AFRL-RI-RS-TR-2019-128

MULTI-LAYER MODEL OF SWARM INTELLIGENCE FOR RESILIENT AUTONOMOUS SYSTEMS

EMBRY-RIDDLE AERONAUTICAL UNIVERSITY

JUNE 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2019-128 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
EDWARD VERENICH
Work Unit Manager

/ S /
JULIE BRICHACEK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUNE 2019	FINAL TECHNICAL REPORT	SEP 2016 – DEC 2018

4. TITLE AND SUBTITLE

MULTI-LAYER MODEL OF SWARM INTELLIGENCE FOR RESILIENT AUTONOMOUS SYSTEMS

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-16-1-0203

5c. PROGRAM ELEMENT NUMBER
63788F

6. AUTHOR(S)

Massood Towhidnejad
Jayson Clifford
Jake Neighbors
Ramin Rashedi

5d. PROJECT NUMBER
S2MY

5e. TASK NUMBER
RA

5f. WORK UNIT NUMBER
SE

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Embry-Riddle Aeronautical University
600 S. Clyde Morris Blvd.
Daytona Beach, FL 32771

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISC
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2019-128

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The objective of the this project is to develop a distributed multi-layer autonomous UAS planning and control technology for gathering intelligence in Anti-Access Area Denial (A2/AD) environments populated by intelligent adaptive adversaries. These resilient autonomous systems are able to navigate through hostile environments while collecting intelligence and minimizing the loss of assets. Our approach incorporates artificial life concepts, with the high-level architecture divided into three biologically inspired meta-layers: cyber-physical, reactive, and deliberative. The key concepts of our approach are: A layered architecture of intelligence on a spectrum from reactive to deliberative; the use of artificial life algorithms providing robust, complex behavior from a set of simple rules; communication of information between agents via observation; the use of a genetic algorithm (GA) to assist the team in selecting and adapting the best solutions for each layer. This distributed cooperative system of intelligent assets will provide adaptable, scalable performance to accomplish mission goals in challenging environments.

15. SUBJECT TERMS
Adaptive Reasoning, Adaptive Strategies, Communication Resiliency, Dynamic Adaption, Fault Tolerant

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**EDWARD VERENICH**
U	U	U	UU	24	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Section Page

LIST OF FIGURES

1. SUMMARY

The goal of the ERAU approach to this problem was to develop an advanced autopilot for autonomous drone control that would perform well in all of the scenarios provided by AFRL. Our approach can be divided into the following major components:

- The use of biomimetic artificial life algorithms to provide robust, adaptive behavior via combining a number of simple rules that interact with each other, both within the memory of the drones and between drones via the perceived environment.
- The stratification of information and actions into multiple layers, each with a view of the world that is constructed by prefacing layers — filtering, correcting, and augmenting sensor data.
- The use of a highly parallel genetic algorithm using real-coded values, rank-based migration, blend crossover, and a self-adaptive mutation rate to tune the parameters of the artificial life algorithms.
- Communication of information between drones primarily via passive observation and/or simple data transmissions to limit radio footprint.
- The use of a fast-time simulation environment to generate data for the genetic algorithm.

Each artificial life algorithm is constrained to a single layer, with the behavior of the algorithm being modified by adjusting a predefined set of parameters. The parameters are often simple numerical values, but in some cases also contain data structures. Layers have a standard interface for communicating with other layers, allowing them to be combined in any way within a vertical architecture. The genetic algorithm is used to tune the parameters of each layer by executing the algorithm in successive runs in the fast-time simulator.

The output of this process is a multi-layer artificial-life based autopilot trained to operate in an Anti-Access Area Denial (A2AD) environment, minimizing the use of expendable resources while maximizing mission performance with respect to predefined mission objectives. The performance of the system is graded on a total of 100 available points divided into categories defining various aspects of mission performance. From this total, there are 15 points allotted to total mission time used, where any time spent executing the mission, starting from the first second of the mission and continuing to the mission time limit, reduces the score by up to 15 points. Using the approach summarized in this report, the team was able to develop a system capable of an average score of 81.8/100 on the AFRL provided scenarios.

2. INTRODUCTION

This report will summarize the work completed by Embry-Riddle Aeronautical University (ERAU) on Resilient Autonomous Systems (RAS), as well as summarize progress and results during the last two quarters of the project. Supported by the Air Force Research Lab (AFRL), the project goals were to increase resiliency over existing automated solutions that offer little autonomous re-planning capability. In this context, increased resiliency is defined as the ability of the system to operate at or above an acceptable level of performance even in unfavorable environments. The goals of Resilient Autonomous Systems (RAS) was to develop a system to coordinate and control autonomous unmanned aircraft systems (UAS) operating with degraded communications in an uncertain and hostile battle space to achieve a number of pre-specified intelligence, surveillance, reconnaissance (ISR) objectives. In this scenario, the area of interest (AOI) is under control of the adversary, which engage in electronic warfare (EW) to disrupt communications and employ formidable integrated air defenses (IAD) to deny access and prevent mobility.

The ERAU and AFRL team have outlined a number of scenarios that set two teams, red and blue, against each other in a shared simulated environment. The objectives for each team are as follows: The blue team must search the hostile enemy environment, collecting intelligence and reporting intel back to base while minimizing losses of expendable resources. The red team must minimize the loss of intelligence to the blue team while maximizing blue team resource expenditure.

Scenarios range from an area of 25 km2 with 10 agents on each team, to 400 km2 and 300 agents on each team. In the simulation, the blue team consists of two entity/asset types: Intelligence, Surveillance and Reconnaissance (ISR) Drone, and Home Base. The default blue team drone is a small multi-rotor aircraft that would fall within the Department of Defense Group 1 UAV category. The Home Base represents an outpost acting as a base of operations for the ISR Drones, and is responsible for asset deployment, refueling, and acting as a repository of intelligence collected by ISR Drones. On the red team, Mobile Anti-Aircraft (Mobile AA) models an infantry soldier equipped with conventional weapons modified specifically to disable or destroy small multi-rotor drones. The Mobile AA are tasked with protecting a point around which they will patrol and engage potential targets, returning to the protection point after certain conditions are met. Mobile Jammers represent communications jamming equipment with an effective jamming range of several kilometers that may also be loaded on to vehicles and moved about the environment.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

The ERAU team was provided a set of scenarios by AFRL that would be used to evaluate the performance of the system. The team broke these scenarios down into more basic scenarios, with each basic scenario defining a particular characteristic of the larger scenario. At the start, random values are used for all parameters, and the system uses the fast-time simulator to execute hundreds of runs of a single scenario in parallel. As the scenario runs are completed, the results are evaluated and the next generation of values is defined. This process is repeated until the system converges on an adequate solution for that scenario. These values are stored and used as a baseline for the development of the values for the next scenario. Once all basic scenarios are completed, the system is run on the AFRL provided scenarios and final scores are recorded. Each AFRL scenario was run 50 times, and the score was generated as an average of the median score and the scores of the two runs one standard deviation around the median score.

