2013

Actual Entities: A Control Method for Unmanned Aerial Vehicles

Erica Absetz
Virginia Commonwealth University

Actual Entities:
A Control Method for Unmanned Aerial Vehicles

A thesis submitted in partial fulfillment of the requirements for the degree Master of Science at Virginia Commonwealth University

by

Erica Lynn Absetz

Bachelor of Science, Virginia Commonwealth University, 2012

Associate of Science, J. Sergeant Reynolds Community College, 2009

Director: Dr. James Ames

Associate Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, VA

April, 2013

Acknowledgements

There have been many people that have supported me at different times in my education that I would like to thank. Thank you to both of my parents for their undying love and support, it motivated me to do the best I could throughout my educational career. Thank you to my best friend, Johnny Xmas, for threatening to never speak to me again if I didn't finish my degree. That helped me power through the rough patches. I would also like to thank Dr. Ames as he took on the last minute task of advising my thesis for this last semester. Above all, I would like to acknowledge the late Dr. Primeaux and the impact that he has had on my educational path. His passion and energy made me fall in love with Artificial Intelligence. He encouraged my pursuit of a Master's Degree and was a valued mentor for the beginning of this thesis as he guided the formation of the concept.

Table of Contents

List of Figures:

Abstract

ACTUAL ENTITIES: A CONTROL METHOD FOR UNMANNED AERIAL VEHICLES

By Erica Lynn Absetz, M.S.

A thesis submitted in partial fulfillment of the requirements for the degree Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2013

Director: Dr. James Ames, Associate Professor, Department of Computer Science

The focus of this thesis is on Actual Entities, a concept created by the philosopher Alfred North Whitehead, and how the concept can be applied to Unmanned Aerial Vehicles as a behavioral control method. Actual Entities are vector based, atomic units that use a method called prehension to observe their environment and react with various actions. When combining multiple Actual Entities a Colony of Prehending Entities is created; when observing their prehensions an intelligent behavior emerges. By applying the characteristics of Actual Entities to Unmanned Aerial Vehicles, specifically in a situation where they are searching for targets, this emergent, intelligent behavior can be seen as they search a designated area and locate specified targets. They will alter their movements based on the prehensions of the environment, surrounding Unmanned Aerial Vehicles, and targets.

Chapter 1: Introduction

A philosopher, Alfred North Whitehead, created the concept of Actual Entities. An actual entity is an atomic unit that changed based on its prehensions of the surrounding environment. Prehension, a term that Whitehead created, is defined as an observation followed by reactions. When multiple Actual Entities are combined into the same environment a colony of prehending entities is created, which exhibits an emerging, intelligent behavior.

The use of Actual Entities had been introduced into the field of computer science by both Primeaux and Saunders [8, 9, 11]. They have used the concept as a modeling tool, a way to find global and local maxima in a 3D space, and as a trust-based learning method for data characteristics. Both acknowledge Actual Entities to be a form of Artificial Intelligence belonging to the classification of Swarm Intelligence (SI). SI is commonly used for most control methods for Unmanned Aerial Vehicles. Now, take a look at Unmanned Aerial Vehicles and how they are sent out in groups to complete a mission. There have been many different types of behavioral control methods created to find the most efficient and dynamic control method. It is important that the control method can work with changing numbers of Unmanned Aerial Vehicles and to not rely heavily on communications.

This thesis will be used to show that Actual Entities can be used as a behavioral control method for Unmanned Aerial Vehicles. It will focus primarily on the application of Actual Entities to Unmanned Aerial Vehicles to complete a searching objective of locating designated targets. Furthermore, the concept of Actual Entities and that of other control methods currently in use will be covered. Finally, the control method will be tested in a simple simulation and the results will be analyzed. Coding samples used for the testing of this concept will be provided in the appendix.

Chapter 2: Actual Entities

Actual entities (AE) are the atomic units of the theory of a philosopher by the name of Alfred North Whitehead. As Hooper [4] states, "actual entities compose the Universe of existence in space and time, and it is therefore important first of all to understand what these entities are." AEs are vector based, a behavior involves an AE and an object then defines a vector relationship between the two. It is important to keep in mind that an AE's current state is a result of its history and the two cannot be separated. An actual entity is essentially a "process", containing its growth between phases, ending in a final and definite achievement. Hooper [4] stated that AEs could be referred to as genetic in that they have a process to "become" and the outcome of that causes "satisfaction". Every AE has its own unique path in its development and it uses its own perspective on the world to mirror it. As discussed by Hooper, actual entities have two sides. One is the emergent creature formed from its achievements and the other is a mode of creation that uses its own acts to gain experience while pulling the Universe together. A passage from one of Whitehead's books [12] helps tie it all together as follows:

An event has contemporaries – This means that an event mirrors within itself the modes of its contemporaries as a display of immediate achievement. An event has a past. This means that an event mirrors within itself the modes of its predecessors as memories which are fused into its own present. An event has a future. This means that an event mirrors within itself such aspects as the future throws back into the present, or in other words, as the present has determined concerning the future. Thus an event has anticipation:

"The prophetic soul

Of the wide world dreaming of things to come."

Chapter 3: Eternal Objects

Actual entities are not the only class of entities that exist in the Universe as described by Whitehead, there is another class called eternal objects. These eternal objects are similar to an actual entity except they do not prehend, as they do not exist in the same manner that actual entities do. Hooper [4] reminds us that eternal objects are constant across all environments; therefore, they provide the important function of being a form of definiteness to actual entities. Anything in the Universe that does not have a reference to a definite object in the temporal world is an eternal object [12]. Eternal objects can be prehended by actual entities, which can affect the "experience" of the actual entity but not that of the eternal object.

In the case of UAVs, where the UAV would be the AE, there are multiple items that could be considered to be eternal objects. The simulation of the concept in this paper considers the targets that are being searched for to be eternal objects, as the targets can be prehended; however, they do not prehend or change themselves. Other possible eternal objects in this case could be terrain, such as mountains and trees.

Chapter 4: Prehension

Prehension is the method in which actual entities change and move. As described by Whitehead [12], the manner in which an actual entity is capable of altering itself is based on its environment. The definition of the word "prehension" is a datum prehended by the substrate activity present in the emergence of every actual entity [4]. His doctrine of prehensions describes this action of settled occasions in the world being used to form actual entities experiences. According to Whitehead [12] an important part of prehension is the concept of causality and both efficient causation and final causation must be acknowledged. The belief that past events decide the course of future events is efficient causation. While, final causation is when the "end" is said to play a part in creating the wanted result. Actual entities serve as an example of both types of causation [4]. The effect of the prehension is based on its subjective aim. The guidelines provided are that a prehension occurs between a prehending and prehended object, of which only the prehending object is altered. There are two types of prehension: positive and negative. A positive prehension prehends and changes the prehending object. Whitehead described positive prehensions as "feelings", even though they are essentially operations. To get a feeling from an actual entity it has been referred to as "physical" and to get one from an eternal object it is "conceptual" [4]. A negative prehension prehends and does not change the prehending object although it does have an impact on the overall satisfaction of the actual entity as it does not help to improve it. Hooper painted the following illustration of a negative prehension to make the point clear. Just as the refusal by an individual to love things worthy of being loved, or to see the truth of a situation, even though it stares him in the face, affects his character and his general "tone", so negative prehensions determine in part the emotional tone of the "satisfaction" of the actual entity [4]. Prehensions involving an actual entity prehending another actual entity are

more specifically referred as physical prehensions; where in the case of an actual entity prehending an eternal object it is referred to as a conceptual prehension. Whitehead never went into any detail of how this should be done computationally; however, Primeaux addressed this in [9] among other things. A distance function is used to determine if the distance between the prehending and the prehended entities is over one of the thresholds. The distance function is D = O - I where the prehending entity is I and the prehended entity is O [11].

Chapter 5: Thresholds

The concept of thresholds, first created by Primeaux [8], represents the radius of sight that each AE holds when examining its environment. With these thresholds there is a Euclidean distance that exists, beyond which all prehensions are negative. Primeaux and Saunders [9] later proposed the concept of having both an internal and external threshold. There is an external threshold, which states that everything beyond that distance causes a negative prehension. There is also an internal threshold, which states that everything within that distance also causes a negative prehension. This creates a band around the AE where prehension occurs, as shown in the figure below.

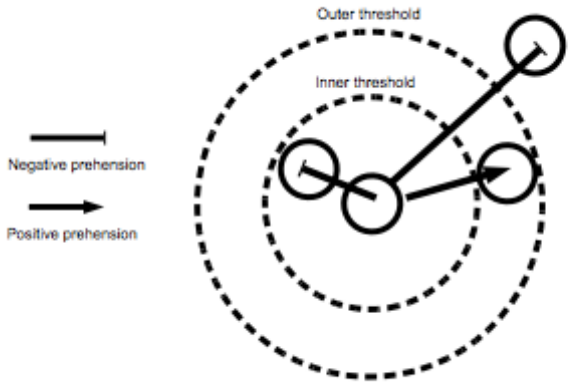

Figure 1: Inner and Outer Thresholds [11]

Chapter 6: Subjective Aim

Subjective aim is a term that refers to the wanted outcome from certain actions. As Hooper [4] states, the subjective aim of the prehension should deal with the immediate present as well as the relevant future. As Saunders [11] mentions, the subjective aim can always be different outside of the relevant future. Primeaux described subjective aim in saying that the subjective aim of a prehension function that causes an acorn to grown into an oak tree, is to make an oak tree. In the case of this application, our subjective aim of our prehensions would be locating the targets, which will be discussed at a later point.

Chapter 7: Colony of Prehending Entities

A COPE is a colony of prehending entities; this concept was introduced by Primeaux to simulate the behavior of actual entities. The actual entities are in a closed, artificial environment. COPEs have two functions: to select two actual entities and to start the prehension process. All entities will prehend every other entity and they all move; this is considered one time step. Most swarms base their decisions on a set of rules; however, COPEs decisions are more function driven. The computational characteristics exhibit emergent behavior at the COPE level and are robust to noise. COPEs require no global control mechanism and their representation is data centric [9]. Therefore, they lend themselves well to decentralized control strategies.

As discussed in [9], the emergent behavior of a COPE is based off the selection of prehensions functions and finding the appropriate critical attributes (CA). Determining both of these can be difficult; however, once found creating the COPE algorithm is simple. Below is the pseudo code algorithm as provided by Lamont [9].

Create a distribution of AEs

Until convergence criteria are met

    Randomly select a pair of AEs

    Select one AE in the pair to prehend the other

    Modify the prehending AE according to its prehension function

The method of COPE to be used in the proposed control method does not randomly select a pair of AEs from the current distribution. Instead, it cycles through each combination of AEs in order to make sure all of them perform a prehension in each clock cycle.

Chapter 8: Example Situation of Actual Entities

The following example of a simulation of an ecosystem is used to better explain the use of actual entities in a computational manner. The ecosystem modeled is that of a jungle and a limited number of animals are selected from that food chain, in order to create a manageable model. Each animal in the environment is an actual entity. The prehension functions for each type of animal would be different in both their thresholds and their reactions. Based on the type of animal the range of sight would be different; for instance, take a frog and a lion. The frog would have a range of about 1 meter, while a lion can see about 60 meters. Therefore, their thresholds would need to be created proportional to those distances. As for the reactions to their individual prehensions, since it is a model of an ecosystem, each animal would react by moving toward food, away from an enemy or moving towards/staying neutral (based on the type of animal) in relation to another animal of the same species. There also would need to be a threshold set to determine when a predator will eat its prey. Once all the difficult decisions around the creation of the prehension functions are made, the simulation can be run by having every actual entity prehend every other and have them all move at the same time.

