2005-06

Swarm intelligence for autonomous UAV control

Frantz, Natalie R.

Monterey California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SWARM INTELLIGENCE FOR AUTONOMOUS UAV CONTROL

by

Natalie R. Frantz

June 2005

Thesis Advisor:	Phillip E. Pace
Second Reader:	David C. Jenn

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Swarm Intelligence for Autonomous UAV Control			**5. FUNDING NUMBERS**
6. AUTHOR(S) Natalie R. Frantz			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Unmanned Aerial Vehicles (UAVs) are becoming vital warfare platforms because they significantly reduce the risk of human life while accomplishing important missions. A UAV can be used for example, as stand-in sensor for the detection of mobile, low-probability-of-intercept battlefield surveillance and fire control emitters. With many UAVs acting together as a swarm, the location and frequency characteristics of each emitter can be accurately determined to continuously provide complete battlefield awareness. The swarm should be able to act autonomously while searching for targets and relaying the information to all swarm members. In this thesis, two methods of autonomous control of a UAV swarm were investigated. The first method investigated was the Particle Swarm Optimization (PSO) algorithm. This technique uses a non-linear approach to minimize the error between the location of each particle and the target by accelerating particles through the search space until the target is found. When applied to a swarm of UAVs, the PSO algorithm did not produce the desired performance results. The second method used a linear algorithm to determine the correct heading and maneuver the swarm toward the target at a constant velocity. This thesis shows that the second approach is more practical to a UAV swarm. New results are shown to demonstrate the application of the algorithm to the swarm movement.

14. SUBJECT TERMS Autonomous Behaviors, Unmanned Aerial Vehicles (UAVs), Particle Swarm Optimization (PSO)			**15. NUMBER OF PAGES** 132
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SWARM INTELLIGENCE FOR AUTONOMOUS UAV CONTROL

Natalie R. Frantz
Ensign, United States Navy
B.S., United States Naval Academy, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2005

Author: Natalie R. Frantz

Approved by: Phillip E. Pace
Thesis Advisor

David C. Jenn
Second Reader

John P. Powers
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned Aerial Vehicles (UAVs) are becoming vital warfare platforms because they significantly reduce the risk of human life while accomplishing important missions. A UAV can be used for example, as stand-in sensor for the detection of mobile, low-probability-of-intercept battlefield surveillance and fire control emitters. With many UAVs acting together as a swarm, the location and frequency characteristics of each emitter can be accurately determined to continuously provide complete battlefield awareness. The swarm should be able to act autonomously while searching for targets and relaying the information to all swarm members. In this thesis, two methods of autonomous control of a UAV swarm were investigated. The first method investigated was the Particle Swarm Optimization (PSO) algorithm. This technique uses a non-linear approach to minimize the error between the location of each particle and the target by accelerating particles through the search space until the target is found. When applied to a swarm of UAVs, the PSO algorithm did not produce the desired performance results. The second method used a linear algorithm to determine the correct heading and maneuver the swarm toward the target at a constant velocity. This thesis shows that the second approach is more practical to a UAV swarm. New results are shown to demonstrate the application of the algorithm to the swarm movement.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This work was supported in part by the Tactical Electronic Warfare Division, Naval Research Laboratory, Code NRL 5700, and the Office of Naval Research, Code ONR 313.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The Unmanned Air Vehicle (UAV) can be applied to more military missions than ever before. The UAV is perfect for military missions that are long and arduous on a crew such as surveillance and reconnaissance. By carrying a small selection of weapons, a UAV can perform the same functions as a piloted plane and destroy a target. The advantage is that the UAV eliminates the potential loss of human life since it can be sent into highly dangerous areas. Previously these areas were avoided because the risk of human life outweighed the potential gains from such a flight. The success of the UAV has encouraged more research and ideas to maximize the advantages of having an unmanned platform. The next generation of UAVs will be smaller and part of a collaborative group. This group will be able to autonomously control their own movements and react to the environment.[1]

A group of UAVs is more capable than a single UAV. The UAVs can divide the workload among the group. The individual UAVs can be equipped for different functions in the mission whether it be surveillance and reconnaissance, strike, or battle damage assessment. Surveillance missions can be completed quickly by covering more search area when the group is spread out. They also offer redundancy to ensure that the mission is completed. If one UAV is destroyed by the enemy or drops out because of mechanical failure, the rest of the group will fill in and carry out the mission. By having more than one UAV assigned to a mission, the probability of success dramatically increases.[1]

The UAV's airframe will be designed and fitted with the appropriate technology to carry out the given mission. The initial design aspect is the software required to program the UAVs to act autonomously. Responding to a central command structure with human control is relatively simple. When given a command, the UAV reacts according. Without human control, the group will need to gather data from the environment, interpret the data, and take appropriate actions to continue on the mission. The UAVs will need to communicate with each other to share information in order to decide when the

mission is completed. Behavior algorithms are necessary in order to act in a decentralized manner and self-organize to complete the mission. The concept of a group of UAVs under autonomous control closely resembles the ideas of swarms and swarm intelligence, which is similar to the concept used by insects and birds. [1]

The swarm must have a realistic and practical method for completing the required mission of finding and 'attacking' a target. Two autonomous UAV control methods were analyzed: the Particle Swarm Optimization (PSO) algorithm and a linear control method. The PSO algorithm was simulated using MATLAB. The actual program is training a neural network to solve a problem, but the concept is analogous to UAVs searching for a target.

Through flock simulation and the derivation of PSO, scientists discovered that a synchronous flock is not essential.[2] The simulated synchronized flock limits the scope of the group because it does not allow of individual exploration of the area. The flock has to tightly travel together; so in order to search the area thoroughly, the entire flock would have to go over all possible locations. By allowing individuals to travel slightly outside the group, the group covers a larger search area at one time. As they identify individual best found positions thus far, the group is able to discover the target faster. For a group to cooperate and achieve goals such as finding a target, the group must communicate. Therefore, communication, rather than synchronization, is necessary for success.[2]

The current PSO algorithm applies to weightless particles in multiple dimensions. The PSO algorithm can offer the advantage of finding the pattern in almost any problem space to reach a solution, but the current sequence can dead end and restart in a new position. It is a waste of computation time and resources to create an algorithm that would have a swarm of UAVs pursue a direction only to find it is the wrong path. If the target cannot be reached from the current path of the swarm, the PSO algorithm's solution is to start over. The swarm needs more guidance and a process to get out of a dead-end situation and back on track. With further research and improvements, the PSO algorithm can be applied to real objects limited to three dimensions.

PSO is focused on minimizing error between the particles and the target. In addition to changing the particle's direction to head toward the target, the algorithm accelerates the particles. When applying PSO to real flying objects, the constant speed changes are the main drawback. Actual UAVs should maintain a constant velocity to operate in a stable and controlled manner to prevent chaos and collisions. The constant velocity will also increase fuel efficiency and decrease strain on the platform. Although the PSO method is not practical, the central idea of minimizing error is completely applicable to UAVs. When the target location is known, error minimization is a valuable tool.

Since the first method investigated did not perform realistically, a second method using a linear model for UAV movement was investigated. The program used to model the linear method was initially designed by a student at North Dakota State University [3]. The simulation focuses on three simple maneuvers for UAV motion. Although the simulation has limited abilities, the concept is easily applicable to real UAVs missions. The program was modified in this thesis to incorporate more realistic mission scenarios.

Compared to the PSO, the linear algorithm produces the most realistic results. The linear algorithm incorporates the ideas that have performed well in the PSO. The swarm does not have to move synchronously, and the UAVs move toward the target by minimizing the error in their position from the target. The error is minimized in a linear fashion since the velocity of the UAV remains constant. Linearity produces great results, and the simulated UAVs are able to find the target quickly and efficiently. The program also handles the UAVs as objects that occupy space. Each UAV has a threshold boundary distance, so they will avoid each other if they get too close. These movements allow the swarm to move toward a destination in space without collisions.

Since the swarm does not travel in formation, the UAVs need to regroup once a target is found. The orbit stations around the target provide organization before the attack. While orbiting, the UAVs can communicate and coordinate when the attack will occur. The orbit circles are also far enough from possible dangerous areas surrounding the target. The simulation shows the distance to be small relative to the size of the target and UAVs, but the radius of the circle is adjustable. Overall the linear algorithm can be more easily simulated and applied to realistic missions on a larger scale.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. SWARM OF UAVS

The Unmanned Aerial Vehicle (UAV) can be applied to more military missions than ever before. The UAV is perfect for military missions that are long and arduous on a crew such as surveillance and reconnaissance. By carrying a small selection of weapons, a UAV can perform the same functions as a piloted plane and destroy a target. The advantage is that the UAV eliminates the potential loss of human life since it can be sent into highly dangerous areas. Previously these areas were avoided because the risk of human life outweighed the potential gains from such a flight.[1]

The success of the UAV has encouraged more research and ideas to maximize the advantages of having an unmanned platform. The next generation of UAVs will be smaller and part of a collaborative group. This group will be able to autonomously control their own movements and react to the environment. Currently two or more people are required to control a single UAV while it is in flight. The military is hoping to decrease the man-power required to operate the UAV. A UAV controlled by a group of people can complete the mission, but a large number of UAVs would require an even larger group of people. Such a large number of operators will be unpractical and inefficient compared to the idea of autonomous control.[1]

A group of UAVs is more capable than a single UAV. The UAVs can divide the workload among the group. The individual UAVs can be equipped for different functions in the mission whether it be surveillance and reconnaissance, strike, or battle damage assessment. Surveillance missions can be completed quickly by covering more search area when the group is spread out. They also offer redundancy to ensure that the mission is completed. If one UAV is destroyed by the enemy or drops out because of mechanical failure, the rest of the group can fill in and carry out the mission. By having more than one UAV assigned to a mission, the probability of success dramatically increases.[1]

The UAV's airframe is designed and fitted with the appropriate technology to carry out the given mission. The initial design aspect is the software required to program

the UAVs to act autonomously. Responding to a central command structure with human control is relatively simple. When given a command, the UAV reacts according. Without human control, the group will need to gather data from the environment, interpret the data, and take appropriate actions to continue on the mission. The UAVs will need to communicate with each other to share information in order to decide when the mission is completed. Behavior algorithms are necessary in order to act in a decentralized manner and still self-organize to complete the mission. The concept of a group of UAVs under autonomous control closely resembles the ideas of swarms and swarm intelligence, which is similar to the concept used by insects and birds.[1]

B. PRINCIPAL CONTRIBUTIONS

The goal of this thesis was to create a detailed simulation of a swarm of UAVs that has autonomous control. The swarm must have a realistic and practical method for completing the required mission of finding and 'attacking' a target. The attack will be simulated by occupying the same point location as the target. Whether each UAV launches a missile or takes pictures while over the target is dependent upon the given mission.

Two autonomous UAV control methods were analyzed: the Particle Swarm Optimization (PSO) algorithm and a linear control method. The PSO algorithm was simulated using MATLAB. The actual program is training a neural network to solve a problem, but the concept is analogous to UAVs searching for a target.

Since the first method investigated did not perform realistically, a second method using a linear model for UAV movement is investigated. The program used to model the linear method was initially designed by a student at North Dakota State University [3]. The simulation focuses on three simple maneuvers for UAV motion. Although the simulation has limited abilities, the concept is easily applicable to real UAVs missions. The program is modified in this thesis to incorporate more realistic mission scenarios.

C. THESIS OUTLINE

Chapter II provides a background for swarms and swarm intelligence. The swarm algorithms that have emerged are briefly described. Since swarm algorithms are an efficient method for training neural networks, the concept of neural networks is briefly reviewed.

Chapter III describes the concept of PSO from theories of flocks of birds to training neural networks to applications for a swarm of UAVs. A MATLAB program simulates a particle swarm using the PSO concept, and the PSO algorithm is compared to the backpropagation algorithm for training a neural network to solve an XOR problem.[4]

Chapter IV describes a linear control approach to organizing a swarm of UAVs. The linear concept is displayed in a program demonstrating a swarm of UAVs attacking a target.

Chapter V summarizes the practicality of using PSO and a linear algorithm for UAV control along with future work with a UAV swarm.

Appendix A describes the PSO function in MATLAB, created by Brian Birge in [4]. Birge also creates a 'demotrain' file to use and illustrate the capabilities of the new PSO function while comparing PSO to backpropagation.

Appendix B demonstrates a neural network backpropagation example through the 'demotrain' file.

Appendix C demonstrates a neural network PSO example through the 'demotrain' file.

Appendix D demonstrates the linear approach program created by Chin Lua from North Dakota State University.[3]

Appendix E demonstrates a modified program from [3].

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. SWARMS

1. Origin

Gerado Beni began using the term 'swarm' to describe his work with cellular robots. It was casually mentioned to him by Alex Meystel at a conference, and Beni found that 'swarm' accurately described cellular automata.[5] The group of cells in robots exhibit similar characteristics found in biological swarms, like insects, such as decentralization, no synchronization, and simplicity in the members. Swarm correctly describes a robotic system, but while roboticists focus on performing tasks with the swarm, biologists take the swarm concept and analyze the social behavior of insects as they perform a function.[5]

2. Self-Organization

Biologists investigate the idea that the swarm will identify a pattern in their system and self organize to find the optimal means of reaching the goal. Roboticists are trying to create patterns in the behavior so that the cells will self-organize. Communication is a key parameter in allowing the members to interact and self-organize their tasks. The characteristics of a swarm are extended from decentralization, no synchronization, and simplicity to include communication among the members.[5] In real swarms, such as bees and ants, members of a group can use direct communication with each other or indirect communication, referred to as stigmergy, to interact through the environment.[6]

a. Positive and Negative Feedback

Bees and ants are constantly pursuing food sources for their colonies to sustain their existence, therefore, location of a food source is a common example of social behavior and easy to demonstrate through experimentation. Their ability to establish paths to a food source illustrate one level of self organization.[6]

Bees directly communicate the location of a food source through dancing. A bee's dance will show the distance and direction of the food source to the others. When one food source is superior in quality, the bees take advantage of the better source as more bees dance to indicate that location. Experiments indicate that differences in the rate of dancing and abandonment create positive feedback and cause more bees to follow

the best quality path. Not all bees will congregate to the superior food source. A small population will continue to go to the alternate food source or elsewhere as a response to negative feedback in the system. Negative feedback is generated through saturation, exhaustion, crowding, and competition at the food source. These realistic limits stabilize the system so all bees are not on one track. [6]

Ants are most well known for communicating through the environment with pheromones to indicate the quality of a path. These pheromones provide a positive feedback method for the colony to reinforce the trail. The constant amplification of the trail persuades the other ants to continue along the same path, providing positive feedback. Negative feedback is introduced through the same general complications as bees: saturation, exhaustion, crowding, and competition at the food source.[6]

b. Randomness

Both bees and ants rely on the randomness of individuals in the group. It may seem counterintuitive to believe that self-organization is created among randomness, but randomness allows the introduction of new ideas into the group. It can provide simply new paths to a food source or more general new methods and solutions that allow for growth of the colony. Randomness is also a source of optimization. For example, two food sources that are identical in quality and equal distance from the bee hive should be utilized symmetrically. Experimentally, the deviation of a few bees will cause a swing to one source because those few bees recruit more bees and those continue to recruit even more. The same principle applies to ants. As more ants go along a path the pheromone strength becomes greater. One path is amplified and becomes the optimal path.[6]

Multiple interactions occur throughout the group, causing more actions and reactions. System characteristics of positive feedback, negative feedback, and randomness provide the balance needed to keep the group responsive to an ever-changing environment.[6]

3. Swarm Intelligence

The self organization of the group into ordered patterns is an intelligent characteristic. For a swarm to form ordered patterns, it needs to 'analyze' patterns while finding the optimal method. This characteristic could allow the swarm to have 'intelligence.'[5]

Beni struggled with the definition of swarm intelligence since the word intelligence has also been so loosely used. One preliminary definition of an intelligent swarm according to Beni is "a group of 'machines' capable of 'unpredictable' material computation." He has to revisit the definition of a machine as "an entity capable of mechanical behavior, i.e., of transferring and/or processing matter/energy." Unpredictability is also difficult to define, but Beni links it to the computational power of the system. He is looking for "a system (the intelligent swarm) which cannot be predicted in the time it takes to form a new material pattern (of its own components)." In his paper clarifying definitions related to swarms, he finally settles on "*Intelligent swarm*: a group of non-intelligent robots ('machines') capable of universal material computation."[5]

Hundreds and even thousands of non-intelligent machines can comprise an intelligent swarm. There are advantages of having simple components in a group over having complex centralized components. Through self-organization and pattern identification, the individual machines working together as a swarm can accomplish tasks that could not have been possible by a single machine. Logistically, the individual members of the swarm are easier to design and build, so these simple components potentially can be cheaply replaced, interchanged, or disposed of.[5]

The unpredictable function of a swarm comes from the method that it 'learns.' The concept of universal material computation allows for the creativity of the designer. There are numerous algorithms in existence allow the swarm to compute a possible process to complete various tasks.

B. SWARM ALGORITHMS

1. Ant Colony Optimization

The behavior of ants within a colony inspired experiments and eventually algorithms to mimic the ants. The fundamental task of an ant colony is to find food sources. While performing this task, the ants are able to find the shortest path to that food source. This natural optimization is tested by Deneubourg through the bridge experiments.[6] The binary bridge experiment consists of two paths of equal length from their nest to the food source, so initially all ants choose a random path. As they continue to choose at random, ants travel on one path, the pheromone intensity increases on that path. As few more ants break the 50-50 chance, they attract more ants, until the majority is on one

path, such as explained for randomness. Two equal length paths can be branched out to a longer path and a shorter path. Deneubourg also created a bridge from the nest to the food with two longer branches, shown in Figure 1.

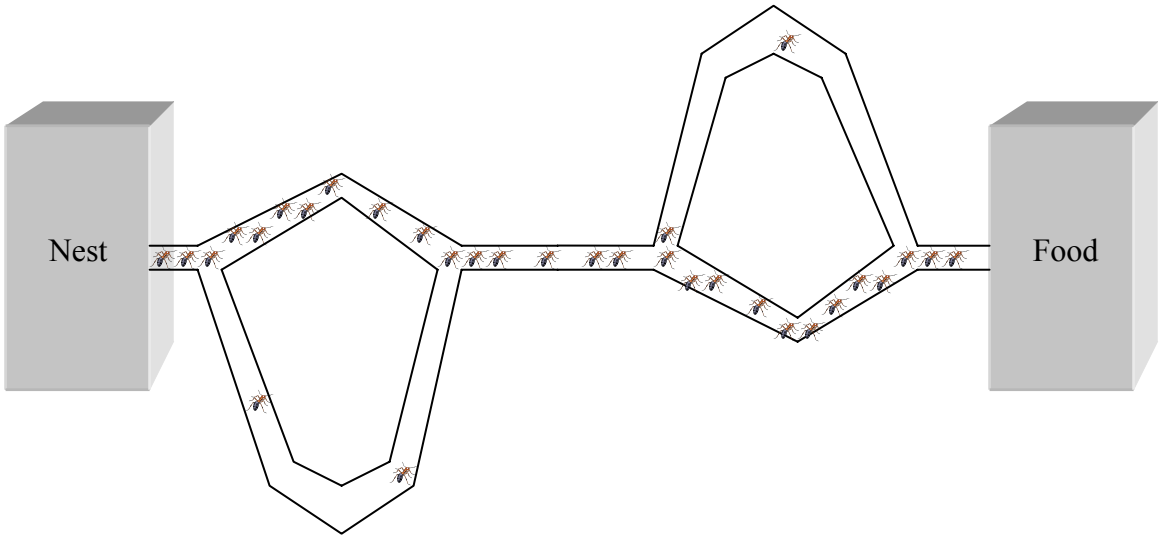


Figure 1. Bridge Experiment

Again, all paths are initially chosen at random. The ants that take the shortest path to the food source and back to the nest obviously make it back to the nest first. Their route is twice as intensified by pheromones than other options because the other ants on the longer paths have only passed over their route once. This initial difference is amplified until the majority of the ants are on the shortest path.[6]

The ant colony optimization concept stimulated numerous more experiments and research. A few algorithms such as the Traveling Salesman Problem (TSP), Ant System (AS), Ant Colony System (ACS), and AntNet can be found in [6]. Overall the algorithms have limited success with performance for the problems originally intended. They are instead applied to combinatorial optimization, communications network routing, and packet-switching communications networks. Since they offer a more promising future in this area, other directions were taken for swarm theories.[6]

2. Evolutionary Computation

Genetic and evolutionary algorithms are two examples of algorithms that use a population set to evolve to a solution. These algorithms are similar in concept but executed differently. The differences are becoming undecipherable in most cases, and ge-

netic and evolutionary algorithms are almost interchangeable in meaning. Both mimic natural evolution by using survival of the fittest and allowing manipulation of the population. Each individual in the population is assigned a fitness value based on the problem. By collecting the individual with higher fitness, the population will progress toward the solution to a given problem. The difference lies in the reproduction, crossover, and mutation process. The fitness values for the problems indicate where they are in a problem 'space.' They represent how 'close' the individual is to the goal, so the population is 'searching' the problem space for the solution.[6]

a. Genetic Algorithm

The genetic algorithm follows the general pattern of initializing the population, calculating each individual's fitness in the population, reproducing the selected individual to create a new population, imposing crossovers and mutations on the populations, and repeating the process over again until the desired population is reached. The population size is typically between 20 and 200 since it directly affects the computation time. Larger populations can search more of the entire solution 'space,' but the computational cost is too great. The initial population can be randomly chosen or contain a few 'seeded' individuals with selected 'traits.' The deserved initial population should cover a wide variety of 'traits' to avoid limiting the algorithm from the start.[7]

Calculating the fitness function of each individual can be a complex process, but the idea is simply to sort out the best individuals that satisfy the solution. Various processes for calculating the fitness function exist. Each individual is assigned a fraction of a roulette wheel to correspond with the fitness value. The fraction on the roulette wheel indicates the probability of an individual being selected. There are also numerous variations of the probability assignment procedure. The basic idea remains the same: a higher level of fitness has better chance of being chosen on the wheel. For example, individual A has a fitness value of 0.4 and individual B 1.2. Individual B will occupy three times more space on the roulette wheel and is three time more likely to be selected than individual A. Once all the individuals have been chosen, they proceed to the main step of genetic theory.[7]

The population experiences crossover to model the results of sexual reproduction. The probability of crossover and type of crossover are specified for each prob-

lem. The probability typically ranges for 0.6 to 0.8, causing 60 to 80 percent of the population to experience crossover. The basic crossover type is one-point and easily described using binary bits. Take two individuals with the following characteristics:

11010111
01010001.

They experience crossover at a randomly chosen point indicated by the vertical line below:

11010|111
01010|001.

The bits to the right of the vertical line will be exchanged. The two resulting individuals after crossover are:

11010001
01010111.

Mutation is introduced after crossover. Mutation has a much lower probability of occurrence, generally down to 0.001. Mutation simply involved the flipping of a random bit. Since the probability is extremely low, it may only occur to one individual in the entire population each generation.[7]

b. Evolutionary Algorithm

Evolutionary approaches focus on frequent mutations to change the population rather than genetic recombination. The general pattern for an evolutionary algorithm is initializing the population, exposing the population to the environment, calculating each individual's fitness in the population, mutating individuals at random, recombining to create a child population, reevaluating the entire population of parents and children, selecting individuals to create a new population, and repeating the process over again until the desired population is reached. Mutating the parent population before the reproduction phase is similar to individuals being altered by the environment growing up in life. This early mutation can be more successful than mutating the child population. The individuals also recombine in a specified manner in the algorithm rather than swapping randomly selected bits. The child then becomes a combination of both parents, even with their mutations. Overall, both genetic and evolutionary algorithms are similar and help inspire new ideas for modeling social behavior.[7]

C. NEURAL NETWORKS

1. Basics

An Artificial Neural Network (ANN), or commonly referred to as simply a Neural Network (NN), is modeled after neurons and synapse connections in the brain. The design of a neural network will help simulate artificial behavior by establishing patterns when exposed to a situation. The multi-layer perceptron neural network has proven to be an excellent approximator of most non-linear functions. This neural network entwines three key components: an input layer, one or more 'hidden' layers, and an output layer.[8] Figure 2 illustrates how the elements are connected.

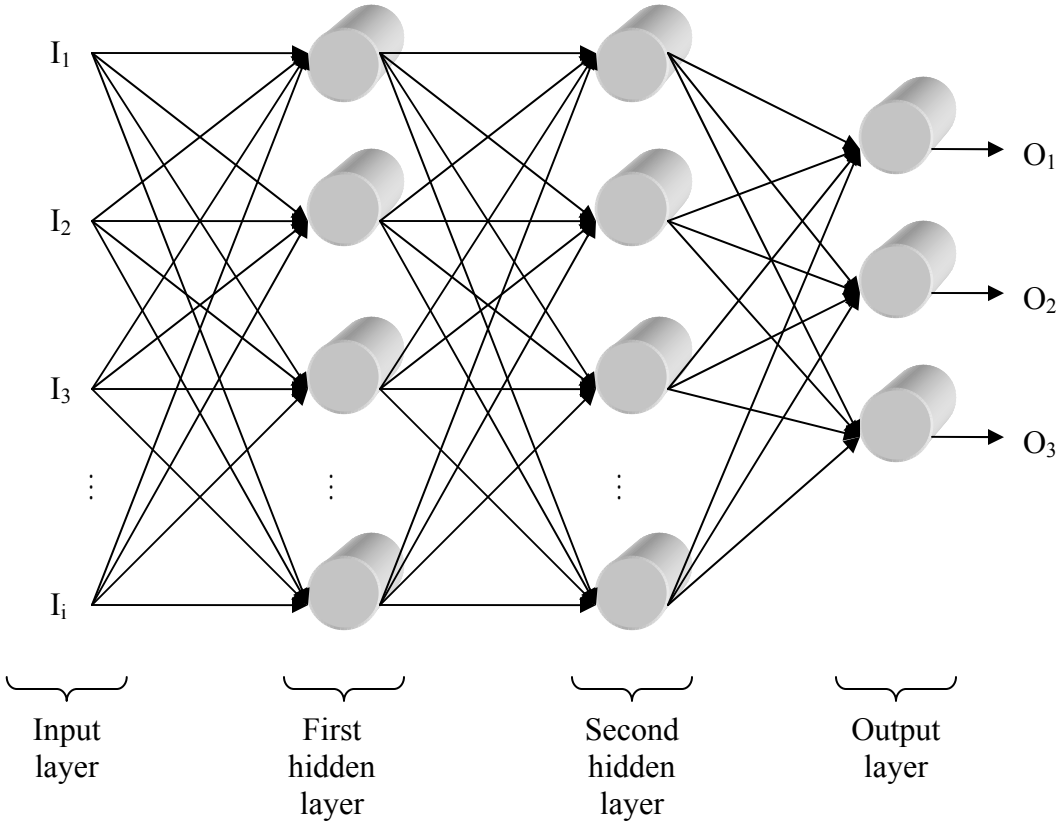


Figure 2. Multilayer Perceptron Neural Network Architecture with Two Hidden Layers

The circles represent the 'neurons,' or nodes, which are individual perceptions. The inputs range from 1 to i, the first hidden layer nodes from 1 to j, second hidden layer nodes from 1 to k, and so forth. The arrows represent the weighted connections between neurons. The weights in the first hidden layer will be referred to as $w_{i,j}$, to represent the weight from the i-th input to the j-th node of the first hidden layer. The weight values determined the established pattern of the neural network. The output response is dependent upon the weighted connections.[8]

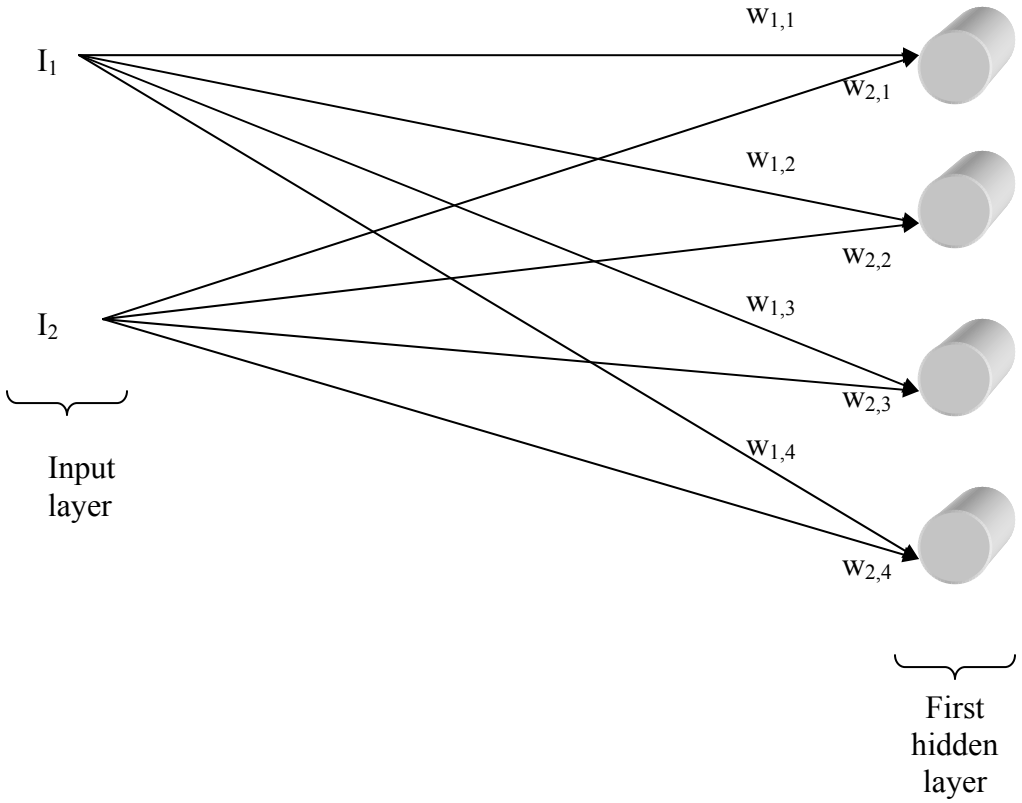


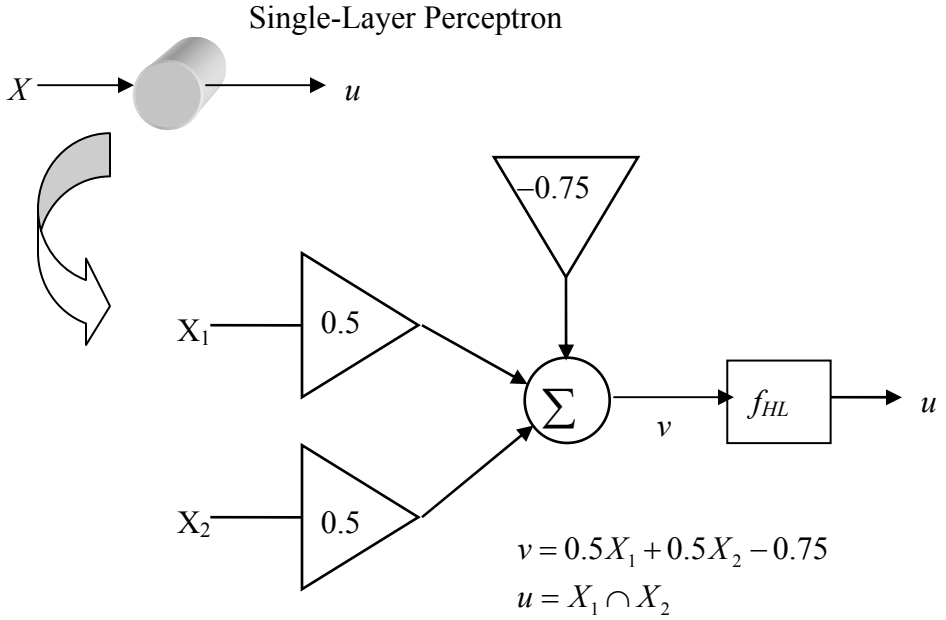
Figure 3. Illustration of Weight Connections for Two Inputs

a. Single-Layer Perceptrons

A single-layer perceptron has the ability to handle linear functions and to act as binary logic such as AND, OR, or NOT, demonstrated in Figures 4 through 6. These binary logic units are evaluated using a hard-limited non-linear activation function:

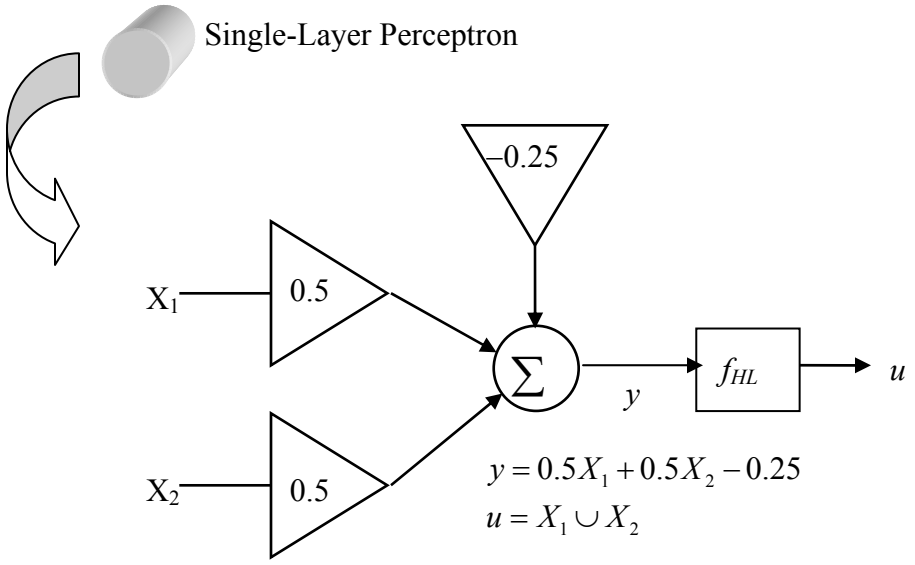
$$f_{HL} = \begin{cases} 1 & v > 0 \\ 0 & v \le 0 \end{cases}. \tag{1}$$

The connection from the input to the perceptron is given a weight, and the perceptron can also have a bias value to satisfy the desired pattern. The values inside the triangles represent the values of the weighted connections.[8]



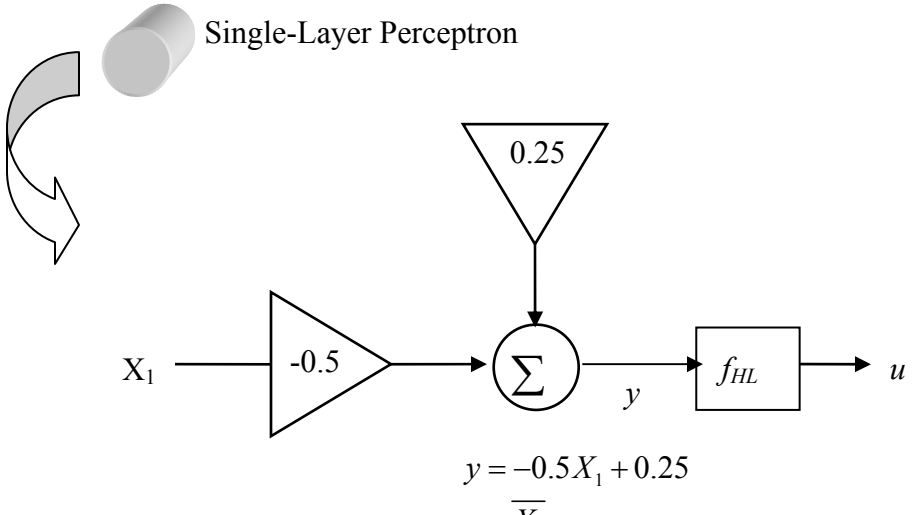
*X*1	*X*2	*v*	*u*
0	0	-0.75	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.25	1

Figure 4. Single-Layer Perceptron as an AND Binary Logic Unit



X1	X2	v	u
0	0	-0.25	0
0	1	0.25	1
1	0	0.25	1
1	1	0.75	1

Figure 5. Single-Layer Perceptron as an OR Binary Logic Unit



X1	v	u
0	0.25	1
1	-0.25	0

Figure 6. Single-Layer Perceptron as an NOT Binary Logic Unit

b. Multi-layer Perceptrons

There are three requirements for a multi-layer perceptron neural network. The neural network must have one or more hidden layers, meaning more than simply an input and output layer. The hidden layers are what allow the network to respond to more complex patterns. The second requirement is that the network must have a high degree of connectivity, so each node from one layer will be connected to all the nodes in the following layer.[8]

The third requirement for the perceptrons in a multi-layer perceptron neural network is that they must use a differential nonlinear activation function. The sigmoid nonlinearity is a differentiable function and also provides values between 0 and 1. The advantage is that the values from 0 to 1 are analogous to probability distribution and provide easier pattern recognition. The sigmoid function is defined as follows:

$$f_{sigmoid}(v) = \frac{1}{1+e^{-\beta v}} \tag{2}$$

where β is the gain.[8] As shown in Figure 7, as β increase, the slope of the sigmoid function becomes steeper.

