYB-TA - Percentage of processed targets over time. ...................... 179
4-51 HYB-TA - Average number of per-target processings. ...................... 179
4-52 ERT-TA - Percentage of processed targets over time. ...................... 180
4-53 ERT-TA - Average number of per-target processings. ...................... 180
4-54 Comparison of average time taken to process all targets for different algorithms. 181

4-55 Comparison of average number of per-target processings for different algorithms. 182

4-56 Case Study I: a complete mission of searching for a sunk ship using SDHCP for global search, BL-TA for task allocation, and T-RW for local target search. 184

4-57 Case Study I – Average distance to target. 185

4-58 Case Study I – Average spread. 185

4-59 Case Study I – Number of AUVs that reached target over time. 186

4-60 Case Study I – Number of lost AUVs over time. 186

4-61 Case Study I – Overall distance traveled by the swarm. 187

4-62 Case Study II: a complete mission of searching for a group of intermixed targets using CSF for global search, BB-TA for task allocation, and T-RW for local target search. 188

4-63 Case Study II – Average distance to target. 189

4-64 Case Study II – Average spread. 190

4-65 Case Study II – Number of AUVs that reached target over time. 191

4-66 Case Study II – Number of lost AUVs over time. 191

4-67 Case Study II – Overall distance traveled by all AUVs. 192

4-68 Case Study II - Percentage of processed targets over time. 192

4-69 Case Study III: a complete mission of searching for a group of spatially separated, distinct targets using CSF for global search, BB-TA for task allocation, and T-RW for local target search. 194

4-70 Case Study III – Average spread. 195

4-71 Case Study III – Number of AUVs that reached target over time. 195

4-72 Case Study III – Overall distance traveled by all AUVs. 196

4-73 Case Study III - Percentage of processed targets over time. 196
Case Study IV: a complete mission of searching for a group of spatially separated intermixed targets using VTS for global search, BB-TA for task allocation, and T-RW for local target search.

- Case Study IV – Average spread.
- Case Study IV – Number of AUVs that reached target over time.
- Case Study IV – Number of lost AUVs over time.
- Case Study IV – Overall distance traveled by all AUVs.
- Case Study IV - Percentage of processed targets over time.
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Robotic PSO algorithm parameter values.</td>
<td>70</td>
</tr>
<tr>
<td>3.2</td>
<td>Virtual Tether Search algorithm parameter values.</td>
<td>76</td>
</tr>
<tr>
<td>3.3</td>
<td>Constrained Spiral Flocking algorithm parameter values.</td>
<td>83</td>
</tr>
<tr>
<td>3.4</td>
<td>Thresholds and stimuli used by Hybrid Task Allocation. ATF: Adaptive Threshold Function</td>
<td>103</td>
</tr>
<tr>
<td>4.1</td>
<td>Case Study I: values of the parameters used in mission gain calculation.</td>
<td>187</td>
</tr>
<tr>
<td>4.2</td>
<td>Case Study II: values of the different variables/parameters used.</td>
<td>189</td>
</tr>
<tr>
<td>4.3</td>
<td>Case Study III: values of the different variables/parameters used.</td>
<td>193</td>
</tr>
<tr>
<td>4.4</td>
<td>Case Study IV: values of the different variables/parameters used.</td>
<td>199</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The field of Swarm Robotics (SR) emerged from the interaction and overlap of multiple disciplines and concepts including Artificial Intelligence (AI), Swarm Intelligence (SI), Robotics, Machine Learning (ML), and Systems Biology. It is concerned with studying the mechanisms used in natural swarms like ant and bee colonies, fish schools, animal herds, bird flocks, etc. and developing bio-inspired mechanisms to accomplish tasks that are otherwise difficult or impossible to be accomplished by single units. It also explores the design and use of other non-biologically inspired techniques that can achieve desired collective and emerging behaviors.

Robotic swarms use the principles of decentralized control, self-organization (SO), group dynamics, stigmergy (indirect communication through the environment), division of labor, among others to accomplish tasks beyond the capabilities of single individuals. Collective behaviors emerge as a result of the execution of simple rules by individuals in the swarm, and are considered one of the main goals in swarm robotics.

As a natural extension to underwater acoustic sensor networks (UASNs) and a complement to lack of active mobility in nodes, the use of underwater swarms has recently attracted many research teams. Underwater environments are known for their harsh conditions man-
ifested in continuous surface and deep water currents, moving bodies, high corrosion rates, limited communication bandwidth, and high attenuation. Many of these limitations can be better addressed by the use of Autonomous Underwater Vehicle (AUV) swarms when it comes to data collection or physical manipulation at/of a target located deep below the water surface.

In this work, we utilize the powers of SR to accomplish complex underwater missions. Due to the limitations of underwater communication, we focus on developing decentralized algorithms that depend mainly on sensing, with occasional use of communication over short ranges. We use combinations of biological inspiration and heuristics to accomplish this goal, with performance as our main consideration.

1.1 Motivation and Outline

Although numerous works in the literature address the design and implementation of individual mission stages, no comprehensive framework or approach exists to organize these contributions and bridge the gaps between them. Such a framework would be essential in serving the purpose of general and abstract mission planning. For underwater swarms, the body of literature is even more limited. Our goal in this research is two-fold: 1) Suggesting a generalizable framework for underwater mission planning that is application independent, and 2) Developing efficient decentralized algorithms for successfully accomplishing different phases of a typical mission.

The work is divided into the following components: First, a framework for underwater AUVs swarm mission planning is presented. This is where a typical mission is studied and important stages are pointed out. A discussion of the design and features of our specially designed simulator for underwater swarms follows framework presentation. Next, we propose
a decentralized algorithm for the initial self-organization stage after developing suitable AUV (re)orientation algorithms to account for its physical constraints. After that, algorithms for global target search, task allocation, and local target search are explained. These are followed by the integration of mission stages and the development of a mission evaluation measure. Finally, we present our MiniBrain model for autonomous learning following an introduction to Reinforcement Learning (RL). These sections are summarized below:

- **Mission Planning Framework** – typical underwater swarm mission stages are identified and formalized;

- **Simulator Design** – design and implementation of our underwater SWarm sIMulator (SWIM) are presented;

- **Vehicle Orientation** – algorithms for (re)orienting a physically constrained AUV are presented;

- **Initial Self-Organization (SO)** – a decentralized algorithm is introduced and validated through simulation;

- **Global & Local Target Search** – decentralized algorithms are explained and validated through simulations and comparisons;

- **Task Allocation** – decentralized algorithms are explained and validated through simulations and comparisons;

- **Interplay Between Mission Stages** – interplay and seamless integration between different stages are emphasized

- **Reinforcement Learning and MiniBrain** – Reinforcement Learning basics are covered and our MiniBrain model for autonomous learning is introduced
1.2 Planet Earth - The Water Planet

Liquid water covers more than 70% of the surface of the Earth, yet, less than 10% of the ocean has been explored so far [78, 76]. Surprisingly, 90% of the inhabitable space on earth is in the ocean [58]. Its average depth is 3.7km (∼ 2.3mi) while the deepest part of the ocean, located in the Mariana trench, is around 11km (∼ 7mi) [86, 2]. As a natural consequence of this enormous volume occupied by the ocean, it is the source of at least 50% of the oxygen on earth and has most of the organisms on the planet [58]. It is full of riches like oil, natural gas, minerals (e.g. in contains nearly 20 million tons of gold), and undoubtedly, fishes which provide the greatest supply of worldwide protein that humans consume [76].

1.3 The Challenging Underwater Environment

The nature of underwater environment and its intrinsic constraints make it unique when compared to ground or air. Different from air, light and radio waves are highly attenuated underwater. The viable alternative is acoustic waves, which still suffer from limited bandwidth. For this reason communication as well as localization are severely affected in underwater environment. Different types of currents and perturbations are another limiting factor. Other factors that make oceans and seas a really challenging environment include the unboundedness and the characteristic three dimensional nature. In the following sections, we discuss some of these limitations in more detail.

1.3.1 Underwater Communication

As far as communication is concerned, several limitations manifest: limited bandwidth, multipath propagation, time-varying channels, and spatial variation just to name a few. Bandwidth and transmission range are greatly affected by the signal-to-noise ratio (SNR).
The latter is, in turn, determined by transmission loss and noise level. Transmission loss takes two forms: energy spread with distance and sound absorption with both distance and the frequency range. This imposes a limitation on the usable bandwidth. Additionally, transmission loss changes spatially based on the presence of shadow zones. Noise on the other hand can be ambient or man-made (e.g. ship movements). Another factor affecting acoustic communication is multipath propagation, which is based on water depth (shallow or deep water), frequency, and transmission range. Channel temporal variations that result from multipath propagation also cause problems like inter symbol interference (ISI) [7]. In order to have successful and efficient communication, all these factors must be taken into consideration when selecting the acoustic communication system and during mission planning. The effect of these factors is especially obvious when a single or small number of AUVs are used and even in sparsely deployed underwater sensor networks. In robotic swarms, they are expected to have a weaker effect as swarms are meant to consist of large numbers of agents that are usually dense. However, another factor comes into play; mobility, which introduces its own issues. Some works and real implementations exist in the literature that use other forms of communication like light signals [8] or RF communication [9].

1.3.2 Localization

Differently from terrestrial sensor, vehicular, and mobile wireless networks, Global Positioning System (GPS) cannot be used in underwater environments. Radio signals are highly attenuated in such environments rendering the technology useless. Alternative technologies like Inertial Measurement Units (IMUs) and Doppler Velocity Log (DVL) have been used for underwater localization, but suffer from accumulated measurement errors and high prices for accurate units. The use of a combination of acoustic ranging and surfacing (to get satellite based location information) for localization has been shown to provide better local-
ization [18]. However, surfacing is not always possible for all missions. Many factors come into play when a localization mechanism is to be designed and implemented for use in AUVs and UASNs. Some of these are: passive and active mobility and its associated uncertainty, long transmission and propagation delays due to low bandwidth and range-bit-rate product, and limited energy supply.

### 1.3.3 Currents

Ocean currents take many forms; the two major ones are deep water currents (thermohaline circulation) and great surfaces currents (surface circulation). Other currents and vertical water movements include eddies and tidal currents, upwelling, and downwelling. Surface currents are mainly formed by winds, solar heat, gravity, and the Coriolis Force (a force resulting from earth’s rotation causing objects to deflect to the right in the northern hemisphere and to the left in the southern hemisphere) [12, 85]. These surface currents form large circular patterns called gyres that flow clockwise in the northern hemisphere and counter-clockwise in the southern. Surface currents extend down to 400 meters below the surface. Eddies, swirling loops of water, are bends generated in the flow of surface currents and can also be driven by the topology of the ocean floor [87]. Deep water currents are generated at the poles when surface water becomes very cold during the long, dark days of the winter which makes it denser and denser. As only freshwater undergoes the freezing process, cold, saline, dense water rapidly sinks down to the bottom of the ocean then starts spreading slowly. This process produces the deep water current that flow from the poles towards lower latitudes and form the great conveyor belt that cycles the global ocean. On the surface warmer water flows from the equator to replace that water, keeping the circulation continuing indefinitely [86]. Tidal currents are the only type of currents affected by the interaction between earth, moon, and the sun. They are caused by the rise and fall of the tide that causes the water to move.
horizontally near the shore [89]. Upwelling (downwelling) takes place when the wind blows along the coastline causing the water to move perpendicularly away (towards) the coast. This causes water from the bottom of the ocean to upwell (downwell) near the coast [87].

All these currents make the ocean a very adverse environment and put a big burden on marine systems designers to take their effect into consideration at design, test, and deployment times.

1.3.4 Open Ocean

Unlike indoor or controlled outdoor environments which are used in ground robotic swarms, the ocean is an environment that is wide-open with no clear bounds to prevent agents from dispersion and loss. They span vast areas and are not as accessible to emergency intervention as in ground or aerial scenarios. This drives the need for self-bounding techniques that are fitted for such open environments. In this work, we pay special attention to this characteristic and develop algorithms that are able to constrain the search area within a predetermined radius.

1.4 A Compelling Need

There is a continuous and growing need for the ability to retrieve objects from the bottom of the sea, and this need requires the appropriate tools to fulfill it. Examples of applications where this need emerges include: locating sunk ships, airplanes, treasures, monuments, inspecting submerged nuclear waste, taking soil samples for scientific research, exploration of lost underwater towns, inspecting leakages in underwater oil pipes, inspecting fiber-optic cables for cuts, etc. Use of remote sensing systems or other means like ROVs, ships pulling large hydrophones (towed array sonar), single AUVs, or divers is either expensive or unsuit-
able for deeply submerged objects. Swarms of AUVs provide a viable alternative as they can autonomously perform the mission and reach to places where other techniques can’t. Their active mobility gives them an added advantage over other static sensing approaches like UASNs.

1.5 Problem Definition

In this section, the problem this research is trying to solve is first stated, then constituent parts are highlighted to facilitate the solution process. These parts will serve as seeds for the general mission planning framework that is presented later in Section 3.1.

1.5.1 Statement

The problem this work is trying to solve can be summarized as follows:

*Given a specific underwater mission and a robotic swarm of fixed size, released from a ship relatively close to a target’s location, how to successfully find and process the target while taking mission constraints and practical limitations into consideration?*

*Mission constraints include but are not limited to: time constraint, mission profit, and maximum search radius.*

*Practical limitations include but are not limited to: physical vehicle constraints, agent loss probability, and limited communication and sensing ranges.*

Although this statement is may seem very general and intractable, it is very typical when it comes to the requirements of any successful mission. Because this is a difficult problem, we follow a divide-and-conquer approach. First, to simplify the design process,
it is useful to divide the task at hand into multiple subtasks/stages, specify the details of each, and finally combine them in order to reconstruct the original problem. As mentioned in the statement, we consider the scenario of a ship releasing a swarm of robots into the water in order to perform a specific mission, e.g. black box search function, and return back to the ship when the task has been successfully accomplished. The swarm first disperses into water and self-organizes its agents to keep appropriate separation and become ready to move to the target destination. This stage can be called: initial self-organization. Next, the swarm can optionally form a shape, e.g. teardrop, while it transports towards the target destination. This consists of two parts, the first is the formation of a shape that guarantees energy balancing among swarm agents, facilitates the transport of the group of agents, and encounters minimum resistance from water currents. The second part is the transport process itself, which involves keeping minimum and maximum distances between agents, planning the global search path, and moving at consistent speeds that lead to the arrival to the destination in the smoothest way possible. The next stage is to search locally and evenly distribute themselves, at the destination, to allow performing the task in the most efficient way. Once the processing or acquisition stage has been accomplished, the swarm repeats the previous steps of reorganizing its agents and then moving in a cohesive way back to the origin where the data has to be delivered. Alternatively, and in the simplest scenarios, agents can float from the target’s location directly to the surface where they can be picked up (this approach is used in this work for simplicity). During this return trip, additional processing may be done where collected data can be processed by the individual agents, and probably collectively by exchanging minimal information. This allows collected data to be ready when the swarm is recovered. Further processing can still be done after pickup.

Integration of these individual stages forms the complete mission that this research is studying. Smooth transition between stages is another important aspect that requires careful
attention to successfully achieve mission goals.

In Chapter 3, we use these stages to build a general mission planning framework usable in different applications, study most of these individual stages, and their integration. We also provide a measure for the degree of mission success at the end of that chapter.

In the next section, an overview of SR is presented to familiarize the reader with its basic principles and pave the road for the rest of this work.

1.6 Swarm Robotics Overview

Although the field of Swarm Robotics is relatively new, several research directions have emerged within the field, each has multiple research teams focusing on these specific subfields. In this section we give a brief overview of some of these subfields and concepts. Figure 1-1 summarizes the most common areas and highlights the ones that are studied in this work.

1.6.1 Biological Inspiration

Nature has numerous examples of biological systems that involve the concept of swarming. Examples include but are not limited to organized flight in birds (bird flocks) [5, 33], animal herds, ant and bee colonies, bacteria [69], termites, fish schools [23], wasps, locusts, fireflies, stem cells, glow worms, and many others.

These biological systems use different mechanisms to communicate either explicitly, e.g. using sound like in birds, or in an implicit way through stigmergy; communication through the modification of the environment. Ant pheromones are an example of such mechanism. The common feature of all these systems is that they use sets of simple rules to emerge a collective behavior at the swarm level that is far more complex than individual abilities.

Research in the field of swarm robotics benefits greatly form these biological swarms
by drawing inspiration from them to develop the targeted artificial swarms. Works that highlighted these inspiration sources include \[69, 57\]. In \[24\], a comprehensive overview of bio-inspired design patterns are presented which can greatly assist swarm engineers in the design and implementation of bio-inspired artificial swarms.

1.6.2 To Bio or to Mix

It is important for the swarm engineer to keep in mind that the ultimate goal of the artificial swarm is to accomplish the task under consideration as efficiently as possible. Therefore, only trying to mimic natural swarms and not carefully considering the requirements of the
application can lead to undesirable results. A balance between biological inspiration and heuristic or optimization techniques is required to build practical artificial swarm systems. Finding the correct balance is still an open research problem that multiple teams around the world are actively working on.

1.6.3 Motion Coordination

Motion coordination involves path planning and pattern formation [51]. In path planning, the goal of the robots is to plan a path (usually seeking the optimal one) between two locations while taking obstacles and physical constraints of the vehicles into consideration. It can be based on local information, available through sensors, or global information that is either defined a priori or provided by a central unit. Additionally, it can be in a static environment, where obstacle locations and shapes are known, or in a dynamic environment. In the latter case, planning is much difficult because by the time the path is planned, information used for planning may have become outdated. Another difficulty in path planning is dealing with 3D environments especially in the case of underwater and aerial swarms [51].

Pattern formation or formation control is a closely related concept to path planning. Its goal is to generate and maintain a specific formation (shape) at the swarm level. Control can be done in a centralized manner, where a central agent issues motion commands, or in a decentralized way where agents try to collectively generate the formation and maintain it through local coordination. It is also possible to have a combination of local and global coordination. Path planning plays an important role in formations as it assists in transitions between different formations along collision-free trajectories [28, 51].
1.6.4 Flocking

Flocking is the coherent motion of a group of self-propelled particles or organisms as a result of application of a set of simple, local interactions between them [80, 91]. It is a phenomenon that is observed in multiple biological systems and organisms like birds, fish, locusts, animals, among others. Although the rules applied by each agent are usually relatively simple, the resulting overall behavior appears to sophisticated.

This concept has been adopted by swarm robotics researchers [91], computer graphics designers and game programmers [66], physicists [80], and many other disciplines [33]. In the context of swarm robotics, it helps maintaining the unity of the swarm and can enable it to perform some tasks that individuals cannot if they are alone.

1.6.5 Emerging Behavior

Emerging behavior is the global or systematic result of local interactions between the individuals forming a population [27]. This behavior is not observable, and most of the time unpredictable, at the individual level. For example, flocking, described in Section 1.6.4, is a behavior that emerges from the application of simple social forces like collision avoidance, flock centering, and matching of velocity and heading [55, 82].

1.6.6 Self-Organization

Self-organization (SO) is defined as a set of dynamical mechanisms, taking place independently and individually on different agents at the local level, that lead to the emergence of collective behavior or a pattern at the global level of the swarm [4]. Research in self-organization in distributed robotic systems dates back to the early 90’s. For example, [92] introduced the concept of cellular robotics and the Cellular Robotic System CEBOT, and
presented a self-organization approach using random walks and simple rules. Their primitive-life inspired approach was evaluated using appropriate evaluation functions and the effect of balance between selfishness and cooperation of robotic cells and the number of cells carrying genetic information on the effectiveness of the approach were studied. In [49], an evolutionary self-organization approach for collective task handling in distributed robots was developed and applied to the task of constructing a global spatial map of an unknown environment. In 2002, [79] presented the Digital Hormone Model (DHM); a model for self-organization applicable to multi-robot systems. The model was an integration of the reaction-diffusion model and stochastic cellular automata. It used digital hormone-like signals and hormone concentrations in the neighboring space of a robot to control its movement. Recently, Ye et al. [99] introduced a composite mechanism for self-organization in agent networks. Agents used a trust model, a Q-learning algorithm, and weighted relations to adapt their behaviors with other agents in order to achieve better task allocation.

1.6.7 Decentralization

Decentralization is one of the characteristics of swarms by definition. It is mainly concerned with making decisions autonomously and independently in a way that emerges the desired behavior at the swarm level. Although natural swarms like bee, termite, and ant colonies have a queen, the queen does not act as a central controller of the swarm. These swarms have emergency mechanisms that can deal with the sudden death of the queen, showing that the swarm can survive even when the queen dies [9]. In artificial swarms, both centralized and decentralized designs exist. However, in underwater scenarios having a central decision maker that commands swarm members is impractical due to communication and localization limitations discussed in Sections [1.3.1] and [1.3.2] respectively. For this reason, decentralization is a very important design goal when it comes to underwater swarms. This dissertation
follows a completely decentralized approach in the design of algorithms at all levels of the mission.

1.6.8 Existing Systems

As the field of swarm robotics is relatively new, most of the proposed systems in the field are still within lab environments undergoing continuous development and experimentation. Although ground and aerial swarms have already been developed by some groups, we focus only in this section on swarms that have been specifically designed for underwater use. The list of swarm systems presented here is by no means exhaustive, it is only intended to give an overview of some of the existing underwater swarm robotic systems.

CoCoRo

Collective Cognitive Robots (CoCoRo) is a joint project funded by the European Commission with participants from universities in Austria, Belgium, Germany, Italy, and United Kingdom. One of the main goals of the project is the development of small autonomous underwater vehicles that are aware at the individual level and that utilize innovative bio-inspired algorithms to generate cognition at the global swarm level. The researchers in the project also use an evaluation approach similar to the one used by scientists to study cognition in natural organisms; they use sophisticated experiments to test the swarm and determine its cognition level and how it compares to nature \[15\]. Figure [1-2] shows a picture of the “Lily” AUV platform developed in the project. The picture is taken from the project’s website.
Figure 1-2: CoCoRo Project: Collective Cognitive Robots; the “Lily” platform in action. The image is taken from [75].

Figure 1-3: Serafina platform. The image is taken from [102].

Serafina

Serafina is another project conducted by a team of researchers at the Australian National University. The project targets the development of small, organized AUVs that can form a scalable, robust, and fully autonomous swarm. Some of the main focus areas of the group are on relative and global localization, spontaneous communication with neighbors and routing abilities, and consideration of vehicle constraints and sensor set optimization. The project tries to close the gaps between theoretical aspects and real-world dynamic environment with its limitations and uncertainty [102]. Figure 1-3 shows the Serafina platform. The picture is taken from the project’s website.
Figure 1-4: SWARMs Project: ROV repairing the issues detected by an AUV in the application of corrosion prevention in offshore installations. The image is taken from [53].

SWARMs

Smart and Networking Underwater Robots in Cooperation Meshes is an industry-led European project with a consortium of 30 partners including large technology companies, small- and medium-sized enterprises, universities, and research institutes located in 10 European countries. Its goal is to develop an integrated platform for next generation underwater missions using standard current generation heterogeneous AUVs, Unmanned Surface Vehicles (USVs), and ROVs that work cooperatively by forming a mesh. The project is application oriented and targets five main applications: 1) Corrosion prevention and detection, 2) Chemical pollution monitoring, 3) Detection, inspection, and tracking of plumes, 4) Berm building, and 5) Seabed mapping. Some of the other goals of the project include: increasing the autonomy of used AUVs and the use of ROVs in complex tasks, developing languages for inter-vehicle cooperation and coordination, developing new communication mechanisms, and coming up with innovative decision making and environment perception techniques for AUVs [53]. Figure 1-4 shows an illustration of the use of a heterogeneous swarm in corrosion prevention in offshore installations. The picture is taken from the project’s website.
1.7 Contributions

Our main contributions in this dissertation can be summarized as follows: 1) a general framework for underwater mission planning, 2) three novel global search algorithms for unbounded underwater environment, 3) an algorithm for initial self-organization, 4) an optimized same-position reorientation algorithm for use in certain mission stages, 5) three autonomous task allocation algorithms, 6) three local target search algorithms, 7) a measure of mission utility, and 8) the design of a human brain-inspired model to support learning and complete autonomy.

In the next chapter, we provide a literature review of research related to mission planning in general, then we review literature related to concepts used in individual mission stages.
Chapter 2

Literature Review

Relevant research spans multiple areas due to the aggregating nature of the targeted problem. The first area is mission planning research, which targets global design and assessment of robotic-team missions. Other research areas relevant to this work are more focused on the individual stages of a mission like self-organization, search, and task allocation. Recent advances in each of these areas will be presented in the following sections.

2.1 Mission Planning

Although research focused on planning underwater missions from the global perspective is relatively limited, there are some big projects and research groups that have achieved significant progresses in this area. For example, TRIDENT by Sanz et al. [72, 73, 74, 71] is a European project started in 2010 with the objective of designing, developing, and implementing a heterogeneous team of robots for use in intervention tasks in unstructured and uncertain marine environments. The team consists of an Autonomous Surface Craft (ASC) and an Intervention AUV (I-AUV) equipped with a redundant robotic arm and a dexterous hand for manipulation in multipurpose intervention tasks. Within the context
of global mission planning, this project makes prominent contributions as it clearly defines and carefully studies surveying, target localization, and manipulation mission stages. The distinguishing factor between their work the proposed research is that their robotic team can be thought of as a multi-robot system (MRS), while we use robot swarm that has a lower bound on the number of agents used. Additionally, they pay special attention to the integration between the used AUV, arm, and dexterous hand and their cooperative operation. That level of detail is beyond the scope of our work; we deal with manipulation and intervention at an abstract level.

RAUVI (Reconfigurable Autonomous Underwater Vehicle for Intervention Missions) [64, 60] is another project funded by the Spanish Ministry of Research and Innovation that successfully designed and implemented a complete intervention mission involving survey and object retrieval stages and tested it in a semi-realistic scenario (water tank) to retrieve a plane’s black box. The project is focused on a single AUV with manipulation capability.

Kothari et al. [43] study the issues and challenges associated with the design and implementation of real world robust underwater AUV missions. They propose to use a set of maneuvering (motion), sensor, communication, and payload primitives to facilitate the specification of mission plans. By offering these primitives, a plan can be designed either by human operator or automatically using methods like control or constraint problem solving. Coordinated control of vehicle teams is also considered and restrictions on that coordination are highlighted. Their methodology can work hand-in-hand with our proposed mission planning framework to enable realistic mission planning.

Mission planning frameworks for swarms in other environments also exist in the literature, e.g. Unmanned Aerial Vehicles (UAVs) [45, 95] and mobile robots [34, 101].
2.2 Initial Self-Organization

Self-Organization (SO) is a global, collective, emergent behavior that results from the local interactions between neighboring agents by applying simple, decentralized rules [7]. Although the term ‘self-organizing’ was introduced long ago in late 1940’s by cybernetician W. R. Ashby [3], the earliest known flocking model (which allows swarm members to travel in cohesion and synchrony) based on self-organization was introduced by C. Reynolds in 1986 [66, 67]. Several models for attraction-repulsion have since been proposed in the literature; for example, [59, 80] used the Morse Potential flocking model to represent this behavior.

In [33], Hildenbrandt et al. modeled behaviors of starlings in terms of social forces. They defined social force as the sum of separation, attraction (cohesion), and alignment forces following Reynolds’ original model [67].

Although several contributions have been made in the field of self-organization: Self-Organizing aggregation [98], SO shape-formation [37, 70, 41, 93], emergent SO behavior [18, 61], SO navigation [49], and SO structure formation [92, 100], the ones focusing on the initial transition from randomness to order in underwater missions are very few. For example, Sousselier et al. [81] propose a line formation algorithm for a swarm of small Unmanned Underwater Vehicles (UUVs) using emergent behavior based on local positioning along with imprecise global positioning. The algorithm does not take vehicles’ physical constraints into consideration and assumes that vehicles know their own absolute position. In the approach we propose, we account for physical and reorientation constraints and do not assume global position knowledge.
2.3 Target Search

Target search in swarm robotics has benefited from meta-heuristics proposed in different fields for similar usages. Search meta-heuristics can be broadly classified into: stochastic, physics-related, probabilistic, swarm intelligence, neural, immune, evolutionary, and fuzzy logic algorithms. Manjarres et al. [52] provide an extensive overview of these meta-heuristics. Researchers have proposed several search algorithms that are based on these meta-heuristics.

Particle Swarm Optimization (PSO) [21] falls under the category of artificial intelligence algorithms. It has been frequently used in swarm robotics. One of the earlier adaptations of PSO to enable its real-world implementation on robots was proposed by Hereford in [32]. The adapted version was called distributed PSO (dPSO). Physical motion constraints like minimum turning radius and maximum turn angle were taken into consideration in the design. Another goal was to minimize communication between robots to save energy. Simulations showed a percentage of 99-100% target finding when it is near the middle of the search area and an improved performance with lower maximum velocities of the vehicles. In [65], a one-to-one mapping of PSO particles to robots of a swarm was done. The robotic constraints and differences from particles proposed in the standard version of PSO were taken into consideration in the design of the adapted PSO. Two algorithm versions were studied: PSO-inspired algorithms with and without a global positioning. They also accounted for the difference between the continuous-time and discrete-time nature of robots and PSO, respectively, through the inclusion of a fixed amount of robot move time between successive discrete steps of the algorithm.

Couceiro et al. [17] surveyed multi-robot search inspired by swarm intelligence, where five cooperative search algorithms were compared computationally and benchmarked through simulations and real experiments. They showed that evolutionary algorithms usually give
better results than non-evolutionary ones. Robotic Darwinian Particle Swarm Optimization (RDPSO) algorithm \[15, 14\] was also found to outperform the others in terms of convergence speed and handling of multiple dynamic sources, giving evidence that socio-biological inspired algorithms are suitable for search problems that can be formulated as optimization problems.

Two hybrid swarm-fuzzy target-search strategies have been proposed by Venayagamoorthy et al. in \[94\]. The first approach, fuzzified swarm of robots, used a fuzzy term in the canonical (standard) particle swarm optimization, to replace part of PSO dynamics. The second method, swarm-fuzzy controllers, used a swarm of robots running fuzzy controllers. The two strategies were compared to the greedy search approach and shown to surpass it in terms of convergence percentage, time, and number of iterations. The approaches were studied only in two-dimensions. Keeter et al. \[40\] proposed four random walk lévy flight based algorithms for cooperative search in aquatic environments. This included independent, bounded-region, biased angle, and biased jump length sparse target searches. Simulations and experiments showed that these randomized algorithms outperform systematic raster sweep.