Chapter 9: Swarm Intelligence

Swarm intelligence has been observed in school of fish, swarms of different insects, and flocks of birds. As Krause [5] points out, this group living does enable problems to be solved that would be impossible or extremely hard for an individual from one of these groups. These behaviors have been studied and the ability that the groups have of solving cognitive issues larger than what one individual could facilitate has become known as swarm intelligence (SI). This term was coined by Beni and he used it to describe the organization of simple agents using nearest neighbor interaction [1]. Individual entities in SI are not sophisticated on their own; however, when their actions are combined more intricate tasks can be done. Each tends to have its own set of simple functions that it can perform. The SI is an emergent behavior that comes from these simple functions being done by multiple individuals and causes the appearance that they are all communicating and working together. The definition proposed by [5] for SI is "two or more individuals independently, or at least partially independently, acquire information and these different packages of information are combined and processed through social interaction, which provides a solution to a cognitive problem in a way that cannot be implemented by isolated individuals."

Optimization problems commonly use SI as shown in [1, 6]. In these cases the SI is being used for ant colony optimization, particle swarm optimization, and a continuous optimization algorithm for tuning real and integer parameters. A common optimization that SI is used in conjunction within teaching environments is the traveling salesman problem. SI is actively being used in regards to unmanned aerial vehicles (UAVs) and their decentralized planners.

Chapter 10: Swarms vs. Actual Entities

The similarities and difference between SI and AEs are important since SI is commonly used with UAVs and AEs are being proposed as a control method for UAVs. Primeaux suggested that AEs should be considered as a form of SI. However, as Saunders [11] points out, to prove this concept all SIs must be modeled by AEs and AEs must be able to do everything that a SI model can.

The main difference between a swarm and AEs are the way in which the data is set up. In the case of AEs, the data and the acting agents are one and the same. On the other hand, Saunders [11] points out that swarms have the concern of the number of agents that are acting on the data. The similarities between the two are that both have emergent, intelligent behavior, both need convergence to find a solution, and both require the parameters to be adjusted in order to achieve an acceptable level of results [11].

Chapter 11: Centralized/Decentralized Planners

Chapter 11.1: Centralized Planners

With centralized planning the process of allocating tasks and path planning is all solved at one central point. This is normally a cluster of computers at the grounds station that do all of the heavy computation. Situational Awareness (SA) is communicated to a central point, normally a server, and used to create a plan for all of the UAVs. Centralized planning is more structured, has more computational power, and less UAV to UAV communication. On the down side, there is a single point of failure and the mission range is restricted based on the network [3]. As pointed out by Parunak [7] the use of a centralized planner in a rapidly changing combat situation would have an unacceptable time delay as data from the UAVs would need to be processed and new commands would need to be generated. Also, in 2002 the centralized model that was used needed a team to control a single UAV; this made large swarms impossible due to the amount of needed manpower [7]. Centralized planners were commonly used as UAV control strategies; however, they are not suited for use with swarms that has lead to more research into the use of decentralized planners.

Chapter 11.2: Decentralized Planners

In nature, biological swarms have shown that they can successfully complete tasks using emergent behaviors without global communication for centralized control [2, 9]. This has lead to more interest in mimicking their behavior and searching for decentralized control strategies for UAVs. For decentralized planning the whole problem or part of it is solved locally on each

plane. The ground stations only job is to receive feedback from the planes and release tasks. Decentralized planning has no single point of failure, an extended mission range, and can be flexible regarding dynamic targets. On the other hand, there is a need for processing on the plane, decentralized planning relies heavily on the communication bandwidth, and it can take longer to find a solution. In the implementation of a decentralized planner each plane uses the same algorithms to come to a solution based on their individual SA. If an infinite communications bandwidth were assumed, each plane would have the same SA and solution. However, communications bandwidths are not infinite and that is where the decentralized planner runs into problems. When the planes cannot all communicate with each other this causes them to have different SA and can cause each plane to come up with different global solutions. This is why most decentralized planners use a consensus algorithm, which would help to converge on an SA before any part of the tasks are assigned. This has its own problems that accompany it involving both time and data. To converge on an SA data would need to be transmitted from each plane; this paired with low-bandwidth environments can cause severe latency, which can extend the time spent assigning tasks [3].

Chapter 12: Control Strategies

There are many ideas to what makes an effective control strategy for swarms and/or UAVs. One that is simple and to the point is brought up in Parunak [7] and consists of four requirements, also referred to as the Four D's. These are diverse, distributed, decentralized and dynamic. A control system must be diverse in its functions, the information it can handle, the entities that it can communicate with, and the sources that it can get information from. Distributed systems are important where there is any concern over issues with long-range communications bandwidth, by being distributed each component can just talk with its neighbors to pass messages along. Decentralized systems are important in order to not have a single point of failure and they decrease chance of a time delay. Finally, dynamic abilities are important as environments are constantly changing, especially in warfare situations [7].

Chapter 12.1: Control Strategies – Baseline

The baseline strategy is a very simple decentralized control strategy, which was used by Bonabeau [2] as the main strategy to compare other control strategies with to determine their efficiency. The UAVs fly in a straight line until a boundary of their designated search area is reached, then they turn whatever degree is needed to avoid crossing the boundary. In the figure below, the results that Bonabeau [2] had from using this strategy are shown. The left side is the percentage of the search field visited in 1,000 units of time compared to the number of UAVs. The right side shows the coverage that each UAV has based on the number of UAVs currently in the field.

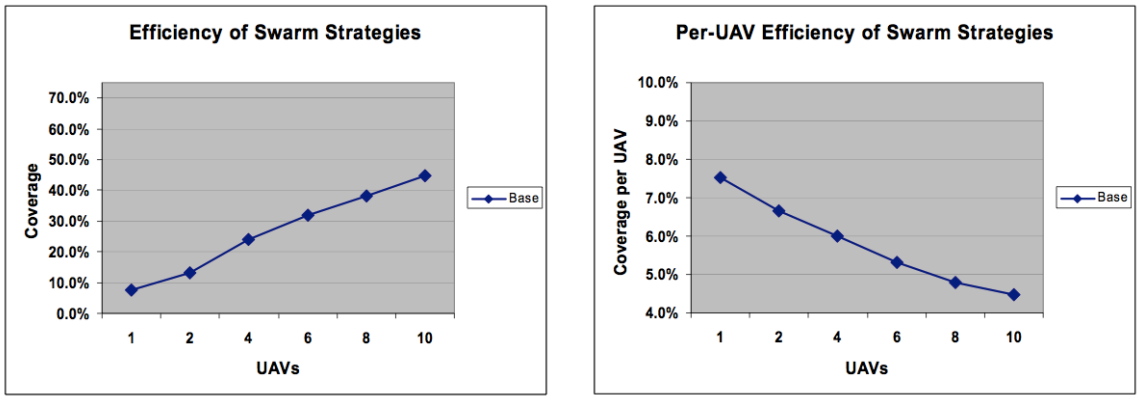

Figure 2: Baseline Efficiency [2]

As expected from these trials by Bonabeau [2] the number of UAVs increase and so does the percent of coverage. However, with the increase in the number of UAVs each individual UAV has less coverage as they begin to fly over areas already covered.

Chapter 12.2: Control Strategies – Random

The random control strategy uses the baseline strategy as the core of its method, with the added option at each time step for the UAV to change its course by a small random angle [2]. This is another type of decentralized control strategy that was used for comparisons by Bonabeau [2]. This is also referred to as the jitter strategy.

Chapter 12.3: Control Strategies – Repulsion

The methods used in this decentralized repulsion control strategy are reminiscent of how prehension works with actual entities. Each UAV has a radius around it in which it checks for other UAVs, if any are found then the UAV moves in a direction away from them [2]. This type of behavior is based off the thought that if the UAV moves away from other UAVs it is more

likely to cross areas in the search field that have not been touched. The figure below shows a single run by Bonabeau [2] with 10 UAVs and a 60-unit repulsion radius.

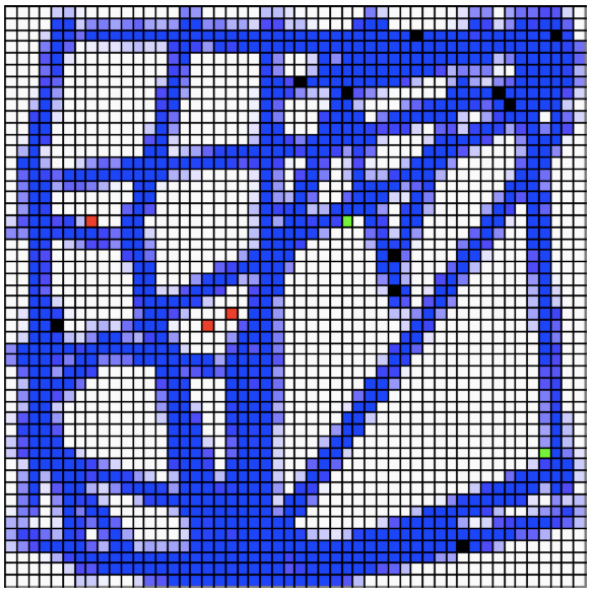

Figure 3: Simulation of Repulsion Control Strategy [2]

Chapter 12.4: Control Strategies - Pheromones

Ants are studied and modeled for use in swarm theory for their methods of using pheromones to determine shortest paths. Due to the impressive coordination without ever having direct communication, but rather from using pheromones in their environments, their methods are being imitated for use in UAV coordination. Pheromone dynamics are used as a control strategy for coordinating multiple UAVs in [2, 7]. In [2] when the UAVs move over each cell on the search field they drop a pheromone marker showing that they were there. The UAVs can sense the pheromone markers and then adjust their headings to move to areas without them. Bonabeau [2] does not stay true to how pheromones work, in that they don't take into consideration the dispersion of the pheromones over time. This could affect the end results of their research, as

there would be a higher chance of cells being revisited by UAVs, as the pheromones fade away and are no longer sensed. The figure shown below is the coverage achieved by [A] using this control strategy with 10 UAVs. Bonabeau [2] discusses that this strategy creates a much more elaborate pattern while searching and leaves less unexplored space in comparison to the repulsion strategy, where there are wide areas left open.

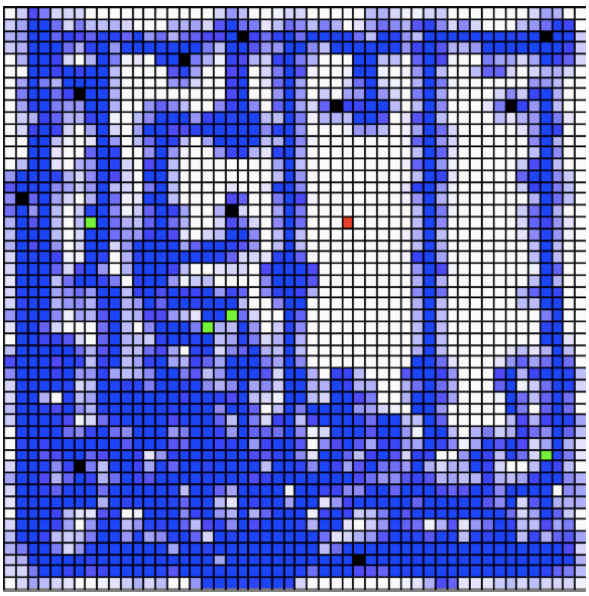

Figure 4: Simulation of Pheromone Control Strategy [A]

Digital pheromones are also used as a control strategy in [7], where the pheromones fields constructed in the search environment are the potential fields that guide the movements of the UAVs. They use a grid of place agents, which in a battle space would be unattended ground sensors, to hold and distribute information regarding the pheromones, locations of the UAVs, locations of other place agents, and locations of the targets. When a UAV enters a certain region of the search field it interacts with the place agent to determine whether the pheromone flavor is repulsive, meaning another UAV was there and left it, or attractive, meaning that a target is close

by. In this implementation of pheromones, Parunak [7] takes into account the true nature of how pheromones behave (as seen in ant colonies) in a way that Bonabeau [2] does not. They take into account the aggregation, evaporation and diffusion of the pheromones. One oddity with the method of Parunak [7] is that they discuss attractive pheromones for the targets, but do not mention where those come from, if they are preset, etc. Following is a figure showing Parunak's [7] path planning using their method of digital pheromones.