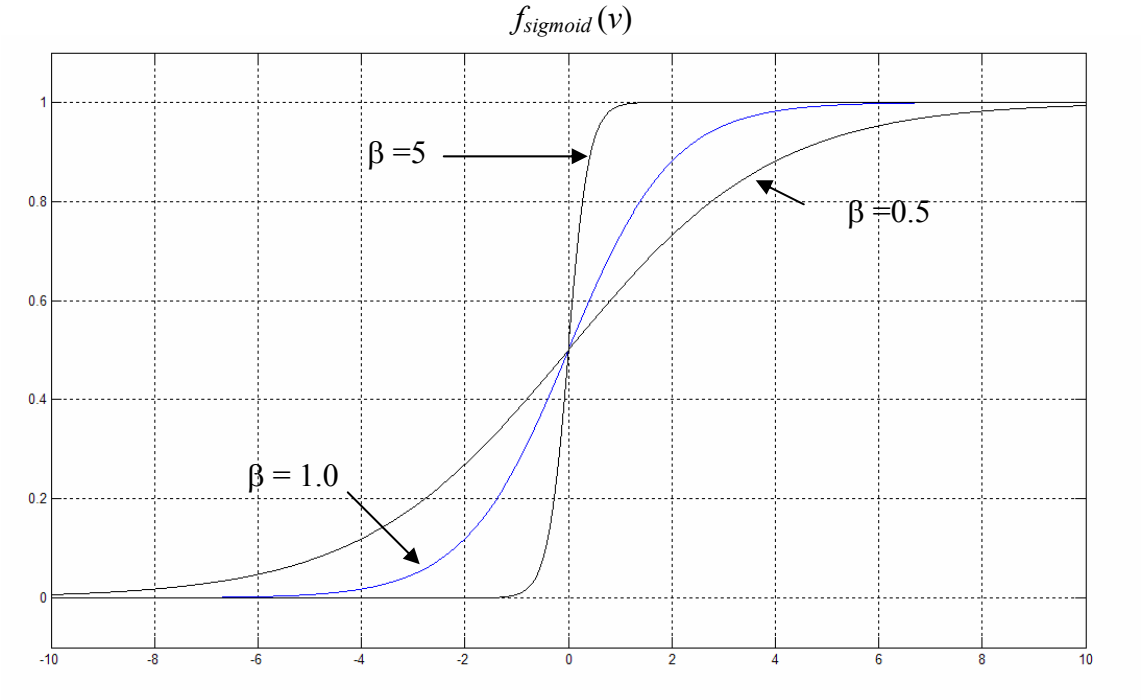


Figure 7. Graph of Sigmoid Nonlinearity Function

c. Exclusive OR (XOR) Problem

The XOR problem is an example of problem that must use a multi-layer perceptron because a nonlinear pattern is required to solve an XOR equation.[8] The two inputs to an XOR equation will produce an output according the values in Table 1.

*X*1	*X*2	*Output*
0	0	0
0	1	1
1	0	1
1	1	0

Table 1. Binary XOR Table

When drawing the unit hypercube, no line exists that would separate correctly (0,0) and (1,1) from (0,1) and (1,0), therefore XOR requires a nonlinear solution. A single-layer perceptron has a linear decision boundary, so it cannot be used to solve this problem. Touretzky and Pomerleau designed the XOR solution in 1989 to have one hidden layer with two nodes.[8] Figure 8 represents the architecture of the design.

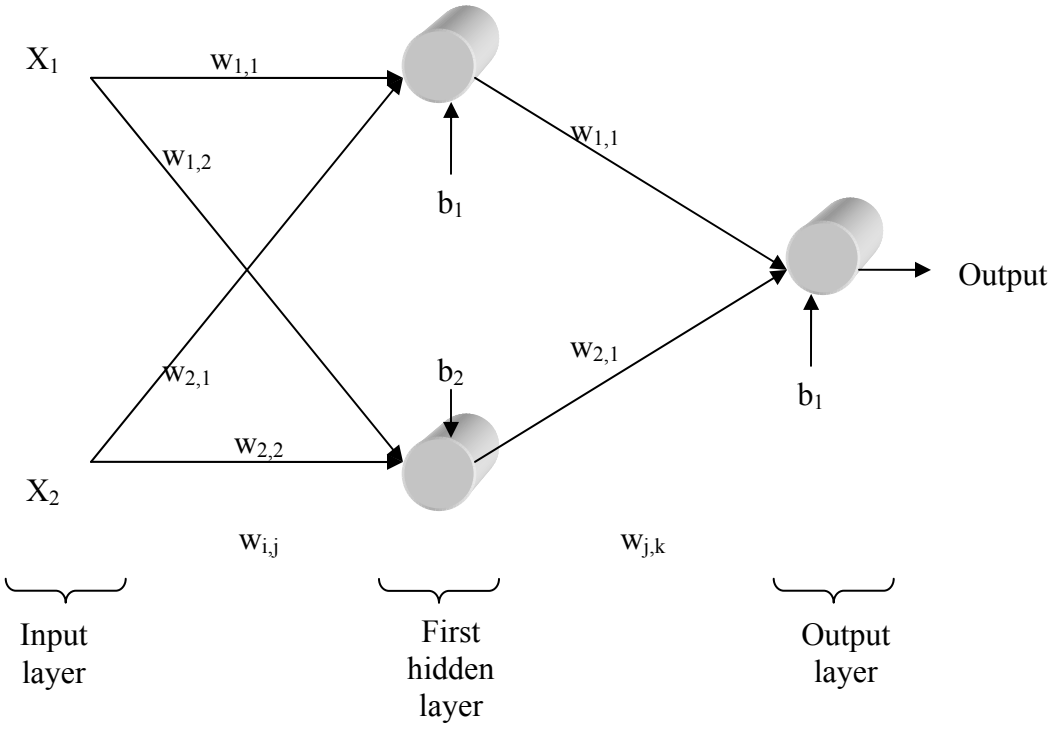


Figure 8. Architecture of XOR Problem

The activation function for their model is the linear threshold function, where $T = 0$, as shown in Figure 9. It is similar to a sigmoid function with a large β value.

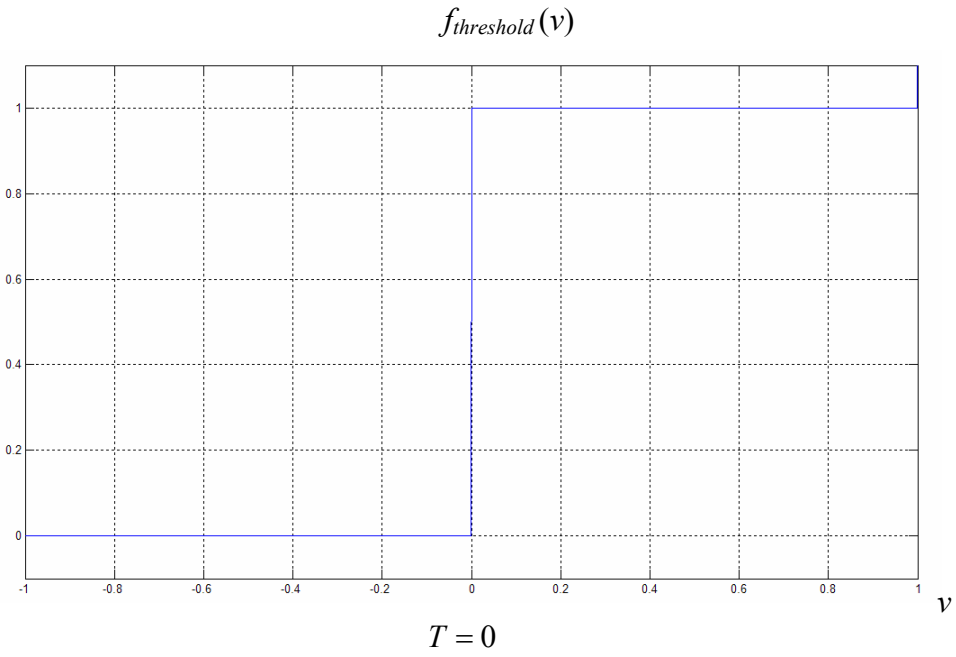


Figure 9. Graph of Linear Threshold Function

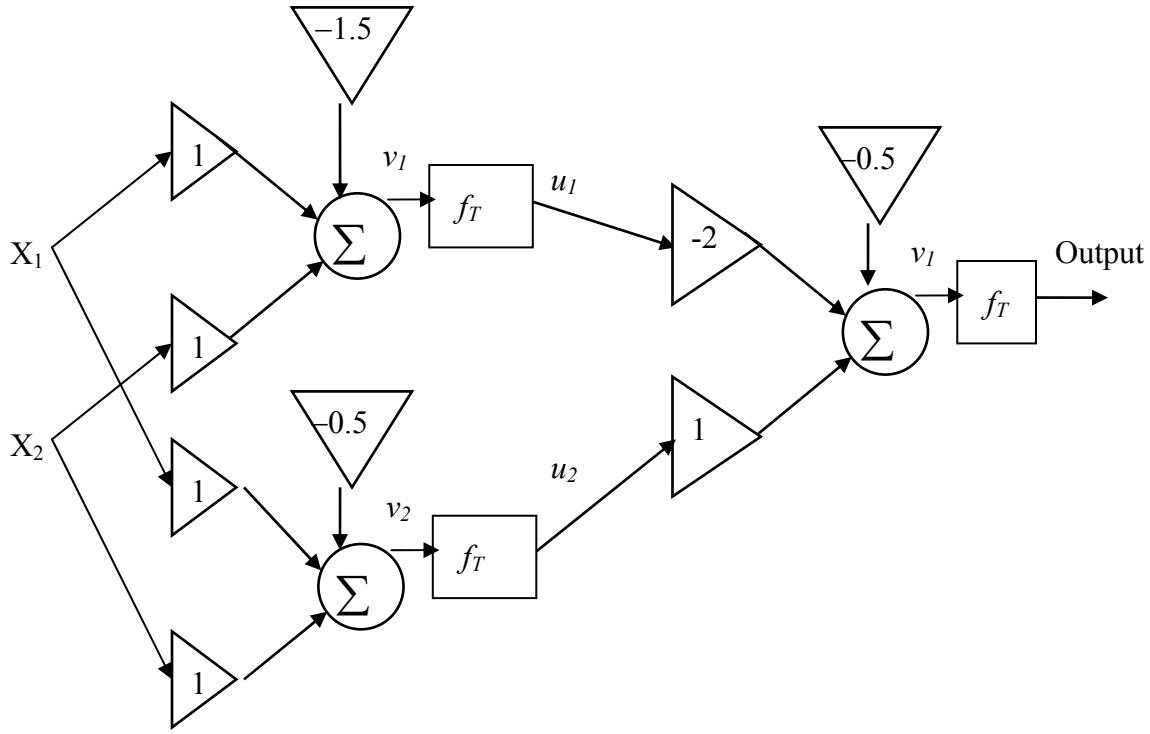
The weights and biases can be any combination of values. The following values are commonly chosen because of their simplicity and are already published in [8]. The weights $w_{i,j}$ and biases b_j are defined for the input layer as follows:

$$w_{1,1} = w_{1,2} = 1$$
$$b_1 = -1.5$$
$$w_{2,1} = w_{2,2} = 1$$
$$b_2 = -0.5.$$

The weights $w_{j,k}$ and biases b_k are defined for the hidden layer as follows:

$$w_{1,1} = -2$$
$$w_{2,1} = 1$$
$$b_1 = -0.5.$$

The signal flow diagram in Figure 10 illustrates the network with the numeric weights and biases. The outputs at each stage are computed below.



X1	X2	v1	v2	u1	u2	v1,output	Output
0	0	-1.5	-0.5	0	0	-0.5	0
0	1	-0.5	0.5	0	1	0.5	1
1	0	-0.5	0.5	0	1	0.5	1
1	1	0.5	1.5	1	1	-1.5	0

Figure 10. Signal Flow Diagram of XOR Problem

2. Backpropagation

The multi-layer perceptron neural network has the ability to 'learn' or create a pattern to minimize the mean-squared error between the generated output and the desired output. There are other training algorithms, but the backpropagation algorithm is the most commonly demonstrated.[8] The backpropagation algorithm relies on the simple difference equation of the desired output $(y_{desired})$ and the actual output (y) to find the error of the network output (e_y),

$$e_y = y_{desired} - y. \tag{3}$$

As shown in Figure 11, the neural network produces the output values after each iteration, and each iteration is referred to as an epoch. After each epoch, the mean-squared error of the network is evaluated to see if it has reached a minimum. The learning process continues on epoch by epoch until an arrangement of weights and biases produce the minimal error for a problem. By correlating the data and creating a pattern in the network through training, when the network is presented with input data outside the training set, the network will produce reasonable output values for the patterned function.[8]

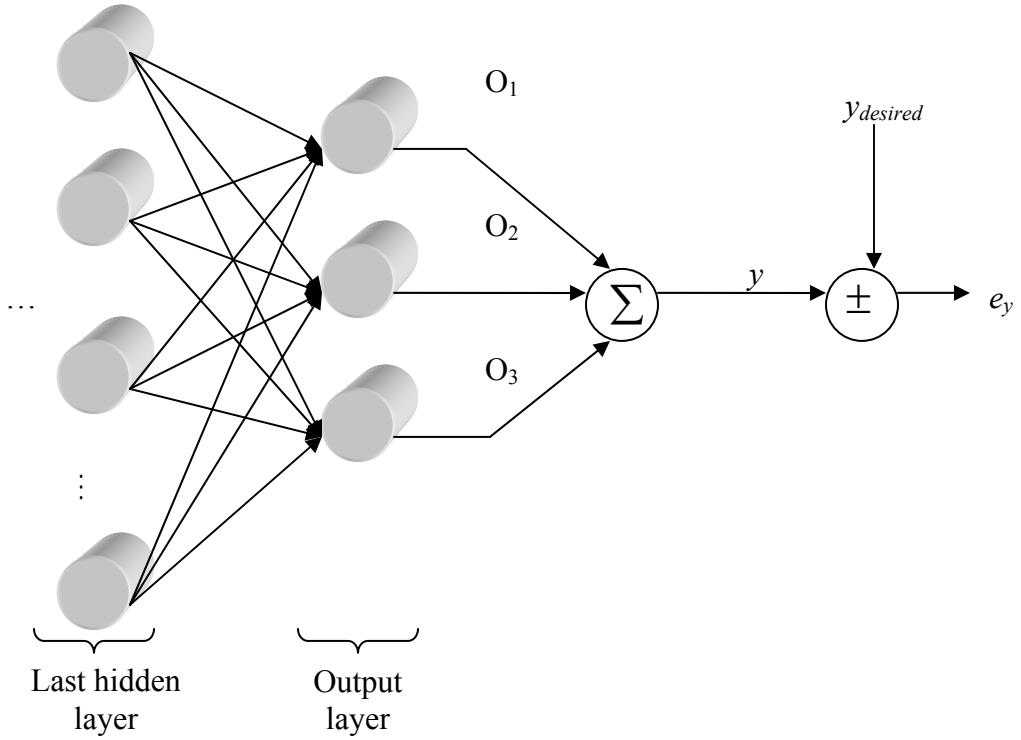


Figure 11. Error of the Network Output

a. Forward Path

Since there can be multiple hidden layers, the output y is generated at the end of the forward path, as shown in Figure 11 and derived in [8]. One hidden layer is illustrated in Figure 12. The first term v_j is the collection all inputs and weights on the node, where m is the total number of inputs (for simplicity, bias values are not included)

$$v_j = \sum_{i=1}^{m} w_{i,j} I_i. \tag{4}$$

The sigmoidal non-linearity function is solved for each value of v_j :

$$f_{sigmoid}\left(v_j\right)=\frac{1}{1+e^{-\beta v_j}}. \tag{5}$$

Any value of β can be used. But once β is defined for a neural network, it will not be changed. The same process can be continued for all cascaded layers where $f_{sigmoid}\left(v_j\right)$ will be multiplied by the next layer of weights until the final layer. The output of the network is

$$y = f_{sigmoid,final}(v_{final}). \tag{6}$$

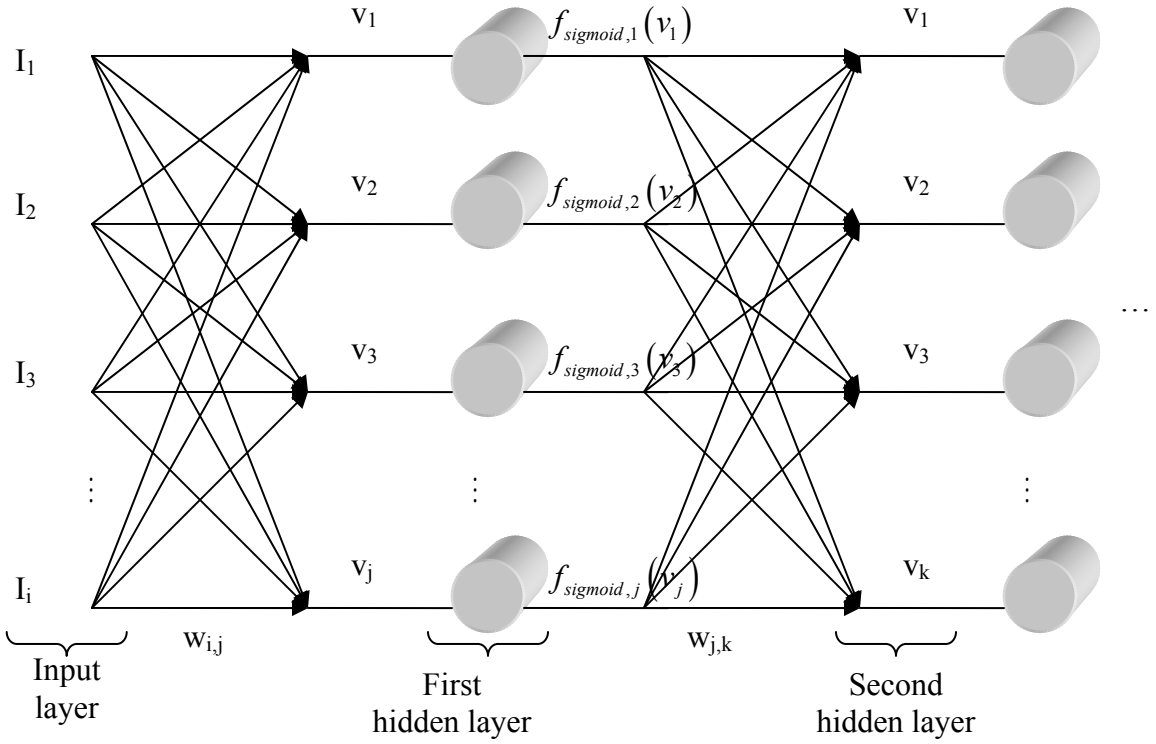


Figure 12. Forward Path

b. Backward Path

The backward path shown in Figure 13 is a result of the backpropagation algorithm. For each epoch, the error signal is propagated back through the network. By following the backward path, the weights of each connection will change according to

$$\Delta w_{i,j} = \eta e_y f_{sigmoid}\left(v_i\right) y_j. \tag{7}$$

The degree of changes in the weight value depend on a value referred to as the learning rate η.[8] The learning rate is directly related to the response time of the neural network in identifying a pattern. If η is too small, the network will require more iterations to con-

verge on a pattern. If η is too large, the weights will adjust significantly, typically overshooting the desired pattern, causing the network to diverge rather than converge.[9]

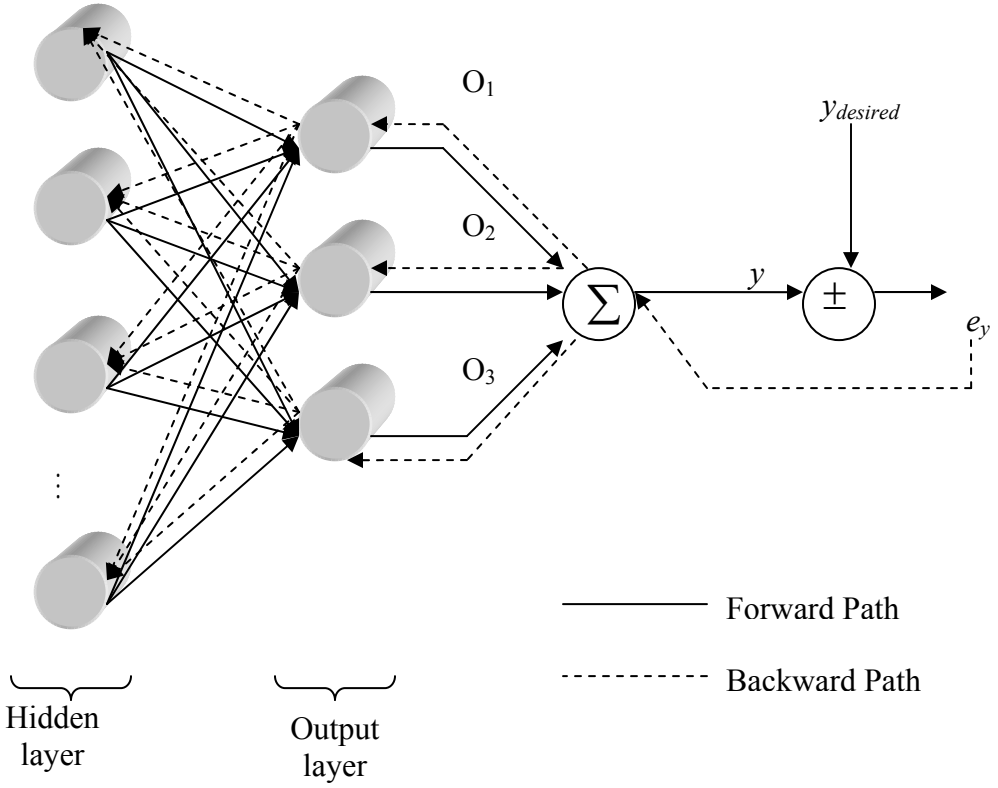


Figure 13. Backpropagation

The backpropagation algorithm is effective when solving the non-linear neural networks problems.

D. BACKGROUND CHAPTER SUMMARY

A swarm is a group of simple individuals that display characteristics such as decentralization, no synchronization, and communication amongst the group. A swarm is able to self-organize to complete tasks as a group. For artificial swarms, behavioral algorithms have been created to model the real swarm characteristics of bees and ants. The genetic algorithm and evolutionary algorithm are two methods for finding a solution in a search space. Both algorithms perform well, so the algorithms continue to be improved upon. The focus of Chapter III is PSO, which is an emerging evolutionary algorithm.

Artificial neural networks are another method for simulating artificial behavior. Backpropagation is the most common method for finding the arrangement of weights and biases needed to solve a problem. Backpropagation is often compared to other evolutionary algorithms such as PSO. PSO is described in Chapter III, and its performance is compared to backpropagation.

III. PARTICLE SWARM OPTIMIZATION ALGORITHM

The PSO algorithm is similar to evolutionary algorithms but altered to model flocks of birds. A flock of birds displays more desirable behavioral characteristics than the previous insect swarms. Therefore, the PSO algorithm incorporates more desirable movements than the previous evolutionary algorithms. The PSO equation determines the velocity of individual in the swarm. A simulated flock can locate a target while moving according to the PSO equation. The PSO algorithm also applies to neural networks, and it performs better than backpropagation.

A. THEORY

1. Flocks

Scientists are attempting to model the intriguing actions in bird flocking. The synchronous motions of the flock can be observed as they quickly change directions simultaneously, scatter, and regroup. The flock's social behavior is similar to swarms of insects, schools of fish, and herds of animals. Behaviors are stimulated through environmental factors. Whether finding food, avoiding predators, or pursuing better environmental conditions such as temperature, the groups typically move with a purpose. Their dynamic behavior in a flock appeared to be related to the distances between each bird in the flock. The initial simulations of flocks of birds were based on modeling these distances. The theory was that birds try to maintain an optimum distance between neighbors, which results in the synchronous movement.[2]

The initial theories are lacking realism. They are lacking any avoidance measures, so the simulated birds will collide on the screen. Since two real objects cannot occupy the same space, another approach had to be taken. Another consideration is an evolutionary algorithm. In evolutionary algorithms, the positions of the population change because the current population creates another child population with similar positions. This position change is dependent upon the recombination of parent positions. The child does not occupy the same position as either parent, but the child's position is nearby.[7] The positions changes are abrupt, and there is not a smooth transition from one generation to the next. The evolutionary algorithm needs to be modified.

To provide a realistic algorithm, the population of a flock has to remain the same without 'reproducing.' Real flocks of birds do not reproduce during mid-flight. To reach all the desired locations in the space, the population needs to physically move to the new position rather than simply relocate. The individual members of the population will move from position to position by an assigned velocity. To move together as a flock, the original concept of maintaining an optimum distance difference gave way to velocity matching. The velocity of an individual is changed to match the velocities of their neighbors. The agents, the common term for the individuals in the simulated flock, will begin to synchronize their movements by having the same velocity vectors. The initial simulations also lack stimulants to trigger changes in behavior. Once the simulated flock joins together in formation, it continues in one direction uninterrupted. To cause change, 'craziness' is implemented, so a random variable is added to the matched velocities in order to create variations in the flock movement.[2]

Frank H. Heppner, a zoologist at the University of Rhode Island, created simulations that involve a flock being attracted to a 'roost.'[2] The roost is a selected position on the screen to give the flock a target. After each iteration, the agents can determine their distances from the roost by using a two-dimensional XY distance equation referred to as the 'cornfield vector,' where

$$distance = \sqrt{(presentx - 100)^2 + (presenty - 100)^2} \tag{8}$$

for an agent at point $(presentx, presenty)$ and a roost at the point $(100,100)$. Each agent is allowed memory so that it can remember its best distance achieved. The best position distance for each agent i will be referred to as $pbest[i]$ and separated into components $pbestx[i]$ and $pbesty[i]$. Each agent can easily adjust the velocity components to move toward $pbest$. The degree of adjustment is limited by a predetermined value called $p\_increment$. Variation is still added to flock by randomizing the velocity adjustment from 0 to $p\_increment$. The new velocity vectors are:

$$\begin{aligned}
&\text{if } presentx[i] > pbestx[i], \text{then } vx[i] = vx[i] - \text{rand}() * p_increment \\
&\text{if } presentx[i] < pbestx[i], \text{then } vx[i] = vx[i] + \text{rand}() * p_increment \\
&\text{if } presentx[i] > pbesty[i], \text{then } vy[i] = vy[i] - \text{rand}() * p_increment \\
&\text{if } presentx[i] > pbesty[i], \text{then } vy[i] = vy[i] + \text{rand}() * p_increment.
\end{aligned} \tag{9}$$

Another advantage of a flock, or any group from insects to herds, is the communication within the group. Sharing information allows the group to benefit from one individual's discovery. The group can collectively achieve the goal, whether it is finding food or another desired object. Incorporating this social behavior into the artificial flock meant each agent had to retain the value of the best position out of the entire flock, the global best position. The array index $gbest$ is used to indicate this global position, so $pbestx[gbest]$ is the X component global best position and $pbesty[gbest]$ is the Y component global best position. The array of agent values and $gbest$ is updated every iteration to reflect the most recent positions and the global best position. The following equations update the agent velocity vectors:

$$\begin{aligned}
&\text{if } presentx[i] > pbestx[i], \text{then } vx[i] = vx[i] - \text{rand}() * g_increment \\
&\text{if } presentx[i] < pbestx[i], \text{then } vx[i] = vx[i] + \text{rand}() * g_increment \\
&\text{if } presentx[i] > pbesty[i], \text{then } vy[i] = vy[i] - \text{rand}() * g_increment \\
&\text{if } presentx[i] > pbesty[i], \text{then } vy[i] = vy[i] + \text{rand}() * g_increment,
\end{aligned} \tag{10}$$

where $g\_increment$ serves the same purpose as $p\_increment$ earlier. When $g\_increment$ is a large value, the velocities are over-adjusted. The flock quickly clusters around the roost within a few iterations. When $g\_increment$ is smaller, the flock circles around the target. The flock has time to demonstrate synchronized movement as it approaches and lands on the roost. This simulation proves to be realistic, and it works without the original concepts such as velocity matching and craziness.[2]

To further improve performance, the velocity change needs to be more proportional to the distance of the agent from the targeted object. Instead of classifying all targets either greater than or less than the best position, the distance and direction of the present position from the personal best position can be incorporated into the equation:

$$vx[i] = vx[i] + \text{rand}() * p\_increment * \big(pbest[i] - present[i] \big). \tag{11}$$

Further testing shows that the values of $p\_increment$ and $g\_increment$ are irrelevant after adjusting the algorithm for acceleration. To incorporate both the personal best position and the global best position, the random terms are multiplied by 2 to give a mean value of 1.[2] The final equation is settled upon:

$$vx[i] = vx[i] + 2*\text{rand}()*\left(pbestx[i] - presentx[i]\right) + 2*\text{rand}()*\left(pbestx[gbest] - presentx[i]\right). \quad (12)$$

Other versions of the equation have been tested, but Equation (12) is the most effective. Thoroughly derived in [12], it is known as the Particle Swarm Optimization (PSO) equation, credited to James Kennedy and Russell Eberhart in 1995. Kennedy and Eberhart decided that a swarm of particles was a more appropriate name for the flock of agents since the simulations began to take on the swarm and swarm intelligence characteristics previously outlined.

2. Parameters

The Particle Swarm Optimization equation is commonly cast in the form

$$v_i(k+1) = w(k)v_i(k) + \alpha_1[\gamma_{1i}\left(p_i - x_i(k)\right)] + \alpha_2[\gamma_{2i}\left(G - x_i(k)\right)] \quad (13)$$

in [4]. The PSO equation is to be applied to the position equation of a particle

$$x_i(k+1) = x_i(k) + v_i(k+1) \quad (14)$$

where:

i = the particle index,

k = the discrete time index,

$v_i(k)$ = the current velocity of the i-th particle,

$v_i(k+1)$ = the velocity of the i-th particle at the subsequent time index,

$x_i(k)$ = the current position of the i-th particle,

$x_i(k+1)$ = the position of the i-th particle at the subsequent time index,

p_i = the personal best position of the i-th particle,

G = the global best position found by all particles in the swarm,

γ_{1i}, γ_{2i} = different random numbers from 0 to 1,

$w(k)$ = inertia weight function to determine the influence of the current velocity on the subsequent velocity, and

α_1, α_2 = acceleration constants.

All of the parameters of the equation are experimentally evaluated to find optimal values. The acceleration constants α_1 and α_2 are commonly set at 2 to maintain a mean value of 1 for the random numbers. The inertia weight function $w(k)$ is typically either a constant or a decreasing linear function. The inertia weight function determines the influence of the current velocity on the subsequent velocity. The benefit of having a decreasing linear function is that the influence of the current velocity also decreases as the training progresses. This idea will help the overall velocity change decrease as the training progresses. As the particles near the target, the decreasing velocity will cause the particles to advance more slowly. The decreasing speed allows a particle to make the small adjustments in position required to find the exact location of the target.[9]

The swarm will converge on a target, but a gradual paced convergence ensures that the space was thoroughly searched before settling on a target. When the convergence is quick, the swarm is suddenly drawn to the target, and the result would be a dangerous collision of particles. A maximum velocity V_{max} is used to limit $v_i(k+1)$ outside the PSO equation. When $v_i(k+1) < V_{max}$, $v_i(k+1)$ will be used to improve the position. When $v_i(k+1) > V_{max}$, V_{max} is used instead. If V_{max} is too large, the swarm will gather too quickly, limiting potential solutions.[9]

Other factors included in the simulation are search space range and population size. The search space range needs to allow enough freedom for a thorough search so that the swarm is not restrained from the start. Any particles outside the designated search space range will not find a position close to the target, so they will be directed back to their personal best position inside the search area. The population size is a trade-off between convergence rate and computational time. The compromised size is 25 particles since the performance increase from 20 to 25 particles was greater than 25 to 30 particles. Any population over 30 particles becomes a computational burden.[9]

B. RECENT RESULTS

1. Comparison to Backpropagation

The two-dimensional examples have been proven to work with PSO. Since most problems are nonlinear and multi-dimensional, the PSO equation is applied to training neural networks. Eberhart continues to apply the PSO concept to become an approxima-

tor that evolves to solve the problem.[7] The goal is to train a neural network better than previous methods such as training by backpropagation.

Based on computational requirements in [9], Venu Guidise and Ganesh Venayagamoorthy demonstrate that the PSO is more successful than backpropagation. They put the two algorithms through the neural network to solve the nonlinear quadratic equation $y = 2x^2 + 1$ by reaching a goal error of 0.001. The comparison is based on the total number of computations required to reach the goal error averaged over 10 trials. In short, the experiment shows that PSO requires fewer computations than backpropagation. Other factors are still important in determining performance, so PSO is not proven to be overall better than backpropagation. They did not adjust the multiple parameters of the PSO equation once they settled on the default values for the experiment. Backpropagation also has a greater dependence on bias values than PSO, and that difference is not investigated either.[9]

2. PSO Toolbox

Brian Birge created a Particle Swarm Optimization toolbox (PSOt) for the MATLAB scientific programming environment.[4] The toolbox provides additional functions that allow an operator to use when programming. This toolbox is designed to work alone or with the already developed neural network toolbox. In the Neural Network toolbox, the neural network is trained through backpropagation. Birge created functions to train the neural networks with PSO. He outlines the capabilities and a few examples in [3]. Other parameters are outlined in Appendix A, but the inertial weight function will be described since it has a significant effect on the next velocity. For $w(k)$, Kennedy and Eberhart discovered that the best experimental value so far is a linear decrease from 0.9 to 0.4 over 1000 iterations.[7] Birge uses a linear decrease from 0.9 to 0.2 over 1500 iterations as a default parameter for his PSO function. The function also evaluates a solution for the neural network in nine dimensions.[4]

Birge creates a demonstration that compares backpropagation to PSO in [4]. This demo program can train a neural network to solve the XOR function previously discussed. The demo program allows the user to select zero, one, or two hidden layers for the neural network and whether the network is to be trained by PSO or backpropogation.