### 2.4 Task Allocation

Task allocation can be classified into three main categories from the agent’s perspective \[9\]: Autonomous (independent decisions), heteronomous (dependent decisions), and hybrid (combination) task allocation. Furthermore, autonomous task allocation can involve: rule-based, threshold-based, probabilistic, or decentralized reinforced control. Other classifications exist in the literature, however, we adopt this scheme in this work due to its relevance.

Ducatelle et al. \[20\] studied the problem of task allocation in robotic swarms. Multiple concurrent tasks were considered and two algorithms were proposed and compared in terms of scalability and robustness. The first was light-signaling based while the second used
gossip-based information exchange. Tasks were announced by certain robots and all others assigned themselves to tasks using the proposed algorithms. Gossiping algorithm was shown to perform better for small numbers of robots and for highly cluttered environments, while the two algorithms performed nearly the same for simple environments.

A social-welfare inspired task-allocation approach for multi-robot systems was presented by Kim et al. in [42]. The proposed algorithm was distributed and intended for uncertain dynamic environments. Resource inequality was defined based on Atkinson’s inequality index [4], and Atkinson’s welfare function was adapted to derive resource welfare. Tasks were allocated to individual robots based on the maximization of task completion ratio while minimizing resource (energy) usage. It is worth mentioning that the approach utilized inter-robot communication and, in some cases, tight-coordination. The superiority of the algorithm to a market-based approach was verified through simulations.
Chapter 3

Methodology

This chapter covers the contributions made by this work. First, our mission planning framework is described and its stages are specified. This is followed by an examination of two common design approaches in Swarm Robotics and a description of the design approach followed in this work. Third, we present our simulator which was specifically designed for underwater swarms. Next, we present a simple vehicle orientation algorithm and an optimized one specially designed for same-position reorientation. This is followed by the explanation of a self-organization algorithm for initial transition from chaos to order. Global target search algorithms developed in this work are then explained in details. Task allocation and its developed algorithms are presented afterwards. To assist task allocation, local search algorithms are then presented in the following section. Mission stages integration and mission profit calculation are then discussed. Finally, we give a brief overview of Reinforcement Learning (RL) to serve as the basis for the section to follow in which our brain-inspired platform, MiniBrain, is presented.
3.1 Mission Planning Framework

A typical underwater swarm mission can be thought of as a sequence of indispensable steps wherein, if a step is removed the mission becomes either impossible or very expensive. Because of the nature of robotic swarms, consisting of large numbers of simple robots, there must be a mechanism to properly release the agents from a central location into the water. This initial release must guarantee that agents will be able to form an initial cohesion so that they don’t lose their local connectivity, disperse apart, and possibly divide into disjoint groups. For this reason, an initial swarm release or unpacking stage must exist, where agents are released from a dense-packing state into the target environment and activated. In this work, environment refers to aquatic environments such as oceans, seas, lakes, and rivers. Once the swarm is injected into the environment and agents are activated, an initial self-organization stage should follow. In that stage, agents employ decentralized control algorithms to achieve the previously mentioned cohesion. At the same time, they maintain repulsion forces that prevent the swarm members from colliding with each other. In this same stage, agents can optionally decide, still in a decentralized manner, the global shape that eases transport while maintaining a balance in global energy consumption. We call this stage: initial self-organization. The next stage is path planning, where the swarm will collectively move along a path that achieves certain local constraints and is formed by applying distributed, simple rules. The goal in this stage is to maintain cohesion, move collectively towards a common target, and minimize energy consumption and forward-trip time. When the target is found (more details are presented in Section 3.1.1 below), the fourth stage, self-organization and task accomplishment, starts. In this important stage, the swarm re-organizes itself in a pattern that matches the target and optimizes data collection. This is application-specific, however, we present two target coverage examples below in framework specification. The fifth and sixth stages are, again, self-organization and path planning.
They are counterparts of stages two and three, respectively, but with different constraints and rules as the return trip has a different nature from the forward trip. Finally, if the swarm successfully returns to the base, the last stage would be swarm pickup and recovery. Figure 3-1 shows the above discussed stages. In the following section, we discuss the details of each of these stages.

### 3.1.1 Framework Specification

The four main concepts that mission planning framework relies on are: self-organization, shape-formation, path planning, and task-allocation/division of labor. A large body of literature exists for each of these domains, however, a unified framework where mission planning based on their proper consolidation does not yet exist to the best of our knowledge. The following sections detail each of the seven mission stages using these concepts.
Swarm Release/Unpacking

This stage is application specific and depends on the environment. For example, in ground swarms, agents will start from an initial stationary position based on initial placement; this can be random or according to a specific pattern based on the choice of the designer and the requirements of the design. It should, however, be noted that the correct definition of “swarm” entails that large numbers of agents be used, which makes initial organization of agents according to a specific pattern difficult to accomplish. Another consideration is that the impact of initial configuration on the swarm’s performance should be minimal. This can be achieved if local rules are carefully chosen so that agents can quickly transition from the initial disorganized state into the self-organized, efficient-functionality state. In the case of underwater swarms, the swarm can be either injected into the water on an agent-by-agent basis, giving them the chance to self-organize as they are deployed, or they can be released simultaneously and then allowed to self-organize. The first approach is very slow, especially for large swarms. For this reason, and for a general mission, it is more appropriate to select the second approach of releasing agents simultaneously. A suitable assumption is that agents are initially densely-packed in a container, e.g, cage, and are liberated into the water almost at the same time by opening the container. Stage I in Figure 3-1 shows an example release mechanism.

Initial Self-Organization and Shape-Formation

The purpose of initial self-organization is to maintain a balance between cohesive and repulsive forces among agents as well as achieving some notion of consistency. Using appropriate degree of separation prevents collisions. On the other hand, cohesion saves agents from dispersing away from the swarm and being lost. Several models for attraction-repulsion have been proposed in the literature [66, 3, 59]. For example, [59] and [80] used the Morse Poten-
tial flocking model given by Eq.3.1 to model this behavior.

\[ v_F^i(t + \delta t) = [G_S^i \exp\left(\frac{-r_c(t)}{20}\right) - G_A^i \exp\left(\frac{-r_c(t)}{20}\right)] , \quad (3.1) \]

where \( v_F^i \) is the change in the flocking potential of agent \( i \) for a \( \delta t \) time increment, \( G_S^i \) and \( G_A^i \) are separation and aggregation (cohesion) gains for agent \( i \), respectively, and \( r_c(t) \) is the distance of the closest neighbor in a set of \( n \) neighbors (usually limited; e.g. 4-7) at time \( t \). \( G_S^i \) and \( G_A^i \) are selected based on the desired degree of separation. For example, Oyekan et al. [59] used 1 and 0.99, respectively.

In [33], Hildenbrandt et al. modeled behaviors of starlings in terms of social forces. They defined *social force* as the sum of separation, attraction (cohesion), and alignment forces following Reynolds’ original model [3]. Eq’s.3.2, 3.3, 3.4, and 3.5 show these three forces and their combination to form the social force affecting agent \( i \).

\[ f_s = -\frac{w_s}{|N_i(t)|} \sum_{j \in N_i(t)} g(d_{ij})d_{ij} \quad (3.2) \]

\[ f_c = C(t) \frac{w_c}{|N^*_i(t)|} \sum_{j \in N^*_i(t)} X_{ij}d_{ij} \quad (3.3) \]

\[ f_a = w_a \left( \frac{\sum_{j \in N^*_i(t)} \mathbf{e}_{x_j} - \mathbf{e}_{x_i}}{\left\| \sum_{j \in N^*_i(t)} \mathbf{e}_{x_j} - \mathbf{e}_{x_i} \right\|} \right) \quad (3.4) \]

\[ \mathbf{F}_{Social} = f_s + f_c + f_a \quad (3.5) \]

where \( f_s, f_c, f_a \) are the separation, cohesion, and alignment forces w.r.t. agent \( i \), respectively, in Newtons (N). \( w_s \) is the weighting factor for separation (1N), \( N_i(t) \) neighborhood of agent \( i \) at time \( t \), \( d_{ij} \) is a unit vector in the direction of \( j \) (from \( i \)), \( g(d_{ij}) \) is the halved
Gaussian (see [33] for details). $C_i(t)$ is the degree of centrality of agent $i$ in the group (length of the average vector of the direction towards its neighbors $N_G$), $w_c$ is a weighting factor for cohesion, $N_i^*(t)$ are agents located in the topological neighborhood (reduced neighborhood) of agent $i$, and $X_{ij}$ is an indicator of whether $d_{ij}$ is inside a radius $r_h$ of a hard sphere within which agents are attracted to each other. $w_a$ is a fixed weighting factor for alignment and $e_{xi}, e_{xj}$ are the forward directions for agents $i$ and $j$, respectively.

By applying these forces, a swarm can eventually achieve self-organization and travel in a bird-like flock. In this work, we utilize this mechanism combined with other assistive mechanisms to speed up the desired initial SO.

The second associated concept, shape formation, was extensively studied for the 2D case, but not as much for 3D scenarios. One of the few works in 3D domain is the work of Yeom [100]. He developed a multi-agent based approach for constructing 3D shapes inspired by biological morphogenesis. Cell processes like differential cell-adhesion, gene-regulation, and inter-cell communication were used as a basis for building the model. A genetic encoding scheme for multi-agent robots was also presented. The agents used behavioral and constructional polices to make local decisions and a genetic algorithm-based evolutionary process was used, where fitness of the agents was continuously evaluated using suitable fitness functions. Diffuser and sensor model was used for inter-agent communication and each agent decided its next behavior based on neighbor position information. The target of the approach was dynamically reconfigurable bio-inspired systems, but it can be adapted to the case of shape formation in robotic swarms. This may be suitable for underwater as well as aerial swarms.

Examples of other swarm robotic 3D shape formation approaches include [54, 62]. Many other works covering formation control exist in the literature like [6, 97, 36, 39, 22], however, most of these works study the process in 2D as mentioned before. While they can work well for ground robots, not all of them can be easily extended to the 3D case.
In this work, we do not consider the problem of shape-formation/formation control as it is optional and not required by all missions. This allows us to focus more on essential stages like target search and task allocation.

**Target Search and Path Planning**

Target search is application dependent and is sometimes considered part of the path planning process when target is known as will be shown in some of the example works below. For generality, the case of unknown target location should be considered. This case is very common in search and rescue, and discovery operations. Examples of target search techniques proposed in the literature have already been presented in Section 2.3.

Path planning has been studied in different contexts for ground, aerial, and underwater single and teams of robots. Here, we focus only on underwater path planning. Obstacle avoidance is also usually studied as part of this process as it is pertinent to motion in complex environments. Aghababa [1] proposed five evolutionary algorithms (EAs) for underwater path planning and obstacle avoidance. This included a genetic algorithm (GA), a mimetic algorithm (MA), particle swarm optimization (PSO) and ant colony optimization (ACO) algorithms, and a shuffled frog leaping algorithm (SFLA). Path planning was formulated as a nonlinear optimal control problem (NOCP) and each of these algorithms was used to solve the problem with the goal of minimizing a time-energy cost function. Simple and complex environments containing obstacles and energy sources were considered and results were compared to the conjugate gradient penalty method (CGP) to show the efficiency of the proposed algorithms. It was assumed, however, that the environment and map are known a priori, which may not always be possible.

Cochran et al. [11] explored the use of extremum-seeking in planning the paths of underwater vehicles in 3D. Their approach has the advantage that the path is computed based on
local information (sensed gradient of the chemical or other substance/phenomenon) without location awareness, which is especially useful in underwater environments because of the absence or extreme difficulty of position tracking and identification. Their approach was a generalization of the application of the same technique in 2D. They also considered different vehicles and actuation types as well as static and moving targets.

In a different context, evolutionary robotics was utilized by Sperati et al. in [83] to form shortest paths between a source and a target area as a dynamic chain for a swarm of robots. Individual robots used red and blue LEDs to communicate indirectly, where the form of communication evolved during the chain formation process. The limitation of the approach is that it required a continuous back and forth transport of robots between the source and target areas. For some applications, this is acceptable, but for underwater missions, this may not be the best approach as targeted areas could be deep and energy and time constraints are a main concern. The approach targeted exploration and navigation in unknown environments, but presence of obstacles was not studied.

Taking into consideration the effect of external fields, a two-step algorithm based on level-sets was proposed by Lolla et al. [50]. The goal was to find the time-optimal path between start and final positions for underwater vehicles moving in time-varying flow fields. In the first step, a forward wave-front is evolved from the start to the end location (where the vehicle acts as a marker on that front) and its evolution is tracked until it reaches the destination. In step two, the path of the vehicle is tracked by solving particle tracking equation backward in time from destination to source to determine the path that takes the minimum time; this is the time-optimal path. They showed that the optimal path is governed by Eq.3.6, which when integrated backward in time starting from the final position $x = x_f$, will result a particle trajectory corresponding to the optimal path.
Figure 3-2: Swarm navigation in the presence of time-varying flows - figure follows Figure 1 in [50].

\[
\frac{d\mathbf{x}}{dt} = -\mathbf{V}(\mathbf{x}, t) - F \frac{\nabla \phi(\mathbf{x}, t)}{|\nabla \phi(\mathbf{x}, t)|} \tag{3.6}
\]

\[
\phi(\mathbf{x}) = \begin{cases} 
  d(\mathbf{x}), & \text{if } \mathbf{x} \text{ is outside the front} \\
  -d(\mathbf{x}), & \text{if } \mathbf{x} \text{ is inside the front}
\end{cases} \tag{3.7}
\]

\[
d(\mathbf{x}) = \min_{\mathbf{x}_i} |\mathbf{x} - \mathbf{x}_i|, \text{ for all } \mathbf{x}_i \in \text{front} \tag{3.8}
\]

\[
\frac{\partial \phi(\mathbf{x}, t)}{\partial t} + F|\nabla \phi(\mathbf{x}, t)| + \mathbf{V}(\mathbf{x}, t) \cdot \nabla \phi(\mathbf{x}, t) = 0, \tag{3.9}
\]

where \( \mathbf{x} \in \mathbb{R}^n \) is a position vector in space, \( \mathbf{x}_s \) and \( \mathbf{x}_f \) are the start and final positions, respectively as shown on Figure 3-2. \( \mathbf{V}(\mathbf{x}, t) \) is a time-dependent external velocity field, \( F \) is the vehicle’s nominal speed w.r.t. the velocity field, \( \phi(\mathbf{x}) \) (or \( \phi(\mathbf{x}, t) \)) is the signed distance function; an arbitrarily-selected time-varying scalar field for which the level sets are found, and \( d(\mathbf{x}) \) is the shortest distance from a point \( \mathbf{x} \) in space to the front (see Ref. [50] for more
details). \( \phi(x, t) \) is evolved in the first step of the algorithm using the initial value partial differential equation 3.9.

**Target Identification and Task Accomplishment**

This stage can be further broken into two sub-stages: 1) At-target self organization and target coverage, and 2) Task partitioning and allocation.

When the target has been identified as a result of the search process or direct transport from source to destination (in case of known target location), the next step is to self-organize in order to adequately cover the target, with load balancing and/or division of labor taken into consideration. Target coverage by swarms of robots has been studied by many teams. For example, Rutishauser et al. [68] proposed an algorithm for collaborative coverage by a team of miniature robots based on communication for environments with unknown extensions. Coverage time was shown to decrease linearly with the increase in the number of robots. The algorithm was validated through quantitative analysis, experiments with real miniature robots, and a discrete event simulation (DES), where quantitative and qualitative results matched. In the worst case, it was shown to degrade to the case of independent, random coverage when communication and positional information were affected by noise.

In [77], the multi-agent boundary coverage problem was studied, where robots search a bounded 2D environment for a set of targets and encircle them. The advantage of the approach is its ability to adapt to changes in the environment such as number of targets, and that search and encircling patterns need not to be defined a priori. The drawback is that every point in the environment has to be inspected.

Staňková et al. [84] devised a Stackelberg-games based approach (StaCo) for the multi-robot Voronoi coverage problem. Although the approach does not solve the problem in the context of target coverage, it can be used for that purpose. The proposed scheme consisted
of a heterogeneous team of leader and follower robots, where leaders had more advanced perceptive abilities than followers. The former group selects locations that aid the latter to achieve faster and more efficient convergence when their local objectives are optimized; this takes place without communication. Theoretical analysis and study of different cases showed that StaCo outperforms the classical Lloyd’s algorithm and is similar to it in the worst case. It is worth mentioning that this approach was applied in 2D and its application to complex 3D environments still needs to be investigated.

The second sub-stage is task partitioning, division of labor, and task allocation. Labella et al. [44] implemented and analyzed an ant-foraging inspired algorithm for labor division that used simple, local adaptive rules to guide the behavior of individual robots. The approach emphasized the importance of local learning on the overall behavior and emerging labor division in the group. Agents did not have to communicate and only depended on local adaptivity. The algorithm was employed in an object retrieval task and the analysis showed that communication between agents is not necessary for efficient task accomplishment. Object retrieval efficiency of the group was measured using an efficiency index as in Eq. 3.10:

\[ v = \frac{\text{performance}}{\sum_{\text{robots}} \text{duty time}}, \quad (3.10) \]

where \textit{performance} was defined as the number of retrieved objects and \textit{duty time} is the time spent by each robot searching or retrieving (the time it was on duty). In [63], the effect of communication on task partitioning among robots was studied. The similarity of the problem of deciding whether or not to partition a task with the multi-armed bandit problem was exploited. Proposed solutions to the latter in the field of Reinforcement Learning were shown to be applicable in the case of task partitioning in robotic swarms. The authors used three algorithms from the multi-armed bandit problem’s literature along with a previously
designed ad-hoc algorithm to test the task partitioning behavior of robots in simulation. These algorithms used local cost estimates to guide individual decisions. They were compared to four reference algorithms that did not use estimate values to make decisions. The tests were done for the social (with communication) and non-social (without communication) cases. Results showed that communication can be helpful in making faster local decisions, however, it can cause lower awareness of environment variations.

Examples of works that studied the problem of task allocation in robotic swarms were presented in Section 2.4. The social-welfare inspired task-allocation approach highlighted there (developed in [42]) defined Resource inequality as in Eq. 3.11 based on Atkinson’s inequality index [4]. Atkinson’s welfare function was adapted to derive resource welfare as provided in Eq. 3.12. Tasks were allocated to individual robots based on the maximization of task completion ratio while minimizing energy usage. The superiority of the algorithm to a market-based approach was verified through simulations.

\[ I_R = 1 - \left[ \frac{1}{n} \sum_{i \in R} \left( \frac{R e_i}{\overline{R e}} \right)^{1-\varepsilon} \right] \frac{1}{1-\varepsilon} \]  

(3.11)

\[ W_R = \overline{R e}(1 - I_R) = \left( \frac{1}{n} \sum_{i \in R} R e_i^{1-\varepsilon} \right) \frac{1}{1-\varepsilon}, \]  

(3.12)

where \( n \) is the number of robots, \( R e_i \) is the resource (energy) residual for robot \( i \), \( \overline{R e} \) is the average resource residual of the team of robots, \( \varepsilon \) is the strength of penalty for the inequality (\( \varepsilon \in [0, \infty) \), penalty increases as \( \varepsilon \) increases), and \( R \) the team of robots.

### 3.2 A Holistic Design Approach

In the design of robotic swarms, researchers usually use behavior-based or automatic design approaches [8]. According to Brambilla et al., behavior-based approaches can be further clas-
sified into: \textit{probabilistic finite-state machine}, \textit{virtual physics-based}, and \textit{other} design methods. The main characterizing element of behavior based approaches is that the designer manually designs individual behaviors trying to achieve the collective global behaviors intended for the swarm. Automatic design, on the other hand, can be either \textit{Reinforcement Learning-based}, \textit{evolutionary robotics-based} or \textit{other learning and automatic design} methods that cannot be clearly classified under any of the previous two. The characterizing element of the this approach is that its techniques can automatically generate behavior based on learning and/or evolution, reducing the burden on the designer.

Swarm robotics analysis, is usually done by the use of models \cite{3}, where the two main categories are macroscopic \cite{46, 56, 25, 30, 29} and microscopic \cite{10, 16, 13} models. In macroscopic models, the focus is on the swarm as a whole, while in microscopic models, analysis is done at the individual level, inter-individual interactions, and individual-environment interactions.

In this dissertation, we use inspiration from these design and analysis techniques and follow a holistic/hybrid approach that combines many of the above approaches. The following sections elaborate more on the above classification and combines design and analysis concepts to eventually reach our holistic design approach.

\subsection{Macroscopic Level: the Swarm}

As mentioned in the above introduction, a common category of models for swarm robotics analysis is macroscopic models. These models usually use continuum equations (differential equations and gradients) that neglect the individuality of swarm members and describe the collective behavior of the swarm. They can be thought of as the parallel to virtual physics-based design approaches. This is because latter approaches usually use attraction and repulsion forces and potential fields to achieve a desired behavior, which can still be
described by vector fields and gradients. Therefore, the term *macroscopic* can be used to describe design and analysis approaches that deal with the global collective behavior taking place at the swarm level.

### 3.2.2 Microscopic Level: the Agent

Similar to the above case, automatic design is mainly based on learning and evolution, which usually take place at the agent-level. Because microscopic models focus their attention on agent-level behaviors and on its interaction with its surroundings, the term *microscopic* can be used for these design and analysis approaches.

### 3.2.3 Holistic Approach: the Agent and the Swarm

Due to the importance of the two previously discussed design and analysis levels (which are concerned with the collective swarm-level behavior and individual’s design and behavior), we follow a holistic design approach in this work that pays close attention to designs at both levels. For example, we use a virtual physics-design in which attraction and repulsion forces are used to achieve the desired collective behavior (e.g. flocking). On the other hand we design the internals of agents that enable them to learn from experience and use deterministic and/or probabilistic behaviors (e.g. controller’s FSM and MiniBrain model). We use concepts at both levels like probabilistic finite state machines, social forces, and reinforcement learning to build a more capable swarm.

### 3.3 Underwater SWarm sIMulator (SWIM)

In this section we describe the motivation for building our Underwater SWarm sIMulator (SWIM) and its main packages and functionalities. The simulator was written in Java and
is therefore platform independent. It is a 3D simulator that simulates (and emulates for some components) swarms of underwater robots. For this reason and like any software, it has minimum hardware requirements. These requirements are determined by the underlying game development engine used (jMonkeyEngine [35]) and the physics engine it utilizes (jBullet [38]). All the simulations in this work were run on a Dell Alienware 17 gaming laptop with Intel® Core™ i7 processor, 16GB of RAM, and a NVIDIA® GeForce® GTX 970M graphics card on a Windows 10 platform.

3.3.1 Motivation

There are three main motivations behind the development of SWIM: 1) the desire to have full control over all parameters, simulation modes, and the ability to extend or customize different modules comprising the simulator, 2) the need for a simulator, targeted specifically at underwater swarm robotics, which supports physics and provides a realistic 3D environment, and 3) the need to allow simulation of complete missions by providing extensible and reusable templates and skeleton codes for different mission stages.

3.3.2 Simulator Design

We follow a balanced design approach between simulators that model swarms as point masses that can move omni-directionally and simulators that focus on the internal workings of individual robots and can simulate highly sophisticated operations running internally in the robot. In SWIM, we take physical vehicle constraints into consideration, thus do not model AUVs as point masses but instead as robots with limited degrees of freedom. At the same time a considerable level of intelligence is built into each individual agent to enable autonomous decision making. The simulator is also equipped with numerous debugging capabilities that ease the development to a great extent and also can serve as means for generat-
Figure 3-3: Snapshot of the Complex-Environment mode of SWIM simulator.

ing visually appealing descriptors of different parameters. Fig 3-3 shows a snapshot of the Complex-Environment mode of the simulator. In the next section, high-level organization of the simulator is presented, then the following sections elaborate on specifics of individual components.

Main Components

SWIM was designed with complete underwater swarm mission implementations in mind. It is for this reason that package structure clearly reflects this goal. Figure 3-4 shows organization of SWIM packages. There are eight top-level packages used in SWIM: core, algorithm, sim, physics, test, gui, exception, and util. As can be seen in Figure 3-4, algorithm package has sub-packages for each mission stage (selforg, search, taskalloc, and surfacing) and an integration package that provides algorithms for integration of different stages into a complete mission. It also contains sub-algorithms used by any of these stages like flocking, learning, and motion and a package for vehicle’s motion calibration and a common package
with shared components among some of the other packages.

**core** package contains the core functionality of the simulator: abstract classes specifying the agents, communication channel and messages implementation, networking support (graph theory-based), and some motion-related constructs.

The third and the biggest package set in SWIM is the **sim** package. This is where most of the work takes place. The **sim** package has a group of important classes in **sim.uwswarm**, the most important of which is **UWVehicleControl**. This class along with **sim.Simulator** can be thought of as the heart of SWIM. This can be easily noticed by observing their combined code size (which is around 8000 lines). **UWVehicleControl** is the controller run by every agent in the swarm.

The package **sim.uwswarm.intelli** contains our implementation of the MiniBrain learning and decision making model discussed in Section 3.11. The **brain** package contains brain’s structural components, both biological and artificial, different brain constructs, and function-
alities. Packages: resource and sense connect the brain to AUV’s resources like battery-life and time tracking, and to different senses provided by the sensory input of the vehicle.

swim.physics is used for building and initializing the physics world to enable support of forces, velocity, acceleration, etc. swim.test provides classes for automated test running. The remaining packages (exception, gui, and util) have self-explanatory names.

Simulation Modes

SWIM supports five simulation modes that can be run independently: SelfOrganization-Mode, Search-Mode, TaskAllocation-Mode, Integration-Mode, and GeneralTests-Mode. The first three are used whenever their respective algorithms need to be tested in isolation from the full mission. These modes provide the proper context to apply the associated mission stage. For example, in TaskAllocation-Mode, the vehicles are placed directly near to the target to allow task allocation algorithms to start execution the moment the simulation is run. Integration-Mode uses the selected integration algorithm (BrainlessAllStageIntegration in current implementation) to connect different mission stages and allow for a complete mission test. GeneralTests-Mode was created to allow developers/researchers to test incomplete algorithms or do any tests that do not necessarily lie under any of the above categories.

Additionally, SWIM has two environment-complexity modes: Complex-Environment, and Simple-Platform modes. The first builds an environment that is more realistic (like the one shown in Figure 3-3): it uses a peaky terrain for sea-floor, and realistically looking water with waves that can have adjustable height and direction and gives the underwater effect when the camera is moved below the sea surface. It also has waves sound effect that makes the scene more natural. This mode is however not always the best option, especially when large swarm sizes are used or the algorithms being tested are computationally intensive.
The second mode abstracts many of these effects away and only uses a simple flat platform for tests. Although it does not closely resemble a real underwater environment, it retains physics support like the previous mode. The main differences are: appearance, terrain flatness, and speed of simulations. This mode was used for all the tests in this work due to its significantly lower load on the processor and allowance for better allocation of computational resources. It is worth mentioning though that in that mode, the AUVs still move in the 3D water column which is clearly not as simple as motion is 2D.

Bulk Automated Tests

In addition to the default tests that can be run in each simulation mode, there is a TestRunner class in the swim.test package that can be used for bulk automated tests. This class runs in the main thread and repeatedly creates instances of another thread (SimRunner) that runs the selected simulation. After creating every instance, it waits until its done doing it job and then starts a new thread. This enables running the same simulation multiple times for validation and result averaging purposes. After each run a set of text files containing the results for different measures are generated and numbered by the run’s number. Further processing of these files can then be done by the researcher.

Strengths and Limitations

Like any simulator, SWIM has strengths and limitations. However, we believe that its strengths outweigh its shortcomings. Additionally, being developed in Java and built in a modular, easily expandable way, there is a big potential for improvements by the underwater swarm robotics community. We begin with the limitations:

- Swarm scalability is limited as each swarm agent runs a controller that is computationally intensive
• Data-structures are not currently optimized for best performance

• Multi-threading is not utilized to its fullest. Use of per-agent threads and their effect on overall performance need to be considered

• The effect of water currents is not currently considered

• Collisions and collision avoidance are not currently supported

On the bright side, SWIM has many features that can greatly assist researchers in adding their own algorithms, conducting experiments, and comparing different techniques. These features are listed below:

• Supports simple communication and graph theory-based networking

• Provides a 3D environment for simulating underwater swarms

• Provides physics support through jMonkeyEngine’s underlying physics engine

• Has numerous debugging capabilities that significantly reduce development and test efforts

• Allows researchers to program internals of the controller used by swarm agents

• Has a modular design that allows easy extension

• Has a simple structure that ramps the learning curve

• Provides manual and automated test modes, simple and complex test environments, and modes for individual and combined mission stages

• Takes into consideration physical vehicle constraints

• Emulates sensors and many other components
Vehicle and Controller Design

The used vehicle model follows the physical appearance of Serafina AUV [102]. It is equipped with five range sensors that can detect nearby vehicles, proximity to the seabed and surface, and the targets being searched. These sensors are directional with their main lobes modeled as spherical cones of adjustable resolution. They point in the vehicle’s northern, southern, eastern, western, and downward directions as shown in Figure 3-5.

AUV’s controller runs a Finite State Machine (FSM) to select proper actions based on sensor input data, built-in logic, and learning models. The FSM consists of four main states (Explore, Update, Turn, and Do Task) in addition to an initialization state as shown in Figure 3-6.