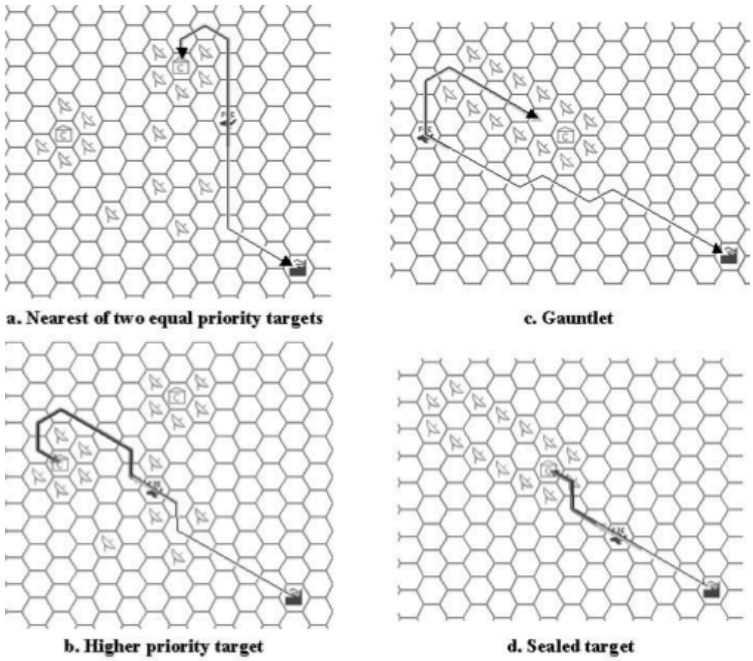

Figure 5: Path Planning with Digital Pheromones [7]

Chapter 12.5: Control Strategies – Evolutionary Algorithms

The environments that UAVs are exposed to are constantly changing; this creates a challenge for control mechanisms, as they would need to be dynamic in order to react to these changes. Using an evolutionary algorithm, particularly for path planning, would allow for dynamic reactions. Both Lamont [6] and Rathburn [10] use this control strategy for path planning; they each focus

on a subset of this class referred to as Genetic Algorithms (GA). With a GA the initial population is created randomly, then the algorithm is run in a loop until convergence is met. The members of a generation are essentially chromosomes with enough alleles to represent all the variables needed. Each loop of a GA represents one generation and each generation can experience both mutations and crossovers. A mutation is when a random member of a generation has a random variable of its design changed. A crossover is when two random members of a generation are chosen and a handful of variables are switched between the two, this is shown in the following figure.

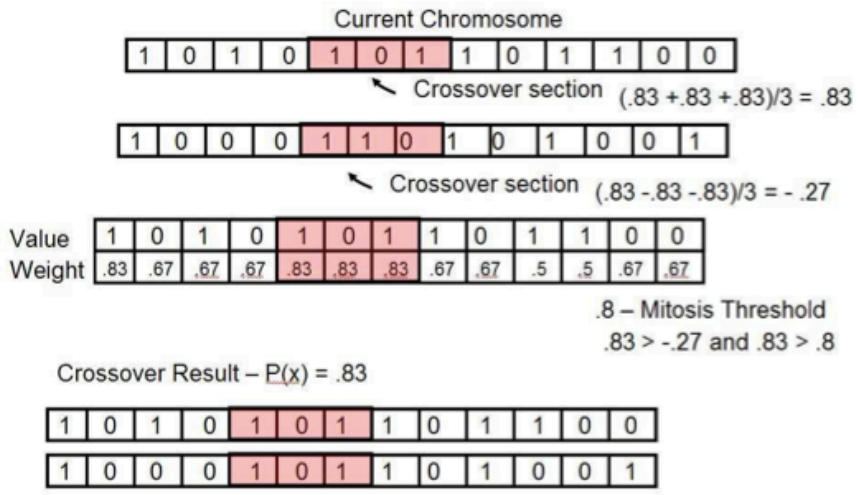

Figure 6: Genetic Algorithm – Crossover [6]

Once those are complete, a fitness function is used in order to rate each member of the generation. The fitness score is used to represent the probability that each individual member could appear in the next generation. A visual way to imagine this process would be a roulette wheel, where members of the generation are represented as many times as the value of their

fitness score. The wheel is then spun to pick each member of the new generation. The final solution is chosen at convergence, normally when a percentage of the generation is above a predetermined fitness score or when the same member design dominates the generation. Due to the method used to pick the final solution, the most optimal solution is not always accepted. Lamont [6] uses the GA to select a combination of control parameters or the weighting that would be applied to rule sets. The chromosomes are constructed from combining the weights and the behaviors each weight is associated with. This system uses rule sets that are weighted in order to decide the next action that will take place, which creates what Lamont refers to as a behavioral archetype (BA). The following figure shows an example of a chromosome.

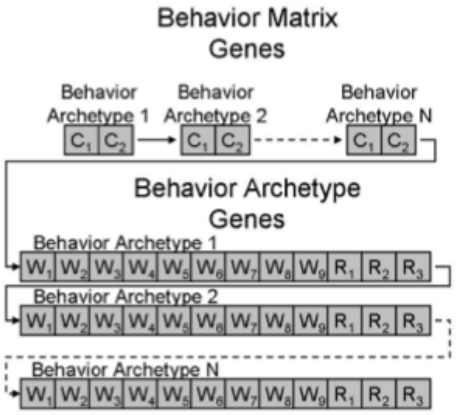

Figure 7: Chromosome of Behavior Archetype Genes [6]

The ten rules that are used in this method and manipulated with the GA are: flat align (vector aligning with neighbors), separation from the cluster, cohesion with the cluster, obstacle avoidance, collision detection/avoidance, target orbiting, attraction towards the center of all targets, weighted attraction to the closest target, target repel, and weighted target repel.

On the other hand, Rathburn [10] uses GA in an entirely different way. Each member of a generation is a sequence that determines that path that the UAV would take. Two types of segments make up this sequence; they are a straight line and a curve. The length, radius and speed are the variables that can be affected by mutations. The method for the basic functionality of the GA is altered by Rathburn [10] from the default described above. The population starts out with 20 members and grows to 40 members by adding the mutated members to the original. Randomly choosing a variable for mutation, mutating every current member, and saving the mutated copies to the generation creates these extra 20 members. Round robin is used for the selection of the next generation, comparing two randomly chosen members at a time. The figure below is the overview of this method.

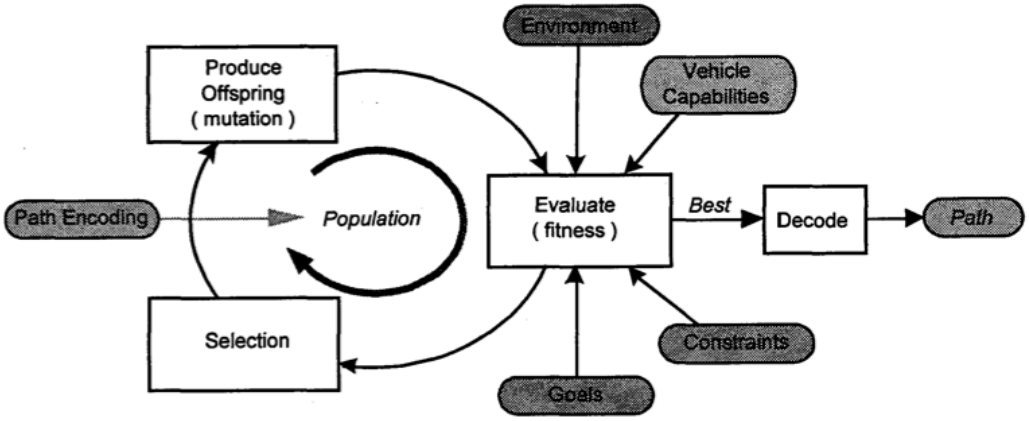

Figure 8: Overview of an Evolutionary Algorithm Method [10]

Both methods effectively utilize the evolutionary algorithm in their UAV control methods for path planning. They have shown that evolutionary algorithms work well for dynamic planners as they allow for a continuous update to the direct path or the rules/weights that influence the movements of the UAVs.

Chapter 13: Actual Entity Problem Domain

The objectives are for the AEs to form a COPE and use prehension to navigate the designated search area and locate the targets within that area. They should store the locations of the target and begin to return to their start location, once all of them have been found. Making decisions independently and communicating strictly through prehensions, should increase the search effectiveness of the COPE.

Chapter 13.1: Simulation of the UAVs and Search Area

The simulation is used to simply demonstrate the concept of the AE behavioral control method in a two-dimensional environment. There are many real world variables that are not taken into consideration during the simulation. Some of the constraints that would be taken into consideration in a real world domain would be weather conditions, communications bandwidth constraints, sensor ranges and unreliability, the ability of a target to move, altitude of the UAVs and noise in the data. The simulation is created was created using the programming language Java.

Chapter 14: Actual Entity Behavioral Control

Earlier, a method of determining an effective control strategy for UAVs was discussed. This method is called the Four D's and was created by Parunak [7]. The Four D's are diverse, distributed, decentralized and dynamic. When comparing the proposed AE behavioral control method to these four requirements it satisfies them all. The method is diverse in that it can easily be manipulated to handle different types of information and recognize entities as either AEs or eternal objects. It is distributed in that each AE simply prehends the AEs that are within range, there is no dependence on large communication ranges. The control method is decentralized as each AE acts independently. Finally, the method is dynamic so that it can adjust to an increase or decrease in UAVs; it can also adjust to changes in the environment as movements are determined every clock cycle as the current environment is observed.

Chapter 14.1: Actual Entity Representation

Actual Entities are represented as a Java class, so that multiple instances can be created in order to represent each UAV needed for the simulation. Each AE has a number of variables that it stores and actions that can be preformed. The constructor for creating each AE instance requires input parameters of the index to be assigned, the starting x and y coordinate, and the total number of targets located in the search area. Regarding the targets, each AE stores the total number of targets it has found/verified to be found, the unique index and coordinates of each of those targets, and the total number of targets needed to be found before heading home. The AE contains a Boolean variable called headingHome that is set to true, once all the targets are found, preventing further prehensions from occurring while the AE is heading to its final destination. Another variable in the AE is the heading; this is based off of the hands on a clock if it were

overlaid onto the flat two-dimensional search area. Each AE has methods to set and retrieve each of these variables. The AE class also holds the methods that decide whether or not to prehend another AE or eternal object and the prehension method, which will be discussed in section 3.

Chapter 14.2: Eternal Object Representation

Eternal objects are also represented using a Java class, for this simulation they represent the targets. Targets fall under the category of eternal objects, as they are not changed by their interactions with AEs; however, the AE is changed by prehending the target. The constructor for the eternal object class has the input parameters of the index, an x and y coordinate where it is located, and an integer value that determines the identity. The identity variable allows for other types of eternal objects to be created.

Chapter 14.3: Prehension Algorithm

The prehension method is used to direct each individual UAVs movements, based on what it observes in the environment. This algorithm can be manipulated to consider as many variables as any system would want considered. In its current state, each individual UAV is impacted by other UAVs within a certain radius, targets, and the border of the search area.

