Investigation of coordination algorithms for swarm robotics conducting area search

Lau, Dylan Z.
Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/47293
INVESTIGATION OF COORDINATION ALGORITHMS
FOR SWARM ROBOTICS CONDUCTING AREA SEARCH

by

Dylan Z. Lau

September 2015

Thesis Co-Advisors: Timothy H. Chung
Duane Davis

This thesis was performed at the MOVES Institute
Approved for public release; distribution is unlimited
INVESTIGATION OF COORDINATION ALGORITHMS FOR SWARM ROBOTICS
CONDUCTING AREA SEARCH

Lau, Dylan Z.

Naval Postgraduate School
Monterey, CA 93943

Approved for public release; distribution is unlimited

This master’s thesis developed an operational coordination controller for enabling sensor-based coverage algorithms for large-scale swarm unmanned aerial vehicles (UAVs). The prototype controller was validated by live-fly field experiments of swarm UAVs carried out in Naval Postgraduate School’s (NPS’s) field laboratory at Camp Roberts.

The proliferation of hobbyist drones, or UAVs, in recent times has presented the opportunity to field large-scale swarm UAVs. A coordinated swarm of UAVs can prove useful in a surveillance or search mission by rapidly covering the area through data collection from multiple vantage points simultaneously. This thesis designed and implemented an onboard swarm search controller that coordinates the swarm for an autonomous area search with different coverage algorithms. The coverage algorithms are evaluated with simulation and validated by live-fly field experiments.
INVESTIGATION OF COORDINATION ALGORITHMS FOR SWARM ROBOTICS CONDUCTING AREA SEARCH

Dylan Z. Lau
Civilian, Defence Science and Technology Agency, Singapore
B.Eng., Nanyang Technological University, 2007

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION
from the

NAVAL POSTGRADUATE SCHOOL
September 2015

Author: Dylan Z. Lau
Approved by: Timothy H. Chung
Thesis Co-Advisor
Duane Davis
Thesis Co-Advisor
Christian Darken
Chair, MOVES Academic Committee
Peter J. Denning
Chair, Department of Computer Science
THIS PAGE INTENTIONALLY LEFT BLANK
This master’s thesis developed an operational coordination controller for enabling sensor-based coverage algorithms for large-scale swarm unmanned aerial vehicles (UAVs). The prototype controller was validated by live-fly field experiments of swarm UAVs carried out in Naval Postgraduate School’s (NPS’s) field laboratory at Camp Roberts.

The proliferation of hobbyist drones, or UAVs, in recent times has presented the opportunity to field large-scale swarm UAVs. A coordinated swarm of UAVs can prove useful in a surveillance or search mission by rapidly covering the area through data collection from multiple vantage points simultaneously. This thesis designed and implemented an onboard swarm search controller that coordinates the swarm for an autonomous area search with different coverage algorithms. The coverage algorithms are evaluated with simulation and validated by live-fly field experiments.
# Table of Contents

1 Introduction 1
   1.1 Rise of Hobbyist UAVs ........................................ 1
   1.2 Multi-Vehicle Swarming of UAVs. ............................ 3
   1.3 Coordinating a Swarm of UAVs for the Perfect Area Search .... 8
   1.4 Evaluating Swarm UAVs Coverage Algorithms .................. 10
   1.5 Main Contributions ............................................. 11
   1.6 Organization of This Thesis .................................... 11

2 Approach 13
   2.1 Discretized Search Area ....................................... 13
   2.2 Maintaining the Global Search Map ............................ 15
   2.3 ARSENL Fixed-Wing UAV (Ritewing Zephyr II) ................. 18
   2.4 ARSENL Fixed-Wing Swarm UAV System Architecture ............ 20
   2.5 Swarm Search Controller Design ............................... 21
   2.6 Coverage Algorithm Evaluation Using Simulation ................ 29

3 Experimentation and Results 33
   3.1 Simulation and Field Scenario .................................. 33
   3.2 Simulation Experimentation ..................................... 34
   3.3 Greedy Coverage Simulation Results .......................... 35
   3.4 Fixed Lane Coverage Simulation Results ....................... 36
   3.5 Field Experiment Results ...................................... 37

4 Simulation Result Analysis 45
   4.1 Time for Complete Coverage Analysis ............................ 45
   4.2 Missed Coverage and Overlap Analysis .......................... 49
   4.3 Load Balancing Analysis ........................................ 49
# List of Figures

<p>| Figure 1.1 | Types of unmanned aerial vehicle (UAV) and their accessibility | 3 |
| Figure 1.2 | Vicon motion capture system in 20 micro quadrotors flight | 5 |
| Figure 1.3 | Perfect area search paths | 9 |
| Figure 2.1 | Searcher sweeping along the search area | 13 |
| Figure 2.2 | Searcher sweeping along the discretized search area | 14 |
| Figure 2.3 | Difference between two approaches for search cells interval | 15 |
| Figure 2.4 | Difference approach to maintaining the global search map | 16 |
| Figure 2.5 | Advanced Robotic Systems Engineering Laboratory (ARSENL)’s fixed-wing swarm UAV (Ritewing Zephyr II) | 19 |
| Figure 2.6 | Processing unit of the ARSENL’s fixed-wing swarm UAV (Ritewing Zephyr II) | 20 |
| Figure 2.7 | ARSENL fixed-wing swarm UAV system architecture | 21 |
| Figure 2.8 | User interface to issue swarm search order and the resulting search grid | 22 |
| Figure 2.9 | State machine diagram of the swarm search controller | 24 |
| Figure 2.10 | Flowchart of greedy coverage algorithm | 27 |
| Figure 2.11 | Flowchart of fixed lane coverage algorithm | 28 |
| Figure 2.12 | ARSENL swarm UAVs simulation environment | 30 |
| Figure 2.13 | Sample coverage algorithm results from simulation | 31 |
| Figure 3.1 | Scenario for simulation runs and August 2015 live-fly field experiments | 34 |
| Figure 3.2 | Time for complete coverage vs. number of searchers (greedy) | 36 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Time for complete coverage vs. number of searchers (fixed lane)</td>
<td>37</td>
</tr>
<tr>
<td>3.4</td>
<td>NPS Field Laboratory entrance sign at McMillian Airfield</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>Scenario used for the 14 July 2015 field experiments</td>
<td>39</td>
</tr>
<tr>
<td>3.6</td>
<td>Five UAVs used to validate the swarm search controller running greedy coverage algorithm</td>
<td>40</td>
</tr>
<tr>
<td>3.7</td>
<td>Full flight trajectories taken by the swarm search UAVs during Field Experiment 1</td>
<td>41</td>
</tr>
<tr>
<td>3.8</td>
<td>Full flight trajectories taken by the swarm search UAVs during Field Experiment 2</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Time for complete coverage vs. number of searchers (combined)</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Flight paths of fixed lane coverage algorithm with four searchers</td>
<td>47</td>
</tr>
<tr>
<td>4.3</td>
<td>Flight paths of fixed lane coverage algorithm with six searchers</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample flight paths of greedy lane coverage algorithm with five searchers</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of greedy coverage algorithm load balancing by number of searchers</td>
<td>50</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Function of autopilot sensors found in smartphones</td>
<td>2</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Mitigation of small payload limitations by swarming</td>
<td>4</td>
</tr>
<tr>
<td>Table 1.3</td>
<td>Comparison of different implementations of swarming UAVs</td>
<td>7</td>
</tr>
<tr>
<td>Table 1.4</td>
<td>Comparison between rotary-wing UAVs and fixed-wing UAVs</td>
<td>8</td>
</tr>
<tr>
<td>Table 2.1</td>
<td>Advantages and disadvantages of flat (global) and hierarchical (regional) search map</td>
<td>17</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Advantages and disadvantages of centralized/decentralized decision making</td>
<td>18</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Approach recommendations for scenario requirements</td>
<td>18</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Simulation experiment parameters</td>
<td>34</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Future field experiment test plan</td>
<td>43</td>
</tr>
</tbody>
</table>
# List of Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Advanced RISC Machines</td>
</tr>
<tr>
<td>ARSENL</td>
<td>Advanced Robotic Systems Engineering Laboratory</td>
</tr>
<tr>
<td>DOD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>FPV</td>
<td>first-person view</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>MOVES</td>
<td>Modeling, Virtual Environments and Simulation</td>
</tr>
<tr>
<td>NPS</td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>UAV</td>
<td>unmanned aerial vehicle</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>USG</td>
<td>United States government</td>
</tr>
</tbody>
</table>
Executive Summary

The technological advances brought by revolutions in, for example, the smartphone, have paved the way for the proliferation of hobbyist unmanned aerial vehicles (UAVs). Further, the accessibility of hobbyist UAVs in recent years brings the opportunity for researchers to deploy swarm UAVs. Swarm UAVs have the potential to improve current surveillance or search missions by collecting data from multiple vantage points simultaneously. Researchers have been steadily increasing the swarm size and complexity of the swarm UAVs they are studying. However, autonomous outdoor swarm behaviors are still restricted to formation flying or "flocking." This thesis seeks to advance the field by coordinating an autonomous outdoor swarm of UAVs for an area search.

Theoretical work has identified that the key challenges to coordinating swarm UAVs for area searches include preventing the swarm UAVs from overlapping with each other during the search, and also ensuring that all of the search area is covered. Several approaches are discussed to overcome these two challenges. The following approaches are found to be most suitable for the thesis to implement. To prevent overlaps, a global search map of all swarm UAV positions is maintained through sharing of positional information using a wireless link. A centralized master searcher uses the global search map to decide the search path for itself and other swarm UAVs. The coverage of the search area is tracked by discretizing the search area.

For this thesis, a new swarm search controller is designed and programmed to coordinate swarm UAVs for an area search. The swarm search controller runs onboard the ARSENL fixed-wing swarm UAV (Ritewing Zephyr II). The swarm search controller’s role in the ARSENL UAV System Architecture is explained, including details about the controller’s software design. The swarm search controller is demonstrated to successfully coordinate area search missions during live-fly field experiments and validates the thesis’ approaches.

Also, two coverage algorithms are implemented in the swarm search controller and validated during the live-fly experiments. The first coverage algorithm is a simple greedy algorithm that assigns discretized search cells to swarm UAVs based on closest distance. The second coverage algorithm is a fixed lane algorithm that minimizes overlaps by pre-
allocating discretized search cells to swarm UAVs. For this study, 150 simulation runs are made using both coverage algorithms with different numbers of searchers. The results from the simulation runs are analyzed for statistical significance, and future live-fly experiments are planned to validate the simulation findings.

Analysis of the simulation results reveals that having more searchers does not guarantee a shorter time for complete coverage. The relative position of the searchers’ starting position to the search area and the orientation of the search area can have a significant impact to the time for complete coverage. The greedy coverage algorithm has more consistent results when more searchers are involved in the search, while having fewer searchers for the fixed lane coverage algorithm gives more consistent results. Overall, the fixed lane coverage algorithm usually provides a shorter time for complete coverage than the greedy coverage algorithm.

The thesis has shown that research in the swarm UAV field is no longer about realizing the swarm UAVs. The field has matured enough that researchers can shift their focus to pursuing actual real-world applications for swarm UAVs. The live-fly validated swarm search controller represents the first steps in advancing this new focus. By experimenting with different coverage algorithms in the swarm search controller, the thesis opens up a new paradigm by demonstrating that swarm UAVs are ready for software experimentation. This research will undoubtedly be joined by many new and exciting contributions from future researchers.
I would like to thank Professor Timothy H. Chung, my advisor, for welcoming me into the Advanced Robotic Systems Engineering Laboratory (ARSENL) team, a positive and high-performance project team he created, and my co-advisor Professor Duane Davis for patiently guiding me. My gratefulness also goes to the Singapore Defense Science and Technology Agency for providing me with the opportunity to pursue my academic interest at this world-class institution.

I would also like to thank my teammates in ARSENL for advice, guidance, and help, especially Michael Clement, Marianna Jones, and Michael Day.

Also deserving thanks are all of my professors and great friends in the Modeling, Virtual Environments and Simulation (MOVES) Institute, who have taught and supported me throughout my stay at Naval Postgraduate School (NPS).

Of course, my thanks go to my parents and family, for sharing their love and encouragement, even if it meant me moving so far away. More than anything I would like to thank my wife, Mei Lu, for the unconditional support, patience, and love she has always provided, and the promising future we have dedicated to each other with our son, Lucas.
CHAPTER 1:
Introduction

The smartphone revolution has brought about rapid technological advances in electronic miniaturization and battery capacity [1]. These technological advances have paved the way for the proliferation of hobbyist drones, or unmanned aerial vehicles (UAVs). Going for no more than a few thousand dollars each, these UAVs provide autonomous Global Positioning System (GPS) and waypoint navigation capability to anyone without the need for formal training or infrastructure [2]. The reduced costs and increased capability of these UAVs have created an opportunity to deploy large groups (swarms) of UAVs to cooperatively achieve a common goal (or goals). A potential use of a swarm of UAVs is a surveillance or search mission where they can be coordinated to collect data from multiple vantage points simultaneously. Theoretical work identifies the key challenges to coordinating swarm UAVs for area searches. Criteria for evaluating area searches are discussed.

1.1 Rise of Hobbyist UAVs
A UAV is an aircraft that has no pilot onboard. “It can be remotely controlled by a pilot at a ground control station or fly autonomously based on pre-programmed flight plans or more complex dynamic automation systems” [3]. Autonomous UAVs used to be expensive, and only militaries could afford to fly them. Now, people interested in flying an autonomous UAV can easily purchase them from any of the major online retailers for less than a few thousand dollars [2]. Bloomberg reported in 2014 that “Atlanta Hobby, one of the largest independent suppliers of civilian drones in the United States, has seen business jump to about $20 million in annual sales, a 10-fold increase from five years ago” [4]. The sales of these hobbyist UAVs have been growing at a very fast rate.