The controller runs in a loop where, in each iteration, the state of the FSM is updated based on the current values of different system variables. When the vehicle is released into the water, it starts in the move-sense-process cycle of the Explore state. This allows it to start scene exploration and sensor data acquisition and processing. When the next iteration of the controller’s update loop is reached, the vehicle’s state automatically transitions to
the *Update* state which has been added to the FSM to allow search and other algorithms’ updates to modify FSM’s behavior. It is a general state used as a container that can contain any relevant updates from the algorithm(s) being used. In the case of search algorithms for example, abstracting the search process in the form *straight-line travels* and *turns*, the expected outcome of the *Update* state is whether to turn or continue exploring (the regular linear motion). This combination of straight-line travels and turns constitutes the series of paths the vehicle has to take to eventually reach the target. Therefore, if it is determined by the update of the search algorithm that a turn is needed, the FSM changes state to the *Turn* state. This is where simple or composite turns can take place. On the other hand, if it is determined that no turn is needed, the FSM goes back to the *Explore* state. In the turn state, a sequence of rotations (yaws) and *translations* (*translations* are used here to describe small location changes between the increments of discretization of continuous turns, *travels* are used to describe pure straight-line location changes) limited by the vehicle’s Minimum
Turn Radius (MTR), Maximum Turn Angle (MTA), and minimum speed, are performed (cf. Section 3.4 for definitions of these terms). The sequence may also contain sections where straight-line travel is necessary. When the turn sequence has been completed, the FSM goes back to the Update state. It is important to notice that the turn sequence can take multiple controller-loop iterations to complete. This means that more than one controller update may be required before the search algorithm’s next update can be utilized. The FSM continues execution in this same manner until the target is found within the internal cycle of the Explore state. At that point, the FSM transitions to the Do Task state. This state is where most of the operations related to task allocation take place. With the help of the Update state, updates coming from the task allocation algorithm can be put into action.

A sixth state, Surfacing or Source Search, can be added to the FSM shown in Figure 3-6 to indicate the surfacing or source search process that follows the task execution stage. It has been omitted here for simplicity and can be though of as part of the Do Task state.

### 3.4 Vehicle Reorientation

Simply simulating an AUV as a particle that uses omni-directional motion is not realistic. AUVs in general have constrained maneuverability and can only execute turns with a maximum turn angle (MTA) and an associated minimum turn radius (MTR). Minimum turn radius can be defined in terms of the maximum turn angle as the radius of the minimum circle the vehicle can travel on the circumference of when moving at the slowest speed and using the Maximum possible, nearly instantaneous, Turn Angle. Figure 3-7 illustrates these parameters with respect to an AUV. These two parameters, among others, determine how sharp a vehicle can turn in a certain amount of time. In the design of our simulator careful attention was paid to vehicle orientation algorithms in order to account for these constraints.
In the next two subsections, we present a simple reorientation algorithm that achieves a desired AUV orientation with minimal required effort, and an optimized reorientation algorithm that is targeted at usages where it is necessary to orient/reorient the vehicle while maintaining its original position.

### 3.4.1 Simple Reorientation

This orientation/reorientation algorithm is a straightforward implementation of turning, taking into consideration the above discussed vehicle limitations. As can be seen from Figure 3-7, the desired turn angle $\theta$ can be a greater than MTA. This means that achieving this desired turn cannot be done in one step. Additionally, $\theta$ is not necessarily an integer multiple of $\phi/2$. Therefore, $\theta$ needs to be approximated by the nearest multiple of MTA. We call this the quantization error in this dissertation. It should be obvious from the figure, using simple trigonometry, that new orientation of the vehicle will be rotated $\theta \pm \delta$ degrees from the original heading, where $\delta$ is the quantization error.

- **Reversing Direction**: due to the minimum turn radius and maximum turn angle constraints, reversing direction will result an arc of length $\pi r_{\text{min}}$ and a displacement of $2r_{\text{min}}$ or $d_{\text{min}}$ (from original location) assuming that the vehicle only executes turns at
its lowest speed, where $r_{\text{min}}$ is the vehicle’s minimum turn radius. This is illustrated by Path A in Figure 3-8.

- **Turning Right & Left:** similar to reversing direction, these turns will result an arc and displacement, but this time the arc has length $(\pi/2)r_{\text{min}}$. The resulting displacement is $r_{\text{min}}$ as shown by Path B.

- **Orientations in the 1st and 2nd Quadrants:** any turn angle lying in the first and second quadrants will result an arc of length $\theta r_{\text{min}}$ as represented by Path C in Figure 3-8. The vehicle will be displaced by a distance $2r_{\text{min}}\sin(\theta_1/2) = r_{\text{min}}\sqrt{2 - 2\cos \theta_1}$ (length of the chord) from its original position.

- **Orientations in the 3rd and 4th Quadrants:** turning in a direction lying in these two quadrants will result a path with arc length $\theta_2 r_{\text{min}}$. Similar to the previous case,
the resulting displacement will be $r_{\text{min}}\sqrt{2 - 2\cos\theta_2}$. This is also shown by Path D in Figure 3-8.

The downside of this kind of orientation is that it is accompanied by a displacement. In some cases, this displacement does not represent a problem, especially when the MTR is negligible compared to the sensing range. In other cases, however, this displacement can lead to cumulative error that can significantly affect the goal of reorientation. For the latter reason, we propose an optimized reorientation algorithm in the next section that avoids this displacement.

### 3.4.2 Optimized Same-Position Reorientation

Inspired by close-packing of spheres and how it results minimum occupation of volume, we devised a reorientation algorithm that achieves same-position reorientation with minimal traveled distance. We present the algorithm in this section and provide the proof of its optimality.

**Concept**

The concept of the algorithm is illustrated in Figure 3-9. One can think of the illustrated circles as two spheres (the circle with the dashed perimeter represents the first sphere and the lightly colored circle represents the second sphere). The vertical axis can be thought of as a wall extending from the origin only in the upward (positive) direction. The first sphere has a joint fixing it to the origin but is free to move around it to the left and right. The second sphere can only move up, down or left (around the bottom of the wall the the first sphere), and always has to touch the wall. The first sphere cannot turn clockwise through the wall, it can only start on left as shown and turn to the opposite position on the right around the bottom of the wall. As the first ball moves around the origin from the far left to
Figure 3-9: Concept of the optimized reorientation algorithm.

the far right, it pushes the second ball upwards. The two balls always tough each other; if the
the first turns left around the joint, the second will follow it downwards turning around the
bottom of the wall if necessary. The vehicle is always located at the origin and its heading is
in the positive vertical direction. The orientation/reorientation path can be thought of as a
string attached to the origin, going upwards, then around the second sphere in the clockwise
direction, in-between the spheres back to the origin, and finally, straight away from the
departure point tangential to the first sphere.

In the following section, we provide analysis of the different paths and show that the
resulting paths will always have the shortest length given the MTR and MTA constraints.
**Theory**

We start by pointing out that for a turning-constrained vehicle with a minimum turn radius, if the vehicle were to try to do a full \((2\pi)\) in-place rotation, that would result a circle with perimeter \(2\pi r_{min}\), where \(r_{min}\) is the vehicle’s minimum turn radius. Therefore, our goal is to ensure that any additional traveled distance (when reorienting in any other direction) is kept at a minimum increase above that unavoidable reorientation minimum. In all the analysis in this section, the AUV will be assumed to be located at the origin (start of an illustrated path). AUVs have been removed from figures for clarity of illustration.

The top-left path in Figure [3-10] shows that if an AUV tries to reorient into the left (right) direction, it may elect to travel a distance equal to \(r_{min}\) both at the start and end of its turning path, replacing one of the four \(\pi r_{min}/2\) arcs in the above mentioned unavoidable minimum turn path with a straight distance of length \(2r_{min} = d_{min}\). Simple analysis shows that the resulting path length will be \(6.712r_{min}\) as shown in figure. However, using the concept of close-packing of spheres (as in the lower-left path in figure), analysis (done next in this section) shows that the resulting path length is \(6.4916r_{min}\), which is shorter than the path with the two straight arms (two-armed path, hereafter). The top-right path in the same figure is an example of a path that achieves the goal of reorienting left but does not satisfy the constraint of having the new orientation at the original position.

Next, we demonstrate that under the MTR constraint, any two-armed path with straight start and end line segments will be longer than the proposed general path that starts with a straight line segment and ends with a curve following the surface of the first close-packing sphere through the origin.

To fully characterize the general reorientation path used in this algorithm, three lengths need to be found as shown in Figure [3-11]: \(b\), the length of the start line segment, the length of the arc \(\overline{ADB}\) on the second sphere (facing the angle \(2\pi - \alpha\)), and the length of the arc
\[ \frac{\theta}{6}, \alpha = \frac{\pi}{3} \]

Total length
\[ = 2r_{\text{min}} + 1.5\pi r_{\text{min}} \]
\[ = 6.712r_{\text{min}} \]

\[ a^2 = r_{\text{min}}^2 + (2r_{\text{min}})^2 - 2r_{\text{min}}(2r_{\text{min}}) \cos \theta \]
\[ = r_{\text{min}}^2 + 4r_{\text{min}}^2 - 4r_{\text{min}}^2 \cos \theta \]
\[ = (5 - 4 \cos \theta)r_{\text{min}}^2 \]
\[ a = (\sqrt{5 - 4 \cos \theta})r_{\text{min}} \]

(3.13)
Figure 3-11: Finding generalized length of the reorientation path.

\[ b = \sqrt{a^2 - r_{min}^2} \]
\[ = \sqrt{(5 - 4 \cos \theta)r_{min}^2 - r_{min}^2} \]
\[ = 2(\sqrt{1 - \cos \theta})r_{min} \]
\[ \therefore b = 2\sqrt{2}r_{min} \sin \frac{\theta}{2} \] (3.14)

To find the length of \( \widehat{ADB} \), we first need to find \( \alpha \), which is the sum of \( \delta \) and \( \omega \) as given by Eq. 3.15. \( \delta \) and \( \omega \) can be easily found from trigonometric relations as before.
$\alpha = \delta + \omega$

$= \cos^{-1} \frac{r_{\text{min}}}{a} + \cos^{-1} \frac{3r_{\text{min}}^2 + a^2}{4ar_{\text{min}}}$

$= \cos^{-1} \frac{1}{\sqrt{5 - 4 \cos \theta}} + \cos^{-1} \frac{2 - \cos \theta}{\sqrt{5 - 4 \cos \theta}}$  \hfill (3.15)

The length of arc $\overline{ADB}$ can now be found in terms of $r_{\text{min}}$. Arc $\overline{BC}$ can be directly found as a function of $\theta$. Length of the general reorientation path can now be found as given in Eq. 3.16 as a function of $\alpha$, $\theta$ and $r_{\text{min}}$.

$$l_{p1} = b + (2\pi - \alpha)r_{\text{min}} + \theta r_{\text{min}}$$

$$= \left[2\sqrt{2}\sin \frac{\theta}{2} + 2\pi - \alpha + \theta \right]r_{\text{min}}$$  \hfill (3.16)

It is also worth noting that the angle $\beta$ (with the vertical axis) of the target direction is directly related to $\alpha$ and $\theta$ as given by Eq. 3.17. As noticed from Eq. 3.15 $\alpha$ is a function of $\theta$. The two angles change in the interval $[0, \pi/2]$. $\alpha$ grows at around double the rate of $\theta$’s growth for lower values of $\theta$ and slows down as $\theta$ gets bigger as shown in Figure 3-12. The figure also shows the change of $\beta$ with $\theta$. It can be easily seen that as $\alpha$ and $\theta$ approach $\pi/2$ degrees each, $\beta$ will approach $\pi$. This can also be seen in Fig 3-13, however, the path shown in that figure is not optimal as in that special case, there is a shorter path defined by a start and end curves as will be seen later in this section. For all other cases, the resulting path will be the shortest as there is no shorter two-curve alternative (for the start and end parts of the path) and also the two-armed alternative with straight line segments will always be longer as shown next.
\[ \beta = \alpha + \theta \quad (3.17) \]

Now, we find path length for the two-armed alternative path, which uses start and end
Figure 3-14: Deriving the general equations for the two-armed path with straight start and end line segments. Figure 3-14 shows a two-armed path as well as dashed spheres used for obtaining the previous path for comparison. To find the length of this path, length \( k \) of arc \( ADB \) and one of the two arms \((m)\) need to be found. \( m \) can be easily found from the geometry of the figure and is given by Eq. 3.18. The length of the arc is found from the angle \( \mu \) which is provided in Eq. 3.19. Using these lengths, path \( l_{p2} \) can be found as a function of \( \beta \) and \( r_{\text{min}} \) as given in Eq. 3.20.

\[
m = r_{\text{min}} \tan \varphi = \frac{r_{\text{min}}}{\tan \frac{\pi - \beta}{2}}
\]  

(3.18)
\[ \mu = \pi + 2\varphi \]  

\[ l_{p_2} = 2m + k = \frac{2r_{\text{min}}}{\tan\left(\frac{\pi - \beta}{2}\right)} + (\pi + 2\varphi)r_{\text{min}} \]

\[ = \frac{2r_{\text{min}}}{\tan\left(\frac{\pi - \beta}{2}\right)} + 2\pi r_{\text{min}} - \beta r_{\text{min}} \]

\[ \therefore l_{p_2} = \left[ \frac{2}{\tan\left(\frac{\pi - \beta}{2}\right)} + 2\pi - \beta \right]r_{\text{min}} \]  

Eqs. 3.16 and 3.20 describe the lengths of the two paths being analyzed in terms of either \( \theta \) (in case of \( l_{p_1} \)) or \( \beta \) (for \( l_{p_2} \)). By observing from Figures 3-12 and 3-13 and Eq. 3.17 that a change in \( \theta \) in the range \( [\pi/6, \pi/2] \) is equivalent to a change in \( \beta \) in the range \( [\pi/2, \pi] \), we can use path equations to draw the change in path length with change in \( \beta \). Figure 3-15 shows log\((l_{p_1})\) and log\((l_{p_2})\) versus \( \beta \) for \( \beta \in [\pi/2, \pi] \). As seen from figure, \( l_{p_1} \) grows linearly while \( l_{p_2} \) grows polynomially. The same result can be obtained for reorientations in the first and second quadrants by considering the change in path length for \( \theta \in [0, \pi/6] \) and \( \beta \in [0, \pi/2] \).

As previously pointed, the only case where the proposed general path \( l_{p_1} \) will not be the shortest is when reorienting backwards as shown in Figure 3-16. In that case, because the top (second) sphere can actually break the wall and find a better close-packing alternative, the path can be made shorter. This is possible because the goal direction (backward) is the only direction that can be obtained with starting and ending arcs without breaking the minimum turn radius constraint as shown in the bottom path of Figure 3-16. This can be further noticed by looking at the bottom-right path in Figure 3-17 which shows that an
Figure 3-15: Change of path length with $\beta$ for the optimal path ($l_{p_1}$) and the two-armed path ($l_{p_2}$).

An attempt to use this same path for any other reorientation direction will violate the MTR constraint. The same figure shows examples of the use of the proposed generalized path in orienting the vehicle in arbitrary directions.

The above analysis shows that paths of the proposed reorientation algorithm will always be shorter than the alternative two-armed path with straight-line start and end (when the latter is possible). Given that the only possible shorter path than the two-armed one is the proposed path as governed by the close-packing properties, and given the minimum turn radius that prohibits generating shorter curves than the ones used by the proposed path, we conclude that the proposed path is the shortest. □

3.5 Initial Self-Organization

The goal in this stage is to enable the initially randomly oriented AUVs to synchronize their heading after release in preparation for the next target search stage. This step is important to prevent the AUVs from dispersing and getting lost in the ocean if the search algorithm itself does not have a built-in self-syncing mechanism. A decentralized self-organization
algorithm that uses only sensing capabilities to achieve the initial SO goal was developed. The algorithm is based on a set of simple speed change rules, Reynolds’ flocking rules [66], the simple vehicle reorientation algorithm presented in Section 3.4.1, and a common notion of orientation (magnetic north, determined by an electronic navigation compass, with a degree of error). Reynolds’ flocking rules are: attraction to, repulsion from, and alignment with flockmates (see Figure 3-18 as presented earlier in Section 3.1.1). The basic idea is to sense magnetic north direction, reorient towards that direction, reduce speed to minimum if the vehicle was already heading in north direction or increase the speed with a percentage proportional to the length of reorientation path, then enter a speed synchronization stage, and finally, execute Reynolds flocking rules to force the swarm to travel in cohesion.
3.5.1 Algorithm

Pseudo code of the algorithm is shown in Algorithm 1. The algorithm starts by estimating magnetic north direction (e.g. using an Inertial Measurement Unit IMU), which will have some error percentage. Maximum reorientation path length is then found using information about the AUV’s minimum turn radius (MTR) and the worst case reorientation a vehicle may need to make (line 2). Speed synchronization period is then set to a predefined value (line 3) that allows the AUVs enough time to synchronize their speed after reorientation towards magnetic north. This value can be calculated using AUV’s dynamic range \( \frac{v_{\text{max}}}{v_{\text{min}}} \), maximum reorientation path length, and the side length of the AUVs drop-off area. Length
of path needed to reorient the AUV towards the measured magnetic north is then calculated on line 4. Next, the vehicle is reoriented using the simple orientation algorithm presented in Section 3.4.1 towards the estimated magnetic north (line 5). The ratio of the vehicle’s reorientation path length and the maximum path length (line 6) is then used as the argument of the normalized tunable half sigmoid function [19], described by Eq. 3.21 and plotted in Figure 3-19 to determine the percent of full vehicle speed to be used by the vehicle. The constant $k$ determines the concavity/convexity of the function and a value of 0.01 was used to draw the curve in Figure 3-19 and in the simulations. The purpose of using this function is to give the AUVs that were originally oriented away from magnetic north a larger speed, while giving the ones that were oriented in a direction close to magnetic north a much slower speed. This is done because the former AUVs take a longer path to reach their final (approximate) magnetic north orientation than the latter ones. By delaying the latter AUVs, the others can catch up and maintain cohesion with them. This serves as the first mechanism to allow speed synchronization among vehicles that are originally oriented differently. On line 13, speed synchronization start time is stored and syncing starts on line 14. To further allow the AUVs to match their speeds, they use their north and south sensors to detect vehicles that are ahead and behind, respectively (lines 15 and 16). Using this information, vehicles that sense no front or back neighbors and the ones that sense both travel at medium speed (line 17 and 18). The ones that only sense back neighbors reduce speed to minimum, and the ones

Figure 3-18: Reynolds’ flocking (Image taken from: http://www.red3d.com/cwr/boids/).
that only sense front neighbors increase speed to maximum (lines 19-24). Because this speed changes occurs every iteration during the speed synchronization period, AUVs eventually succeed to synchronize their speeds. When the speed synchronization period ends, vehicles set their speeds to the same medium speed and start applying Reynolds’ flocking to maintain unity (lines 26-28). Figure 3-20 shows a snapshot taken from SWIM simulator for a group of 100 AUVs running the initial self-organization algorithm. It can be noticed that there are some outliers. For the purposes of this work these outliers are usually brought back to the swarm by the effect of the global search algorithm that directly follows initial SO. We leave algorithm tuning and improvement as future work.

\[ F(P) = \frac{kP}{k - P + 1} \]  

\[ (3.21) \]
Algorithm 1 Initial Self-Organization Algorithm

1: $D_{MN} \leftarrow \text{EstimateMagNorthDirection}()$
2: $L_{\text{max}} \leftarrow \text{CalculateMaxReorientPathLength}()$
3: $T_{\text{syn}} \leftarrow \text{predefined}_\text{sync}_\text{period}_\text{length}$
4: $L_o \leftarrow \text{CalculateReorientPathLength}(D_{MN})$
5: ReorientTowardMagNorth($D_{MN}$)
6: $P \leftarrow L_o / L_{\text{max}}$
7: $r_{fs} \leftarrow \text{NormalizedTunableHalfSigmoid}(x)$
8: if $P \neq 0$ then
9:     vehicle_speed $\leftarrow \text{GetPercentOfFullSpeed}(r_{fs})$
10: else
11:     vehicle_speed $\leftarrow v_{\text{min}}$
12: end if
13: $t_{\text{start}} \leftarrow \text{CurrentTime}()$
14: while $\text{CurrentTime}() - t_{\text{start}} \leq T_{\text{syn}}$ do
15:     north_neighbors_count $\leftarrow \text{GetNorthNeighCount}()$
16:     south_neighbors_count $\leftarrow \text{GetSouthNeighCount}()$
17:     if (north_neighbors_count $= 0$ and south_neighbors_count $= 0$) 
18:         or (north_neighbors_count $\neq 0$ and south_neighbors_count $\neq 0$) then
19:         vehicle_speed $\leftarrow$ mid_speed
20:     else if north_neighbors_count $= 0$ then
21:         vehicle_speed $\leftarrow$ min_speed
22:     else if south_neighbors_count $= 0$ then
23:         vehicle_speed $\leftarrow$ max_speed
24:     end if
25:     travelAtSpeed(vehicle_speed)
26: end while
27: vehicle_speed $\leftarrow$ mid_speed
28: UpdateVehicleSpeed(vehicle_speed)
29: StartReynoldsFlocking()
3.6 Global Target search

In this section, six global search algorithms are presented: three novel algorithms, an adapted version of PSO called R-PSO, and two simple classical algorithms. The latter two are only included for comparison purposes. We start with a brief introduction to PSO and how it was adapted for use on physically constrained AUVs. Next, we explain Virtual Tether Search (VTS) and Constrained Spiral Flocking (CSF). Before presenting our third algorithm, Swirling Divided Hexagonal Close Packing (SDHCP), we first present Simple Sweeping (SSW), one of the two classical algorithms mentioned above, on which we build to obtain SDHCP. We conclude this section with Simple Random Walk (SRW), which is a very well-known naive algorithm.
3.6.1 Robotic PSO (R-PSO)

Particle Swarm Optimization is widely accepted as a distributed optimization technique. It is a simple algorithm that uses two update rules to evolve the velocity and position of each “particle” in the swarm. The particles are initially randomly placed in the search space, and then allowed to “fly” throughout the space to perform the search. Each particle uses three pieces of information to update its velocity: its current velocity, the position of the its best known solution, and the position of the best known solution in the neighborhood (or the whole swarm, in the global version of the algorithm). To retrieve the neighborhood best solution, communication is required. Solution here refers to the value of the function being optimized at a specific position. For example, if the particles are moving on the surface of a 2D function that has peaks and nadirs, and the goal is to find the global maximum, a particle’s best solution is the one that has the highest function value found so far. In the context of a robotic swarm, a solution is the value of a sensed gradient, e.g. of chemical concentration.

Each of the three components used in velocity update has a weight associated with it as follows: current velocity is weighted by inertia weight, self-best component by cognitive weight, and neighborhood-best component by social weight. The second and third components are also multiplied by random vectors to account for individual differences and provide local minima escape mechanisms. Velocities and positions of the particles are updated at each step of the algorithm. Once the velocity is updated, particle’s position can be easily changed using the new velocity. Eq. 3.22 and Eq. 3.23 show the two update rules.

\begin{align*}
v_{i+1} &= wv_i + r_1 c_1 (p_p - p_i) + r_2 c_2 (p_l - p_i) \tag{3.22} \\
p_{i+1} &= p_i + v_{i+1} \tag{3.23}
\end{align*}
where $w$ is the inertia weight, $c_1$ and $c_2$ are the cognitive and social weights, respectively, $r_1$ and $r_2$ and the two random vectors, $p_p$ is particle’s position of the best solution it found so far, $p_l$ is the position of best solution found in the neighborhood, and $p_i$ and $v_i$ are the current position and velocity, respectively.

**Vehicle Constraints**

AUVs use thrusters as their transport mechanism. Position and orientation of these thrusters control the vehicle’s ability to move and turn. As opposed to point masses, vehicles have volumes and a limitation on the maximum turn angle; the angle by which the vehicle can turn within a specific period of time. As PSO was not initially intended for use on real robots, these constraints make its direction implementation on vehicles unrealistic. To enable its use by physical robots, several constraints have to be taken into account. These constraints have been highlighted in a previous work [90] and are briefly discussed in the next section.

**PSO-Finite State Machine Integration**

Although it is natural that PSO updates and vehicle controller updates should be synchronized, outcome of the search process may be unpredictable if care is not taken in designing the vehicle’s FSM and integrating it with PSO. Therefore, we focus in this section on PSO-FSM integration.

In the original PSO algorithm, every particle’s velocity and position are updated at each time step. This update takes into account the effects of particle’s current velocity, its history of the best solution it has found, and the best solution that has been found in the neighborhood (in neighborhood version). This assumes that: 1) Velocity and position updates are instantaneous, 2) The three components mentioned above are still valid by the time the update can be applied. On real robots and AUVs, these two assumptions don’t hold. The
first is not possible because of the physical constraints imposed on the AUV’s ability to change velocity and position within a specific time frame. These constraints also lead to the invalidation of the second assumption: if PSO updates continue to be calculated while the AUV is still applying an update, new values can cause confusion or be stale by the time it is possible to apply them. One option is to drop these updates until current update finishes execution. Another possible solution is to keep track of these updates and cancel current update if a significantly better one is found. In the second case, however, there is a possibility that performance will degrade if the interruption rate is high.

In order to address these concerns, we execute PSO updates as a subprocess in the Update state of the vehicle’s FSM and elect to use the second approach of blocking PSO updates when a turn sequence (update execution) is in progress. The details of this process are shown in Figure [3-21]. It starts by retrieving vehicle’s best position from the best positions buffer and broadcasting it to current neighbors. We keep a history of the last 20 positions/solutions and use them to find the best solution. Next, vehicle’s position and velocity are estimated, followed by a calculation of self-best update component. Neighbors’ solutions are then retrieved to pick the best. Vehicle’s best and neighbors’ best are then used along with estimated velocity to calculate the velocity update. If greater than a predefined threshold $T$, the update is put into effect by triggering a FSM change to the Turn state. Otherwise, the vehicle continues exploration. As mentioned above, during the execution of an update, the encompassing Update state prevents this subprocess from running.

Algorithm

Pseudo code of Robotic PSO (R-PSO) is provided in Algorithm 2. The algorithm runs in controller’s loop of the AUV indicated here by the loop on lines 1 and 32. On line 2, self-best position is retrieved and broadcast at all times. Next, a check is made of whether
the target has been found (lines 3-6). If that is the case, the AUV stays at target and skips the rest of the algorithm. When executing an update, the next turn sequence used to realize the update is executed and the rest of the algorithm is, again, skipped (lines 7-10). If the target is not around and no active update is taking place, the process described in Figure 3-21 is executed: current velocity and position are estimated (lines 11 and 12), AUV’s best position is retrieved from positions buffer (line 13), best position pointer is calculated based on self-best and current positions (lines 14-18), a random vector is generated (line 19), self-best vehicle update component is found using cognitive weight, the random vector, and self-best pointer (line 20), and a similar set of steps are used for neighborhood-best component (lines 21-28). Self-best and neighborhood-best components are then combined with weighted current velocity to find the AUV’s new velocity on line 29. The new velocity is then used by the AUV given it exceeds the threshold described in Figure 3-21 (line 30). Finally, a flag is set to indicate that an update is taking place to block next updates (line 31).

Table 3.1 lists the values used in the simulation for the three weights used in the algorithm.
Table 3.1: Robotic PSO algorithm parameter values.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>Inertia weight</td>
<td>vehicle’s motion inertia weight</td>
<td>0.8</td>
</tr>
<tr>
<td>$c_1$</td>
<td>Cognitive weight</td>
<td>vehicle’s cognition (sensed gradient)</td>
<td>0.2</td>
</tr>
<tr>
<td>$c_2$</td>
<td>Social weight</td>
<td>social effect on perception</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Algorithm 2 Robotic Particle Swarm Optimization

(* A physically constrained version of standard PSO *)

1: loop
2: BroadcastSelfBestPos()
3: if target_found then
4:   StayAtTarget()
5:   continue
6: end if
7: if TurnInProgress() then
8:   ExecuteNextTurnSubSequence()
9:   continue
10: end if
11: $v_i \leftarrow$ CurrVehVelocity()
12: $x_i \leftarrow$ CurrVehPosition()
13: $p_i \leftarrow$ VehBestPosition()
14: if NotSet($p_i$) then
15:   best_pos_pointer $\leftarrow$ 0
16: else
17:   best_pos_pointer $\leftarrow$ $p_i - x_i$
18: end if
19: $r_1 \leftarrow$ GenerateRandVector()
20: $P_{best} \leftarrow c_1 \times r_1 \times best\_pos\_pointer$
21: $p_i \leftarrow$ NeighBestPosition()
22: if NotSet($p_i$) then
23:   neigh_best_pos_pointer $\leftarrow$ 0
24: else
25:   neigh_best_pos_pointer $\leftarrow$ $p_i - x_i$
26: end if
27: $r_2 \leftarrow$ GenerateRandVector()
28: $L_{best} \leftarrow c_2 \times r_2 \times neigh\_best\_pos\_pointer$
29: $V_{k+1} \leftarrow w \times V_k + P_{best} + L_{best}$
30: ReorientAUV($V_{k+1}$)
31: NotifyTurnInProgress()
32: end loop
3.6.2 Virtual Tether Search (VTS)

R-PSO has many requirements that limit its use. These include: a) the need for a localization technique to allow for the determination of self and neighbor positions, b) the need for a gradient to be sensed for finding the target (with the possible loss of the AUV if it does not initially sense the gradient), and c) the need for communication. It also suffers from the possibility of being trapped in local minima, which requires additional mechanisms to prevent this from happening. These limitations can be addressed by our proposed Virtual Tether Search (VTS) algorithm as will be explained in the next sections. We start by presenting the basic idea followed by a more formal explanation of its theory, and end by presenting the algorithm.

Concept

The basic idea is similar, to some extent, to the *tetherball* game, with the difference of using a variable-length (elastic) tether that has a maximum allowed length. AUVs keep track of their initial drop-off location by continuously estimating the straight traveled distance, *the tether*, and the turns executed since the last tether-length and direction calculation. The likelihood of turning in a direction that can lead to overshooting the maximum tether length diminishes radially outwards from the drop-off location.

This design allows the AUVs to stay within a constrained area determined by the maximum tether length and prevents them from getting lost. It also obviates the need for a gradient to follow in the search for target; vehicles move randomly within the aforementioned area following some distribution (e.g. normal or uniform). This also means that it is not possible for a vehicle to be trapped in a local minimum like in R-PSO. No communication is required as every vehicle keeps its own tether. Additionally, localization is not needed as the AUV can keep track of the sequence of travels and turns to estimate the tether direction and length.
using *dead reckoning*. At any time, the AUVs keeps track of only two vectors: the previous tether and the vector pointing back to that tether from its current location; the delta vector. Figure [3-22] shows the basic idea of VTS. The above referenced vectors are: the tether $T_{old}$ and update $\delta_T$ vector. When a tether update is required, they are used to estimate the new tether $T_{new}$. After each tether update, the vehicle travels for a distance assuring that the following tether will not exceed the maximum tether length, with a 1% chance of updating the tether during that travel. The product of the old tether’s length and the angle between that tether and the heading direction of the vehicle is used as a control variable for determining the probability of selecting the next turn angle. The larger the product the closer the angle to $\pi$, simulating a rebound off the perimeter (of the circle bounding the search area) back towards the origin. For small products, the turn angle approaches zero meaning that the AUV will tend to travel straight near the origin (drop-off location). A more formal analysis is provided in the next section.