Appendix B outlines an example for training the neural network with one hidden layer to solve the XOR problem using backpropagation. The sum-squared error is plotted after each iteration. Figures 14 through 16 illustrate how the algorithm is trying to converge to a sum-squared mean error of 0.02, represented by the red dotted line in the plots. If the network has not reached the goal error of 0.02 by 1000 epochs or remains at the same error for an extended period of time, the training starts over with different initial weight values. The equations used for determining of the new initial weights can be found in [4] and [7]. This example finds a solution after 2688 iterations. The first 1000 epochs in Figure 14 only reach the error of 0.998464, extremely far from 0.02. The training is started over with different initial weights.

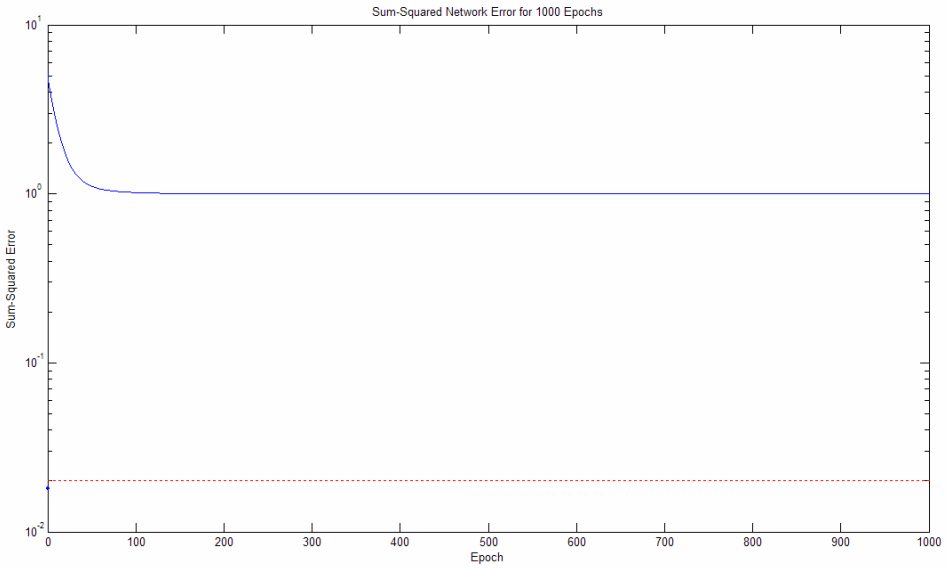


Figure 14. Plot of First 1000 Epochs of the Backpropagation Algorithm for an XOR Problem

The lowest error at the next 1000 epochs in Figure 15 is 0.542084. Since these 1000 epochs in Figure 15 also do not reach the goal error of 0.02, the training is started over again with another set of different initial weights.

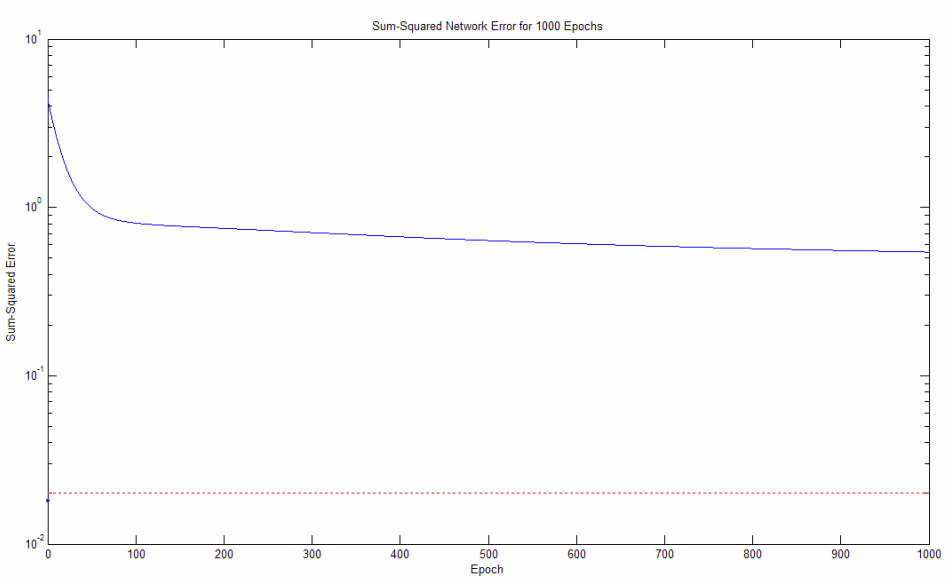


Figure 15. Plot of Next 1000 Epochs of the Backpropagation Algorithm for an XOR Problem

The next set of iterations converges at a faster rate because the initial weights are now closer to the desired goal. After 688 iterations, the algorithm converged to meet the goal of 0.02, shown in Figure 16. By reaching an error of 0.02, the program indicates that the neural network is trained for the XOR problem.

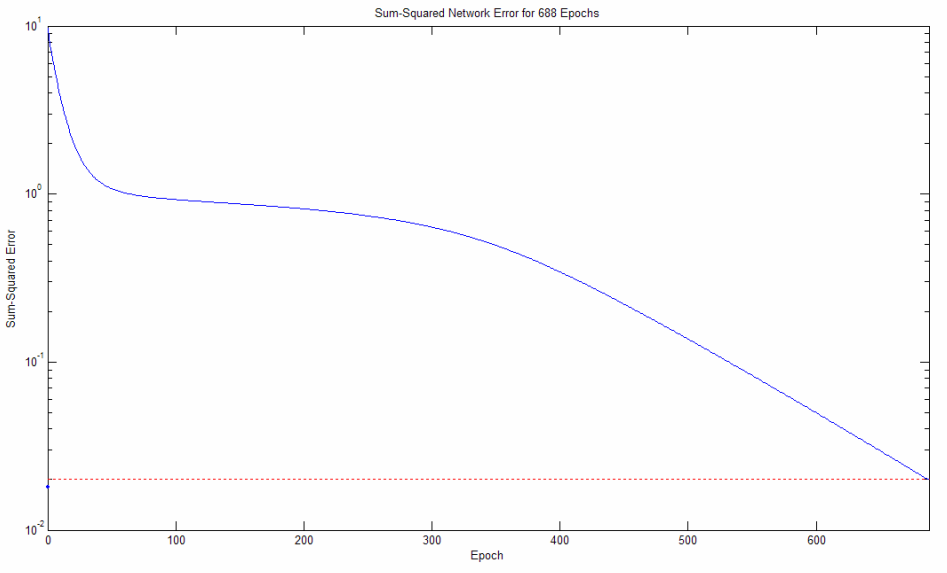


Figure 16. Plot of Final 688 Epochs of the Backpropagation Algorithm for an XOR Problem

Figures 17 through 21 illustrates an example for training the neural network with one hidden layer to solve the XOR problem using PSO; the full program output outlined in Appendix C. The PSO program plots the sum-squared error after each iteration (same process as in backpropagation). The additional plot in the PSO demonstration provides an illustration of the particle positions, personal best positions, and global best position. The first dimension values of the positions are on the x-axis while Birge plots the ninth dimension values on the y-axis.[4] The blue dots represent the particle locations, and the black dots represent each particle's personal best position. The red crosshair represents the swarm's global best position. The final plot shows a magenta line to represent the movement of the global best position around the search space. This example finds a solution after 1340 iterations. The first 1000 iterations algorithm converged to a sum squared error of 0.66687. Figures 17 through 19 show the progression during the 1000 iterations at 25, 600, and 1000 epochs. The location of the global best position barely moves throughout the search space.

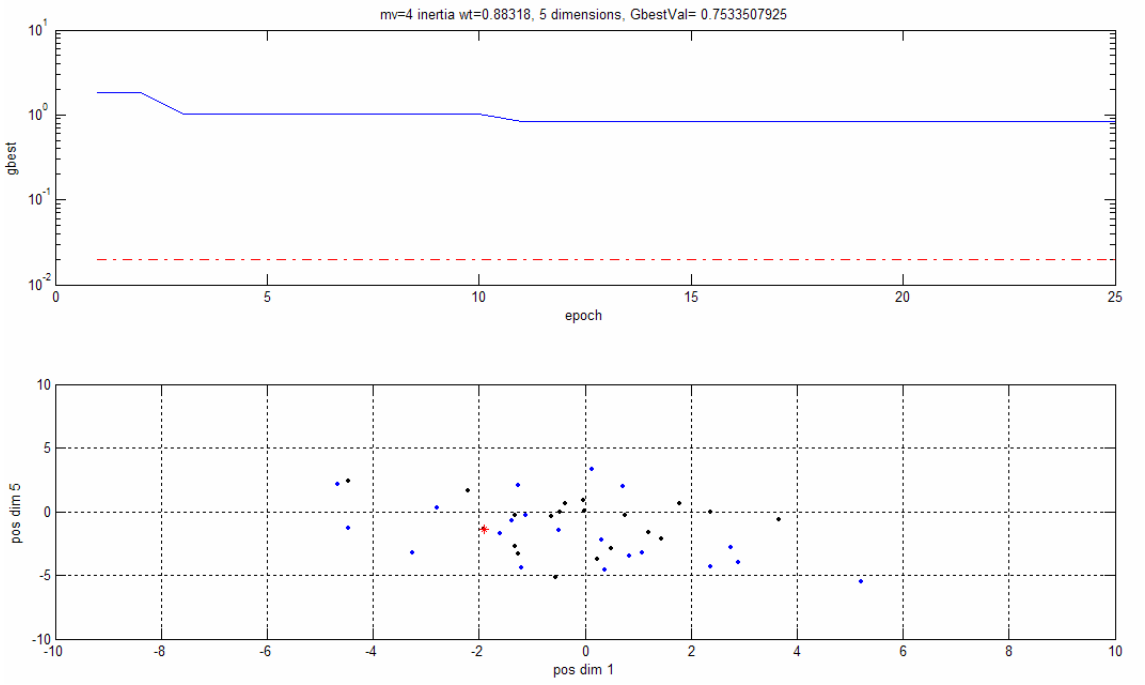


Figure 17. Plot of First 25 Epochs of the PSO Algorithm for an XOR Problem

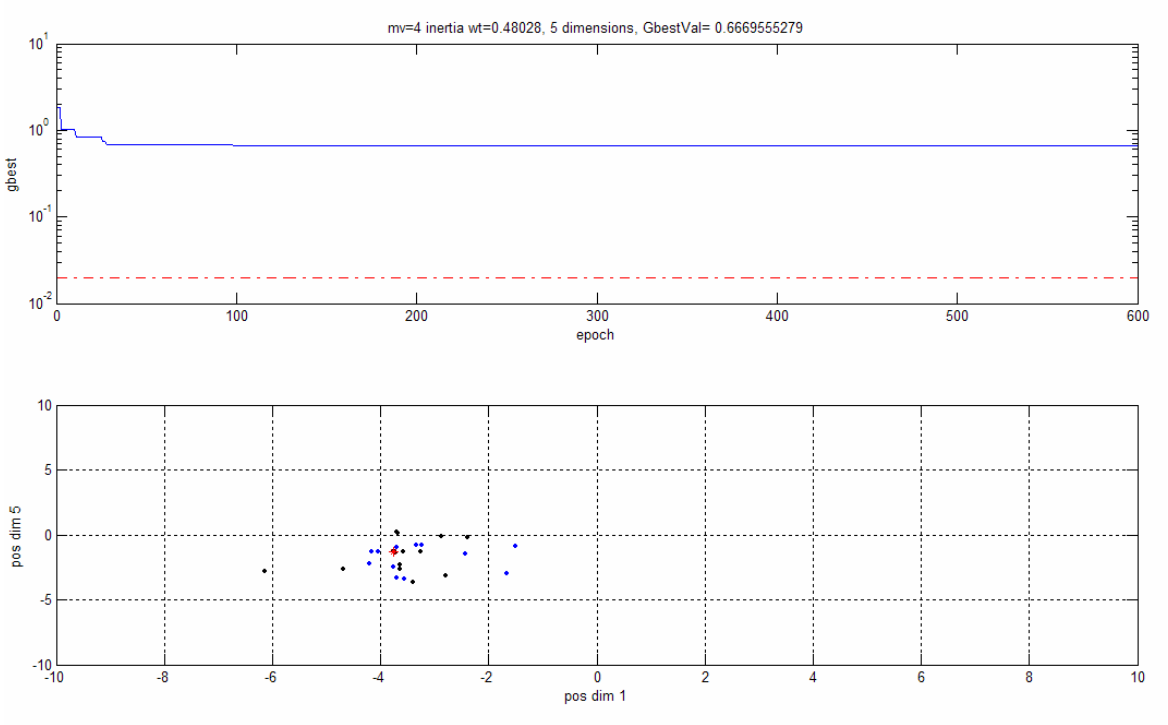


Figure 18. Plot of First 600 Epochs of the PSO Algorithm for an XOR Problem

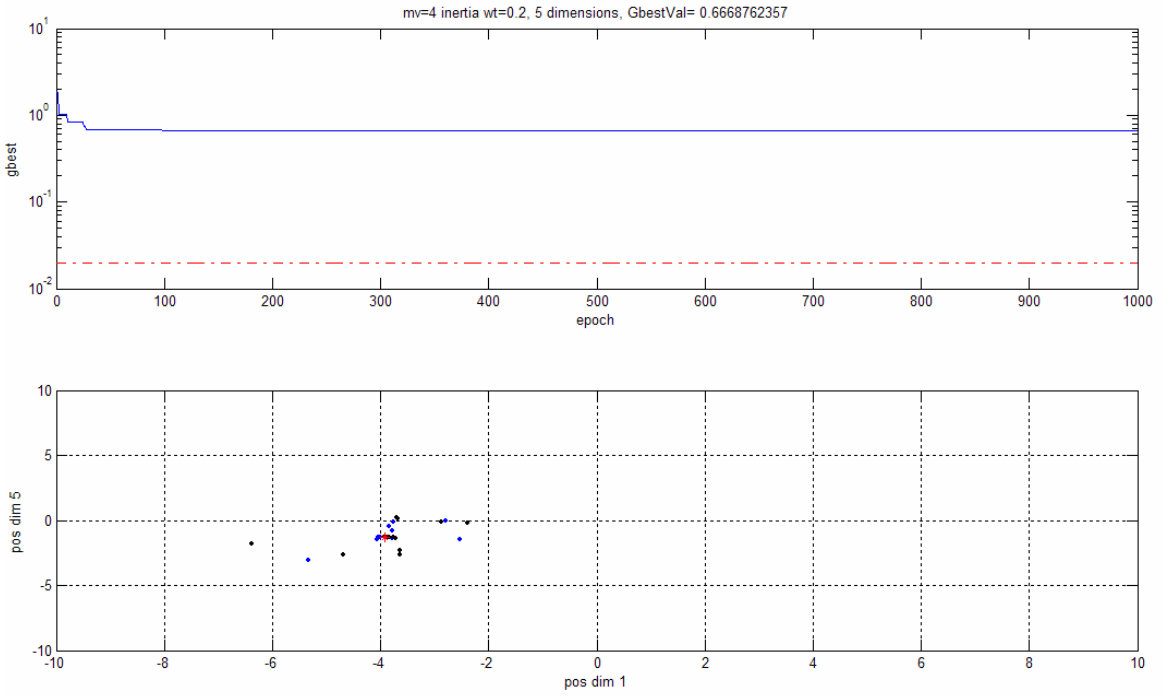


Figure 19. Plot of First 1000 Epochs of the PSO Algorithm for an XOR Problem

The algorithm has failed to reach a sum-squared error of 0.02 after 1000 epochs, so it starts over as indicated in Figure 20.

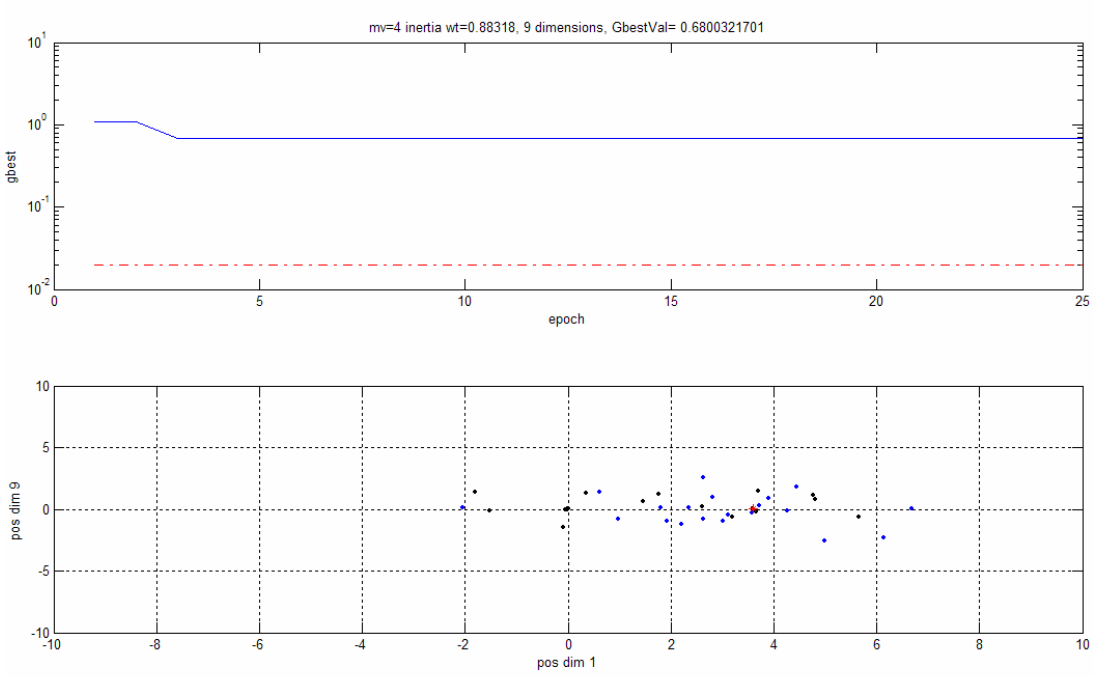


Figure 20. Plot of Next 25 Epochs of the PSO Algorithm for an XOR Problem

As shown in Figure 21, the program reaches a solution by 340 epochs. The magenta line represents the path of problem solving, which is the path of the global best position over the 340 epochs. The blue line also indicates sum-squared error of 0.02. As a result, the neural network is trained by the PSO algorithm to solve the XOR problem after 1340 epochs.

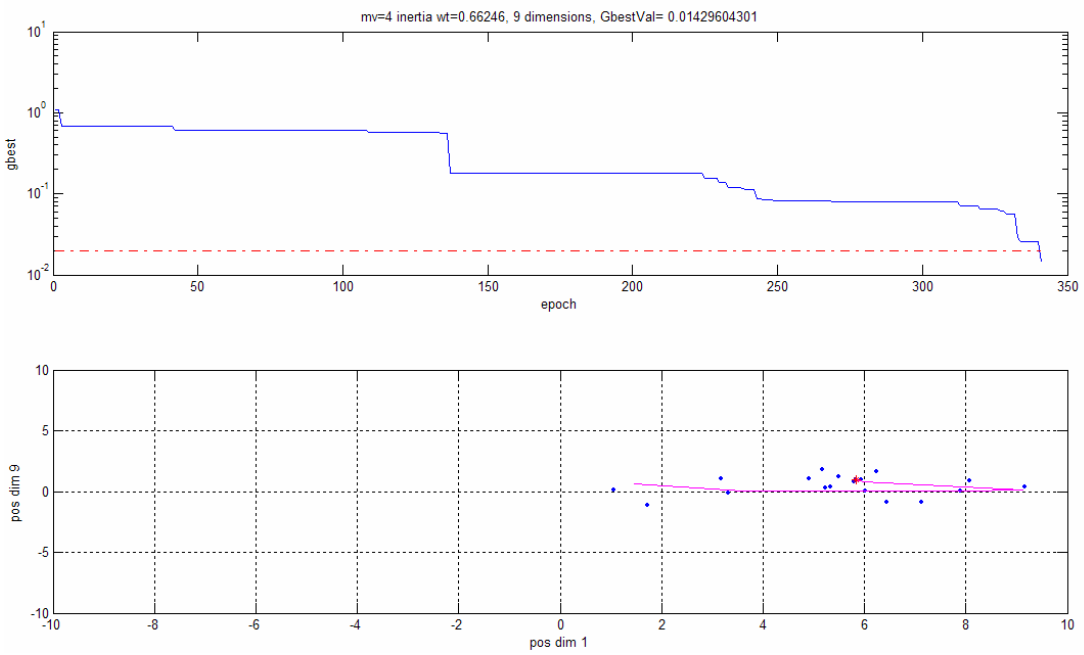


Figure 21. Plot of Next 340 Epochs of the PSO Algorithm for an XOR Problem

C. CONCLUSIONS

These demonstrations are only simple examples of particle swarm abilities. The PSO algorithm performs the same as, if not better than, the backpropagation algorithm. The PSO method typically finds the solution faster than backpropagation. The toolbox for MATLAB is helpful because it allows the training of a neural network by the PSO algorithm without having to write a new code each time. The PSO algorithm simulation in MATLAB is analogous to UAVs searching for a known target using a non-linear method. Unfortunately, PSO does not apply realistically to large objects such as UAVs searching for a target. Particles are accelerating at all different rates and directions, and these maneuvers are not practical when the particle is a real aircraft. Changing velocity is not fuel efficient, especially when it is altered repeatedly. The smaller UAVs are more agile than the larger version, but they are still not designed to handle aggressive changes in speed and abrupt changes in direction. The PSO algorithm requires both aggressive maneuvers. Collision avoidance is an essential concept that is not even considered with the particle swarm. The UAVs would collide with each other because the next velocity vectors and positions are unpredictable and calculated without concern for obstacles nearby. A more realistic approach is still needed to maneuver a swarm of UAVs. A linear algorithm is presented in Chapter IV.

IV. LINEAR ALGORITHM FOR AUTONOMOUS UAV CONTROL

A linear algorithm provides an effective method for maneuvering individuals in a swarm. By keeping velocity constant, the swarm of UAVs are realistically simulated.

A. THEORY

Two concepts from the nonlinear approach are applicable to other methods. First, the swarm does not have to travel as a synchronous flock toward a target.[2] To find a known target, the UAV can simply try to find the shortest distance between itself and the target. Second, minimizing error between the particle and the target is an effective process for finding the target.[4] The acceleration of particles around the search space is the main concern. By keeping a constant velocity, a linear algorithm can be used instead of the previous non-linear equation. To find the target, the direction the UAV is heading needs to change rather than the velocity. A linear control approach should be simpler and more stable since the UAVs will not quickly change directions and locations. Maintaining a constant speed while searching and attacking a target will ensure better fuel efficiency. The linear algorithm has faster computational times, and a straightforward linear simulation can be easily scaled for a larger search space.[3]

Chin Lua, Karl Altenburg, and Kendall Nygard model a group of UAVs as a swarm to conduct a synchronous multi-point attack in [3]. A synchronous attack is different from a synchronous swarm. A synchronous attack means that the swarm will reach the target at the same time, but the individual UAVs are moving asynchronously. The program uses a linear algorithm to determine how each UAV should move. Before deciding on the next position, it considers other factors besides target location. In a real group, each UAV has the ability to gather and evaluate data from the environment with sensors. Sensors allow communication with the other UAVs and determine distances between objects.[3]

1. Sensors

Each UAV carries various sensors to decipher the surrounding environment. In order to coordinate an attack, each UAV must communicate with the others. This communication requires a transmitter and receiver. Communication is a disadvantage when

conducting operations in an enemy area. Each communicated signal puts the UAV at risk since the signal can be traced by the enemy. This program keeps communication to a minimum. The necessary low-power transmissions between UAVs are short-range signals. The communication is limited to the 'visual' range of the sensors. Typically the visual range only includes the immediate neighboring UAVs. Instead of broadcasting to the entire swarm, information is passed from neighbor to neighbor until the whole swarm is informed.[3]

To find the target, the UAV has to be able to receive a long-range signal from the target. Any emissions from the target should be picked up by the receiver. Since receiving is a passive function, the UAV experiences less risk of enemy detection when receiving. Two receivers are now needed to collect both the short-range signals from the low-power transmissions of other UAVs and the long-range signals from emitting targets.[10]

Sensors are also required to determine the direction and distance of the received signals. From the long-range signal, the UAVs need to constantly identify the location of the target to allow for heading corrections. From a short-range signal, UAVs can determine distances. Each UAV can maintain a threshold distance so that they will avoid each other and obstacles that cross that boundary. From the gathered information, the members of the swarm can decide which function to perform. These movements of the swarm can be broken down into three basic maneuvers described in the following section.[10]

The simulated sensors provide the correct distances between objects because all object locations are globally known in the program. In the program, any information transmitted is globally communication, but a UAV is limited to transmitting under the correct circumstances.

2. Swarm Movement

The swarm does not move as a whole unit, but all the individuals in the swarm move according to three maneuvers: avoid, attract, and orbit.[3] Avoiding other UAVs and obstacles in the search space is the foremost function. Avoidance is crucial so that the UAV can survive to carry out the mission. Each UAV has a threshold boundary distance, so any UAV entering the threshold distance will cause both UAVs to avoid each other. To avoid an obstacle within a specified visual range, the UAV will turn counter-

clockwise until the obstacle is not longer in the path of the UAV. The attraction maneuver is required to direct the heading of the UAV toward the target. The target is at a known location for this simulation. In a two dimensional simulation, the distance and direction of the target is in terms of XY -coordinates. The UAV is constantly correcting its position and heading to minimize the distance to the target. Attraction is also used to create an orbit. When a UAV wants to orbit a circle, waypoints are mapped out to form a circle. The closest waypoint to the UAV is the reference point, and UAV is attracted to the next point counterclockwise. As the UAV moves, the points update, so the UAV is always attracted to the next point. The control architecture in Figure 22 dictates the hierarchy in movements. All five behaviors functions are derived from the three basic maneuvers.[3]

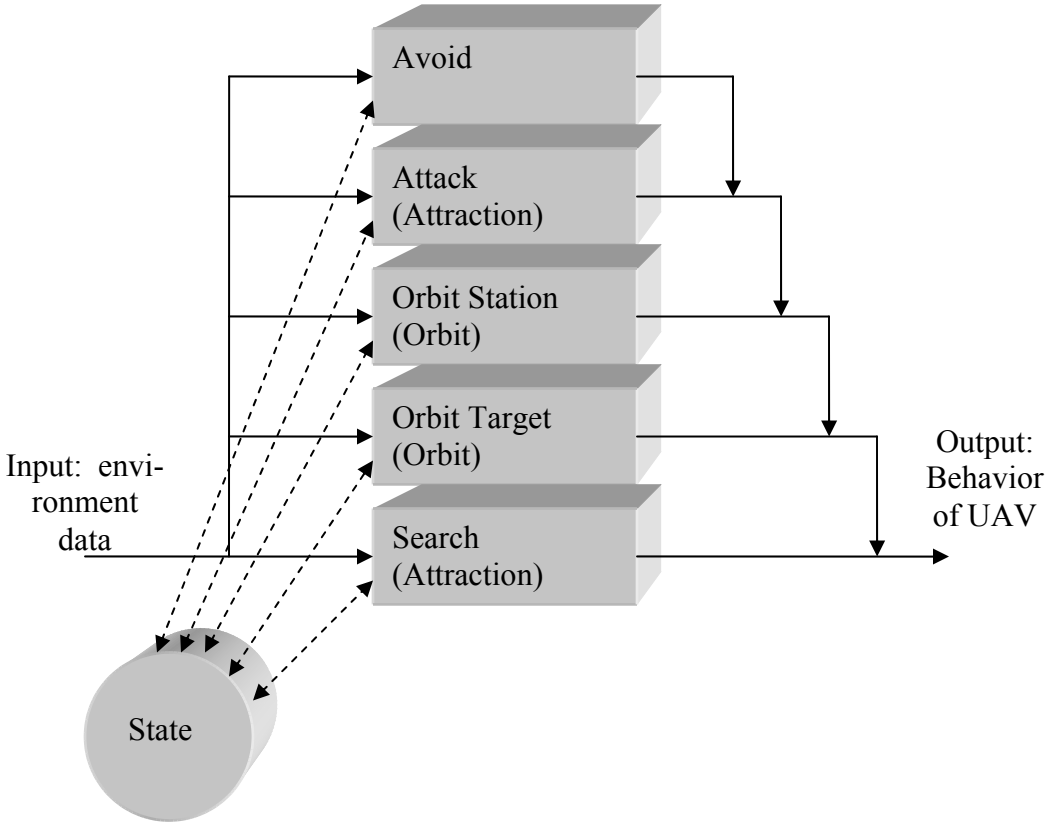


Figure 22. Control Architecture

The sequence of events is intuitive. The UAVs are in a search state unless they are required to avoid each other or to begin the attack sequence. It is important to understand that the UAVs continue to avoid each other during the attack sequence if necessary.

The UAVs enter the avoidance state often in the simulation, but the simulation covers only a small area. In reality, the distances are much greater so the UAVs are not operating in such a close proximity as indicated in the simulation. The attack sequence is to first orbit the target, then to orbit the station, and finally to attack the target. The attack sequence is better described with the example in the following section.[3]

B. RECENT RESULTS

The program swarm.java was created by Chin Lua, Karl Altenburg, and Kendall Nygard at North Dakota State University [3]. The simulation is a two-dimensional demonstration of a synchronized multi-point attack by UAVs. The population of UAVs can be varied from 1 to 18, but it is 8 for this example. The UAVs are represented by green triangles. The other parameters that are adjustable are speed, turning ability, repulsion range, and visual range. The display for these parameters is located in the upper left corner. Upon initialization, all UAVs will head 'south' until a target is located, shown in Figures 23 and 24. Although avoidance is the highest priority, there are no triggers before the first target to cause any UAV to have to avoid each other or to change direction.

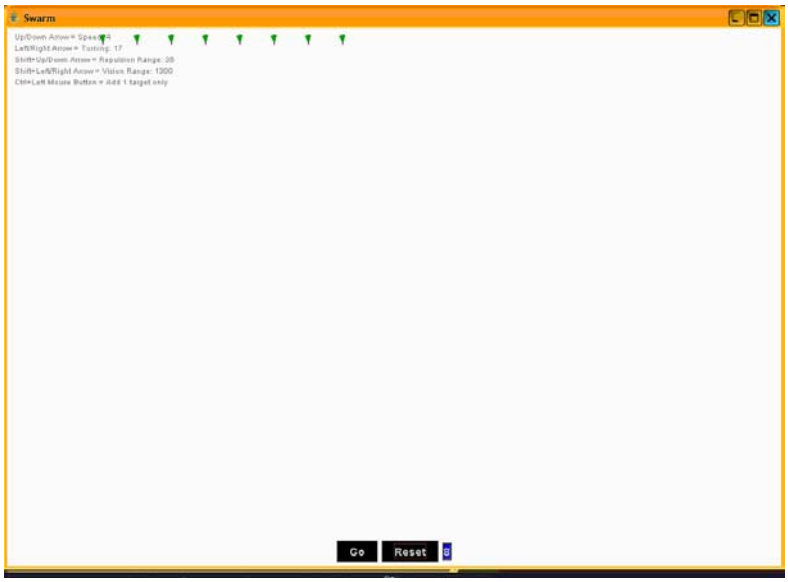


Figure 23. Swarm.java Program: UAVs at Initialization

Figure 24. Swarm.java Program: UAVs Head South

This program requires interaction from the user, so with the mouse, the user can place a target (represented by a blue circle) anywhere within the screen. There are two circles, an inner circle and outer circle, around the target that act as guides for the UAVs to follow. The target's inner and outer circles are predetermined to be safe distances from the target. A safe distance is out of any enemy detection range or counterattack range. Neither circle is illustrated, but the UAVs will head toward the inner circle until an orbit circle appears on the outer circle. In Figure 25, the outline of the inner circle can be detected from the circular path that five UAVs are following.

Figure 25. Swarm.java Program: UAVs Travel Around Inner Circle

The center for an orbit circle is 1 of 18 waypoints spaced 20 degrees apart around the outer circle. The outer circle outline will follow the center of the orbit circles. The first orbit circle is visually displayed in Figure 26. The orbit circle is represented by a black dotted circle. The location of the first orbit circle around the outer circle is chosen at random for the program, but it could also be determined according to the surroundings. A UAV will head toward the orbit circle if it is within its visual range. When the UAV color changes from green to purple, the UAV is heading toward an orbit circle or is currently orbiting the circle. Once a UAV touches the orbit circle, it becomes a station owner. Through the simulated local communication, the transmitter on the UAV broadcasts that it is the station owner. The purpose of the station circles is to collect all the UAVs in the same general area to locally communicate in order to coordinate the attack. The station circles are out of range of enemy detection, so the time allotted to station keeping is adjustable.

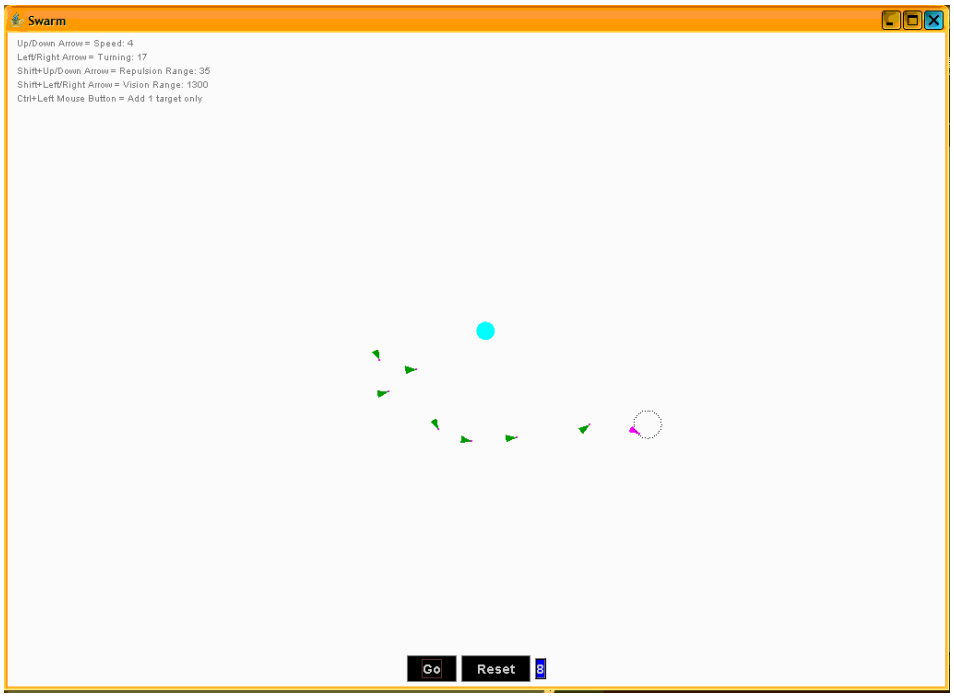


Figure 26. Swarm.java Program: First Orbit Circle

The remaining UAVs search for the successive orbit circles that will appear counterclockwise around the outer circle in Figure 27.