The algorithm starts with a loop that will cycle through an array of AEs representing the UAVs. In each loop the current UAV is assigned to the name of uav1, the current x and y coordinates are assigned to x1 and y1, and the current heading is assigned to currHeading.

The Boolean variable of headingHome is checked; if the value is true, the AE will not preform any prehensions and it will simply return to where it started. If the value if false, another loop is

started to cycle through the array of AEs representing the UAVs. This loop will skip over the first UAV, as an AE should not prehend itself. This second UAV will be assigned the name of uav2, and its current x and y coordinates are assigned to x2 and y2. Now, another method named toPrehendOrNot is called with the parameters of uav1 and uav2 being passed into it. This method determines if uav2 is within the threshold of uav1 and returns the value of true if that is the case. If true, the algorithm then checks if uav2's value of headingHome is true. If it is the values of the number of targets found and the target locations are copied to uav1, the value of headingHome for uav1 is set to true, and the heading is set to point toward the starting area. If uav2 is not heading home, the prehension continues by comparing the targets found by each UAV. In the case of uav2 having found targets that uav1 hasn't yet, those locations would be added to the location array that uav1 has and the total number of targets found is also updated. After this is complete, the headings of the two UAVs are compared and if uav2's heading is within a range of 1 to either side of uav1's heading there will possibly be a change to uav1's heading. The algorithm finds what side of uav1 uav2 is located on, in order to calculate if the heading of uav1 should be increased or decreased to curve away from uav2. This is to stop them from going exactly the same direction in their search.

Once uav1 is done prehending all of the other AEs, it begins to prehend the eternal objects, which are the targets. A loop is created to cycle through all of the eternal objects in the targets array. In each loop, the target is assigned the name of eo, the x and y coordinates are assigned to x2 and y2, and the unique index is assigned to index. The method of toPrehendEternalObjectOrNot is called with the parameters of uav1 and eo to determine if the target is within the predetermined eo threshold. If true, the target location is compared to uav1's target location array to see if it is already present. If it is not, the target location is added and the

targets found counter is incremented. The headingHome Boolean of uav1 is also switched to true, if the new number of targets found is equivalent to the total number of targets.

Using the counter results from tests that were run, the new heading of uav1 is assigned. Based on the heading the x and y coordinates are updated in that direction. If uav1 is heading home and is within 20 of the (0,0) starting location, the UAV becomes inactive. Uav1 is updated in theCOPE array and the whole process is started over with the next UAV in the AE array.

Chapter 14.4: Convergence

Convergence is based on two factors. The preferred type of convergence is met when all UAVs are heading home, meaning they have found all the targets. This is not consistently accomplished in a reasonable amount of time for every simulation; so, it is necessary to have a maximum time. If all targets are not found by the maximum allowed time the program is terminated. The method for convergence can be found in the COPE class file.

Chapter 15: Design of Experiments

The experiments test the AE design and prehension algorithm using a combination of different numbers of planes and targets. The targets locations are randomly generated at the beginning, each time the program is run. The UAVs start lined up down the left side of the search area. The experiments were run with the UAVs increments of 5 and 10 and the targets incrementing 1, 2, and 5. The AEs outer threshold was tested at 40 and 60. The maximum allowance of time used in convergence was tested at 2000 and 5000.

In the beginning of each test, the UAVs are lined up down the left side of the search area starting at (0,0). Each UAV is represented by its unique index number and the text is the color white. The number of targets specified are randomly generated and placed in the search area. Each of these is also represented by their unique index number and the text is green. The starting coordinates of the targets are printed to the terminal as a reference, as shown in the following figure.

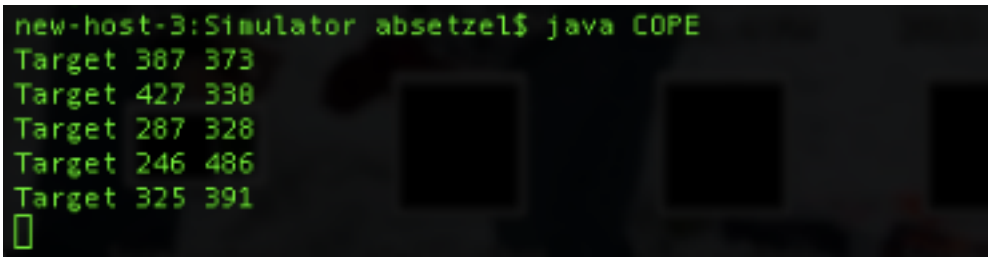

Figure 9: Beginning of Simulation – Target Coordinates

The starting display of the simulator can be seen in the figure below.

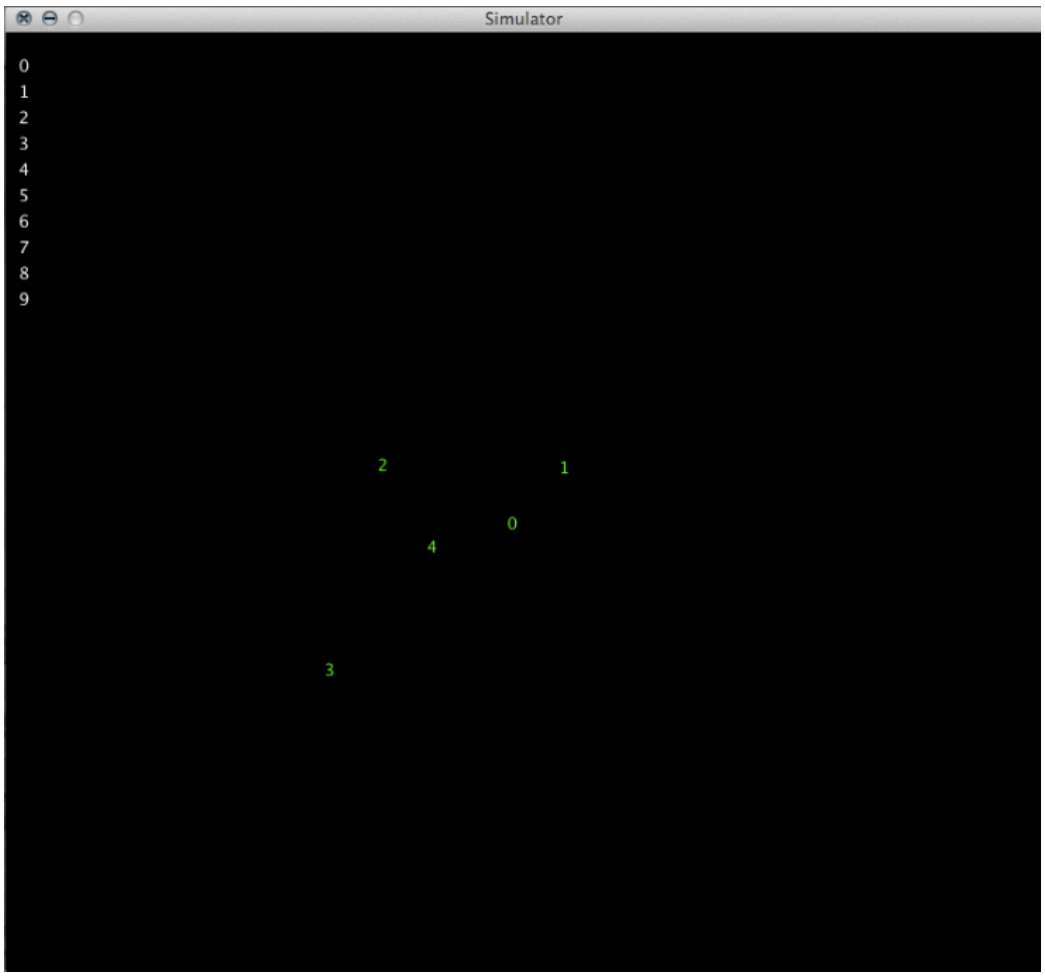

Figure 10: Beginning of Simulation for 10 UAVs and 5 Targets

On convergence the program stops running and the current locations of the UAVs can be examined. The following figure is for the 2000 maximum time constraint with a threshold of 60, 10 UAVs and 5 targets.

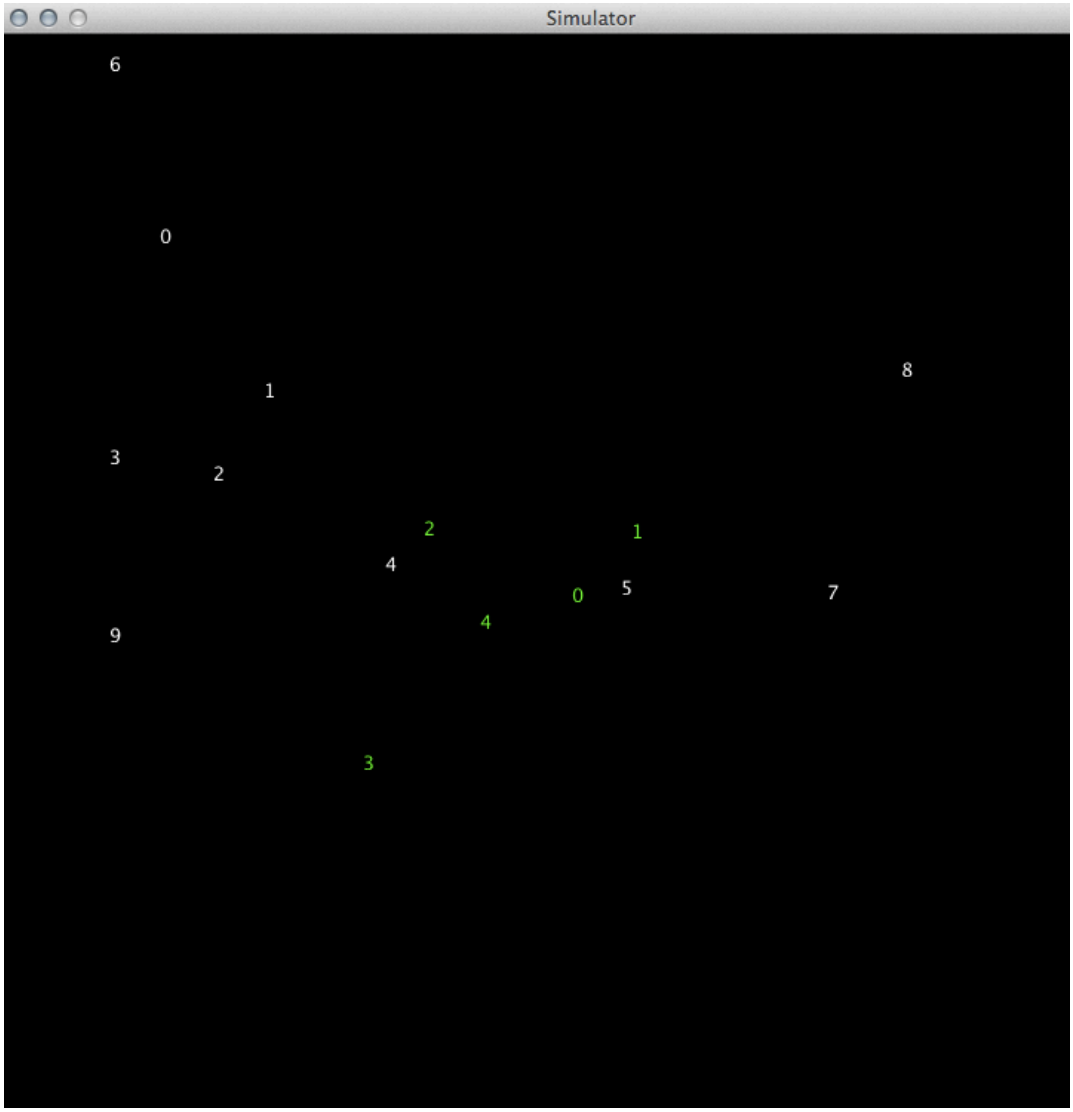

Figure 11: End of Simulation

Chapter 15.1: Results and Statistical Analysis

The results show that using AEs as a control method can be effective for performing searches. However, the algorithm and the variables need to be fine tuned in order to produce consistent results. There were 12 searches completed for each of the two maximum convergence times. The following figure shows how many of those searches could be completed within the time constraints.