**Theory**

Assume that the initial drop-off location of AUV $a_i$ is $P_i^l$. The AUV travels at a constant velocity $V_i$ for a distance $d_i$ in the radially outward direction from that location. The probability of making a bigger turn needs to increase as the vehicle moves towards the edge of the search area (to keep the AUV within a limited search area). As the AUV should not exceed the maximum tether length $T_{max}$, the probability of making a turn (at an angle $\theta$) that results overshooting that length must be equal to zero at $T_{max}$. In other words, the probability of making a $\pi$ degrees turn must be 1 at the perimeter as shown in Figure [3-22](a). Exceeding $T_{max}$ can occur when either continuing the radial travel ($\alpha_t = \pi$) or by turning an angle $\theta < \pi$, when $T_{max}$ has been reached. The desired behavior can be envisioned as an odometer’s pointer with its center of rotation placed on a dot that moves from the center.
of the disc outwards on its radius (Figure 3.22(b)). Initially, the pointer points in the dot’s travel direction. As the dot moves, the pointer turns away from the radius until it eventually coincides again with it, this time pointing inwards, when the dot has reached the perimeter of the disc.

To achieve the algorithm’s intended behavior, we use the normalized tunable half sigmoid function \[19\] (presented in Section 3.5) along with a normal distribution to add some uncertainty as will be explained next. We start by finding the tether vector. If the current vehicle’s position is \(P_i^c\), the initial tether vector \(T_i^l\) for vehicle \(a_i\) can be found by the difference between initial and current positions as: \(T_i^l = P_i^l - P_i^c\). Because of the absence of a global positioning/localization mechanism, these positions are not available. Using dead reckoning, the AUV can estimate the initial tether’s length and direction by measuring the
traveled distance and relative heading change at turning time. It is easy to see that initial tether’s direction is opposite to the AUV’s travel direction. Assuming that the AUV calculates the length of the tether every $t$ seconds, the distance $d^i_t$ traveled within this period can be found as $d^i_t = \|V_i\|t$. When the AUV is first released, there is no tether specified. After it has traveled a distance $d^i_t$, the initial tether is calculated. Therefore, the length of the initial tether is: $T^i_{t_{max}} = d^i_t = \|V_i\|t$. The full specification of the initial tether vector is then: $T^i_{t_{max}} = -d^i_t u = -\|V_i\|t u = -V_i t$, where $u = V_i/\|V_i\|$ is the unit vector in the direction of $V_i$. Now, that the initial tether has been specified, the AUV calculates the maximum distance $d_{t_{max}}$ it can travel without violating the maximum tether constraint assuming no turns will
be made other than the one at the end of that distance. Figure 3-22(c) shows that distance where \( T^c_i \) is the current tether (\( T^c_i = T^f_i \) in this case), \( \alpha^c \) is the angle with that tether, and \( T_{max} \) is the maximum tether. Initially, this distance is equal to the difference between the maximum tether length and the, very short, initial tether as both tethers are aligned and the AUV travels radially outwards. For the following tether updates, this maximum distance can be easily calculated using the law of cosines, Eq. 3.24, and finding the roots. The AUV then starts to travel that distance, where tether updates continue at the regular intervals (every \( t \) seconds) along the way. Given that updates take place every \( t \) seconds, \( m = d_{t_{\max}} / d_{t_i} \) updates are possible along the path. At every tether update, the vehicle can either decide to continue traveling along that path or execute a turn with a probability \( p = 0.01 \). As previously discussed, the farther the vehicle is from the origin, the sharper the turn will be.

By using the normalized tunable half sigmoid, shown in the middle of Figure 3-23 and the tether length-turn angle product as described in the previous section, it is possible to obtain larger turning angles with the increase in distance from origin. To determine the turn angle, first the product \( P = T_{old} \alpha_{t} \) of the current tether’s length \( T_{old} \) and the angle \( \alpha_{t} \) the vehicle makes with that tether is found. This product is then normalized using the maximum tether length-maximum angle product \( T_{max} \alpha_{max} = \pi R_{max} \). This normalized product can then used to compute the value of the normalized tunable half sigmoid function using Eq. 3.21 where \( k = -1.2 \) was used to make the function convex as shown in the middle of Figure 3-23. The value of the function can, in turn, be used to compute the turning angle by specifying the fraction of \( \pi \) to be used (i.e. by finding \( \pi F(P) \)). To introduce nondeterminism and increase the range of angles used at a specific radius, \( P \) is used instead as the mean \( (\mu = P) \) of a normally distributed random variable with a standard deviation of \( \sigma = 0.8 \). The value of that random variable is what we use instead to calculate the normalized tunable half sigmoid as shown at the bottom of Figure 3-23. Eq. 3.21 can now be replaced by Eq. 3.25.
Table 3.2: Virtual Tether Search algorithm parameter values.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{max}}$</td>
<td>Max. tether length</td>
<td>Max. distance from AUVs to origin</td>
<td>80</td>
</tr>
<tr>
<td>$\alpha_{\text{max}}$</td>
<td>Max. AUV-tether angle</td>
<td>Maximum angle between AUV &amp; tether</td>
<td>$\pi$</td>
</tr>
<tr>
<td>$k$</td>
<td>Convexity coefficient</td>
<td>Normalized tunable half sigmoid coeff.</td>
<td>-1.2</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Standard deviation</td>
<td>Normally dist. rand. variable standard dev.</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Calculating the turning angle, the AUV can then turn towards the new direction. The new tether can easily be calculated using Eq \[3.26\] where $\delta_T$ is a vector representing the fraction of $d_{t_{\text{max}}}$ traveled before a turn was triggered and pointing back to the most recent tether $T_{\text{old}}$.

\[
d^2_{t_{\text{max}}} - (2T^c \cos \alpha^c)d_{t_{\text{max}}} + (T^c)^2 - T^2_{\text{max}} = 0 \tag{3.24}
\]

\[
F(X) = \frac{kX}{k - X + 1}, X \sim N(P, \sigma^2) \tag{3.25}
\]

\[
T_{\text{new}} = T_{\text{old}} + \delta_T \tag{3.26}
\]

Table 3.2 lists values of the different parameters of VTS used in the simulation. In the next section, pseudo code of the algorithm is presented and explained.

Algorithm

Algorithm 3 shows the pseudo code of Virtual Tether Search. The algorithm runs in a loop which either applies the algorithm’s updates or keeps the AUV at the target once it is found. Lines 2-9 are used for initialization. Line 3 sets the previous position to the initial drop-off location of the vehicle. Line 4 uses current and previous locations to get the initial tether vector $T_{\text{old}}$. Current location is then stored as the previous location on line 5 for future
iterations. The distance traveled so far $\delta_T$ is set to zero on line 6. Flags for indicating whether a straight travel has been broken and initialization status are set on lines 7 and 8, respectively. Next, lines 10-13 check if the target has been found and force the vehicle to stay at target accordingly. On line 14, the distance traveled is updated based on current and previous positions. The new tether is found by the sum of the old tether and the traveled distance on line 15. Previous position is updated again on line 16. The angle $\alpha$ between the new tether and the vehicle’s heading direction is found on line 17 and tether length is extracted on line 18. Next, the new tether, maximum tether length, and the angle are used to calculate the maximum distance $d_{t_{\text{max}}}$ that can be traveled without exceeding the search area’s radius on line 19. With a low probability, a turn may be triggered as shown on line 20. If a turn is triggered (lines 21-28), first a check of whether the straight travel was broken in the previous iteration is done to avoid breaking a travel in consecutive iterations. If it was not just broken, the traveled distance is set to the maximum allowable as on line 25 to trigger a turn. If the the distance traveled so far has not exceeded $d_{t_{\text{max}}}$, the AUV continues to travel in a straight line (line 30), updating the traveled distance along the way, otherwise, it resets $\delta_T$ to zero as on line 33. In case of a triggered turn, tether length-angle product is found and normalized as on lines 35 and 36. The normalized product is then used as the mean of a normal random variable $X$ with standard deviation $\sigma_g$ (line 37). The latter is, in turn, used as the input to the normalized tunable half sigmoid function, as shown on line 38, to get the new turn angle $\theta$ and direction. The new direction is then used on line 39 to reorient the AUV before it starts to travel in a straight line again.

3.6.3 Constrained Spiral Flocking (CSF)

One shortcoming of the two previous algorithms is that they do not maintain swarm unity as in natural swarms like fish schools and bird flocks; AUVs travel in different directions
Algorithm 3 Virtual Tether Search
(* Search an Area with Limited Radius *)
1: loop
2: if not initialized then
3: \( P_{prev} \leftarrow P_{drop-off} \)
4: \( T_{old} \leftarrow P_{prev} - P_{curr} \)
5: \( P_{prev} \leftarrow P_{curr} \)
6: \( \delta_T \leftarrow 0 \)
7: just_broke_traveling \leftarrow \text{false} \)
8: initialized \leftarrow \text{true} \\
9: end if
10: if target_found then
11: \text{StayAtTarget()} \\
12: continue \\
13: end if
14: \( \delta_T \leftarrow P_{prev} - P_{curr} \)
15: \( T_{new} \leftarrow T_{old} + \delta_T \)
16: \( P_{prev} \leftarrow P_{curr} \)
17: \( \alpha \leftarrow \cos^{-1}(T_{new} \cdot u_V) \) \hspace{1cm} \triangleright u_V \text{is a unit vector in the direction of velocity} \\
18: T \leftarrow \text{GetTetherLength()} \\
19: d_{t_{max}} \leftarrow \text{CalculateMaxDistToTravel}(T_{new}, T_{max}, \alpha) \\
20: turn_triggered \leftarrow \text{TriggerTurnWithLowProb()} \\
21: if turn_triggered then
22: if just_broke_traveling then
23: just_broke_traveling \leftarrow \text{false} \\
24: else
25: \( \delta_T \leftarrow d_{t_{max}} \) \\
26: just_broke_traveling \leftarrow \text{true} \\
27: end if \\
28: end if \\
29: if \( \delta_T < d_{max} \) then
30: \( \delta_T \leftarrow \text{TravelAtMaxSpeed}(d_{t_{max}}) \) \\
31: continue \\
32: else
33: \( \delta_T \leftarrow 0 \) \\
34: end if
35: \( \text{len_angle_prod} \leftarrow \alpha \times T \)
36: \( \text{len_angle_prod}_{\text{norm}} \leftarrow \text{len_angle_prod} \div \alpha_{max}T_{max} \)
37: \( \text{len_angle_prod}_{\text{norm}} \leftarrow \text{len_angle_prod}_{\text{norm}} + \sigma_i X_i \) \triangleright \text{Gaussian error. Only values } \in [0,1] \text{ are used} \\
38: V_{\text{VTS}} \leftarrow \text{GetNextTurnDirection}(\text{len_angle_prod}_{\text{norm}}) \hspace{1cm} \triangleright \text{use the normalized half sigmoid function's value for } \text{len_angle_prod}_{\text{norm}} \text{ to select turn angle } \theta \in [0,\pi] \\
39: \text{ReorientAUV}(V_{\text{VTS}}) \\
40: end loop
and don’t consider unity/cohesion, or the swarming effect, as necessary. To scan large, unbounded areas at high speeds while maintaining swarm unity, we propose Constrained Spiral Flocking (CSF) search. The next subsections describe CSF in detail.

Concept

The basic idea is to allow AUVs to move in an expanding motion covering a large area, then contracting back to the source if a target is not found. To achieve this, CSF uses a logarithmic spiral and its mirror image as the two main paths of movement of AUVs. The first spiral originates at drop-off location and expands outwards until a certain, predefined radius is reached. The second spiral then starts where the first ends and develops in a shrinking manner until vehicles are pulled back to origin. This is illustrated in Figure 3-24.

To maintain the swarming effect, AUVs, again, apply Reynolds’ [67] flocking rules to achieve cohesion, separation, and alignment among themselves. This flocking effect is combined with the paths determined by the aforementioned spirals by the use of spiral and flocking weights.

Theory

A logarithmic spiral can be represented by Eq. 3.27 where \( r \) is the growing radius of the spiral (distance from origin), \( \theta \) is the angle with some fixed axis, and \( a \) and \( b \) are scale factors.

\[
\begin{align*}
  r &= a e^{b \theta} \\
  (3.27)
\end{align*}
\]

To obtain two concatenated spirals, Eq. 3.27 can be combined with the standard Triangular Function (Eq. 3.28) to achieve the growing then shrinking effect. Mathematically, this can be expressed as a spiral with a radius \( r_{sp} \) that changes according to the triangular function and an angle \( \theta_{sp} \) that increases proportionally to \( r_{sp} \) as given by Eq. 3.29 and 3.30 where
$r_{\text{max}}$ is the maximum radius of the spiral, $t \in [0,2r_{\text{max}}]$, and $s_t$ is the angle step size. This representation was specially chosen for implementation purposes. Spirals shown in Fig. 3-24 were obtained using $r_{\text{max}} = 40$ and $s_t = 0.1$.

Flocking is achieved by applying the three rules previously presented in Eqs. 3.3, 3.4, and 3.2.

\[
tri(x) = \land(x) = \begin{cases} 
1 - |x|, & |x| < 1 \\
0, & \text{otherwise}
\end{cases}
\]  

(3.28)

\[
r_{sp} = r_{\text{max}} \left[ 1 - \frac{1}{r_{\text{max}}} (t - r_{\text{max}}) \right]
\]  

(3.29)

\[
\theta_{sp} = s_t \pi r_{sp}
\]  

(3.30)

The direction traveled $V^T_i$ by AUV $i$ is then determined by the weighted sum of the flocking $V^f_i$ and spiral $V^s_i$ directions as in Eq. 3.31 with the aid of spiral $w_s$ and flocking $w_f$ weights. Table 3.3 lists values of the different CSF parameters used in the simulations presented later.

\[
V^T_i = w_s V^s_i + w_f V^f_i
\]  

(3.31)

**Algorithm**

Pseudo code of CSF is shown in Algorithm 4. The algorithm is initialized by setting the spiral’s radius to grow, setting in-place turning as inactive (used when the target is not found, and the AUV returns to drop-off location, to wait for pickup), and setting the active turn radius of the AUV equal to its MTR (2-7). If the target is found, the AUV broadcasts a message notifying nearby vehicles and stays close to it (lines 8-11). If not found, it listens to neighbors and sets target to found or not accordingly (lines 12-17). Next, spiral travel and
flocking direction components are calculated and their weighted sum is found to determine CSF’s travel direction (lines 18-20). The AUV is then reoriented in that new direction (line 21). If in-place turning was activated (meaning that the active radius is equal to the MTR), the AUV will just proceed to next iteration (lines 22-24) as the reorientation based on the turning direction will have already been applied (on line 21). At the end of each iteration, the turning radius is either grown of shrunk by a fixed increment \(\delta_R\) (lines 26 and 31) based on a flag controlled by the maximum radius of the spiral. Growing and shrinking are not allowed to exceed or fall below the maximum spiral radius and minimum turn radius, respectively. Once the spiral reaches its maximum radius, it starts to shrink until it reaches it is MTR where is continues to circle until picked up.
Algorithm 4 Constrained Spiral Flocking

1: loop
2:   if not Initialized then
3:     grow_radius $\leftarrow$ true
4:     inplace_turn $\leftarrow$ false
5:     $R_T^{\text{active}}$ $\leftarrow$ min_turn_radius
6:     Initialized $\leftarrow$ true
7:   end if
8:   if target_found then
9:     BroadcastTargetFound()
10:    StayAtTarget()
11:    continue
12: else
13:    target_found $\leftarrow$ ListenToNeighbors()
14:    if target_found then
15:       continue
16:    end if
17: end if
18: $V_s$ $\leftarrow$ CalcSpiralDirection($R_T^{\text{active}}$)
19: $V_f$ $\leftarrow$ CalcFlockingDirection() $\triangleright$ Reynolds flocking
20: $V_{CSF}$ $\leftarrow$ $w_s V_s + w_f V_f$
21: ReorientAUV($V_{CSF}$)
22: if inplace_turn then
23:   continue
24: end if
25: if grow_radius then
26:   $R_T^{\text{active}}$ $\leftarrow$ $R_T^{\text{active}} + \delta_R$
27:   if $R_T^{\text{active}} == R_s^{\text{max}}$ then
28:     grow_radius $\leftarrow$ false
29:   end if
30: else
31:   $R_T^{\text{active}}$ $\leftarrow$ $R_T^{\text{active}} - \delta_R$
32:   if $R_T^{\text{active}} == R_T^{\text{min}}$ then
33:     inplace_turn $\leftarrow$ true
34:   end if
35: end if
36: end loop
Table 3.3: Constrained Spiral Flocking algorithm parameter values.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_s$</td>
<td>Spiral weight</td>
<td>weight of spiral component</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_f$</td>
<td>Flocking weight</td>
<td>weight of flocking component</td>
<td>0.2</td>
</tr>
<tr>
<td>$r_{s,max}$</td>
<td>Max. spiral radius</td>
<td>Max. spiral expansion radius</td>
<td>40</td>
</tr>
<tr>
<td>$w_c$</td>
<td>Cohesion weight</td>
<td>flocking alg. cohesion weight</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_s$</td>
<td>Separation weight</td>
<td>flocking alg. separation weight</td>
<td>0.5</td>
</tr>
<tr>
<td>$w_a$</td>
<td>Alignment weight</td>
<td>flocking alg. alignment weight</td>
<td>0.999</td>
</tr>
</tbody>
</table>

3.6.4 Simple Sweeping (SSW)

A simple sweeping algorithm was implemented for comparison purposes. This type of algorithms are also known in the literature as raster scan or raster sweep algorithms, and are usually used by single AUVs. In our implementation, when the vehicles are first dropped, they reorient towards magnetic north direction before they start executing straight, limited range sweeps in a direction normal to magnetic north. This can be any of the two possible directions. The AUVs first travel a distance equal to half of the maximum individual sweep’s length as shown in Figure 3.25. They follow that by a 180 degrees turn and continue sweeping at full length. When sweeping is complete, i.e. the maximum number of sweeps $n_{max}$ has been reached, the AUV heads back towards its original drop-off location. This is done by turning 90 degrees in drop-off direction, followed by traveling a distance equal to $d_{min}n_{max}$ where $d_{min} = 2r_{min}$ is the vehicle’s minimum turn diameter ($d_{sense}$ can be used instead). As a final step, AUVs turn by an angle $\frac{\pi}{2}$ towards the initial position and travel a distance of $\frac{l_s}{2}$, where $l_s$ is the full sweep length. When the vehicle reaches the source, it activates an in-place turn until picked up. If, at any time, the target is found, sweep sequence execution is canceled. Algorithm 5 shows the algorithm’s pseudo code, which is straightforward to understand.
3.6.5 Swirling Divided Hexagonal Close Packing (SDHCP)

Inspired by the arrangements of equal-sized spheres/circles to occupy minimal volumes through the use of Hexagonal Close Packing (HCP), we devised Swirling Divided HCP search. Another reason for developing SDHCP is to solve the directionality problem of SSW. The details of the algorithm are presented in the following sections.

Concept

Assuming that sensing range of an AUV has a spherical shape, the goal is to allow each AUV to travel in a sequence of deterministic steps, similar to simple sweeping, that cover searched area in the most compact way. Simple sweeping has the drawback of being directional; if the search is started in a specific direction, it continues in that direction and will fail if the target is not in the search direction. SDHCP tries to mitigate this by dividing the swarm into six sub-swarms that travel in directions separated by $\frac{\pi}{3}$ angles, while still performing the sweeps. The original design of SDHCP (called DHCP) allowed each of these swarms to sweep one of the six triangles forming the resulting hexagon in a growing sweep pattern as shown in Figure [3-26] by the gray-highlighted triangle, then return back to the origin when the target

Figure 3-25: Simple Sweeping algorithm illustration.
Algorithm 5 Simple Sweeping

1: loop
2: if not Initialized() then
3:     \( V_N \leftarrow \text{MeasureMagneticNorthDir}() \)
4:     ReorientAUV(\( V_N \))
5:     for predefined_sweeps_number do
6:         AddSweepSubSeqToTurnSeq()
7:     end for
8:     composite_sweep_seq_completed \( \leftarrow \) false
9:     SetInitialized()
10:    continue
11: end if
12: if composite_sweep_seq_completed then
13:     ExecuteInplaceTurn() \( \triangleright \) back at source
14:     continue
15: end if
16: if target_found then
17:     StayAtTarget()
18:     continue
19: end if
20: composite_sweep_seq_completed \( \leftarrow \) ExecuteNextSweepSubSeq()
21: end loop

is not found. This design had the drawback that at least 4 of the six sub-swarms fail to find the target and end up returning to the origin. To better utilize resources, SDHCP was devised. The improvement that SDHCP adds is that it allows each sub-swarm to first diverge along one side of the triangle then do a sweep along the perimeter of the hexagon defined by that divergence step. After it completes \( \frac{7}{6} \) cycle around the perimeter, it diverges again and does another cycle along the next larger hexagon’s perimeter in the opposite direction (hence the swirling prefix). The process continues until the predefined number of divergence steps has been reached, at which point the sub-swarm returns to origin if no target is found. This process is illustrated in Figure 3-27. It is worth noting that this modification assures that the target will be found in any direction by all six sub-swarms given that it falls within a distance less than or equal to the side length of the largest hexagon.

This algorithm requires the use of the initial self organization stage described in Sec-
Every AUV flips a coin to pick one of the six predefined directions relative to the measured magnetic north. It then starts to execute the sequence shown in Figure 3-27, which can be broken into six main parts:

- **Odd divergence Sequence:** This refers to the main sequence used by the AUVs to diverge from source location. It consists of traveling a distance equal to double the sensing range $2r_{\text{sense}} = d_{\text{sense}}$, followed by a right turn (left turn could be equivalently used) at an angle $\theta = \frac{2}{3}\pi$ from the vehicle’s travel direction.

- **Clockwise circling Sequence:** traveling a distance $i \times d_{\text{sense}}$, where $i$ is the sweep number, 7 times interleaved with $\theta = \frac{2}{3}\pi$ turns in the clockwise direction.

- **Even divergence Sequence:** a turn in the opposite direction (to the turn used in odd divergence) at an angle $\frac{\pi}{3}$, followed by traveling a distance equal to double the sensing range $2r_{\text{sense}} = d_{\text{sense}}$, followed by a left turn (right turn could be equivalently
used) at an angle $\theta = \frac{2}{3}\pi$ from the vehicle’s travel direction.

- **Counter-clockwise circling Sequence:** traveling a distance $i \times d_{sense}$, where $i$ is the sweep number, 7 times interleaved with $\theta = \frac{2}{3}\pi$ turns in the counter-clockwise direction.

- **Last circling:** Instead of using an opposite turning direction at the end of the circling, a turn in the same direction at an angle $\theta = \frac{2}{3}\pi$ allows the swarm to head back to origin. To complete its trip to origin, it has to travel a distance equal to $d_{sense}$ multiplied by the max number of sweeps $n_{max}$.

- **In-place turn:** After the AUV returns back to its initial drop-off location, it starts executing an in-place turn until picked up.
Algorithm

Algorithm 6 shows the pseudo code of SDHCP. First, initialization takes place (lines 2-12): magnetic north direction is estimated (line 3), a random multiple of $\frac{\pi}{3}$ is picked and used to rotate the magnetic north direction (line 4), Reynolds’ flocking direction is calculated (line 5), the weighted combination of the rotated magnetic north and flocking directions is found and used as the AUV’s new orientation direction (lines 6 and 7), and sweep sequence defining the path that will be traveled by the AUV is built and activated (lines 8 and 9). In every iteration, a check is done to see if the target has been found (line 13). If found, the AUV broadcasts an notification to nearby vehicles and stays at target (lines 14-16), otherwise, it checks if any of the neighbors broadcast a target found message (line 18). If the AUV does not find the target and the sweep sequence finishes, this means that it has returned back to origin and needs to start an in-place circling behavior until picked up (lines 23 and 24).

3.6.6 Simple Random Walk (SRW)

Standard random walk algorithm was implemented taking into consideration the physical constraints of the vehicles. When a random turn is to be executed (done every 6 simulation ticks in our implementation), the vehicle turns according to its minimum turn radius and maximum turn angle constraints. This, in fact, gives the generated path a more natural look as opposed to point mass implementations. Pseudo code of the algorithm is very simple and is provided in Algorithm 7.

3.7 Task Identification and Allocation

In this work, we follow the taxonomy proposed by Burger in [9] for task allocation in swarm robotics. In his work, Burger refined the yardsticks that can be used to describe how swarm-
Algorithm 6 Swirling Divided Hexagonal Close Packing Search

1: loop
2:     if not initialized then
3:         $V_N \leftarrow \text{MeasureMagneticNorthDir}()$
4:         $V_{N_r} \leftarrow \text{RotateMagNorthByMultOfSixty}(V_N)$
5:         $V_f \leftarrow \text{CalcFlockingDirection}()$          \Comment{Reynolds’ flocking}
6:         $V_{SDHCPS} \leftarrow w_{N_r} V_{N_r} + w_f V_f$
7:         $\text{ReorientAUV}(V_{SDHCPS})$
8:         sweep_seq $\leftarrow \text{BuildSweepSequence}()$
9:         $\text{StartSeqExecution}(\text{sweep_seq})$
10:        initialized $\leftarrow$ true
11:        continue
12:      end if
13:      if target_found then
14:         $\text{BroadcastTargetFound}()$
15:         $\text{StayAtTarget}()$
16:         continue
17:      else
18:         target_found $\leftarrow \text{ListenToNeighbors}()$
19:         if target_found then
20:             continue
21:         end if
22:      end if
23:      if sweep_seq_completed then
24:         $\text{ExecuteInplaceTurn}()$          \Comment{back at source}
25:      end if
26:  end loop

Algorithm 7 Simple Random Walk

1: loop
2:     if target_found then
3:         $\text{StayAtTarget}()$
4:         continue
5:     end if
6:     $V_r \leftarrow \text{GenerateRandomTurnDirection}()$
7:     $\text{ReorientAUV}(V_r)$
8:  end loop
robotic a system is and highlighted the differences between multi-robot systems and swarm robotics. His taxonomy for task allocation distinguished between three main types of task allocation in swarms: 1) Heteronomous, 2) Autonomous, and 3) Hybrid task allocation. A brief overview of these techniques is presented in the next section.

3.7.1 Task Allocation Techniques

In this section, we summarize the techniques presented by Burger to highlight the relevance of different techniques in underwater environment and to pave the road for our selected and proposed techniques in the sections to follow.

Heteronomous Task Allocation

Heteronomous task allocation encompasses the techniques in which task allocation is controlled by decisions made outside the agent either by a central entity (centralized) or communication with other agents (distributed).

*Centralized (single point of failure):*

- **Omniscient (not scalable - computation/communication):** assumes all-knowledgeable central entity, no contribution of workers to the decision making process.

- **Blackboard Control (high computational expense, unpredictable environment dynamics):** a central decision maker/task allocator uses a blackboard as its knowledge base, while workers send information to the blackboard to provide that knowledge. The central controller allocates tasks based on this information and either sends tasks directly to workers or through the blackboard.

- **Centralized Reinforced Control:** the central controller/leader uses learning to adapt and optimize its task allocation process to the dynamics of the environment.
Distributed (negotiation increases communication cost):

- **Market Based Approaches** (swarm and neighborhood sizes affect performance): mainly based on the use of an auctioning mechanism to allow agents to bid for tasks. Bidding can be based on cost offered by the bidder or on fitness to the announced task. Trading the assigned tasks can be allowed or prohibited. Bids can also be dynamic with lower and upper bounds on the acceptable prices for tasks.

- **Virtual Blackboard**: every agent is given a private blackboard and uses broadcast messages to update other blackboards. A global virtual blackboard emerges that serves as the leader in the system. It requires reliable communication and execution of the same algorithm by individual agents.

- **Opinion Dynamics**: Each robot holds an opinion about which task is better to allocate compared to other tasks. Before actual allocation, a robot negotiates with a random group of preset size using a decision rule like majority vote to decide which task is better. The rate of accomplishing the actually better task gives it a better chance to be allocated to more robots as more of the ones who already participated in it will be available to affect the opinions of others.

- **Broadcast of Local Eligibility**: robots broadcast their eligibility for all of their behaviors and the robot with the highest eligibility performs the task. When a robot performs a task, it sends an inhibition message to other robots in its range to prevent them from performing it.

The drawback of Heteronomous task allocation is that it usually requires high computational and communicational expenses as coordination is done in a top-down manner (leader communicating to all robots and vice versa).
Autonomous Task Allocation (Decentralized)

In autonomous task allocation, decisions are made independently by the agent in a decentralized manner. Burger classified Autonomous task allocation into the following types of control:

- **Rule-Based Control**: a set of rules determine the behavior of the robot. This behavior is what the robot uses to fulfill a certain task. A behavior can be atomic or complex, where a complex behavior consists of a set of basic behaviors. Rule-based control works by activating suitable behaviors from the robot’s repertoire of complex and basic behaviors. Although rule-based control is very powerful, its disadvantage is that it is difficult to emerge swarm-level behavior based on definition of simple rules at the agent’s level.

- **Threshold-Based Control**: this type of control uses a threshold to determine when a task can be activated. If the threshold is absolute, i.e. a constant value that needs to be exceeded for the task to be activated, control is said to use activation threshold model. When the threshold is used as a function-parameter to determine how likely a task is to be done, control is said to use an adaptive threshold function. In both cases, a stimulus (based on sensory input for example) is compared to the threshold to determine activation status or likelihood. Thresholds used by different robots can be different to prevent concurrent activation of the same task.