This surge in hobbyist UAVs in recent years can be attributed to the smartphone revolution. The economies of scale and technological transformation brought by the smartphone revolution has lowered the prices of autonomous UAVs so much that anyone can fly them as a hobby today [1]. The smartphone revolution arguably started in 2007, when Apple introduced the iPhone to the world [5]. Through the years since 2007, many technolog-
ical advances that stemmed from meeting the demands of smartphone users have helped to make the hobbyist UAV a reality. For a UAV to fly autonomously, it needs to have an autopilot, an onboard microprocessor that steers the aircraft based on inputs from the sensors onboard. The traditional desktop processor uses an amount of power that only large military-grade UAVs can afford to carry. To meet the increasing computing needs of smartphone applications while meeting the power limitations of a small mobile device, microprocessor designers such as Advanced RISC Machines (ARM) made developments to their hyper-efficient “reduced instruction set computing” architecture. This type of architecture reduces cost, heat, and power use, making them very suitable for smartphones and hobbyist UAVs. In 2010, 95% of the processors used in smartphones were ARM-based processors.

The sensors required by the autopilot are also found in smartphones, as shown in Table 1.1.

<table>
<thead>
<tr>
<th>Sensors</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gyroscopes</td>
<td>Measure rates of rotation and provide orientation information</td>
</tr>
<tr>
<td>Magnetometers</td>
<td>Detect magnetic fields and function as digital compasses</td>
</tr>
<tr>
<td>Barometers</td>
<td>Measure atmospheric pressure to calculate altitude</td>
</tr>
<tr>
<td>Accelerometers</td>
<td>Measure the force of gravity</td>
</tr>
<tr>
<td>GPS</td>
<td>Provides location and time information</td>
</tr>
</tbody>
</table>

Table 1.1: Function of autopilot sensors found in smartphones, after [8], [9]

Because of their use in the smartphone, sensors have also become smaller and cheaper. Other major components of the hobbyist UAV have undergone major transformations. The wireless radio modules have come to embrace 2.4GHz technology, resulting in shorter antennas. Lithium-based batteries have all but replaced nickel cadmium (NiCad) and nickel metal hydrid (NiMH) batteries, due to their higher energy density.

In a way, the hobbyist UAV is just a flying smartphone. The economies of scale and technological advances brought about by the smartphone revolution are what made the rise of hobbyist UAVs possible today. Other than hobbyist UAVs, there are also other types of UAVs in the world. Figure 1.1 from [2] shows the different types of UAV and compares the availability of the UAVs.
A hobbyist UAV, such as the “DJI Phantom” shown in Figure 1.1, comes with an integrated 1080p video camera and live first-person view (FPV) Wi-Fi streaming of video and telemetry; its cost was about US$700 at the time this thesis was written [12].

1.2 Multi-Vehicle Swarming of UAVs

As hobbyist UAVs became more accessible, potential applications of swarming those UAVs started to surface. An article published in 2010 titled “Towards Autonomous Micro UAV Swarms” suggested that the following application scenarios could be supported by UAV swarms [13]:

- patrol of harbor or borders
- inspection of inaccessible areas such as natural disaster sites, dams, or electric lines
- search phase of a search and rescue mission
- dangerous jobs such as mine detection or fire fighting
• pollution control and environment monitoring
• police video surveillance

The potential applications are huge despite the hobbyist UAVs having a relatively small payload and short endurance compared to the bigger military and commercial UAVs shown in Figure 1.1. The hobbyist UAV cannot see as well or fly as long as its bigger cousins. However, these limitations can be mitigated when the UAVs are deployed in a swarm, as seen in Table 1.2.

<table>
<thead>
<tr>
<th>Limitation</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower quality sensor</td>
<td>More sensors at different locations</td>
</tr>
<tr>
<td>Shorter flight endurance</td>
<td>Team-based recharging schemes [14]</td>
</tr>
<tr>
<td>Shorter communication range</td>
<td>Meshed communication networks [15]</td>
</tr>
</tbody>
</table>

Table 1.2: Mitigation of small payload limitations by swarming

In 2001, researchers from the University of the West of England, United Kingdom, used three helium balloons to create a group of flying robots that used infrared sensors to position themselves with each other [16]. The main drawback of using the helium balloons was the severe weight limitation. The researchers could only afford to attach small, lightweight fan units on the balloons that could provide only enough thrust to propel the balloons indoors where the air currents were low.

With the technological advances from the smartphone revolution described in Section 1.1, researchers from Ecole Polytechnique Fédérale de Lausanne, Switzerland, deployed a swarm of 10 autonomous fixed-wing UAVs outdoors in 2011 [17]. The swarm UAVs flew in formation by making use of GPS receivers and wireless modules to share positional information with each other. The swarm UAVs had to avoid collisions with each other, however, by flying at different preassigned flight altitudes.

In an article published in 2013, researchers from the University of Pennsylvania described how they implemented and operated a swarm of 20 micro quadrotors with precision control in an indoor testbed [18]. The quadrotors were able to avoid collisions with processing from a central ground station. The researchers concluded in the paper that “while our quadrotors rely on an external localization system for position estimation and, therefore,
cannot be truly decentralized at this stage, these results represent the first step toward the development of a swarm of micro quadrotors.”

In the indoor testbed, a Vicon motion capture system [19] was used to sense the position of each UAV at 100Hz, as shown in Figure 1.2 [18]. The position data was sent to a desktop computer where high-level control and planning was done in a MATLAB environment, and commands were sent to each quadrotor at 100Hz.

![Figure 1.2: Vicon motion capture system in 20 micro quadrotors flight, from [18]](image)

More recently, in a paper published in September 2014, researchers from Eötvös University, Budapest, Hungary, presented a decentralized multi-copter flock that performed stable autonomous outdoor flight with 10 flying agents [20]. Instead of a central ground station that calculates navigational instructions for UAVs in the swarm, the UAVs were able to fly in formation and avoid collisions using onboard processing.

Along with the increase in swarm UAV capability, the size of swarm UAVs has also increased significantly. In August 2015, the team of faculty and researchers from Naval Post-
graduate School (NPS) Advanced Robotic Systems Engineering Laboratory (ARSENL) scaled up and flew a swarm of 50 autonomous fixed-wing UAVs at Camp Roberts, CA [21]. Table 1.3 adapted from [20] compares the various implementations of swarming UAVs in the past few years.
<table>
<thead>
<tr>
<th>Authors</th>
<th>Year</th>
<th>Vehicle</th>
<th>N</th>
<th>Decentralized?</th>
<th>Collision-avoidance?</th>
<th>Terrain</th>
<th>Dependency</th>
<th>Uniqueness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welsby et al. [16]</td>
<td>2001</td>
<td>helium balloon</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>indoor</td>
<td>arena</td>
<td>first 3D, relative IR positioning</td>
</tr>
<tr>
<td>Hauert et al. [17]</td>
<td>2011</td>
<td>fixed-wing</td>
<td>10</td>
<td>yes</td>
<td>weak / not crucial</td>
<td>outdoor</td>
<td>GPS</td>
<td>first 10 autonomous</td>
</tr>
<tr>
<td>Bürkle et al. [13]</td>
<td>2011</td>
<td>quadrotor</td>
<td>5</td>
<td>no interactions</td>
<td>no</td>
<td>outdoor</td>
<td>GPS, ground control station</td>
<td>extendible framework</td>
</tr>
<tr>
<td>Hoffmann et al. [22]</td>
<td>2011</td>
<td>quadrotor</td>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td>indoor/outdoor</td>
<td>GPS+base station outdoor, over-head camera indoor</td>
<td>extendible framework, nice vehicle dynamics</td>
</tr>
<tr>
<td>Kushleyev et al. [18]</td>
<td>2012</td>
<td>quadrotor</td>
<td>20</td>
<td>no</td>
<td>not applicable</td>
<td>indoor</td>
<td>VICON, central computer</td>
<td>20 units, best precision control</td>
</tr>
<tr>
<td>Turpin et al. [23]</td>
<td>2012</td>
<td>quadrotor</td>
<td>4</td>
<td>SW yes, HW no</td>
<td>yes</td>
<td>indoor</td>
<td>VICON, central computer</td>
<td>quick dynamics</td>
</tr>
<tr>
<td>Stirling et al. [24]</td>
<td>2012</td>
<td>quadrotor</td>
<td>3</td>
<td>yes</td>
<td>not applicable</td>
<td>indoor</td>
<td>ferromagnetic ceiling</td>
<td>relative positioning</td>
</tr>
<tr>
<td>Quintero et al. [25]</td>
<td>2013</td>
<td>fixed-wing</td>
<td>3</td>
<td>no</td>
<td>no</td>
<td>outdoor</td>
<td>GPS, ground control station</td>
<td>distributed sensing</td>
</tr>
<tr>
<td>Vasarhelyi et al. [20]</td>
<td>2014</td>
<td>quadrotor</td>
<td>10</td>
<td>yes</td>
<td>yes</td>
<td>outdoor</td>
<td>GPS</td>
<td>robust outdoor flocking algorithm</td>
</tr>
<tr>
<td>Chung et al. [21]</td>
<td>2015</td>
<td>fixed-wing</td>
<td>50</td>
<td>yes</td>
<td>no</td>
<td>outdoor</td>
<td>GPS</td>
<td>first 50 fixed-wing autonomous</td>
</tr>
</tbody>
</table>

Table 1.3: Comparison of different implementations of swarming UAVs, after [20]
From Table 1.3, the most popular vehicle for swarming UAVs is the quadrotor (rotary-wing). With the ability for vertical takeoff and landing, rotary-wing UAVs do not require a runway or launcher that fixed-wing UAVs require. Moreover, only rotary-wing UAVs can hover in place and have the ability to operate in a tight indoor environment. On the flip side, fixed-wing UAVs have more efficient aerodynamics that provide higher speeds, thus enabling a larger survey area per given flight. A fixed-wing UAV swarm is more suitable for area surveillance applications while the rotary-wing UAV swarm is more suitable for indoor or inspection application where precision maneuvering is required. The comparison between fixed-wing UAVs and rotary-wing UAVs is shown in Table 1.4.

<table>
<thead>
<tr>
<th>UAV Type</th>
<th>Rotary-wing UAVs</th>
<th>Fixed-wing UAVs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td>Can hover</td>
<td>Efficient aerodynamics</td>
</tr>
<tr>
<td>Ability</td>
<td>Operate in tight indoor environment</td>
<td>Fly at higher speed (larger survey area per given flight)</td>
</tr>
<tr>
<td>Application</td>
<td>Indoor or inspection</td>
<td>Area surveillance</td>
</tr>
</tbody>
</table>

Table 1.4: Comparison between rotary-wing UAVs and fixed-wing UAVs

### 1.3 Coordinating a Swarm of UAVs for the Perfect Area Search

Building on the previous work listed in Section 1.2, this thesis seeks to develop swarm UAVs towards actual real-world applications. A potential popular application for swarming UAVs is a surveillance or search mission where they can be coordinated to collect data from multiple vantage points simultaneously. Flying swarm UAVs in large formations or “flocking,” where individual UAVs negotiate and coordinate their positions in the swarm, has been achieved [20]. What can be investigated further is the swarm’s coordination for a collective surveillance or search mission.

Given a fixed area $A$, the following can be assumed about the perfect area search [26]:

- There is no search overlap (every point in $A$ is searched once before any point is searched twice).
- There is no search conducted outside area $A$.
- All of area $A$ is covered by the sensor.
Having no overlap and coverage outside the area ensures that no search effort is wasted. No gaps in search coverage guarantees complete coverage of the area.

The sensor used for the area search can be assumed to be a *cookie-cutter*, or sometimes called a *definite range law*, type with range $R$. This sensor always detects everything within a specified range and never detects anything outside that range [26]:

\[
P(\text{detection}) = \begin{cases} 
1, & \text{range} \leq R \\
0, & \text{range} > R 
\end{cases} 
\]  

A single UAV equipped with the cookie-cutter sensor can conduct the perfect area search by using parallel sweeps (mowing the lawn), spiral-in, or spiral-out paths [27]. The paths prevent the overlap of one searched segment with another, and no search effort is placed outside the search area. The paths can be visualized in Figure 1.3.

![Figure 1.3: Perfect area search paths](image)

The resulting time for the single UAV to complete the perfect area search can be calculated with the following formula [27]:

**Notations:**

- sweep width = $W = 2R$ (distance)
- search speed = $V$ (distance/time)
- sweep rate = $VW$ (area/time)

\[
T = \frac{A}{VW} 
\]  

9
Essentially, the time $T$ to complete the area search is the area divided by the sweep rate. The sweep rate is the search speed $V$ of the UAV multiplied by the sweep width $W$, which is determined by the range $R$ of the cookie-cutter sensor.

Assuming all the UAVs in a swarm have the same search speed and sensor range, the formula can be modified to calculate the time for a UAV swarm with $N$ number of UAVs to complete the perfect area search.

Additional notations:
- number of UAVs = $N$ (integer)

$$T = \frac{A}{VWN} \quad (1.3)$$

The search area is divided equally among the swarm UAVs to have the best or smallest search time.

So the problem or challenge in coordinating a swarm of UAVs for a perfect area search is to make all the UAVs search the area in such a way that they do not overlap with each other while ensuring that all of the search area will be searched. This problem is divided into two parts, and the approach to resolve them is further explained in their respective sections:

1. The first part is ensuring that all of the area has been searched through discretization of the search area (Section 2.1).
2. The second part is the prevention of overlaps through the maintenance of a “global search map” (Section 2.2).