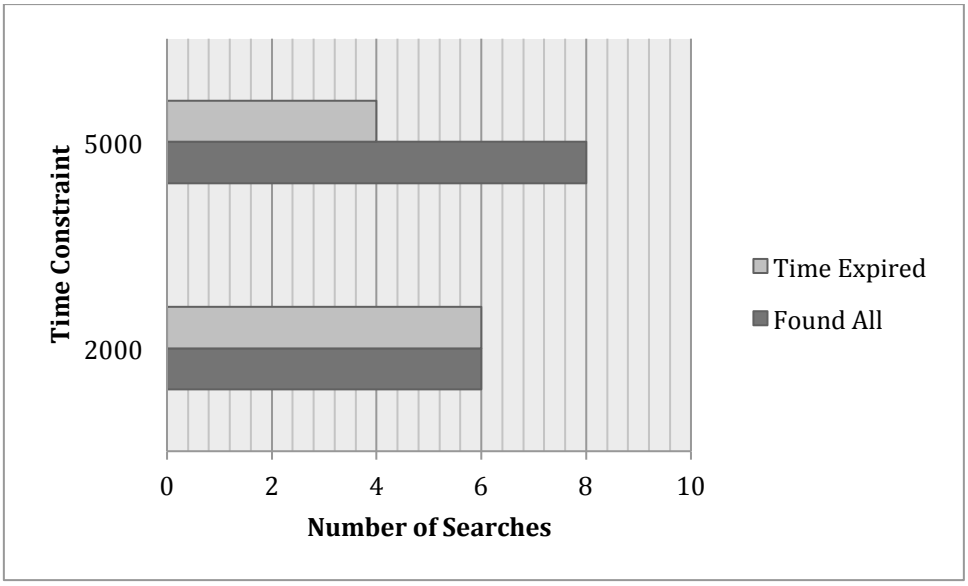

Figure 12: Number of Searches Completed vs. Convergence Time Constraint

As shown, for the 2000 seconds time constraint there were an even amount of searches completed as those that ran out of time. However, when expanding the time constraint to 5000 seconds a 2:1 ratio of completed searched to those with the time expired emerged.

The searches also highlighted a performance difference when altering the external threshold of the UAVs for other UAVs. This was tested at 40 and 60; the results in the figure below demonstrate how much changing the thresholds can affect the data.

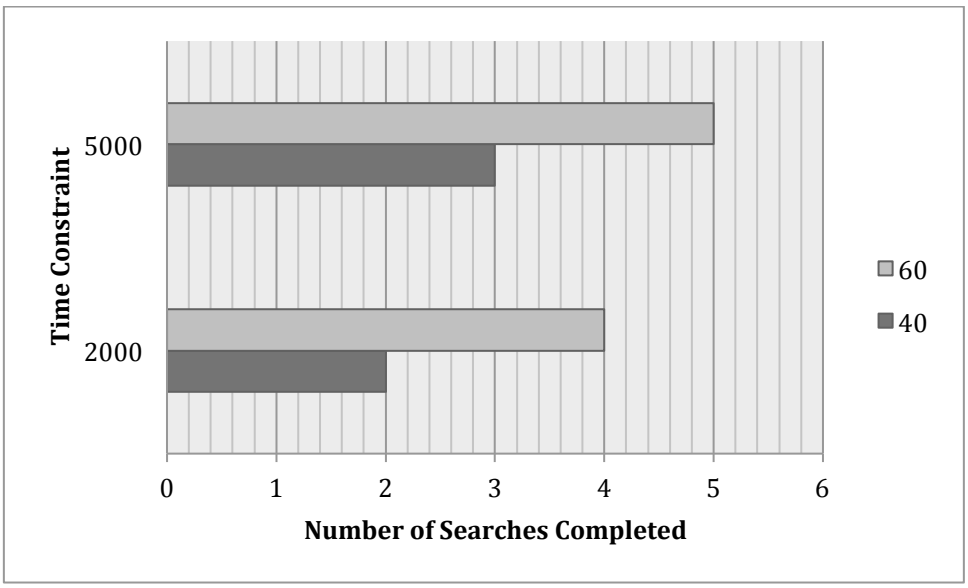

Figure 13: Number of Searches Completed at Time Constraints Separated by Thresholds

In the case of both time constraints, the UAVs with a threshold of 60 completed either double or close to double the number of searches when compared with those with a threshold of 40. When comparing all the collected data, there was one section where the searches were consistently completed that was when the UAV number was ten and the threshold was 60. The following figure shows the number of searches successfully completed in terms of the number of UAVs.

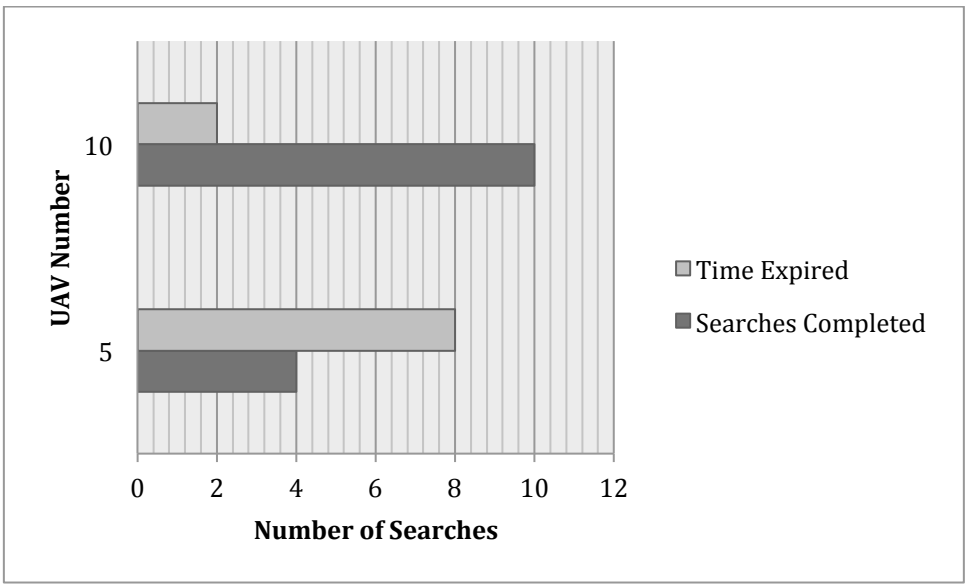

Figure 14: Number of Completed Searches Based on Number of UAVs

As shown in the figure, the majority of searches involving ten UAVs were successful; while, the majority of searches involving five UAVs ran out of time.

The majority of searches involving only one target were successfully completed in comparison to when two or five targets were used.

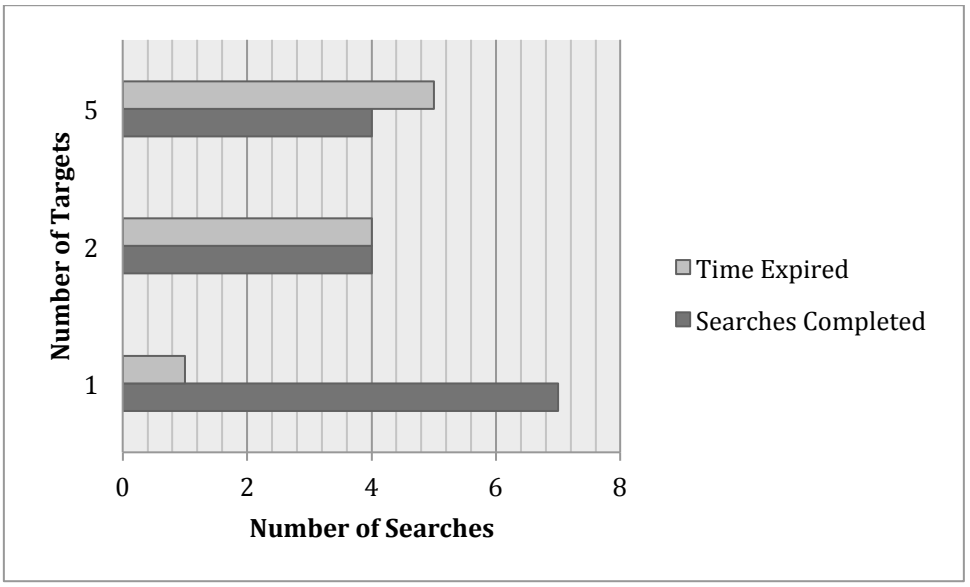

Figure 15: Number of Targets Compared to Searches Successfully Completed

With any more than one target the chance of successfully locating all of the targets drops to around 50%.

Based on the number of UAVs and the external thresholds set, patterns emerged in the movement of the UAVs during the searches. In many cases, the UAVs split into two groups; one that would go straight across the search area and one that would move downward. In other cases they would take turns slowly moving at a diagonal across the search area before turning separate directions. The following figure shows a search involving ten UAVs with external thresholds of 40 splitting into two search groups.

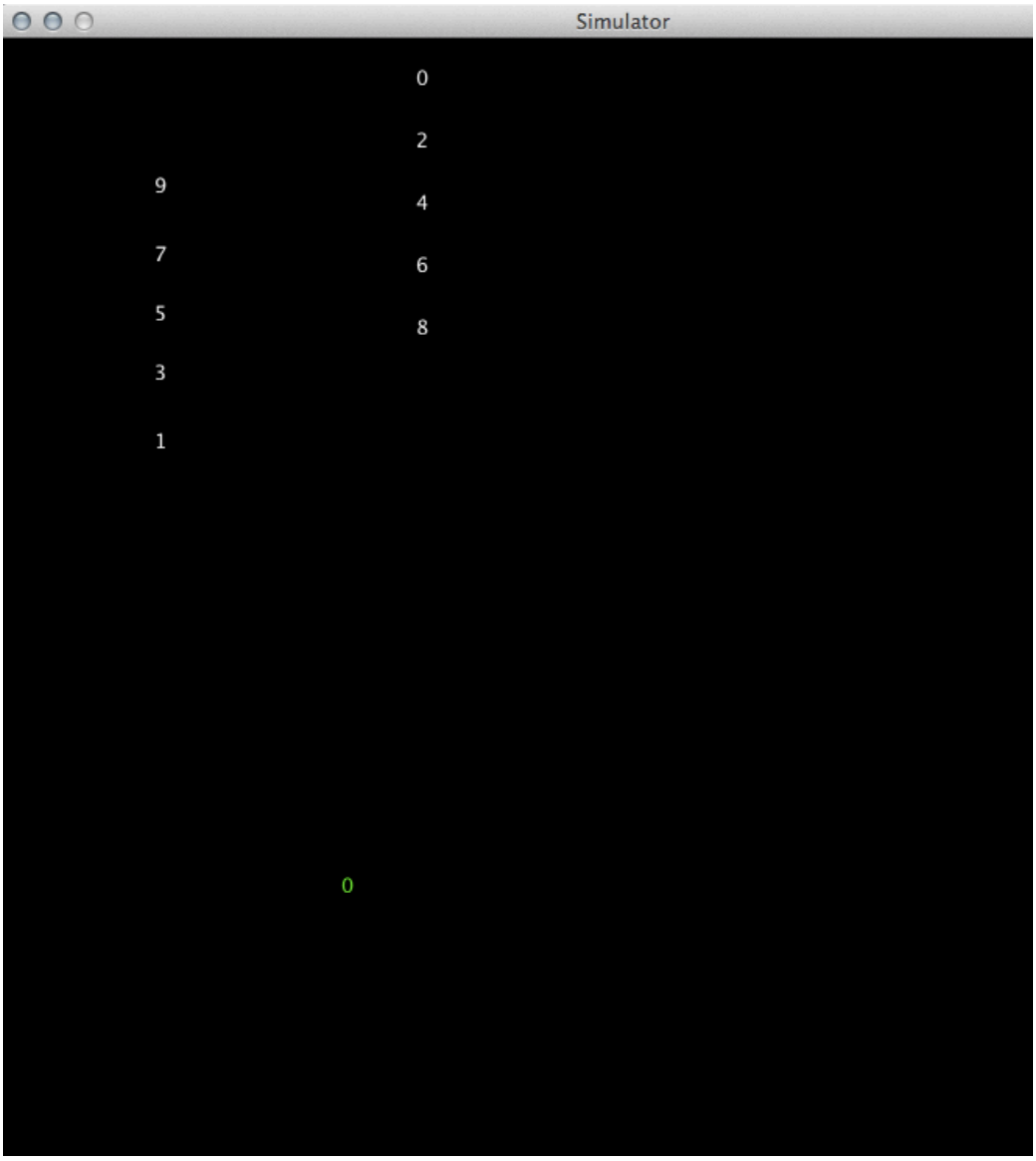

Figure 16: Emergent Search Pattern with 10 UAVs

They stayed in evenly spaced lines until they hit a border, at which point the shape and direction of the group began to change. In the next figure, another emergent pattern can be seen involving five UAVs.