- **Probabilistic Control**: in this type of control, each robot uses a discrete probability distribution to determine which task to activate. This distribution can be different across robots. The key point in probabilistic task allocation is how to define the discrete probability distribution. Additionally, probabilities may be adapted according to some criteria. For example, Burger [9] proposed the use of adaptive threshold...
functions to specify the motivations used in determining the probabilities.

- **Decentralized Reinforced Control**: techniques of reinforcement learning are employed to self-allocate to tasks in this control approach. Basically, the robot positively reinforces task allocation actions that led to a satisfactory reward. The goal is to find a *policy* that maximizes the reward associated with task accomplishment. Design of a suitable reward is the challenging part in this type of control.

**Hybrid**

Hybrid task allocation, uses combinations of heteronomous and autonomous mechanisms to achieve the target allocation. Burger divided Hybrid control into:

- **Interlaced Control**: in this type of hybrid task allocation, techniques from heteronomous and autonomous task allocation are combined to produce a new type of control.

- **Side-by-side Control**: here techniques from the both approaches work side-by-side in an alternating or concurrent way.

From the above overview, it should be clear enough that Autonomous Task Allocation is the most suitable option for underwater swarms. The reason is that limited communication can severely affect the performance of task allocation techniques that depend heavily on communication. Additionally, dependence on a central unit to serve as the leader is not reliable in such environment; with the relatively high agent loss probability, such central agent can be a single point of failure.

For the above reasons, we elect to use Autonomous Task Allocation in this work. Algorithms that fall under individual categories (of the ones discussed above) are presented,
as well as others which can be thought of as hybrid autonomous task allocation techniques. The developed algorithms are presented in detail in the next few sections.

### 3.7.2 Response Threshold-Based Task Allocation

As presented in Section 3.7.1, threshold based task allocation can take multiple forms. Adaptive threshold functions (also called response thresholds) highlighted by Burger [9] have been previously used by Bonabeau et al. [7] to describe task allocation and specialization in ant colonies. An advantage of adaptive threshold functions is that they allow for specialization as different agents can have different thresholds that can be adapted to define different response levels to similar stimuli.

We adopt this approach in this work due to its relevance to underwater environment where communication is severely limited and autonomous task allocation is the best choice. Due to the difference in the possible dynamics between above-ground and underwater swarms, at-target motion plays an significant role in the task allocation process as agents are not as free to stop between allocations as in the case of above-ground agents. Combining adaptive threshold functions with specially designed local search algorithms significantly affects the results of the task allocation process.

In the next section, we describe two commonly used adaptive threshold functions and show how we use them in the context of an underwater task allocation scenario. Local search algorithms used along with these functions are presented in Section 3.8.

**Exponential and Fractional-Polynomial Response Thresholds**

Adaptive threshold functions (ATFs) relate a threshold $\theta$ and to a stimulus $s$. They determine the likelihood of responding to an active stimulus. An exponential response threshold can be expressed as in Eq. 3.32. In this threshold function, when the stimulus is equal to the
threshold, the likelihood of response becomes 0.632. A response threshold can also take the form of a polynomials fraction, which we call *fractional-polynomial response threshold* in this work, that takes the form shown in Eq. 3.33 where $n$ determines the steepness of the response curve and is $> 1$ (usually, $n = 2$ is used). Likelihood of response is 0.5 in this kind of response threshold when the stimulus equals the threshold.

$$T_{\theta} = 1 - e^{-s/\theta}$$ \hspace{1cm} (3.32)

$$T_{\theta} = \frac{s^n}{s^n + \theta^n},$$ \hspace{1cm} (3.33)

In the context of this dissertation, the search stage’s target was chosen to be either a ship or a set of submerged barrels lumped into one or three spatially-separated groups (to represent a task with clear partitions in the second case). AUVs used their sensors to detect barrels and adjacent vehicles at target location. Number of barrels was used as the stimulus $s$ and the threshold $\theta$ was chosen to be the number of vehicles sensed by the AUV near to the target. At any point in time, each AUV is oriented differently and has different number of sensed targets. Therefore, the thresholds $\theta$ are different from one vehicle to another. This causes the vehicles to respond differently for the same conditions that are currently active in the environment.

The algorithms using these two types of response threshold functions are relatively simple and their only difference is the application of the specific function. We denote the exponential response-threshold based algorithm as ERT-TA and the fractional polynomial-based one as FP-TA. The pseudo code of the former is provided as an example in Algorithm 8.
Algorithm

The algorithm runs repeatedly within the main loop of the AUV’s controller (lines 1 and 18). First, sensors are consulted for the numbers of nearby vehicles and targets (lines 2-3). If at least one target is sensed (line 4), a check of the number of sensed vehicles is done (lines 5-9), otherwise, the AUV simply continues to search locally for the next target using ones of the local search algorithms presented in Section 3.8 on line 16. When the number of sensed vehicles is zero (line 5), the response threshold function’s value is set to one (line 6) to force the AUV to process the nearest target (line 13). If there are vehicles nearby (line 7), the functions value is calculated using the numbers of sensed targets and vehicles as on line 8. To simulate probability, the value of a uniform random variable is compared to the function’s value on line 10 to decide whether to continue searching for the next target (line 11) or to process the current one (line 13). It should be noticed that as mentioned in the previous section, the adaptive threshold function determines when taking an action becomes more likely. As the function’s value becomes smaller and smaller, the uniform random variable’s value is expected to exceed that value more often, implementing the described behavior. FP-TA’s only difference from the ERT-TA is that line 8 would be replaced by the function provided by Eq. 3.33.

3.7.3 Beacon Based Task Allocation

When the target consists of spatially separated groups, it is helpful to have some sort of indicator that a group of targets lie at a certain location. This can help nearby AUVs that have not found targets yet to be attracted to the target(s) location. This can also be very useful in case of having different types of targets as is the case in this work. Vehicles that can flag certain target groups, e.g. by using a special light color or flashing pattern, can have help improve task allocation and division of labor. Additionally, the use of beacons
Algorithm 8 Exponential Response-Threshold Task Allocation

1: loop
2: \( N_v \leftarrow \text{Count}(\text{SenseNearbyVehicles}) \)
3: \( N_t \leftarrow \text{Count}(\text{SenseNearbyTargets}) \)
4: if \( N_t > 0 \) then
5: \hspace{1em} if \( N_v = 0 \) then
6: \hspace{2em} \( \text{resp\_thresh\_fn} \leftarrow 1 \)
7: \hspace{1em} else
8: \hspace{2em} \( \text{resp\_thresh\_fn} \leftarrow 1 - e^{-N_t/N_v} \)
9: \hspace{1em} end if
10: \hspace{1em} if UniformRandom() > \text{resp\_thresh\_fn} then
11: \hspace{2em} SearchLocallyForOtherTargets() \hspace{1em} \triangleright \text{e.g. using Bubble-Chain RW}
12: \hspace{1em} else
13: \hspace{2em} ProcessNearestTarget()
14: \hspace{1em} end if
15: \hspace{1em} else
16: SearchLocallyForOtherTargets()
17: end if
18: end loop

reduces overcrowding at targets, which can lead to performance degradation, by spreading the agents over different target groups based on individual target processing preferences.

We propose beacon based task allocation (BB-TA), where vehicles can voluntarily decide to serve as beacons for other vehicles based on certain criteria in a completely autonomous way. These vehicles act as attractors of other vehicles as well as guides for the self-organization of the vehicles at the targets.

Concept

Each AUV runs the same algorithm, however, based on which agents reach a target first and on the sensed number of neighboring AUVs, an agent can decide to volunteer as a beacon for a period determined by the number of agents it attracts over time. When an AUV encounters a target, it checks its sensors for any neighboring vehicles. If the number of neighbors is below some threshold, it decides with a low probability to become a beacon.
Before doing so, it first does a localized target area sweep to estimate the number of targets at the location. During the sweep, it keeps track of the average per-sweep targets count. At the end of this sweeping process (which has a predefined number of sweeps), the AUV decides whether to start serving as a beacon or to give up doing so based on the per-sweep targets count is has observed. In case decides to continue, the AUV starts a circular motion that surrounds the area spanned by the sweep sequence it just completed. It also turns on a relevant light indicator reflecting the density of targets in the area and the majority target type or another indicating a uniform mix of targets. When the circling behavior has started, the AUV keeps track of neighbors count over a predefined period and gives up the beaconing behavior if insufficient neighbors count has been observed.

The alternative behavior that the AUV can perform, when having selected not to serve as a beacon, is to look for targets and process them. This is accomplished using the local search algorithms proposed in Section 3.8. During that search, the AUV can also be attracted to an active beacon. Attraction takes place as the vehicle continuously uses its light detection sensors for any nearby beacons. When the lights of a beacon are detected, the vehicle checks its personal target-type preference against the type indicated by the lights and also takes into consideration the indicated target density. Based on this information and on a small unconditional engagement probability, the agent either travels towards the targets indicated by the beacon or ignores the signals and continues motion.

Algorithm

Pseudo code of Beacon Based Task Allocation is provided under Algorithm 9. The algorithm starts by checking the amount of work done (line 2); if sufficient work has been done (compared to goal number of target processings) the algorithm stops and returns (lines 3 and 4). The check on line 6 is used when an AUV decides to become a beacon and consequently
activates the target surrounding behavior. During that phase, the AUV keeps track of the number of neighboring AUVs to determine the usefulness of being a beacon to other swarm members. Therefore, it uses a predefined time interval for counting neighbors then averages that number over the period. If the average is below a minimum acceptable number, the AUV gives up being a beacon and continues normal execution of BB-TA algorithm. This behavior is represented by lines 7-15. Lines 16-19 are used for building the raster scan sequence used by an AUV to sweep targets before it actually becomes a beacon. If an AUV decided to become a beacon and has just completed the sweep used for gathering information about the targets, it passes the check on line 20. At that moment, it starts surrounding the targets and turns the indicator lights on as described by lines 21-24. If at any time a target is found or it had decided to sweep but has not started yet, it does another set of checks to finally decide whether to process the nearby targets or look for others. This check is done on line 25. If the AUV did not decide to do a sweep and the sensed target is of the preferred type, there are no neighbors, it has already joined a beacon, or a small random target processing probability has been satisfied, it processes the closest target given that it has not done so previously (lines 29-31). Otherwise, it skips the target at hand and proceeds with the local search (lines 32-33). In the case where: 1) the AUV had decided to do a sweep but did not start yet, or 2) when there are no neighbors around, or 3) when there are neighbors but a small probability of becoming a beacon has been satisfied, and for any of these three cases, given that: 1) there are no beacons around, and 2) with a random probability of becoming a beacon, the AUV continues straight through the sensed targets until no more targets are sensed, then sweeps back. These steps are described by lines 37 to 52. If the scanned targets satisfy a minimum number constraint, the AUV starts surrounding them and becomes an actual beacon, otherwise, it behaves like a normal AUV. When an AUV encounters one or more beacons while not serving as a beacon itself (line 53), it checks their indicator lights
Algorithm 9 Beacon Based Task Allocation

1: loop
2:    if work_done ≥ goal_work_amount then
3:        Stop()
4:        return
5:    end if
6:    if target_surround_activated then
7:        elapsed_surround_time ← TrackElapsedNeighAvgTime()
8:        period_AUV_count ← UpdateSensedAUVsCount()
9:        if elapsed_surround_time ≥ averaging_period then
10:           period_AUV_avg ← CalcPeriodAvgAUVCount()
11:           if period_AUV_avg ≤ min_accept_count then
12:              GiveUpBeaconing()
13:        end if
14:    end if
15:    if first_entered then
16:        sweep_sequence ← GenerateSweepMotionSequence()
17:        first_entered ← false
18:    end if
19:    if sweep_completed and not target_surround_activated then
20:        SurroundTargets()
21:        TurnTargetIndicatorLightsOn()
22:        target_surround_activated ← true
23:    end if
24:    if target_found or (decided_to_sweep and not sweep_started) then
25:        rand_sel_prob_cond_achieved ← (UniformRandom() > 0.75)
26:        if not decided_to_sweep and
27:           (is_preferred_targ or no_neighbors or
28:            rand_sel_prob_cond_achieved or joined_beacon) then
29:            closest_target ← GetClosestSensedTarget()
30:        end if
31:        if closest_target ∉ P then
32:            ExploitTarget()
33:        else
34:            SearchLocallyForAnother()
35:        end if
36:    end if
Algorithm 9 Beacon Based Task Allocation (continued)

37: \(\text{beacons\_nearby} \leftarrow \text{CheckForNearbyBeacons()}\)
38: \(\text{neighbors\_around} \leftarrow \text{CheckNeighbors()}\)
39: \(\text{low\_prob\_cond\_achieved} \leftarrow (\text{UniformRandom()} \leq 0.05)\)
40: \(\text{rand\_sel\_prob\_cond\_achieved} \leftarrow (\text{UniformRandom()} \geq 0.5)\)
41: if (decided\_to\_sweep and not sweep\_started) or
   ((not neighbors\_around) or
    (neighbors\_around and low\_prob\_cond\_achieved)) and
    not beacons\_nearby and rand\_sel\_prob\_cond\_achieved) then
   if SensedTargetCount() > 0 then
      TravelStraight()
      decided\_to\_sweep \leftarrow true
      return
   end if
   if not sweep\_started then
      StartSweeping()
      sweep\_started \leftarrow true
      is\_beacon \leftarrow true
   end if
   end if
42: if beacon\_nearby and not is\_beacon then
43:   for each nearby beacon do
44:     lights \leftarrow \text{CheckLights()}\)
45:     prob\_joining\_med\_count\_achieved \leftarrow (\text{UniformRandom()} \geq 0.25)
46:     if preferred\_target\_type \in \text{lights} or
        large\_num\_targets \in \text{lights} or
        (med\_num\_targets and prob\_join\_med\_count\_achieved) then
        closest\_target \leftarrow \text{GetClosestSensedTarget()}\)
47:     if closest\_target \notin \text{P} then
        ExploitTarget()
48:     else
        SearchLocallyForAnother()
49:     end if
50:   end if
51: end for
52: end if
53: else
54: SearchLocallyForAnother()
55: end if
56: end loop
and based on the criteria on line 57, it either decides to join a specific beacon or elects to
continue searching for other targets (line 62). Line 68 describes the case when the AUV has
not found a target yet nor has it decided to sweep but did not start yet. In that case, it just
continues to look for targets.

3.7.4 Hybrid Task Allocation

This work proposes a hybrid task allocation approach that uses concepts from rule-based
and threshold-based autonomous task allocation control methods as categorized by [9]. By
combining mission-level requirements and sensory inputs, better-informed target processing
decisions can be made. The concept of the algorithm is presented in the next section.

Concept

The basic idea of the proposed hybrid approach is to use sensory information about neighbor-
ing vehicles, preferred, and other target counts along with mission requirements like elapsed
mission execution time, mission’s time constraint, and goal number of target processings for
current AUV to make decisions related to target processing. This is done by using adaptive
threshold functions (ATFs) that use these sensed counts and mission requirements as their
thresholds and stimuli, combined with rules that govern how ATFs are used and prioritized.
Table 3.4 shows the ATFs used in HYB-TA algorithm. \( f_1 \) is used to account for nearby AUVs
that are actively processing targets. \( f_2 \) accounts for the percentage of sensed targets that
are of the preferred (preset) type. \( f_3 \) becomes more likely to fire as the time an AUV has
spent to search for a target grows longer. In \( f_4 \), the percentage of targets processed relative
to the goal number is accounted for. The average of these four ATFs is used as the decision
mechanism for processing a nearby target or proceeding with a next-target search. If the
decision is to process a target, functions \( f_i \) are used to determine the type of target to be
Table 3.4: Thresholds and stimuli used by Hybrid Task Allocation. ATF: Adaptive Threshold Function.

<table>
<thead>
<tr>
<th>ATF</th>
<th>Stimulus</th>
<th>Threshold</th>
<th>Activation Based on</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1 = 1 - e^{-N_t/N_t}$</td>
<td>sensed static AUVs</td>
<td># of sensed targets</td>
<td>actively proc. AUVs</td>
</tr>
<tr>
<td>$f_2 = e^{-N_p/N_t}$</td>
<td># of pref. targets</td>
<td># of sensed targets</td>
<td>% of pref. targets</td>
</tr>
<tr>
<td>$f_3 = e^{-T_{el}/T_{elapse}}$</td>
<td>elaps. mission time</td>
<td>mission time constr.</td>
<td>elaps. search time</td>
</tr>
<tr>
<td>$f_4 = 1 - e^{-N_a/N_a}$</td>
<td>actual target proc.</td>
<td>goal target proc.</td>
<td>% of goal achieved</td>
</tr>
<tr>
<td>$f_i = e^{-N_i/N_t}$</td>
<td># of type $i$ targets</td>
<td># of sensed targets</td>
<td>% of target type $i$</td>
</tr>
</tbody>
</table>

processed. In this case, regardless of the values of functions $f_i$, priority is always given to preferred target type, if any. Only if there are no preferred targets around and one or more of these functions trigger activation of processing, another type may be processed. If counts of other sensed target types (for which activation was triggered) are different, the type with the highest count is given the priority, otherwise, a random type is selected for processing.

In the next section, the algorithm is explained in greater detail.

Algorithm

Algorithm 10 shows the pseudo code of Hybrid Task Allocation (HYB-TA). The algorithm runs in a loop (lines 1 and 28) which can be broken out of when sufficient work has been done or another criterion has been met. At the beginning of each iteration (line 2), stimuli and thresholds listed in Table 3.4 are updated based on sensor data and internal state variables of the vehicle’s controller. If the number of sensed targets $N_t$ is greater than zero (line 3), further checks are done to decide on target processing, otherwise, search for next target is continued on line 26. On line 4 adaptive threshold functions $f_1$ to $f_n$ are calculated using the thresholds and stimuli described in Table 3.4 and updated on line 2. Average of functions $f_1$ to $f_4$ is used as the activation threshold for target processing on line 5 and 6. If target processing is triggered (lines 7 - 21), type of target to process is determined according to priority and activations of functions $f_5$ to $f_n$ which are representative of percentages of
Algorithm 10 Hybrid Task Allocation (HYB-TA)

1: \textbf{loop}
2: \textit{UpdateStimuliAndThresholds()}
3: \textbf{if} $N_t > 0$ \textbf{then}
4: \hspace{1em} \textit{CalculateATFs}(f_1, ..., f_n)
5: \hspace{1em} Threshold $\leftarrow$ Average($f_1, f_2, f_3, f_4$)
6: \hspace{1em} \textbf{if} UniformRandom() $>$ Threshold \textbf{then}
7: \hspace{2em} \textbf{if} PreferredTargetCount() $> 0$ \textbf{then}
8: \hspace{3em} type\textunderscore to\textunderscore process $\leftarrow$ preferred\textunderscore target\textunderscore type
9: \hspace{2em} \textbf{else}
10: \hspace{3em} \textbf{if} AnyATFActivated($f_5, ..., f_n$) \textbf{then}
11: \hspace{4em} \textbf{if} AllTargetCountsEqual() \textbf{then}
12: \hspace{5em} type\textunderscore to\textunderscore process $\leftarrow$ SelectTypeAtRandom()
13: \hspace{4em} \textbf{else}
14: \hspace{5em} type\textunderscore to\textunderscore process $\leftarrow$ TypeWithLargestCount()
15: \hspace{3em} \textbf{end if}
16: \hspace{2em} \textbf{else}
17: \hspace{3em} SearchForAnotherTarget()
18: \hspace{2em} continue
19: \hspace{2em} \textbf{end if}
20: \hspace{1em} \textit{ProcessTarget}(type\textunderscore to\textunderscore process)
21: \hspace{1em} \textbf{else}
22: \hspace{2em} SearchForAnotherTarget()
23: \hspace{1em} \textbf{end if}
24: \hspace{1em} \textbf{else}
25: \hspace{2em} SearchForAnotherTarget()
26: \hspace{1em} \textbf{end if}
27: \hspace{1em} \textbf{end loop}

different target types. Otherwise, search for next target continues on line 23. On line 7, priority is given to the preferred target type if its count is greater than zero (regardless of whether other types’ ATF have fired or not). Only if there are no preferred targets around, ATFs $f_5$ to $f_n$ are checked for activation (line 10). If none of them has been activated, target search, again, continues on lines 17 and 18. For ATFs that fired, counts of targets of the respective types are compared (line 11). If all counts are equal (line 12) a random type is selected for processing. On the other hand, if counts are different, priority is given to the type with the largest count (line 14).
Algorithm 11 Blind Task Allocation (BL-TA)

1: loop
2: \textit{sensed\_targets} ← \textit{SenseNearbyTargets()}
3: \textit{N}_t ← \textit{Count(sensed\_targets)}
4: if \textit{N}_t > 0 then
5: \hspace{1em} \textit{ProcessClosestTarget()}
6: else
7: \hspace{1em} \textit{SearchForAnotherTarget()}
8: end if
9: end loop

3.7.5 Blind Task Allocation

This algorithm has been added only for comparison purposes and is very trivial. It does not distinguish between different target types and \textit{blindly} processes whatever target the AUV encounters. The pseudo code for the algorithm is very simple and is provided in Algorithm 11 only for completeness. If a target is encountered, it is processed, otherwise, search for the next target takes place.

3.8 Local Target Search

When the target has been found, AUVs have to significantly reduce their speeds and start special motion patterns suited for the limited area covered by the target(s). This local search can have direct impact on the outcome of the task identification and allocation processes; if the AUV does not carefully try to maintain appropriate proximity, it may end up missing the target or wasting time without finding the nearby target(s).

In this section, we introduce three local search algorithms that can improve AUVs’ ability to maintain target neighborhood and continuously find yet undiscovered targets. The first, \textit{Retracted-Sequence Random Walk} (RS-RW) uses a stack to keep track of a predefined count of arbitrary moves and to \textit{retract} these moves when a target is not found. \textit{Bubble-Chain}
Random Walk (BC-RW), uses the optimized same-position reorientation algorithm presented in Section 3.4.2 to maintain current location while making constrained jumps followed by returns to that location in case of failure to find targets in the jump's direction. The third and last local search algorithm is Tethered Random Walk (T-RW), which improves RS-RW by replacing sequence retraction with a direct return to original position using a tether similar to the one used in VTS. The three algorithms are described in detail in the next three sections.

3.8.1 Retracted-Sequence Random Walk

The development of this algorithm was motivated by the undesirable dispersive behavior that naive random walk algorithms exhibit. A mechanism for limiting the span a random walk algorithm covers while maintaining the random exploration behavior was required. To address this issue, a stack (with a length that can be selected according to the designer’s needs) was used to keep track of a sequence of turn-translate moves which were generated randomly. If after executing that sequence the AUV does not succeed in finding a target, it retracts the sequence, returning back to its original location. From there, it can generate a new sequence and the process continues. If a target is found, the stack is reset and the target is processed according to the rules of the task allocation algorithm used. After the processing the target, the process continues. Figure 3-28 illustrates the basic idea of RS-RW. There are four sequences shown with the names A through D. Each sequence consists of three subsequences numbered from 1 to 6, and each subsequence consists of a two parts with each being either a turn or a translation. At the end of each sequence, if the target is not found, the AUV performs a direction reversal turn and retracts the sequence back to origin. Path D shows that the vehicle successfully found a target.
Algorithm

Algorithm 12 lists the pseudo code of RS-RW. On line 2, the stack that holds the subsequences forming an active sequence of turns and translations is checked. If it is not full yet and a target is found, it is cleared (line 3) as there is no need then to keep track of the subsequences. Another check is done on line 5 for the case when the stack is full. If it is full, meaning that end of sequence has been reached without finding a target, the AUV reverses direction (line 6), the active sequence is cleared (line 7), and subsequences are retrieved from the stack and reversed to form a return sequence back to original location (lines 8-11). The resulting sequence is then executed (line 12). If none of the two previous checks evaluates to true, it means that the AUV is still executing a search. Therefore, it picks a random turn direction, turn angle, and a capped travel distance (lines 15-17) and uses them to create the next subsequence (line 18). This subsequence is then added to the active sequence and the stack (lines 19 and 20) and then executed (line 21).
Algorithm 12 Retracted-sequence Random Walk

1: loop
2: if not stack\_completely\_filled and target\_found then
3: \hspace{1em} Clear(stack)
4: end if
5: if StackFull(stack) then
6: \hspace{1em} ReverseAUVDirection()
7: \hspace{1em} ClearActiveSequence()
8: \hspace{1em} while NotEmpty(stack) do
9: \hspace{2em} reversed\_seq ← ReverseSequence(PopNextSubsequence(stack))
10: \hspace{2em} AddToActiveSequence(reversed\_seq)
11: \hspace{1em} end while
12: \hspace{1em} ExecuteActiveSequence()
13: \hspace{1em} return
14: end if
15: turn\_direction ← PickRandTurnDirection()
16: turn\_angle ← PickRandTurnAngleLessThan(\(\pi\))
17: travel\_dist ← PickRandDistLessThan(max\_dist)
18: sequence ← BuildSubsequence(turn\_direction, turn\_angle, travel\_dist)
19: \hspace{1em} AddToActiveSequence(sequence)
20: \hspace{1em} AddSeqToStack(stack, sequence)
21: \hspace{1em} ExecuteActiveSequence()
22: end loop

3.8.2 Bubble-Chain Random Walk

In RS-RW, exploration sequences are curved and relatively short-ranged. This causes occasional trapping of AUVs in areas where new targets are not close enough to be discovered by the application of the stacked sequences. To overcome this issue, instead of using curved exploration paths, straight lines terminated by direction reversals, when a target is not found, can be used. This allows exploring a wider area and speeds up the local search process. Each time the AUV returns, it can generate another random jump in a different direction until a target is found. This can be thought of as random sweeping in all directions. To still avoid being trapped in-place, longer-than-average jumps can be generated with a low probability. Additionally, with a low probability, the AUV can decide not to return from a jump. When a target is found, it is processed, then the process restarts. Figure 3-29 shows an example
Algorithm

Algorithm 13 shows the pseudo code of the algorithm. It is meant to be called within the vehicle controller’s update loop and, therefore, is called repeatedly. The algorithm starts by setting the vehicle’s speed to the at-target speed which is much lower than the global search speed (lines 2-5). If a motion sequence specified by the algorithm is currently being executed (lines 6-7), the algorithm just returns and is called on the next iteration. On line 8, a random distance in the range from $[0, \text{predefined}_\text{max}_\text{dist}]$ is generated and set as the distance to travel. A uniform random variable is then checked on line 9 to decide if this distance will be replaced by a longer one from a set of predefined distances to allow occasional escapes with a low probability (as shown on lines 10-12). Next, a coin flip determines whether to turn or continue straight to allow exploration of the opposite subspace to the one just swept (line 13). If it is decided to turn (lines 14-18), a random angle in the range $[0, 2\pi]$ is generated.

Figure 3-29: Bubble-Chain Random Walk algorithm (BC-RW) illustration.

illustration of how the algorithm works.
Algorithm 13 Bubble-Chain Random Walk

1: loop
2:     if not initialized then
3:         SetAUVSpeed(at_task_speed)
4:         initialized ← true
5:     end if
6:     if ExecutingSequence() then return
7:     end if
8:     dist_to_travel ← GenerateRandCappedDistance(predefined_max_dist)
9:     escape ← EscapeWithLowProbability()
10:    if escape then
11:        dist_to_travel ← PickDistInSet(S)  ▷ S is a set of predefined long distances
12:    end if
13:    result ← FlipFairCoin()  ▷ With a 50% probability, continue straight, i.e. don’t reorient to explore the other
direction
14:    if result = head then
15:        θ ← PickAngleInRange(2π)
16:        travel_direction ← RotateAUVHeadingDirBy(θ)
17:        ReorientAUV(travel_direction)  ▷ at same-position using the optimized
reorientation algorithm
18:    end if
19:    sequence ← AddDistToMoveSequence(dist_to_travel)  ▷ With high probability ...
20:    if UniformRandom() < 0.9 then
21:        sequence ← AddBackwardsTurnToMoveSequence(sequence)
22:    end if
23:    ExecuteSequence(sequence)
24: end loop

and used to generate a new heading direction that is in turn used to reorient the vehicle.

On line 19, the previously generated straight travel distance is added to the move sequence
that will be executed. With a high probability (lines 20-22), a direction reversion sequence
is added to the sequence to be executed. At the end (line 23), the built sequence is put into
action.
3.8.3 Tethered Random Walk

Tethered Random Walk (T-RW) is an extension to the two previously presented algorithms; it has the direction reversals (bubbles) that are used in both algorithms and sequences of turns and travels like in RS-RW. Additionally, it replaces the retraction in RS-RW with a tether (similar to the concept of VTS (Section 3.6.2)) to avoid following the same path twice and also expand the covered area during the search. T-RW also increases local search range using an adaptive threshold function based on the number of times it tried to find a target but failed. Figure 3-30 shows the paths followed by the AUV when it executes T-RW.

Algorithm

Pseudo code of T-RW is provided in Algorithm 14. Each search trial consists of three consecutive turn-travel sequences as in the case of RS-RW. The algorithm keeps track of the number of trials made with unsuccessful target finding. At the beginning of each iteration, a check is made to see if the 3 subsequences used to form a search trial have been performed (line 2). If this is the case, the number of failed trials is incremented by 1 (line 3) and the virtual tether tying the AUV to its original starting position is estimated using the sequence of 3 vectors denoting the start position of each subsequence (line 4). These vectors can be easily tracked using dean reckoning in a real situation. The vehicle then reorients towards home position using the optimized same-position orientation algorithm presented in Section 3.4.2 and travels straight for a distance equal to the tether length (lines 5 and 6). On the other hand, if the number of tried sequences is less than 3, the current sequence is cleared (line 9) and the parameters used for creating a new sequence are calculated (lines 10, 11, and 17). If the number of failed trials at that time is large enough to trigger activation of the ATF, the maximum travel distance cap (used in the new sequence) is increased (lines 12-16). The sequence is then executed (lines 19 and 20) and the vector pointing to the start
The position of this sequence is estimated and stored for later use in finding the tether direction (line 21).

In later performance analysis and algorithm comparisons, we use T-RW as the local search algorithm due to its advantages over the other two: 1) it allows exploring a larger local search area, 2) it searches faster than the other two, and 3) it has a learning ability. This does not mean that the other two are not as useful; they can be as useful if not more in applications that try to minimize energy consumption (BC-RW) or calculations (BC-RW and RS-RW).