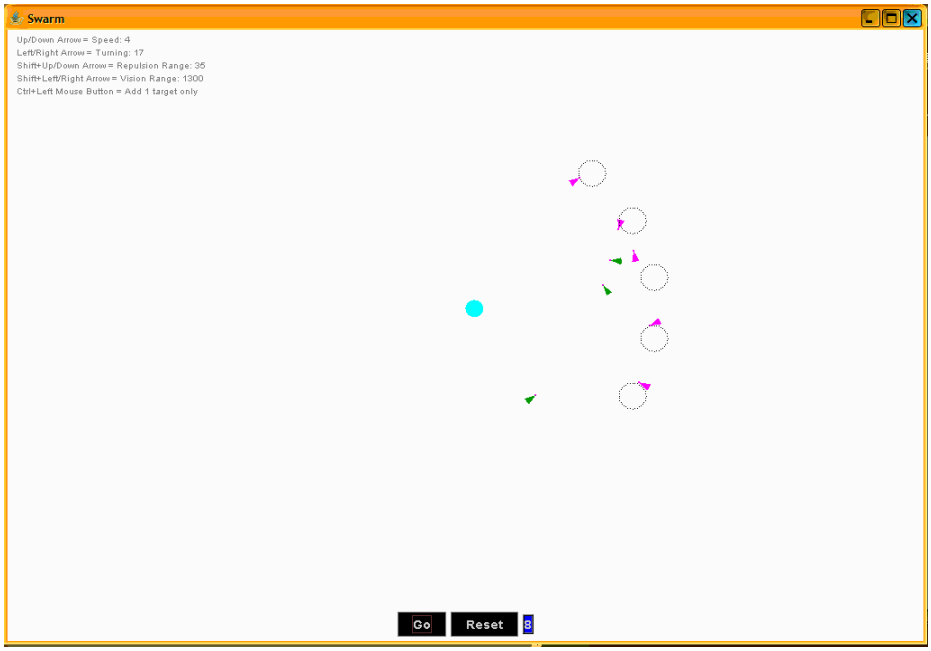


Figure 27. Swarm.java Program: 5 Orbit Circles

Figure 28 shows that all UAVs will become station owners and continue to orbit the circle.

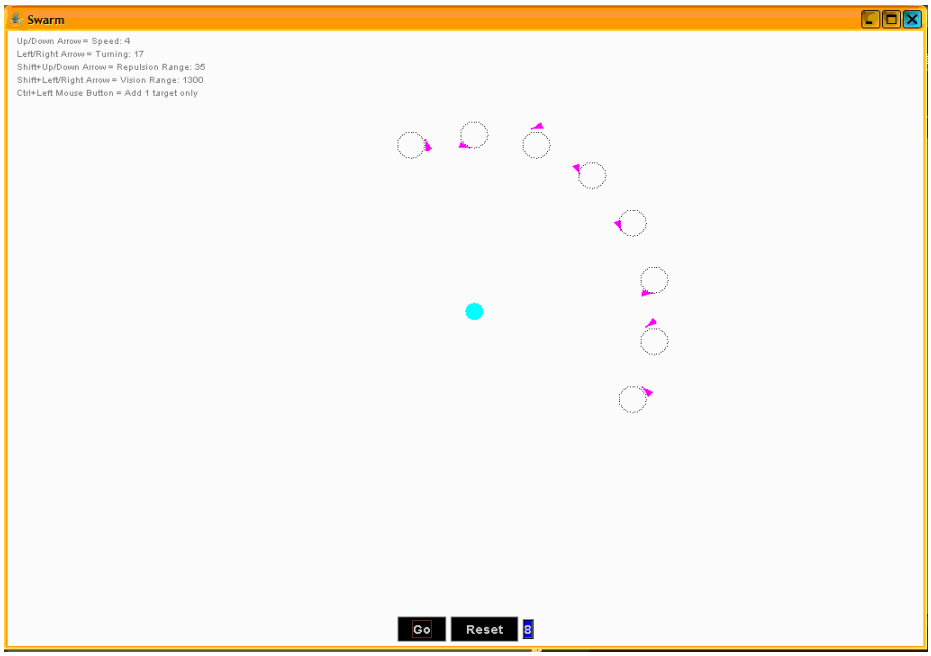


Figure 28. Swarm.java Program: 8 Orbit Circles

The attack is the only synchronized action. By orbiting around the station circles, the UAVs are allowing the other UAVs to reach the target. The first UAV to complete a specified number of orbits around the circle, analogous to the amount of fuel used, will issue an order of attack. The attack order is locally communicated from one UAV to the neighboring UAVs. It is irrelevant to specify which UAV issued the attack order. Since the UAVs travel at a constant velocity, delay is introduced into the program to indicate when each UAV should leave the orbit circle. The goal is to approach the target at the same time and from symmetric locations around the target. Since there are only 8 UAVs, the UAVs travel around the inner circle until they are equally spaced by 40 degrees. There are still timing errors in the simulation because the location of each UAV around the orbiting the circle is not synchronized. When a UAV leaves the orbit circle, it returns to a green color. Figures 29 and 30 show the UAVs leaving the orbit circle and traveling around the inner circle to take attack positions 40 degrees apart.

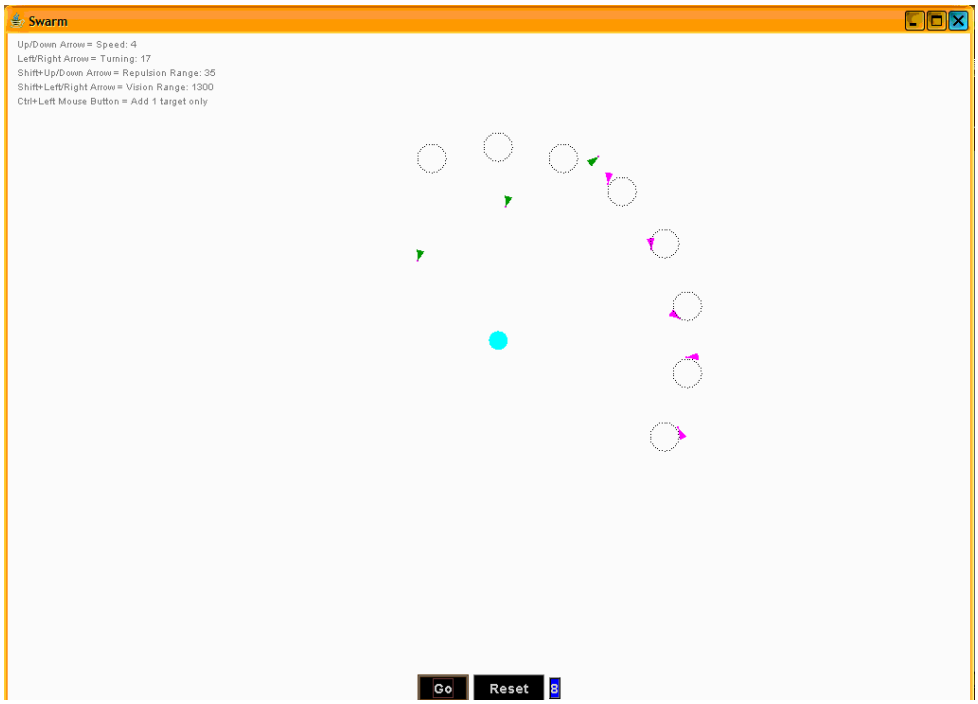


Figure 29. Swarm.java Program: UAVs Begin Attack Sequence

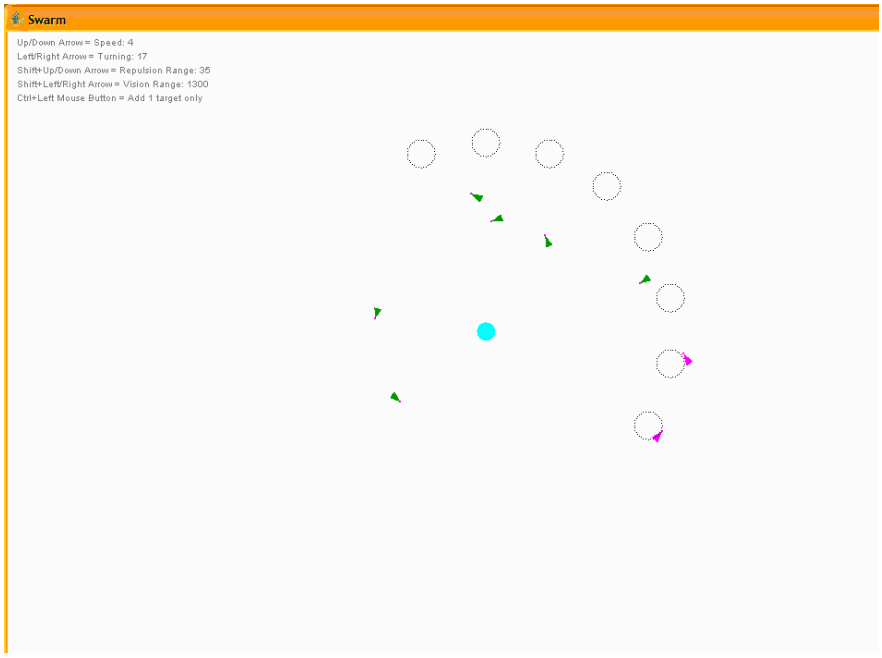


Figure 30. Swarm.java Program: 6 UAVs Are Taking Attack Positions

Once the UAVs have reached the 40-degree spacing around the inner circle, they turn toward the target. When a UAV touches the target, the UAV explodes, which is represented by light and dark blue pixels in Figure 31. After the UAV explodes, the orbit circle it possessed earlier will disappear from the screen as in Figures 31 and 32. In these figures, the second station keeper hits the target first followed by the rest of the swarm.

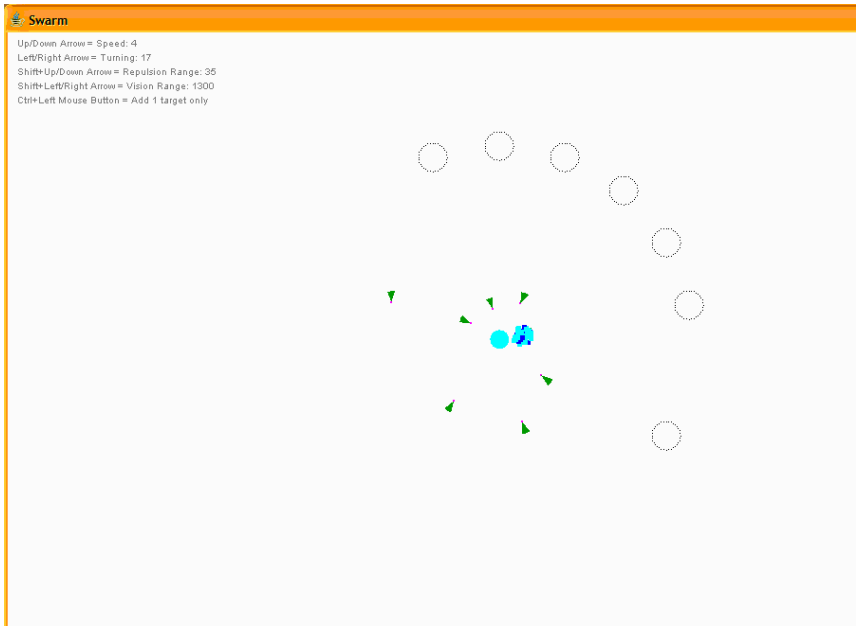


Figure 31. Swarm.java Program: All UAVs Are In Attack Positions

Figure 32. Swarm.java Program: Attack Sequence

In Figure 33, the last UAV has just turned toward the target. The delayed attack is due to timing errors and avoidance maneuvers when traveling to the inner circle from the orbit circle.

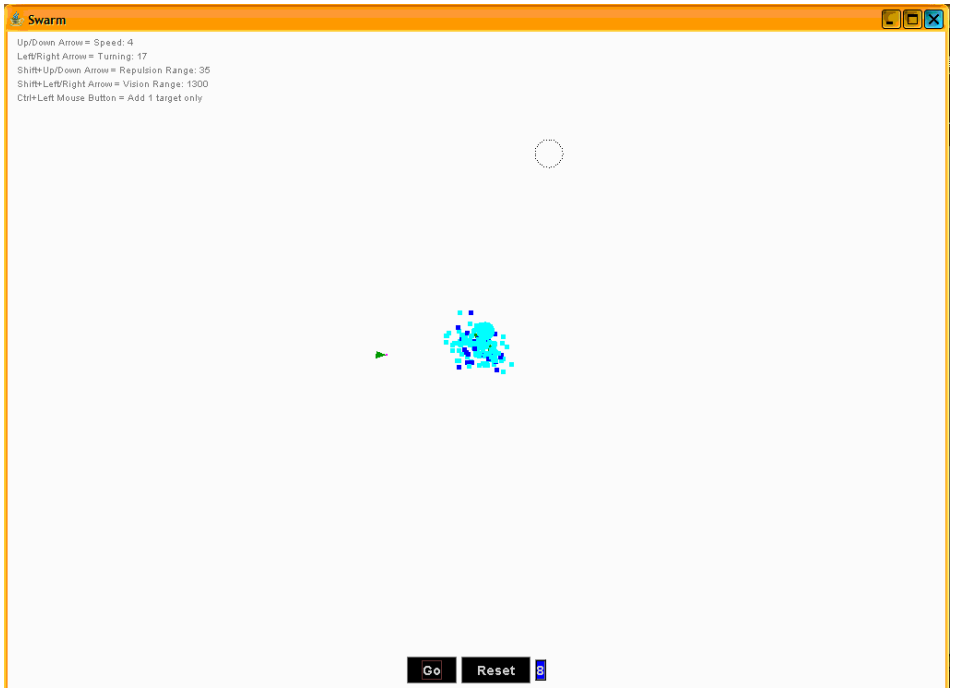


Figure 33. Swarm.java Program: Last UAV Attacks

When the last UAV hits the target in Figure 34, all of the orbit circles are also gone, indicating that all station keepers reached the target.

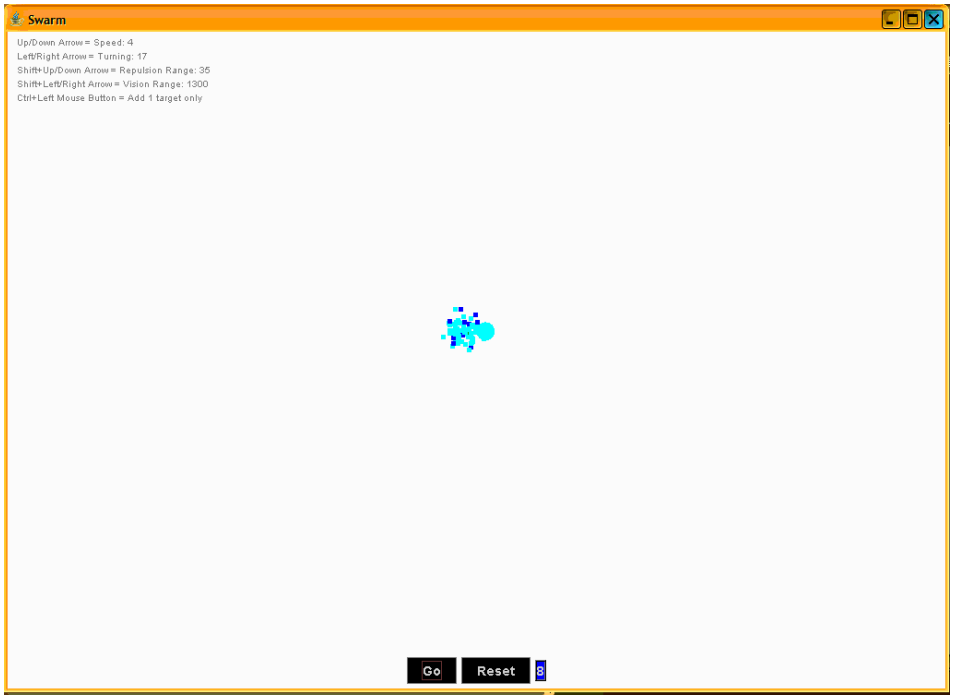


Figure 34. Swarm.java Program: Attack Completed

The program is designed for only one target. After the attack is completed, the target disappears, and the program is over. Additional targets can be placed around the screen, but obviously there are no UAVs left to coordinate an attack. To simulate another attack, the user must press the reset button at the bottom of the screen.

C. NEW RESULTS

The program has been modified to allow the UAVs to pass over the target without exploding. Surveying and taking pictures or releasing bombs and missiles are more realistic missions than the previous suicide mission. The goal is to allow the swarm of UAVs to seek out and attack the target one after another in a 'stream raid' fashion. Once all of the UAVs have passed over the target, the target disappears from the screen. The UAVs are now ready to attack the next target that appears. The population size is 8 UAVs for this scenario. For the original program, the population sizes of 1 through 18 were observed and the attack sequence was timed. The results are shown in Table 2.

UAV Population Size	Number of Trials	Average Attack Time
1	10	34.91
2	10	39.64
3	10	41.88
4	10	43.53
5	10	43.99
6	10	46.47
7	20	48.69
8	20	47.77
9	20	51.31
10	20	50.74
11	20	51.02
12	20	50.87
13	20	50.71
14	20	50.46
15	20	50.15
16	20	48.81
17	20	51.59
18	20	52.05

Table 2. UAV Population Size and Average Time of Attack

The duration of the attack increases with population size up to eight UAVs. The increase in population also increased the number of avoidance maneuvers required throughout the simulation, attributing to the longer time required to complete the required orbits around the station circle and attack. After the population size reaches nine UAVs, the duration of attack remained relatively constant despite additional UAVs. A population of eight provided an acceptable attack time for the size. Eight UAVs requires less avoidance maneuvers when finding the target. Eight UAVs also creates only eight orbit stations. Since each orbit station is spaced 20 degrees around the circle, these first eight stations occupy almost half of the outer circle from 0 degrees to 160 degrees. With this setup, there are no UAVs directly across the circle from each other. The heading for the targets after passing over the target for this simulation is arbitrary chosen to be south. The simulation can also be altered to send the UAVs to any other heading or allow them to pass through the target completely and continue on that attack heading without interfering with any UAVs still in the attack sequence.

The other parameters that are still adjustable in the program are speed, turning ability, repulsion range, and visual range. The initial, search, and station-keeping actions

of the UAVs are unaltered from the original program until the attack sequence. As depicted in Figure 35, all UAVs still become station owners and continue to orbit the circle until the attack order is issued.

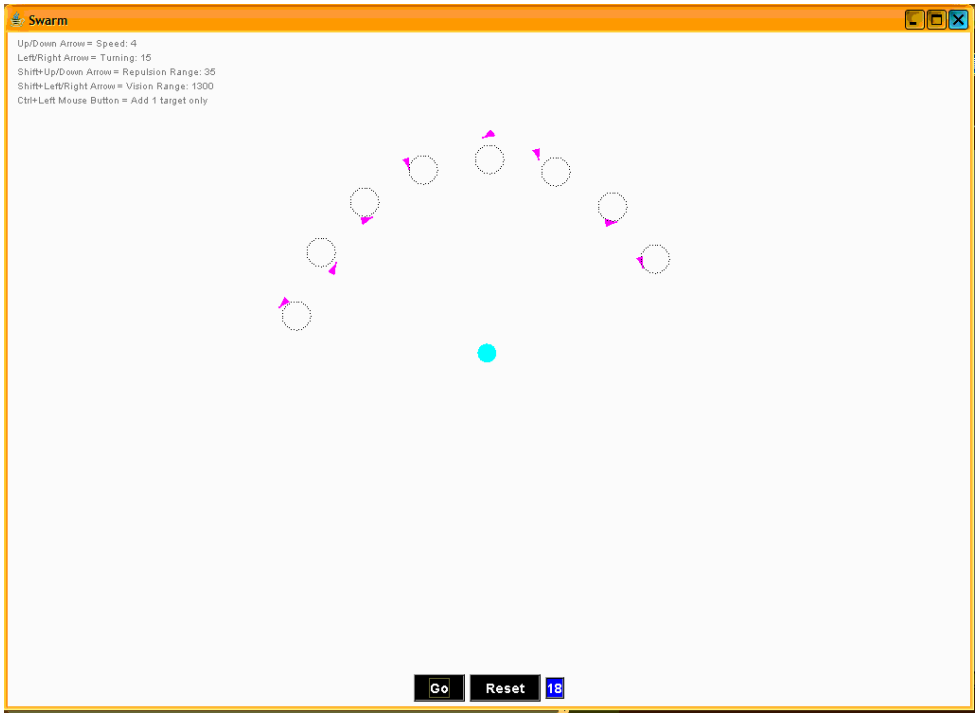


Figure 35. Modified Swarm.java Program: 8 Orbit Circles

The number of orbits necessary depends upon the time required for all UAVs to become station keepers and the amount of fuel allotted for the attack sequence. The first UAV to complete a specified number of orbits around the circle, analogous to the amount of fuel used, issues an order of attack. The attack order is locally communicated from one UAV to the neighboring UAVs. Each UAV leaves the orbit station at delayed intervals to sequentially pass over the target and prevent collisions. The UAV with the highest index number, which is the UAV that became a station keeper last, is the first to attack the target. When a UAV leaves the orbit circle, it returns to a green color. Figure 36 shows the beginning of the modified attack sequence from when the first UAV leaves an orbit circle.

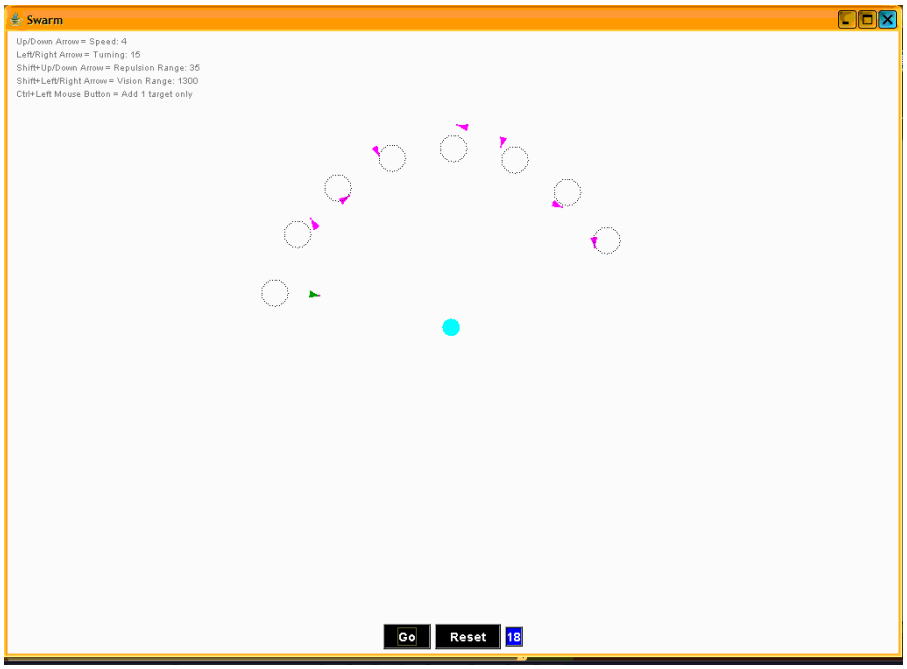


Figure 36. Modified Swarm.java Program: First UAV Heads Towards Target

At the appropriate delayed time, the second UAV leaves the orbit circle and heads toward the target as shown in Figure 37.

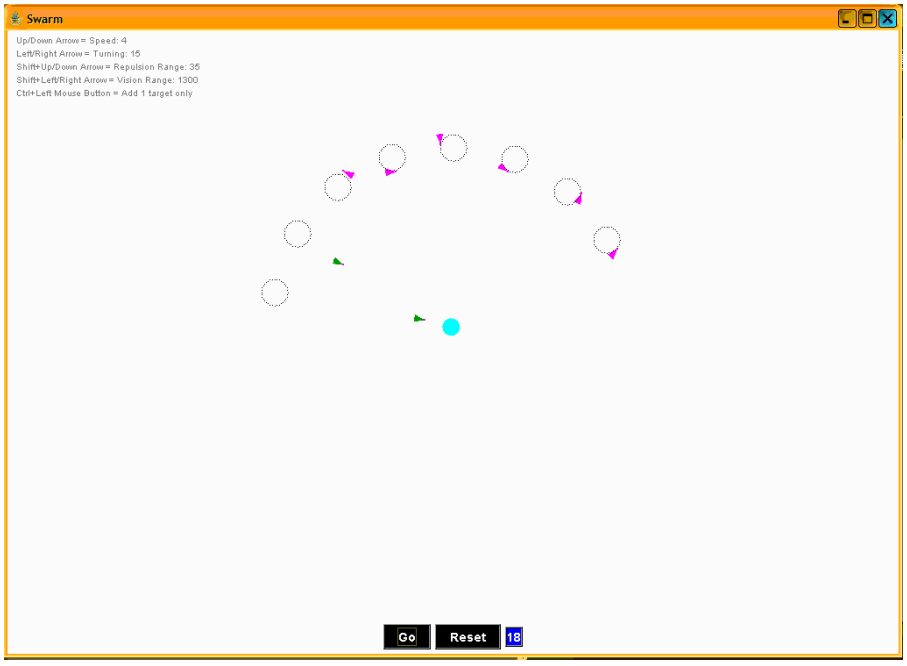


Figure 37. Modified Swarm.java Program: Second UAV Heads Toward Target

Each UAV passes over the target in an orderly manner and heads 'south.' The direction heading after the attack can be specified by the designer. Figure 38 and 39 show the UAVs passing over the target and turning to head south. Once a UAV has passed over the target, it enters search mode.

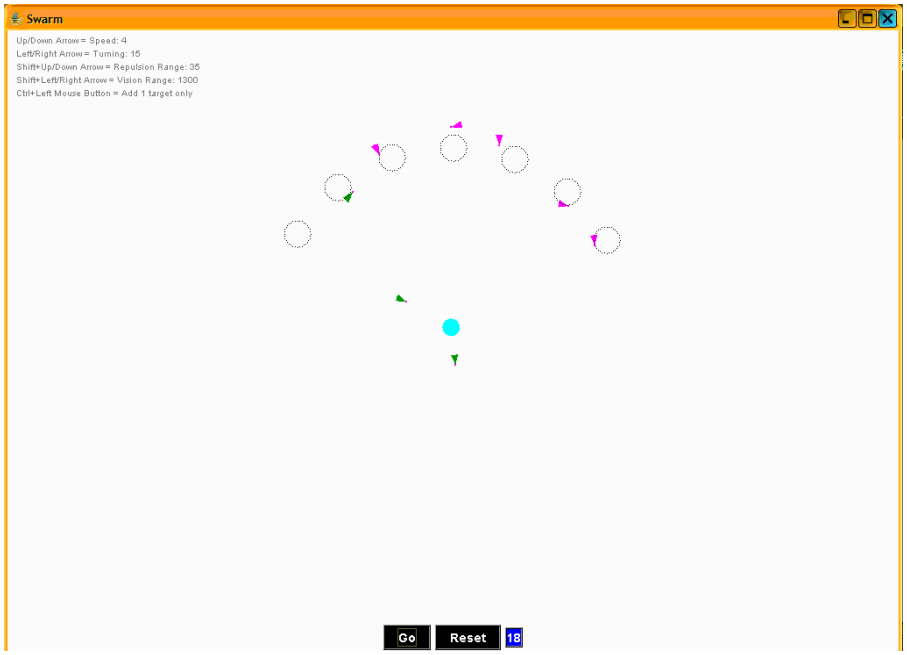


Figure 38. Modified Swarm.java Program: First UAV Passes Over Target

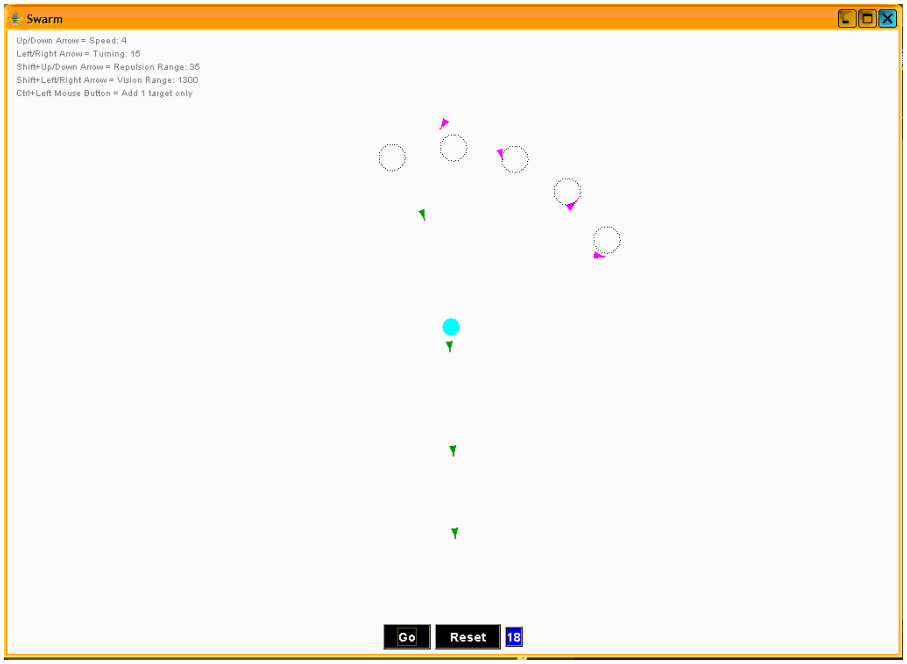


Figure 39. Modified Swarm.java Program: UAVs Continue Attack

Because the screen recycles UAVs from the bottom to the top, the UAVs going south appear again at the top of the screen as in Figure 40.

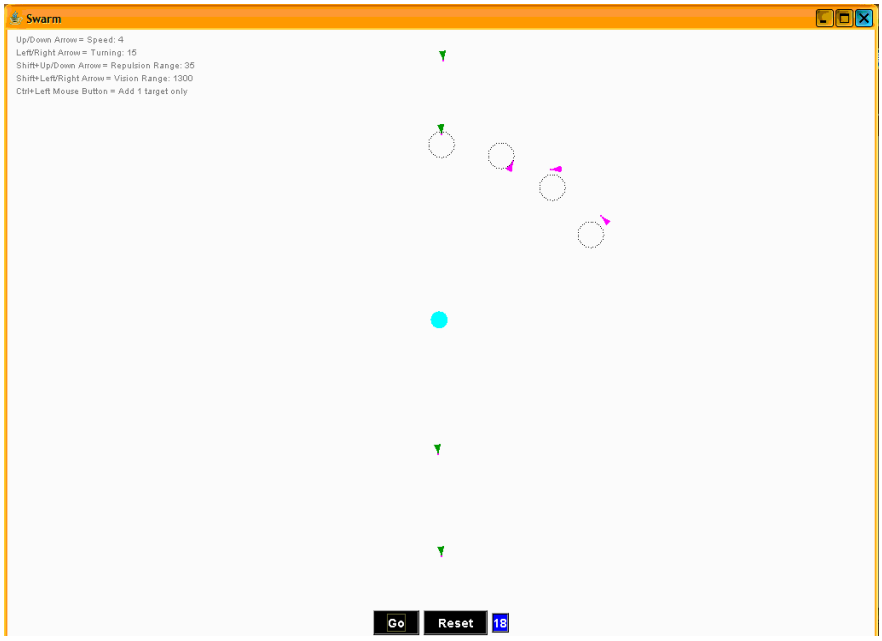


Figure 40. Modified Swarm.java Program: UAVs Continue Attack While Recycled UAVs Appear at the Top of the Screen

Realistically the UAVs would continue south without interfering with the attack. The simulated attack does not run smoothly because the attacking UAVs have to avoid the recycled UAVs. Figure 41 reveals that the UAV are in a staggered line because of previous avoidance maneuvers.

Figure 41. Modified Swarm.java Program: All UAVs Have Passed Over the Target and Head South Until Another Target is Found

The UAVs continue south until another target is detected (the user must place another target on the screen). Once the target is placed, the UAVs again head toward the inner circle search for the orbit circles as in Figure 42. In Figure 43, a UAV finds the first orbit circle for the second attack sequence.

Figure 42. Modified Swarm.java Program: New Target is Detected by the Swarm

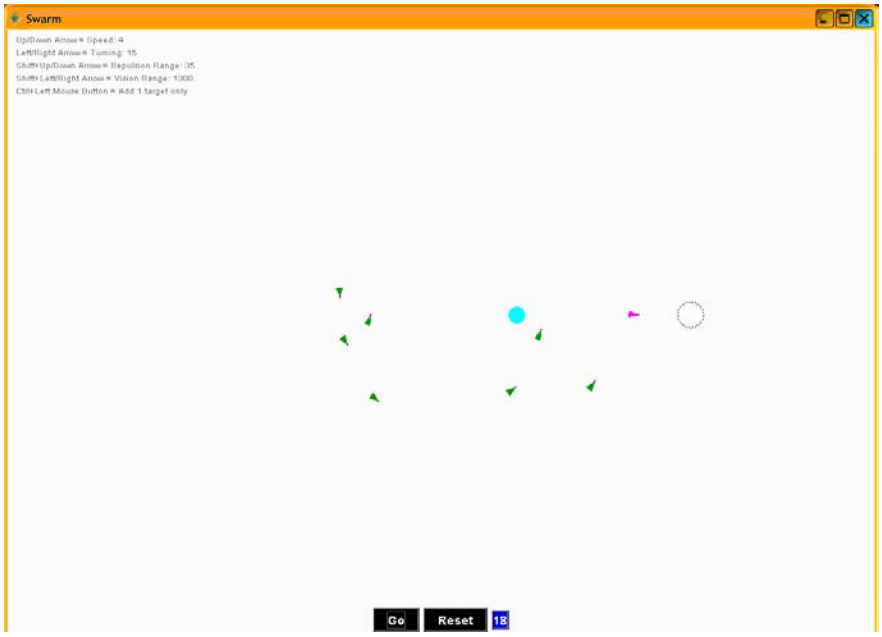


Figure 43. Modified Swarm.java Program: UAV Finds First Orbit Circle

The UAVs all become station keepers again and carry out the same sequence as described for the first attack. They continuously search and attack targets for the rest of the simulation.

D. CONCLUSIONS

The linear approach to an attack sequence allows for a smoother operation. The swarm of UAVs proceeds in a realistic manner. The UAVs maintain a constant velocity, and the simulation incorporates the aerodynamic capabilities of the UAV by setting a maximum turn angle. Most importantly, they avoid each other to prevent collisions which would compromise their ability to complete a mission.

