ABSTRACT


African wildlife parks are currently in need of both anti-poaching surveillance and the funding for it. In this thesis, I have successfully developed a tunable architecture for autonomous unmanned aerial systems that services this need. The architecture for the aerial system uses scouts and trackers (agents), data maps, a queue, and the manipulation of certain performance parameters to achieve the desired result. Ideal equilibrium numbers (design numbers) of scouts and trackers are chosen based on the area of the territory being surveilled, the number of objects (animal groups), the time to see the entire area, the longest elapsed time for the field (field being wherever there isn’t an object), the percent area covered of objects, and the longest elapsed time for the objects. This architecture is designed to feed videography. In the future, the footage of the objects and the field will be stitched together to form a video of those regions in which objects are found. As such, the algorithm will be well-suited for wildlife monitoring and search and rescue applications.
Navigation of Systems of Unmanned Aerial Vehicles for Monitoring Wild Animal Populations in the Arid Savanna

by
Alexander Cemil Purut

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the degree of Master of Science

Mechanical Engineering

Raleigh, North Carolina 2019

APPROVED BY:

_______________________________  _______________________________
Larry Silverberg                  Matthew Bryant
Committee Chair                  

_______________________________
Edgar Lobaton
BIOGRAPHY

Alexander Cemil Purut was born on June 30, 1995 in Durham, North Carolina. He was raised in Hickory, North Carolina and graduated from Discovery High School in 2013. After high school, Alex attended North Carolina State University in Raleigh, North Carolina. He graduated in 2017 with a Bachelor of Science in Mechanical Engineering. Shortly afterwards, returned to North Carolina State University to pursue his Master of Science in Mechanical Engineering.
ACKNOWLEDGMENTS

I would like to thank Dr. Silverberg for his support and guidance in the development of this algorithm and my masters courses. I would also like to thank my parents for always helping in whatever way they could.
# TABLE OF CONTENTS

LIST OF TABLES .................................................................................................................. v

LIST OF FIGURES ............................................................................................................... vi

Chapter 1: Introduction ........................................................................................................ 1

Chapter 2: Method .................................................................................................................. 4

Chapter 3: Results ................................................................................................................. 14

Chapter 4: Summary and Conclusions ................................................................................ 29
LIST OF TABLES

Table 1  Maps ........................................................................................................................................5
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Flow Chart and Queue</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2</td>
<td>AWM Performances over Time</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3</td>
<td>AWM Performances over the Design Number of Scouts</td>
<td>21</td>
</tr>
<tr>
<td>Figure 4</td>
<td>AWM Performances over the Design Number of Trackers</td>
<td>23</td>
</tr>
<tr>
<td>Figure 5</td>
<td>K Performances</td>
<td>26</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

Unmanned aerial systems and aerial videography are advancing rapidly [1]. Data is being provided with higher spatial and temporal resolution [2], more of it can be stored, and these services are safer and offered at a much lower cost than in the past [3]. It will not be long before advances in both unmanned aerial vehicle systems and aerial videography will make it economical to monitor large terrains of land and transport that imagery to computers everywhere in the world. The implications of aerial wildlife monitoring (AWM) are far-reaching. In Africa, images of highly valued wildlife found in large terrains of parkland and game reserves will be able to be studied by students and scientists worldwide in the comfort of their workplace environments. The revenues derived from specialized AWM services may provide African parks and game reserves with the resources they need to protect their uniquely valued species, potentially reversing the decades-long trend that otherwise ends with their extinction.

While the future impact of AWM to mitigate poaching is expected to be significant, to date, the documented research pertaining to the contribution of unmanned aerial systems to anti-poaching is in its early stages [3]. A military drone was loaned to Kruger National Park in South Africa for their anti-poaching efforts [4]. An effort in Namibia examined the efficacy of unmanned aerial vehicles in anti-poaching [5]. The ability to protect rhinos [6] and elephants [7] from poaching has also been studied. On the other hand, there has been a larger contribution by unmanned aerial vehicles (UAVs), although not necessarily systems of UAVs, to environmental and conservation research [8,2,9]. These UAVs have been applied to tropical and polar environments, and to observing hard to reach locations. As with anti-poaching, the capabilities of UAVs and systems of UAVs are unique when the interest lies in monitoring objects that are the size of a dog and larger and when the interest lies in near real-time monitoring and in real-time
monitoring. Besides the anti-poaching application, search and rescue using UAVs can be performed more effectively and quickly than by traditional aerial and ground methods. Likewise, animal counting can be more precise with UAVs than with multirotor aircraft, although UAVs may be slower [9, 10]. Medium sized animals are hard often to detect from altitudes above 60 m but detection from altitudes less than 60 m have been very successful [11, 12]. It’s important to note that smaller UAVs cannot operate during excessively windy conditions and need their batteries replaced frequently. (The small UAVs require recharging every half hour to an hour.)