### 3.9 Mission Integration

Individual mission stages need to be combined in a seamless way in order for the mission to be accomplished efficiently. Decisions regarding the appropriate times to start stage transitions...
Algorithm 14 Tethered Random Walk

1: loop
2: if executed_seq_count == 3 then
3:     $N_{nt} \leftarrow N_{nt} + 1 \triangleright N_{nt}$: Number of sequence executions with unsuccessful target finding
4:     tether_vector $\leftarrow$ EstimateTether(tether_sections)
5:     ReorientInPlaceTowardsDirOf(tether_vector)
6:     TravelStraight(Length(tether_vector))
7:     return
8: end if
9: ClearActiveSequence()
10: turn_direction $\leftarrow$ PickRandTurnDirection()
11: turn_angle $\leftarrow$ PickRandTurnAngleLessThan($\pi$)
12: resp_thresh $\leftarrow 1 - e^{-N_{nt}/N_{nt}^{max}} \triangleright N_{nt}^{max}$: Max. allowable number of sequence executions without successful target finding
13: max_dist $\leftarrow$ lower_default_max_dist
14: if UniformRandom() < resp_thresh then
15:     max_dist $\leftarrow$ higher_default_max_dist
16: end if
17: dist_to_use $\leftarrow$ UniformRandom() * max_dist
18: sequence $\leftarrow$ BuildSubsequence(turn_direction, turn_angle, dist_to_use)
19: AddToActiveSequence(sequence)
20: ExecuteActiveSequence()
21: tether_sections $\leftarrow$ tether_sections + CalcVectorPointingToStartOfCurrSeq()
22: end loop

and how to perform them are critical to the success and performance of the mission. For example, starting a formation (shape) change stage when close to the target in preparation for a load-balanced target-coverage can have a big effect on the overall mission time.

For these reasons, SWIM has a special package, integration, that provides stage integration algorithms. Currently, a simple BrainlessAllStageIntegration algorithm that, uses a predefined order of stages, runs the stages in that order and uses a notification mechanism that algorithms in the stage being executed use to notify it about their termination status. Another integration algorithm is currently being developed using MiniBrain learning model (cf. Section 3.11) in order to provide a more intelligent approach for task integration.

Due to the importance of this area of the mission planning process, it needs further
investigation and the development of a generalized approach that can be applied to any generic mission.

3.9.1 Mission Profit

To evaluate the overall performance of the mission, an appropriate measure is required. Because the resources available to each agent are limited, a mission profit (utility) function that takes into consideration the costs incurred by these resources needs to be formulated. We begin by assuming that a swarm of fixed size of $N$ robots densely packed at the source is released into the water, nearly simultaneously, at time $t_i$ (i for initial). We define overall mission time $T_m$ as the time between $t_i$ and $t_f$ (f for final) at which all surviving agents are successfully recovered. Percentage of mission completion as well as recovered agents are represented as fractions $R_c$ and $R_v$, respectively. The shorter the mission time and the larger the percentages of mission completion and recovered agents, the better the performance. In a time critical mission, which we assume is the case, a time constraint $T_r$ has to be respected. The ratio between the $T_r$ and mission time $T_m$ defines the degree of time compliance $R_t$. Mission profit or gain $G_m$ can be represented as in Eq. 3.34.

$$G_m = R_t R_v (\alpha G_d + (1 - \alpha) L_f), \quad R_v \in [0, 1]$$  \hspace{1cm} (3.34)

$$\alpha = \begin{cases} 0, & \text{if target not found} \\ 1, & \text{if target found} \end{cases}, \quad (3.35)$$

$$G_d = \frac{V_m}{C_m} = \frac{R_c V_{act}}{C_m}, \quad R_c \in [0, 1], \quad (3.36)$$

$$L_f = \frac{1}{V_{act} + C_m}, \quad (3.37)$$
where \( T_m = T_{ft} + T_{bt} + T_{ct} \), \( T_{ft} \) and \( T_{bt} \) are forward and back (return) trip times, respectively, \( T_{ct} \) is core task time, \( R_t = T_r / T_m \) is the degree of time compliance, \( G_d \) is the dollar gain, \( V_m \) is mission’s obtained value/profit, \( V_{act} \) is the actual value of targets being searched, \( C_m \) is mission monetary cost, and \( L_f \) is the mission-failure loss. Mission profit as just defined is suitable for case of a single target that can be considered as a whole (e.g. a ship) or consisting of parts of equal values (i.e. a homogeneous target). In \( Eq. \, 3.36 \), the quantity \( G_d \) represents the dollar gain for the single or homogeneous target just described. For a heterogeneous target consisting of different sub-targets (e.g. barrels of different types as we do in this work) or of parts of different values, the dollar gain becomes as given in \( Eq. \, 3.38 \)

\[
G_d = \frac{V_m}{C_m} = \frac{\sum_{i=1}^{n} R_{c_i} V_{act_i}}{C_m}, \quad R_{c_i} \in [0, 1], \quad (3.38)
\]

where \( R_{c_i} \) is the completed fraction of sub-target \( i \) and \( V_{act_i} \) is the actual value of that sub-target.

A good value for mission profit depends on the concrete mission’s specific parameters. This means that by knowing the target’s monetary value, costs of AUVs, operational cost, and the feasible time range for mission accomplishment, the maximum attainable mission gain can be estimated. That maximum can then be used as a reference to compare the actual mission utility to. This will become clearer in the case studies considered at the end of this dissertation, where complete missions are evaluated.

Now that individual mission stages and the associated decentralized algorithms developed in this work have been covered, we provide the background needed for the development of our brain-inspired learning model. In the next section, the basic concepts of Reinforcement Learning are presented. Many of these concepts are shared with MiniBrain and, hence, this introduction paves the road for the description of the model.
3.10 Reinforcement Learning

Reinforcement Learning (RL) is an unsupervised machine learning technique. It is concerned with agent-environment interaction and how an agent can learn through experience. It is characterized by rewards that the agent tries to maximize through selection of positively reinforced actions. Actions taken by the agent can have an effect not only on the short-term reward, but also on the long-term rewards as well. RL is especially useful in dynamic environments where the interaction is not always deterministic. In this work, we use RL as a basis for developing our MiniBrain model that adds a level of cognition to the AUV, enabling it to make better autonomous decisions. Therefore, it is important to cover the basics of RL in order to facilitate the development of the model in Section 3.11. This section provides a short introduction to RL to serve that purpose. It is based on the concepts and techniques explained by Richard S. Sutton and Andrew G. Barto in their book: “Reinforcement Learning: An Introduction” [88]. It is not meant to be comprehensive by any means and the interested reader is referred to that book for an extensive coverage of the topic.

3.10.1 Basic Concepts

RL has many important concepts that need to be introduced to enable clear understanding of the upcoming sections. These concepts are presented in the following sections.

Agents, the Environment, Rewards, Actions, and States

A reinforcement learning system consists of two main components: an agent that has specific goals and uses learning and decision making to achieve these goals, and the environment with which the agent interacts. The environment has different states that it switches between and offers new situations and rewards to the agent based on the selected action [88].
shows the interface and interaction between the agent and the environment.

Interaction can take place in continuous time as well as in discrete time or even at arbitrary decision making intervals. Considering the discrete case for simplicity, the environment will be in state $s_t \in \mathcal{S}$ at time $t$, where $\mathcal{S}$ is the set of all possible states. An agent can take an action $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ is the set of all actions available to the agent to select from in state $s_t$. In response to that action, the environment’s state changes and the agent is presented with a new state $s_{t+1}$ and a reward $r_{t+1}$ for selecting that action (where $r_t \in \mathcal{R}$, the set of all possible rewards). This reward and the new state are the new input to the agent and are used similarly to decide next step’s action. This process can continue indefinitely in continuing tasks or terminate, forming an episode, in episodic tasks.

**Rewards, Reward Function, Returns, and Discounting**

Rewards are a critical part of reinforcement learning; they need to be carefully designed in order for the agent to achieve the desired goal. Instead of rewarding the agent for doing part or all of the task at hand in a specific way, rewards should be given for actually achieving goals/subgoals that lead to the desired outcome. A very important aspect of rewards is that they must be unalterable by the agent; they are outside of its control and are presented by the environment. A reward is a numerical value that indicates the desirability of either a state or a state-action pair \[88\]. By a state-action pair, we mean selecting a specific action
when in some state. The mapping from different states/state-action pairs to rewards is done by a reward function.

As the goal of the agent is to maximize rewards in the long run, future rewards should be represented in a way that can enable us to define such long term reward. Return is used for this purpose and is defined (in its simplest forms) as the sum all future rewards in episodic tasks, i.e. tasks that have a terminal state \(s_T\). For continuing tasks, trying to sum of all future rewards can lead to an infinite reward. Therefore, the concept of discounting has to be introduced: a discounted return discounts a future reward proportionally to its distance in the future from immediate reward following action selection. This is done by weighting rewards by powers of a discount rate \(0 \leq \gamma < 1\) as given by Eq. 3.39. This discounting assures that return will always be finite as long as reward values are bounded. When \(\gamma = 0\), the agent is myopic meaning that it only tries to maximize immediate reward, while a \(\gamma \to 1\) strongly takes future rewards into account and the agent is farsighted. A combined representation of return that works for both episodic and continuing tasks can be obtained by summing the terms in Eq. 3.39 up to \(T\), the terminal state in an episodic task and using \(T = \infty\) for continuing tasks and \(\gamma = 1\) for episodic tasks as in Eq. 3.40.

\[
\begin{align*}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \\
R_t &= \sum_{k=0}^{T} \gamma^k r_{t+k+1}, T = \infty \text{(continuing)} \text{ or } \gamma = 1 \text{(episodic)}
\end{align*}
\]

Policy

An important concept in reinforcement learning is the policy that an agent uses to make decisions. A policy is a mapping from environment states to probabilities of selecting different actions. The agent continuously updates its policy attempting to reach the optimal policy.
Values, State-Value, and Action-Value Functions

A learning agent has to take actions at different times to achieve the objectives of the active mission stage. As not all actions lead to the desired progression towards the goal of that stage, a mechanism for scoring actions by rewarding or penalizing them based on the outcome of their selection is necessary. Recalling that rewards are the immediate results of actions in the reached next state, it is clear they are shortsighted and do not necessarily mean that, in the long run, future rewards will be maximized. On the other hand, if a state is followed by other states that have high rewards, this state is said to have a high value. Therefore, values represent long-term gains while rewards represent immediate, short-term gains. It is for this reason that we are more concerned with values than rewards as the agent’s goal is to maximize long-term rewards (i.e. values). While rewards are provided by the environment, values are not as easy to obtain because they are based on observations made by the agent throughout its entire lifespan.

*Value functions* are used for finding the expected returns and two types of these functions can be defined: *state-value* (or simply, *value*) functions and *action-value* functions. Both types are defined with respect to specific policies as rewards are associated with actions which are governed by a policy. *Value functions* are used for estimating the return of starting at a state \( s \) at time \( t \), following policy \( \pi \) afterwards. The value of state \( s \) under policy \( \pi \) is denoted by \( V^\pi(s) \). *Action-value functions* estimate the expected return for selecting an action \( a \) in state \( s \) at time \( t \) then following policy \( \pi \) afterwards and are denoted by \( Q^\pi(s, a) \). Eqs. 3.41 and 3.42 define value and action-value functions, respectively. Although these functions depend on future rewards, they can be estimated by keeping averages of past rewards that
were previously noticed starting from the respective states and following the policy or taking actions in these states then following the policy. As the number of times these states are visited approaches infinity, the averages will approach the real values of these states or state-action pairs. As then the number of times these states are visited approaches infinity, the averages will approach the real values of these states or state-action pairs.

$$V^\pi(s) = E^\pi\{R_t|s_t = s\} = E^\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\}$$

$$Q^\pi(s, a) = E^\pi\{R_t|s_t = s, a_t = a\} = E^\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\}$$

(3.41) (3.42)

The simplest way for estimating an action-value function $Q_t(a)$ of an action $a$ at time step $t$ is by averaging the received rewards $r_i$ of selecting that action over the number of times $k_a$ it was selected in the past [88]. Eq. 3.41 shows this average. The actual action-value function $Q^*(a)$ represents the reward received from the point the action is selected and on. $Q(a) \rightarrow Q^*(a)$ in the limit as $k_a \rightarrow \infty$.

$$Q_t(a) = \begin{cases} 
\frac{1}{k_a} \sum_{i=1}^{k_a} r_i, & k_a > 0 \\
0, & k_a = 0 
\end{cases}$$

(3.43)

Policy Evaluation (Prediction Problem), Policy Improvement, and Policy Iteration (Control Problem)

Policy evaluation is the process of computing the state-value function for a given policy. It is also called the prediction problem as it tries to predict the value of a state/state-action pair. One way of doing this is through the use of iterative policy evaluation, which uses the recursive Bellman equation (see [88] for more details) to refine a value function initialized to zero until a very close estimate $V$ of the actual value function $V^\pi$ is obtained.

Policy improvement is a natural step in reinforcement learning as the goal of computing
the value function of a policy is to try making the policy better. It is the process of coming up with a new policy that improves the original policy by making it greedy towards the former policy’s value function. One way to do it is by choosing the actions in each state that maximize the action-value function of that state.

Policy Iteration is the combination of policy evaluation and policy improvement. As explained above, the result of policy evaluation is a value function for that policy. In policy improvement, a new, better policy is obtained by making the original policy greedy with respect to this value function. By evaluating that new policy, a new value function can be obtained which can, in turn, be used to improve the policy. Policy iteration is this iterative process of evaluating and improving policies until an optimal policy is obtained. This is guaranteed when the policy space is finite as is the case in Markov Decision Processes (MDPs). Because this iterative process targets the estimation of the optimal policy, it is also called the control problem.

Generalized Policy Iteration

Interaction between policy evaluation and policy improvement can take many forms. One of them is as described in policy iteration in the previous section, where the two processes alternate and each completes before the other starts. Other forms include interleaving the two processes in different ways. The general process if interaction between policy evaluation and policy improvement regardless of the details of how this happens is called Generalized Policy Iteration (GPI). The result of interaction between policy evaluation and improvement is the stabilization of the value function and policy where no further improvements can be made, hence, the optimality of both the value function and policy.
3.10.2 Reinforcement Learning Methods

Reinforcement learning problems can be solved using many methods, among which Dynamic Programming (DP), Monte Carlo (MC), and Temporal Difference (TD) methods are the most famous. Dynamic Programming assumes a complete knowledge of the environment, meaning that a model of the environment is available, e.g. in the form of a Markov Decision Process (MDP), that provides the complete probability distribution underlying the transitions between different states. This is possible when the state space is limited, however, it requires high computational effort. For dynamic environments, where a model is difficult to obtain, Monte Carlo methods are better suited as they depend on experience whether based on on-line or simulated interaction with the environment. This experience provides samples of the transitions provided by the complete probability distribution in the DP case, but can still succeed in finding the optimal value function and policy. MC methods use return averages for value function estimation. One drawback of MC methods is that they are defined only for episodic tasks, that are guaranteed to terminate, in order to ensure well-defined returns [88].

An important difference between DP and MC methods results from the fact that DP uses a complete model of the environment while MC does not. In the absence of a model, policy evaluation needs to be done by estimating action-value functions instead of state-value functions. This is because models provide transition probabilities to different states and thus it is possible to select the action that provides the best reward-state pair starting from a given state. When no model is available, the best the agent can do is to evaluate actions through the exploration of state-action pairs and hence action-values are the way to go. This emphasizes the importance of exploring new actions in all states in order to enable policy improvement.

The third method, Temporal Difference (TD), is a combination of MC and DP meth-
ods. It does not require a model of the environment link in MC methods, i.e. learns from experience, and it calculates its value function estimates from other estimates during policy evaluation like in DP without having to wait until the end of the episode like in MC to know the return.

Now that basic concepts of RL have been highlighted, the development of MiniBrain learning model and its concepts should be straightforward. Details of the model are presented in the following section.

3.11 Mini-Brain: a Brain-Inspired Learning Model

This work suggests the use of a brain-inspired model, MiniBrain, to assist agents (AUVs) in making better, autonomous decisions that lead to the desired emergent behavior. Our model is meant to take Reinforcement Learning (RL) a step further by including agent’s intrinsics in the decision making process (in addition to agent-environment interaction discussed in the previous section). The model mimics to some extent the sequence of actions that take place in human brain during learning and interaction with real-world. Because decisions made by humans involve both logical and emotional aspects, our model uses these same aspects to select the final decision made by the agent.

Although the model is not used in this dissertation to generate results, we provide the design of its different components and their interaction and leave testing and scenario generation as future work. The model has been fully designed and implemented, and is currently being tested.
3.11.1 Analogy to Human Brain

Figure 3-32 shows the anatomy and functional areas of human brain. Taking a closer look at the functional areas, one can notice that there are four major areas that control most of the functions in the brain: 1) Motor Function Area (areas 3, 12, and 14), 2) Emotional Area (area 6), 3) Sensory Area (area 9), and 4) Higher Mental Functions (area 13). Remaining areas are related to these four major areas in one way or another. We pay special attention only to these four areas for simplicity of design and because they bear most of the functions necessary to make informed decisions by the agent and to control its actions and behaviors.

The main areas of MiniBrain are illustrated in Figure 3-33. As can be easily noticed, the four biological brain areas just discussed are mapped to four similar areas in Mini-
Brain. The figure shows two main interfaces: 1) AUV-Environment interface, and 2) AUV controller-MiniBrain interface. AUV’s controller is an example of the controller that any autonomous agent uses to decide between different states based on sensory inputs and is usually implemented as a Finite State Machine (FSM) as in this work. In the context of this dissertation, this controller is also where the algorithms used in different mission stages are run and continuously updated. The separation between the controller and MiniBrain makes the latter modular and allows it to be paired with any autonomous agent’s controller and easily integrated with the agent. To use MiniBrain, the controller calls a `makeDecision()` function to excite the brain and allow it to make a decision about actions that are later put into effect by the controller. Actions are executed by the controller because it has access to the physical actuators of the agent, however, the type of motor function performed by the controller is governed by MiniBrain. The second interface, between AUV and the environment, is physical (as opposed to the first interface which is of a more logical nature). This is the interface that distinguishes the agent as a physical entity from the surrounding environment. Through its sensors, the agent can sense the stimuli or triggers and track the rewards received after performing actions.

In this section, we describe the high-level flow of signals throughout MiniBrain and the main components shown in Figure 3-33 and leave the details of each component and the definitions of brain constructs to the following sections.

As mentioned above, the entry point to MiniBrain is AUV controller’s request to the brain to make a decision. This can be seen in Figure 3-33 at the Controller-MiniBrain interface. When MiniBrain is requested to make a decision, the first thing that takes place internally is that it excites the sensory area (SA henceforth). When excited, SA fetches sensor data from the AUV’s controller. It is the controller that actually interfaces with the environment, as would normally be expected, through its sensors and actuators. MiniBrain,
like in humans resides inside the head of the agent and receives its signals from the body and its senses. Here, the controller serves, to a certain degree, as the agent’s body. SA is connected to both the higher mental function (HMF) and emotional area (EA). As a result of functions performed internally within the SA, excitation signals are fired that, in turn, excite these two areas. The result of this excitation is the utilization of sensory data by the HMF which also takes into consideration the current dominant emotion in the EA to make decisions that can be either: purely logical, mixed, or purely emotional. Based on these decisions, HMF fires an action signal that excites the motor function area (MF) which it is connected to. When the latter is excited, appropriate motion-sequence or algorithm is activated and associated information is passed back to the controller to actuate the relevant motion/action. During their operation, different MiniBrain areas make use of a set of static- and dynamic-association maps. Some of these maps (the ones that lie outside any specific area) are shown on the left of Figure 3-33. Static and dynamic maps are marked with the symbols $S$ and $D$, respectively.
3.11.2 Definitions

Before proceeding with the details of MiniBrain areas and their interactions, some terminology and definitions of constructs and concepts need to be introduced. To begin with, we define a trigger or stimulus as:

**Definition 3.1.** (Trigger or stimulus) A predefined combination of sensory data (of both environmental phenomena and internal hardware components) and current states of certain system and time tracking variables define a trigger that can stimulate a mental state change.

Triggers used in the current implementation of MiniBrain take the form of a sense (like: multiple neighbors, no neighbors, neighbor collision, no targets, close to surface, etc.), time difference, reward size, action reinforcements (the count of positive or negative reinforcements of previously taken actions), or a battery level. These are summarized in Figure 3-34.

As can be seen from Figure 3-33, another brain construct that is associated with triggers is emotions (and feelings, which are not shown in figure). Although different definitions of feelings and emotions exist [47, 31, 26], which are sometimes contradicting, we use the definitions provided in Merriam-Webster dictionary in this work for simplicity. These definitions
Definition 3.2. (Feeling) “An awareness by your body of something in it or on it.”

Definition 3.3. (Emotion) “A strong feeling (such as love, anger, joy, hate, or fear).”

Again, despite the contradicting definitions that exist, we elect to use the definitions that associate feelings with body senses and emotions with the mental state that arises. This selection is more consistent with the scientific name of the the associated brain area (emotional area) that was presented previously.

In order to incorporate emotions and feelings in our MiniBrain design, The Feeling Wheel developed by Dr. Gloria Willcox [96] was used. To keep the design simple, only some of the feelings relevant to the agent and associable with data captured by its sensors were used. Figure 3-35 shows a simplified Feeling Wheel containing only the feelings used in MiniBrain.

In the design of the emotional area, a set of connected neural networks is used to represent the strengths of different feelings. These networks have different neural activity levels at
different times (EA design is explained in Section 3.11.5). We define a dominant emotion in terms of the neural activity of these networks as follows:

**Definition 3.4. (Dominant Emotion)** The emotion that is formed by the feeling with the neural network that has the largest neural activity is called the dominant emotion.

For the agent to be able to make a good decision about the action to take, it has to have a representation of the state of the environment. This representation can be obtained by examining the current situation. In the context of this work, a situation is defined as follows:

**Definition 3.5. (Situation)** The combination of trigger(s) and the resulting, stimulated dominant emotion define a situation in which the agent has to make a decision on how to behave.

When in a specific situation, the agent then has to make a decision with the goal of achieving its short- and long-term goals. The action taken by the agent, combined with the current situation define the behavior of the agent. Therefore, we define agent behavior as follows:

**Definition 3.6. (Behavior)** The action taken by the agent in a specific situation defines its behavior. i.e. a behavior is the combination of trigger(s), stimulated dominant emotion, and an action.

Figure [3-36] shows the relations between triggers, emotion, action, situation, and behavior.

Having defined the different constructs used by MiniBrain, the next few sections will explain brain areas and their associations in greater detail. We start by explaining the top-down structural transition from the top-level MiniBrain concept to its areas and subareas in the next section.
3.11.3 MiniBrain Areas and Subareas

It was shown in Section 3.11.1 that like in human brain, MiniBrain consists of brain areas responsible for its functions. From a structural point of view, MiniBrain can be represented as in Figure 3-37 (top) in terms of its constituent areas. The figure also is a functional representation in terms of the signals that MiniBrain receives and produces. The bottom figure further illustrates the structure and function of each brain area. As can be seen, each brain area consists of one or more subareas, works by receiving an excitation signal, doing some function, and producing another excitation signal to other areas to which it is connected. In the current implementation, MF, HMF, and SA all have a single sub-area each. EA is the only area that consists of multiple subareas as will be seen in Section 3.11.5.

3.11.4 Sensory Area (SA)

Sensory area is the entry point to MiniBrain. Its purpose is to communicate with the controller to extract sensory data and convert it into signals for exciting EA and HMF. Figure 3-38 shows the structure and function of SA. When MiniBrain is excited, it excites SA which requests its Trigger Synthesizer to synthesize triggers based on current sensory data. The synthesizer gets its sensory data from the controller by checking triggering events and their levels. Examples of triggering events include but are not limited to: last neighbor encounter, first target encounter, start of mission, start of search, etc. Trigger Synthesizer
therefore generates a list of triggers each formed by a group of signals (cf. Figure 3-34), having a priority, a start time, and a deadline for its satisfaction. SA then fetches this list of triggers and maps them to their associated emotions using a static Triggers-to-Emotions map that is heuristically predefined. After these mappings are found, the corresponding emotional subareas within EA are excited based on trigger priorities. This results an elevated level of neural excitation in some subareas more than others. The outcome is a specific dominant emotion as previously defined. Next, SA excites HMF area, passing it the set of triggers.

3.11.5 Emotional Area (EA)

Emotional area consists of six subareas corresponding to the six major feelings shown in Figure 3-35: Mad, Sad, Peaceful, Powerful, Joyful, and Scared. Each sub-area consists of a number of neural networks (2-3 in current implementation) as shown in Figure 3-39. Each neural network currently consists of five neurons. Neural Nets (NNs) in these subareas represent the different feelings that lie under the respective major feeling (again, as seen
from Figure 3-35. Every NN is a fully connected graph and holds only excitation level information corresponding to the feeling it represents. Individual neurons have excitation level information stored in them each, and the overall excitation level of a specific NN is the sum of excitation levels of all its neurons. Additionally, there are directed pairing connections between NNs in the same sub-area and undirected ties between different emotional subareas. These connections reflect the effect of some emotions on others and are used to excite target NNs when the neural excitation level in the paired NN or sub-area exceeds a specific level.

Figure 3-40 shows the general structure of a sub-area in the EA. The relationships between the considered sub-area and other sub-areas is signified through the pairing (left). For a given Neural Network in this sub-area, each neuron has its data content and neighbor connections. On the NN level, a NN is paired to associated NNs through pairing relations (right). It is worth noting that this structure can be used in all other three areas of MiniBrain (SA, HMF, and MF), however, in the current implementation, it was only used in EA for simplicity.

When the sensory area excites EA (cf. Figure 3-38), the excitation specifies the neural
Figure 3-39: Subareas, neural networks, and emotion-emotion interactions inside the emotional area of the MiniBrain.

Figure 3-40: General structure of an emotional sub-area.

network to excite in terms of the emotional sub-area and the targeted feeling. EA first extracts this information from the signal (see Figure 3-41) and uses it to excite the specified
NN. This is followed by random fading effects that are applied to every individual NN. This effect emulates the natural decay of excitation level over time and helps to neutralize emotions when not strongly excited for an extended period. The firing behavior is not employed in EA as it is more relevant to have HMF poll EA for current dominant emotion as will be seen in Section 3.11.6. For this reason, the firing signal is replaced by a polling functionality as shown on the right of Figure 3-41. When HMF polls EA for dominant emotion, EA first finds the NN with highest neural excitation (step (4) in figure) then provides the associated emotion (step (5)) to HMF.

3.11.6 Higher Mental Function (HMF)

This area has a critical role in the operation of MiniBrain: it combines logic with emotions stimulated in EA to make a decision about agent’s behavior. Structure and functionality of this area are shown in Figure 3-42. When HMF receives an excitation signal from SA, it first extracts triggers from the signal and polls EA for dominant emotion then combines them to build a situation (step 1 in figure). Next, it consults with the predefined mission goals (step
2) to get a mapping from the active mission stage to its associated action using the Goals-to-Actions map. That action is temporarily used in conjunction with the built situation to form a behavior (step 3 in figure). Active mission stage is then set in the controller (step 4) as the stage described by this long-term behavior (behaviors associated with mission stages are long-term as opposed to short-term behaviors which are triggered by emotions, triggers, and rewards). Additionally, active behavior is set to be this behavior. The behavior is then passed to the Behavior Assessor for assessment (step 5). If it was not previously assessed, triggers that were used to form the behavior are stored in a trigger registry for comparison with post-behavior-execution triggers. This enables checking whether triggers have be fulfilled or not. Long-term and short-term behavior scores are maintained by the behavior assessor as well as a mapping from each previously encountered situation to behaviors assessed (or being assessed) for that specific situation. This enables selecting the behavior with the highest score whenever the same situation is encountered again. In the assessment process, behavior assessor makes use of a static Actions-to-Expected Rewards map. That map associates a
reward of some value to each of the actions available for selection. Given the expected rewards and the rewards sensed from the environment in addition to trigger fulfillment information, reward assessor is able to score a specific behavior. After the predefined mission stage is used to define the long-term behavior to perform, other checks are made to determine if a short-term behavior (e.g. emergency behavior) needs to be triggered based on the current status of resources like battery level, etc. and elapsed mission time (steps 6 and 7). If none of these checks trigger the activation of a short-term behavior, emotions, other types of triggers, and rewards are checked. If all these checks fail to activate any short-term behaviors, the active mission stage’s behavior (the predefined long-term behavior) is chosen for execution. In the case of short-term behavior selection, behavior is assessed/reassessed by the behavior assessor (step 8). For long-term behaviors, a behavior is assessed at regular intervals to measure its progress towards its goal, while for short-term behaviors, assessment is done only when the behavior has ended. At the end (step 9), HMF builds an excitation signal containing the action to be taken and passes it to the MF to be executed. It is worth mentioning that the functionality of the HMF just described results in decisions that can be purely logical or emotionally affected based on the stimuli received from the environment; this behavior mimics human decisions, which is the goal of designing MiniBrain.

3.11.7 Motor Function (MF)

Motor Function area (Figure 3-43) is responsible for translating the actions selected by HMF area into a “language” that the controller can understand and use for actuating the desired behavior. When MF is first created, the dynamic Actions-to-Motor Functions map is created (step 0). The map is dynamic because it allows adding new mappings when the agent’s dominant emotion is Creative. This allows expanding the dictionary of ways in which actions can be preformed giving the agent the ability to learn previously unknown behaviors
that are then assessed for performance and can be reused. When the HMF excites MF with a signal containing information about the action to perform, MF first extracts this information from the signal (step 1). After extracting the information, the above mentioned map is used for mapping the action to a motor function (step 2). Next, motor function parameters are extracted from the resulting motor function and used to select and adapt the parameters of the final action to be performed (step 3 and 4). Motor function parameters are: motion pace, duration, motion periodicity, direction, and motor function type as shown in Figure 3-44. Motion periodicity specifies whether a motion is periodic or aperiodic and how many repetitions in the former case. Motor function type can be a composite turn sequence (a sequence of turns and straight travels) or a mission-stage algorithm (e.g. global search, task allocation, initial self-organization, etc.). If a composite turn sequence is selected, the sequence is retrieved from the Sequence Repository, which has a set of predefined sequences, but similar to Action-to-Motor Function map, it is dynamic to allow adding new sequences. For both, negatively rewarded entries are removed to avoid unbounded growth in their size. On the other hand, if a mission-stage algorithm (which defines a long-term behavior) is
selected for execution, information about the algorithm is passed to the controller (where these algorithms are defined) for execution. After a sequence or an algorithm is selected, AUV controller actuates it (step 5). As may be noticed from figure, similar to EA, MF does not produce an excitation signal as it directly instructs the controller to perform the selected action.