### 1.4 Evaluating Swarm UAVs Coverage Algorithms

It can be assumed from Section 1.3 that there is no single straightforward way to conduct a perfect area search. There will be many different area coverage algorithm attempts to be as close as possible to the perfect area search. It does not matter if the algorithm is for a single UAV or a swarm of UAVs; there are two key metrics used to evaluate those area coverage algorithms:

1. Time for complete coverage (How long does it takes for all points in the search area
1.5 Main Contributions
This thesis demonstrates a way for large-scale swarm UAVs to coordinate among themselves to conduct autonomous area search. It paves the way for swarm UAVs to perform practical applications such as police surveillance, cross-border and harbor patrol, and search and rescue. The swarm UAV controller developed for the thesis also allows different coverage algorithms to be evaluated in simulation; it helps to assess and compare potential coverage algorithms. This saves development effort as not all algorithms need to go through field testing.

1.6 Organization of This Thesis
This thesis is divided into five chapters:

- Chapter 1 provides the reason for the proliferation of hobbyist UAVs, the motivation
for swarming UAVs, and the technical challenges of coordinating swarm UAVs for area search.

- Chapter 2 describes the approach to overcoming these technical challenges.
- Chapter 3 presents the results of the approach in simulation and live-fly field experiments.
- Chapter 4 explains the insights and findings from the results.
- Chapter 5 concludes the thesis and proposes future work to build upon the thesis.
CHAPTER 2: Approach

The primary objective of this thesis is implementing the onboard coordination controller on swarm UAVs for an autonomous live-fly area search. Successful implementation of the coordination architecture is the research goal. The secondary objective of this thesis is evaluating coverage algorithms for performing the area search. This chapter includes explanations for the approach taken as well as the assumptions, designs, and evaluation metrics.

2.1 Discretized Search Area

As a searcher sweeps along the search area, the area behind the searcher’s sweep width is assumed to be covered. This can be visualized in Figure 2.1. By placing discrete “search cells” to cover the whole search area, the searcher can be made to sweep over the search area, as seen in Figure 2.2.

Depending on the interval between the search cells, the search area can be considered to be fully covered when all the search cells are swept over. The interval between the search cells has to be greater or equal to the sweep width to prevent any overlaps and yet as small as possible to prevent any gaps. Choosing the sweep width as the search cells interval causes
gaps to appear in the corner when the UAV turns. To prevent gaps when turning, the sweep width has to be greater than the interval by $\sqrt{2}$. In other words, the interval is $\frac{1}{\sqrt{2}} \approx 70\%$ of the sweep width. Figure 2.3 shows the difference between the two approaches for choosing the search cells interval.

Figure 2.2: Searcher sweeping along the discretized search area

Unless the search area does not require any of the UAVs to turn, there is no way to conduct the perfect area search where there are absolutely no overlaps and gaps in the search. Between the two approaches in Figure 2.3, using the search width as the search cells interval will leave gaps whenever the UAV turns. The other approach will take around 30% more time than the previous approach. For this thesis, the search area is assumed to be completed when all the search cells are swept over by at least one UAV in the swarm.
2.2 Maintaining the Global Search Map

It is relatively straightforward for a single UAV to keep track of what it has searched. With GPS, it knows where it has been and where it still needs to go without overlaps. For a swarm of UAVs, this information resides in the individual UAVs’ memories. Somehow the information needs to be shared among all members of the swarm so, collectively, they know where the swarm has been and where they need to go.

The simplest method is for all the UAVs to broadcast their positions to every other UAV via a wireless link, enabling each receiving UAV to maintain its own copy of the global search map. The quality of the global search map is vital to preventing overlaps. An outdated global search map causes a UAV to move into a position that may have already been searched. The perfect global search map will be one that takes no time to update across all members of the swarm. In reality, this is not possible because the radio module onboard the UAV needs time to encode and decode the position information on the radio frequency. As the number of UAVs increase, the latency for the global search map to get updated will increase as well. The risk of an inconsistent global search map may occur as the network links between the UAVs deteriorate due to the growing number of UAVs.

Another method to maintain the global search map is to break the map down and divide the search area into multiple hierarchical levels of representation. This “divide and conquer” approach is more scalable than the flat representation used in the earlier method. So rather than sharing information from one end of the search area to the other end, the search area...
is divided into regions where smaller groups of UAVs maintain a “regional” search map of its region. The visual difference between these two methods can be seen in Figure 2.4.

![Flat search map](image1)
![Hierarchical search map](image2)

**Figure 2.4: Difference approach to maintaining the global search map**

The search area is divided into four regions in Figure 2.4. Within each region, a group of UAVs is under the control of a “master searcher,” which coordinates the boundary of the regions with other master searchers. The master searcher and the other “slave searchers” only need to broadcast their positions to UAVs in the same region to maintain the regional search map. The advantages and disadvantages of the two methods are summarized in Table 2.1.

With a reliable search map, the next issue for the searchers to decide is where to go next. Assuming the search map is 100% consistent across all searchers, each searcher can run an algorithm to select its next destination and use the same algorithm to predict other searchers’ destinations as well. If the search map is not the same across the entire swarm, the searchers may make the wrong decision and cause overlaps. Given the potential of unreliable search maps, the master searcher can also take on the role of an arbiter at a search region. The master searcher receives positional information from all searchers and maintains a single centralized regional search map. The master searcher can make the decision of where each searcher (including itself) has to go without fear of conflicts. Another advantage of this centralized approach is that it is easier to debug if something goes wrong; there
<table>
<thead>
<tr>
<th>Methods</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single flat global search map</td>
<td>• Resilient to failure of any UAV</td>
<td>• Network performance limits scalability</td>
</tr>
<tr>
<td></td>
<td>• Simple to implement</td>
<td></td>
</tr>
<tr>
<td>Multiple hierarchical regional</td>
<td>• Scalable to large search area</td>
<td>• Failure of a master searcher affects whole region of searchers</td>
</tr>
<tr>
<td>search maps</td>
<td></td>
<td>• Additional complication in coordination efforts between master searchers</td>
</tr>
</tbody>
</table>

Table 2.1: Advantages and disadvantages of flat (global) and hierarchical (regional) search map

is only one place to look. The slave searchers can deploy with cheaper units as they require less memory and processing power. The main disadvantage is that the master searcher becomes a single point of failure, as with the case of the hierarchical search map.

The advantages and disadvantages of centralized and decentralized decision making are summarized in Table 2.2.

There are advantages and disadvantages to each of the approaches in maintaining the global search map. The choice is largely dependent on the scenario of the area search. Table 2.3 shows the recommendations for which search map approach to use depending on the scenario requirements.

For this thesis, the goal is to coordinate a swarm UAV to conduct an area search as quickly as possible with minimum overlaps. A centralized decision-making, single flat global search map approach is selected for the swarm UAV controller.
### Table 2.2: Advantages and disadvantages of centralized/decentralized decision making

<table>
<thead>
<tr>
<th>Methods</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decentralized decision making</td>
<td>• Resilient to failure of any single UAV</td>
<td>• Additional complication in coordination efforts between searchers</td>
</tr>
<tr>
<td></td>
<td>• Scalable to very large number of searchers</td>
<td></td>
</tr>
<tr>
<td>Centralized decision making</td>
<td>• Cheaper slave searchers as less memory and processing power needed</td>
<td>• Failure of a master searcher affects whole region of searchers</td>
</tr>
<tr>
<td></td>
<td>• Simple to debug</td>
<td>• Master searcher’s performance affects scalability of the number of searchers</td>
</tr>
</tbody>
</table>

### Table 2.3: Approach recommendations for scenario requirements

<table>
<thead>
<tr>
<th>Single flat global search map</th>
<th>Decentralized decision making</th>
<th>Centralized decision making</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Large number of searchers</td>
<td>• Precision search required</td>
</tr>
<tr>
<td></td>
<td>• Hostile environment where UAV failure is possible</td>
<td></td>
</tr>
<tr>
<td>Multiple hierarchical regional search map</td>
<td>• Large search area</td>
<td>• Large search area</td>
</tr>
<tr>
<td></td>
<td>• Large number of searchers</td>
<td>• Precision search required</td>
</tr>
<tr>
<td></td>
<td>• Hostile environment where UAV failure is possible</td>
<td></td>
</tr>
</tbody>
</table>

#### 2.3 ARSENAL Fixed-Wing UAV (Ritewing Zephyr II)

The platform selected to implement the coordination algorithms is the Ritewing Zephyr II hobby airframe. It is the latest UAV used for ARSENAL’s swarm UAV field exercises. Figure 2.5 shows a picture of the UAV.

The 56-inch wide UAV has a speed of around 20 meters per second and can fly for approximately 45 minutes. This means the UAV has a flight range of up to 54 km. In the center of the UAV is the processing unit where two processor boards reside. Figure 2.6 reveals
the processing unit in detail. The two processor boards are based on the ARM architecture
described in Section 1.1. The autopilot used is the Pixhawk system from 3DR [9]. The
gyrosopes, magnetometers, barometers, accelerometers, and GPS are connected to the
Pixhawk where it runs open-source, community-developed software on a real-time operat-
ing system (NuttX). The second processor board (autonomy payload) is the ODROID U3
which runs the Robot Operating System (ROS) [28]. ARSENL researchers programmed
controllers in Python and run them in the ROS to control UAV behaviors. Section 2.4
describes how various components interact with each other.
2.4 ARSENL Fixed-Wing Swarm UAV System Architecture

Figure 2.7 shows the system architecture of the UAV. To fly the UAV, the autopilot gathers environment readings from its onboard flight sensors to determine its position, axes, heading, and altitude. It shares this information with the autonomy payload. Depending on which controller is being run in the autonomy payload, the controller decides where the UAV should go next. For the swarming controllers, the autonomy payload can share its positional information with other UAVs in the swarm in network messages via a wireless link (Wi-Fi radio). The autonomy payload may also receive commands from other UAVs via the network messages. Once the controller decides where to go, it sends waypoint commands to the autopilot where it steers the wings and throttles the motor to fly to the waypoint.
2.5 Swarm Search Controller Design

The bulk of the thesis implementation is in coding the swarm search controller for the autonomy payload. The source code for the swarm search controller is in Appendix: Source Code. A state machine diagram of the swarm search controller is shown in Figure 2.9. When the swarm search controller activates, the UAV searcher will go into the slave searcher state. In this state, the UAV searcher takes on a simple role: its only task is to fly to any waypoint assigned to it. In the meantime, other parts of the autonomy payload broadcast the UAV searcher’s positional information on the wireless link. Every UAV searcher can keep track of the position of every other UAV in the swarm.

A swarm search order is issued to the swarm UAV via a network message through the wireless link. This swarm search order contains the information of the search area, which UAV takes the role of the master searcher in the swarm and the swarm search algorithm to use. The user interface to issue this order can be seen in Figure 2.8.

Once the swarm search order is received, the UAV searcher designated to be the master searcher transits to the master searcher state. All other UAV searchers remain as slave
searchers. The master searcher is in charge of the search operation and maintains the global search map. The master searcher starts by generating the search grid based on the search area information in the swarm search order. The Zephyr II UAV has a turning radius of around 70 meters at full speed. So the sensor range of the searchers is assumed to be 75 meters and the search cell’s interval is set at 150 meters (2 times the sensor range). Once the search grid is created, the search operation is ready for ingress. All the searchers are then assigned to fly to the center of the search grid. The master searcher issues waypoints to itself also.

![User interface to issue swarm search order and the resulting search grid](image_url)

Figure 2.8: User interface to issue swarm search order and the resulting search grid

When one of the searchers arrives within 150 meters of the search grid, the search operation is deemed active and the clock counter for the search starts. This clock counter is used to compare the time for complete coverage of different algorithms. Based on the coverage algorithm used, the master searcher assigns search cells in the search grid to the searchers. Details of the coverage algorithms implemented for this thesis are found in Sections 2.5.1 and 2.5.2. Whenever the master searcher sees that a searcher has visited its assigned search cell, the master searcher updates the global search map and assign other unvisited search
cells to the searcher. When all search cells are visited, the search operation is deemed completed and the clock counter stops. The master searcher then sends all the searchers back to their original positions for the next swarm search order.
Figure 2.9: State machine diagram of the swarm search controller
2.5.1 Greedy Coverage Algorithm

During the active phase of the search operation shown in Figure 2.9, the master searcher assigns unvisited search cells for the searchers. The way the master searcher assigns the search cells depends on the coverage algorithm specified in the search order. For the purpose of proving the concept of the swarm search controller, a simple “greedy” coverage algorithm is selected for implementation. The greedy coverage algorithm is one of the easiest algorithms to implement and serves well as a benchmark for other future coverage algorithms. The flowchart of this greedy coverage algorithm is shown in Figure 2.10.

When a searcher reaches its assigned search cell, the greedy coverage algorithm just chooses the closest unvisited search cell as the searcher’s next search cell. This repeats until all the search cells are visited. The greedy coverage algorithm is used for the validation of the swarm search controller during Field Experiment 1 described in Section 3.5.1.

2.5.2 Fixed Lane Coverage Algorithm

The fixed lane coverage algorithm is designed with the goal of improving upon the greedy coverage algorithm after the successful testing of the swarm search controller in Field Experiment 1. Analysis of the greedy coverage algorithm in Section 4.1 shows that the time for complete coverage can be improved by reducing the flight overlaps between the searchers.