The objective of my thesis research is to examine how an autonomous system of UAVs could regularly monitor a large terrain in the Namibian arid savanna. In particular, monitoring the wild animals living on the savanna, recognizing that AWM is becoming more and more viable [13]. We now know that employing a larger number of smaller UAVs is preferred over employing a smaller number of larger UAVs largely due to cost and maintenance considerations [2]. We also now appreciate that UAVs in AWM can fly sufficiently close to many of the African species while not disturbing them [2, 12]. In Namibia and in the arid savanna more generally, the density of the animals is relatively low. Most of the terrain is made up of open land, low lying tree cover and bush and many of the animal species tend to be found in groups. One assumes that AWM is not capable of covering the entire terrain continuously because of its large size. However, UAVs can continuously monitor animal groups within the terrain when a sufficient number of UAVs are employed and the density of the animal groups is sufficiently small. At the very least, the UAVs can rotate through different animal groups, analogous to police ground surveillance of streets in urban areas. Current and past images from an AWM system may then be stitched together to create a near-real-time video of the territory. These real-time or near-real-time images of animal groups
can be merged to create a video that is made available online (to be accomplished in a separate development effort that is beyond the scope of this research).
CHAPTER 2

Method

The objective of this research is to gain insight into an AWM algorithm architecture that governs the continuous navigation of a system of UAVs. The overall task of these vehicles is to monitor both the terrain of a given territory and the wildlife groups within the territory. Toward this end, I developed an algorithm that accomplishes this objective. The focus of this study focuses on the necessary components in the algorithm and the relationships between them, i.e. the architecture, as opposed to perfecting the specific sub-tasks of any one component. This should be kept in mind when reading this thesis.

Hereinafter, the following terminology will be employed. Each UAV is referred to as an agent, the terrain as the region, the territory as the area, and each animal group as an object. Recognizing that the overall task is two-fold – to monitor both regions and objects while periodically needing to recharge – each agent was assigned one of the following assignments:

Scout: scan the area
Tracker: monitor an object
Recharger: travel to recharge point

The algorithm would then make reassignments as appropriate.

Next, I broke up the area into a grid of regions to keep track of the status of all of the parts of the area. I represented the statuses as numbers that would be updated in real-time. There are multiple types of these numerical grids and each numerical grid is referred to as a map. The different maps are given in Table 1.
Table 1: Maps

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight map</td>
<td>dictates scout monitoring</td>
</tr>
<tr>
<td>initial unseen map</td>
<td>regions not yet covered</td>
</tr>
<tr>
<td>past object map</td>
<td>regions with objects</td>
</tr>
<tr>
<td>currently viewed region</td>
<td>regions currently in view</td>
</tr>
</tbody>
</table>

The weight map dictates the status priorities given to the different regions within the area. These priorities are given with respect to the need for the scouts to monitor those regions. The values become larger as the time that has elapsed since last agent viewed it increases. It also becomes larger if the region was last viewed to have an object within it. Thus, regions with objects grow in value faster than those without. The values of the weight map are set to zero over the regions currently being viewed. The scouts receive tasks to travel toward the regions that have the highest weight map values.

The initial unseen map is used during the initial stages of the operation when the system of agents is scanning the area to get a complete image of it and to learn in which regions the objects are located. It simply identifies which regions have not yet been viewed at least once. One might envision the AWM system being deployed daily with coverage of the entire area completed within several hours leaving the rest of the day to have visual sight over the entire area.

The past object map describes the regions in which objects were last found to be located. In the arid savanna, the animal groups move slowly over the terrain and the purpose of the development here was to determine a general architecture for the AWM algorithm. Therefore, for the present purposes, the objects were assumed to be stationary. This relaxed the need to update
the object map over time but otherwise does not change the overall architecture. It is understood
that the inclusion of the past object map in the current architecture makes it possible to consider,
at a future time in the development effort, the effect on AWM algorithm performance of the
movements of the objects over the regions. The currently viewed region map simply describes the
regions currently in view.

The flow of the algorithm and a queue that prioritizes assignments in the AWM are given
in Fig. 1.

Figure 1. Flow Chart and Queue

As shown, the AWM algorithm cycles around (1) reading every map, (2) assigning tasks
to agents, and (3) executing tasks, that is, traveling to designated locations. As the agents travel
toward their assigned locations, a queue associated with each agent governs the execution of tasks.
A synchronous reassignment time was selected, according to which certain tasks are reassigned
after a constant amount of time has passed since the previous assignment. Asynchronous
assignments are also executed.
All of the tasks represent locations in x and y in the area. General tasks are assigned to agents designated as scouts and trackers. When a scout completes a general task (reaches its target location) it receives a new general task. When a tracker completes a general task, it does not change tasks. Thus, the tracker stays in one place. A new object task occurs when an agent spots an object that was previously uncatalogued at that location, spawning an investigation of the object. When that agent is a tracker, it will not act on that new object task but it will add the task to another scout or recharger. The new object task enters the queue above the general task but below the recharger task and the previously assigned new object tasks. Thus, a recharger will recharge first and then check out the object. The recharge task is to travel to the location of a recharge station. It is assigned when the agent’s battery is sufficiently low. This task, when assigned, always enters the queue of an agent at the highest priority, so it is initiated by the agent immediately after it is assigned. When the recharger completes its recharge task, the agent turns into a scout.