### 3.11.8 Relevance to Swarm Robotics

In the previous sections, the design and operation of MiniBrain were presented. We now reflect on the relevance and applicability of this approach to Swarm Robotics. During the design and testing of the algorithms presented in this work for the different mission stages, it became very obvious that the execution of fixed, predefined motions in a dynamic environment like oceans will always have its limitation: the inability of the agent to adapt to that dynamism degrades its performance in all stages and at the mission-level. As presented in Section 3.10, Reinforcement Learning adds an extra layer of cognition to the agent that enables it to learn from its previous actions based on observed rewards in a completely autonomous way. Undoubtedly, Reinforcement Learning has been of significant benefit in many
applications due to the learning “powers” it equips agents with. In an attempt to expand these benefits and equip agents with human-like cognition, MiniBrain was proposed. It does not replace RL but, instead, extends it by enabling cognition within the agent itself in addition to the classic agent-environment interaction and interface defined by RL. By mimicking human-brain internal interactions, we are hoping to reproduce human-like behaviors that assist in achieving the ultimate goal of the mission under consideration. From a practical point of view, the implementation of MiniBrain is very realistic as it has been already implemented and is currently being tested in simulation. Our SWIM simulator, was used and it was possible to test with multiple AUVs. Due to the continuously lowering single-board PC prices like BeagleBone Black and Raspberry Pi, and their rapidly improving performance, it is now possible to build intelligent agents at very low prices. We, therefore, anticipate that in the very near future, large swarms running MiniBrain and MiniBrain-like frameworks will be as popular as cellphones nowadays.
Chapter 4

Results and Discussion

In this chapter, individual global search and task allocation algorithms are tested and their performance is analyzed. Groups of algorithms of the same stage are then compared to evaluate their relative performance. This is followed by a set of case studies that test complete missions integrating different combinations of these algorithms. Each case study considers a different target configuration. For all case studies, mission utility is calculated to measure the degree of mission success.

4.1 Experimental Setup

SWIM simulator was used to test and validate all the algorithms presented in this work. A stage with dimensions 256 x 256 x 25 (width × length × depth) World Units (WUs) was used. AUVs equipped with five directional sensors pointing in the AUV’s north, east, west, south, and downward directions, were used. The AUVs were released from a cubic-shaped virtual cage from a height of 90 WUs. When they reach a depth below 10 WUs, they start to execute the algorithm under consideration. Two types of targets were used in different tests: a ship and barrel groups. Additionally, barrel groups consisted of three types: red,
green, and blue. Each of these targets had a monetary value associated with its processing. Random and fixed locations of targets were used for testing in different scenarios and swarm drop-off location was either at origin or at (-30, 90, 0).

To provide a fair comparison between algorithms, simulation length was set to 10,000 ticks. Each algorithm was run and allowed to find the target within that time span. 25, 50, and 100 AUVs swarm sizes were used for each algorithm with the simulation run 10 times for each swarm size. The same drop-off and target locations were used for all algorithms. Additionally, whenever an AUV reached the boundaries of the search stage, it was detached and considered lost. Although in a real environment, those boundaries are not present and the algorithms that guarantee boundedness within an area of limited radius would typically have zero losses, we elected to only point this fact out when relevant to keep a fixed comparison base. Multiple performance metrics were used and results were automatically stored at the end of each simulation. The following section describes the metrics used and why they were selected.

4.2 Performance Metrics

To evaluate the six global search algorithms and compare their performance, seven metrics were defined: average distance to target, average spread, number of lost AUVs, number of AUVs that reached target, numbers of sent and received messages, and overall combined distance traveled by the swarm.

- **Average distance to target:** $\frac{1}{n} \sum_{i=1}^{n} d_{it}, d_{it}$: distance from AUV $i$ to target. This metric measures how the swarm progresses towards target over time (simulation ticks).

- **Average spread:** $\frac{1}{n} \sum_{i=1}^{n} \|p_i - p_{avg}\|, p_i$: AUV $i$’s position, $p_{avg}$: average position. The purpose of this metric is to determine if the swarm maintains its unity and does not get
dispersed and lost.

- **Numbers of lost AUVs & ones that reached target:** for fairness of comparison, the AUVs that reach the boundaries of the stage are considered lost. Number of AUVs that find target is also tracked. The algorithms that suffer from large count of lost agents are undesirable as they increase the cost of the mission. High percentage of AUVs reaching the target indicates the efficiency of the algorithm.

- **Number of sent & received messages:** used in algorithms that use communication either as an essential part or to improve performance. Communication should be minimized, therefore, no or limited communication is highly desirable.

- **Overall distance traveled:** this measure combines the distances traveled by all AUVs in the swarm throughout the whole mission. It is a good indicator of how efficient the mission is as it is a direct reflection of energy consumption.

In the following section, we present the results of the global target search mission stage. The simulations were run in the **Search-Mode** which only tests the search algorithm under consideration in a target finding scenario.

### 4.3 Target Search

In this section, we compare the six search algorithms presented in Section 3.6. We start by analyzing each algorithm individually then do the comparison at the end of the section. Performance measures presented in Section 4.2 do not apply for all algorithms as, for example, some of them don’t use communication at all. The measures relevant to each algorithm will be studied and the ones common for all algorithms will be used in the final comparison.
4.3.1 Performance of VTS

Because of the initial drop-off location and target placement, part of the search area of VTS crossed the boundaries of the search platform. This resulted in a percentage of lost AUVs for all swarm sizes. As the probability of reaching the part of the search area which exceeds the boundaries increases with swarm size (because VTS distributes swarm members uniformly throughout the search area), the chance of losing AUVs increases with swarm size. Therefore, the number of AUVs that were lost was proportional to swarm size. This of course affected the number of vehicles that reached target. Another factor affecting that number is the unified simulation tick count used to terminate the simulation as mentioned in the previous section. Due to these two factors, the number of AUVs that reached target for the 25 AUVs swarm was 6.8% larger than the 50 AUVs swarm. The latter was 2.4% better than the 100 AUVs swarm. These differences closely match the differences in percentages of alive AUVs by the end time of the simulations, which were: 8% and 2%, respectively. The small mismatch is due to the percentage of alive AUVs that did not reach the target by the end of the simulations. The percentages of at-target AUVs and the numbers of lost AUVs are shown in Figures 4-1, 4-2, and 4-5. As can be noticed from Figure 4-1, the number of AUVs that reached target grew logarithmically for all swarm sizes. It is worth mentioning that in a truly open environment, the above reported loss is not expected to occur for VTS as the AUVs reported as lost in the simulation were only considered so because of exceeding the platform’s boundary; VTS guarantees that tether will keep the vehicle within the predefined search area until it eventually finds the target.

To better understand how the algorithm behaves, the average distance to target, shown in Figure 4-3 should be considered. It is clear that regardless of swarm size, the average distance decreases exponentially over time. All swarms succeed to bring their AUVs close enough to the target such that the average distance at the end of the simulation is below 20
World Units (WUs henceforth).

Figure 4-4 shows the average spread of swarm members. The transient at the beginning of the curve is due to the initial drop-off. This transient will be noticed for all algorithms. In the simulation, the swarm is released from a cube that mimics release from a large cubic cage. When the swarm is initially dropped the 3D distances between AUVs shrink rapidly as this cubic pattern collapses into an almost 2D square when they hit the surface of the water. This is the reason for the first part of the transient. This is followed by an initial dispersion of agents, which is the reason for the second part of the transient (temporary increase in spread).
After this initial transient, the AUVs start to spread uniformly throughout the search area causing the average spread to even out. Starting at around 1337 simulation ticks, the spread starts to decrease slowly as more vehicles start to find the target, condensing there, and hence pulling the average spread down.

It should be clear that although the average spread decreases over time, this decrease is very limited. This is attributed to some AUVs still searching around at random locations within the search area by the time 10000 ticks were reached. If the simulation were allowed to continue until all AUVs reached the target, an abrupt drop in spread would be observed.

This high average spread can be looked at from two different views: one can criticize that swarms should travel in cohesion by flocking together, while others may see this unnecessary as long as the swarm will reach the target and cooperate to do the task. We try to reach a middle ground here by pointing out that this application-specific. For example, if the application requires performing intermediate tasks along the way to the target, VTS would not be the best choice. On the other hand, if no such intermediate work is needed, there is no reason to force the swarm to move in unity, especially if no commutation is needed as in VTS.
The overall distance traveled by the swarm is shown in Figure 4-6 for the three swarm sizes. It can be noticed that as the swarm size doubles, the distance increases four times, i.e. there is a quadratic increase in distance. Distance growth over time is logarithmic though, as more agents find the target and their contribution to distance growth stops.
4.3.2 Performance of CSF

CSF has the characteristic that it relatively delays target finding, but then allows the successful swarm members to find it in-bulk within a narrow time window. This can be seen in Figure 4-7 by noticing that, in the period from 1337 to 4677 ticks (i.e. in only 3341 ticks), all swarm sizes managed to transfer the successful fraction of swarm members to the target. The growth in the number of AUVs that reached target is almost linear within that window, with different rates for different swarm sizes; the bigger the swarm, the faster the rate. This happens because of the flocking effect that keeps swarm members together, therefore, once a subgroup of the swarm starts to find target, the remaining groups follow quickly. The nonlinearities noticed within the aforementioned window (for a specific swarm size) are attributed to segmentations in the swarm; due to the inability of all swarm members to catch up with others, a few sub-swarms may result, but with small distance between them. For example, for the 50 AUVs case (red curve), two sub-swarms were formed and the second closely followed the first in target finding. For the 25 AUVs case (blue curve) flocking effect was not strong enough to keep a strong cohesion between AUVs. Consequently, target arrival is spread over a period causing it to be relatively slower compared to the other two
By observing Figure 4-7, it can be noticed that the average number of AUVs that reached target are 14, 27, and 46 for swarm sizes 25, 50, and 100, respectively. This shows that nearly half of the swarm did not reach target for all swarm sizes. Figure 4-8 shows that the average number of lost AUVs is 4, 11, 32 for swarm sizes 25, 50, and 100, respectively. This is only the case because those AUVs reached the boundary of the search platform and, hence, were discarded. As in VTS, this would not happen in a real open environment and should not be considered at all as a limitation of the algorithm (as will be seen in the case studies). Similar to VTS, CSF assures the survival of the whole swarm (except in very rare cases where few agents may go astray by facing opposing agents when they fail to sync with the swarm and, as a result, the velocity update vector is updated in a way that leads to a complete reversal of direction leading to that straying. Dealing with this scenario is left as future work although it can be greatly minimized by applying the initial self-organization algorithm presented in this work). Different from VTS, CSF brings unsuccessful AUVs back to the original drop-off location. Therefore, the loss percentage is expected to be very negligible in practice if at all.
Figure 4-9 summarizes the percentages of alive, lost, and successful AUVs for different swarm sizes. As would be expected, the larger the swarm the more probable that a larger portion would touch the boundary causing a larger loss percentage. The percentage of successful AUVs is linearly proportional to the percentage of alive AUVs. This means that swarm size did not have effect on the percentage of AUVs that reach target, however, as previously pointed, it has an effect on the rate by which the swarm members find the target. From the search point of view, this indicates that increasing the swarm size does not carry much value to the count of successful AUVs, but from the task allocation perspective, it means that more AUVs will be available for accomplishing subtasks almost simultaneously. This is a great value for task allocation. It is worth noting that this is expected as CSF’s behavior is mostly deterministic: if the target is within the spiral’s coverage and there are enough AUVs to compensate for its logarithmic expansion, the target will be found.

One additional advantage of increasing swarm size can be observed from the blue curve in Figure 4-7: swarm size has a direct effect on the speed of finding the target (first target encounter). Swarm sizes 50 and 100 were able to find the target around 840 time steps before the 25 AUVs swarm.
Average distance to target (Figure 4-10) initially increases almost linearly, with some fluctuations caused by the spiral pattern as the AUVs swirl alternately towards and away from the target, and then decreases rapidly as swarm members reach the target with very short time differences. As pointed out previously, swarms of sizes 50 and 100 AUVs succeed to find the target faster than the 25 AUVs swarm. Distance changing pattern for the two bigger swarms proposes that based on the expected task size (at the target), one of the two sizes may be preferred to the other taking into consideration the available budget as well. The small sine-like waves at the tale of the curve are caused by the unsuccessful AUVs that return to the drop-off location and keep circling in-place waiting for pickup. As they move on the closer side of the circle to the target, they lower the average slightly and when they are on the far side of the circling path they increase average. All swarm sizes share a final distance around 33 WUs to the target. The reason for the higher distance (compared to VTS for example) is that the AUVs that return back to the drop-off location pull the average up. In the case of VTS, AUVs don’t return to drop-off location.

Similar to average distance to target, average spread first increases almost linearly with fluctuations, but at a much faster rate. This increase is attributed to the exponential growth
rate of the spiral and the inability of flocking forces to maintain strong cohesion between
swarm members. Therefore, they continue spreading until they reach their maximum spread
at the maximum radius of the spiral. If the target is found, AUVs start accumulating
rapidly at the target (cf. Figure 4-7) causing the average spread to start falling at a rate
equal to the accumulation rate minus the rate at which unsuccessful AUVs depart towards
the drip-off location. Once the successful vehicles have complete accumulation at the target
and unsuccessful ones reach the original drop-off location, the spread stabilizes at a value
determined by the distance between target and drop-off location and radius of the in-place
circling at the latter location as shown in figure. All swarms had a final spread value of
around 47 WUs.

As the AUVs find the target, they start broadcasting messages notifying nearby AUVs
that haven’t sensed the target yet of the its presence. Figure 4-12 shows that the number of
sent messages starts to grow linearly as vehicles start to accumulate at the target. The rate
of growth is proportional to swarm size. Figure 4-13 shows the average numbers of received
messages over time for the three swarm sizes. It can be easily noticed that these averages
closely follow the number of AUVs that reached target (Figure 4-7), which is intuitive: when
Figure 4-11: CSF – Average spread.

Figure 4-12: CSF – Number of sent messages.

an AUV is in communication range of broadcasting AUVs, its receives a message, knows the target is nearby, slows down and eventually finds it causing the average number of AUVs that reached target to grow alike. The number of sent messages is much larger than received ones as the vehicles at the target broadcast at a constant rate, while messages are received only when there are nearby AUVs.

Overall distance traveled by different swarm sizes is shown in Figure 4-14. It is relatively high especially for the 100 AUVs swarm. The average distance increased around fourfold by doubling the size from 25 to 50 AUVs and sixfold by doubling from 50 to 100 AUVs. The
cause of this increased overall traveled distance is attributed to the exponentially growing nature of logarithmic spirals used by CSF.

4.3.3 Performance of R-PSO

R-PSO did not perform very well in terms of the number of AUVs that reached target (see Figures 4-15 and 4-16). The average numbers of AUVs were: 35 (35%), 18 (36.4%), 10 (38%) for swarm sizes 100, 50, and 25, respectively. This poor performance is explained by the large number of AUV losses that take place early in the simulation as can be seen in
Figure 4-15: R-PSO – Number of AUVs that reached target.

Figure 4-16: R-PSO – Number of lost AUVs.

Figure 4-16 This loss happens at the fastest rate for the largest swarm and with slower rates as the swarm size decreases. The reason is that in R-PSO, AUVs follow a gradient to locate the target and there is no characteristic bounding behavior in the algorithm itself. Therefore, when the swarm is first released, the AUVs spread in all directions based on their individual initial orientation and only the ones that happen to encounter the gradient are those which are likely to survive; others disperse and get lost very quickly. Even the ones that find the gradient are not guaranteed to find the target and not get lost.

Figure 4-17 summarizes the average percentages of AUVs lost, surviving, and that reached
target. 25 and 50 AUVs swarms had similar percentages of loss, while the 100 AUVs swarm a slightly lower percentage. The average percentages of target arrivals are very similar across all swarm sizes, but decrease slightly with increase in swarm size. The actual reason for this decrease is not known and needs further investigation.

In general, average distance to target decreases very slowly in R-PSO as shown in Figure 4-18. The transient at the beginning happens because just after drop-off, AUVs travel fast in different directions before some of them start sensing the gradient (in the simulations the drop-off location was close to, but not inside the gradient). Therefore, the AUVs that are initially oriented towards the target pull the average distance down quickly as they travel towards it before reaching the gradient. At the same time, there other vehicles that are traveling in directions completely opposite to the target’s direction. These latter vehicles try to pull the average up. This “competition” produces the ripples appearing on that transient part of the curve. After the departing vehicles leave the search platform (and get discarded), the average starts to smooth and decrease slowly as more AUVs find the target. The ripples on the long tale of the curve are caused by the continuous turns in different directions that the vehicles do as part of their position updates as dictated by the intrinsics of the PSO.
algorithm. It can be noticed that the average distance is higher (for the flat part of the curve) for the 100 AUVs swarm than the 50 AUVs swarm, and the same for the 50 and 25 AUVs swarms. The cause of this increased average for the first swarm in both cases is probably driven by the wider area which larger numbers of AUVs cover during their search in the case of larger swarms. This interpretation still needs to be validated.

The initial spread (Figure 4-19) falls rapidly again because of the drop-off effect as mentioned before. After this decrease, an increase follows because of the dispersion of the swarm outwards from the drop-off location. When AUVs head away from the gradient (leading to the target) reach the boundary of the search platform and are discarded, and as the remaining ones start to slow down after reaching the interface with the gradient, the spread starts to decrease again. The vehicles that reached the boundary of the gradient now start following the increase in gradient from different directions causing the spread to grow slightly again (period from $\sim 1337$ to $\sim 4009$) and then stabilize as they continue applying position update rules. The average spread remains nearly constant until the end of the simulation because at the termination time there were still many AUVs that did not find the target, pulling the average spread up. It should be noticed that the steady state average spread is high ($\sim 50$
WUs). This goes hand-in-hand with the curves in Figure 4-15 in demonstrating the slow convergence of R-PSO.

R-PSO uses communication as a main component of the algorithm. Figure 4-20 shows that average number of sent messages grows super-linearly over time for all swarm sizes, with that number increasing sixfold by increasing the swarm size from 25 to 50 AUVs and tripling when swarm size increases from 50 to 100 AUVs. The average number of received messages on the other hand increases logarithmically, with each doubling of swarm size causing approximately a four times increase in number of messages received over time. The logarithmic increase again closely follows the curves showing the average numbers of AUVs that reached the target over time (Figure 4-15), showing that finding the target is correlated to the number of received messages (which indirectly contribute to directing the AUVs towards the target).

The average overall distance traveled by the swarm in case of R-PSO is relatively large, especially for larger swarms. The curves in Figure 4-22 show the logarithmic growth of average overall distance over time. The distance increases nearly fivefold with each swarm doubling. This large average is caused by the way R-PSO works; each AUV continuously updates its position and reorients based on previous best and local best positions. This
A continuous update of position causes the path followed by each vehicle to grow quickly over time leading to an increased overall average.

4.3.4 Performance of SDHCP

In SDHCP, the average number of AUVs that reached target increased almost linearly with time within a relatively short window of time (\(\sim 1003 \rightarrow \sim 5334\)) for all swarm sizes. As SDHCP works by allowing AUVs to follow a hexagon with growing side length (with fixed growth step-size), where AUVs travel around the current hexagon either clockwise or counter-clockwise, once a group of AUVs finds the target, subsequent groups follow very closely. Additionally,
because all groups move on the perimeter of the same active hexagon, the delay between group arrivals is small. These two factors contribute to the linear growth in average number of AUVs reaching target observed in Figure 4-23. The final value of that average for swarm sizes 25, 50, and 100 were: 11 (45.2%), 25 (49%), and 47 (47%), respectively. There is a point to clarify here: similar to CSF, the low numbers of AUVs finding the target (and the high losses shown in Figure 4-24) are not caused by the inability of SDHCP to deliver the swarm to the target, instead, they are caused by the fact that the maximum side-length hexagon intersected with the boundary of the search platform, resulting in large losses. Figure 4-25 shows that different swarms had very close percentages of lost, surviving, and at-target AUVs. In a truly open environment, SDHCP is guaranteed to either deliver some/all swarm members to the target and return unsuccessful ones back to the drop-off location. Therefore no loss should happen in the absence of adverse conditions. It should be observed from Figure 4-23, 4-24 and 4-25 that the percentage of AUVs that reached target is linearly proportional to the average number of alive AUVs (around 2% less or around 95% of them) for all swarm sizes.

By closely studying Figures 4-26 and 4-27 jointly, it can be observed that the average distance to target as well as average spread have an alternating growing and shrinking
pattern (the spikes) that matches across the two curves for each swarm size. When a swarm transitions from one hexagon to the next bigger, the average spread and distance from the target increase. When that swarm follows the perimeter of the reached hexagon, the spread and average distance decrease causing the spikes on the curves. Studying the number of lost AUVs (Figure 4-24) along with these two curves, one can notice that starting at around the third spike, the spread grows on a certain spike until an increase in loss rate takes place (when AUVs start getting lost rapidly at the boundary), then decreases rapidly due to the loss of far AUVs. The average distance to target also drops as a consequence. The alternating
Figure 4-25: SDHCP – Percentages of surviving, lost, and AUVs that reached target for different swarm sizes.

Figure 4-26: SDHCP – Average distance to target.

increase-decrease in average distance and spread continues until all remaining AUVs reach the target. At that point, the average distance to target falls to its minimum (∼17 WUs) and the average spread stabilizes at ∼49.5 WUs. The initial increase in average spread’s curve for all swarm sizes is due to the expanding behavior of the growing hexagons. Global drop in spread is triggered by more AUVs reaching the target as observed from the average distance to target curve.

Number of sent and received messages follow a pattern similar to those in CSF for the
same reasons provided in Section 4.3.2. Figures 4-28 and 4-29 show the change in average number of sent and received messages over time for swarm sizes 25, 50, and 100.

The overall distance traveled by different swarms is high in SDHCP. The reason stems from the design concept of SDHCP: it uses close-packing principles to cover all points of the search space and these principles are applied by each AUV. This leads to long traveled paths by individual AUVs, and consequently the swarm. The average overall distance increases fivefold by doubling the swarm from 25 to 50 AUVs, and around sevenfold for the doubling from 50 to 100 AUVs. The growth over time follows an S-curve as the start hexagon is
relatively small, then it starts to build up linearly, followed by a saturation stage when the AUVs start accumulating at the target. This large overall traveled distance is the downside of SDHCP, however, if the application requires such high resolution coverage of the search space, SDHCP is a good choice.

**4.3.5 Performance of SSW**

SSW can deliver the whole swarm to the target fast, especially for large swarm sizes (100 and above). This, however, is contingent to the presence of the target in the sweep direction,
otherwise, sweeps in all four major directions are required to guarantee target finding. As can be observed from Figure 4-31, the last AUV in the 100 AUVs swarm found the target on average at 3885 ticks. For all swarm sizes, the steps on the curve correspond to intermediate time between arrivals of closely subsequent swarms. Segmentation occurs because of slight initial displacements as well as errors in measuring magnetic north direction (the common base for sweep direction calculation). In all cases, the whole swarm was able to reach the target (Figure 4-32), which is the exceptional aspect about SSW. However, as pointed above, if the target is not in sweep direction, 0% of the swarm will find it, therefore, it is a a double-edged sword.

Figure 4-33 shows the change in average distance to target over time for different swarm sizes. Swarms of sizes 50 and 100 have very similar distance decrease rates, while the 25 AUVs swarm has a lower rate. All the swarms share the spiky pattern resulting from the sweeps performed. The spikes emerge because a single sweep will have a point at which the distance to target is minimum and other sections at which the distance is maximum. The overall trend is a decreasing average distance as more and more AUVs reach the target. The smallest swarm size suffers from a delay compared to the other swarms because it has

Figure 4-31: SSW – Number of AUVs that reached target.
a smaller thickness, making its average distance to the target always larger than the other two. Additionally, the errors in magnetic north estimation have a more significant effect on the spread (cf. Figure 4-34), increasing the probability of having a larger average distance due to the effect of far AUVs. In larger swarms, these errors cancel each other, reducing the effect on the average. The 50 AUVs swarm has some delay too in delivering the last AUVs to the target, but not as long as in the 25 AUVs case.

Average spread (Figure 4-34) initially decreases as the AUVs reorient towards individually-estimated magnetic north to align themselves and start sweeping as a group. During the
Figure 4-34: SSW – Average spread.

sweeps, spread relatively increases due to the displacements between swarm members as they travel straight along the sweep direction. At the end of each sweep, the AUVs slow down to perform a 180 degrees turn. This is where spread drops rapidly before starting to increase again. This repeated increase and decrease in spread is what causes the spikes shown in Figure 4-34. When the target is found, AUVs accumulate and the spread stabilizes. The spread of the 25 AUVs swarm spans a longer period before it stabilizes because it takes longer to find the target and because of the significance of magnetic north estimation errors on the small size of the swarm. The steady state average spreads are: 48.5, 45.8, and 43.9 WUs for swarm sizes 100, 50, and 25, respectively because of the effect of swarm size on the area it can span.

Like in SDHCP, SSW is based on complete coverage of the directional search space. This causes the average overall traveled distance (Figure 4-35) to be large, again, especially for the 100 AUVs swarm. By doubling the size from 25 to 50 AUVs, the average overall traveled distances increases around sixfold and from 50 to 100, eightfold. This is another limiting factor to the use of SSW (in addition to directionality). Similar to SDHCP, the application may trade off distance (i.e. energy consumption) for resolution.
4.3.6 Performance of SRW

Average number of AUVs that reached target was very low in SRW: 4, 8, and 18 for swarm sizes 25, 50, and 100, respectively (Figure 4-36). This was expected as random walk does not have any built-in mechanism to bound the swarm’s dispersion. This results in a fast loss of AUVs. Only *lucky* AUVs that happen to pass by the target succeed to find it. Interestingly, these vehicles find the target relatively fast as can be seen from figure, probably because if this does not take place early enough in the simulation, they will end up being lost like others.
As pointed above, the loss rate is very high in SRW. Figure 4-37 shows AUV loss for the three swarm sizes. Within a relatively short period (3074 ticks) all the swarms had lost at least 70% of their members. The number of lost AUVs increases linearly with swarm size resulting nearly similar percentages of lost AUVs as can be seen in Figure 4-38. All swarms ended up losing around 80% of their members. The surviving AUVs were mostly at the target, except for very few that were still wandering by the time the simulation ended (also shown in Figure 4-38).
Figure 4-39: SRW – Average distance to target.

Average distance to target is shown in Figure [4-39] for the three swarm sizes. After the initial drop-off, the swarm disperses away from the source. The larger the swarm, the earlier in time the average distance to target will start increasing. However, the extent to which average distance increases reduces with increase in swarm size. The first phenomenon can be explained by looking at the average spread curves in Figure [4-40]. One can easily notice that, just after drop-off, spread decreases rapidly due to the condensing effect of changing from the 3D cubic drop-off topology to a square 2D topology as pointed out previously. It can also be seen that while this effect continues longer in time for smaller swarm sizes, it does not for larger ones. As the number of AUVs increases, the rate at which they depart from drop-off location increases, causing the average spread to start growing faster. This leads to pulling the average spread high earlier in time as shown by the up-ramping section of the red and black curves for 50 and 100 AUVs swarms, respectively (compare to the blue curve). This increase in spread causes the average distance to target to increase earlier in larger swarms due to the effect of target-departing AUVs on pulling the average up. After all AUVs heading away from the target are lost (at the peaks of the average distance curves), the average distance and spread start decreasing slowly as more vehicles find the target. This
takes a longer period for the 25 AUVs swarm because vehicles that are still on the search platform are spread over the whole area pulling the average high. Eventually, both average distance and spread stabilize when most of/all remaining AUVs find the target.

The second phenomenon (reduction in the extent of average distance increase with swarm size) is caused by the fact that, in larger swarms, there are more non-departing AUVs that reduce the effect (tendency to increase the average) of departing vehicles on average distance and spread.

Average overall distance traveled by different swarms in SRW is low compared to other
algorithms. It increases nearly fivefold with every swarm size doubling as shown in Figure 4-41. Its increase with time follows an S-curve, where saturation is reached when the remaining vehicles start accumulating at the target.

4.3.7 Algorithms Performance Comparison

Having analyzed the performance of every algorithm individually and highlighted its strengths and weaknesses, it is time to compare the different algorithms to have a better understanding of which algorithm is suitable for which scenario. We start by comparing the average number of lost AUVs (Figure 4-42). Obviously, the worst performing algorithm is SRW; it has the highest numbers of losses for all swarm sizes. R-PSO comes next, which always had more than half of the swarm lost for the three swarm sizes. This performance was expected for both algorithms as they do not have any built-in self-bounding capability. The next algorithms with apparently relatively large numbers of lost AUVs are SDHCP and CSF. Although a first look may indicate that they are not very efficient, this is not correct. As emphasized in Sections 4.3.2 and 4.3.4, the main and only reason for this loss is that the coverage range of these algorithms exceeded the boundaries of the search platform. Otherwise, there would be no loss in these two algorithms. Even in the specific test scenario used, their performance is still better than R-PSO. VTS had very low loss count which, again, was caused by the search crossing the search platform’s boundary. No loss is expected in normal conditions for a truly open environment in VTS. SSW had zero losses for all swarm sizes. Although this is great, it is only achievable if the target falls in the search direction of SSW, otherwise, all AUVs would fail to find the target. We conclude that: 1) SRW and R-PSO suffer from high loss percentage and are not suitable for open, unbounded environments, 2) CSF, SDHCP, and VTS can assure zero to very low loss count in an unbounded environment, and 3) SSW is suitable for applications where the approximate target direction is known and can guarantee
Next, we examine the average number of AUVs that reached target (Figure 4-43). Again, SRW was the worst performer, with minimum number of AUVs that found target for all swarm sizes. In SSW, all AUVs were successful in finding the target in all three swarms. R-PSO performed twice as good as SRW. Among the three self-bounding algorithms, VTS performed best because, despite its intersection with the boundary of the platform, its uniform distribution of AUVs over its coverage area caused only a small fraction of AUVs to be lost. CSF and SDHCP had comparable performance and, as mentioned previously, along with VTS they all could have performed as good as SSW.