The fixed lane coverage algorithm prevents flight overlaps by pre-allocating search cells to the searchers. The search cells are allocated together as a single lane, either a row or column of the search area. There will be no flight overlaps as each searcher flies a straight line in its own dedicated lane. The limitation of this algorithm is there must be more searchers than the number of lanes from the shortest side of the rectangular search area. If there are more lanes than searchers, the algorithm can not allocate the lanes fully and the entire search has to be conducted with the greedy coverage algorithm instead. Future implementation of the fixed lane coverage algorithm is expected to resolve this limitation.

The flowchart of this fixed lane coverage algorithm is shown in Figure 2.11.

When a searcher reaches its assigned search cell, the fixed lane coverage algorithm chooses the closest unvisited search cell in the searcher’s lane as the searcher’s next search cell. This
repeats until all search cells in the lane are visited. Since each searcher begins the search at the end of a lane, this approach ensures that each lane is searched without backtracking or overlap. The search operation ends when all the search cells in all searchers’ lanes are visited. The fixed lane coverage algorithm is validated during the Field Experiment 2 described in Section 3.5.2.
Figure 2.10: Flowchart of greedy coverage algorithm
Figure 2.11: Flowchart of fixed lane coverage algorithm
2.6 Coverage Algorithm Evaluation Using Simulation

ARSENL uses JSBSim to simulate the flight of the Zephyr II UAV [29]. JSBSim is an open source flight dynamics model that is integrated with the ARSENL Zephyr II UAV’s autopilot software. So instead of receiving and sending commands/information to actual components on the UAV, JSBSim simulates the effect of those commands in the virtual environment and updates the autopilot via simulated readings to the autopilot’s sensors. This software-in-the-loop approach allows minimum or no change from development source code to actual deployment source code.

During simulation, visualization of the UAV searchers is provided, as seen in Figure 2.12. The internal state of the swarm search controller can also be printed to the screen to aid debugging and troubleshooting. At the end of each search, the master searcher prints the global search map, as seen in Figure 2.13. The time for complete coverage results and the allocation of search cells among the searcher are used to compare the performance between different coverage algorithms as described in Chapter 4.
Figure 2.12: ARSENL swarm UAVs simulation environment
Figure 2.13: Sample coverage algorithm results from simulation
This chapter presents the scenarios used for the experiment simulation runs and the live-fly field experiments. The results from the simulation runs of the scenarios are shown here. These simulation results are used to evaluate the coverage algorithms in Chapter 4. The events during the live-fly field experiments that are used to validate the swarm search controller and coverage algorithms are described here as well.

3.1 Simulation and Field Scenario

Figure 3.1 shows the scenario, which is the same between the simulation experimentation and Field Experiment 2. The search area is a 900 meter by 600 meter rectangle just north of McMillan Airfield at Camp Roberts, CA. The search area consists of 24 search cells that are 150 meters apart from each other, with the bottom left search cell at latitude 35.720474 and longitude -120.77481. The red boundary shown in Figure 3.1 is the safety fence that encloses the experimentation area. Any UAVs that breach the fence are automatically triggered to return to a rally point near the airfield for remedial action. The swarm UAVs begin each search at the standby point indicated by a white star in Figure 3.1.
3.2 Simulation Experimentation

The purpose of the simulation experiment is to evaluate and compare the greedy coverage algorithm and the fixed lane coverage algorithm. The effect of the number of searchers on the coverage algorithm is also taken into consideration. The experiment parameters are shown in Table 3.1.

<table>
<thead>
<tr>
<th>Coverage algorithm</th>
<th>No. of searchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>4</td>
</tr>
<tr>
<td>Fixed lane</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.1: Simulation experiment parameters

There are five sets of simulation runs for the experiment parameters: three for the greedy coverage algorithm and two for the fixed lane coverage algorithm. The search area in the
scenario is a six-columns-by-four-rows rectangle. Since the fixed lane coverage algorithm can only allocate lanes according to the number of rows or columns, there is no need to experiment the fixed lane coverage algorithm with five searchers.

It is assumed from the Central Limit Theorem that the time for complete coverage of each set of experiment parameter simulation runs is normally distributed. Having 30 simulation runs for each set of experiment parameters will provide a fairly good approximation of the normal distribution. A good normal distribution is required for assessing the statistical significance between the time for complete coverage means. Thus, a total of 150 simulation runs are made for the experimentation.

3.3 Greedy Coverage Simulation Results
Figure 3.2 shows the scatter chart of the 90 greedy coverage simulation runs.

The mean for the time for complete coverage with four searchers is 56.7 seconds (30 simulation runs). Using 95% confidence intervals, the upper limit of the time for complete coverage with four searchers is 58.5 seconds, while the lower limit is 54.9 seconds. The mean for the time for complete coverage with five searchers is 55.2 seconds. The upper limit of the time for complete coverage with five searchers is 56.8 seconds, while the lower limit is 53.5 seconds. The mean for the time for complete coverage with six searchers is 52.5 seconds. The upper limit of the time for complete coverage with six searchers is 53.8 seconds, while the lower limit is 51.2 seconds.
3.4 Fixed Lane Coverage Simulation Results

Figure 3.3 shows the scatter chart of the 60 fixed lane coverage simulation runs.

The mean for the time for complete coverage with four searchers is 49.5 seconds (30 simulation runs). Using 95% confidence intervals, the upper limit of the time for complete coverage with four searchers is 50.1 seconds, while the lower limit is 48.9 seconds. The mean for the time for complete coverage with six searchers is 57.1 seconds. The upper limit of the time for complete coverage with six searchers is 58.4 seconds, while the lower limit is 55.8 seconds.
3.5 Field Experiment Results

The swarm UAVs field experiments are conducted at McMillan Airfield, Camp Roberts, CA. Camp Roberts’ restricted military airspace provides a safe and secure environment for ARSENAL researchers to fly large scale swarm UAVs. The McMillan Airfield’s 3,500 ft long landing strip is also essential for landing ARSENAL’s fixed-wing UAVs. Figure 3.4 shows the entrance sign of NPS Field Laboratory at McMillan Airfield.

3.5.1 Field Experiment 1: 14 July 2015

The objective of Field Experiment 1 is to demonstrate this thesis’ prototype of an onboard swarm UAV controller to coordinate and conduct an autonomous area search. This is the
first time for ARSENLS swarm UAVs to generate dynamic UAV flight paths using onboard processing to accomplish a mission; the only information required by the swarm UAVs are the size and location of the search area. Even though the controller is tested successfully in simulation, there is always a risk of real-world issues not captured in the simulation:

- whether computation by the controller is sustainable and scalable
- whether actual communications conditions (e.g., latency or losses) will support the coordination between the UAVs
- whether the flight systems will be able to accomplish the commands given to them by the controller

The search area in the Field Experiment 1 scenario is a 600 meter by 450 meter rectangle with 12 search cells, as shown in Figure 3.5.
Five UAVs were used as the swarm UAVs in the field experiment. Figure 3.6 shows the five UAVs being ready for the swarm search controller’s flight trials.

One of the UAVs failed to take off from the launcher, but the concept of the swarm search controller coordinating the swarm UAVs for area search was still proven successfully with the four remaining UAVs. The master searcher flying in the swarm UAVs generated the search grid and assigned search cells to the slave searchers using the onboard Wi-Fi radios. Figure 3.7 shows the full flight trajectories taken by the four UAVs during Field Experiment 1.
3.5.2 Field Experiment 2: 26 August 2015

The first objective of Field Experiment 2 is to test the swarm search controller with a larger search area. The search area is doubled from 12 search cells from Field Experiment 1 (Figure 3.5) to 24 search cells (Figure 3.1). The second objective of Field Experiment 2 is to validate the fixed lane coverage algorithm. Five UAVs were used as the swarm UAVs in the field experiment. The swarm UAVs completed the area search successfully using the fixed lane coverage algorithm and the greedy coverage algorithm. Both objectives of Field Experiment 2 were met. Figure 3.8 shows the full flight trajectories taken by the five UAVs during Field Experiment 2.

3.5.3 Future Field Experiment

A test plan is created for future field experiment to validate the simulation results findings. The test plan is shown in Table 3.2.
Figure 3.7: Full flight trajectories taken by the swarm search UAVs during Field Experiment 1
Figure 3.8: Full flight trajectories taken by the swarm search UAVs during Field Experiment 2
<table>
<thead>
<tr>
<th>Test name</th>
<th>Purpose</th>
<th>Expected Results</th>
<th>Live-fly Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy with 6 UAVs</td>
<td>Gather coverage time for greedy with 6 UAVs</td>
<td>Search completed with all UAVs returning to swarm standby point</td>
<td>Pass?</td>
</tr>
<tr>
<td>Fixed lane with 6 UAVs</td>
<td>Gather coverage time for fixed lane with 6 UAVs</td>
<td>Search completed (columns) with all UAVs returning to swarm standby point</td>
<td>Pass?</td>
</tr>
<tr>
<td>Fixed lane with 5 UAVs</td>
<td>Test the swarm search controller in retiring the extra UAV; gather coverage time for fixed lane with 4 UAVs</td>
<td>Search completed (rows) with 4 UAVs returning to swarm standby point</td>
<td>Pass?</td>
</tr>
<tr>
<td>Greedy with 5 UAVs</td>
<td>Gather coverage time for greedy with 5 UAVs</td>
<td>Search completed with all UAVs returning to swarm standby point</td>
<td>Pass?</td>
</tr>
<tr>
<td>Greedy with 4 UAVs and greedy with 2 UAVs</td>
<td>Test the swarm search controller in coordinating multiple area search at the same time; gather coverage time for greedy with 4 UAVs</td>
<td>Search completed with 4 UAVs returning to swarm standby point while the other 2 UAVs are still searching</td>
<td>Pass?</td>
</tr>
<tr>
<td>Greedy with 2 UAVs (from previous run) and fixed lane with 4 UAVs</td>
<td>Test the swarm search controller in coordinating multiple area search with different coverage algorithms at the same time</td>
<td>Search completed with all UAVs returning to swarm standby point</td>
<td>Pass?</td>
</tr>
<tr>
<td>Greedy with 10-15 UAVs</td>
<td>Test the swarm search controller’s ability to handle 10-15 UAVs</td>
<td>Search completed with all UAVs returning to swarm standby point</td>
<td>Pass?</td>
</tr>
</tbody>
</table>

Table 3.2: Future field experiment test plan
CHAPTER 4: Simulation Result Analysis

This chapter describes the analysis on the simulation experimentation results. The purpose of the experiment is to evaluate and compare the greedy and fixed lane coverage algorithms in the swarm search controller for an area search. Of the criteria to evaluate coverage algorithms listed in Section 1.4, “time until complete coverage,” “missed coverage,” and “load balancing” are examined in this thesis. The other criteria, “communication requirements,” “communication robustness,” and “computational requirements” are not as critical considering that both coverage algorithms are demonstrated successfully in live-fly field experiments. The scenario used for the simulation is the same as the Field Experiment 2.

4.1 Time for Complete Coverage Analysis
A total of 150 simulation runs are used for the analysis. There are 30 simulation runs for each set of the five experiment parameters. The times for complete coverage for all the simulations runs are consolidated in a scatter chart shown in Figure 4.1. The mean for each set of experiment parameters is indicated on the chart along with the 95% confidence intervals upper and lower limits.

4.1.1 Having More Searchers Does Not Guarantee Shorter Time for Complete Coverage
Figure 4.1 seems to indicate that having more searchers when using the greedy coverage algorithm reduces the time for complete coverage. However, the only statistically significant results to support this hypothesis are when six searchers using the greedy coverage algorithm are compared to four searchers using the same greedy coverage algorithm.

When using the fixed lane coverage algorithm, the opposite was actually true. The four searchers using the fixed lane coverage algorithm completed the coverage in significantly less time than the six searchers using the fixed lane coverage algorithm. This is a counterintuitive finding. The reason why the six searchers took more time than four searchers was because of the searchers’ starting position (standby point). The starting position was
east of the search area. This placed the searchers along the row’s axis and allowed the four searchers to “slide” into the search area immediately, as shown in Figure 4.2. On the other hand, when using six searchers, the searchers needed to pre-position at one end of the column before they could enter the search grid; this means one of the six searchers always needed to fly across the search area to the last column on the other side before flying down or up the column. This is shown as UAV “1” in Figure 4.3.

It should also be noted that were the search area re-oriented (i.e., six rows versus four columns), the six searchers’ time for complete coverage would have outperformed the four searchers’ time for complete coverage by a wide margin because the six searchers would have less transit time and covered the search area more efficiently.
This finding suggests that the relative position of the searchers’ starting position to the search area can have a significant impact to the time for complete coverage. A potential improvement to the fixed lane coverage algorithm is the search master determining whether the search area will better searched by row or column. The search master considers the number of searchers and the orientation of the search area before choosing to search by row or column.

### 4.1.2 Consistent Performance

The standard deviations for the greedy coverage algorithm with four, five, and six searchers are 5.11, 4.58, and 3.56 respectively. The standard deviation for the fixed lane algorithm with four and six searchers are 1.65 and 3.62, respectively. A smaller standard deviations means that there is less variance in the results of that experiment parameter. This is shown in Figure 4.1.

The greedy coverage algorithm gives more consistent results when more searchers are involved in the search. For the fixed lane coverage algorithm, having more searchers gives less consistent results. The reason for this difference is because the six searchers using fixed lane coverage algorithm have a longer transit time than the four searchers using the fixed
lane coverage algorithm. The additional variance in the searchers’ transit time reduces the consistency of the time for complete coverage.