3.1. Layered Architecture and Artificial Life

Our approach to this problem is highly biomimetic, and these aspects of the system can be seen in: The structure of each asset's control system and how that control system is trained, the social interactions between assets, and the resilience of the system across various environments, including those that are highly unfavorable to the asset.

The driving structural architecture is a set of strict layers. Each asset is governed by simple rules stratified into a number of vertically integrated layers. This layered architecture was inspired generally by the layered structure of the mammalian cerebral cortex, and specifically the columnar hypothesis of the neocortex, where the cortex can be thought of as a composition of modular columns of neurons. This approach does not generate a generic structure for encoding behavior like a neural network can, but instead focuses on using some of the same concepts applied to pre-defined behaviors. In our approach, each layer has a unique identifier, a set of parameters, and a standardized interface. Layers interact with each other, and can be combined via the interface, creating sets of rules that exchange structured data. This allows multiple sets of rules to interact and create emergent behavior while still retaining a well-encapsulated and strict structure to the underlying system. When used on a vehicle, the software of this control system must ultimately interface with the real world through sensors and actuators. In our approach, information from sensors and output to actuators and/or motors is considered to be a representation of the cyber-physical world boundary. Perceptions of the real world, such as those generated by sensors, cross this boundary and are ingested by the first/lowest layer, which can modify or adjust these perceptions before passing them on to the next highest layer. This scheme also applies to actions traveling from the last/highest layer downward toward the actuators and motors. This data filtering and mutation is a key concept in our layered architecture, allowing each layer to change the perceived world or modify actions for any higher layers. By allowing perceptions and actions to be added, removed, and modified, the order of layers becomes another highly effective way to adjust the behavior of the system.

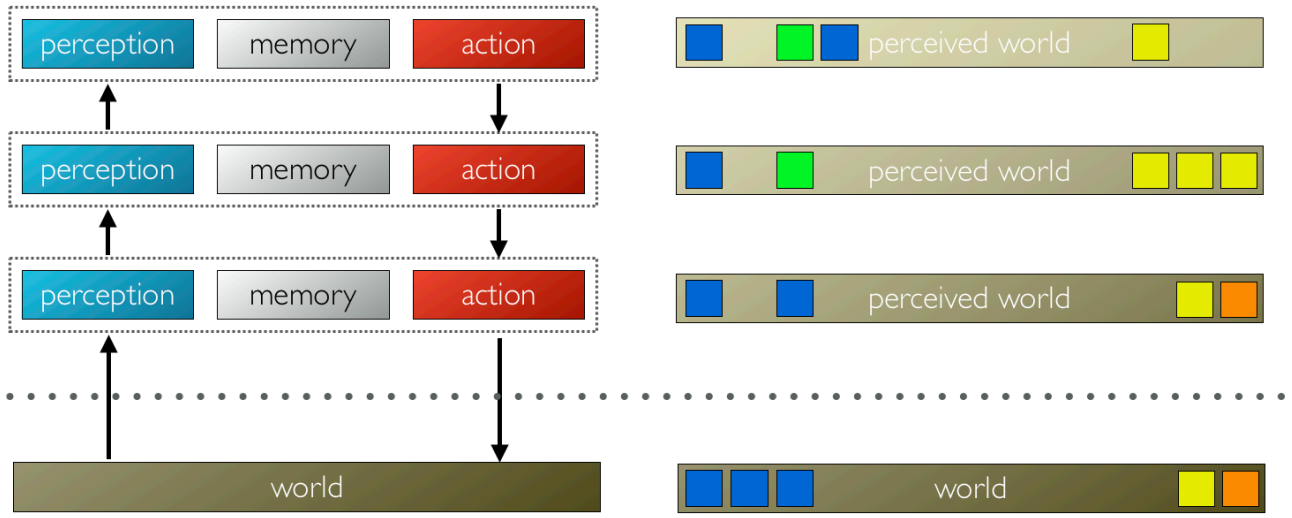

Figure 1. Layered Architecture

The primary input from sensors to our system is from a simulated machine vision system, providing a 300m radius area in which highly accurate information is provided on any assets entering within that range. This information includes a global unique identifier (GUID), entity type, alignment (team), position, velocity, and intelligence (type and quantity of intel available for capture). The remaining sensors are for intelligence gathering, and are intended to provide a means for assets to detect intelligence data attached to other assets in the environment. There are multiple types of intelligence sensor, each one with its own characteristics such as effective range and gathering rate. The duration required for the capture intelligence in the environment varies based on the type and quantity of intelligence available.

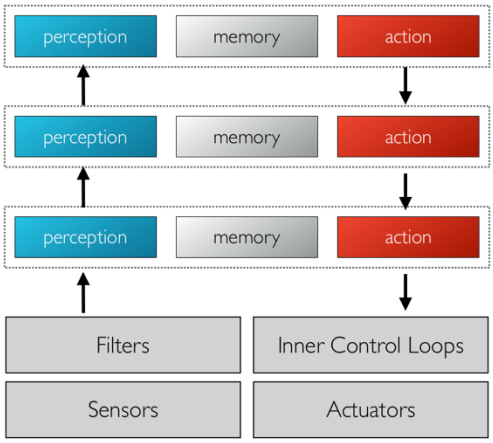

Figure 2. Layers Interfacing With Typical Drone Controller

The outputs of the system are a vector movement command, which is used directly to drive the simulated multi-rotor's direction and velocity, and communications, which are used by a simulated radio to communicate information to any assets within range. These simulated devices are meant to mirror the real devices on a typical drone controller setup, and the layers were designed with these devices comprising the bottom most layer in the architecture.

The society of assets on each team is different, as each team consists of fundamentally different assets. The blue team consists of two basic types of assets, the ISR Drone and the Home Base. The drones can be further subdivided by equipage into a large number of different sub-types. The control system on board each drone can be customized for its equipage, allowing for drones carrying high value payloads to behave in a more risk averse way than their low-value counterparts. For example, low value drones may be used to quickly survey an area, looking for targets and dangers. Using this information, the high-value drones can enter the area with better pre-

intelligence, mitigating some risk, and allowing the drones to narrow their search area. The red team consists of mobile anti-aircraft, mobile jammer, and intelligence targets. The behavior of the red team is simple and pre-defined, but is capable of dynamic action and is highly effective. The red force control algorithm alternates between patrol and pursuit behaviors depending on observed target state. By default, the red force assets patrol near a specified protection point using a semi-random pattern with a variable radius. Once an adversary enters the protection area, the defense asset will determine the probability of success in attacking the target, and engage the target if needed.