The modified program still needs more improvement because it only works with one stationary target at a time. Other attacking scenarios need to be considered such as attacking a moving target. The swarm should be able to handle more than one target. It would be beneficial to attack the targets by dividing the swarm for the attack sequence and regrouping afterward. If the swarm does not divide, then it should also have the ability to decide which target is more important. The swarm could then pursue the most important target first and then reevaluate the situation after the attack to determine if the other target should be attacked. A three-dimensional version of the program could be created to include elevation changes. The program could also include the simulated loss

of a UAV. On a real battlefield, a UAV will only drop out of the swarm if it is damaged by the enemy or if it encounters mechanical problems.

The program could also include adjustments to the computational ability of the UAVs. Obviously a real UAV will have a distance and time limit because of fuel, but it can at least remain airborne for more than one attack sequence. The simulated swarm should have the ability to monitor fuel levels to have control over determining when the swarm should return to base.

Overall, the linear algorithm performed well, and the program allows for more flexibility in design. In the next chapter, the linear algorithm is compared to the PSO algorithm, and conclusions and future work are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SUMMARY

A. PSO VERSUS LINEAR ALGORITHM

Through flock simulation and the derivation of PSO, scientists discovered that a synchronous flock is not essential.[2] The simulated synchronized flock limits the scope of the group because it does not allow of individual exploration of the area. The flock has to tightly travel together; in order to search the area thoroughly, the entire flock would have to go over all possible locations. By allowing individuals to travel slightly outside the group, the group covers a larger search area at one time. As they identify the individual best found positions thus far, the group is able to discover the target faster. For a group to cooperate and achieve goals such as finding a target, the group must communicate. Therefore, communication, rather than synchronization, is necessary for success.[2]

The current PSO algorithm applies to weightless particles in multiple dimensions. The PSO algorithm can offer the advantage of finding the pattern in almost any problem space to reach a solution, but the current sequence can dead end and restart in a new position. It is a waste of computation time and resources to create an algorithm that would have a swarm of UAVs pursue a direction only to find it is the wrong path. If the target cannot be reached from the current path of the swarm, the PSO algorithm's solution is to start over. The swarm needs more guidance and a process to get out of a dead-end situation and back on track. With further research and improvements, the PSO algorithm can be applied to real objects limited to three dimensions.

PSO is focused on minimizing error between the particles and the target. In addition to changing the particle's direction to head toward the target, the algorithm accelerates the particles. When applying PSO to real flying objects, the constant speed changes are the main drawback. Actual UAVs should maintain a constant velocity to operate in a stable and controlled manner to prevent chaos and collisions. The constant velocity will also increase fuel efficiency and decrease strain on the platform. Although the PSO method is not practical, the central idea of minimizing error is completely applicable to UAVs. When the target location is known, error minimization is a valuable tool.

Compared to the PSO, the linear algorithm produces the most realistic results. The linear algorithm incorporates the ideas that have performed well in the PSO. The swarm does not have to move synchronously, and the UAVs move toward the target by minimizing the error in their position from the target. The error is minimized in a linear fashion since the velocity of the UAV remains constant. Linearity produces great results, and the simulated UAVs are able to find the target quickly and efficiently. The program also handles the UAVs as objects that occupy space. Each UAV has a threshold boundary distance, so they will avoid each other if they get too close. These movements allow the swarm to move toward a destination in space without collisions.

Since the swarm does not travel in formation, the UAVs need to regroup once a target is found. The orbit stations around the target provide organization before the attack. While orbiting, the UAVs can communicate and coordinate when the attack will occur. The orbit circles are also far enough from possible dangerous areas surrounding the target. The simulation shows the distance to be small relative to the size of the target and UAVs, but the radius of the circle is adjustable. Overall the linear algorithm can be more easily simulated and applied to realistic missions on a larger scale.

B. FUTURE WORK

The linear algorithm simulation can be improved according to the suggestions in the conclusion of Chapter 4. The simulation focuses on the swarm's motions, so it does not include all aspects needed for autonomous control. A major part of autonomous control is movement, but the maneuvers are in response to the situation surrounding the UAV. The simulation only includes three inputs that affect movement: the location of the target, whether another UAV has crossed the threshold boundary distance, and the predetermined attack sequence of finding orbit circles. The simulation does not include how the location of the target is determined by the receiver, how the distance between UAV is determined, and how the first orbit station is determined. The ability to attack a target is dependent on the UAV's ability to detect and locate the target.

The sensors on the UAVs need to be incorporated into the design of the swarm. Sensors are needed for target classification or identification and threat evaluation. The swarm is attempting to find targets that are difficult to locate with simple sensors. The sensors on each UAV can create a sensor network used to find the low-probability-of-

intercept (LPI) emitters. The UAVs will have the computational capacity to process the signals from LPI emitters to determine their location.

Another issue not addressed in the linear method simulation is swarm communication while in a search mode. The main advantage of a swarm is the sharing of information. This swarm passes along information such as which UAV is on which orbit circle and when they should leave the orbit circles to attack. Since the swarm is concerned with excess communication, they do not communicate during the search mode. The swarm does not travel in formation, so if one UAV does not recognize that a target is found, the others are not programmed to relay the information. This problem is not illustrated in the simulation because the target location is assumed to found by all UAVs through sensors. The UAVs should be able to communicate when a target is identified so that the entire swarm can begin the attack sequence.

The simulation demonstrates the movement capabilities of a swarm at a fundamental level. Improvements can be made to the swarm movements and coordinated strikes along with individual swarm capabilities. Since maneuvers are based off of the information the UAV has gathered, the best data available to the UAV will produce the best movements for the situation. The sensor and communication technological improvements will offer more performance advantages to the swarm.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MATLAB PSOT TOOLBOX

The MATLAB PSOt toolbox was developed by Brian Birge while at North Carolina State University.[2] The zipfile can be downloaded at:

http://www4.ncsu.edu/~bkbirge/PSO/PSOt.exe

The PSO function is described below:

```
  PSO.M
  a generic particle swarm optimizer
  to find the minimum or maximum of any
  MISO matlab function

  Brian Birge
  Rev 1.0
  1/1/3

  Usage:
     [optOUT]=PSO(functname,D)
    or:
     [optOUT,tr,te]=PSO(functname,D,VarRange,minmax,PSOparams)

  Inputs:
     functname - string of matlab function to optimize
     D - # of inputs to the function (dimension of problem)

  Optional Inputs:
     VarRange - matrix of ranges for each input variable, default -100
to 100, of form:
         [ min1 max1
           min2 max2
              ...
           minD maxD ]

     minmax - if 0 then funct is minimized, if 1 then funct maximized,
default=0

     PSOparams - PSO parameters
       P(1) - Epochs between updating display, works with P(13), de-
fault = 25.
       P(2) - Maximum number of iterations (epochs) to train, default =
2000.
       P(3) - population size, default = 20
       P(4) - maximum particle velocity, default = 4
       P(5) - acceleration const 1 (local best influence), default = 2
       P(6) - acceleration const 2 (global best influence), default = 2
       P(7) - Initial inertia weight, default = 0.9
       P(8) - Final inertia weight, default = 0.2
       P(9)- Iteration (epoch) by which inertial weight should be at
final value, default = 1500
       P(10)- randomization flag (flagg), for PSO conforming to litera-
ture = 2, default = 2:
```

```
                flagg = 0, same random numbers used for each particle
(different at each epoch - least randomness)
                flagg = 1, separate randomized numbers for each parti-
cle at each epoch
                flagg = 2, separate random #'s at each component of
each particle at each epoch (most randomness)
       P(11)- minimum global error gradient, if abs(Gbest(i+1)-
Gbest(i)) < gradient over
                 certain length of epochs, terminate run, default =
1e-9
       P(12)- epochs before error gradient criterion terminates run,
default = 50
                 i.e., if the SSE does not change over 50 epochs, quit
program
       P(13)- plot flag, shows progress display if =1, nothing other-
wise, default = 1

  Outputs:
     optOUT - optimal inputs and associated min/max output of function,
of form:
          [ bestin1
            bestin2
              ...
            bestinD
            bestOUT ]

  Optional Outputs:
     tr    - Gbest at every iteration, traces flight of swarm
     te    - epochs to train, returned as a vector 1:endepoch

  Example:  out=pso('f6',2)

  See Also: TRAINPSO
```

The ‘DemoTrainPSO’ file is from the toolbox and is an example of how to train the neural network to an XOR function. All functions used within the demo can be found in the toolbox. Minor changes such as making the variable ‘tr’ a global variable were added to adjust for the different versions of MATLAB.

```
% DemoTrainPSO.m
% little file to test out the pso optimizer for nnet training
% trains to the XOR function
%
% note: this does *not* minimize the test set function
% rather it tries to train a neural net to approximate the
% test set function
%
% Brian Birge
% Rev 1.0
% 1/1/3

clear all
close all
clc
help demotrainpso
global tr
%     Training parameters are:
%    TP(1) - Epochs between updating display, default = 100.
%     TP(2) - Maximum number of iterations (epochs) to train, default =
2000.
%    TP(3) - Sum-squared error goal, default = 0.02.
%    TP(4) - population size, default = 20
%    TP(5) - maximum particle velocity, default = 4
%    TP(6) - acceleration constant 1, default = 2
%    TP(7) - acceleration constant 2, default = 2
%    TP(8) - Initial inertia weight, default = 0.9
%    TP(9) - Final inertia weight, default = 0.2
%     TP(10)- Iteration (epoch) by which inertial weight should be at
final value, default = 1500
%     TP(11)- maximum initial network weight absolute value, default =
100
%    TP(12)- randomization flag (flagg), default = 2:
%                           flagg = 0, same random numbers used for each
particle (different at each epoch - least random)
%                         flagg = 1, separate randomized numbers for each
particle at each epoch
%                        flagg = 2, separate random #'s at each component
of each particle at each epoch (most random)
%    TP(13)- minimum global error gradient (if SSE(i+1)-SSE(i) < gradi-
ent over
%                certain length of epochs, terminate run, default = 1e-9
%    TP(14)- epochs before error gradient criterion terminates run, de-
fault = 200
%                 i.e., if the SSE does not change over 200 epochs, quit
program

nntwarn off
```

```
epdt=25;
maxep=1000;
reqerr=0.02;
maxneur=30;
popsz=20;
maxvel=4;
acnst1=2;
acnst2=2;
inwt1=.9;
inwt2=0.2;
endepoch=1500;
maxwt=100;
cnt=0; % counter for neuron architecture

% Training parameters, change these to experiment with PSO performance
% type help trainpso to find out what they do
TP=[epdt,maxep,reqerr,popsz,maxvel,acnst1,acnst2,inwt1,inwt2,endepoch,m
axwt,2,1e-9,200];

disp('-------------------------------------------------------------------
----------------------------------');
disp(' ');
disp('1. 1 hidden layer');
disp('2. 2 hidden layers');
disp('3. no hidden layers');
arch=input('  Pick a neural net architecture >');
disp(' ');
disp('1. Particle Swarm Optimization');
disp('2. Standard Backprop');
meth=input('  Pick training method >');
disp(' ');
disp('-------------------------------------------------------------------
----------------------------------');
disp(' ');

% XOR function test set
 P=[0,0;0,1;1,0;1,1]';
 T=[1;0;0;1]';
 minmax=[0,1;0,1];
 l1=0;
 l2=1;
tr(1)=99; % arbitrary choice of initial error just used to update # of
neurons

if arch==3
   [w1,b1]=initff(minmax,1,'tansig');
   if meth==1
      [w1,b1,te,tr]=trainpso(w1,b1,'tansig',P,T,TP);
   elseif meth==2
      [w1,b1,te,tr]=trainbp(w1,b1,'tansig',P,T);
   end
elseif arch==1
   while tr(end)>reqerr
    l1=l1+1;

    [w1,b1,w2,b2]=initff(minmax,l1,'tansig',1,'purelin');
```

```
    if meth==1

[w1,b1,w2,b2,te,tr]=trainpso(w1,b1,'tansig',w2,b2,'purelin',P,T,TP);
    elseif meth==2
      [w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'tansig',w2,b2,'purelin',P,T);
    end
    if l1>maxneur
       break
    end
   end
elseif arch==2
   while tr(end)>reqerr
    if l1>l2
       l2=l2+1;
    else
       l1=l1+1;
    end


[w1,b1,w2,b2,w3,b3]=initff(minmax,l2,'tansig',l1,'logsig',1,'purelin');

    if meth==1

[w1,b1,w2,b2,w3,b3,te,tr]=trainpso(w1,b1,'tansig',w2,b2,'logsig',w3,b3,
'purelin',P,T,TP);
    elseif meth==2

[w1,b1,w2,b2,w3,b3,te,tr]=trainbp(w1,b1,'tansig',w2,b2,'logsig',w3,b3,'
purelin',P,T);
    end
    if l1>maxneur
       break
    end
   end
end
```

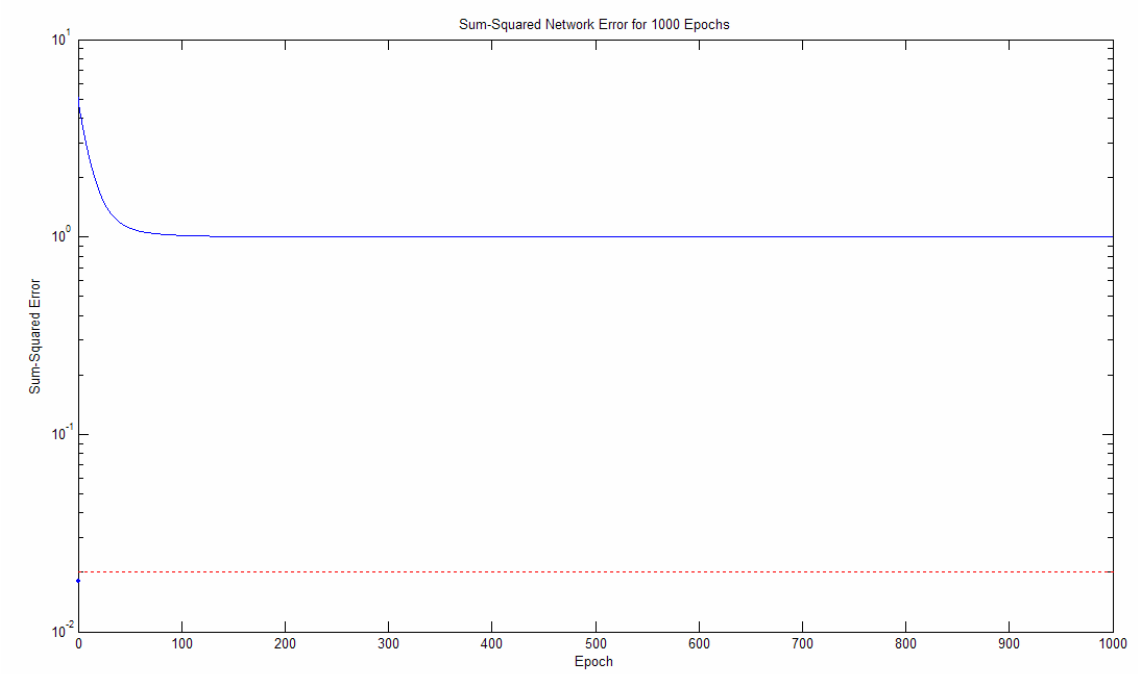
THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. EXAMPLE OF BACKPROPAGATION TRAINING

From Birge's program in [4], this example trains a neural network with one hidden layer for the XOR problem by using backpropagation. It illustrates how the algorithm is trying to converge to a sum mean squared error of 0.02, represented by the red dotted line in the plots. If the network has not reached the goal error of 0.02 by 1000 epochs or remains at the same error for an extended period of time, the training will start over with different initial weight values. For this example, the network was trained in 2688 epochs.

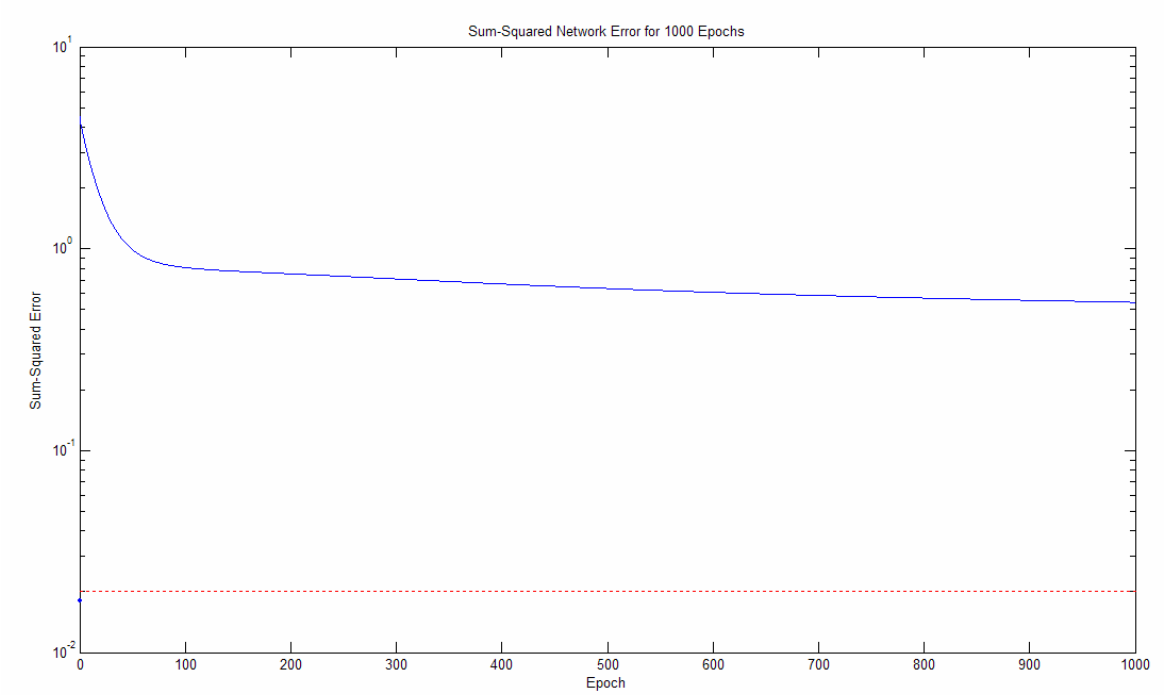
```
  DemoTrainPSO.m
  little file to test out the pso optimizer for nnet training
  trains to the XOR function

  note: this does *not* minimize the test set function
  rather it tries to train a neural net to approximate the
  test set function

  Brian Birge
  Rev 1.0
  1/1/3



--------------------------------------------------------------------------
----------------------------

1. 1 hidden layer
2. 2 hidden layers
3. no hidden layers
  Pick a neural net architecture >1

1. Particle Swarm Optimization
2. Standard Backprop
  Pick training method >2

--------------------------------------------------------------------------
----------------------------

TRAINBP: 0/1000 epochs, SSE = 5.08251.
TRAINBP: 25/1000 epochs, SSE = 1.51519.
TRAINBP: 50/1000 epochs, SSE = 1.1085.
TRAINBP: 75/1000 epochs, SSE = 1.03817.
TRAINBP: 100/1000 epochs, SSE = 1.01752.
TRAINBP: 125/1000 epochs, SSE = 1.00871.
TRAINBP: 150/1000 epochs, SSE = 1.00436.
TRAINBP: 175/1000 epochs, SSE = 1.00212.
TRAINBP: 200/1000 epochs, SSE = 1.00095.
TRAINBP: 225/1000 epochs, SSE = 1.00034.
TRAINBP: 250/1000 epochs, SSE = 1.00001.
```

```
TRAINBP: 275/1000 epochs, SSE = 0.999834.
TRAINBP: 300/1000 epochs, SSE = 0.999734.
TRAINBP: 325/1000 epochs, SSE = 0.999674.
TRAINBP: 350/1000 epochs, SSE = 0.999633.
TRAINBP: 375/1000 epochs, SSE = 0.999603.
TRAINBP: 400/1000 epochs, SSE = 0.999576.
TRAINBP: 425/1000 epochs, SSE = 0.999552.
TRAINBP: 450/1000 epochs, SSE = 0.999528.
TRAINBP: 475/1000 epochs, SSE = 0.999503.
TRAINBP: 500/1000 epochs, SSE = 0.999478.
TRAINBP: 525/1000 epochs, SSE = 0.999451.
TRAINBP: 550/1000 epochs, SSE = 0.999423.
TRAINBP: 575/1000 epochs, SSE = 0.999393.
TRAINBP: 600/1000 epochs, SSE = 0.999361.
TRAINBP: 625/1000 epochs, SSE = 0.999327.
TRAINBP: 650/1000 epochs, SSE = 0.999292.
TRAINBP: 675/1000 epochs, SSE = 0.999254.
TRAINBP: 700/1000 epochs, SSE = 0.999214.
TRAINBP: 725/1000 epochs, SSE = 0.999171.
TRAINBP: 750/1000 epochs, SSE = 0.999126.
TRAINBP: 775/1000 epochs, SSE = 0.999077.
TRAINBP: 800/1000 epochs, SSE = 0.999026.
TRAINBP: 825/1000 epochs, SSE = 0.998971.
TRAINBP: 850/1000 epochs, SSE = 0.998912.
TRAINBP: 875/1000 epochs, SSE = 0.99885.
TRAINBP: 900/1000 epochs, SSE = 0.998783.
TRAINBP: 925/1000 epochs, SSE = 0.998711.
TRAINBP: 950/1000 epochs, SSE = 0.998635.
TRAINBP: 975/1000 epochs, SSE = 0.998553.
TRAINBP: 1000/1000 epochs, SSE = 0.998464.

TRAINBP: Network error did not reach the error goal.
  Further training may be necessary, or try different
  initial weights and biases and/or more hidden neurons.
```



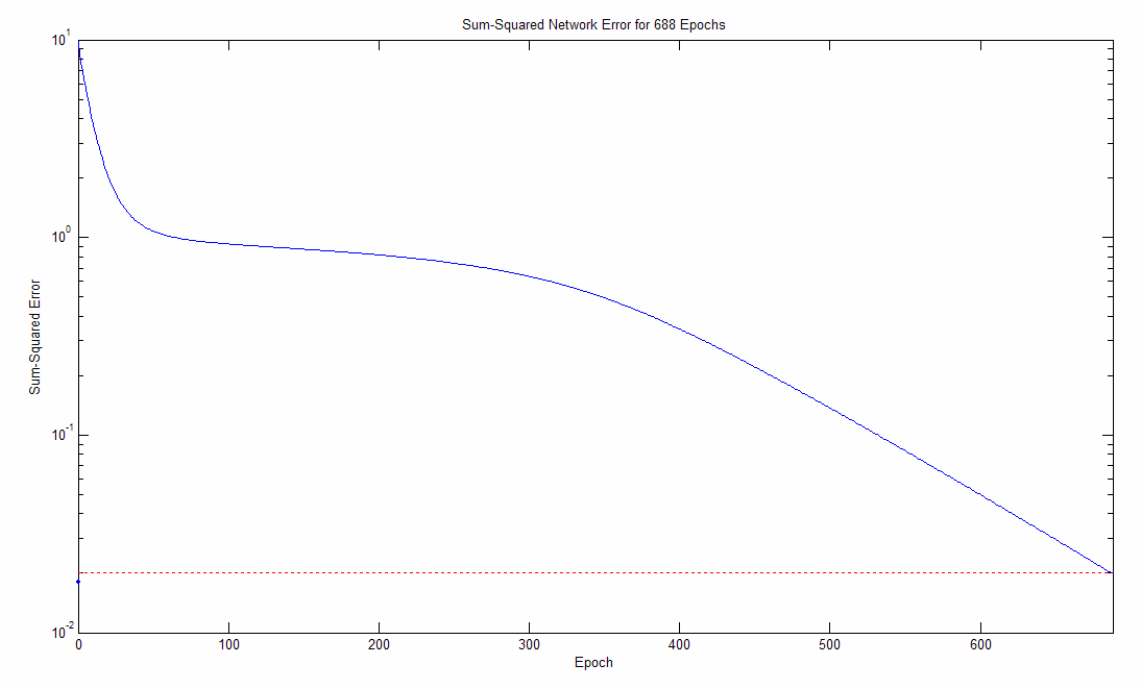

```
TRAINBP: 0/1000 epochs, SSE = 4.52412.
TRAINBP: 25/1000 epochs, SSE = 1.54958.
TRAINBP: 50/1000 epochs, SSE = 0.990459.
TRAINBP: 75/1000 epochs, SSE = 0.850558.
TRAINBP: 100/1000 epochs, SSE = 0.805685.
TRAINBP: 125/1000 epochs, SSE = 0.785602.
TRAINBP: 150/1000 epochs, SSE = 0.772436.
TRAINBP: 175/1000 epochs, SSE = 0.761228.
TRAINBP: 200/1000 epochs, SSE = 0.750545.
TRAINBP: 225/1000 epochs, SSE = 0.739974.
TRAINBP: 250/1000 epochs, SSE = 0.729421.
TRAINBP: 275/1000 epochs, SSE = 0.718898.
TRAINBP: 300/1000 epochs, SSE = 0.708459.
TRAINBP: 325/1000 epochs, SSE = 0.698172.
TRAINBP: 350/1000 epochs, SSE = 0.688107.
TRAINBP: 375/1000 epochs, SSE = 0.67833.
TRAINBP: 400/1000 epochs, SSE = 0.668894.
TRAINBP: 425/1000 epochs, SSE = 0.659843.
TRAINBP: 450/1000 epochs, SSE = 0.651204.
TRAINBP: 475/1000 epochs, SSE = 0.642992.
TRAINBP: 500/1000 epochs, SSE = 0.635211.
TRAINBP: 525/1000 epochs, SSE = 0.627857.
TRAINBP: 550/1000 epochs, SSE = 0.620916.
TRAINBP: 575/1000 epochs, SSE = 0.614372.
TRAINBP: 600/1000 epochs, SSE = 0.608203.
TRAINBP: 625/1000 epochs, SSE = 0.602386.
TRAINBP: 650/1000 epochs, SSE = 0.596897.
TRAINBP: 675/1000 epochs, SSE = 0.591712.
TRAINBP: 700/1000 epochs, SSE = 0.586805.
TRAINBP: 725/1000 epochs, SSE = 0.582153.
TRAINBP: 750/1000 epochs, SSE = 0.577734.
TRAINBP: 775/1000 epochs, SSE = 0.573526.
```

```
TRAINBP: 800/1000 epochs, SSE = 0.569508.
TRAINBP: 825/1000 epochs, SSE = 0.565661.
TRAINBP: 850/1000 epochs, SSE = 0.561966.
TRAINBP: 875/1000 epochs, SSE = 0.558407.
TRAINBP: 900/1000 epochs, SSE = 0.554966.
TRAINBP: 925/1000 epochs, SSE = 0.551628.
TRAINBP: 950/1000 epochs, SSE = 0.548378.
TRAINBP: 975/1000 epochs, SSE = 0.545201.
TRAINBP: 1000/1000 epochs, SSE = 0.542084.

TRAINBP: Network error did not reach the error goal.
  Further training may be necessary, or try different
  initial weights and biases and/or more hidden neurons.
```




```
TRAINBP: 0/1000 epochs, SSE = 9.65683.
TRAINBP: 25/1000 epochs, SSE = 1.6681.
TRAINBP: 50/1000 epochs, SSE = 1.07482.
TRAINBP: 75/1000 epochs, SSE = 0.968458.
TRAINBP: 100/1000 epochs, SSE = 0.926412.
TRAINBP: 125/1000 epochs, SSE = 0.897892.
TRAINBP: 150/1000 epochs, SSE = 0.872191.
TRAINBP: 175/1000 epochs, SSE = 0.845454.
TRAINBP: 200/1000 epochs, SSE = 0.815634.
TRAINBP: 225/1000 epochs, SSE = 0.781185.
TRAINBP: 250/1000 epochs, SSE = 0.740582.
TRAINBP: 275/1000 epochs, SSE = 0.692221.
TRAINBP: 300/1000 epochs, SSE = 0.634751.
TRAINBP: 325/1000 epochs, SSE = 0.567952.
TRAINBP: 350/1000 epochs, SSE = 0.493859.
TRAINBP: 375/1000 epochs, SSE = 0.417085.
TRAINBP: 400/1000 epochs, SSE = 0.343367.
TRAINBP: 425/1000 epochs, SSE = 0.277244.
```

```
TRAINBP: 450/1000 epochs, SSE = 0.220864.
TRAINBP: 475/1000 epochs, SSE = 0.174358.
TRAINBP: 500/1000 epochs, SSE = 0.136771.
TRAINBP: 525/1000 epochs, SSE = 0.106777.
TRAINBP: 550/1000 epochs, SSE = 0.0830447.
TRAINBP: 575/1000 epochs, SSE = 0.0643837.
TRAINBP: 600/1000 epochs, SSE = 0.0497819.
TRAINBP: 625/1000 epochs, SSE = 0.0384025.
TRAINBP: 650/1000 epochs, SSE = 0.0295648.
TRAINBP: 675/1000 epochs, SSE = 0.0227212.
TRAINBP: 688/1000 epochs, SSE = 0.0198019.
```



THIS PAGE INTENTIONALLY LEFT BLANK

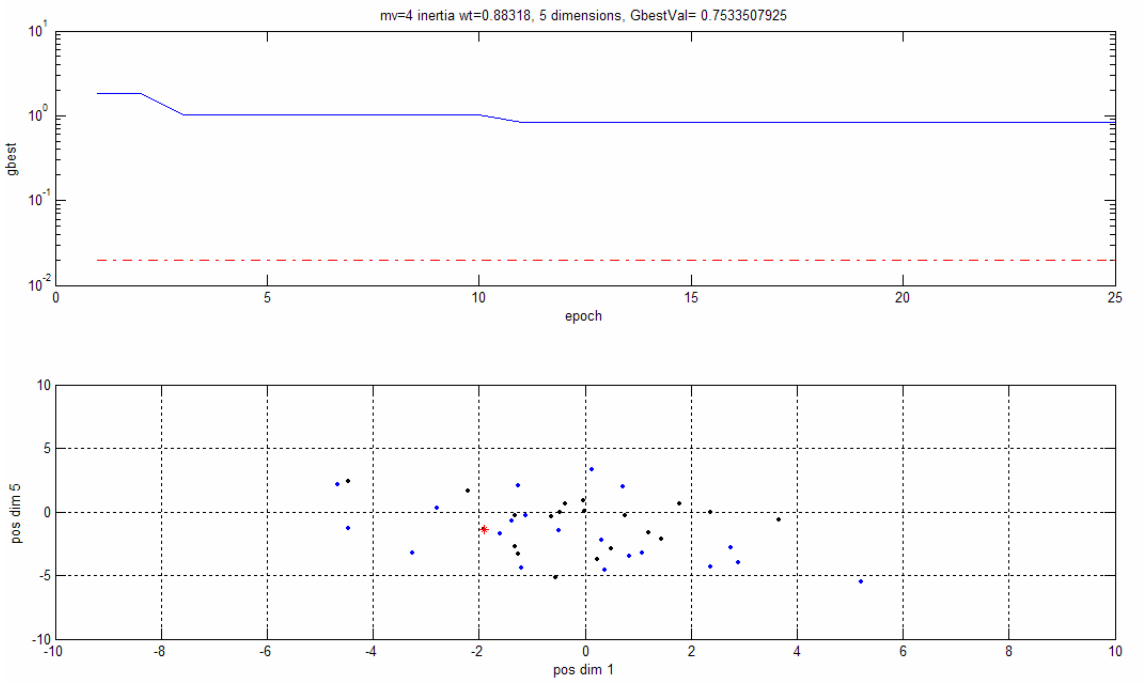
APPENDIX C. EXAMPLE OF PARTICLE SWARM OPTIMIZATION TRAINING

From Birge's program in [4], this example trains a neural network with one hidden layer for the XOR problem by using Particle Swarm Optimization. It can be compared to the backpropagation example. It illustrates how the algorithm is trying to converge to a sum mean squared error of 0.02, represented by the red dotted line in the plots. If the network has not reached the goal error of 0.02 by 1000 epochs or remains at the same error for an extended period of time, the training will start over with different initial weight values. The blue dots represent the particle locations, and the black dots represent each particle's personal best position. The red crosshair represents the swarm's global best position. The final plot will show a magenta line to represent the movement of the global best position around the search space.