4.1.3 Shortest Time for Complete Coverage

From Figure 4.1, the shortest time for complete coverage is the fixed lane coverage algorithm with four searchers. The fixed lane coverage algorithm with six searchers fared poorly because of their starting position. The greedy coverage algorithms had a longer time for complete coverage because of the overlaps. This is shown in Figure 4.4. Based on the equation derived in Section 1.3,

\[ T = \frac{A}{VWN} \]

where the “perfect” area search time for the scenario with four searchers is as follows:

\[ T = \frac{900 \text{ m} \times 600 \text{ m}}{20 \text{ m/s} \times 150 \text{ m} \times 4} = 45 \text{ seconds} \] (4.1)

The mean of the fixed lane coverage algorithm with four searchers is 49.5 seconds, which
is close to the perfect area search time. It is still possible for the greedy coverage algorithm to have a shorter time for complete coverage though. There are 24 search cells in the search area. If the greedy coverage algorithm has 24 searchers, all the searchers only have to fly directly to one search cell and they will achieve a shorter time than the fixed lane coverage algorithm. There will be significant overlap in coverage though.

4.2 Missed Coverage and Overlap Analysis
The fixed lane coverage algorithm prevents any overlapping between the searchers by allocating dedicated lanes to the searchers. The sweep width is the same as the search cells interval so there are no overlaps and gaps. The gap that appears in the corner when a searcher turns (Figure 2.3) is also prevented as the fixed lane coverage algorithm does not require any of the searchers to make any turns. The greedy coverage algorithm cannot prevent overlap and missing coverage due to turns by the searchers.

4.3 Load Balancing Analysis
The fixed lane coverage algorithm divides the search cells equally among the searchers so the load is balanced evenly among the searchers. The greedy coverage algorithm, on the
other hand, does not consider load balancing at all. The results from the 90 greedy coverage algorithm simulation runs, however, indicates that the load between the searchers is shared more evenly when the number of searchers increases. This result is shown in Figure 4.5.

![Figure 4.5: Comparison of greedy coverage algorithm load balancing by number of searchers](image)

(Four searchers)

(Five searchers)

(Six searchers)
This thesis explains the rise of swarm UAVs and the motivation for using swarm UAVs to conduct an area search. The thesis identified the key challenges of coordinating swarm UAVs for an area search. The thesis examined possible approaches to overcoming these challenges, and the most suitable approach for the thesis was selected for implementation. A new swarm search controller, programmed to run onboard swarm UAVs, validated the selected approach with live-fly field experiments. Different coverage algorithms, programmed into the swarm search controller, were evaluated using simulation. Subsequent live-fly field experiments validated the coverage algorithms.

5.1 Summary

The technological advances brought by the smartphone revolution paved the way for the proliferation of hobbyist UAVs. The accessibility of hobbyist UAVs in recent years brings the opportunity for researchers to deploy swarm UAVs. Swarm UAVs have the potential to improve current surveillance or search missions by collecting data from multiple vantage points simultaneously. Researchers have been steadily increasing the swarm size and complexity of the swarm UAVs they are studying. However, autonomous outdoor swarm behaviors are still restricted to formation flying or “flocking.” This thesis seeks to advance the field by coordinating autonomous outdoor swarm UAVs for an area search.

Theoretical work identifies that the key challenges to coordinating swarm UAVs for area searches are, preventing the swarm UAVs from overlapping with each other during the search and ensuring that all of the search area is covered. Several approaches were discussed to overcome the two challenges. The following approaches were found to be most suitable for the thesis to implement. To prevent overlaps, a global search map of all swarm UAV positions was maintained through sharing of positional information using a wireless link. A centralized master searcher used the global search map to decide the search path for itself and other swarm UAVs. The coverage of the search area was tracked by discretizing the search area.
For this thesis, a new swarm search controller was designed and programmed to coordinate swarm UAVs for an area search. The swarm search controller runs onboard the ARSENL fixed-wing swarm UAV (Ritewing Zephyr II). The swarm search controller’s role in the ARSENL UAV System Architecture is explained, including details about the controller’s software design. The swarm search controller coordinated area search missions successfully during live-fly field experiments and validated the thesis’ approaches.

Two coverage algorithms were implemented in the swarm search controller and validated during the live-fly experiments. The first coverage algorithm is a simple greedy algorithm that assigns discretized search cells to swarm UAVs based on closest distance. The second coverage algorithm was a fixed lane algorithm that minimizes overlaps by pre-allocating discretized search cells to swarm UAVs. For this study, 150 simulation runs were made using both coverage algorithms with different numbers of searchers. The results from the simulation runs were analyzed for statistical significance, and future live-fly experiments are planned to validate the simulation findings.

5.2 Main Findings

This thesis’ main finding is that it is possible for outdoor swarm UAVs to coordinate and generate dynamic flight paths using onboard processing to complete an area search mission. Theoretical work shows the key problems to solve for coordinating swarm UAVs to conduct an area search. The problems are preventing the swarm UAVs from overlapping with each other during the search and ensuring coverage of all of the search area.

Analysis of the simulation results reveals that having more searchers does not guarantee a shorter time for complete coverage. The relative position of the searchers’ starting position to the search area and the orientation of the search area can have a significant impact to the time for complete coverage. The greedy coverage algorithm had more consistent results when more searchers were involved in the search, while having fewer searchers for the fixed lane coverage algorithm gave more consistent results. Overall, the fixed lane coverage algorithm usually requires a shorter time for complete coverage than the greedy coverage algorithm.
5.3 Future Work and Recommendation

A new paradigm is opened up by this thesis as it shows that swarm UAVs are ready for software experimentation. The thesis demonstrated this readiness by experimenting with different coverage algorithms in the swarm search controller. Further work to improve the software algorithms in the swarm search controller is recommended.

The fixed lane coverage algorithm produces a time for complete coverage that is very close to the perfect time for complete coverage, calculated using the equation explained in Section 1.3:

\[ T = \frac{A}{VWN} \]

However, the fixed lane coverage algorithm requires the number of searchers to be more than the number of lanes from the shortest side of the rectangular search area. A potential future work is to find another coverage algorithm that produces the same or better time for complete coverage and does not have the limitation on the size of the search area.

The swarm search controller uses a centralized master searcher decision-making approach. If the size of the swarm UAVs increases a lot, a decentralized decision-making approach may be necessary to prevent the swarm UAVs from having a bottleneck at the master searcher. A potential future work is to convert the swarm search controller to be decentralized.

It is foreseeable that the swarm UAVs will be used to search for targets rather than just area searches. However, real-time video surveillance is a complex task and requires heavy processing. The ODROID U3 autonomy payload onboard the ARSENL swarm UAVs may not have enough processing power to handle the task. A “smart” camera can be utilized to handle the task. The smart camera captures image and also processes them [30]. A potential future work is to integrate the smart camera with the ODROID U3 and have the swarm UAVs search for a target.

The thesis findings are based on the 900 meter by 600 meter search area. The search area used is small enough to fit within Wi-Fi radio range of any swarm UAVs. If the search area expands beyond Wi-Fi radio range, the master searcher may not receive the positional information from the slave searchers to form the correct global search map. Likewise,
the slave searchers may not receive the search cell assignment from the master searcher. A potential future work is to implement the hierarchical regional search maps, which are shown in Table 2.1, to extend beyond Wi-Fi radio range.

5.4 Conclusion

The thesis has shown that research in the swarm UAV field is no longer about realizing the swarm UAVs. The field has matured enough that researchers can shift their focus to pursuing actual real-world applications for swarm UAVs. The live-fly validated swarm search controller represents the first steps in advancing this new focus. By experimenting with different coverage algorithms in the swarm search controller, the thesis opens up a new paradigm by proving that swarm UAVs are ready for software experimentation. This research will undoubtedly be joined by many new and exciting contributions from future researchers.
APPENDIX: Source Code

The soft copy of this code can be accessed at: http://faculty.nps.edu/thchung (under Resources)

sourceCodes/Swarm_Searcher_ctrl.py

```python
# Standard python library imports
import sys
import time
import math
from argparse import ArgumentParser

# ROS library imports
import rospy
import std_msgs.msg as stdmsg

# ACS imports
import ap_msgs.msg as apmsg
import ap_srvs.srv as apsrv
import ap_lib.ap_enumerations as enums
import ap_lib.nodeable as nodeable
import ap_lib.waypoint_controller as wp_controller
import autopilot_bridge.srv as apbrgsrv
import autopilot_bridge.msg as apbrgmsg
from ap_lib.gps_utils import *

# Base name for node topics and services
NODENAME = 'swarm_searcher'
```
# Global variables (constants)

# Local Enumeration for swarming uav search states

SEARCH_READY = 1  # Waiting search area
SEARCH_INGRESS = 2  # Received search area and transiting to it
SEARCH_ACTIVE = 3  # Searching in search area
SEARCH_EGRESS = 4  # search end/completed and transiting to staging area
SEARCH_TRACKING = 5  # search uav detected something is tasked to track it
SEARCH_FAULT = 99  # search uav has shown a fault and kept out of the operation

SEARCH_CELL_FULLY_VISITED = 0  # All cells visited
SEARCH_CELL_FULLY_ASSIGNED = 1  # All cells assigned
SEARCH_CELL_STILL_AVAILABLE = 2  # Cells still available for assignment

SEARCH_CELL_RADIUS = 150  # Interval between search cell in metres
SEARCH_ALTITUDE = 50  # Height to conduct search in metres
SEARCH_SAFETY_ALTITUDE_INTERVAL = 15  # Height between UAV of the same cell in metres
SEARCH_CELL_THRESHOLD = 100  # Distance to determine that cell is searched (must be greater than infinite loiter radius)
SEARCH_ARRIVAL_THRESHOLD = 150  # Distance to start assigning cells in search grid to UAV
SEARCH_MAX_THRESHOLD = 100000000  # Max number for comparison
SEARCH_TIMEOUT_INTERVAL = 15  # 15 secs interval to check if searcher get closer to assigned waypoint
SEARCH_TIMEOUT_STRIKES = 3  # Number of strikes a UAV can get from timeout before being thrown out of the search operation
SEARCH_TIMEOUT_LOITERDISTANCEBUFFER = 10  # Buffer for UAV doing loiter such as Egressing after reaching the swarming waypoint

SEARCH_SWARM_SEARCH_READY_STATE = enums.SWARM_READY  # enum state when UAV is ready for swarm behavior
SEARCH_SWARM_SEARCH_ACTIVE_STATE = enums.SWARM_ACTIVE  # enum state when UAV is executing an active swarm behavior

# "struct" to store data for search search cell
class cell()
  LLA=None
  grid=None
  assigned=False
  assignedTo=None
  visited=False
  visitedBy=None
  visitedTimeStamp=None

# "struct" for Waypoint Msg
class wpMsg()
  recipientVehicle_id = None
  waypoint = apbrgmsg.LLA()
  searchCell_x = None
  searchCell_y = None

# "struct" to info of each search UAV
class searchUAVdata()
  status=None
  subSwarmID=None
  receivedState=None
  pose=None
  assignedAltitude=None
  assignedCell=None
  assignedTime=None
# Class member functions:

class SwarmSearcher(wp_controller.WaypointController):

    search_master_searcher_id = None  # variable for master searcher id
    # 2D Array to store the 'cell' objects
    searchGrid = None
    # Search Swarm UAV status
    searchUAVMap = dict()

    # Assume Search Grid message states BottomLeft LLA plus length and width
    bottomLeftLLA = None
    centerofSearchGridLLA = None
    cols = None
    rows = None

    # default search Algo enum
    # selectedSearchAlgo = 1
    selectedSearchAlgo = None
    numOfSearcher = 0
    rowSearch = False
    prePlannedPathPlanned = 0
# SearchSubSwarmID

searchSubSwarmID = 0

numOfRunsToDo = 0  # 0 means unlimited
numOfRunsDone = 0  #

# @param nodename: name of the ROS node in which this object exists
# @param ownAC: ID (int) of this aircraft
def __init__(self, nodename, ownAC, args):
    wp_controller.WaypointController.__init__(self, nodename, 
                                          enums.SWARM_SEARCH_CTRLR)
    self.DEBUG_PRINT = False
    self.WARN_PRINT = False
    # Boolean flag to determine Centralised search node
    self.SEARCH_MASTER = False
    self.SEARCH_STATUS = SEARCH_READY
    self.ownID = ownAC
    self.ExecuteMasterSearcherBehavior = False

    # Used to determine altitude for wpt orders
    self._wp_rel_alt = None
    self._crnt_wp_id = None

    # Counter used to assign safety altitude to each UAV
    self.uavAltitudeCounter = 0

    # Time when search operation begins
    self.search_Operation_StartTime = None
    # Time when current search runs begins
self.search_Run_StartTime = None
# Boolean when first search cell found
self.search_Run_firstSearch = False

# Boolean flag to determine if any waypoint message to send
self.wpmsgQueued = False

self._getWpSrvProxy = None
self._deactivateSrvProxy = None
self._swarmSearchPublisher = None
self._assignedSearchMessage = apmsg.SwarmSearchWaypointList()