3.2. Genetic Algorithm

The team made use of evolutionary algorithms to select and adjust the behavior of the system. In our algorithm, the combined layer order list and layer parameter dictionaries represent the chromosome of each individual, while each parameter can be considered a single gene in the chromosome. This form of encoding allows the genome to human readable, and allows for crossover between multiple genomes with a diverse structure and length. Additionally, our algorithm features a parallel island model with rank based migration, real-coded gene values, blend crossover, and a self-adaptive mutation rate. Many of these features were included to reduce stagnation while decreasing the mean time to convergence of the solution. The process the system follows in using the GA is:

Figure 3. Section of Real-Value Coded Genome

- The system allocates memory and computational resources based on the number of islands and the number of assets being simulated. Each island is given a single dedicated virtualized compute unit, allowing that island to run in parallel with multiple other islands with no coupling of performance between islands.
- The algorithm ingests the pre-defined scenario, including the location and composition of the assets, and any pre-existing genomes that may have been transferred from previous training sessions.
- Each island is seeded with individuals that are instantiated from the initial genome, and optionally will pre-mutate the population to diversify the first generation's results.
- The system uses a fast-time simulation (developed by this team in a previous phase) to conduct parallel execution of the scenario on each island.
- Once complete, the system parses the simulator output and calculates individual scores for each member of the population using a predefined fitness function.
- The individual fitness scores are combined into one population fitness score, and the apex (best scoring) individuals are identified.
- The apex individuals are bred using blend crossover, and mutated using an adaptive mutation rate. Blend crossover allows for choosing gene values of children that lie on the continuum of values between the two parents (with a specified offset on either end). The mutation and breeding functions are bounded by a schema that defines the datatype, minimum, and maximum values for each parameter.

- The islands are reset to initial values, and the populations are reseeded using a combination of the apex individuals and their children.
- This process is repeated until some termination condition is reached.

The fitness function consists of eight weighted variables that represent a particular characteristic of an individual within the scenario results:

- The number of targets where intel was successfully collected vs. total available
- The time each intel target was collected with respect to total mission time
- The number of targets successfully located vs. total available
- The time each target was located with respect to the total mission time
- The amount of fuel used
- The length of time alive in the mission

Each variable is calculated based on the data in the simulation logs and is used to define the ranking of individuals within the combined population.

Later in the project, as the complexity of the scenarios and our solution increased, the team implemented staged evolution. This approach uses a set of randomized scenarios that are targeted

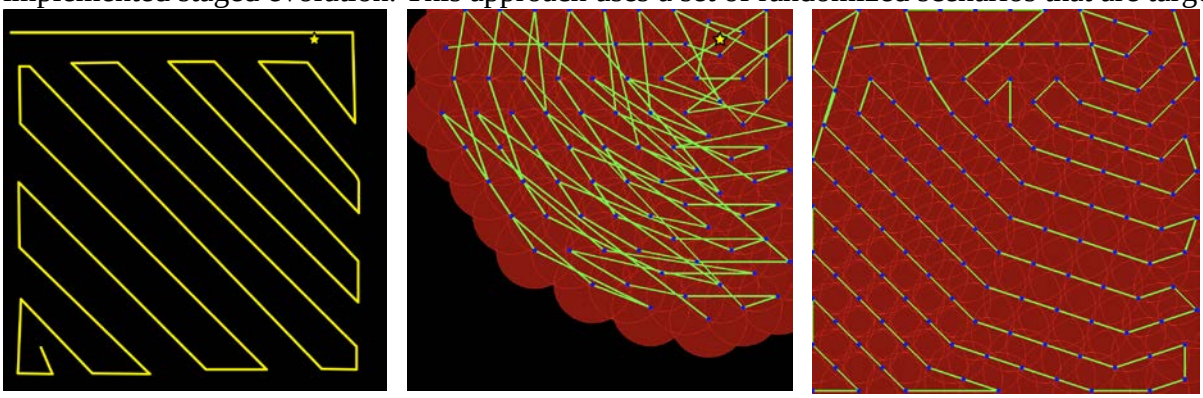

Figure 4. Search Pattern as Defined by a Human Operator, the Genetic Algorithm in Early Stage of Evolution, and the Final Output of the Genetic Algorithm

to exercise a particular portion of the genome, with the fitness function being adjusted to match the more specific behavior being developed. Once the behavior reaches a satisfactory level of performance, that section of the genome is locked. This section remains in the genome for successive runs that develop additional behaviors, and over time a complete set of behaviors encoded within a single genome is developed. If necessary sections may be unlocked at any time to allow for re-adjusting parameters that may be co-dependent. This approach allows for faster convergence on a solution and more predictable results. An example of the impact of the GA on performance of a single layer can be seen in figure 9. This figure shows the actual path and selected waypoints of a drone running several navigation layers and a layer that decides which area to search next. The navigation layers have their values protected against change by the GA, while the search layer has its values enabled for change by GA. The first picture shows what a human operator might define as a waypoint-navigation based search pattern for a drone. The next picture

shows an early output of the GA trained search layer. The last picture shows the eventual output of the GA trained search layer.

3.3. Vector Fields

Vector fields are defined by a function that can be used to calculate a vector for any two-dimensional point in the environment. This vector is then used by the system to move the vehicle, providing the vector directly as guidance. There is a vector field associated with each entity type in the environment, and as the vehicle observes an entity, it calculates the field associated with that entity. Multiple entities may be observed at once, and in this case the vector field for each is calculated separately and then combined to produce an aggregate field. When this is visualized, it naturally resembles a moving flow in space, represented below as a set of arrows in a plane with a specific magnitude (shown by color) and direction.