The architecture described above is very broad and can be implemented using a variety of different approaches. The first and simplest pre-associates a fixed number of agents to serve as scouts and the others to serve as trackers. If the agents did not require recharging, this approach could be reasonable. However, every agent needs to recharge periodically, causing a significant inefficiency. When two agents are returning from recharging, one of them being a scout and the other a tracker, the scout could happen to be closer to the object that was previously covered by the tracker. In this scenario, it would be more efficient for that scout to turn into a tracker and the tracker into a scout. Furthermore, when scouts are first covering the area, utilization of trackers as scouts is advantageous. More generally, every time a scout spots an object, not allowing the scout to turn into a tracker necessitates a tracker to be sent to the object.
A second and more efficient approach is to develop simple heuristics that govern trackers turning into scouts and vice versa. A third and even more efficient approach employs more sophisticated algorithms that have the ability to weigh competing tasks and assignments. This approach divides the problem into time segments that begin and end when one of the scouts reaches a goal, when an agent becomes a recharger, or when a recharger finishes recharging. An algorithm, such as the Hungarian method [14], determines which agent is given which assignment and which task over a given time segment. The method would look to find the optimum set of assignments and tasks, that is, which agent is given which assignment and corresponding task. This takes into account the paths of the agents, the calculated time until the next update, and where the agents will be located for the next update. In this way, a horizon is developed. As with the Hungarian method, the longer the horizon, the more optimized the selected paths. While a larger horizon is better, this comes at the expense of additional computational effort. In this paper, for simplicity, the results section considers only the second approach.

The implementation of any AWM algorithm starts with considering the capabilities of the agents and their number. In this research, I considered agents with equal capability, each traveling at the same constant speed and capable of traveling for the same, fixed amount of time before needing to recharge. The capabilities selected in the results section were typical of relatively inexpensive UAVs that are readily available. Once the capabilities are selected, the next question that arises addresses the number of agents needed. Given the two-fold nature of the objective, the answer to this question naturally divides into the required number of scouts for the monitoring of the area and the required number of trackers for the monitoring of the objects.

The design number of scouts dictates the amount of time that is required to cover the entire area. This time in terms of the videography, can be broadly viewed as dictating a refresh time.
Every region of the area that is not currently being viewed shows an image of previous footage that has been stitched together. In the stitched portion of the area, the time required to cover the entire area dictates the largest time delay of any one of the photos. Due to their local movements, footage of objects should be taken in real-time. The static nature of the terrain allows photos of the terrain to be stitched into the image of the area with a reasonably significant time delay (on the order of hours). The time delay should not be too long owing to changes in light conditions, but some delay is acceptable, none the less. By employing a larger design number of scouts, the time delay can be decreased. Thus, there is an inherent trade-off between the design number of scouts and the time delay present in any stitched photo found in the videography. Additionally, an improvement to the stitched still photo around the agents would be a real-time video stitched together of previously gathered loops of footage. Thus, while regions with agents would show real-time video, other regions would show older video.

With respect to the design number of trackers, it is ideally equal to the number of objects. It was recognized that when the system is first launched the locations of the objects are unknown. Thus, it is advantageous for all of the agents to be initially assigned as scouts and, as objects are found, convert scouts into trackers. This minimizes the time to obtain a complete image of the area and the time to locate all of the objects. Note that the amount of time to obtain a complete image is approximately equal to the maximum time delay of the photos in a stitched image.

During the initial scanning of the area, at some point, no more trackers become available for the scanning. This leaves a minimal number of scouts to continue scanning the area. The minimum number of scouts is an important parameter that influences the performance of the algorithm. Thus, the design number of scouts is the minimal design number of agents to be designated as scouts. Similarly for trackers, the design number of trackers is the maximum design
number of trackers allowed. (The maximum design number of trackers is equal to the design number of agents less the design number of scouts.) The design number of trackers is ideally equal to the number of trackers required to cover all of the objects. However, there are cases in which the number of objects changes as animal groups split up or merge together. Therefore, some latitude is needed in the algorithm for the design number of trackers, being that it is fixed, to accommodate these changes.

Overall, the description of the AWM algorithm’s performance can be divided into its transient performance, during which the system is first scanning the entire area, and its steady-state performance during which it has reached an equilibrium of sorts.