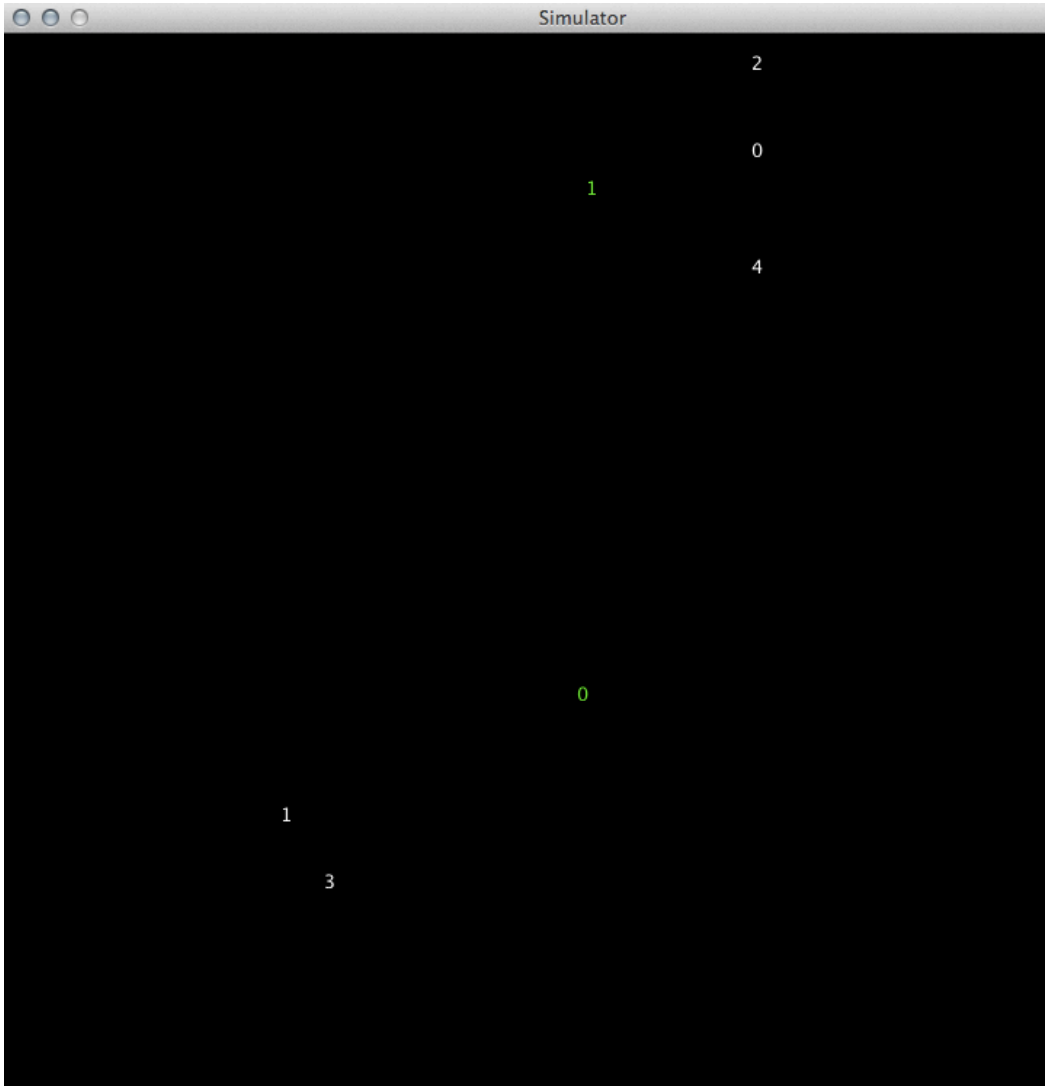

Figure 17: Emergent Search Pattern with 5 UAVs

This type of grouping aided the search, as one group would normally locate the targets along the top of the environment while the other group would locate the targets along the bottom. They would then cross paths, prehend the found targets from the other UAVs and start heading home.

Chapter 16: Conclusions

The use of AEs as a behavioral control method can be effective with the appropriate calibrating to the algorithm. Emergent patterns in the searches supported the ability of AEs to have emergent, intelligent behavior. The success of the searches varied greatly, some of which is due to the random generation of targets for each run and some of which is from the need of the prehension algorithm and variables to be more finely tuned. The results show promise for this control method and it is believed that future work on the method would create a viable behavioral control for UAVs.

Chapter 16.1: Future Work

There are a few categories of future work involving the use of AEs as a UAV control method. The first relates to the prehension function; many more variables could be taken into consideration within this function. An example of these, would be the addition of variables applying to sensors used by UAVs. The second relates to the simulation and testing methods. The simulator could be expanded into a three dimensional environment that also accounts for terrain and environmental variables. The testing should become more comprehensive not only in testing this control method, but comparing it with other control methods using the same simulator to provide consistency across the results. The third relates to the application of the method using the same hardware that would be used in actual UAVs. This would allow tests to determine the amount of processing power needed and how much of a communication bandwidth would be needed.

List of References:

[1] Blum, C., & Li, X. (n.d.). *Swarm intelligence in optimization.*

[2] Bonabeau, E., Shargel, B., Gaudiano, P., & Clough, B. (n.d.). *Swarm intelligence: a new c2 paradigm with an application to control of swarms of uavs.* 8th ICCRTS Command and Control Research and Technology Symposium.

[3] Choi, H., Brunet, L., & How, J. P. (2009). *Consensus-based decentralized auctions for robust task allocation.* IEEE Transactions on Robotics, 25(4), 912-926.

[4] Hooper, S. E. (1941). *Whitehead's philosophy: Actual entities.* Philosophy, 16(63), 285-305.

[5] Krause, J., Ruxton, G. D., & Krause, S. (2009). *Swarm intelligence in animals and humans.*

[6] Lamont, G. B. (2007). *Uav swarm mission planning using evolutionary algorithms - part I.*

[7] Parunak, H., Purcell, M., & O'Connell, R. (2002). *Digital pheromones for autonomous coordination of swarming uav's.* American Institute of Aeronautics and Astronautics.

[8] Primeaux, D. (2000). *Trust-based learning of data characteristics by an actual entity.* Proceedings of the ACIS 1st International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing.

[9] Primeaux, D., & Saunders, B. (2006). *Finding global and local maxima in 3d space using swarm-like behavior in a colony of prehending entities*. Informally published manuscript, Computer Science, Virginia Commonwealth University, Richmond, VA.

[10] Rathburn, D. *An evolution based path planning algorithm for autonomous motion of a uav through uncertain environments.*

[11] Saunders, B. S. (2007). *Observational intelligence: An overview of computational actual entities and their use as agents of artificial intelligence*. (Master's thesis), Virginia Commonwealth University

[12] Whitehead, A. N. (1978). *Process and Reality.* New York, NY: The Free Press.

Appendix A: Environment Simulator Class Code

```
/*****************************************************
 * Author(s)    : Erica Absetz
 * Date         : March 6, 2013
 * File         : EnvironmentSimulator.java
 * Language     : java
 *****************************************************/
import java.awt.Color;
import java.awt.Dimension;
import java.util.Random;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class EnvironmentSimulator{

        static JFrame myFrame;
        static JLabel[] labelArray, targetLabelArray;

        public EnvironmentSimulator() {
      myFrame = new JFrame("Simulator");

      myFrame.setLayout(null);
      myFrame.setPreferredSize(new Dimension(800, 800));
      myFrame.getContentPane().setBackground(Color.black);
      myFrame.setResizable(false);
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      myFrame.pack();
      myFrame.setVisible(true);
   }

   public static void createJlabels(int numOfLabels) {
      labelArray = new JLabel[numOfLabels];
      for (int i = 0; i < numOfLabels; i++) {
         labelArray[i] = new JLabel("" + i);
      }
         }

         public static void createTargetJLabels(int numOfLabels){
                  targetLabelArray = new JLabel[numOfLabels];
                  for (int i = 0; i < numOfLabels; i++) {
         targetLabelArray[i] = new JLabel("" + i);
      }
         }
```

```
    public static void initializeLabels(){
            int length = labelArray.length;
            Random generator = new Random();
  int x, y;

  //starting at fixed point
  int inc = 1;
  for (int i = 0; i < length; i++){
    x = 10;
    y = inc * 20;
    inc++;
    labelArray[i].setForeground(Color.white);
    labelArray[i].setBounds(x, y, 10, 10);
    myFrame.add(labelArray[i], 0);
  }
    }

    public static void initializeTargetLabels(){
            int length = targetLabelArray.length;
            Random generator = new Random();
  int x, y;

  //random target locations
  for (int i = 0; i < length; i++){
    x = generator.nextInt(501) + 100;
    y = generator.nextInt(501) + 100;
    targetLabelArray[i].setForeground(Color.green.brighter());
    targetLabelArray[i].setBounds(x, y, 10, 10);
    myFrame.add(targetLabelArray[i], 0);
  }
    }

    public static void update(int index, int x, int y) {
  labelArray[index].setBounds(x, y, 10, 10);
}

public static void move(int index, int x, int y) throws InterruptedException {
  labelArray[index].setBounds(x, y, 10, 10);
}

public static void deactivate(int index) throws InterruptedException {
  labelArray[index] = null;
}

public static void moveALL(ActualEntity[] theCOPE) {
```

```
            ActualEntity[] cope = theCOPE;
        int index = 0;
        int x = 0;
        int y = 0;
        for (int i = 0; i < theCOPE.length; i++) {
            index = theCOPE[i].getIndex();
            x = theCOPE[i].getX();
            y = theCOPE[i].getY();
            if (x == 0 && y == 0) {
                labelArray[index].setBounds(10, 10, 10, 10);
            } else if (labelArray[index]!=null) {
                labelArray[index].setBounds(x, y, 10, 10);
            }
        }
    }
}
```