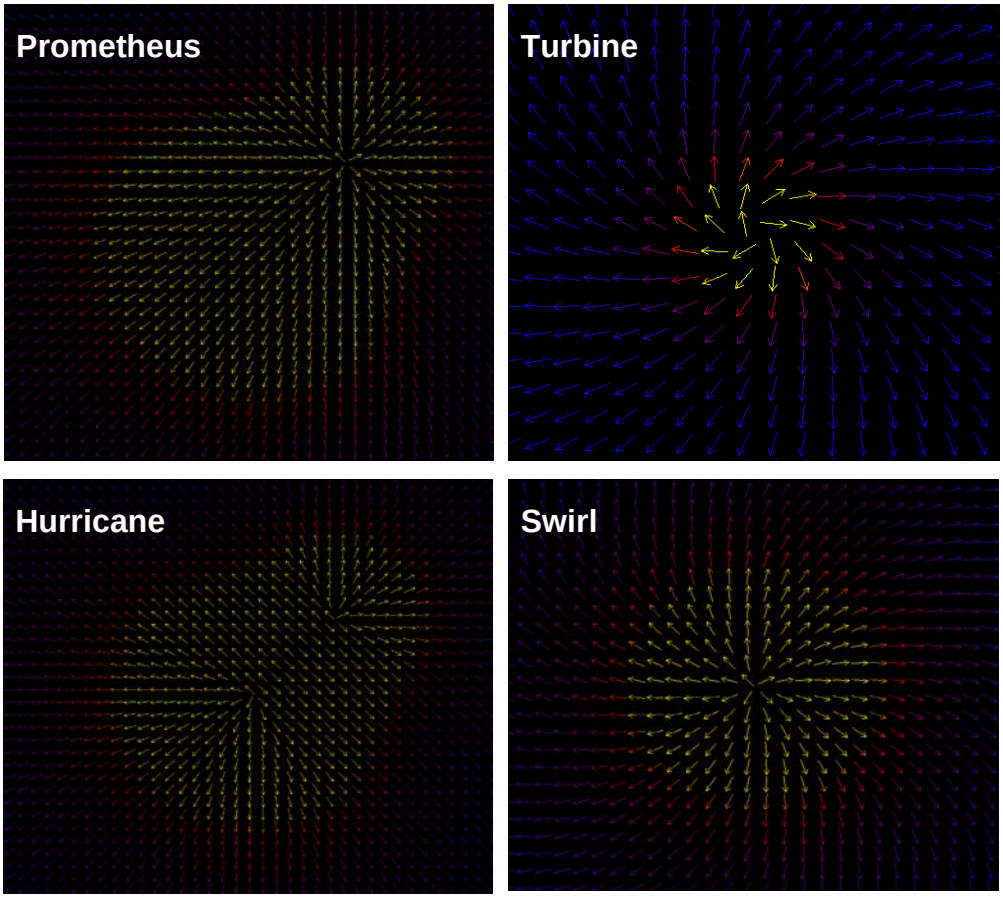

Figure 5. Vector Field Visualization Examples

3.4. Autonomy Test Environment for Python

During the course of the project, the ERAU team developed the Autonomy Test Environment for Python (ATEPy) to provide a simulation environment to execute RAS scenarios and develop solutions. ATEPy enables fast-time or real-time simulation of a large number of autonomous or human guided assets to navigate through communication denied, hostile, two-dimensional simulated environment with a binary third axis (2.5D). Scenarios for the simulation can be created from a scenario configuration file using a well-defined schema, or by using a web-based graphical user interface (GUI) to build, organize, and run scenarios. The simulation outputs comprehensive log files detailing the state of each asset during the simulation run, and the GUI can be used to watch playback of completed scenarios from a top-down view of the battle-space.

In the simulation, the blue team consists of two entity/asset types: Intelligence, Surveillance and Reconnaissance (ISR) Drone, and Home Base. The default ISR Drone provided by ATEPy is a small multi-rotor aircraft that would fall within the Department of Defense Group 1 UAV category. The Home Base represents an outpost acting as a base of operations for the ISR Drones, and is responsible for asset deployment, refueling, and acting as a repository of intelligence collected by ISR Drones. Each asset is configurable, allowing for various equipage choices prior to the start of the mission. The types (classes) of equipment module provided by default in ATEPy are: Fuel, Vision, Sensor, Communications, Jammer, and Weapon. Assets have physical limitations defined by the scenario. These limitations include durability, number of racks and hard-points with associated power and quantity limitations, payload capacity, and cost. As equipment is added to the asset, performance and cost are adjusted accordingly. Additional configurations can be created by adjustments to the scenario file. The home base can be configured to perform any task during the mission, such as mission-level coordination of ISR Drone equipage and mission parameters between sorties. The Home Base and ISR Drone both use the rack and hard-point configuration scheme, although the default number of hard-points available to the Home Base is significantly higher than those available to the ISR Drone.

Multiple default fuel modules are available, including various battery configurations and a human stamina model. The ISR Drones can be equipped with battery pack type fuel that depletes based on a medium-fidelity battery discharge and vehicle energy model. The battery type fuel can be replenished at the Home Base, while the human stamina type model regenerates continuously over time. Additional or adjusted fuel limitations can be created by making adjustments to the scenario file. Should the asset deplete its fuel or reach zero durability, the asset is marked as destroyed, and is no longer able to perform any actions within the simulation (i.e. communications, movement, weapons usage).

The vision module is required equipment on all assets, and by default provides a 300m radius area in which highly accurate information is provided on any assets entering within that range with every step of the simulation. This information includes a global unique identifier (GUID), entity type, alignment (team), position, velocity, and intelligence (type and quantity of intel available for capture). Unlike communications, vision is unaffected by jamming.

Sensors, or intelligence collection sensors, are intended to provide a means for assets to detect intelligence data attached to other assets in the environment. Like vision, they are unaffected by jamming, but are by default more limited in the rate at which information can be gathered. There are multiple types of sensor, each one with its own characteristics such as effective range and intelligence gathering rate. For a sensor equipped to the friendly asset to collect intelligence from an enemy asset, the sensor must match the type of intelligence available on the enemy asset, and be within effective range. The duration required for the capture varies based on the type and quantity of intelligence available. While configurable to any value, default scenarios are configured to use sensors with collection rates ranging from one intelligence capture per second to upwards of two minutes per capture. Once the intelligence is captured, the intelligence is stored in the asset's memory where the asset may choose to transmit the intelligence back to the Home Base or another ISR Drone. For the purpose of scenario evaluation, once the intelligence reaches the Home Base, the simulation considers that intelligence to be successfully collected.

Communications modules allow assets to communicate with each other using a medium-fidelity communications model. Each asset, limited by available hard-points, may be equipped with any number communications modules. The communications module allows for adjustment of power, frequency, and spectrum bandwidth. Data rate between any two assets is a function of signal to noise ratio (SNR), frequency, bandwidth, power, antenna directivity, and distance. Communications modules abstract the effects of data encoding, error correction, frequency hopping, and spread spectrum. Jammers can be added to scenarios, and may be stationary or mobile. Jammers are configured similarly to communications modules. A simple default jamming module is provided that introduces a higher level of noise across the entire spectrum available to the communications modules. Increased noise changes the SNR, and thus lowers the data rate available between affected communications modules. Once a pre-defined minimum cutoff is reached, no communication is possible.