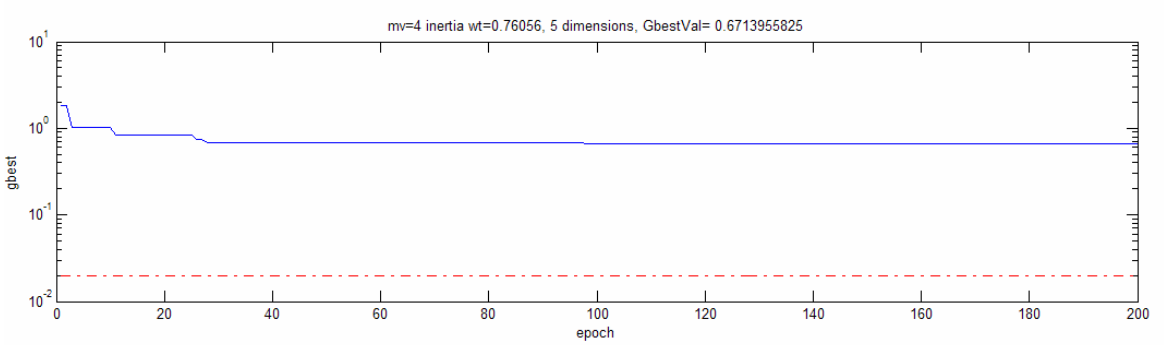
For this example, the network was trained in 1340 epochs.

```
  DemoTrainPSO.m
  little file to test out the pso optimizer for nnet training
  trains to the XOR function

  note: this does *not* minimize the test set function
  rather it tries to train a neural net to approximate the
  test set function

  Brian Birge
  Rev 1.0
  1/1/3



--------------------------------------------------------------------------
----------------------------

1. 1 hidden layer
2. 2 hidden layers
3. no hidden layers
  Pick a neural net architecture >1

1. Particle Swarm Optimization
2. Standard Backprop
  Pick training method >1

--------------------------------------------------------------------------
----------------------------
```

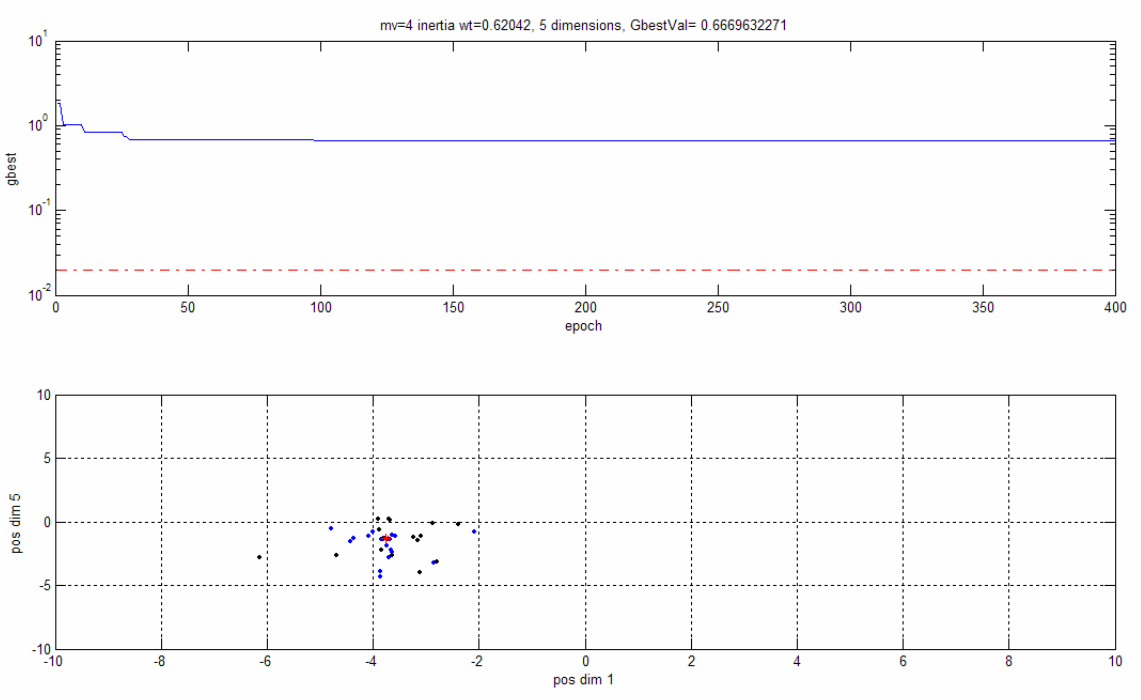
```
TRAINPSO: 25/1000 epochs,  gbest SSE = 0.7533507925
  mv  = 4,  iwt = 0.8831831832
```



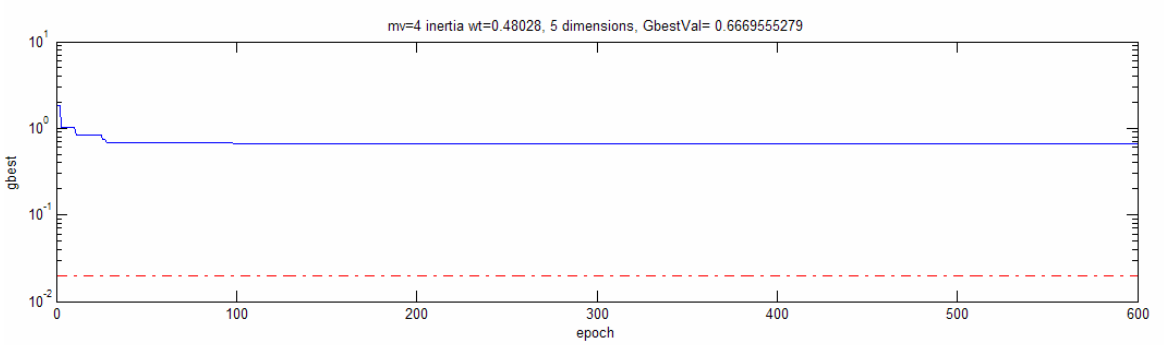
```
TRAINPSO: 50/1000 epochs,  gbest SSE = 0.6794204617
  mv  = 4,  iwt = 0.8656656657
TRAINPSO: 75/1000 epochs,  gbest SSE = 0.6794204617
  mv  = 4,  iwt = 0.8481481481
TRAINPSO: 100/1000 epochs,  gbest SSE = 0.6713955825
  mv  = 4,  iwt = 0.8306306306
TRAINPSO: 125/1000 epochs,  gbest SSE = 0.6713955825
  mv  = 4,  iwt = 0.8131131131
TRAINPSO: 150/1000 epochs,  gbest SSE = 0.6713955825
  mv  = 4,  iwt = 0.7955955956
TRAINPSO: 175/1000 epochs,  gbest SSE = 0.6713955825
  mv  = 4,  iwt = 0.7780780781
TRAINPSO: 200/1000 epochs,  gbest SSE = 0.6713955825
  mv  = 4,  iwt = 0.7605605606
```



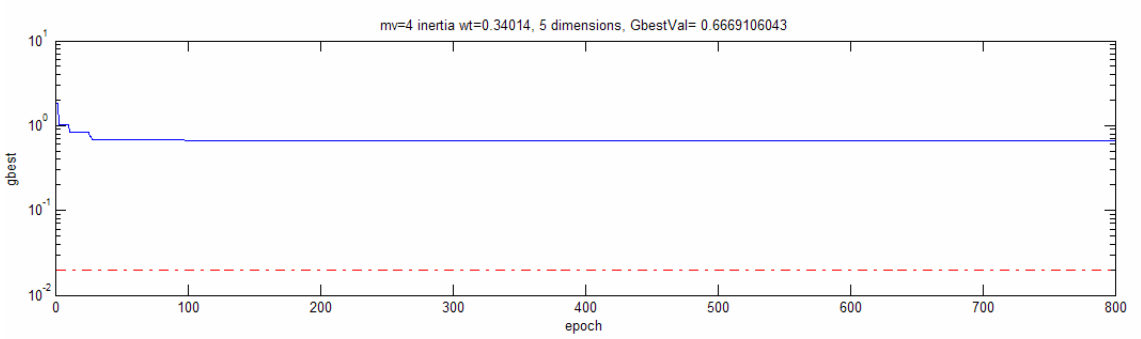
```
TRAINPSO: 225/1000 epochs,  gbest SSE = 0.6699003323
  mv  = 4,  iwt = 0.743043043
TRAINPSO: 250/1000 epochs,  gbest SSE = 0.6699003323
  mv  = 4,  iwt = 0.7255255255
TRAINPSO: 275/1000 epochs,  gbest SSE = 0.6696477244
  mv  = 4,  iwt = 0.708008008
TRAINPSO: 300/1000 epochs,  gbest SSE = 0.6696477244
  mv  = 4,  iwt = 0.6904904905
TRAINPSO: 325/1000 epochs,  gbest SSE = 0.6696477244
  mv  = 4,  iwt = 0.672972973
TRAINPSO: 350/1000 epochs,  gbest SSE = 0.6680743326
  mv  = 4,  iwt = 0.6554554555
TRAINPSO: 375/1000 epochs,  gbest SSE = 0.6673640869
  mv  = 4,  iwt = 0.6379379379
TRAINPSO: 400/1000 epochs,  gbest SSE = 0.6669632271
  mv  = 4,  iwt = 0.6204204204
```



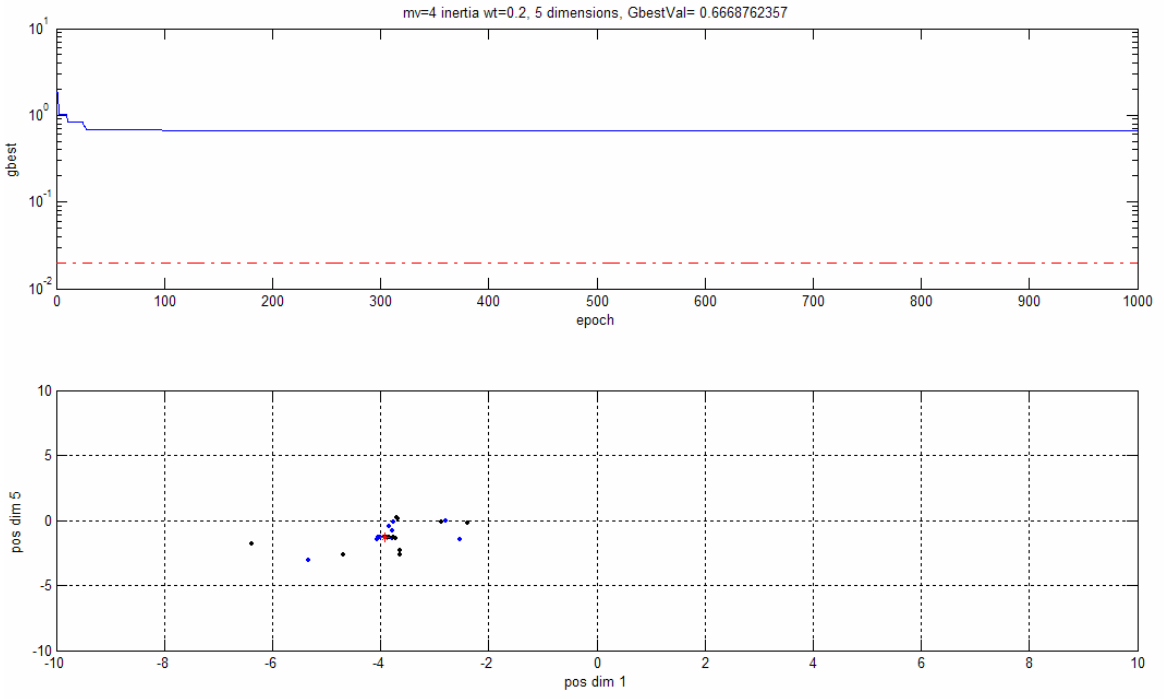
```
TRAINPSO: 425/1000 epochs,  gbest SSE = 0.6669562435
  mv  = 4,  iwt = 0.6029029029
TRAINPSO: 450/1000 epochs,  gbest SSE = 0.6669562435
  mv  = 4,  iwt = 0.5853853854
TRAINPSO: 475/1000 epochs,  gbest SSE = 0.6669559919
  mv  = 4,  iwt = 0.5678678679
TRAINPSO: 500/1000 epochs,  gbest SSE = 0.6669559918
  mv  = 4,  iwt = 0.5503503504
TRAINPSO: 525/1000 epochs,  gbest SSE = 0.6669556083
  mv  = 4,  iwt = 0.5328328328
TRAINPSO: 550/1000 epochs,  gbest SSE = 0.6669555494
  mv  = 4,  iwt = 0.5153153153
TRAINPSO: 575/1000 epochs,  gbest SSE = 0.666955536
  mv  = 4,  iwt = 0.4977977978
TRAINPSO: 600/1000 epochs,  gbest SSE = 0.6669555279
  mv  = 4,  iwt = 0.4802802803
```



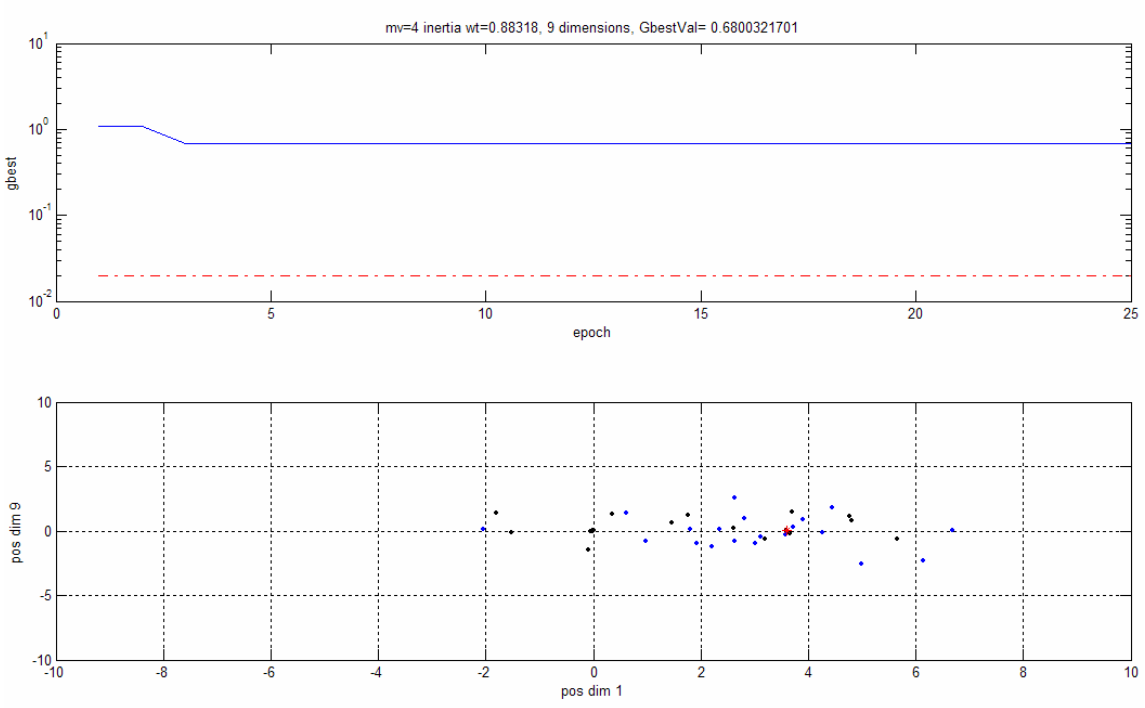
```
TRAINPSO: 625/1000 epochs,  gbest SSE = 0.6669555202
  mv  = 4,  iwt = 0.4627627628
TRAINPSO: 650/1000 epochs,  gbest SSE = 0.6669554313
  mv  = 4,  iwt = 0.4452452452
TRAINPSO: 675/1000 epochs,  gbest SSE = 0.6669305226
  mv  = 4,  iwt = 0.4277277277
TRAINPSO: 700/1000 epochs,  gbest SSE = 0.6669208268
  mv  = 4,  iwt = 0.4102102102
TRAINPSO: 725/1000 epochs,  gbest SSE = 0.6669160247
  mv  = 4,  iwt = 0.3926926927
TRAINPSO: 750/1000 epochs,  gbest SSE = 0.6669147554
  mv  = 4,  iwt = 0.3751751752
TRAINPSO: 775/1000 epochs,  gbest SSE = 0.6669146701
  mv  = 4,  iwt = 0.3576576577
TRAINPSO: 800/1000 epochs,  gbest SSE = 0.6669106043
  mv  = 4,  iwt = 0.3401401401
```



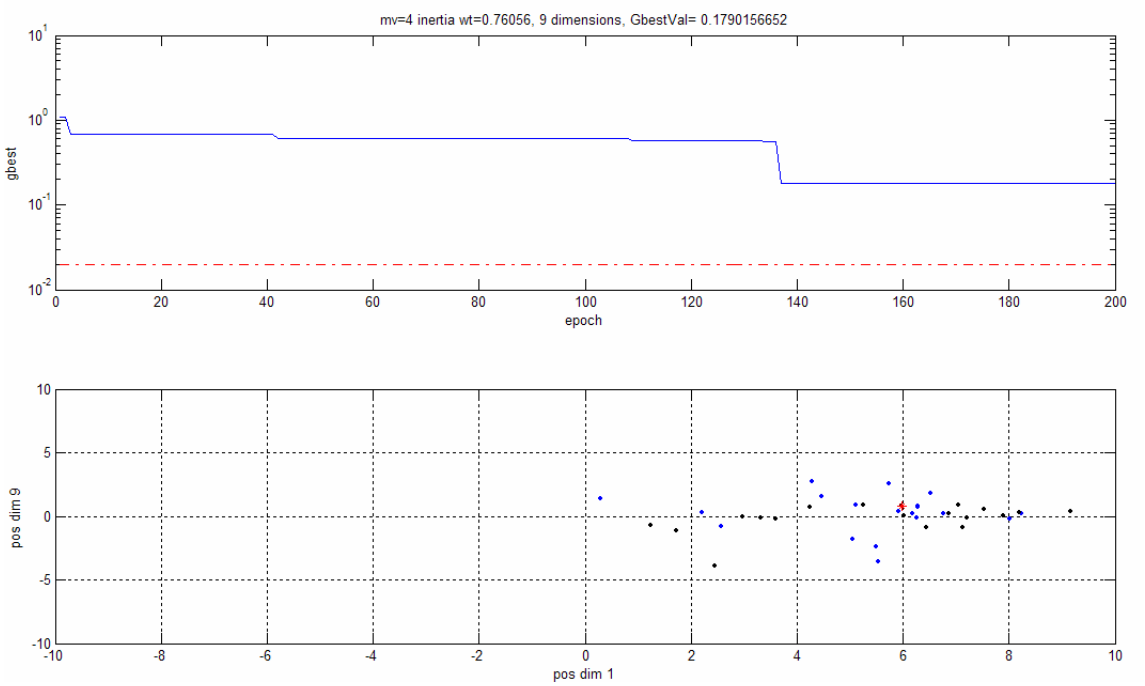
```
TRAINPSO: 825/1000 epochs,  gbest SSE = 0.6669098761
  mv  = 4,  iwt = 0.3226226226
TRAINPSO: 850/1000 epochs,  gbest SSE = 0.6669083356
  mv  = 4,  iwt = 0.3051051051
TRAINPSO: 875/1000 epochs,  gbest SSE = 0.6669082001
  mv  = 4,  iwt = 0.2875875876
TRAINPSO: 900/1000 epochs,  gbest SSE = 0.666883728
  mv  = 4,  iwt = 0.2700700701
TRAINPSO: 925/1000 epochs,  gbest SSE = 0.6668833307
  mv  = 4,  iwt = 0.2525525526
TRAINPSO: 950/1000 epochs,  gbest SSE = 0.6668832896
  mv  = 4,  iwt = 0.235035035
TRAINPSO: 975/1000 epochs,  gbest SSE = 0.6668776723
  mv  = 4,  iwt = 0.2175175175
TRAINPSO: 1000/1000 epochs,  gbest SSE = 0.6668762357
  mv  = 4,  iwt = 0.2
TRAINPSO: Network error did not reach the error goal.
*************end of training
**************************************************
```



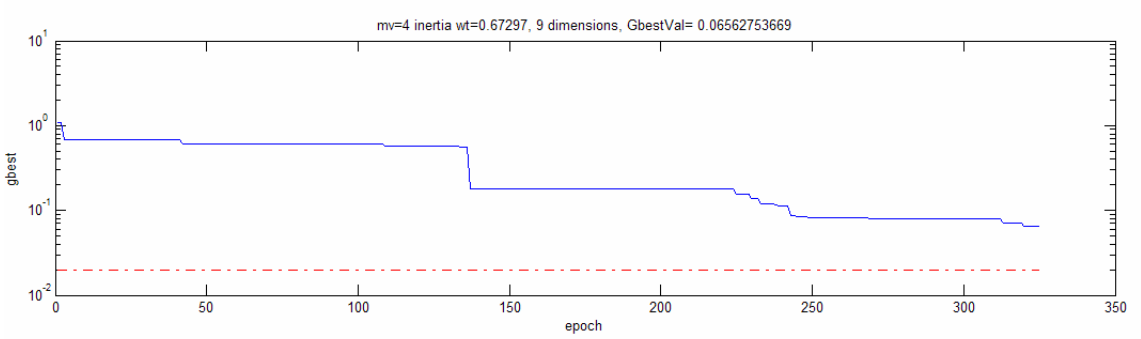
```
TRAINPSO: 25/1000 epochs,  gbest SSE = 0.6800321701
  mv  = 4,  iwt = 0.8831831832
```



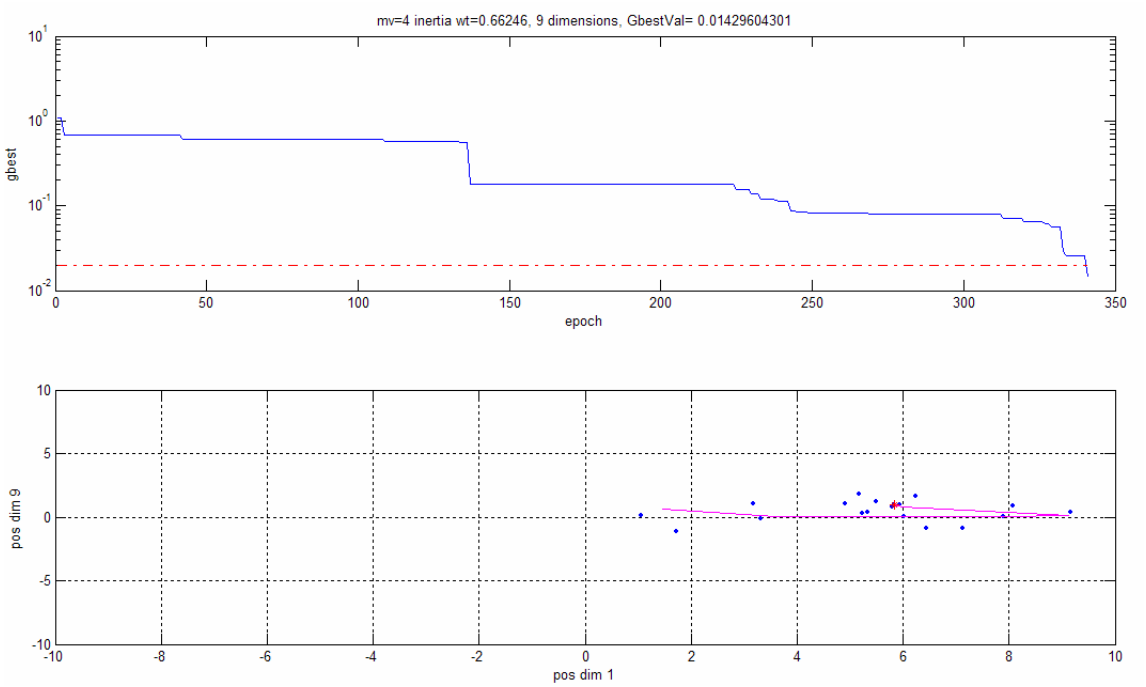
```
TRAINPSO: 50/1000 epochs,  gbest SSE = 0.6119237022
  mv  = 4,  iwt = 0.8656656657
TRAINPSO: 75/1000 epochs,  gbest SSE = 0.6022325759
  mv  = 4,  iwt = 0.8481481481
TRAINPSO: 100/1000 epochs,  gbest SSE = 0.6022325759
  mv  = 4,  iwt = 0.8306306306
TRAINPSO: 125/1000 epochs,  gbest SSE = 0.5813485776
  mv  = 4,  iwt = 0.8131131131
TRAINPSO: 150/1000 epochs,  gbest SSE = 0.1790156652
  mv  = 4,  iwt = 0.7955955956
TRAINPSO: 175/1000 epochs,  gbest SSE = 0.1790156652
  mv  = 4,  iwt = 0.7780780781
TRAINPSO: 200/1000 epochs,  gbest SSE = 0.1790156652
  mv  = 4,  iwt = 0.7605605606
```



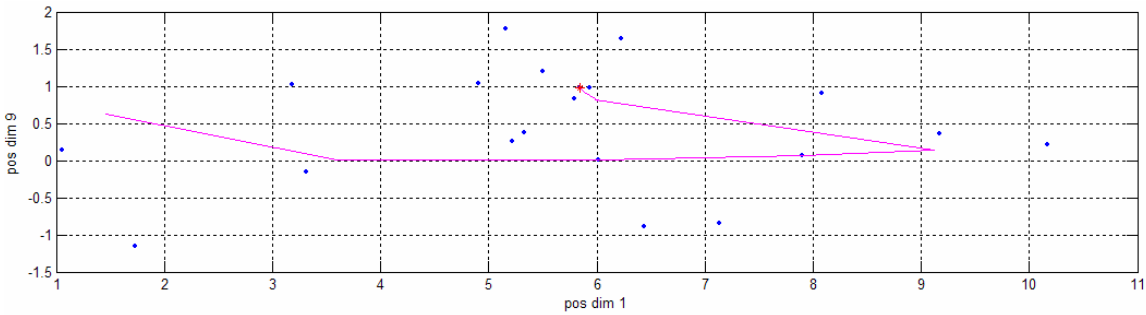
```
TRAINPSO: 225/1000 epochs,  gbest SSE = 0.1549225367
  mv  = 4,  iwt = 0.743043043
TRAINPSO: 250/1000 epochs,  gbest SSE = 0.0813382016
  mv  = 4,  iwt = 0.7255255255
TRAINPSO: 275/1000 epochs,  gbest SSE = 0.07992175607
  mv  = 4,  iwt = 0.708008008
TRAINPSO: 300/1000 epochs,  gbest SSE = 0.07975230164
  mv  = 4,  iwt = 0.6904904905
TRAINPSO: 325/1000 epochs,  gbest SSE = 0.06562753669
  mv  = 4,  iwt = 0.672972973
```



```
************************************* Reached Goal ******************
TRAINPSO: 340/1000 epochs,  gbest SSE = 0.01429604301
  mv  = 4,  iwt = 0.6624624625
************************************* end of training ***************
```



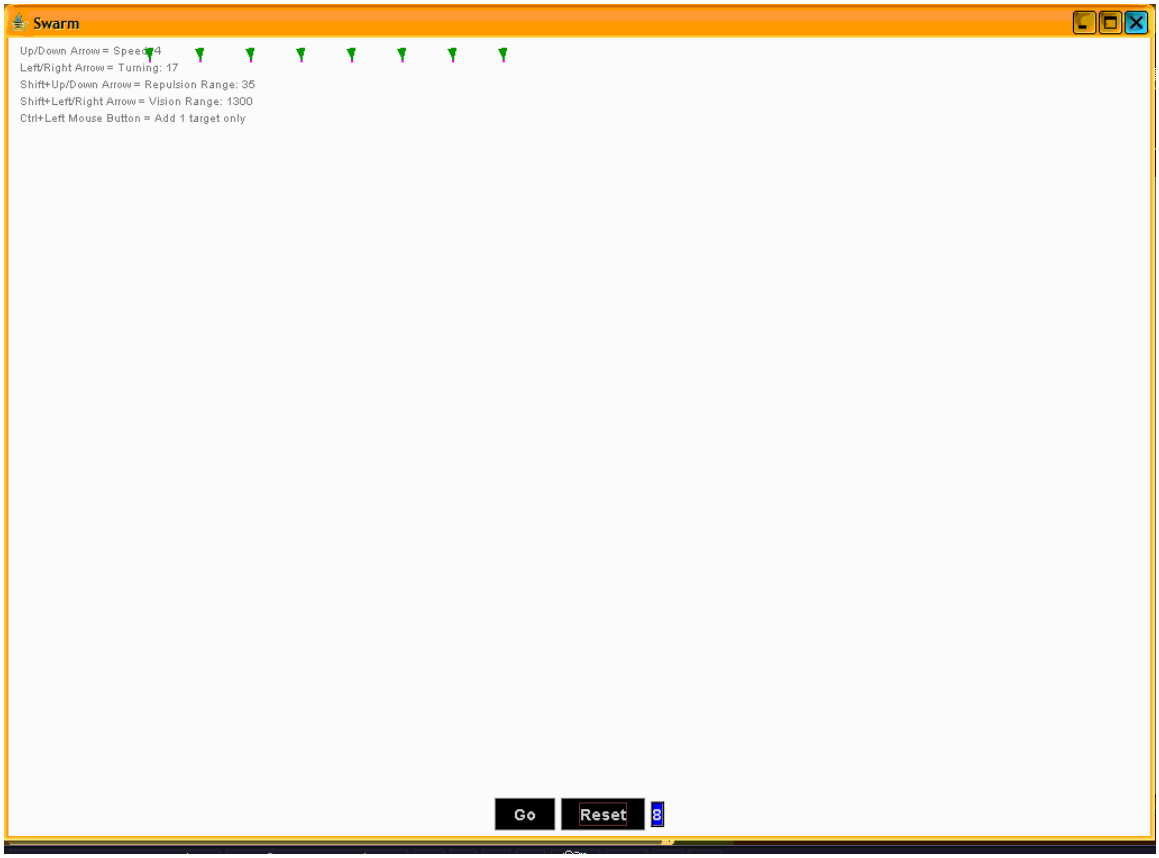
Zoom in on the final plot. The magenta line represents the movement around the search space of the best global position from 0 to 340 iterations.



THIS PAGE INTENTIONALLY LEFT BLANK

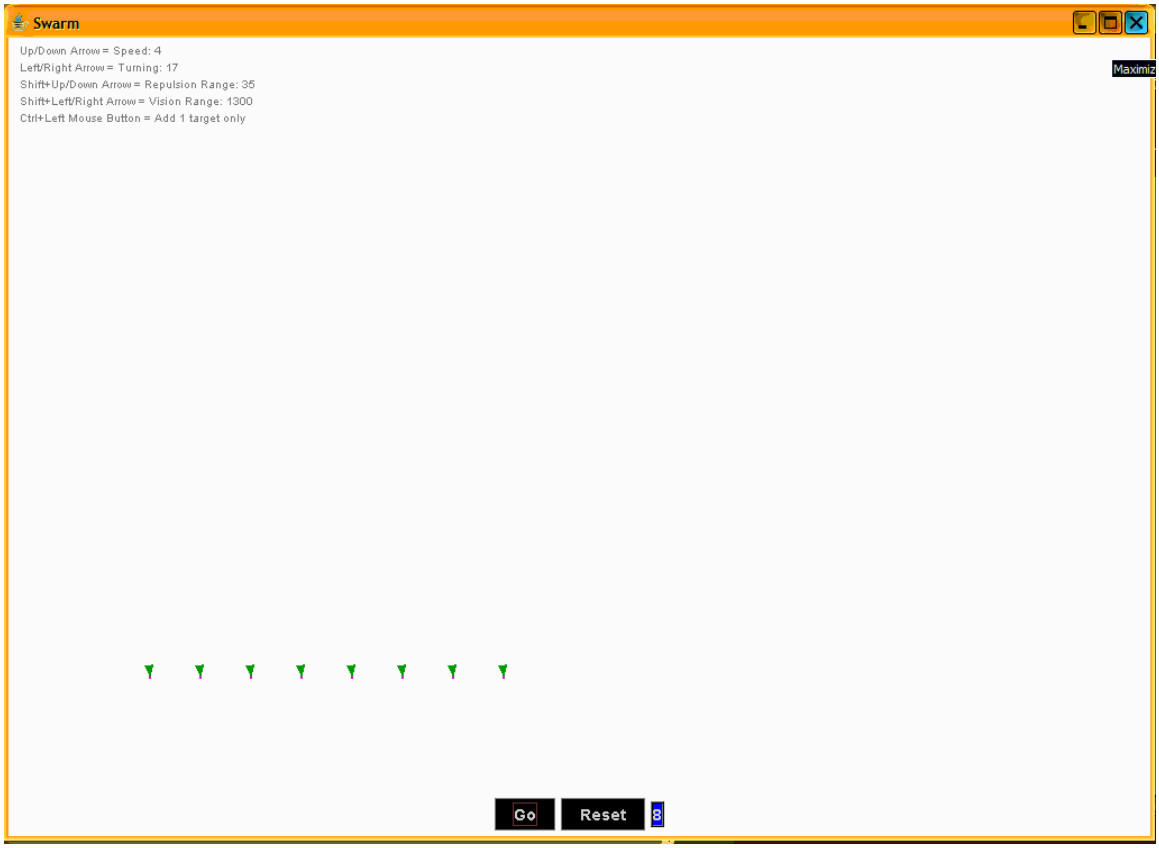
APPENDIX D. ORIGINAL SWARM.JAVA SIMULATION

The program swarm.java was created by Chin Lua at North Dakota State University [3]. The simulation is a two dimensional demonstration of a synchronized multi-point attack by UAVs. The population of UAVs is 8 for this example.



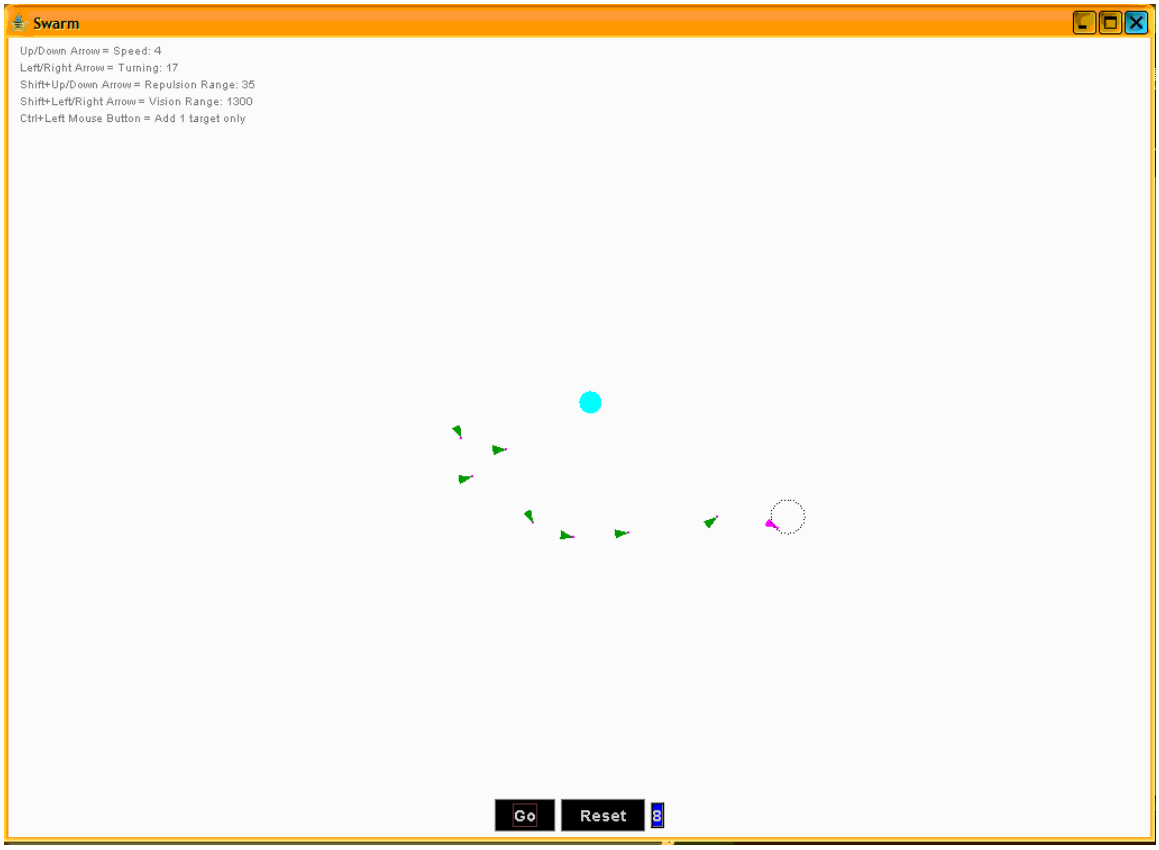
The parameters that are adjustable in the program are speed, turning ability, repulsion range, and visual range.

The UAVs continue to head ‘south’ until a target is added to the screen.