The third metric we compared the algorithms based on is the overall distance traveled. Although SRW may seem to be the best performing algorithm here, it is not. The apparently short average overall traveled distance looks as such because a very large number of AUVs were lost and the average is found only over the remaining AUVs. The latter ones, clearly travel a very low combined distance before finding the target. VTS is the actual best
Figure 4-43: Comparison of number of AUVs that reached target for different algorithms. Although it lost a very small percentage of AUVs (cf. Figure 4-42), it still had the smallest average overall distance for all swarm sizes. For R-PSO, it would be difficult to accurately tell if it has a good traveled-distance performance. The reason is that for the three swarm sizes, it lost more than 50% of the AUVs, and the shown averages are only taken over the remaining AUVs, similar to SRW. It could be debatable though that if the those AUVs are already lost, why would it be needed to account for the distances they travel in the calculation of average? This brings us back to the question of whether any of these two algorithms should be used in an open environment in the first place. We recommend that they do not be used in such scenarios. SDHCP had he worst average traveled-distance performance. As mentioned in Section 4.3.4, this is caused by the space-coverage attributes of the algorithm and should be tolerated if high resolution sweeping of the search space is needed. SSW has a slightly better performance than SDHCP, but still worse than VTS due to its exhaustive space sweeping attributes similar to SDHCP.

From the above comparison, we can conclude that the self-bounding algorithms proposed
in this work have superior performance compared to the non-self-bounding ones: they assure the preservation of swarm members and finding the target by all or at least very large portion of the swarm. On the downside, SDHCP and CSF suffer from large overall traveled distances, which implicitly means more overall energy consumption. Therefore, whenever energy is a concern VTS can be used because it has low overall traveled distance (low energy consumption), guarantees swarm members preservation, and delivers all AUVs to the target. However, this will be at the expense of longer target-finding time. On the other hand, if high resolution search area coverage and high robustness are a requirement, SDHCP can be used. CSF’s ability to find the target rapidly makes it suitable for applications were time is a critical factor.
4.4 Task Allocation

To compare the performance of the proposed task allocation algorithms, TaskAllocation-Mode of SWIM was used. The results presented in this section are for three groups of clustered targets placed at the vertices of an equilateral triangle as shown in Figure 4-45. Each target group consisted of 10 barrels of a specific type, where the type is indicated by the color as shown in figure. If an AUV processes a preferred target type, it takes one of two predefined times (30 seconds), while processing another type takes a longer time (50 seconds). Swarms of sizes 100, 50, and 25 were tested. For all swarms, the drop-off location was at the origin (center of the triangle) from a height of 90 WUs (i.e. at (0, 90, 0)). All algorithms were set to use Tethered Random Walk (T-RW) described in Section 3.8.3. The simulation was set to terminate once all targets have been processed and, at that time, it stored the numbers of per-target processings for each of the 30 targets to an output file. The simulator also outputted the times taken to process percentiles of the targets. Each algorithm was run 10 times for each swarm size and the results were averaged. In the following set of sections, we first analyze the performance of every algorithm individually, then follow that by a comparison of all algorithms in the last section.
4.4.1 Performance of BL-TA

Although Blind-Task Allocation (BL-TA) is not a candidate approach that would normally be used (because AUVs applying it do not use any preference when processing encountered targets; they just process whatever target they come across), it was tested to serve as a baseline for comparison of the proposed algorithms. Figure 4-46 shows the average percentages of targets processed over time for swarm sizes 100, 50, and 25. Average times taken to process 100% of targets were: 838, 1440, and 2014 simulation ticks, respectively. It can be noticed that the performance of the 50 and 100 AUVs swarms was similar to some extent until 90% of the targets were processed. Due to the non-distinguishing nature of BL-TA between targets to process, the long times spent on processing already processed targets significantly slow the 50 AUVs swarm from processing all targets in a reasonable time. Average numbers of per-target processings were 5.625, 6.2, 3.6 for the 100, 50, and 25 AUVs swarms, respectively as shown in Figure 4-47. Again, this confirms the over-processing of targets in the case of 50 AUVs swarm; the average number of processings per target is larger than the case of 100 AUVs.
4.4.2 Performance of BB-TA

Beacon-Based Task Allocation performed very well for swarm sizes 50 and 100 AUVs. This can be seen in Figure [4-48] where the average time taken to process 100% of the targets was: 895, 990, 1781 ticks for swarm sizes 100, 50, and 25, respectively. The performance of the 100 and 50 AUVs swarms was very similar, however, taking a look at Figure [4-49] one can notice that the 50 AUVs swarm had a lower average per-target processings (4.4). For the 100 and 25 AUVs swarms, average per-target processings were: 6 and 3.7, respectively. It is worth noting that by doubling the swarm size from 25 to 50, there was a 44.4% reduction in the average time needed to process the same number of targets with an increase of 19% in the average number of per-target processings.

4.4.3 Performance of HYB-TA

Hybrid-Task Allocation performed badly for 25 and 50 AUVs swarms. On the other hand, it performed very well for the 100 AUVs swarm. Average time spent in processing different percentages of targets is shown in Figure [4-50] for the three swarm sizes. The average times taken to process all targets are: 816, 1850, 2864 ticks for 100, 50, and 25 AUVs, respectively.
By doubling the swarm from 25 to 50 AUVs, there was 35% improvement in the average time needed to process all targets. Doubling from 50 to 100 AUVs resulted a 56% improvement. Average number of per-target processings for the 50 and 100 AUVs swarms were very similar in the case of HYB-TA (5.4 and 5.5, respectively). This can be seen in Figure 4-51.
4.4.4 Performance of ERT-TA

Exponential Response Threshold-Task Allocation (ERT-TA) had a performance comparable to BL-TA for swarm sizes 50 and 100 and performance comparable to HYB-TA for the 25 AUVs swarm. Doubling the swarm from 25 to 50 AUVs resulted a 43% improvement, while from 50 to 100, a 45%. Compared to other algorithms, ERT-TA had the best averages for per-target processings as can be seen from Figure 4-53. Increasing the swarm size resulted an increase in the average but, overall, the averages were the smallest.
4.4.5 Algorithms Performance Comparison

Figure 4-54 summarizes the average times taken by all algorithms to process all targets. BB-TA performed the best for swarm sizes 25 and 50, while HYB-TA was the best for the 100 AUVs swarm. On the other hand, HYB-TA was the worst for swarm sizes 25 and 50 as pointed out before. Surprisingly, BL-TA had a good overall performance despite the fact that it is “blind” with respect to the targets it picks to process. Although ERT-TA performed below average in terms of the average time it took for all swarms, it was almost the best in
terms of the number of per-target processings as can be seen in Figure 4-55. BB-TA had a comparable average (actually lower) for the 50 AUVs swarm and HYB-TA had the same average as ERT-TA for the 100 AUVs swarm. By carefully examining the two figures (4-54 and 4-55), it can be noticed that BB-TA was the best performer of all algorithms for the case of 50 AUVs; it had the least average time to process all targets (990 ticks) and the least average number of per-target processings (4.4). The worst among all algorithms with respect to average per-target processings for: the 100 AUVs swarm was BB-TA, the 50 AUVs swarm was BL-TA, and for the 25 AUVs swarm was HYB-TA.

4.5 Case Studies

To study the robustness of the proposed algorithms in different scenarios and for different target configurations, we consider four case studies in this section: 1) a single target (sunk ship), 2) multiple targets (barrels), of different kinds (red, green, and blue) and processing
values, that are intermixed and distributed over a small area at some random location, 3) barrels of these three kinds that are placed in groups of uniform kind at three spatially-separated locations, and finally, 4) intermixed barrels located at three spatially-separated, random locations. Different global search algorithms are used to present a larger variety of tests and for further validation. We start with the single target scenario in the next section.

### 4.5.1 Case Study I: Single Target

A swarm of 50 AUVs was used to search for a sunk ship. The ship was randomly positioned (with the random position being (-8.305941, 6.0, -38.095573), where the origin is located at the center of the search stage). Swirling Divided Hexagonal Close Packing (SDHCP) was used for global search. Blind Task Allocation (BL-TA) was used because of the single target configuration and because the ship is different from physically distinct targets (like barrels) that can be processed based on type. Therefore, BL-TA, despite its simplicity, was sufficient.

![Average per-Target Processings](image)

Figure 4-55: Comparison of average number of per-target processings for different algorithms.
to assist individual AUVs assign themselves to the task at hand (processing ship-related data). It was assumed that the ship consists of different sections and that AUVs which succeed to find the ship randomly process a fraction of these sections before deciding to end the execution of task allocation algorithm and proceed to the surfacing algorithm. For local target search, Tethered Random Walk (T-RW) was used due to its speed and reduced overall traveled distance for its execution. A simple integration algorithm, Brainless All-Stage Integration, was used for integrating the algorithms used in all mission stages. Figure 4-56 shows snapshots of the mission at the beginning, end, and two distinct intermediate times.

Figures 4-57–4-61 show the values of similar metrics to the ones used in previous sections, but this time for the scenario considered in Case Study I. In Figure 4-57 average distance to target swings up and down due to the expanding and swirling behaviors of the six sub-swarms. It eventually drops to its minimum when all surviving members find the target.

Average spread follows a similar pattern to average distance to target, but initially increases due to the expansion of sub-swarms as they diverge from origin. Similar to average distance, it stabilizes after all surviving AUVs reach the target.

In Figure 4-59 number of AUVs that reached target over time is shown. It can be noticed that 43 AUVs eventually found the target while 7 were considered lost (see Figure 4-60) as they exceeded the search stage’s boundary. Again, in a truly open environment this is not expected to occur.

The overall traveled distance by the swarm over time is shown in Figure 4-61. It increases almost linearly until all surviving AUVs reach the target, at which point it stabilizes.

Table 4.1 lists the values used in calculating mission’s gain. The overall obtained dollar value from the combined processing of different ship sections is determined by the sum of values of non-overlapping sections of the ship that were processed. For example, using values in Table 4.1 if the average number of sections processed by a single AUV that don’t
Figure 4-56: Case Study I: a complete mission of searching for a sunk ship using SDHCP for global search, BL-TA for task allocation, and T-RW for local target search.
overlap with sections processed by any other AUV is 10, the total obtained dollar value is:

\[
\frac{10}{500} \times 1,000,000 \times 43(AUVs) = 860,000.
\]

Using this value and other values from Table 4.1 dollar gain, degree of time compliance, and mission gain can be calculated as follows:

Dollar gain = \( G_d = \frac{V_m}{C_m} = \frac{860,000}{(15,000 \times 50 + 10,000)} = \frac{860,000}{760,000} = 1.13157895 \)

Degree of time compliance = \( R_t = \frac{T_r}{T_m} = \frac{9500}{(5649 - 866)} = 1.98620 \)

Mission gain = \( G_m = G_d \times R_t \times R_c = 1.13157895 \times 1.98620 \times 0.86 = 1.932887 \)
4.5.2 Case Study II: Multiple, Spatially Co-Located, Intermixed Targets

The first multiple targets scenario we consider is a group of 40 intermixed (red, green, and blue), spatially co-located barrels. The targets were randomly placed and Constrained Spiral Flocking (CSF) was used for global target search. For task allocation, Beacon Based-TA (BB-TA) was used. Similar to the previous case study, local search was done using T-RW. Table 4.2 provides the values of different variables/parameters used in this mission. Four
Figure 4-61: Case Study I – Overall distance traveled by the swarm.

Table 4.1: Case Study I: values of the parameters used in mission gain calculation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single AUV’s value</td>
<td>$15,000</td>
</tr>
<tr>
<td>Number of AUVs used</td>
<td>50</td>
</tr>
<tr>
<td>Operational cost (for the whole swarm)</td>
<td>$10,000/day</td>
</tr>
<tr>
<td>Percentage of AUVs recovered</td>
<td>86%</td>
</tr>
<tr>
<td>Mission time constraint</td>
<td>9500 ticks</td>
</tr>
<tr>
<td>Mission start time</td>
<td>866 ticks</td>
</tr>
<tr>
<td>Mission completion time</td>
<td>5649 ticks</td>
</tr>
<tr>
<td>Ship full processing value</td>
<td>$1,000,000</td>
</tr>
<tr>
<td>Ship sections to process</td>
<td>500</td>
</tr>
<tr>
<td>Average sections processed by an AUV</td>
<td>10</td>
</tr>
</tbody>
</table>

Snapshots of the swarm taken at different times are shown in Figure [4-62](#).

Average distance to target over time is shown in Figure [4-63](#). It follows the typical behavior of CSF by first oscillating while growing due to the spiral divergence pattern the algorithm uses: as the agents move away from the target, the distance grows and when they turn towards the target, it decreases. When the swarm eventually finds the target, the distance stabilizes at a low value a little above 20 WUs. The relative increase in the period from $\sim1400$ to $\sim4500$ ticks is due to the effect of the local search algorithm that spreads the AUVs around the targets increasing the average distance slightly.

Average spread is shown in Figure [4-64](#) and follows the typical behavior of CSF (diverging
Figure 4-62: Case Study II: a complete mission of searching for a group of intermixed targets using CSF for global search, BB-TA for task allocation, and T-RW for local target search.
Table 4.2: Case Study II: values of the different variables/parameters used.

<table>
<thead>
<tr>
<th>Variable/Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of red barrels (type I targets)</td>
<td>17</td>
</tr>
<tr>
<td>Number of green barrels (type II targets)</td>
<td>12</td>
</tr>
<tr>
<td>Number of blue barrels (type III targets)</td>
<td>11</td>
</tr>
<tr>
<td>Total number of barrels</td>
<td>40</td>
</tr>
<tr>
<td>Number of AUVs used</td>
<td>100</td>
</tr>
<tr>
<td>Drop-off location</td>
<td>origin (center of stage)</td>
</tr>
<tr>
<td>Target group location</td>
<td>(40.587517, 20.0, -87.12479)</td>
</tr>
<tr>
<td>Single AUV’s value</td>
<td>$15,000</td>
</tr>
<tr>
<td>Operational cost (for the whole swarm)</td>
<td>$10,000/day</td>
</tr>
<tr>
<td>% of processed type I targets</td>
<td>94.1%</td>
</tr>
<tr>
<td>% of processed type II targets</td>
<td>100.0%</td>
</tr>
<tr>
<td>% of processed type III targets</td>
<td>100.0%</td>
</tr>
<tr>
<td>% of AUVs recovered</td>
<td>77%</td>
</tr>
<tr>
<td>Mission time constraint</td>
<td>9500 ticks</td>
</tr>
<tr>
<td>Mission start time</td>
<td>221 ticks</td>
</tr>
<tr>
<td>Mission completion time</td>
<td>2480 ticks</td>
</tr>
<tr>
<td>Single type I target processing value</td>
<td>$20,000</td>
</tr>
<tr>
<td>Single type II target processing value</td>
<td>$10,000</td>
</tr>
<tr>
<td>Single type III target processing value</td>
<td>$15,000</td>
</tr>
</tbody>
</table>

Figure 4-63: Case Study II – Average distance to target.

oscillation) up until the target is found. As opposed to the expected rapid decrease (cf. Figure 4-11) in spread when the target is encountered, the spread stays high and does not decrease because of the effects of local search and task allocation algorithms.

Figures 4-65 and 4-66 show the average number of AUVs that reached target and the
ones lost, respectively. Number of AUVs that reached target jumps abruptly from 0% to 90% in a very short period (1387–1507 ticks). This can also be easily seen from Figure 4-62 (two lower sub-figures). Another observation from Figure 4-65 is that this number starts to decrease slowly afterwards. This is due to the combined effect of the targets being very close to the edge of the search stage and the spreading caused by the local search (T-RW) algorithm. While the vehicles search for the next targets to process, some of them get lost when they reach the boundary. This behavior can also be noticed in Figure 4-66 at ∼1589 where AUVs start getting lost at a slower rate than the preceding abrupt increase.

Overall traveled distance grows very quickly due to the large number of AUVs in the swarm and the exponentially increasing radius if the spiral (see Figure 4-67). After the target is found, increase in overall traveled distance is negligible.

Figure 4-68 shows the percentage of targets processed over time for the three target types combined. The swarm was able to process 98% of the targets and the percentages shown in Table 4.2 for each target type.

Using the values provided in Table 4.2, mission gain can be calculated as follows:

Mission cost = $15,000 × 100 + $10,000 = $1,510,000
Mission value \( V_m = 16 \times $20,000 + 12 \times $10,000 + 11 \times $15,000 = $605,000 \)

Dollar gain \( G_d = V_m / C_m = $605,000 / $1,510,000 = 0.40066224 \)

Degree of time compliance \( R_t = T_r / T_m = 9500 / (2480 - 221) = 4.20540 \)

Mission gain \( G_m = G_d \times R_t \times R_c = 0.40066224 \times 4.20540 \times 0.77 = 1.29741 \)
4.5.3 Case Study III: Multiple, Spatially Separated, Distinct Targets

In this case study, a set of three groups of distinct (not intermixed), spatially separated and randomly placed targets were used. Each group had only targets of the same type. CSF, BB-TA, and T-RW were used as the global search, task allocation, and local search algorithms, respectively. Variables and parameter values used in this scenario are listed in Table 4.3.
Table 4.3: Case Study III: values of the different variables/parameters used.

<table>
<thead>
<tr>
<th>Variable/Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of red barrels (type I targets)</td>
<td>20</td>
</tr>
<tr>
<td>Number of green barrels (type II targets)</td>
<td>20</td>
</tr>
<tr>
<td>Number of blue barrels (type III targets)</td>
<td>20</td>
</tr>
<tr>
<td>Number of AUVs used</td>
<td>100</td>
</tr>
<tr>
<td>Drop-off location</td>
<td>origin (center of stage)</td>
</tr>
<tr>
<td>Target group I location</td>
<td>(28.551508, 20.0, -76.62064)</td>
</tr>
<tr>
<td>Target group II location</td>
<td>(60.287617, 20.0, -20.82378)</td>
</tr>
<tr>
<td>Target group III location</td>
<td>(55.58027, 20.0, -81.550995)</td>
</tr>
<tr>
<td>Single AUV’s value</td>
<td>$15,000</td>
</tr>
<tr>
<td>Operational cost (for the whole swarm)</td>
<td>$10,000/day</td>
</tr>
<tr>
<td>% of processed type I targets</td>
<td>45.0%</td>
</tr>
<tr>
<td>% of processed type II targets</td>
<td>100.0%</td>
</tr>
<tr>
<td>% of processed type III targets</td>
<td>25.0%</td>
</tr>
<tr>
<td>% of AUVs recovered</td>
<td>100%</td>
</tr>
<tr>
<td>Mission time constraint</td>
<td>9500 ticks</td>
</tr>
<tr>
<td>Mission start time</td>
<td>260 ticks</td>
</tr>
<tr>
<td>Mission completion time</td>
<td>4623 ticks</td>
</tr>
<tr>
<td>Single type I target processing value</td>
<td>$20,000</td>
</tr>
<tr>
<td>Single type II target processing value</td>
<td>$10,000</td>
</tr>
<tr>
<td>Single type III target processing value</td>
<td>$15,000</td>
</tr>
</tbody>
</table>

The swarm had 100 AUVs and was released at the center of the search stage. Figure [4-69] shows snapshots of the mission at the beginning, end, and at two arbitrary intermediate times. The red paths observed on the fourth sub-figure denote the paths traveled by AUVs that joined active beacons.

Average spread followed a similar pattern to the previous case study, but with fewer oscillations due to the relative closeness of the nearest target group to the drop-off location. This is shown in Figure [4-70].

No AUVs were lost in this mission and all of them found targets. Figure [4-71] shows the number of AUVs that reached target over time. The notion of target here refers to any target in any target group. Most of the AUVs hit the green barrels group because it was the first group along the travel path of the swarm. It is for this reason that all type II targets were processed (cf. Table 4.3). Despite the spatial separation between target groups, the local
Figure 4-69: Case Study III: a complete mission of searching for a group of spatially separated, distinct targets using CSF for global search, BB-TA for task allocation, and T-RW for local target search.
search algorithm enabled some of the AUVs to locate the other two groups resulting type I and III target processing percentages of 45% and 25%, respectively.

Overall traveled distance by the swarm is shown in Figure 4-72. The distance first increases rapidly due to the expanding spiral path, then continues to grow slowly as AUVs perform the local search. Eventually, traveled distance reaches saturation when all AUVs have completed the mission.

As mentioned above, not all type I and III targets were processed. The overall percentage of targets processed in this case study is 57%. Figure 4-73 shows the time taken to process
different target percentages.

For mission gain calculation, we use the values listed in Table 4.3. Following a similar approach to the one used in the precious section, mission gain can be determined through the following set of calculations:

Mission cost = \( C_m = 15,000 \times 100 + 10,000 = 1,510,000 \)

Mission value = \( V_m = 9 \times 20,000 + 20 \times 10,000 + 5 \times 15,000 = 455,000 \)

Dollar gain = \( G_d = V_m/C_m = 455,000 / 1,510,000 = 0.3013245 \)
Degree of time compliance = \( R_t = T_r/T_m = 9500/(4623 - 260) = 2.1774 \)

Mission gain = \( G_m = G_d \times R_t \times R_c = 0.3013245 \times 2.1774 \times 1.0 = 0.6561 \)

Although there is a high degree of time compliance and the whole swarm was recovered, the dollar gain is very low because of the low processing percentages of target types I and III. This low dollar gain resulted in a very low mission gain.

4.5.4 Case Study IV: Multiple, Spatially Separated, Intermixed Targets

In this fourth and final case study, a set of three spatially separated target groups, each consisting of intermixed target types, are used. Target numbers in the three groups were: 25, 18, and 16. Virtual Tether Search (VTS) was used for global target search and similar to the previous case study, BB-TA and T-RW were used for task allocation and local search, respectively. Figure 4-74 shows four snapshots taken at different times during the simulation and Table 4.4 lists values of variables and parameters used in this scenario.

Average spread over time is shown in Figure 4-75. The initial drop in average spread (due to drop-off) is followed by a sharp increase caused by the initial divergence of AUVs away from the source as part of VTS's operation. As the AUVs move around within the disc formed by the maximum search radius (the tether), spread continues to fluctuate but in a globally decreasing manner, as more AUVs find targets, until it finally stabilizes.

As expected from VTS, all AUVs were able to find targets that fell within the search radius as shown in Figure 4-76. After that initial localization of targets, only one AUV gets lost (see Figure 4-77 and the tiny spike between 3319 and 3336 ticks in Figure 4-76) due to the proximity of target group 2 to the boundary of the search stage and the effect of T-RW local search.

The overall traveled distance by the swarm is relatively low as previously pointed out
Figure 4-74: Case Study IV: a complete mission of searching for a group of spatially separated intermixed targets using VTS for global search, BB-TA for task allocation, and T-RW for local target search.
Table 4.4: Case Study IV: values of the different variables/parameters used.

<table>
<thead>
<tr>
<th>Variable/Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of red barrels (type I targets)</td>
<td>25</td>
</tr>
<tr>
<td>Number of green barrels (type II targets)</td>
<td>18</td>
</tr>
<tr>
<td>Number of blue barrels (type III targets)</td>
<td>16</td>
</tr>
<tr>
<td>Total number of barrels</td>
<td>59</td>
</tr>
<tr>
<td>Number of AUVs used</td>
<td>100</td>
</tr>
<tr>
<td>Drop-off location origin (center of stage)</td>
<td></td>
</tr>
<tr>
<td>Group 1 location</td>
<td>(102.51054, 20.0, -109.774666)</td>
</tr>
<tr>
<td>Group 2 location</td>
<td>(-80.231476, 20.0, -40.76004)</td>
</tr>
<tr>
<td>Group 3 location</td>
<td>(41.828266, 20.0, -11.941044)</td>
</tr>
<tr>
<td>Single AUV’s value</td>
<td>$15,000</td>
</tr>
<tr>
<td>Operational cost (for the whole swarm)</td>
<td>$10,000/day</td>
</tr>
<tr>
<td>% of processed type I targets</td>
<td>72.0%</td>
</tr>
<tr>
<td>% of processed type II targets</td>
<td>68.4%</td>
</tr>
<tr>
<td>% of processed type III targets</td>
<td>56.3%</td>
</tr>
<tr>
<td>% of AUVs recovered</td>
<td>99%</td>
</tr>
<tr>
<td>Mission time constraint</td>
<td>9500 ticks</td>
</tr>
<tr>
<td>Mission start time</td>
<td>251 ticks</td>
</tr>
<tr>
<td>Mission completion time</td>
<td>4335 ticks</td>
</tr>
<tr>
<td>Single type I target processing value</td>
<td>$20,000</td>
</tr>
<tr>
<td>Single type II target processing value</td>
<td>$10,000</td>
</tr>
<tr>
<td>Single type III target processing value</td>
<td>$15,000</td>
</tr>
</tbody>
</table>
and its change over time is shown in Figure 4-78.

The swarm was only able to locate two (groups 2 and 3) of the three target groups as the third group happened to fall outside the search radius. However, because of the intermixed nature of the target groups, it was able to process a fair percentage of targets from each target type (cf. Table 4.4). The total percentage of targets processed was 65.55%. Figure 4-79 shows the change in that percentage over time until it reaches this final value.

Using the values provided in Table 4.4 mission gain terms and mission gain itself can be calculated as below:
Mission cost $C_m = \$15,000 \times 100 + \$10,000 = \$1,510,000$

Mission value $V_m = 18 \times \$20,000 + 12 \times \$10,000 + 9 \times \$15,000 = \$615,000$

Dollar gain $G_d = V_m/C_m = \$615,000/\$1,510,000 = 0.407285$

Degree of time compliance $R_t = T_r/T_m = 9500/(4335 - 251) = 2.32615$

Mission gain $G_m = G_d \times R_t \times R_c = 0.407285 \times 2.32615 \times 0.99 = 0.9379$

The main driver for mission gain decrease is dollar gain; the failure of the swarm to locate the third target group caused it to lose a significant portion of the monetary gain that could have otherwise been achieved.
Figure 4-79: Case Study IV - Percentage of processed targets over time.
Chapter 5

Conclusions and Future Work

This work studied different aspects of the important topic of applying Swarm Robotics (SR) to underwater environment. It has multiple contributions to the process of mission planning and assessment.

First, a general mission planning framework was developed to serve as a basis for mission design and planning. The framework unifies portions of the rich body of research in SR and uses them as its building blocks with the goal of realizing robust underwater search missions. Starting from the drop-off stage, where Autonomous Underwater Vehicles (AUVs) are released, the mission is divided into consecutive stages that work in synchrony to assure mission completion and successful recovery of the swarm. These stages are: initial self-organization, global target search, task identification and allocation, local target search, and the AUVs recovery.

Second, and as an extension to the first contribution, each of the stages forming the mission was carefully studied and relevant decentralized algorithms were developed to efficiently achieve the stage’s goals.

For global target search, an adaptation of the classic Particle Swarm Optimization (PSO) algorithm was first done to make it usable on physically constrained AUVs, followed by the
development of three novel algorithms for global search: Constrained Spiral Flocking (CSF), Virtual Tether Search (VTS), and Swirling Divided Hexagonal Close Packing (SDHCP). Each of these algorithms draws inspiration either from a natural phenomenon or organism (like in CSF), or real world observations (e.g. VTS from tether ball and SDHCP from close-packing of equal spheres) that produce a desired global behavior and achieve high performance.

To enable the referenced global search algorithms, as well as the algorithms developed for other stages, to be run realistically on the physically constrained AUVs, vehicle reorientation was an unavoidable factor that needed to be taken into consideration. For this reason, a simple reorientation and an optimized same-position reorientation algorithms were developed. These algorithms are usable in different situations and are both essential to the realistic implementation of algorithms in all mission stages. For the latter reorientation algorithm, we proved that its path length is always shorter than the two-armed (one that uses straight travel paths at the beginning and end, and a curved path in between) alternative for all orientations.

To enable tasks to be accomplished at the target, three task allocation algorithms were developed: Exponential Response Threshold based Task Allocation (ERT-TA), Hybrid Task Allocation (HYB-TA), and Beacon Based Task Allocation (BB-TA). ERT-TA uses adaptive threshold functions (ATFs) to activate task processing based on stimuli and thresholds, HYB-TA uses a combination of ATFs and rule-based control to prioritize task accomplishment, and BB-TA uses beacons to guide other AUVs to targets and a set of rules to prioritize tasks.

Local search is useful in localizing next targets to process in the case of multiple targets and to overcome target misses in the case of a single target. For this reason, three local search algorithms were developed: Bubble-Chain Random Walk (BC-RW), Retracted-Sequence Random Walk (RS-RW), and Tethered Random Walk (T-RW). These algorithms share the attribute that they bound the local search area and minimize the chance of divergence from
the targets’ location.

Connecting the above stages is done through the use of a simple integration algorithm that seamlessly combines consecutive mission stages into a whole, which can assure successful mission accomplishment.

Third, a mission assessment measure, called mission profit/gain/utility, was developed to serve as an indicator of the degree of mission success.

For all of the above-mentioned algorithms, detailed analysis of simulation results was done for isolated algorithm executions. Additionally, cross-algorithm comparisons were done to highlight the strengths and weaknesses of each algorithm. Four case studies of complete missions utilizing combinations of these algorithms were also presented to further validate the efficiency of the proposed algorithms. Mission gain was calculated in each case study to evaluate the performance of the specific mission.

Fourth, this work extends Reinforcement Learning’s classic agent-environment interaction by improving the cognitive abilities of AUVs through the introduction of MiniBrain; a human brain-like model for cognition and learning. By mimicking the main areas of human brain and utilizing them to the agent’s benefit, it can make better-informed decisions and develop new techniques to perform the task at hand.

Future work will mainly focus on extensive testing of the developed MiniBrain model and using it to improve AUVs’ ability to find targets, behave appropriately in emergency situations, and speed up task allocation and accomplishment.

Furthermore, joint utilization of RL techniques and MiniBrain will be studied with the goal of benefiting from agent-level, agent-environment level, and swarm level cognitions.
Bibliography