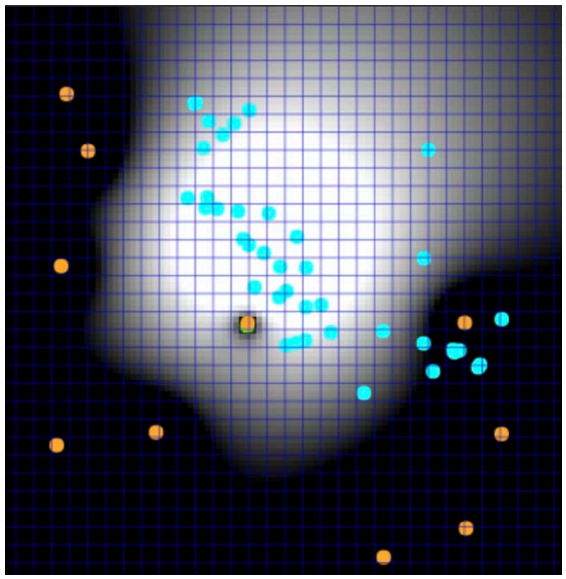

Figure 6. Visualization of Jamming in ATEPy

Weapons modules allow assets to damage other assets in the environment. ATEPy provides various default weapon modules that simulate small arms fire from either mobile or fixed assets with low fidelity. The simulation takes in to account the movement of the asset firing the weapon as well as the relative distance to and transverse velocity of the target. Targets that are moving with a high transverse velocity may be harder to hit, while those traveling directly toward or away from a weapon will be easy to hit.

ATEPy is designed to run inside a containerized environment on Linux. This is a form of virtualization at the operating system level, allowing multiple containers to run on the same Linux

instance and take advantage of Linux kernel resource isolation features. ATEPy uses an API over a network interface for simulator command and control. The API is available as either a RESTful web service endpoint or JSON-RPC calls over WebSocket. All functionality is available over both connection methods, with the WebSocket having a higher theoretical maximum number of requests per second. This architecture allows for massively parallel simulation and ease of use on cloud-based systems.

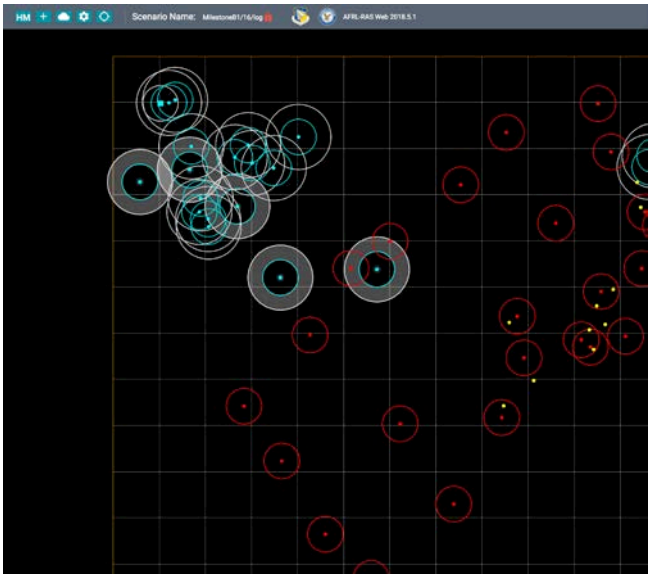

Figure 7. Screenshot of Scenario Playback in ATEPy Web Interface

In order to execute scenarios within ATEPy, scenarios are first defined by the user, either using the GUI or manually via a standardized JSON file, specifying the assets in the environment and any simulation specific settings such as tick rate and mission length. Reconnaissance data for each scenario can be generated procedurally. The user can select an already defined scenario, and then choose to generate recon data that will randomize and reduce the scenario data related to the enemy force by a specified amount. Assets can ask ATEPy for scenario details including map size, mission length, step time, and asset locations. The information returned is limited; if a friendly asset requests to view the current scenario, the response will include all friendly asset starting positions and equipage but will not include any true enemy force information (with the exception of reconnaissance data). If there is recon data the simulator will include what recon was provided in the scenario definition.

To run a scenario, users execute the container and optionally provide parameters defining the scenario input and preferred log file output. Once ATEPy is running, the user may use the network API commands to control execution of scenarios. Once a scenario simulation has been commanded to begin, ATEPy waits until all assets have submitted their first commands (called intents), and then begins the simulation loop. Each iteration in the simulation loop is called a step. The step is the minimum atomic unit of time within the simulated environment, the length of which can be defined by the user and is referred to as the step time. The simulation loop performs steps until the scenario's defined duration has been reached or conditions within the system prohibit the simulation from continuing (such as all assets being destroyed). The amount of real time required to execute each step depends on a number of factors, and is dominated by the number of assets being simulated, the efficiency of the algorithms used for asset control, and, if used, waiting on network input. Each iteration of the loop waits for commands to be enqueued from all assets being simulated. Any assets being controlled via the network interface will add network latency to the real time required for step execution. After a scenario has been concluded, the results are provided to the user in the form of a log. These logs are in JSON format, and range in size from 10-50 megabytes per hour of simulation time. Each log provides a comprehensive record of what occurred in each simulation step.

The simulator also provides an option to allow for human operators alongside autonomously controlled assets. In this mode, the Home Base is equipped asset code allowing a human operator in conjunction with a communications module to relay commands to friendly assets. Human interactions can assist in realistic scenario execution, investigating how operators would interact with the proposed semi-autonomous craft in the simulated environment. Since the operator interacts with the assets though the Home Base communications module, the situational awareness of the operator is realistically restricted to the equipage of the assets and their operational environment. This way, the environment portrayed to the user is truthfully restricted to the capabilities of the assets, including bandwidth, latency, jamming, and vision limitations.

Finally, the simulation offers an observer channel that offers an unrestricted view of the environment. This feature is intended to be used for performance evaluation and as a visualization of scenario progress from an unrestricted view. Additionally, observers have the option to create and control assets in a way that are unrestricted by communication, providing pop-up targets and intelligence, allowing for the creation of highly adverse dynamic conditions.