The architecture described above can handle a variety of real-world conditions that will, in a final analysis, need to be considered even though they were not investigated in detail in the results section. The major ones are: (1) non-uniform object size, (2) priority dependence on object type, and (3) object movement. With respect to object size, objects can be smaller, the same, or larger than the size of the region that the agent can view. This affects the agent navigation that takes place when a new object is found. If the object size is significantly smaller than the size of the region in view, then the scout will quickly identify the entirety of the group. If the object is about the same size as the region in view by the agent, then a small circle may be required to see the entire object and identify its shape and size. Finally, if the object is sufficiently larger than the view region, then significant time may be needed to trace out the perimeter of the object and to fill in the inside. If there are enough trackers available, then the object could be split up and multiple trackers could watch different parts of the object and each may move around to increase effective coverage of the object. With regard to different types of objects, different objects could be assigned different values. If, for example, the trackers are tracking many groups of zebras and then a group of rhinos
are found, then the trackers can switch priorities and balance out the types of animals being observed. Moving objects are not too great of an issue as it would only mean that trackers would need to move with the objects, but there are times in which the animals could move very rapidly, causing difficulties with keeping up. In the case where there are more objects than the maximum number of trackers, scouts would be locating “new objects” that aren’t really new; they are just old objects that have moved. In this case, scouts can redefine the new positions of the objects and if an agent finds an old object to be empty or containing a new object, a scout can be sent to further investigate the object.

Having laid out the general architecture of an AWM algorithm, this section closes with a method of benchmarking its performance. We consider the time \( T_0 \) required to cover the entire area and the time \( T_1 \) required to refresh the entire area. Next, consider the following parameters that will influence both of them: the area \( A \) of the territory, the velocity \( v \) of an agent, the width \( w \) of the view region of an agent, the minimum number \( n_s \) of scouts, and the maximum number \( n_T \) of trackers (The number of agents is \( N = n_T + n_s \)). Putting aside the inefficiencies of a specific algorithm, one expects the time \( T (T_0 \text{ or } T_1) \) to increase in proportion to \( A \), in inverse proportion to \( v \), in inverse proportion to \( w \), and in inverse proportion to the number of scouts.

First consider the ideal refresh time \( T^i_1 \). This is the time, after the trackers are occupied over their respective objects, for the coverage over the area to be refreshed. Assuming that the scouts do not overlap one another when covering the area and that the area covered by the trackers relative to the entire area is negligible, we get

\[
T^i_1 = \frac{A}{vwn_s}.
\]
Next, consider the ideal coverage time $T_{0I}$. Recall, this is the period of time that begins when the agents are deployed and ends when they have covered the entire area, during which period the number of scouts scanning the area changes. Imagine that the area is divided into an integer $\phi$ of equal size regions ($\phi > 0$) and that we employ the number of trackers needed to cover the area over all of the objects. Assuming a random distribution of objects, the number of objects in each region is taken to be the same from which the number of scouts that turn into trackers when covering a region becomes the same. When starting out, there are $n_s + n_T$ scouts. After completing the scouting over the first region, at the start of the second region the number of scouts has been reduced by $n_T/\phi$ to yield $n_s + (1 - 1/\phi)n_T$ scouts. Upon the start of the third region, the number of scouts was reduced by another $n_T/\phi$ to yield $n_s + (1 - 2/\phi)n_T$ scouts, until upon the start of the last region there are $n_s + (1 - (\phi - 1)/\phi)n_T$ scouts. More generally, at the start of the $j^{th}$ region, the number of scouts is $n_j = n_s + (1 - j/\phi)n_T$ ($j = 0, 1, 2, \ldots, \phi - 1$). The ideal coverage time is then

$$T_{0I} = \sum_{j=0}^{\phi-1} \frac{A/\phi}{vwn_j}$$

assuming that the area is divided into $\phi$ regions. In the limit, the ideal coverage time is

$$T_{0I} = \lim_{\phi \to \infty} \left( \frac{A}{vn} \lim_{\phi \to \infty} \left( \frac{1}{\phi} \sum_{j=0}^{\phi-1} \frac{1}{n_s + (1 - j/\phi)n_T} \right) \right)$$

In a specific AWM algorithm, the coverage time $T_0$ and the refresh time $T_1$ can now be respectively compared to the ideal coverage time $T_{0I}$ and ideal refresh time $T_{1I}$ as follows:

$$T_0 = K_0 \frac{A}{vwn_{se}} \frac{1}{n_{se}} = \lim_{\phi \to \infty} \left( \frac{1}{\phi} \sum_{j=0}^{\phi-1} \frac{1}{n_s + (1 - j/\phi)n_T} \right)$$

$$T_1 = K_1 \frac{A}{vwn_s}$$

1
where $K$ is a non-dimensional performance parameter that depends on the number of scouts and trackers. Notice above that $n_{Se}$ is an effective number of scouts used to cover the entire area. Ultimately, the goal is to come as close, on average, to $K_0 = 1$ and $K_1 = 1$.