Appendix B: COPE Class Code

```
/**************************************************
 * Author(s)    : Erica Absetz
 * Date         : March 6, 2013
 * File         : COPE.java
 * Language     : java
 **************************************************/
import java.awt.Point;
import java.util.Random;
import java.io.*;
import java.util.Scanner;

public class COPE{

        static final int ROWS = 700;
        static final int COLUMNS = 700;
        static final int UAVNUM = 10;
        static final int TARGETNUM = 5;
        static int x, y;
   static int num;
        static ActualEntity[][] actualEntities = new ActualEntity[ROWS][COLUMNS];
        static ActualEntity[] theCOPE = new ActualEntity[UAVNUM];
        static EnvironmentSimulator simulator = new EnvironmentSimulator();
        static EternalObject[] targets = new EternalObject[TARGETNUM];

   public static void main(String[] args) throws IOException, InterruptedException {

        simulator.createJlabels(UAVNUM);
        simulator.createTargetJLabels(TARGETNUM);
        simulator.initializeLabels();
        simulator.initializeTargetLabels();

        int[][] result = new int[UAVNUM][3];
      Scanner in = new Scanner(System.in);
      Environment.setEnvironment();
      int count = 0;
      Thread.sleep(10000);
      while (!checkConvergence(theCOPE, count)) {
         System.out.println(num);
         count++;
                        ActualEntity.prehend(theCOPE, simulator, targets);
         System.out.println("Prehended: " + count + " Going Home: " + num);
         simulator.moveALL(theCOPE);
         Thread.sleep(200);
      }
```

```
    System.out.println("Prehended: " + count + " Going Home: " + num);
}

public static boolean checkConvergence(ActualEntity[] theCOPE, int count){
    //check for how many heading home or for max time limit
    int counter = 0;
    for (int i = 0; i < theCOPE.length; i++){
        if (theCOPE[i].getHeadingHome() == true){
            counter++;
            num = counter;
        }
    }
    if (counter == UAVNUM){
        System.out.println("All Heading Home at " + count);
        return true;
    }
    else if (count == 5000){
        System.out.println("Out of Time!");
        return true;
    }
    else{
        return false;
    }
}

public static int getUAVNUM(){
        return UAVNUM;
}

public static int getTARGETNUM(){
        return TARGETNUM;
}

public static void setTheCopeArray(ActualEntity[] cope){
        theCOPE = cope;
}

//Set Up initial environment
public static class Environment{
    static Random generator = new Random();

            public static void setEnvironment(){
              //sets empty grid displayed as a 2 dim array
        for (int row = 0; row < ROWS; row++) {
            for (int col = 0; col < COLUMNS; col++) {
              actualEntities[row][col] = null;
```

```
            }
        }
        int inc = 1;

        for (int i = 0; i < UAVNUM; i++) {
            ActualEntity actualEntity;
                Point point;
            point = simulator.labelArray[i].getLocation();
            x = (int)(point.getX());
            y = (int)(point.getY());

            while (actualEntities[x][y] != null) {
                simulator.update(i, x, y);
            }
            actualEntity = new ActualEntity(i, x, y, TARGETNUM);
            actualEntities[x][y] = actualEntity;
                theCOPE[i] = actualEntity;
            inc++;
        }

        for (int j = 0; j <TARGETNUM; j++){
            EternalObject eternalObject;
            Point point;
            point = simulator.targetLabelArray[j].getLocation();
            x = (int)(point.getX());
            y = (int)(point.getY());

            eternalObject = new EternalObject(1, j, x, y);
            targets[j] = eternalObject;
            System.out.println("Target " + x + " " + y);
        }
                }
        }
}
```