The simulator core is structured around a main simulation loop, with each iteration referred to as a single simulation step. The API provides a step command to advance the loop, control assets, and receive inputs regarding simulation state. This step function is executed by the user, providing commands for each action to be performed by assets within the simulated environment such as movement and communications. Once the simulation has simulated a single simulation step, the function returns with the actions performed by the simulation which are called perceptions. Perceptions contain the asset's position, velocity, vision and sensor inputs, incoming messages, and asset state.

The length of time required to execute each step determines how much faster than real time the simulation can run. ATEPy is capable of executing the step in either a synchronous or asynchronous mode. In synchronous mode, the simulator blocks with each step, waiting for commands for every asset before continuing. In asynchronous mode the simulator blocks with each step, but advances either when commands have been sent for every asset or the real-time constraint has been exceeded. When the real-time constraint has been exceeded, the simulation continues to perform the next simulation step with the last received commands. Asynchronous mode is useful to ensure asset simulations are running in a realistic timeframe. Should the simulator step rate be slowed to a specified duration of real time, each command is required to be sent within that duration of real time. Assets that do not meet the constraint will continue to use the last command sent. Assets that have missed deadlines may issue commands in later steps. This allows the rate of simulation environment evaluation and the rate of asset logic evaluation to be decoupled.

Each asset can communicate their commands either individually or in a batch. When communicating all commands as a batch, all assets in the simulation provide all of their commands using a single call to the simulator. The simulator then responds with all simulation results as a response. This method performs only one call, which provides increased performance by reducing the amount of network overhead and the total number of requests. When sending all commands individually, individual step calls are performed for each asset. Each response provides only the results for that individual asset. In addition to being easily scalable, this method is useful for

running the asset simulation code in discrete environments to ensure that no memory is shared that would bypass the rules of the simulation (communications jamming, bandwidth limitations, etc). Using individual commands could also be used in hardware-in-the-loop (HWIL) configurations where specific hardware is used for each asset.

Any combination of these two methods is also possible, where two separate implementations that do not integrate can submit either batch or single commands for their own groups. For example, a group of friendly assets and a group of enemy assets are remotely controlled, each in their own discrete environment.

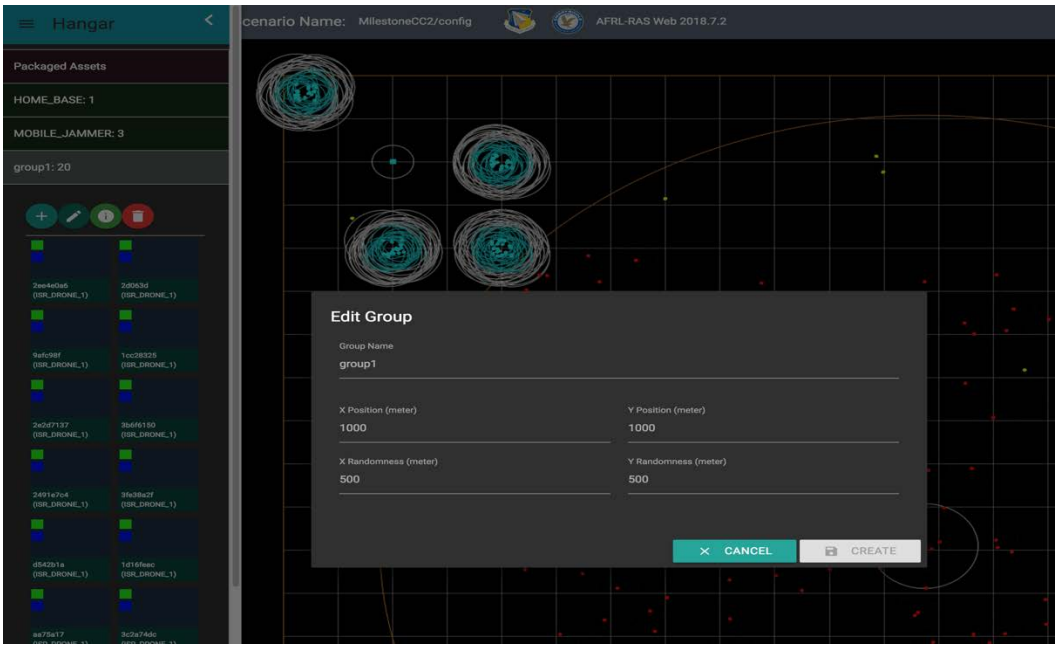

Figure 8. Creating Groups in the ATEPy Web Interface

Since limited information is provided to each asset, a comprehensive simulation log is provided post-simulation containing detailed information on what occurred during the entire simulation, even with details not provided to any asset during the simulation run. This log contains a comprehensive account of what occurred during each step and in some cases can include metadata such as asset event logs. Reducing log file size is important, not only for minimizing storage usage, but also for increasing simulator performance as writing to storage is slow. All logs contain a copy of the scenario configuration, and a log for each asset. These logs contain the asset's position, inbox, outbox, vision, and simulation events. Simulation events contain events such as weapons fire, weapon damage, message jammed, target detected, target acquired, etc. Each message, depending on its use, may contain details for each event. For example, the weapons fire event contains details such as intended target and probability of hit. The log results are designed to be detailed enough to provide a full mission length replay, either for mission playback or as input to an analytical tool for data mining and asset performance analytics. Any analytics can be used time-independent of the simulation runtime and without any need for tools or plugins to be injected into the simulator itself, allowing for stability in maintenance and flexibility in implementation.

The default enemy asset defense is controlled by an algorithm that alternates between patrol and pursuit behaviors depending on observed target state. By default the defenders patrol near a specified protection point using a semi-random pattern with a variable radius. Once an adversary enters the protection area, the defense asset will determine the probability of success in attacking the target, and engage the target if needed. Upon engaging the target, the defender will pursue it towards its predicted location based on the target's current velocity vector. This style is designed to intercept the target before it reaches the protection point, with the intent of causing the target to change path to avoid the enemy. Once the defense asset is within firing range, the probability of hitting the target is taken into account before expending ammunition. The defender may at any time decide the target is no longer worth pursuit and can return back to the protect point or switch to another target.

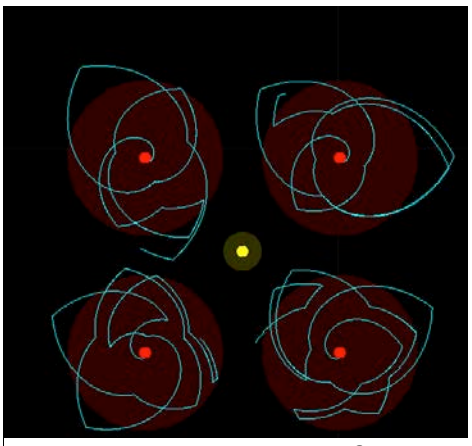

Figure 9. Visualization of Patrol Patterns for the Red Force

3.5. Artificial Life Layers

The team has developed over 40 layers that can be used by the system, which when combined into a single genome and converted from our real-coded dictionary to a classical binary structure, represents an approximately 5,000-bit binary string. Of these layers, ten were selected for the GA to train in the latest AFRL-generated scenarios. To support a discussion of how the interactions between layers works to generate robust, adaptable behavior, we will provide a brief overview of three layers and their parameters.