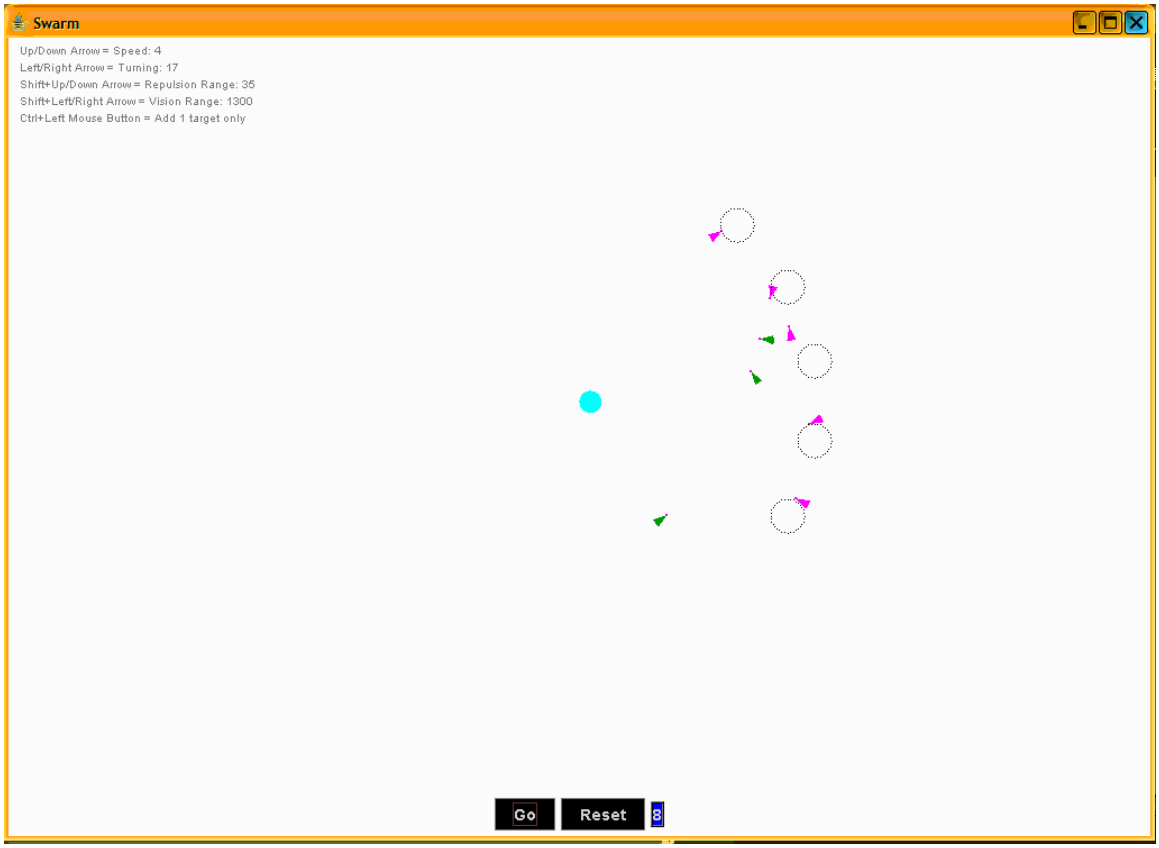


There are two circles around the blue target, an inner circle and outer circle. The UAVs head toward the inner circle until an orbit circle appears on the outer circle.

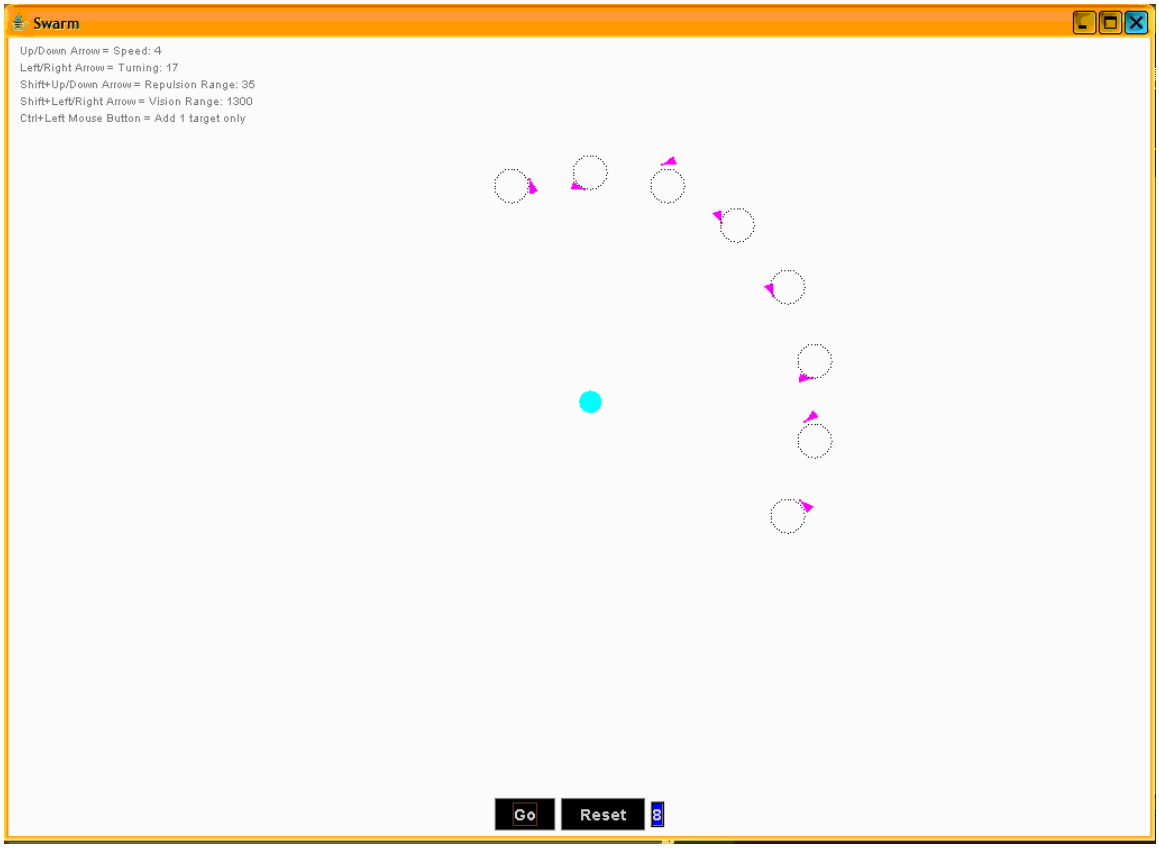
The location of the first orbit circle is chosen at random. A UAV heads toward the orbit circle if it is within its visual range. When the UAV color changes from green to purple, the UAV is heading toward an orbit circle or is currently orbiting the circle. The orbit circle is represented by a black dotted circle.



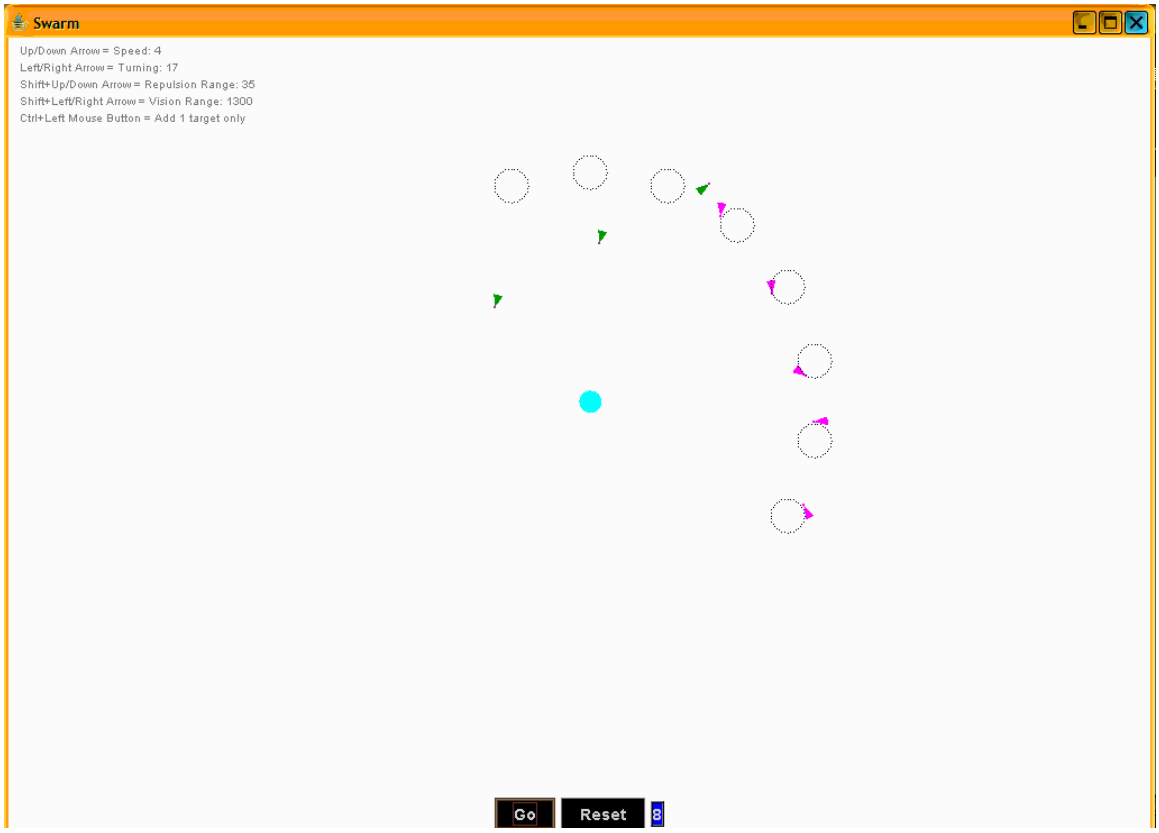
Once a UAV touches the orbit circle, it becomes a station owner. The remaining UAVs search for the orbit circles that appear counterclockwise around the outer circle.



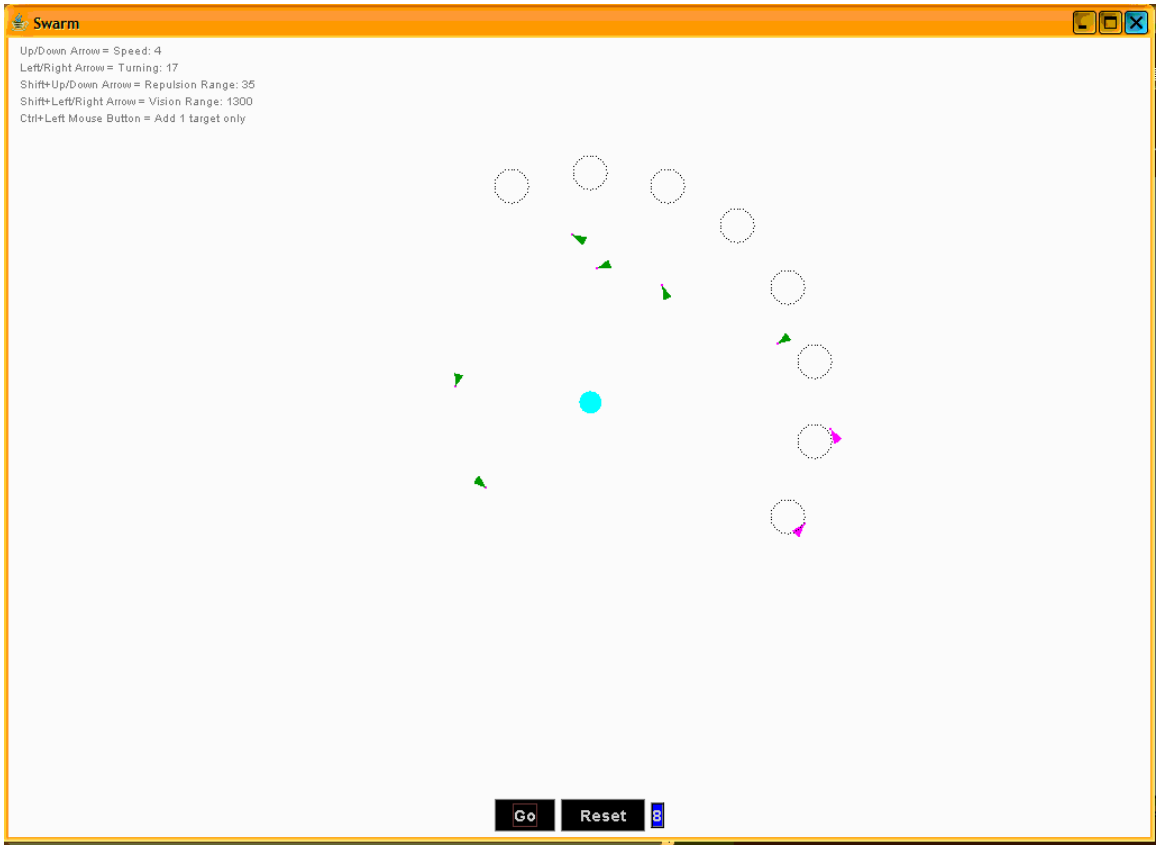
All UAVs become station owners and continue to orbit the circle.



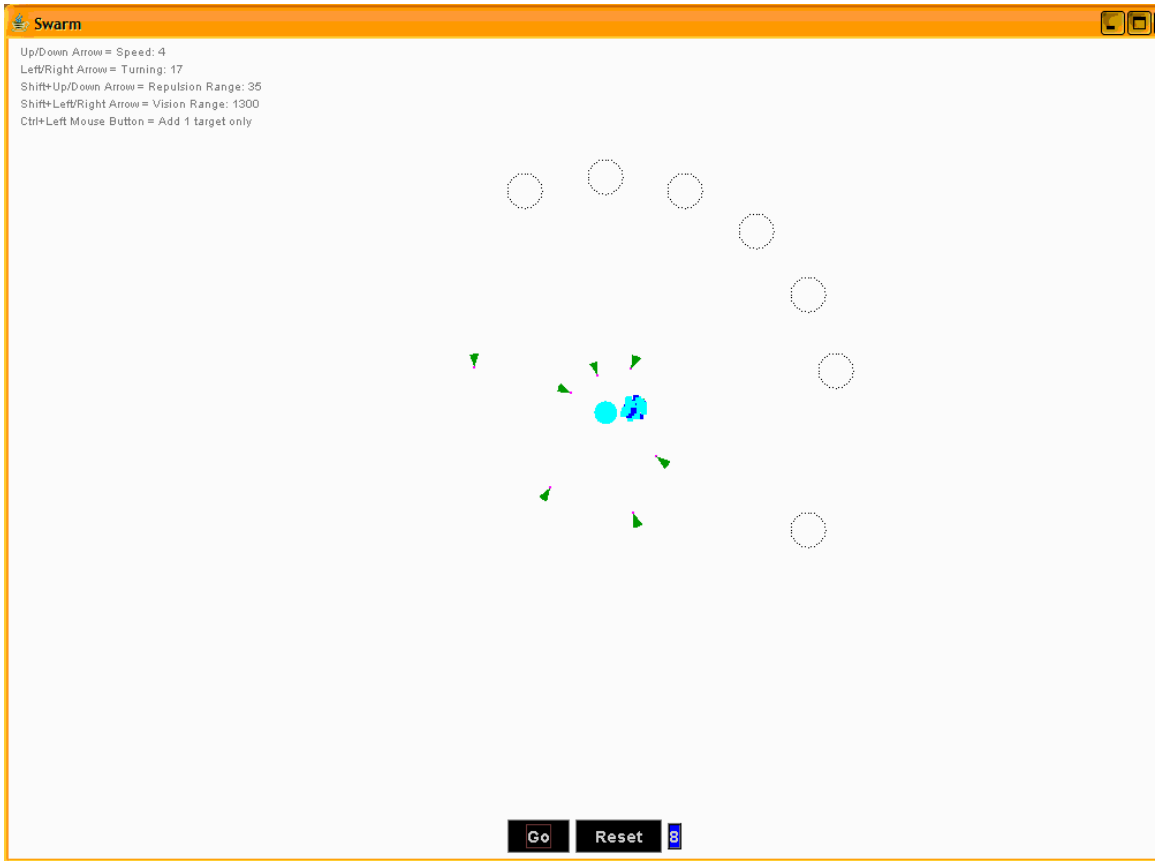
The attack is the only synchronized action. The first UAV to complete a specified number of orbits around the circle, analogous to the amount of fuel used, will issue an order of attack. The real attack order is locally communicated from one UAV to the neighboring UAVs, but simulated attack order is globally known. Since there are only 8 UAVs, the UAVs travel around the inner circle until they are equally spaced by 40 degrees: $360/8 = 40$. When a UAV leaves the orbit circle, it returns to a green color.



Because the UAVs travel at a constant velocity, delay is introduced into the program to indicate when each UAV should leave the orbit circle. The goal is to approach the target at the same time. There are still timing errors in the simulation because the location of each UAV around the orbiting circle is not synchronized.

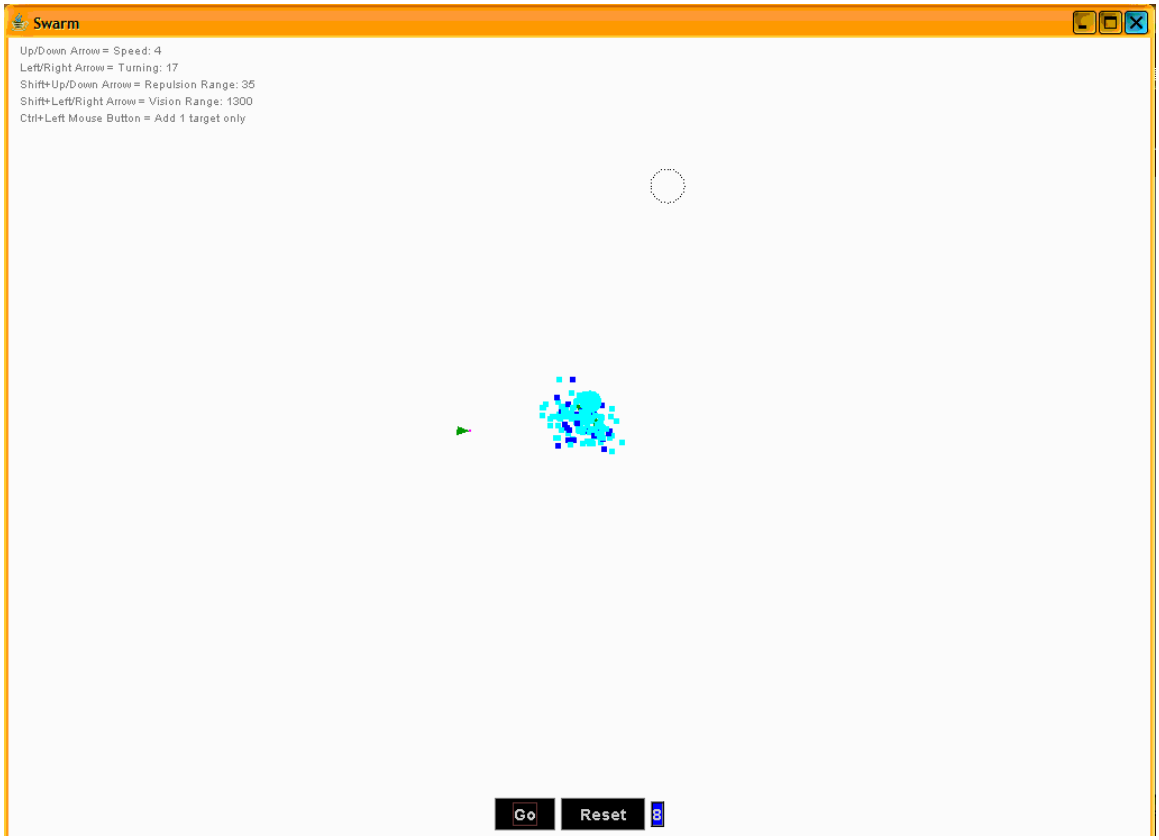


Once the UAVs have reached the 40-degree spacing around the inner circle, they turn toward the target. When a UAV touches the target, the UAV explodes, which is represented by light blue and dark blue pixels. After the UAV explodes, the orbit circle it possessed earlier disappears from the screen. In the following figure, the second station keeper hits the target first.

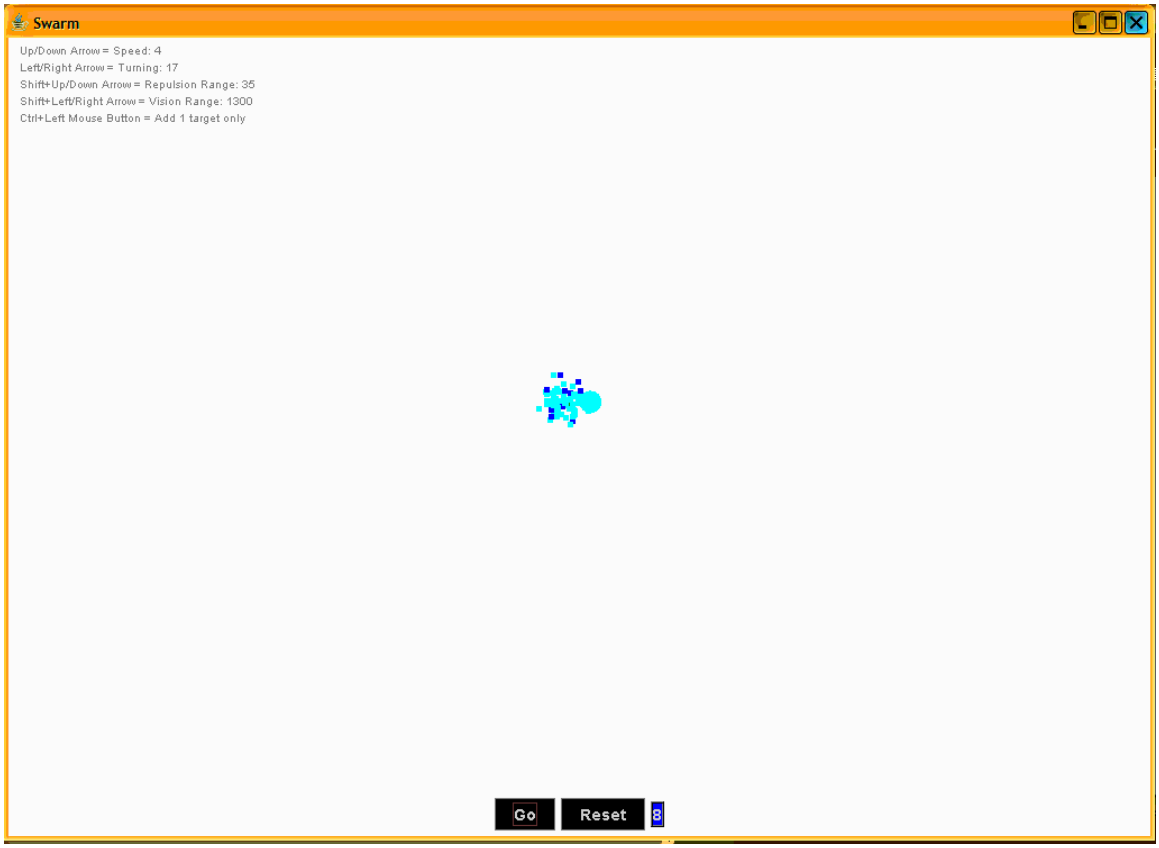


More UAVs hit the target.

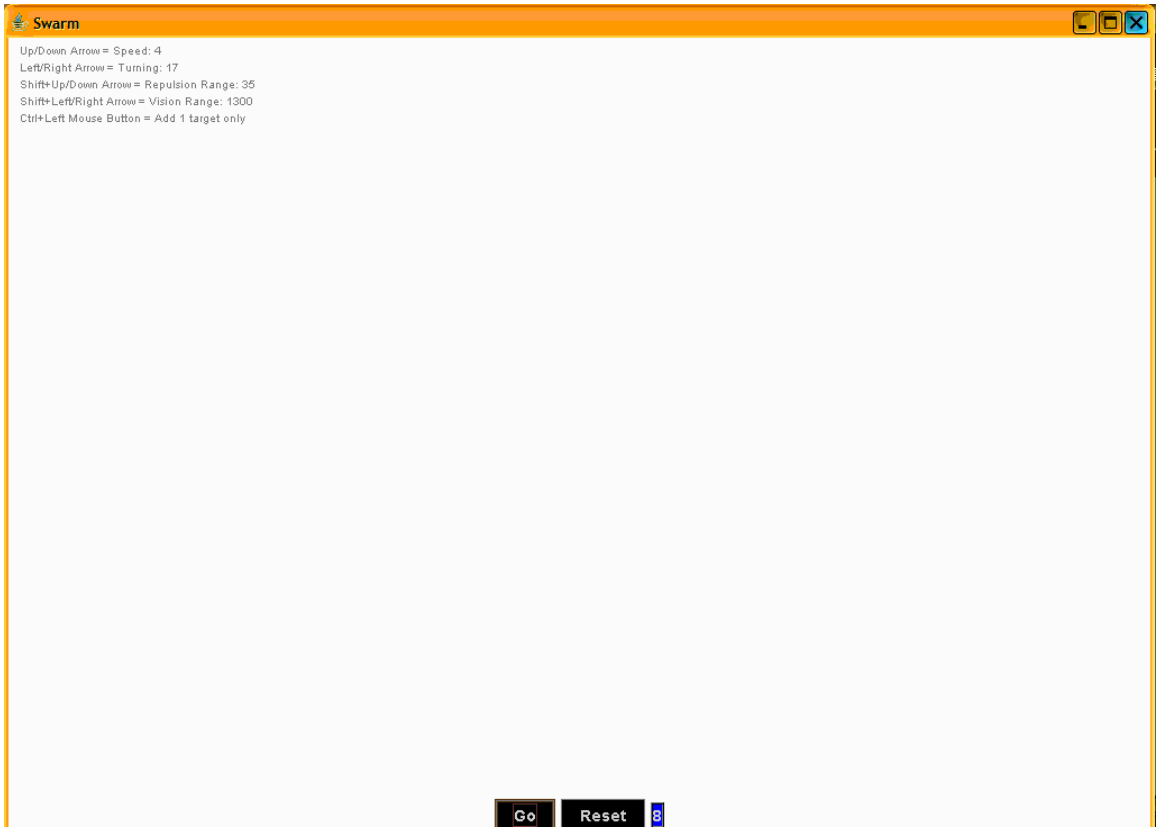
The remaining UAV has just turned toward the target. The delayed attack is due to timing errors and avoidance maneuvers.



The last UAV just hit the target.



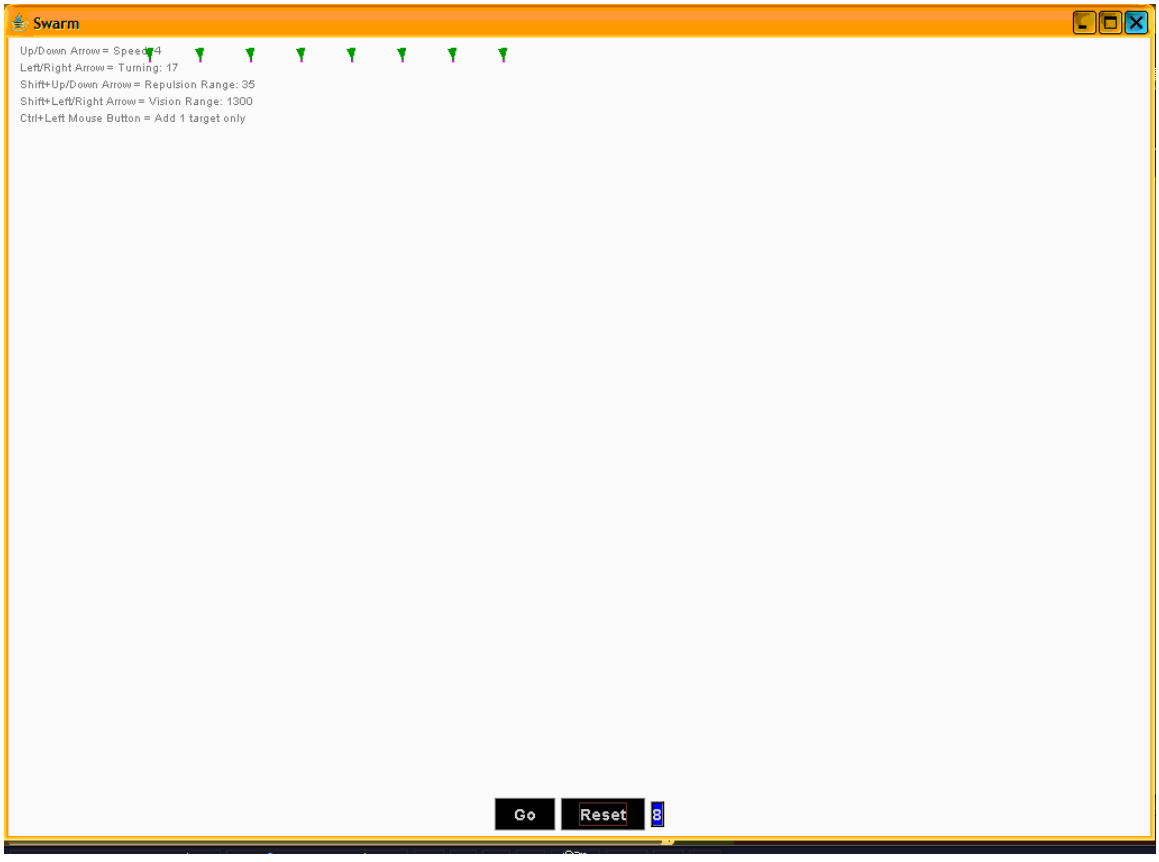
The target disappears, and the program is over. Additional targets can be placed around the screen, but obviously there are not any UAVs left to coordinate an attack. To restart the program, press the reset button at the bottom of the screen.



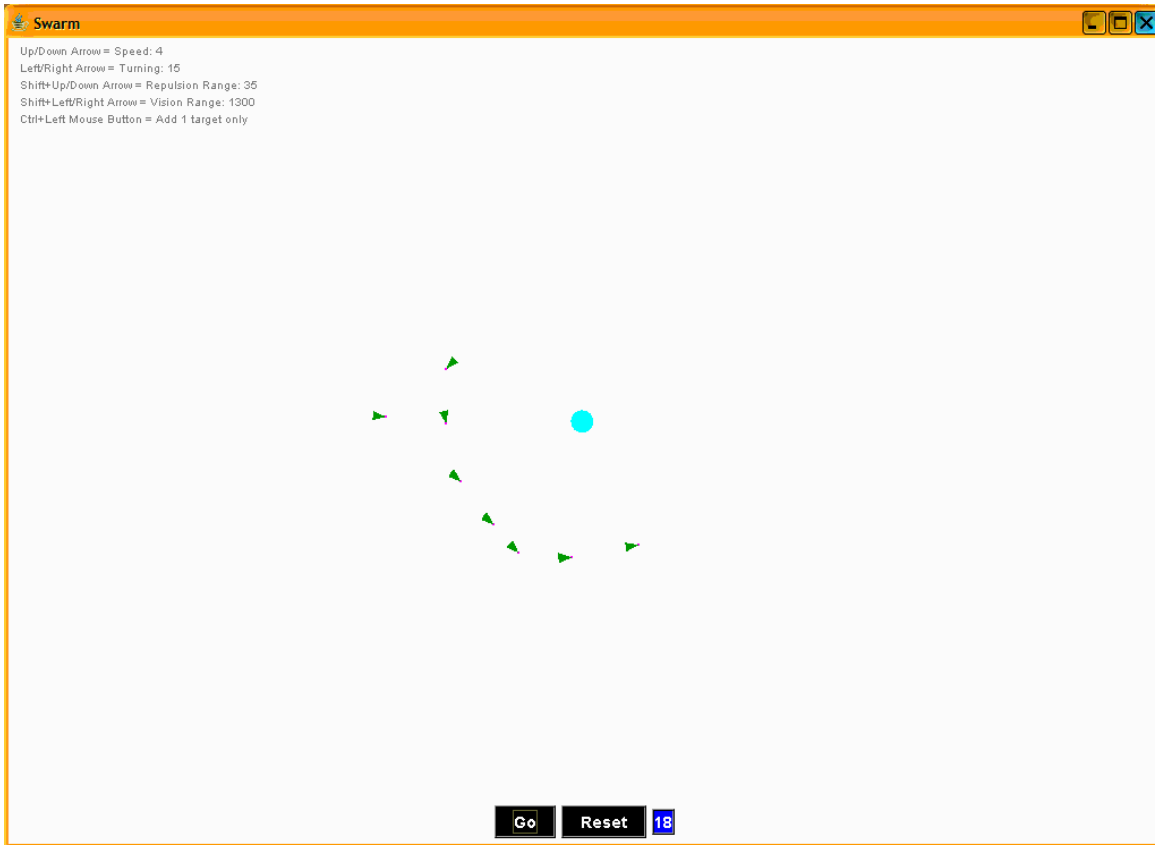
THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. MODIFIED SWARM.JAVA PROGRAM

The program swarm.java was created by Chin Lua at North Dakota State University [3]. The original simulation is a two-dimensional demonstration of a synchronized multi-point attack by UAVs. The program has been modified to allow the UAVs to pass over the target without exploding. Once each UAV has passed over the target, target disappears from the screen. The UAVs are now ready to attack the next target that appears. The population of UAVs is 8 for this example. The parameters that are adjustable in the program are speed, turning ability, repulsion range, and visual range. The parameter display is located in the upper left corner. The actions of the UAVs are unaltered from the original program until the attack sequence. Upon initialization, the UAVs continue to head 'south' until a target is added to the screen.

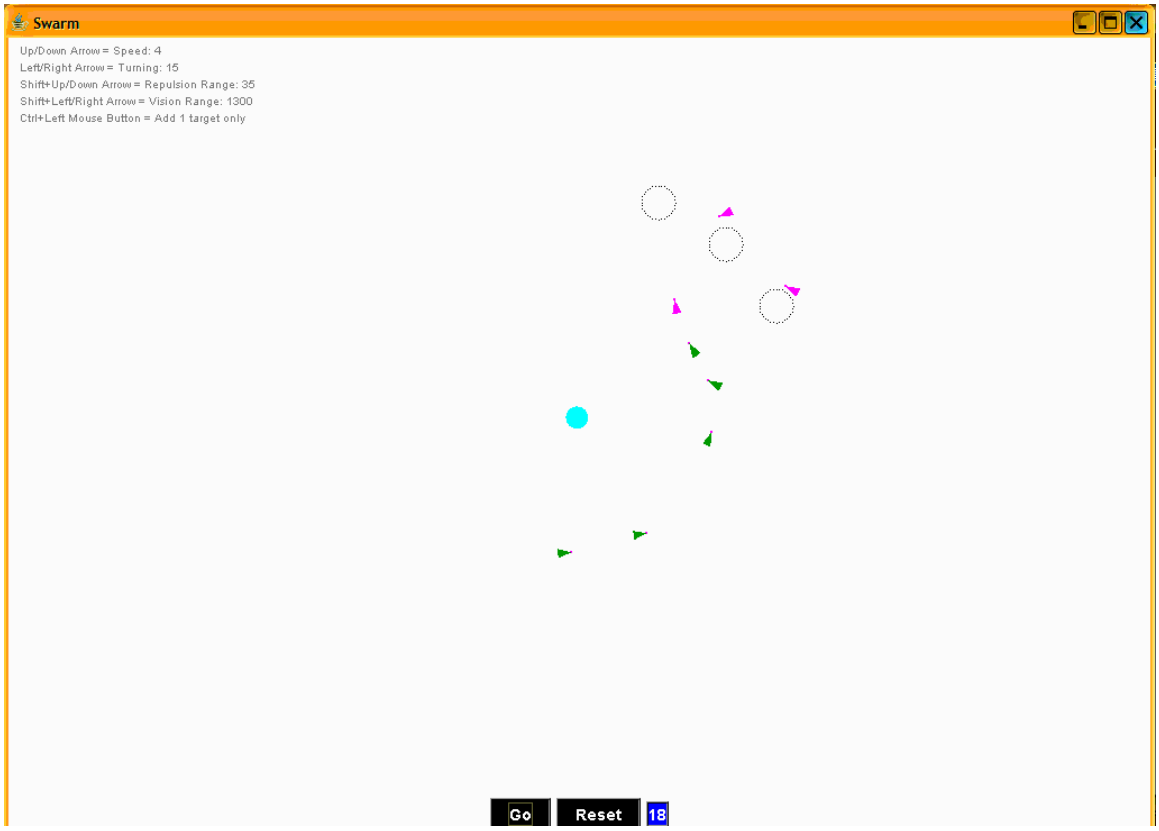


There are two circles around the target, an inner circle and outer circle. The UAVs head toward the inner circle until an orbit circle appears on the outer circle.