The development of the formulas in terms of the coverage time and the refresh time in terms of the entire area $A$, the agent speed $v$, the width $w$, and the number of scouts $n_S$ and $n_{Se}$, are approximations, relying on simplifying assumptions. First, recall that we assumed that the agent vision doesn’t overlap. This is particularly unrealistic over time. Second, objects are not randomly spaced in reality, because of a variety of factors, among them environmental and behavioral. In terms of coverage speed, the best case is when the objects are at the periphery of the area. That way, all agents work as scouts before finding all of the objects at the last minute. By contrast, the worst case would be all of the objects are in the center of the map and are immediately found by the agents. As a consequence, the minimum number of scout agents would scout the entire area. The ideal case is an average of these options and assumes that the objects are equally spaced. This is also unrealistic as this average may not be a realistic average. Third, the coverage of the trackers over the objects isn’t taken into account as it is considered negligible. Finally, the times rely on the fact that no recharging takes place.

It should be noted that when calculating $K$, the number of scouts and trackers between the idealized and simulated case are the same. This obviously makes sense when all the objects are covered by trackers. But, this is also true when there aren’t enough trackers. In this case, the ideal involves the instant teleportation of trackers between their target objects. As such, $n_S$ and $n_T$ remain matching. In the case where there is an excess of trackers, the extra trackers strictly behave as scouts in the ideal case. Thus, when calculating the $K$ value, the extra trackers must be added to the number of scouts and removed from the number of trackers.
CHAPTER 3

Results

Below, an AWM algorithm that follows the architecture developed in Chapter 2 was constructed and its performance assessed. I examined square territories (areas) that contain square 30 m by 30 m groups of animals (objects). The UAVs (agents) viewed square 50 m by 50 m areas, flew at 20 m/s, and were designated as rechargers after 30 min of continuous flight (assuming 45 min total flight duration). The recharge station was located in the center of the area, the recharge time (battery swap) was neglected, and the agents were initially launched from the recharge station with no delay between launches. An agent was designated as reaching a target when it was within 1 m of the target point. The objects were randomly distributed in the area, non-overlapping, and stationary.

In order to only run practical cases, the simulations were run for at most a twenty four hour period and only simulations in which the viewing of the entire area was completed within 16 hours (creating a minimum of 8 hours of near-real-time viewing) and the simulations were cut off 8 hours after viewing the entire area, which was a sufficiently long time to assess steady-state performance.

In the weight map, there are regions that contain objects and regions that don’t. Regions that don’t contain objects only grow in value because of time elapsed. If the region contains an object, it will grow in time at twice the rate as a region without an object. The formula is shown below:

\[ W_i = W_0 + (1 + \beta)W_i \]

where \( W_i \) is the new weight for a given point, \( W_0 \) is the previous weight for the point, \( W_i \) is the weight increment over a step in time, \( \beta \) is 0 if there isn’t an object at the location and \( \beta \) is 1 if there
as an object at the location. Thus, areas with objects grew two times faster than areas without. For any area that hadn’t been seen yet, it was assumed that the area does not contain an object.

For each set of parameters considered, 24 simulations were run. In each one, the objects were randomly distributed over the area such that each object doesn’t touch or overlap another. All of the results in this section are averages over the 24 simulations.

At the start of each simulation, the weight map of the entire area was set to be a sufficiently large value to ensure that the unseen areas are given the highest priority. The value of any unseen point is set to zero when that point is seen. Thus, the large initial values completely disappear once all of the points in the entire area have been seen.

We begin below with examining time responses (See Fig. 2) of an AWM algorithm employing 6 to 16 scouts and 2 to 8 trackers applied to an area containing 8 objects. The performance of the AWM as a function of time is described in terms of (1) percentage coverage of the objects, (2) percentage coverage of the area, (3) number of trackers, (4) the number of scouts, (5) longest elapsed time of the objects, and (6) longest elapsed time of the field. (The field is any region of the area that does not contain an object.) In each of the plots, the data set uses 8 objects and a 30 km² area and AWM performance is averaged over data constituting 24 runs for each parameter set. In the plots on the left, there are only 6 scouts and each line varies by the design number of trackers. In the plots on the right, there are only 8 trackers and each line varies by the design number of scouts. Also, notice that the behavior divides into transient and steady-state regions and one sees numbers of scouts and trackers spiking downward periodically as a result of the recharging.
Figure 2. AWM Performances over Time (8 Objects, 30 km² Area)
In the top-left plot, we see the percent area covered versus time. Each line represents the number of trackers. You can also see that as the scouts find more objects, the percent area covered increases. Notice that at 8 trackers the coverage percentage goes to 100%. This is because the number of trackers and the number of objects is the same. Similarly, 6 trackers cover 75% of 8 objects, thus the percent coverage is at 75%. This continues for all of the numbers of trackers.

There are small spikes upwards on all tracker lines that are not at 100% coverage. This is because of scouts passing over objects that aren’t being covered at that time. When there are 8 trackers, there are spikes about every 30 minutes. These are due to the recharge time which triggers the agent to recharge after 30 minutes from the previous recharge. These spikes appear for fewer than 8 trackers as well. For trackers less than 8, there are also spikes exactly 15 minutes apart. This is due to the cycle time where after every 15 minutes the objects selected to be covered are changed. After this change the trackers have to move to the uncovered objects. The spikes from the recharging shift over time relative to the cycle spikes. This is due to the fact that the recharge time is dependent on how long it has been since the last recharge. When the time is below a threshold, the agents move to the center from varying distances away from the center. Due to this random behavior, the timing of the recharge spikes spreads out and reduces in size over time. Eventually, the spike will be so spread out, that at all times there is at least one recharger. This is why you see the percent area covered equalize a little less than what the theoretical coverage predicts.