The Advanced Boids layer is based on merging concepts from various flocking algorithms. The rule set is simple: Move to the center of group, match the velocity of neighbors, avoid collisions, scatter away from the group for a period of time when danger is near. This implementation of the algorithm uses ten parameters. The affinity for matching velocity governs to what degree and how quickly an asset will match the observed velocity of its neighbors. The group cohesion factor determines how strongly the asset will be attracted to the center of the group. The group separation factor provides an inverted form of the cohesion factor that can be used when the asset enters the scatter state. The danger scatter factor determines the magnitude of velocity away from the danger that the asset will apply to itself when scattering. The danger scatter entities list contains a list of entity types that, when observed, will cause the asset to enter the scatter state. The scatter entity distance determines a maximum distance the entity can be from an entity

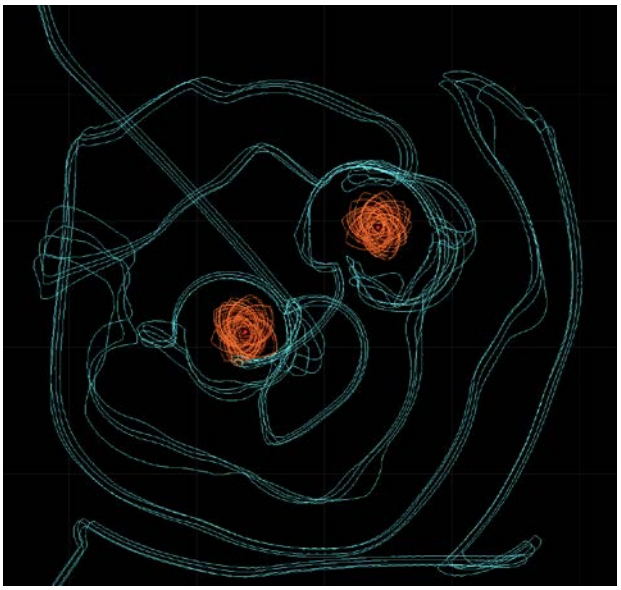

Figure10. Visualization of a Three Member Flock Traversing a Map While Avoiding Red Force Contact

listed in the scatter entity list before it enters the scatter state. The scatter duration is how long to stay in the scatter state. The maximum turn factor, and maximum speed variables limit how quickly the assets are allowed to move and turn – since they are capable of almost immediate direction changes, tuning these variables can produce wildly different flocking behaviors. Finally, the range variable, if used, defines a maximum range an asset can be from the group before it must classify itself as disjoint from the group. This layer allows agents to group together while still retaining some agency in movement about the environment. It is also somewhat effective in overcoming the enemy Mobile AA, since the enemy will be forced to chase just one of the friendly assets, or return to patrolling. Rapid switching between normal and scatter states often results in the enemy force being led off-target, allowing the blue force a narrow window of opportunity to capture that target.

The Simple Predictor layer provides predictions for the location of assets in the environment. For any observation that was previously made of an asset in the environment, if that asset is no longer being observed, the layer provides a simple prediction of where that asset may be now. This layer uses two parameters: The observation age and the friction factor. The observation age limits the maximum length of time the predictor will provide updates to an asset location based on the time of the last observation. The friction factor degrades the predicted velocity of the asset to zero over time. The intent of the predictor is to provide a lightweight algorithm for estimating the location of previously observed assets, focusing entirely on generating perceptions, and does not provide any action output.

The Comms Vision layer, inspired by other work in simple social structures for artificial life algorithms, communicates vision sensor data to neighbors using a simple data structure, and merges received vision data into the asset's own vision data. This layer uses two parameters: Decay rate and reporting rate. The decay rate determines how long an observation that was received from a remote asset will remain in the perceptions of the local asset. The reporting rate limits how often assets report vision information. Comms Vision has a large potential for influencing the behavior of assets, as it vastly expands the awareness of individual assets to include all other assets currently in communication range. It is important to note that what the other assets report does not necessarily represent perceptions being passed from the first (cyber-physical) layer, and thus could be modified by any number of layers previous to being communicated.

To illustrate the function and impact of the architecture, consider the following scenarios and outcomes that the ERAU team used in developing the system: Four teams of assets operating over a 100km^2 area densely populated by Mobile AA and Intel Targets are each assigned a quadrant to search for targets. First, we use an instantiation of the control system executing with a basic set of rules to allow the vehicle to navigate the environment, such as linear guidance to predefined waypoints in each quadrant, with the addition of the Advanced Boids layer to keep the groups together. The outcome of this scenario was that each group would travel to the predefined points, encountering a number of Mobile AA and Intel Targets along the way, collect targets that were seen by the group, and return home.

In a different instantiation of the system, almost identical to the previous, but with the addition of the Comms Vision layer immediately below the Advanced Boids layer, the outcome was changed significantly. Whenever a group would encounter Mobile AA or Intel Targets, that vision data was

reported to all groups and processed by the remote Comms Vision layers on the other assets into world perceptions. These perceptions were then passed to the Advanced Boids layer. Without needing any modification to the data, or being aware of any change to the data over previous executions, this provides the layer with an expanded set of data to work on beyond what the vehicle's own sensors are able to provide. The effect of this was that the assets gained the ability to scatter before encountering dangers themselves, were aware of the location of several targets in their quadrants before arriving to that location, and, without the addition of a search function, were able to effectively search their quadrants for targets due to the high level of scattering.

In a third instantiation of the system, the team added the Simple Predictor layer below the Comms Vision layer. The impact of this addition was that now, not only were the assets reacting to observations well beyond the limits of their own sensors, they were reacting upon predictions of locations of enemy and friendly assets made by their allies. This information was then propagating to the rest of the group, and would update and degrade in waves centered on the location of the last observation of the asset. When observing this behavior, when an enemy asset is observed moving in the direction of another group, the other group of friendly assets will scatter and flee from the predicted location of the enemy, even before the enemy asset has decided to chase that group. When communications are unhindered, the blue force avoided the red force such that it rarely encountered the blue force on patrol, and only when the blue force had decided to engage a target. Although most of our scenarios involve heavy communications jamming, some of these benefits can still be realized, although the propagation delay increases with the level of jamming in the area. It is also worth noting that in scenarios with extremely dense red force placement, the GA will optimize to reduce the effect of this layer on the group, since predictions become increasingly unreliable as the number of random asset interactions increases.