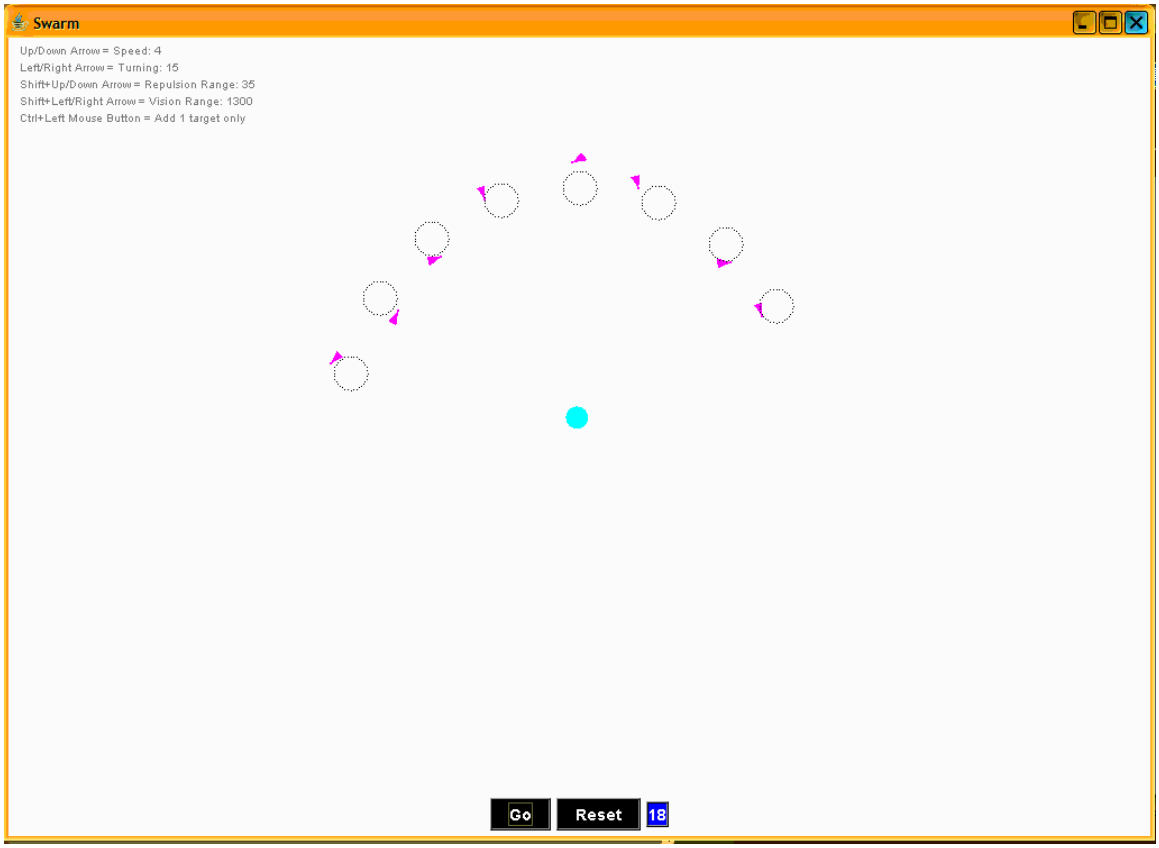


The location of the first orbit circle is chosen at random. A UAV heads toward the orbit circle if it is within its visual range. When the UAV color changes from green to purple, the UAV is heading toward an orbit circle or is currently orbiting the circle. The orbit circle is represented by a black dotted circle.

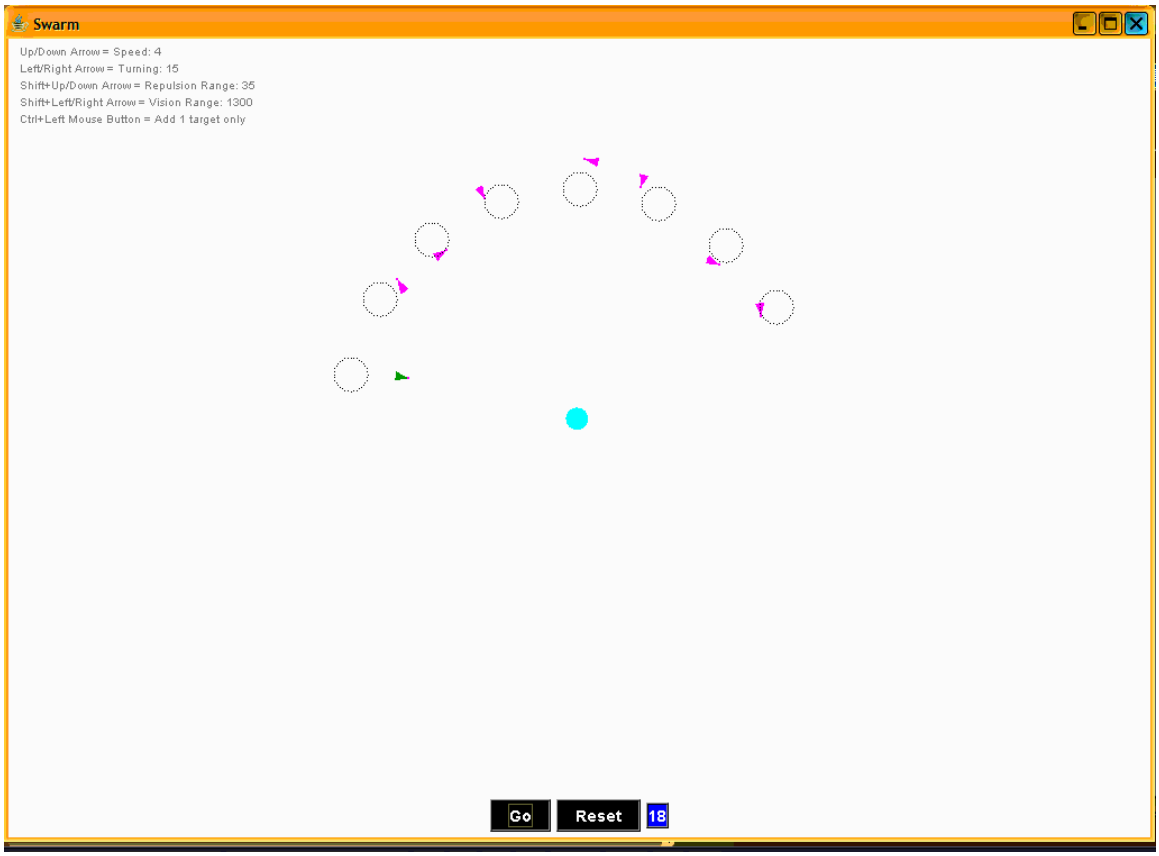
Once a UAV touches the orbit circle, it becomes a station owner. The remaining UAVs search for the orbit circles that appear counterclockwise around the outer circle.

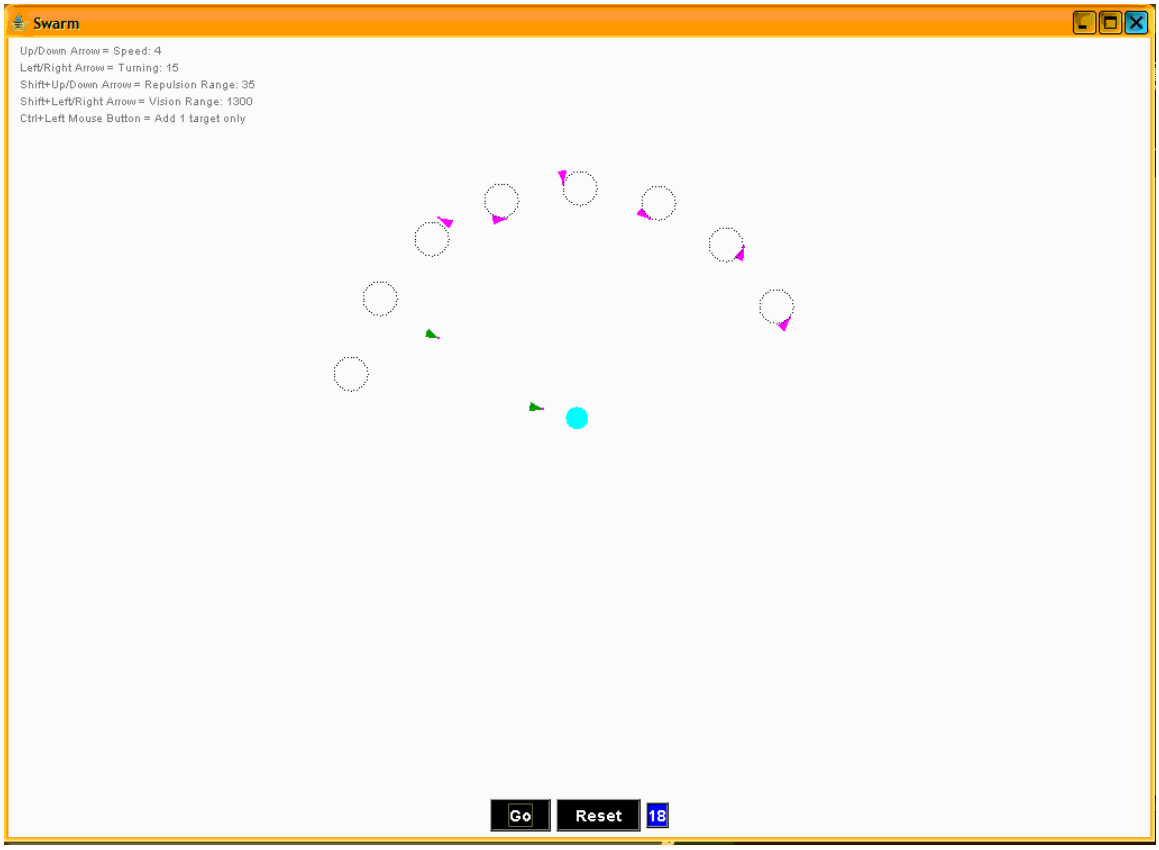


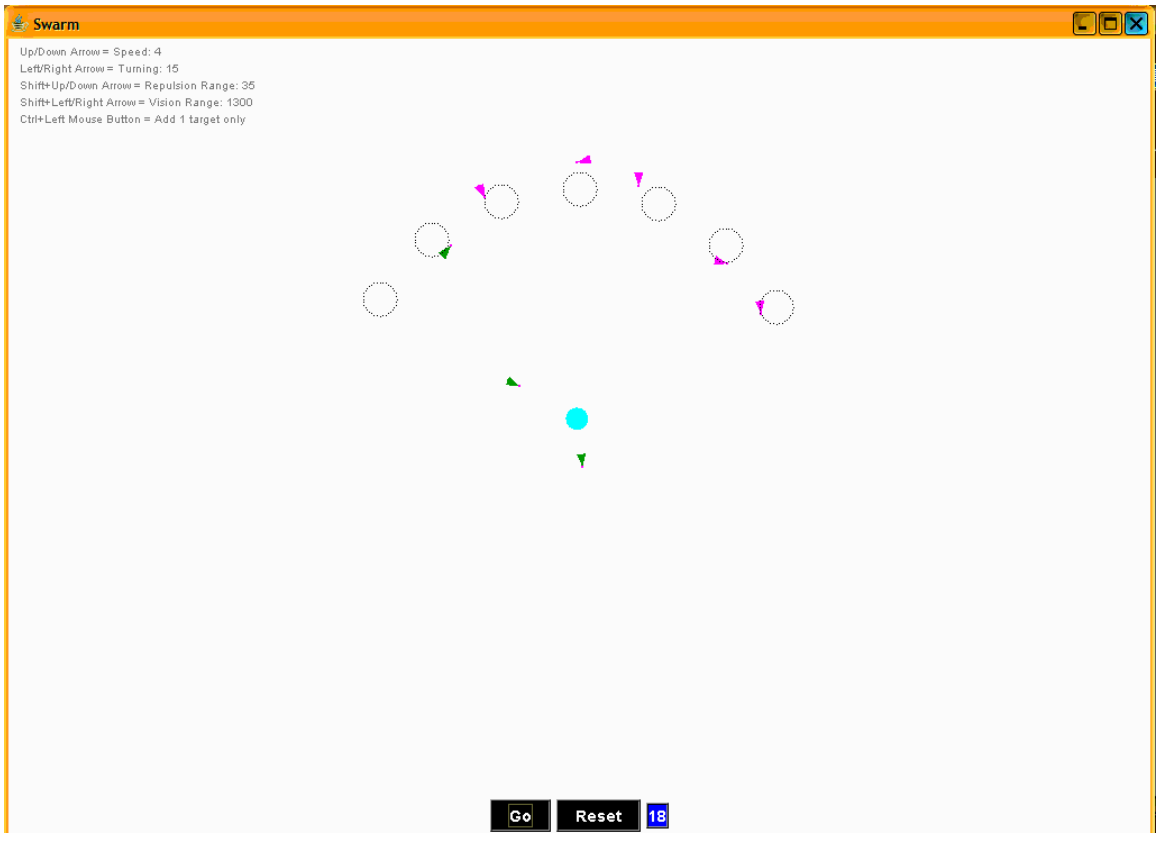
All UAVs become station owners and continue to orbit the circle.

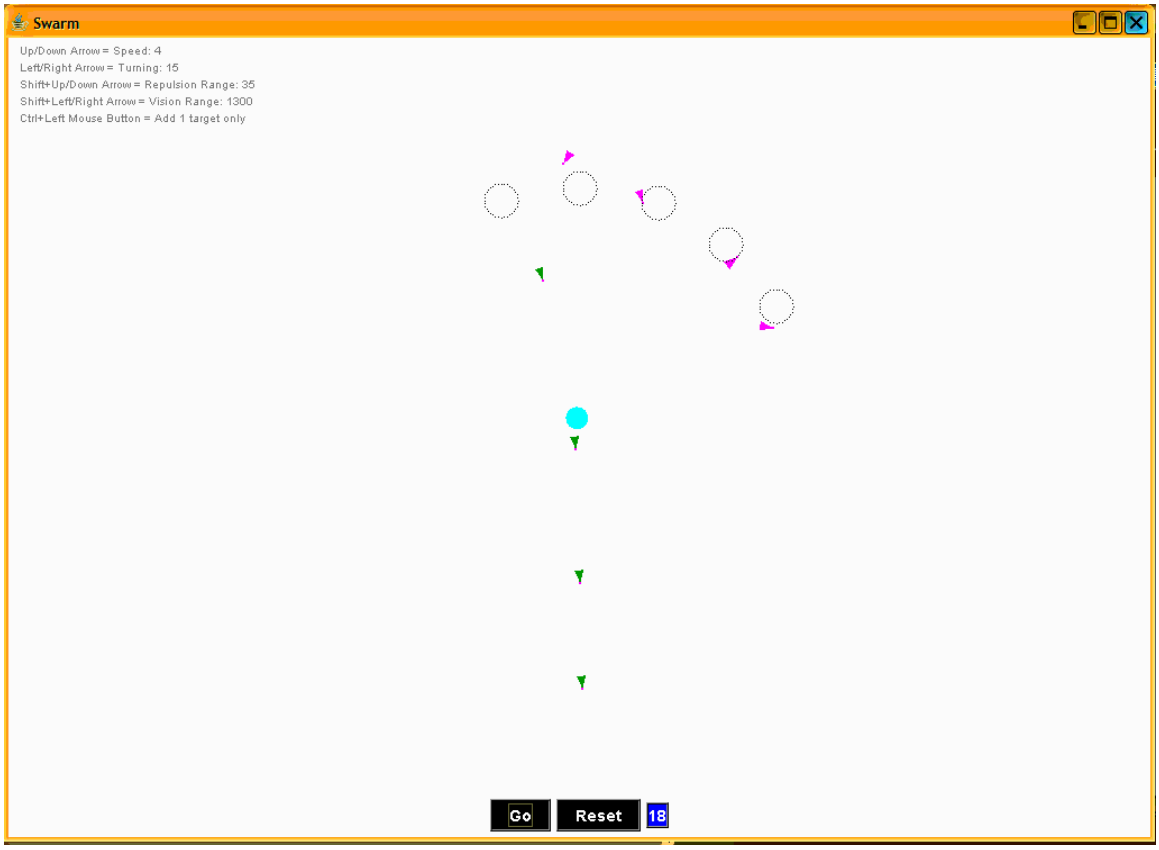


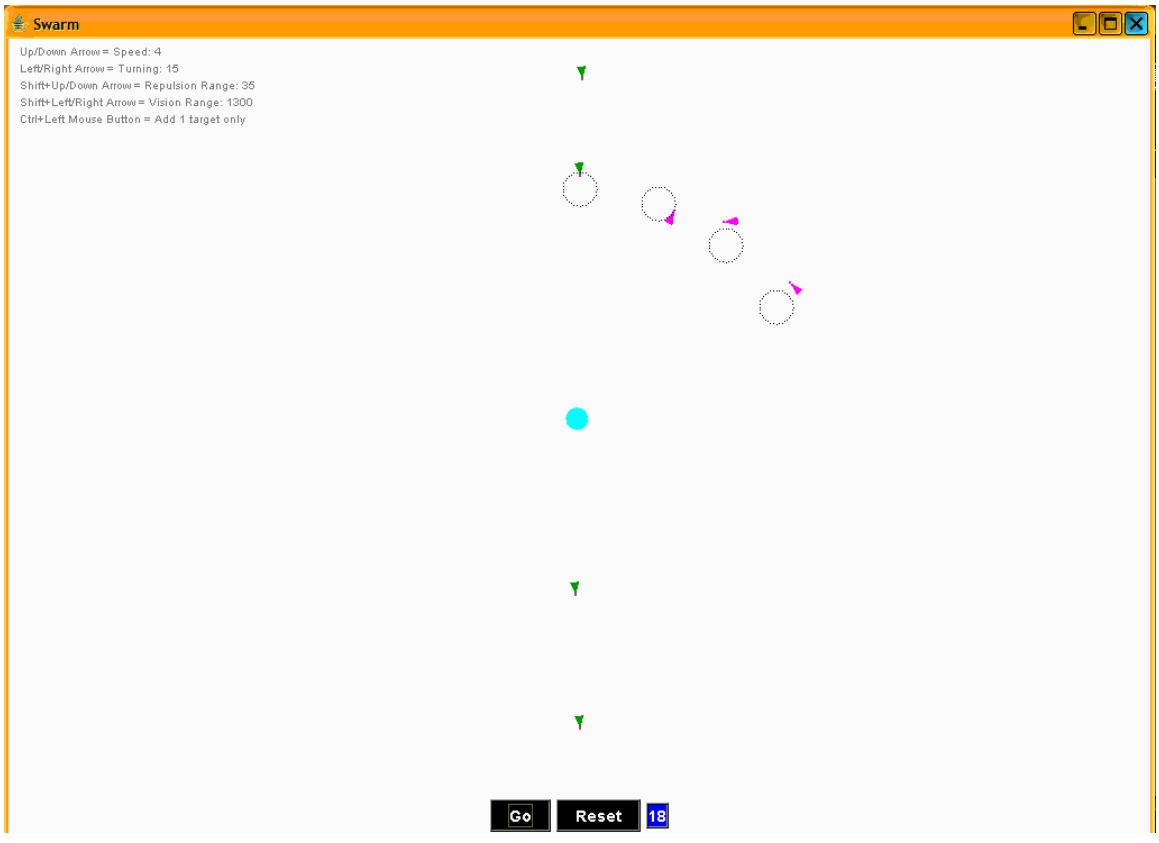
The first UAV to complete a specified number of orbits around the circle, analogous to the amount of fuel used, issues an order of attack. The real attack order is locally communicated from one UAV to the neighboring UAVs, but the simulated attack order is globally known. There is a delay introduced so each UAV leaves the orbit station at different time. The UAV with the highest index number, which is the UAV that became a station keeper last, is the first to attack the target. When a UAV leaves the orbit circle, it returns to a green color. They pass over the target in an orderly manner and head 'south.' The direction heading after the attack can be specified by the designer. Once the UAV has passed over the target, it enters search mode. Because the screen recycles UAVs from the bottom to the top, the UAVs going south appear again at the top of the screen. Realistically the UAVs would continue south without interfering with the attack. The attack does not run smoothly because the attacking UAVs have to avoid the recycled UAVs. The following figures show the attack sequence.

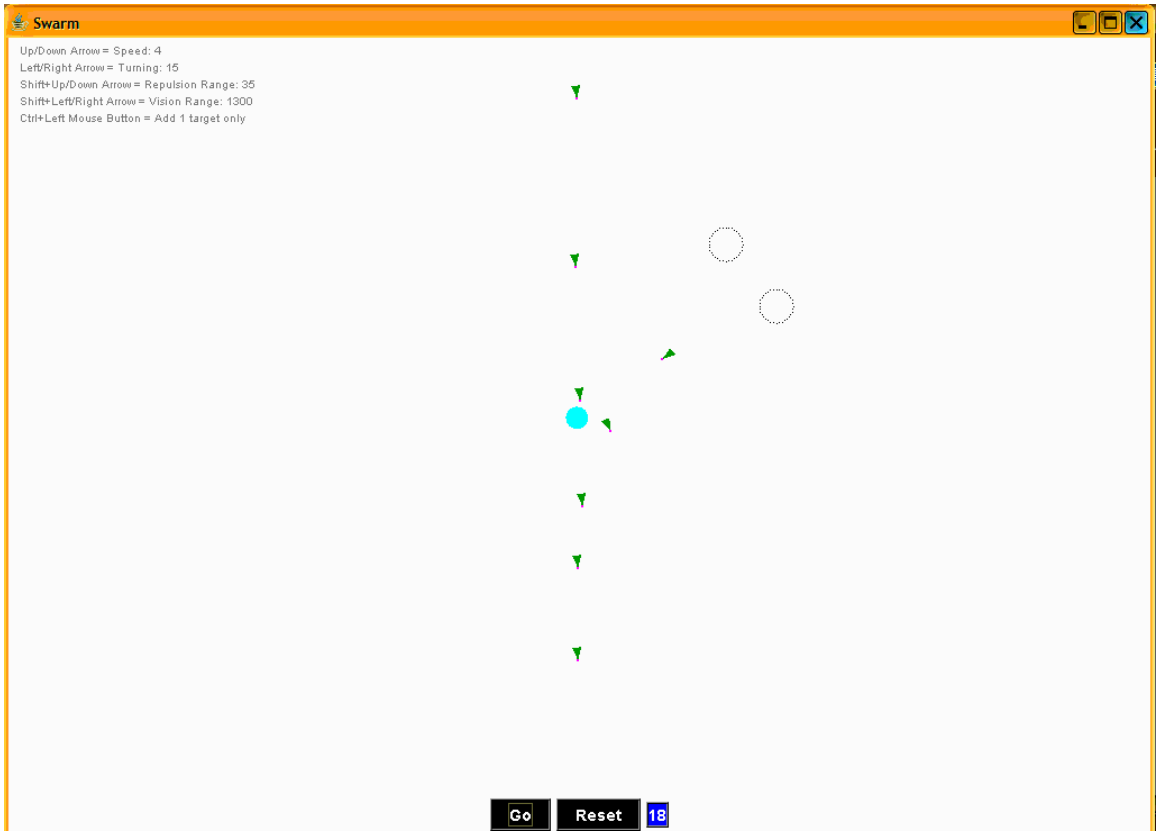



Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18

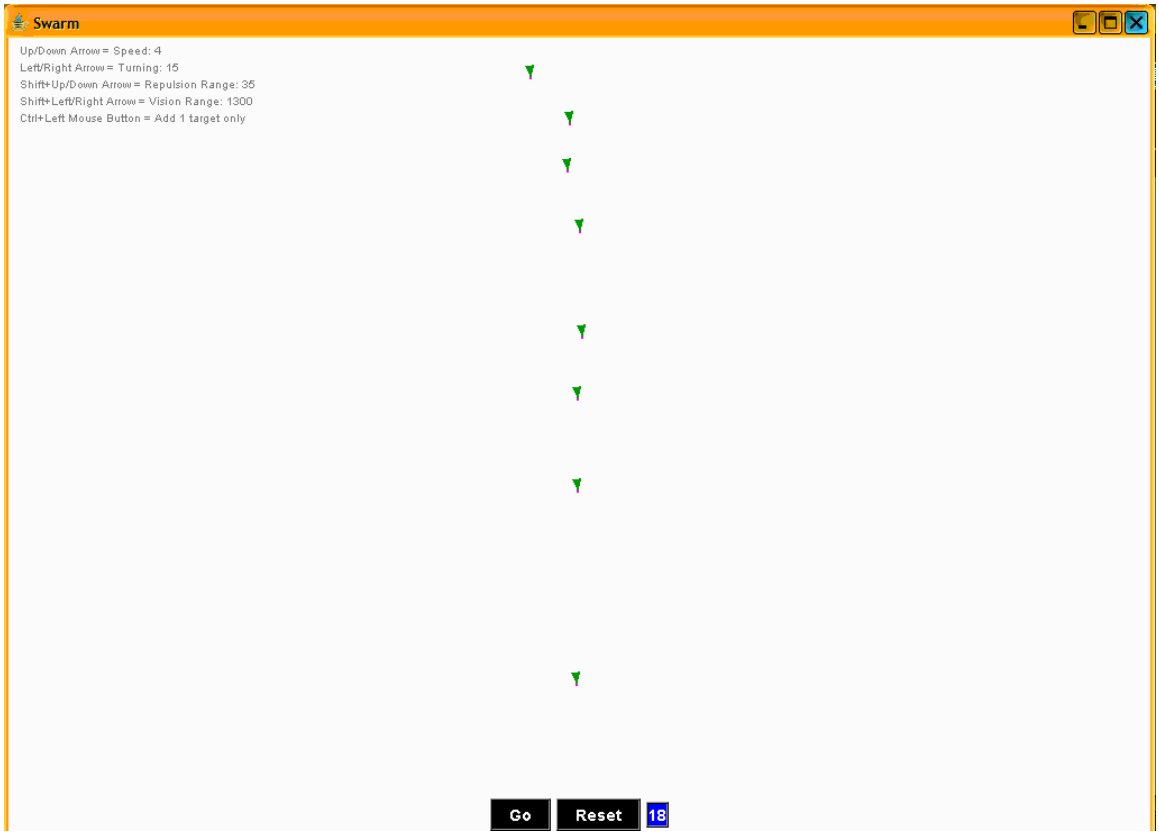

Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18


Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18

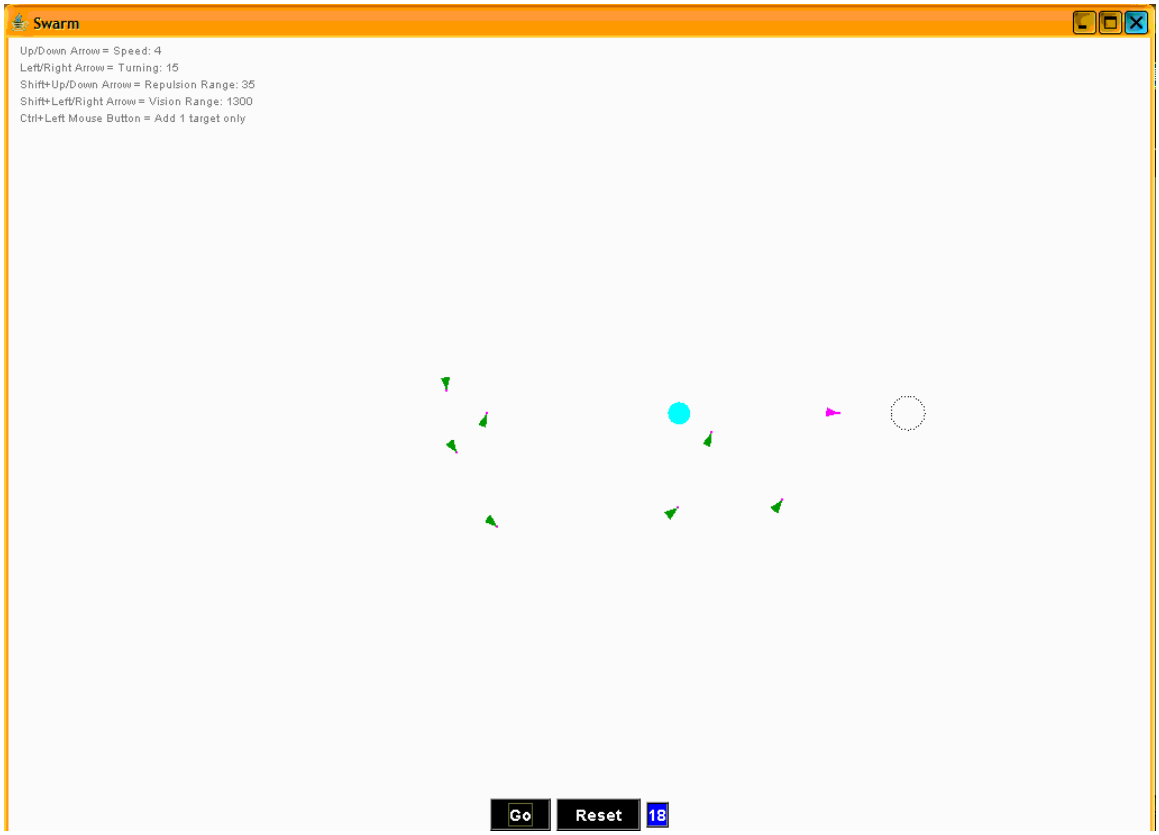

Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18

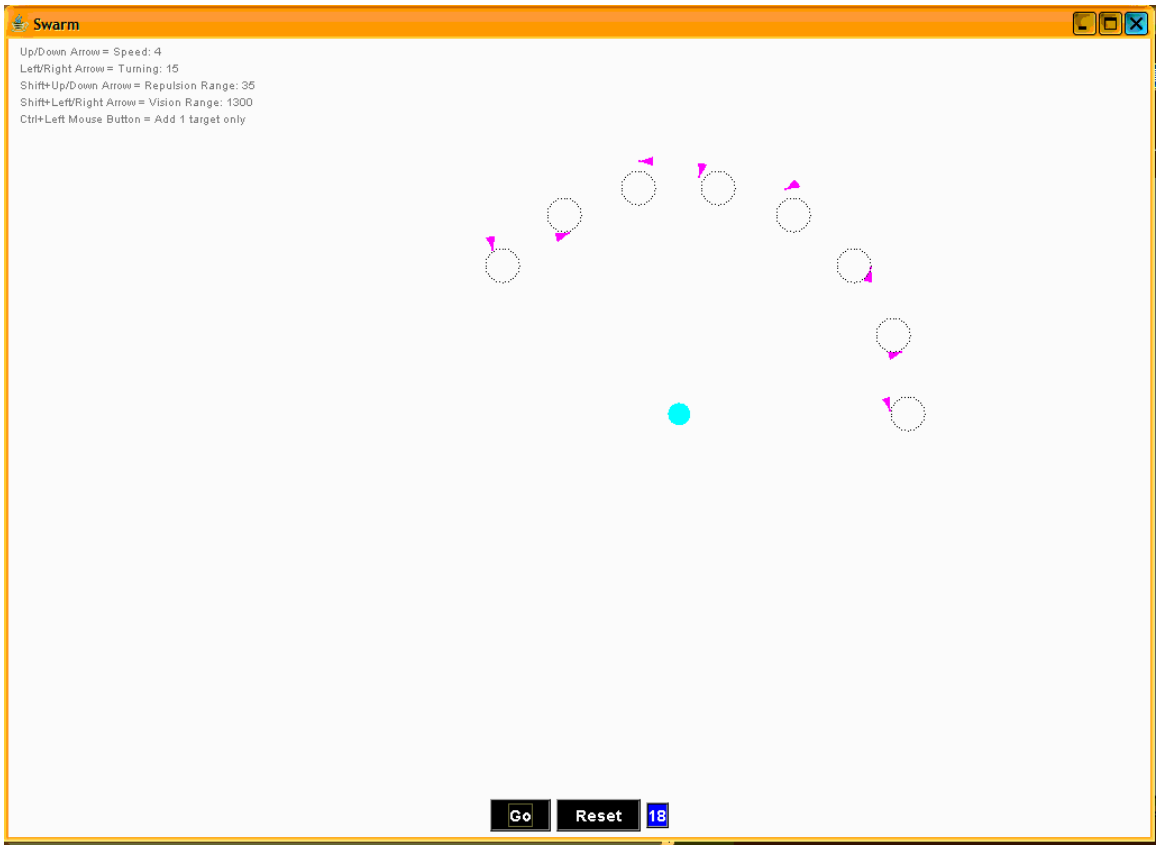

Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18


Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18


Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18


Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18


Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18


Swarm
Up/Down Arrow = Speed: 4
Left/Right Arrow = Turning: 15
Shift+Up/Down Arrow = Repulsion Range: 35
Shift+Left/Right Arrow = Vision Range: 1300
Ctrl+Left Mouse Button = Add 1 target only
Go
Reset
18

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Anup Dargar, Ahmed Kamel, Gordon Christensen and Kendall E. Nygard, "An Agent Based Framework for UAV Collaboration," *Proceedings of the ISCA 11th International Conference on Intelligent Systems*, pp. 54-59, 2002.

[2] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings IEEE International Conference on Neural Networks*, pp. 1942-1948, December 1995.

[3] Chin A. Lua, Karl Altenburg, and Kendall E. Nygard, "Synchronized Multi-Point Attack by Autonomous Reactive Vehicles with Simple Local Communication," *Proceedings of 2003 IEEE Swarm Intelligence Symposium*, pp. 95-102, April 2003.

[4] Brian Birge, "PSOt- a Particle Swarm Optimization Toolbox for Use with Matlab," *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pp. 182-186, April 2003.

[5] Gerardo Beni, "From Swarm Intelligence to Swarm Robotics," *Swarm Robotics* LNCS 3342, Springer-Verlag, Berlin, pp. 1-9, 2005.

[6] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz, *Swarm Intelligence from Natural to Artificial Systems*, Oxford University Press, New York, 1999.

[7] James Kennedy and Russell C. Eberhart, *Swarm Intelligence*, Academic Press, San Francisco, 2001.

[8] Haykin, Simon, *Neural Networks: A Comprehensive Foundation*, 2nd Ed., Prentice Hall, Upper Saddle River, 1999.

[9] Venu G. Gudise and Ganesh K. Venayagamoorthy, "Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks," *Proceedings of 2003 IEEE Swarm Intelligence Symposium*, pp. 110-117, April 2003.

[10] Karl Altenburg, Joseph Schlecht, and Kendall E. Nygard, "An Agent-based Simulation for Modeling Intelligent Munitions," *Proceedings of the Second WSEAS International Conference on Simulation, Modeling and Optimization*, Skiathos, Greece, 2002, http://www.cs.ndsu.nodak.edu/~nygard/research/munitions.pdf, Last visited June 2005.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Prof. John P. Powers
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Prof. Phillip E. Pace
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Prof. David C. Jenn
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. Mr. Kent Frantz and Mrs. Janet Frantz
   Peachtree City, Georgia

THIS PAGE INTENTIONALLY LEFT BLANK