In the top-right plot, we see the percentage of the area initially seen versus time. Unlike the other plots in this figure, the time only goes from 0 hours to about 6 hours. It can be seen that more scouts lead to a faster time to cover the entire area. Each line represents a certain number of scouts. As the number of scouts increases, the percentage of the area seen over time increases, too. However, the last few percent always take longer to cover. This is because the implementation of
the algorithm was segmented. Accordingly, the width of the area is divided into equally wide strips; the number of them set equal to the number of scouts. The scouts are only given tasks within their assigned strip. Initially, all of the agents are scouting the area, thus the area seen increases rapidly. As time progresses, agents turn from scouts to trackers and the chances of overlapping a previously scouted area increases. Eventually, only one scout is left with an unseen area in its strip. Due to the nature of this implementation of the algorithm, that agent is solely responsible for finding the last unseen region in its strip. Thus, the final few percent take longer to find. The curve formed is roughly correlated with the curve formed from the 8 tracker case in the top-left plot before equilibrium. This is expected since the larger the discovered area, the more objects will be found.

The middle-left plot shows the real-time number of trackers. It looks like the top-left plot. This is because the percentage area covered is highly correlated with the number of trackers. Thus, 8 trackers and 8 objects yield about 100% coverage, 6 trackers and 8 objects yield about 75% coverage, etc. It can be seen that the curve formed during the transient state roughly correlates to the curve in the top-right plot for similar reasons as with the top-left plot. Similar to the top-left plot, you can see the spikes due to the recharge times spreading out. It can be seen more clearly that initially there are 0 trackers. Then, after 15 minutes, the first cycle time completes and trackers are assigned. After 30 minutes, recharging for all agents occurs all at once and they all begin to return home. As previously mentioned, the agents return from different distances from the recharge station. Thus, the start times of the recharging are different for each agent.

In the middle-right plot the real-time number of scouts is shown. Initially, all of the agents start out as scouts, then every 15 minutes, scouts become trackers until the design number of scouts and trackers are reached. Thus, this plot is the inverse of the middle-left plot when the spikes from
recharging aren’t taken into account. Similar to the middle-left plot the recharge spikes occur about every 30 minutes. However, the number of scouts stabilizes more rapidly than the trackers in the middle-left plot because this implementation of the algorithm prioritizes the number of scouts over the number of trackers. Thus, the reduction in agents from recharging affects trackers more than scouts. This prioritization was made based on the assumption that scouts have to cover much more area than trackers. Calling scouts back repeatedly to take the place of trackers would be more inefficient overall than the opposite.

The bottom-left plot shows the longest elapsed time of objects over time. Notice that unlike the previous four plots, the line for the 2 trackers is on top and the line for the 8 trackers is on the bottom. For all of the tracker values, the slope is always 1. The line has a slight curve in it since it is made up of the average of multiple simulations. Thus, the initial hump formed is just a byproduct of averaging, in any particular simulation the plot grows at a rate of 1 and occasionally drops to the next highest value. In the case of 8 trackers, 100% of the objects are covered. So, after all of the object areas are discovered, the longest elapsed time only grows when the trackers recharge. For any case with less than 8 trackers, there are always objects not being watched and the trackers still need to recharge occasionally. Thus, the longest elapsed time never reaches 0 in these cases. Increasing the design number of trackers will benefit the longest elapsed time. However, there is a case of diminishing returns that is due to the cycle time (Cycle time is discussed after Fig. 4).

In the bottom-right plot, the longest elapsed time for the field is shown. The design number of trackers has almost no effect on the longest elapsed time for the field and thus is not considered. Similar to the bottom-left plot, this plot has the line for 6 scouts on top and the line for 16 scouts on the bottom. Also, similarly, the plot only grows by 1 and any other slopes are a consequence of averaging multiple simulations. Increasing the number of scouts will reduce the longest elapsed
time after stabilization. However, there are diminishing returns. This is due to the fact that each scout divides the area into that number of strips. Thus, each additional scout provides a reduced benefit. The curve predominantly seen with 6 scouts is a consequence of scouts moving to see the entire area first and then having to move back to review the areas that they covered initially. The more scouts there are, the more suppressed this pattern becomes as the scouts are closer to their initially covered area.

Figure 3 covers the AWM performances over the design number of scouts. The plots on the left use data with only 20 objects and 20 trackers. The plots on the right use instead the constraints of a 30 km² area and for each line that the number of trackers is equal to the number of objects. The first row of plots pertains to the transient time before equilibrium and the second row deals only with the time after equilibrium is reached.
Figure 3. AWM Performances over the Design Number of Scouts

In the top-left plot, the time to see the entire area is shown with respect to the design number of scouts. As the area increases, the time to see the area increases as well. As the design number of scouts increases, the time decreases. However, each additional scout brings a diminishing reduction in time.