Appendix C: Actual Entity Class Code

```
/*****************************************************
 * Author(s)    : Erica Absetz
 * Date         : March 6, 2013
 * File         : ActualEntity.java
 * Language     : java
 *****************************************************/
public class ActualEntity{

        public static final int ACTIVE = 1;
        public static final int INACTIVE = 0;
        public static final int INNERTHRESHOLD = 0;
        public static final int OUTERTHRESHOLD = 60;
        public static final int EOINNERTHRESHOLD = 0;
        public static final int EOOUTERTHRESHOLD = 40;
        public static final int SIZE = 700;

        public int index, speed, range, heading, status, x, y;
        public int targetsFound, totalTargets, timeCount;
        static int[][] targetLocations;
        static boolean headingHome;

        public ActualEntity (int index, int xCoor, int yCoor, int targetNum){

                this.index = index;
                this.x = xCoor;
                this.y = yCoor;
                this.status = ACTIVE;
                this.targetsFound = 0;
                this.totalTargets = targetNum;
                this.heading = 3;  //based on clock hands
                this.headingHome = false;
                this.timeCount = 0;
                this.targetLocations = new int[targetNum][3];

                //index, x, y
                for (int i = 0; i < targetNum; i++){
                        for (int j = 0; j < 3; j++){
                                targetLocations[i][j] = 0;
                        }
                }
        }

        public int getIndex(){
                return this.index;
```

```
}

public int getX(){
        return this.x;
}

public void setX(int xCoor){
        this.x = xCoor;
}

public int getY(){
        return this.y;
}

public void setY(int yCoor){
        this.y = yCoor;
}

public int getStatus(){
        return this.status;
}

public void setStatus(int status){
        this.status = status;
}

public int getTargetsFound(){
        return this.targetsFound;
}

public void setTargetsFound(int num){
        this.targetsFound = num;
}

public int getTotalTargets(){
        return this.totalTargets;
}

public int getHeading(){
        return this.heading;
}

public void setHeading(int num){
        this.heading = num;
}
```

```
public boolean getHeadingHome(){
	return this.headingHome;
}

public void setHeadingHome(boolean choice){
	this.headingHome = choice;
}

public int getTimeCount(){
	return this.timeCount;
}

public void setTimeCount(){
	this.timeCount++;
}

public int[][] getTargetLocations(){
	return this.targetLocations;
}

public void setTargetLocations(int[][] targets){
	this.targetLocations = targets;
}

public void updatedTargetsFound(ActualEntity uav1, ActualEntity uav2){
	int uav1num = uav1.getTargetsFound();
	int uav2num = uav2.getTargetsFound();

	if (uav1num != uav2num){
		if (uav1num > uav2num){
			uav2.setTargetsFound(uav1num);
		}
		else{
			uav1.setTargetsFound(uav2num);
		}
	}
}

//method for movement interactions with other UAVs  - need thresholds, and flag if in
range of target
//if prehended uav has range of target flag up change course
//for deciding to prehend other UAVs
public static boolean toPrehendOrNot(ActualEntity uav1, ActualEntity uav2){
	int finalX, finalY;
	int x1 = uav1.getX();
	int y1 = uav1.getY();
```

```
                int x2 = uav2.getX();
                int y2 = uav2.getY();

                if (x1 > x2){
      finalX = x1 - x2;
   } else {
      finalX = x2 - x1;
   }

   if (y1 > y2){
      finalY = y1 - y2;
   } else {
      finalY = y2 - y1;
   }

   if (finalX < OUTERTHRESHOLD && finalX > INNERTHRESHOLD && finalY <
   OUTERTHRESHOLD && finalY > INNERTHRESHOLD){
      return true;
   }
   else{
      return false;
   }
      }

      public static int moveX(int oldX, int newX) {
   if (oldX > newX) {
      return 1;
   } else {
      return 2;
   }
}

public static int moveY(int oldY, int newY) {
   if (oldY > newY) {
      return 1;
   } else {
      return 2;
   }
}
//for deciding to prehend an eternal object
public static boolean toPrehendEternalObjectOrNot(ActualEntity uav, EternalObject eo){
                int finalX, finalY;
                int x1 = uav.getX();
                int y1 = uav.getY();
                int x2 = eo.getX();
                int y2 = eo.getY();
```

```
                if (x1 > x2){
        finalX = x1 - x2;
    } else {
        finalX = x2 - x1;
    }

    if (y1 > y2){
        finalY = y1 - y2;
    } else {
        finalY = y2 - y1;
    }

    if (finalX < EOOUTERTHRESHOLD && finalX > EOINNERTHRESHOLD && finalY <
    EOOUTERTHRESHOLD && finalY > EOINNERTHRESHOLD){
        return true;
    }
    else{
        return false;
    }
        }


public static void prehend(ActualEntity[] theCOPE, EnvironmentSimulator simulator,
EternalObject[] targets) throws InterruptedException{
        int index = 0;
        int x1 = 0;
        int y1 = 0;
        int x2 = 0;
        int y2 = 0;
        int x, y;
        int headingIncreaseCounter, headingDecreaseCounter, currHeading;
        int b1, b2, b3, b4;
        ActualEntity uav1, uav2;
        EternalObject eo;

        for (int i = 0; i < theCOPE.length; i++){
                uav1 = theCOPE[i];
                if (uav1.getStatus() == ACTIVE){
                        x1 = uav1.getX();
                        y1 = uav1.getY();
                        x = x1;
                        y = y1;
                        headingDecreaseCounter = 0;  //to decide which way to move heading
                        headingIncreaseCounter = 0;
                        b1 = 0;
```

```
                        b2 = 0;
                        b3 = 0;
                        b4 = 0;
                        currHeading = uav1.getHeading();

                        //if heading home aim for 0,0 and stop prehending
                        if (uav1.getHeadingHome()){
                                if ((currHeading < 10 && currHeading > 0) || (currHeading ==
                                12)){
                                        currHeading = 10;
                                }
                        }
                        else{
                        //else prehend other Actual Entities and Eternal Objects
                                for (int j = 0; j < theCOPE.length; j++){
                                        if (i != j && theCOPE[j].getStatus() == ACTIVE){
                                                uav2 = theCOPE[j];
                                                x2 = uav2.getX();
                                                y2 = uav2.getY();

                                                if (toPrehendOrNot(uav1, uav2)){

                                                //if prehended heading home
                                                        if(uav2.getHeadingHome()){
                                                                //copy over locations and total target
                                                                num

        uav1.setTargetsFound(uav2.getTargetsFound());

        uav1.setTargetLocations(uav2.getTargetLocations());
                                                                //set heading home on prehending
                                                                uav1.setHeadingHome(true);
                        System.out.println("HEADING HOME");
                        Thread.sleep(500);
                                                                //decide on movement
                                                                currHeading = 10;
                                                        }
                                                        //else
                                                        else {
                                                                //compare targets found and update
                                                                // through arrays comparing each
row
                                                                int found = 0;
                                                                int[][] uav1Locations =
uav1.getTargetLocations();
```

```
                                                    int[][] uav2Locations =
uav2.getTargetLocations();
                                                    for (int m = 0; m <
uav1Locations.length; m++){
                                                            for (int n = 0; n < 3; n++){
                                                                    if
(uav1Locations[m][n] != uav2Locations[m][n]){
        if(uav1Locations[m][n] == 0){
        uav1Locations[m][n] = uav2Locations[m][n];
                                                                            }
                                                                    }
                                                                    if (n == 2){
                                                                            if
(uav1Locations[m][n] != 0){
        found++;
                                                                            }
                                                                    }
                                                            }
                                                    }
        uav1.setTargetLocations(uav1Locations);
                                                    uav1.setTargetsFound(found);

                        if(uav1.getTargetsFound() == uav1.getTotalTargets()){
                           uav1.setHeadingHome(true);
                           System.out.println("HEADING HOME");
                           Thread.sleep(1000);
                        }

                                                    //compare headings
                                                    // if same direction turn prehending
one turn away  compare headings in range of +1 -1
                                                    if (currHeading == 12){
                                                            if ((currHeading - 1 ==
uav2.getHeading()) || (uav2.getHeading() == currHeading) || (uav2.getHeading() == 1)){
                                                                    //figure out what
sector prehended is in if prehending is 0,0
                                                                    if ((x1 < x2) && (y1
< y2)){  //quad 1
                                                                            if ((10 <=
currHeading) && (currHeading <= 12)){
        headingDecreaseCounter++;
```

```
                                                                    }
                                                                    else if (((4 <=
currHeading) && (currHeading <= 6)) || ((1 <= currHeading) && (currHeading <= 3))){

        headingIncreaseCounter++;
                                                                    }
                                                                }
                                                                else if ((x1 < x2) &&
(y1 > y2)){ //quad 2
                                                                    if ((1 <=
currHeading) && (currHeading <= 3)){

        headingDecreaseCounter++;
                                                                    }
                                                                    else if (((4 <=
currHeading) && (currHeading <= 6)) || ((7 <= currHeading) && (currHeading <= 9))){

        headingIncreaseCounter++;
                                                                    }
                                                                }
                                                                else if ((x1 > x2) &&
(y2 < y1)){ //quad 3
                                                                    if ((4 <=
currHeading) && (currHeading <= 6)){

        headingDecreaseCounter++;
                                                                    }
                                                                    else if (((10
<= currHeading) && (currHeading <= 12)) || ((7 <= currHeading) && (currHeading <= 9))){

        headingIncreaseCounter++;
                                                                    }
                                                                }
                                                                else{ //quad 4
                                                                    if ((7 <=
currHeading) && (currHeading <= 9)){

        headingDecreaseCounter++;
                                                                    }
                                                                    else if (((1 <=
currHeading) && (currHeading <= 3)) || ((10 <= currHeading) && (currHeading <= 12))){

        headingIncreaseCounter++;
                                                                    }
                                                                }
                                                            }
```

```
                                                        }
                                                        else if (currHeading == 1){
                                                                if ((currHeading + 1 ==
uav2.getHeading()) || (uav2.getHeading() == currHeading) || (uav2.getHeading() == 12)){
                                                                        //figure out what
sector prehended is in if prehending is 0,0
                                                                        if ((x1 < x2) && (y1
< y2)){  //quad 1
                                                                                if ((10 <=
currHeading) && (currHeading <= 12)){

        headingDecreaseCounter++;
                                                                                }
                                                                                else if (((4 <=
currHeading) && (currHeading <= 6)) || ((1 <= currHeading) && (currHeading <= 3))){

        headingIncreaseCounter++;
                                                                                }
                                                                        }
                                                                        else if ((x1 < x2) &&
(y1 > y2)){ //quad 2
                                                                                if ((1 <=
currHeading) && (currHeading <= 3)){

        headingDecreaseCounter++;
                                                                                }
                                                                                else if (((4 <=
currHeading) && (currHeading <= 6)) || ((7 <= currHeading) && (currHeading <= 9))){

        headingIncreaseCounter++;
                                                                                }
                                                                        }
                                                                        else if ((x1 > x2) &&
(y2 < y1)){ //quad 3
                                                                                if ((4 <=
currHeading) && (currHeading <= 6)){

        headingDecreaseCounter++;
                                                                                }
                                                                                else if (((10
<= currHeading) && (currHeading <= 12)) || ((7 <= currHeading) && (currHeading <= 9))){

        headingIncreaseCounter++;
                                                                                }
                                                                        }
                                                                        else{ //quad 4
```

```
                                                                    if ((7 <=
currHeading) && (currHeading <= 9)){

        headingDecreaseCounter++;
                                                                    }
                                                                    else if (((1 <=
currHeading) && (currHeading <= 3)) || ((10 <= currHeading) && (currHeading <= 12))){

        headingIncreaseCounter++;
                                                                    }
                                                                }
                                                            }
                                                        else if ((currHeading - 1 <=
uav2.getHeading()) && (uav2.getHeading() <= currHeading + 1)){
                                                            //figure out what sector
prehended is in if prehending is 0,0
                                                            if ((x1 < x2) && (y1 < y2)){
//quad 1
                                                                if ((10 <=
currHeading) && (currHeading <= 12)){

        headingDecreaseCounter++;
                                                                }
                                                                else if (((4 <=
currHeading) && (currHeading <= 6)) || ((1 <= currHeading) && (currHeading <= 3))){

        headingIncreaseCounter++;
                                                                }
                                                            }
                                                            else if ((x1 < x2) && (y1 >
y2)){ //quad 2
                                                                if ((1 <= currHeading)
&& (currHeading <= 3)){

        headingDecreaseCounter++;
                                                                }
                                                                    else if (((4 <=
currHeading) && (currHeading <= 6)) || ((7 <= currHeading) && (currHeading <= 9))){

        headingIncreaseCounter++;
                                                                    }
                                                                }
                                                            else if ((x1 > x2) && (y2 <
y1)){ //quad 3
```

```
                                                                    if ((4 <= currHeading)
&& (currHeading <= 6)){

        headingDecreaseCounter++;
                                                                    }
                                                                    else if (((10 <=
currHeading) && (currHeading <= 12)) || ((7 <= currHeading) && (currHeading <= 9))){

        headingIncreaseCounter++;
                                                                    }
                                                            }
                                                            else{ //quad 4
                                                                    if ((7 <= currHeading)
&& (currHeading <= 9)){

        headingDecreaseCounter++;
                                                                    }
                                                                    else if (((1 <=
currHeading) && (currHeading <= 3)) || ((10 <= currHeading) && (currHeading <= 12))){

        headingIncreaseCounter++;
                                                                    }
                                                            }
                                                    }
                                    }
                                            } // close to prehend or not
                                    } //close if active
                                    //if heading home break
                                    if (uav1.getHeadingHome()){
                                            break;
                                    }
                            } //close cope prehending


                            //if not heading home continue
                            if (!uav1.getHeadingHome()){
                                    //prehend Eternal Objects  //change threshold
                                    for (int k = 0; k < targets.length; k++){
                                            eo = targets[k];
                                            x2 = eo.getX();
                                            y2 = eo.getY();
                                            index = eo.getIndex();

                                            if (toPrehendEternalObjectOrNot(uav1, eo)){

                                                    //compare to found
```

```
                                                    int[][] uav1Locations =
uav1.getTargetLocations();
                        if (uav1Locations[index][1] == 0 || uav1Locations[index][2] == 0){

        uav1.setTargetsFound(uav1.getTargetsFound() + 1);
                                                                uav1Locations[index][0] = index;
                                                                uav1Locations[index][1] = x2;
                                                                uav1Locations[index][2] = y2;

        uav1.setTargetLocations(uav1Locations);
                                                                System.out.println(uav1.getIndex() +
" TARGET FOUND " + index + " from " + x1 + " " + y1);
                            Thread.sleep(1000);
                            //set heading home
                            if(uav1.getTargetsFound() == uav1.getTotalTargets()){
                               uav1.setHeadingHome(true);
                               System.out.println("HEADING HOME");
                               Thread.sleep(1000);
                            }
                                                            }
                                                        }
                        //if heading home break
                        if (uav1.getHeadingHome()){
                           break;
                        }
                                            }
                                        }
                            }// close else for prehensions
                    // if heading toward boundary and 15 away start turning away
                    if ((x1 < 20) || (x1 > (SIZE - 20)) || (y1 < 20) || (y1 > (SIZE - 20))){
                       //check corners
                       if (x1 < 20 && y1 < 20 && (currHeading == 9 || currHeading == 10 || currHeading
== 11 || currHeading == 12)){
                          currHeading = 4;
                          b1++;
                       }
                       else if (x1 > 680 && y1 < 20 && (currHeading == 12 || currHeading == 1 ||
currHeading == 2 || currHeading == 3)){
                          currHeading = 7;
                          b1++;
                       }
                       else if (x1 > 680 && y1 > 680 && (currHeading == 3 || currHeading == 4 ||
currHeading == 5 || currHeading == 6)){
                          currHeading = 10;
                          b1++;
                       }
```

```
            else if (x1 < 20 && y1 > 680 && (currHeading == 6 || currHeading == 7 ||
currHeading == 8 || currHeading == 9)){
                currHeading = 1;
                b1++;
            }

            //checks sides
            else if (x1 < 20 && (currHeading == 8 || currHeading == 9 || currHeading == 10)){
                currHeading = 2;
                b1++;
            }
            else if (x1 > 680 && (currHeading == 2 || currHeading == 3 || currHeading == 4)){
                currHeading = 8;
                b1++;
            }
            else if (y1 < 20 && (currHeading == 11 || currHeading == 12 || currHeading == 1)){
                currHeading = 5;
                b1++;
            }
            else if (y1 > 680 && (currHeading == 5 || currHeading == 6 || currHeading == 7)){
                currHeading = 11;
                b1++;
            }
        }
                    //decide new heading and increment location
        if(b1 > 0){
            //heading already changed
        }
                    else if (headingIncreaseCounter > headingDecreaseCounter){
                        if (currHeading == 12){
                            currHeading = 2;
                        }
            else if (currHeading == 11){
                currHeading = 1;
            }
                        else{
                            currHeading = currHeading + 2;
                        }
                    }
                    else if (headingDecreaseCounter > headingIncreaseCounter){
                        if (currHeading == 1){
                            currHeading = 11;
                        }
            else if (currHeading == 2){
                currHeading = 12;
            }
```

```
                else{
                        currHeading = currHeading - 2;
                }
        }
        uav1.setHeading(currHeading);
        //System.out.println(currHeading);

//x and y is 0,0 in top left
        if (currHeading == 1){
                x1++;
                y1 = y1 - 3;
        }
        else if (currHeading == 2){
                x1 = x1 + 3;
                y1--;
        }
        else if (currHeading == 3){
                x1 = x1 + 3;
        }
        else if (currHeading == 4){
                x1 = x1 + 3;
                y1++;
        }
        else if (currHeading == 5){
                x1++;
                y1 = y1 + 3;
        }
        else if (currHeading == 6){
                y1 = y1 + 3;
        }
        else if (currHeading == 7){
                x1--;
                y1 = y1 + 3;
        }
        else if (currHeading == 8){
                x1 = x1 - 3;
                y1++;
        }
        else if (currHeading == 9){
                x1 = x1 - 3;
        }
        else if (currHeading == 10){
                x1 = x1 - 3;
                y1--;
        }
        else if (currHeading == 11){
```

```
                        x1--;
                        y1 = y1 - 3;
                    }
                    else{
                        y1 = y1 - 3;
                    }

            //fix if too close to edge
             if(x1 < 15)
               {
                  x1 = 15;
               }
               if( x1 > 685)
               {
                  x1 = 685;
               }
               if(y1 < 15)
               {
                  y1 = 15;
               }
               if( y1 > 685)
               {
                  y1 = 685;
               }
                        uav1.setX(x1);
                        uav1.setY(y1);

                        //inactivate plane if near home
                        if (x1 < 20 && y1 < 20 && uav1.getHeadingHome()){
                            uav1.setStatus(INACTIVE);
                        }
                        //make sure cope array is passed back
                        theCOPE[i] = uav1;
                        COPE.setTheCopeArray(theCOPE);
                        System.out.println(uav1.getIndex() + " " + x1 + " " + y1 + " heading: " +
uav1.getHeading() + " targets found: " + uav1.getTargetsFound() + " heading home: " +
uav1.getHeadingHome());
                    } //close if active
                } //close uav1 loop
            } //close method
        } //close AE
```

Appendix D: Eternal Object Class Code

```
/*****************************************************
 * Author(s)    : Erica Absetz
 * Date         : March 6, 2013
 * File         : EternalObject.java
 * Language     : java
 *****************************************************/
public class EternalObject{

	public int identity; //0 for obstacle, 1 for target
	public int x, y, index;

	public EternalObject(int identity, int index, int xCoor, int yCoor){
		this.identity = identity;
		this.index = index;
		this.x = xCoor;
		this.y = yCoor;
	}

	public int getIdentity(){
		return this.identity;
	}

	public int getIndex(){
		return this.index;
	}

	public int getX(){
		return this.x;
	}

	public int getY(){
		return this.y;
	}
}
```