# Implementation of parent class virtual functions

# @param params: no additional parameters required by this method
def callbackSetup(self, params=[]):
    self.createSubscriber("swarm_uav_states", apmsg.SwarmStateStamped, \
        self._process_swarm_uav_states)
    self.createSubscriber("recv_swarm_search_waypoint", apmsg.SwarmSearchWaypointList, \
        self._process_swarm_search_waypoint)
    self.createSubscriber("swarm_search_setup", apmsg.SwarmSearchOrderStamped, \
        self._process_swarmSearch_setup)
    self.createSubscriber("status", apbrgmsg.Status, \
        self.sub_autopilot_status_update)

def publisherSetup(self, params=[]):
self._swarmSearchPublisher = \
    self.createPublisher("send_swarm_search_waypoint", apmsg.SwarmSearchWaypointList, 1)

def serviceSetup(self, params=[]):
    return

def serviceProxySetup(self, params=[]):
    self._deactivateSrvProxy = \
        self.createServiceProxy("set_swarm_behavior", apsrv.SetInteger)
    self._getWpSrvProxy = \
        self.createServiceProxy("wp_getrange", apbrgsrv.WPGetRange)

def runController(self):
    if self.ExecuteMasterSearcherBehavior == True:
        self.wpmsgQueued = False #flag to determine if any waypoint message to send
del self._assignedSearchMessage.waypoints[:] # Clear current message contents

    for vehicle in self.searchUAVMap:
        self.log_dbug("\033[94m TimeLoop: UAV \033[96m" + str(vehicle) + \
            " ] \033[94m Received State [" + str(self.searchUAVMap[vehicle].
            receivedState) + \
            "] pose: \033[0m" + str(self.searchUAVMap[vehicle].pose))

    if self.SEARCH_STATUS == SEARCH_READY:
        if self.numOfRunsToDo == 0 or self.numOfRunsDone < self.numOfRunsToDo:
            self.SEARCH_STATUS = SEARCH_INGRESS
            self.search_Run_StartTime = time.time()
            self.log_dbug("\033[94mSwarm Search Master \033[96m" + str(self.ownID) + \
                "]\033[94m current search run \033[96m" + str(self.
numOfRunsDone + 1) + \\
"033[94m started on 033[96m + \\
str(time.asctime(time.localtime(time.time()))) + "033[0m"

    self.search_Run_firstSearch = False

elif self.SEARCH_STATUS == SEARCH_INGRESS:
    self._assign_ready_UAVs_to_searchGrid()
    self._check_timeout()
    # Check if any of the search uav has reached the search area, set search status to active if true
    reached = self._assign_ingress_UAVs_to_search()

    if reached == True:
        self.SEARCH_STATUS = SEARCH_ACTIVE

elif self.SEARCH_STATUS == SEARCH_ACTIVE:
    self._assign_ready_UAVs_to_searchGrid()
    self._assign_ingress_UAVs_to_search()
    self._check_timeout()

    # Check if search is completed (all cells visited) set status to SEARCH_EGRESS
    searchComplete = self._check_search_operation()

    if searchComplete == True:
        self.numOfRunsDone += 1
        currentTime = time.time()
        timeDelta = currentTime - self.search_Run_StartTime
        self.log_dbug("033[94mSwarm Search Master 033[96m" + str(self.ownID) + \
        "]033[94m current search run 033[96m" + str(self.
        numOfRunsDone) + \
        "033[94m completed after 033[96m" + \
        "033[94m started on 033[96m + \"033[0m"")
"{0:.1f}".format(timeDelta) + "\033[94m secs\033[0m")

for j in range(self.rows):
    for i in range(self.cols):
        timeDelta = self.searchGrid[i][j].visitedTimeStamp - self.
                    search_Run_StartTime
        self.log_dbug("\033[93m[i] \033[96m" + str(i) + "," + str(j) + \" \
                        ) \033[94m Visit by \033[96m" + str(self.searchGrid[i][
                        j].visitedBy) + \" \
                        ) \033[94m after \033[96m" + "{0:.1f}".format(timeDelta)
                        + \" \
                        ) \033[94m secs\033[0m")

self.SEARCH_STATUS = SEARCH_EGRESS

# set all ingress/active uav to egress
for vehicle in self.searchUAVMap:
    if self.searchUAVMap[vehicle].status == SEARCH_INGRESS or \n    self.searchUAVMap[vehicle].status == SEARCH_ACTIVE:
        self.searchUAVMap[vehicle].status = SEARCH_EGRESS
        self.searchUAVMap[vehicle].assignedTime = time.time()
        lla = apbrmsg.LLA()
        lla.lat = self.searchUAVMap[vehicle].originalLLA[0]
        lla.lon = self.searchUAVMap[vehicle].originalLLA[1]
        lla.alt = self.searchUAVMap[vehicle].assignedAltitude
        if vehicle == self.search_master_searcher_id:
            self.publishWaypoint(lla)

elif self.SEARCH_STATUS == SEARCH_EGRESS:
# check if all search slaves has reach original LLA and set status to search ready
# and clear search grid
self._check_timeout()

egressComplete = True
for vehicle in self.searchUAVMap:
    if self.searchUAVMap[vehicle].status == SEARCH_EGRESS:
        egressComplete = False
        currentPose = self.searchUAVMap[vehicle].pose
        tgt_cell = self.searchUAVMap[vehicle].originalLLA
        if gps_distance(currentPose[0], currentPose[1], tgt_cell[0], tgt_cell[1]) < SEARCH_ARRIVAL_THRESHOLD:
            if vehicle != self.search_master_searcher_id:
                _wpMsg = wpMsg()
                _wpMsg.recipientvehicle_id = vehicle
                _wpMsg.waypoint.lat = self.searchUAVMap[vehicle].originalLLA[0]
                _wpMsg.waypoint.lon = self.searchUAVMap[vehicle].originalLLA[1]
                _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
                _wpMsg.searchCell_x = 255
                _wpMsg.searchCell_y = 255
                self._assignedSearchMessage.waypoints.append(_wpMsg)
                self.wpmsgQueued = True
            else:
                self.searchUAVMap[vehicle].status = SEARCH_READY
        if egressComplete == True:
            self.log_dbug("\033][94mSwarm Search Master \033][96m" + str(self.ownID) + \}
"\033[94m Master searcher has egressed and all slave searchers deactivated\033[0m"

self.SEARCH_STATUS = SEARCH_READY
self.searchGrid = None

if self.numOfRunsDone == self.numOfRunsToDo:
    currentTime = time.time()
    timeDelta = currentTime - self.searchOperation_StartTime
    self.log_dbug("\033[94m Swarm Search Master \033[96m[" + str(self.ownID) + \\
    \
    "\033[94m entire search operation completed with \033[96m" + \n    str(self.numOfRunsToDo) + " \033[94m runs \033[96m after \n    \033[96m" + \n    "{0:.3f}\033[0m") . format(timeDelta) + "\033[94m secs\033[0m")

    self.SEARCH_STATUS = SEARCH_READY
    self.searchGrid = None
    self.ExecuteMasterSearcherBehavior = False
    self._deactivateSrvProxy(enums.SWARM_STANDBY)

else:
    self._activate_swarm_search()

    if self.wpmsgQueued == True: # A new network waypoint cmd is triggered this tick
        self._swarmSearchPublisher.publish(self._assignedSearchMessage)

# Object-specific functions
# activate_swarm_search

```python
def activate_swarm_search(self, searchAlgoEnum):
    self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \\
    "]\033[94m generate Search Grid:\033[0m")
    
    # Create the searchGrid
    self.searchGrid = [[cell() for j in range(self.rows)] for i in range(self.cols)]
    # populate the searchGrid
    for j in range(self.rows):
        for i in range(self.cols):
            self.searchGrid[i][j].grid = [i * SEARCH_CELL_RADIUS, j * SEARCH_CELL_RADIUS]
            self.searchGrid[i][j].LLA = gps_offset(self.bottomLeftLLA[0], 
            self.bottomLeftLLA[1], 
            i * SEARCH_CELL_RADIUS, 
            j * SEARCH_CELL_RADIUS)
            
            self.log_dbug("\033[93m[" + str(i) + "," + str(j) + \\
            "] \033[36m Grid: " + str(self.searchGrid[i][j].grid) + \\
            "] \033[0mLLA: " + str(self.searchGrid[i][j].LLA))
            
    self.centerofSearchGridLLA = self.searchGrid[self.cols/2][self.rows/2].LLA
    
    if searchAlgoEnum == 2:  # Preplanned swarm search selected
        self.numOfSearcher = 0
        self.rowSearch = False
        for vehicle in self.searchUAVMap:
            if self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
                self.searchUAVMap[vehicle].preplannedPath = self.numOfSearcher
                self.numOfSearcher += 1
            
            if self.numOfSearcher < self.rows and self.numOfSearcher < self.cols:
                self.log_dbug("\033[94mSwarm Search Master \033[96m["+str(self.ownID)+"]\033[94m"
                
                " Preplanned swarm search algo requires searchers to be at least ")
```

---

296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
the number of rows or columns of the Search Grid: 033[0m"

self.log_dbg("033[94m Swarm Search Master 033[96m["+str(self.ownID)+"]033[94m"
+ \\
"Defaulting to greedy swarm search algo.033[0m")

else:
    self.prePlannedPathPlanned = 0
    self.selectedSearchAlgo = 2
    if self.rows > self.cols:
        if self.numOfSearcher >= self.rows:
            self.rowSearch = True
        else:
            self.rowSearch = False
    else:
        if self.numOfSearcher >= self.cols:
            self.rowSearch = False
        else:
            self.rowSearch = True
    if self.rowSearch == True:
        self.log_dbg("033[94m Swarm Search Master 033[96m["+str(self.ownID)+"]033[94m"
        " Preplanned swarm search algo activated with \
        " rows033[94m search033[0m"
        self.prePlannedPathPlanned = self.rows
    else:
        self.log_dbg("033[94m Swarm Search Master 033[96m["+str(self.ownID)+"]033[94m"
        " Preplanned swarm search algo activated with \
        " columns033[94m search033[0m")
def _assign_ready_UAVs_to_searchGrid(self):  
    for vehicle in self.searchUAVMap:
        if self.searchUAVMap[vehicle].status == SEARCH_READY and \
           self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
            self.searchUAVMap[vehicle].status = SEARCH_INGRESS
            if self.searchUAVMap[vehicle].originalLLA == None:
                self.searchUAVMap[vehicle].originalLLA = self.searchUAVMap[vehicle].pose  
                self.searchUAVMap[vehicle].assignedLLA = self.centerOfSearchGridLLA  
                self.searchUAVMap[vehicle].assignedCell = [self.cols/2, self.rows/2]  
                self.searchUAVMap[vehicle].assignedTime = time.time()  
                self.searchUAVMap[vehicle].timeoutdistance = gps_distance(self.searchUAVMap[vehicle].pose[0], \
                                                               self.searchUAVMap[vehicle].pose[1], \
                                                               self.centerOfSearchGridLLA[0], \
                                                               self.centerOfSearchGridLLA[1])
            self.searchUAVMap[vehicle].timeoutTime = time.time()  
        if vehicle == self.search_master_searcher_id:
            lla = apbrgmsg.LLA()  
            lla.lat = self.centerOfSearchGridLLA[0]  
            lla.lon = self.centerOfSearchGridLLA[1]  
            lla.alt = self.searchUAVMap[vehicle].assignedAltitude
self.publishWaypoint(lla)

else:
    _wpMsg = wpMsg()
    _wpMsg.recipientVehicle_id = vehicle
    _wpMsg.waypoint.lat = self.centerOfSearchGridLLA[0]
    _wpMsg.waypoint.lon = self.centerOfSearchGridLLA[1]
    _wpMsg.alt = self.searchUAVMap[vehicle].assignedAltitude
    _wpMsg.searchCell_x = self.cols/2
    _wpMsg.searchCell_y = self.rows/2
    self.assignedSearchMessage.waypoints.append(_wpMsg)
    self.wpmqgQueued = True
    self.log_dbug("\033[94m" + " Swarm Search node: Searcher \033[96m" + \n        str(vehicle) + "]\033[94m ingress to search area:" + ":033[0m"")

def _assign_ingress_UAVs_to_search(self):
    reached = False
    currentTime = time.time()
    for vehicle in self.searchUAVMap:
        if self.searchUAVMap[vehicle].status == SEARCH_INGRESS and \
            self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
            currentPose = self.searchUAVMap[vehicle].pose
            for j in range(self.rows):
                for i in range(self.cols):
                    if reached == False:
                        tgt_cell = self.searchGrid[i][j].LLA
                        if gps_distance(currentPose[0], currentPose[1], \
                            tgt_cell[0], tgt_cell[1]) < SEARCH_ARRIVAL_THRESHOLD:
reached = True
self.searchUAVMap[vehicle].status = SEARCH_ACTIVE
if self.selectedSearchAlgo == 2:
    if self.searchUAVMap[vehicle].preplannedPath < self.prePlannedPathPlanned:
        if self.rowSearch == True:
            tgt_cell = self.searchGrid[0][self.searchUAVMap[vehicle].preplannedPath].LLA
            tgt_cell2 = self.searchGrid[self.cols - 1][self.searchUAVMap[vehicle].preplannedPath].LLA
            if gps_distance(currentPose[0], currentPose[1], tgt_cell[0], tgt_cell[1]) <
                gps_distance(currentPose[0], currentPose[1], tgt_cell2[0], tgt_cell2[1]):
                self.searchUAVMap[vehicle].assignedCell = [0, self.searchUAVMap[vehicle].preplannedPath]
                self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[0m"")
                str(vehicle) + ";\033[94m assigned entry via: \033[0m"");
                "\033[93m"+str(0)+","+str(self.searchUAVMap[vehicle].preplannedPath) + 
                ";\033[0m"")
        else:
            tgt_cell = self.searchGrid[self.cols - 1][self.searchUAVMap[vehicle].preplannedPath].LLA
            self.searchUAVMap[vehicle].assignedCell = [self.cols - 1, self.searchUAVMap[vehicle].]
preplannedPath]
self.log_debug("\033[94m" + "Swarm Search node: Searcher \033[96m" + str(vehicle) + \\n"\033[94m assigned entry via: " + \\n"\033[93m" + str(self.cols - 1) + \\n"," + str(self.searchUAVMap[vehicle].preplannedPath) +"\033[0m")
else:

tgt_cell = self.searchGrid[self.searchUAVMap[vehicle].preplannedPath][0].LLA
tgt_cell2 = self.searchGrid[self.searchUAVMap[vehicle].preplannedPath][self.rows - 1].LLA
if gps_distance(currentPose[0], currentPose[1], tgt_cell[0], tgt_cell[1]) <
gps_distance(currentPose[0], currentPose[1], tgt_cell2[0], tgt_cell2[1]):
  self.searchUAVMap[vehicle].assignedCell = [self.searchUAVMap[vehicle].preplannedPath, 0]
  self.log_debug("\033[94m" + "Swarm Search node: Searcher \033[96m" + \\n  str(vehicle) +"\033[94m assigned entry via: " + \\n  "\033[93m" + str(self.searchUAVMap[vehicle].preplannedPath) +"," + str(0) +"\033[0m")
else:

tgt_cell = self.searchGrid[self.searchUAVMap[vehicle].preplannedPath][self.rows - 1].LLA
self.searchUAVMap[vehicle].assignedCell = [self.
searchUAVMap[vehicle].preplannedPath, self.rows−1]
self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[+str(vehicle)+ \"
]\033[94m assigned entry via: " + 
"\033[93m[+str(self.searchUAVMap[vehicle].preplannedPath)+","+str(self.rows−1)"
]\033[0m")

else: #terminate extra searchers
    self.searchUAVMap[vehicle].assignedCell = [254,254]
tgt_cell = self.searchUAVMap[vehicle].originalLLA
self.searchUAVMap[vehicle].status = SEARCH_EGRESS
self.log_dbug("\033[94m" + "Swarm Search node: Extra search \033[96m[+str(vehicle)+ 
"]\033[94m removed from search\033[0m")
else:
    self.searchUAVMap[vehicle].assignedCell = [i,j]
self.log_dbug("\033[94m" + "Swarm Search node: Searcher 
[+str(vehicle)+ \"
]\033[94m assigned entry via: " + 
"\033[93m[+str(i)+","+str(j)"+\]033[0m")
self.searchUAVMap[vehicle].assignedTime = currentTIme
self.searchUAVMap[vehicle].assignedLLA = tgt_cell
self.searchUAVMap[vehicle].timeoutDistance = \
gps_distance(currentPose[0], currentPose[1], tgt_cell[0], 
tgt_cell[1])
self.searchUAVMap[vehicle].timeoutTime = currentTIme

#assign UAV to tgt_cell
if vehicle == self.search_master_searcher_id:
    lla = apbrgmsg.LLA()
    lla.lat = tgt_cell[0]
    lla.lon = tgt_cell[1]
    lla.alt = self.searchUAVMap[vehicle].assignedAltitude
    self.publishWaypoint(lla)

else:
    _wpMsg = wpMsg()
    _wpMsg.recipientvehicle_id = vehicle
    _wpMsg.waypoint.lat = tgt_cell[0]
    _wpMsg.waypoint.lon = tgt_cell[1]
    _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
    _wpMsg.searchCell_x = self.searchUAVMap[vehicle].assignedCell[0]
    _wpMsg.searchCell_y = self.searchUAVMap[vehicle].assignedCell[1]
    self._assignedSearchMessage.waypoints.append(_wpMsg)
    self.wpmsgQueued = True

return reached

def _check_search_operation(self):
    runOutofCells=False
    currentTime = time.time()

    for vehicle in self.searchUAVMap:
        if self.searchUAVMap[vehicle].status == SEARCH_ACTIVE and \

self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
currentPose = self.searchUAVMap[vehicle].pose
tgt_cell = self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].LLA

if gps_distance(currentPose[0], currentPose[1], tgt_cell[0], tgt_cell[1]) < SEARCH_CELL_THRESHOLD:
    if self.search_Run_firstSearch == False:
        self.search_Run_firstSearch = True
        timeDelta = currentTime - self.search_Run_StartTime
        self.search_Run_StartTime = currentTime
        self.log_debug("033[94mSwarm Search Master 033[96m[" + str(self.ownID) + "] 033[94m actual search started after 033[96m" + \
        "{0:.1f}".format(timeDelta)+"033[94m secs\033[0m")
    if self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].visited == False:
        self.log_debug("033[94m + "Swarm Search node: Searcher 033[96m" + \
        str(vehicle) + "]033[92m searched 033[94m cell 033[93m" + \
        str(self.searchUAVMap[vehicle].assignedCell) + "]033[0m")
    self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].visited = True
    self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].visitedBy = vehicle
    self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].visitedTimeStamp = time.time()

else:
    self.log_debug("033[94m + "Swarm Search node: Searcher 033[96m" + \

```
str(vehicle) + " ]\033[94m visited cell \033[93m" + \\
str(self.searchUAVMap[vehicle].assignedCell) + " ]\033[0m"
result = self._assign_search_UAVs_to_nextCell(vehicle)

if result[0] == SEARCH_CELL_STILL_AVAILABLE: # assign UAV to tgt_cell
    self.searchUAVMap[vehicle].assignedCell = [result[1], result[2]]
    self.searchUAVMap[vehicle].assignedTime = currentTime
    self.searchGrid[result[1]][result[2]].assigned = True
    self.searchGrid[result[1]][result[2]].assignedTo = vehicle
    self.searchUAVMap[vehicle].assignedLLA = self.searchGrid[result[1]][
        result[2]].LLA
    self.searchUAVMap[vehicle].timeoutDistance = \
        gps_distance(currentPose[0], currentPose[1], 
        self.searchGrid[result[1]][result[2]].LLA[0], 
        self.searchGrid[result[1]][result[2]].LLA[1])
    self.searchUAVMap[vehicle].timeoutTime = currentTime
    self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \\
    str(vehicle) + " ]\033[94m assigned cell \033[93m" + \\
    str(self.searchUAVMap[vehicle].assignedCell) + " ]\033[0m"

if vehicle == self.search_master_searcher_id:
    lla = apbrgmsg.LLA()
    lla.lat = self.searchGrid[result[1]][result[2]].LLA[0]
    lla.lon = self.searchGrid[result[1]][result[2]].LLA[1]
    lla.alt = self.searchUAVMap[vehicle].assignedAltitude
    self.publishWaypoint(lla)

else:
    _wpMsg = wpMsg()
_wpMsg.recipientVehicleId = vehicle

_wpMsg.waypoint.lat = self.searchGrid[result[1]][result[2]].LLA[0]
_wpMsg.waypoint.lon = self.searchGrid[result[1]][result[2]].LLA[1]
_wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
_wpMsg.searchCellX = result[1]
_wpMsg.searchCellY = result[2]
self._assignedSearchMessage.waypoints.append(_wpMsg)
self.wpmsgQueued = True

else:
    if result[0] == SEARCH_CELL_FULLY_VISITED:
        runOutOfCells = True
    self.searchUAVMap[vehicle].status = SEARCH_EGRESS
    self.searchUAVMap[vehicle].assignedTime = currentTime
    self.searchUAVMap[vehicle].assignedLLA = self.searchUAVMap[vehicle].originalLLA
    self.searchUAVMap[vehicle].timeoutDistance = \
        gps_distance(currentPose[0], currentPose[1], \
        self.searchUAVMap[vehicle].originalLLA[0], \
        self.searchUAVMap[vehicle].originalLLA[1])
    self.searchUAVMap[vehicle].timeoutTime = currentTime
    self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \ 
    str(vehicle) + "\033[94m returning home\033[0m")

if vehicle == self.searchMasterSearcherId:
    lla = apbrgmsg.LLA()  
    lla.lat = self.searchUAVMap[vehicle].originalLLA[0]
    lla.lon = self.searchUAVMap[vehicle].originalLLA[1]
    lla.alt = self.searchUAVMap[vehicle].assignedAltitude
self.publishWaypoint(lla)

else:
    _wpMsg = wpMsg()
    _wpMsg.recipientVehicleId = vehicle
    _wpMsg.waypoint.lat = self.searchUAVMap[vehicle].originalLLA[0]
    _wpMsg.waypoint.lon = self.searchUAVMap[vehicle].originalLLA[1]
    _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
    _wpMsg.searchCell_x = 254
    _wpMsg.searchCell_y = 254
    self._assignedSearchMessage.waypoints.append(_wpMsg)
    self.wpmsgQueued = True

return runOutofCells

def _assign_search_UAVs_to_nextCell(self, vehicleID):
    # assign new unvisited cell if possible
    cellResult = SEARCH_CELL_FULLY_VISITED
    currentCol = self.searchUAVMap[vehicleID].assignedCell[0]
    currentRow = self.searchUAVMap[vehicleID].assignedCell[1]
    tgt_cell = [0, 0]
    distance = SEARCH_MAX_THRESHOLD
    currentPose = self.searchUAVMap[vehicleID].pose
    if self.selectedSearchAlgo == 2:
        if self.rowSearch == True:  # find next closest cell in the same row
            for i in range(self.cols):
                if self.searchGrid[i][currentRow].visited == False:
                    if cellResult == SEARCH_CELL_FULLY_VISITED:
                        cellResult = SEARCH_CELL_FULLY_ASSIGNED
if self.searchGrid[i][currentRow].assigned == False:
    cellResult = SEARCH_CELL_STILL_AVAILABLE
    cell = self.searchGrid[i][currentRow].LLA
    tempDist = gps_distance(currentPose[0], currentPose[1], cell[0], cell[1])
    if tempDist < distance:
        distance = tempDist
        tgt_cell = [i, currentRow]
else:
    for j in range(self.rows):  # find next closest cell in the same col
        if self.searchGrid[currentCol][j].visited == False:
            if cellResult == SEARCH_CELL_FULLY_VISITED:
                cellResult = SEARCH_CELL_FULLY_ASSIGNED
                if self.searchGrid[currentCol][j].assigned == False:
                    cellResult = SEARCH_CELL_STILL_AVAILABLE
                    cell = self.searchGrid[currentCol][j].LLA
                    tempDist = gps_distance(currentPose[0], currentPose[1], cell[0], cell[1])
                    if tempDist < distance:
                        distance = tempDist
                        tgt_cell = [currentCol, j]
            if cellResult == SEARCH_CELL_FULLY_VISITED:  # To check if the whole search grid is fully visited
                for j in range(self.rows):
                    for i in range(self.cols):
                        if self.searchGrid[i][j].visited == False:
                            cellResult = SEARCH_CELL_FULLY_ASSIGNED
else:
    for j in range(self.rows):
for i in range(self.cols):
    if self.searchGrid[i][j].visited == False:
        if cellResult == SEARCH_CELL_FULLY_VISITED:
            cellResult = SEARCH_CELL_FULLY_ASSIGNED
        if self.searchGrid[i][j].assigned == False:
            cellResult = SEARCH_CELL_STILL_AVAILABLE
            cell = self.searchGrid[i][j].LLA
            tempDist = gps_distance(currentPose[0], currentPose[1], cell[0], cell[1])
            if tempDist < distance:
                distance = tempDist
                tgt_cell = [i, j]
    if cellResult == SEARCH_CELL_FULLY_VISITED:
        return [SEARCH_CELL_FULLY_VISITED, 0, 0]
    else:
        return [cellResult, tgt_cell[0], tgt_cell[1]]

def _check_timeout(self):
    # scan through all UAVs, if any did not get closer in the last interval, resend LLA
    currentTime = time.time()
    for vehicle in self.searchUAVMap:
        if self.searchUAVMap[vehicle].status in (SEARCH_INGRESS, SEARCH_ACTIVE, SEARCH_EGRESS):
            if self.searchUAVMap[vehicle].timeoutTime + SEARCH_TIMEOUT_INTERVAL < currentTime:

distFromTgt = gps_distance(self.searchUAVMap[vehicle].pose[0], \
    self.searchUAVMap[vehicle].pose[1], \
    self.searchUAVMap[vehicle].assignedLLA[0], \
    self.searchUAVMap[vehicle].assignedLLA[1])

# uav is closer since last check. Update latest distance and time
if distFromTgt < self.searchUAVMap[vehicle].timeoutdistance:
    if self.searchUAVMap[vehicle].status == SEARCH_EGRESS:
        self.searchUAVMap[vehicle].timeoutdistance = \n        distFromTgt + SEARCH_TIMEOUT_LOITERDISTANCEBUFFER
    else:
        self.searchUAVMap[vehicle].timeoutdistance = distFromTgt
def_timeoutTime = currentTime
self.searchUAVMap[vehicle].timeoutCounter = 0
else:
    # resend assigned lla
    if self.searchUAVMap[vehicle].timeoutCounter < SEARCH_TIMEOUT_STRIKES:
        self.searchUAVMap[vehicle].timeoutCounter += 1
        self.log_dbg("\033[94mSwarm Search node: \033[91mTIMEOUT "+ \
            str(self.searchUAVMap[vehicle].timeoutCounter) + \
            "\033[94m for UAV \033[96m" + str(vehicle) + \
            "] \033[94mResend LLA... \033[0m")
        self.searchUAVMap[vehicle].timeoutTime = currentTime

        if vehicle == self.search_master_searcher_id:
            lla = apbrgmsg.LLA()
            lla.lat = self.searchUAVMap[vehicle].assignedLLA[0]
lla.lon = self.searchUAVMap[vehicle].assignedLLA[1]
lla.alt = self.searchUAVMap[vehicle].assignedAltitude
self.publishWaypoint(lla)

else:
    _wpMsg = wpMsg()
    _wpMsg.recipientVehicle_id = vehicle
    _wpMsg.waypoint.lat = self.searchUAVMap[vehicle].assignedLLA[0]
    _wpMsg.waypoint.lon = self.searchUAVMap[vehicle].assignedLLA[1]
    _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
    _wpMsg.searchCell_x = self.searchUAVMap[vehicle].assignedCell[0]
    _wpMsg.searchCell_y = self.searchUAVMap[vehicle].assignedCell[1]
    self._assignedSearchMessage.waypoints.append(_wpMsg)
    self.wpmsgQueued = True

else:  # UAV not getting closer after all the timeout strikes, free
    self.log_dbg("\033[94mSwarm Search node: UAV \033[96m[" + str(vehicle) + 
    "] \033[91m STRIKEOUT! \033[0m")
    if self.searchUAVMap[vehicle].status == SEARCH_ACTIVE:
        # see if another UAV active. if so, continue so they can take
        # over the released cell
        if self._available_activeUAVHandover(vehicle) == True:
            self.searchGrid[ self.searchUAVMap[vehicle].assignedCell[0]][
                self.searchUAVMap[vehicle].assignedCell[1]].assigned =
                False
            self.searchGrid[ self.searchUAVMap[vehicle].assignedCell[0]][
                self.searchUAVMap[vehicle].assignedCell[1]].assignedTo =
None