The bottom-left plot shows the longest elapsed time for the field relative to the design number of scouts. This plot is similar to the top-left plot but they aren’t the same. While the top-
left plot deals with the transient case (before equilibrium), this plot deals with the time after equilibrium has been reached. The effects of area and the design number of scouts are similar to the top-left plot. As the area increases, the longest elapsed time increases. When the design number of scouts increases, the time decreases. Again, with each additional scout, the time decreases by a smaller amount. This trend is similar to the one found in the bottom-right plot in Figure 2. Each additional scout divides up the area by one more unit, reducing the area per scout by smaller and smaller amounts.

You can see in the top-right plot the time to see the entire area compared to the design number of scouts. This plot has the same vertical axis as the top-left plot. As expected, while the number of scouts can reduce the time, the number of objects (and consequently the number of trackers) doesn’t have much of an impact. The small reduction that is present is from the fact that the additional trackers initially act as scouts. However, the additional scouts are soon converted into trackers.

The bottom-right plot shows the longest elapsed time of the field relative to the design number of scouts. As with the top-right plot the effect of the object and trackers is almost nonexistent. As expected, and also shown with the bottom-left plot, the increasing design number of scouts reduces the time at a decreasing rate.

Similar to Figure 3, Figure 4 shows the AWM performances with respect to the design number of trackers. All of the plots are an average after equilibrium and use data that only has 6 scouts. The left plots use data with only 20 objects and the right plots use data with only 30 km² area.
The top-left plot shows the percent area covered of objects depending on the design number of trackers. As the number of trackers increases, the coverage percentage also increases. However, at 20 trackers, all objects are covered but the average isn’t 100%. This is because the trackers have to recharge every 30 minutes. As the area increases, the percentage covered decreases. This is because increasing the area exacerbates the drop from recharging. The larger area means that on average the agent has to travel further to the recharging station.
The top-right plot again shows the percent area covered of objects depending on the design number of trackers. However, here, there are different constraints and the number of objects is varied. For each line, the number of objects is fixed and the number of trackers goes up to the number of objects. As expected, as the number of trackers increases, so does the percent covered. Notice at the end point for each line that the design number of trackers is equal to the number of objects. Similar to the previous plot, the end point isn’t at 100% due to recharging every 30 minutes. This is further exacerbated by the large area used for this plot. Also notice that all of the lines are close to 0 when the design number of trackers is 0. Thus, the slopes of the lines decrease as the number of objects increases. This is because the slope is primarily determined by the ratio of the design number of trackers and the number of objects.

The bottom-left plot shows the longest time elapsed for objects with respect to the design number of trackers. It can be seen that in general more trackers lower the time. However, this isn’t the case from 10 to 19 trackers. In addition, before 10, the curves are different. This is all due to the cycle time. As previously mentioned, every 15 minutes, the previously uncovered objects are selected to be covered. When there are 20 trackers, 100% of the objects are selected. None of the objects have time to build up time and thus the average longest elapsed time is about 0. For 10 to 19 trackers there is always at least one object uncovered. However, after each cycle the uncovered object is always a different object such that all objects spend about the same amount of time uncovered. Thus, there is always at least one object that has to wait a cycle of 15 minutes to be seen. From 7 to 9 trackers, there is always at least one object that has to wait 2 cycles. This continues, for 5 and 6 trackers, 3 cycles, for 4 trackers, 4 cycles, for 3 trackers, 6 cycles, for 2 trackers, 9 cycles, and for 1 tracker, 19 cycles. This can be applied to any number of trackers and objects based on the formula: \[ C = \left\lfloor \frac{n_o}{n_T} \right\rfloor - 1 \] where \( C \) is the number of cycles, \( n_o \) is the number...
of objects, and \( n_T \) is the design number of trackers. This would always be the case if it weren’t for the scouts. When the area is 30 km\(^2\), then this plot follows the formula above. However, when the area is 10 km\(^2\), the point at 2 trackers is about .75 hours. This is because the scouts are helping to update the objects. As can be seen in the bottom-left plot in Figure 3, when the area is 10 km\(^2\) at 6 scouts, the longest time is also about .75 hours. When we compare the 30 km\(^2\) situation in a similar manner at 6 scouts, the longest time is about 6 hours which is far above the roughly 2.25 hours in this plot at 2 trackers. Thus, the final result is that a minimum is taken between \( C = \lceil n_o / n_T \rceil - 1 \) and the longest elapsed time for objects for a fixed area, objects, and scouts. Further effects are shown with the areas that aren’t 30 km\(^2\). There is one other effect on the lines, however, that is much smaller. As can be seen, the smaller areas always have lower times. This is because, the larger the area, the farther agents have to move between objects when a cycle completes.

In the bottom-right plot you again see the longest elapsed time for objects. However, in this plot, the area is fixed to 30 km\(^2\) and 6 scouts. We know from the previous plot explanation that with these parameters the longest elapsed time for objects is about 6 hours. Thus, we can see the effect of strictly the \( C = \lceil n_o / n_T \rceil - 1 \) formula. As such, more trackers generally reduce the longest elapsed time.

In figure 5, you can see the performance factor \( K \) with respect to each independent variable. The blue stars show the actual simulation times. The orange line shows the ideal time curve multiplied by \( K \) (as shown in equation 1). The determined \( K \) for each plot is given in the title. There isn’t a plot of the refresh time versus the design number of trackers. This is because they are unrelated. Overall, the \( K \) value for seeing the entire area is in the neighborhood of 6 and the \( K \) value for the refresh time is in the neighborhood of 4.5.
Figure 5. K Performances
The top left plot shows the effect of the design number of trackers on $K$. It can be seen that as the number of trackers increases, most of the blue stars move from below the orange line to above the orange line. This means that the algorithm is more efficient with a lower design number of trackers and number of objects. This is expected because more trackers and objects exacerbate the inefficiencies of the algorithm. When the design number of trackers gets larger, the time to see the entire area gets smaller. However, each additional tracker only causes a small decrease. This effect would be further minimized if the number of objects didn’t grow with the number of trackers.

Similar to the top left plot, the middle left plot shows the time to see the entire area depending on the number of scouts. Unlike the previous plot however, the $K$ value becomes lower as the number of scouts increases. This is because scouts dilute the inefficiency of the algorithm. As the design number of scouts gets larger, the time to see the entire area gets smaller. However, each additional scout reduces the time by a smaller amount.

The time to see the entire area versus the size of the area is shown in the bottom-left plot. Similar to the top left plot, the $K$ value increases as the area gets longer. Larger area means a longer time to recharge and move between objects. Thus, it is more inefficient. As the area grows, so does the time to see the entire area.

The middle right plot shows the refresh time versus the design number of scouts. Similar to the middle left plot, the $K$ value goes down as the design number of scouts increases. Again, this is due to the minimization of inefficiency. As expected, the refresh time goes down with higher design numbers of scouts.

Finally, the bottom right plot shows the refresh time versus the size of the area. Similar to the bottom left plot, the $K$ value goes up as the area grows. This is for the same reason: larger area
means more time spent moving between objects and to recharge. Also, when the area gets larger, the refresh time also gets larger.
CHAPTER 4

Summary and Conclusions

In this thesis I developed an architecture for AWM and verified that it works. The architecture involves using scouts and trackers, data maps, a queue, and manipulating certain performance parameters to achieve a desired result. With this AWM architecture, agents are designated as scouts, trackers, or as rechargers. Scouts are tasked with traveling around the area to update the images of those regions of the field that have not been seen for some time. Trackers are tasked with watching objects found in the area. If there are not enough trackers to watch over every object, then scouts preferentially and periodically check up on objects. When the AWM system begins, all of the agents are designated as scouts. The scouts use the combination of maps to determine where to go. These destinations are added to each agent’s queue as a general task. If scouts find objects, they will investigate it by adding their locations as new object tasks in the queues, placing them above the general goals in the queues. Occasionally, agents will need to recharge. When this happens, the agent turns into a recharger. The agent places the destination of the recharging station at the top of its queue as a recharging task. After reaching the recharging station and recharging, the agent is reassigned to be a scout. Every so often objects will be picked to be watched over. Some of the scouts will be converted into trackers and be given the destination of the object as their general task.

Furthermore, a K value was developed as a way to determine the efficiency of any implemented architecture. By better tuning the implementation, a lower K value is achieved.

For the purposes of designing an AWM system, ideal equilibrium numbers (design numbers) of scouts and trackers are chosen at the start. This is the number of trackers and scouts that will be present after the entire area is seen. These design numbers are determined based on the area of the land, the number of objects, the time to see the entire area, the longest time of the field
(the field is every portion of the area that isn’t an object), the percent area covered of objects, and average longest time elapsed for objects. In general, a higher design number of scouts implies faster time to see the entire area, and lower longest time of the field. On the other hand, a higher design number of trackers generally means a higher average percent area covered of objects and a lower average time elapsed for objects. Ideally, the design number of trackers is equal to the number of objects.

This AWM architecture is designed to feed videography. In the future, the footage may be stitched together to form a single video with only certain parts of the footage moving and the other parts still. Wherever the agent is, the video would update, otherwise, there is a still image or a short loop of older footage that has been stitched together. Objects are identified with computer vision. As such, the algorithm is autonomous. This AWM architecture is ideal for applications such as wildlife monitoring and search and rescue operations.
REFERENCES


5. vanVuuren, M., vanVuuren, R., Lutz, G., and Silverberg, L. M. "On the Effectiveness of UAS for Anti-Poaching in the Africa Arid Savana," PLOS ONE in review,