```python
self.log_debug("\033[94mSearch cell \033[92m" + \\
str(self.searchUAVMap[vehicle].assignedCell) + \\
" \033[94m released for reassignment\033[0m")
```

```python
else:
    egressUAVID = \
    self._closest_egressUAVHandover(self.searchUAVMap[vehicle].assignedCell[0], \\
                                    self.searchUAVMap[vehicle].assignedCell[1])
```

```python
if egressUAVID != 0: #Found egress UAV to take over
    self.searchUAVMap[egressUAVID].status = SEARCH_ACTIVE
    self.searchUAVMap[egressUAVID].assignedCell = \
    self.searchUAVMap[vehicle].assignedCell
    self.searchUAVMap[egressUAVID].assignedTime = currentTime
    self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].
    assigned = True
```

```python
self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].assignedTo = egressUAVID
```

```python
self.searchUAVMap[egressUAVID].assignedLLA = self.
searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.searchUAVMap[vehicle].assignedCell[1]].LLA
```

```python
self.searchUAVMap[egressUAVID].timeoutdistance = \
gps_distance(self.searchUAVMap[egressUAVID].pose[0], \
```
self.searchUAVMap[egressUAVID].pose[1],
\nself.searchUAVMap[egressUAVID].assignedLLA[0], \nself.searchUAVMap[egressUAVID].assignedLLA[1])
self.searchUAVMap[egressUAVID].timeoutTime = currentTime
self.log_dbug("\033[94m" + "Swarm Search node: Egressed
Searcher \033[96m[" + self.log_str(egressUAVID) + "]\033[94m assigned to take over striedout
\033[96m[" + self.log_str(vehicle) + "]\033[94m assigned cell
\033[93m[" + self.log_str(self.searchUAVMap[vehicle].assignedCell) + "]\033[0m")

if egressUAVID == self.search_master_searcher_id:
    lla = apbrgmsg.LLA()
    lla.lat = self.searchUAVMap[egressUAVID].assignedLLA[0]
    lla.lon = self.searchUAVMap[egressUAVID].assignedLLA[1]
    lla.alt = self.searchUAVMap[egressUAVID].assignedAltitude
    self.publishWaypoint(lla)

else:
    _wpMsg = wpMsg()
_wpMsg.recipientVehicle_id = egressUAVID

_wpMsg.waypoint.lat = self.searchUAVMap[egressUAVID].assignedLLA[0]
_wpMsg.waypoint.lon = self.searchUAVMap[egressUAVID].assignedLLA[1]
_wpMsg.waypoint.alt = self.searchUAVMap[egressUAVID].assignedAltitude
_wpMsg.searchCell_x = self.searchUAVMap[egressUAVID].assignedCell[0]
_wpMsg.searchCell_y = self.searchUAVMap[egressUAVID].assignedCell[1]
self._assignedSearchMessage.waypoints.append(_wpMsg)
self.wpmsgQueued = True

else:
    self.log_dbug("\033[94mSearch \033[91mFAILED! \033[94mcell \033[92m" + \n    str(self.searchUAVMap[vehicle].assignedCell) + \n    " \033[94mreleased for assignment with no UAV to take over\033[0m")
    self.searchUAVMap[vehicle].status = SEARCH_FAULT

def _available_activeUAVHandover(self, faultyUAVID):
    available = False
    for vehicle in self.searchUAVMap:
        if self.searchUAVMap[vehicle].status in (SEARCH_INGRESS, SEARCH_ACTIVE):
            if vehicle != faultyUAVID:
available = True

return available

def _closest_egressUAVHandover(self, cell_x, cell_y):
closestUAV = 0
closestDistance = SEARCH_MAX_THRESHOLD
tgtLLA = self.searchGrid[cell_x][cell_y].LLA
for vehicle in self.searchUAVMap:
    if self.searchUAVMap[vehicle].status == SEARCH_EGRESS:
        distFromTgt = gps_distance(self.searchUAVMap[vehicle].pose[0], \
                                  self.searchUAVMap[vehicle].pose[1], \
                                  tgtLLA[0], tgtLLA[0])
        if distFromTgt < closestDistance:
            closestDistance = distFromTgt
            closestUAV = vehicle

return closestUAV

# ROS Subscriber callbacks – for this object
# Handle incoming swarm_uav_states messages
# @param swarmMsg: message containing swarm data (SwarmStateStamped)
def _process_swarm_uav_states(self, swarmMsg):
    for vehicle in swarmMsg.swarm:
        if vehicle.vehicle_id in self.searchUAVMap:
            self.searchUAVMap[vehicle.vehicle_id].pose = \
            [vehicle.state.pose.pose.position.lat, vehicle.state.pose.pose.position.lon]
self.searchUAVMap[vehicle.vehicle_id].subSwarmID = vehicle.subswarm_id

if vehicle.swarm_state != self.searchUAVMap[vehicle.vehicle_id].receivedState:
    if self.ExecuteMasterSearcherBehavior == True:
        if self.searchUAVMap[vehicle.vehicle_id].receivedState == "SEARCH_SWARM_SEARCH_ACTIVE_STATE" and vehicle.swarm_state == "SEARCH_SWARM_SEARCH_READY_STATE":
            self.log_dbug("\033[92mUAV \033[96m\[ " + str(vehicle.vehicle_id) + " ] \033[94mterminated swarm behaviour. \033[0m"")
            self.searchUAVMap[vehicle.vehicle_id].status = SEARCH_FAULT

if self.ownID == vehicle.vehicle_id:
    currentTime = time.time()
    timeDelta = currentTime - self.search_Operation_StartTime
    self.log_dbug("\033[94mSearch Operation \033[91mSUSPENDED \033[94mafter \033[96m{0:.3f}.format(timeDelta) + "\033[94m secs\033[0m ")

for j in range(self.rows):
    for i in range(self.cols):
        if self.searchGrid[i][j].visited == False:
            self.log_dbug("\033[93m[" + str(i) + "," + str(j) + "] \033[94mwas not visited.\033[0m")

        else:
            timeDelta = self.searchGrid[i][j].visitedTimeStamp - \

self.search_Run_StartTime
self.log_dbug("\033[93m" + str(i) + ", " + str(j) + 
  "] \033[94mVisit by \033[96m" + str(self.searchGrid[i][j].visitedBy) + 
  "] \033[94m after \033[96m" + "{0:.1f}".format(timeDelta) + 
  "] \033[94m secs\033[0m")

self.SEARCH_STATUS = SEARCH_READY
self.searchGrid = None
self.ExecuteMasterSearcherBehavior = False
self._deactivateSrvProxy(enums.SWARM_STANDBY)

self.searchUAVMap[vehicle.vehicle_id].receivedState = vehicle.swarm_state
if self.searchUAVMap[vehicle.vehicle_id].receivedState == SEARCH_SWARM_SEARCH_ACTIVE_STATE:
    # When swarm behaviour just turn active, the first local state is search ready
    self.searchUAVMap[vehicle.vehicle_id].status = SEARCH_READY
else:
    self.searchUAVMap[vehicle.vehicle_id] = searchUAVdata()
    self.searchUAVMap[vehicle.vehicle_id].pose = 
      [vehicle.state.pose.pose.position.lat, vehicle.state.pose.pose.position.lon]
    self.searchUAVMap[vehicle.vehicle_id].subSwarmID = vehicle.subswarm_id
    self.searchUAVMap[vehicle.vehicle_id].assignedAltitude = 
      SEARCH_ALTITUDE + (self.uavAltitudeCounter * SEARCH_SAFETY_ALTITUDE_INTERVAL)
self.uavAltitudeCounter += 1

def _process_swarm_search_waypoint(self, swarmSearchWP):
    # Check if own UAV supposed to recipient of this message
    for waypointMsg in swarmSearchWP.waypoints:
        if self.ownID == waypointMsg.recipientvehicle_id:
            if waypointMsg.searchCell_x >= 254 and waypointMsg.searchCell_y >= 254:
                self._deactivateSrvProxy(enums.SWARM_STANDBY)
                self.log_dbug("\033[94m" + "Swarm Searcher \033[96m" + str(self.ownID)+ \ 
                "]\033[94m proceeding to deactivate\033[0m")

            else:
                lla = apbrgmsg.LLA()
                lla.lat = waypointMsg.waypoint.lat
                lla.lon = waypointMsg.waypoint.lon
                lla.alt = self.searchUAVMap[self.ownID].assignedAltitude
                self.publishWaypoint(lla)
                self.log_dbug("\033[94m" + "Swarm Searcher \033[96m" + str(self.ownID) + \ 
                "]\033[94m proceeding to assigned cell \033[93m" + \ 
                str(waypointMsg.searchCell_x) + ", " + \ 
                str(waypointMsg.searchCell_y) + "]\033[0m")

        else:
            if waypointMsg.searchCell_x >= 254 and waypointMsg.searchCell_y >= 254:
                self.log_dbug("\033[94m" + "Swarm Search \033[96m" + str(self.ownID) + \ 
                "]\033[94m node: Received network wp cmd for Slave Searcher \033[96m" + \ 
                str(waypointMsg.recipientvehicle_id) + "]\033[94m to deactivate"
```python
def _process_swarmSearch_setup(self, swarmSearchSetup):
    try:
        if self._crnt_wp_id is None:
            raise ValueError('Autopilot status has not been received yet')
        self._wp_rel_alt = self._getWpSrvProxy(self._crnt_wp_id, self._crnt_wp_id).points[0].z
        self.search_master_searcher_id = swarmSearchSetup.order.masterSearcherID
        if self.ownID == self.search_master_searcher_id:
            self.log_dbg(\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \\
            \033[94m] \033[94m received command from swarm manager \033[96m[" + \\
            str(swarmSearchSetup) + " \033[0m")

        if self.ExecuteMasterSearcherBehavior == False:
            self.search_Operation_StartTime = time.time()
            self.bottomLeftLLA = [swarmSearchSetup.order.lat, swarmSearchSetup.order.lon]
            self.selectedSearchAlgo = 1 # set 1 (Greedy) as default
            tempVar = swarmSearchSetup.order.searchAreaLength / SEARCH_CELL_RADIUS
            self.cols = int(math.ceil(tempVar))
            tempVar = swarmSearchSetup.order.searchAreaWidth / SEARCH_CELL_RADIUS
            self.rows = int(math.ceil(tempVar))
            self.numOfRunsToDo = 1
            self.numOfRunsDone = 0
            self.searchSubSwarmID = self.searchUAVMap[ self.ownID ].subSwarmID
            self.ExecuteMasterSearcherBehavior = True
            self.SEARCH_STATUS = SEARCH_READY
            self.searchGrid = None
```
self.log_dbug("\033[94mSwarm Search Master \033[96m" + str(self.ownID) + \
"] \033[94m search operation with \033[96m" + str(self.
numOfRunsToDo) + \
"] \033[94m run(s) started on \033[96m" + \
str(time.asctime(time.localtime(time.time()))) + "\033[0m")

for vehicle in self.searchUAVMap:
    self.searchUAVMap[vehicle].assignedAltitude = self._wp_rel_alt
self.log_dbug("\033[94mSwarm Search Master \033[96m" + str(self.ownID) + \
"] \033[94massigned relative altitude for search \033[0m" + \
str(self.searchUAVMap[self.ownID].assignedAltitude))
self._activate_swarm_search(swarmSearchSetup.order.searchAlgoEnum)
self.log_dbug("\033[94mSwarm Search Master \033[96m" + str(self.ownID) + \
"] \033[94musing swarm search algorithm \033[0m" + str(self.
selectedSearchAlgo))
self.set_ready_state(True)
else:
    self.log_dbug("\033[94mSwarm Search Master \033[96m" + str(self.ownID) + \
"] \033[94m ignored command from swarm manager as search
operation is underway. Suspend operation first before
assigning any new operation\033[0m")
else:
    self.ExecuteMasterSearcherBehavior = False
    self.searchUAVMap[self.ownID].assignedAltitude = self._wp_rel_alt
    self.log_dbug("\033[94mSwarm Search node: Searcher \033[96m" + str(self.ownID) + \
"] \033[94massigned relative altitude for search \033[0m")
str(self.searchUAVMap[self.ownID].assignedAltitude))

    self.set_ready_state(True)
    return True

except Exception as ex:
    self.log_warn("Failed to initialize swarm search: " + str(ex))
    self.set_ready_state(False)
    return False

def sub_autopilot_status_update(self, msg):
    self._crnt_wp_id = msg.mis_cur

# Main code
if __name__ == '__main__':
    args = rospy.myargv(argv=sys.argv)
    searcher = SwarmSearcher("swarm_searcher", rospy.get_param("aircraft_id"), args)
    searcher.runAsNode(10, [], [], [])
List of References


Initial Distribution List

1. Defense Technical Information Center  
   Ft. Belvoir, Virginia

2. Dudley Knox Library  
   Naval Postgraduate School  
   Monterey, California