4. RESULTS AND DISCUSSION

The results for each AFRL-provided scenario are presented in this section. Each AFRL scenario was run 50 times, and the score was generated as an average of the median score and the scores of the two runs one standard deviation around the median score. Each scenario consists of two types: the "Permissive" type and the "A2AD" type, where the A2AD type presents a version of the Permissive type with added elements to represent an A2AD environment such as additional jamming, increased red force numbers, and reduced pre-intelligence on enemy asset numbers and locations. Each scenario increases in size and asset density over the previous scenario.

4.1. Scoring

The final score is determined by adding the percentage component variable scores. For any score involving identifying the position of a red force asset, the blue force agent must report the identified position of the red force asset to the home base using the radio. If the blue force asset is destroyed before it is able to communicate the position of the red force asset, the information is lost and will not count toward the score. If the same information, either for position or intel cache values, is reported multiple times, only the first report is counter toward the score. If an agent is not at home base when the mission is over it is counted as lost, and will reduce the blue force survival score. When the scoring was provided by AFRL for the final milestone of the project, the ERAU team noted that there was no mechanism for the agents within the simulator to flag the simulation or mission as complete. Without such a mechanism, the SM variable becomes a fixed deduction of 15% from the score. The ERAU team developed and deployed to ATEPy a way for

the agents to flag the mission as complete, but given the lack of remaining project schedule, many teams were unable to take advantage of this feature. The ERAU team adjusted an existing layer to provide this flag to the simulator based on several introspection variables such as remaining fuel, distance to planned targets, and pre-intelligence, added the layer to the autopilot, then re-ran the evolution process. This allowed the team to regain a few percentage points using the SM variable in some scenarios. However, given the size and time limit of most of the provided scenarios, ending the mission earlier than just a few minutes or even seconds before the allotted time limit proved to be unfeasible while still optimizing for the other component scores.

Table 1. Scoring Provided by AFRL

Variable	Relative Weight (%)	Description
IAPI	30	% identified positions of Red Intel Assets
ISVC	25	% Intel total value captured
SM	15	% of total minutes spare after mission completion
BFS	15	% of Blue Force Assets Survived
AAPI	10	% identified positions of Red Anti-Air
JAPI	5	% identified positions of Red Jammers

For all scenarios, the team used the same agent code and configuration parameters, with the only difference being providing pre-intelligence to the agents if it was available.

4.2. Scenario 1

Time Limit: 30 minutes of simulation time
Permissive: Default ERAU ATEPy asset values. 6x6 km sized environment. Blue Force home base is in the center of the environment. 15 Blue Force assets, all have the same sensor type. Red Force: 5 AA, 5 jammers, and 20 intel sites randomly placed in the environment. All Red Force asset types, intel types, and locations are known beforehand. All 20 intel sites have a single intel item with a value of 1.0.
A2AD Event: Red Force increased to 25 anti-air assets. Blue comm power reduced to half. None of the intel site information is known beforehand.
Scores: Permissive: 87.7, A2AD: 85.9

4.3. Scenario 2

Time Limit: 45 minutes of simulation time
Permissive: Default ERAU ATEPy asset values. 10x10 km sized environment. Blue Force home base is situated at extreme upper-left. 30 Blue Force assets have sensor type A, 10 Blue Force assets have sensor type B. Red Force: 20 anti-air, 10 jammers, 40 intel sites, spread randomly in NE, SE, and SW quadrants. All Red Force asset types, intel types, and locations are known beforehand. Half of the intel sites are of type A and the other half are of type B. Each intel site contains three intel items. Type A intel items are valued at 1.0 and Type B intel items are valued at 2.0.
A2AD Event: None of the intel site information is known beforehand. Red Force increased to 60 anti-air assets. Half of the Red Force AA sites are able to move 3% faster than the default Blue Force speed.
Scores: Permissive: 85.3, A2AD: 76.8

4.4. Scenario 3

Time Limit: 75 minutes of simulation time
Permissive: Default ERAU ATEPy asset values. 16x16 km sized environment. Blue Force home base is situated at extreme upper-right. 15 Blue Force assets have sensor type A. 15 Blue Force assets have sensor type C. 10 Blue Force assets have sensor type D. Red Force: 25 anti-air, 10 jammers, 50 intel sites, spread randomly in NW, SE, and SW quadrants. All Red Force asset types, intel types, and locations are known beforehand. 20 intel sites are of type A, 20 are of type C, and 10 are of type D. Each intel site(target) has four intel items(cache), each of which are valued at 1.0.
A2AD Event: None of the Red Force information is known beforehand. Blue Force assets are unable to refuel. Half of the Blue Force is reduced to 50% of fuel capacity. Red Force increased to 50 anti-air assets. Jamming power is increased from 4W to 10 W.
Scores: Permissive: 83.0, A2AD: 72.1

5. CONCLUSIONS

The ERAU team has developed a multi-layer, artificial-life-based autopilot trained to operate in an Anti-Access Area Denial (A2AD) environment while minimizing the use of expendable resources and maximizing mission performance with respect to predefined mission objectives. For most scenarios the system is capable of attaining very high scores in all categories (sans mission time, which was addressed in the previous section). The team was able to develop a system capable of an average score of 81.8/100 on the AFRL provided scenarios. The behavior of the ERAU system is highly dynamic, and adapts to new scenarios via limited emergent behavior generated via interaction between the agents and within the agents themselves via intra-layer interactions.

The system performs well on a wide variety of scenarios that all share similar tasks (gather intelligence, avoid enemies, communicate information to base) and can adapt to a wide variety of asset performance adjustments including but not limited to changes such as increased enemy jamming range, increased enemy speed, decreased blue force range, and increased enemy weapon effectiveness. However, if new scenarios are developed that differ significantly enough from the original to induce performance degradation, the system can be retrained from the existing baseline by providing the current genome as the input to the first generation of the new population, preserving as much of the learning from the previous scenario types as possible. Additionally, due to the inherently modular architecture of the system, layers can be added or removed as needed to meet new mission requirements while utilizing the existing training and simulation framework. Addition or removal of layers may generate new values in the agent genome, but the design of the system allows for the genome to be easily extended and even combined/bred with genomes containing different layer parameter definitions.

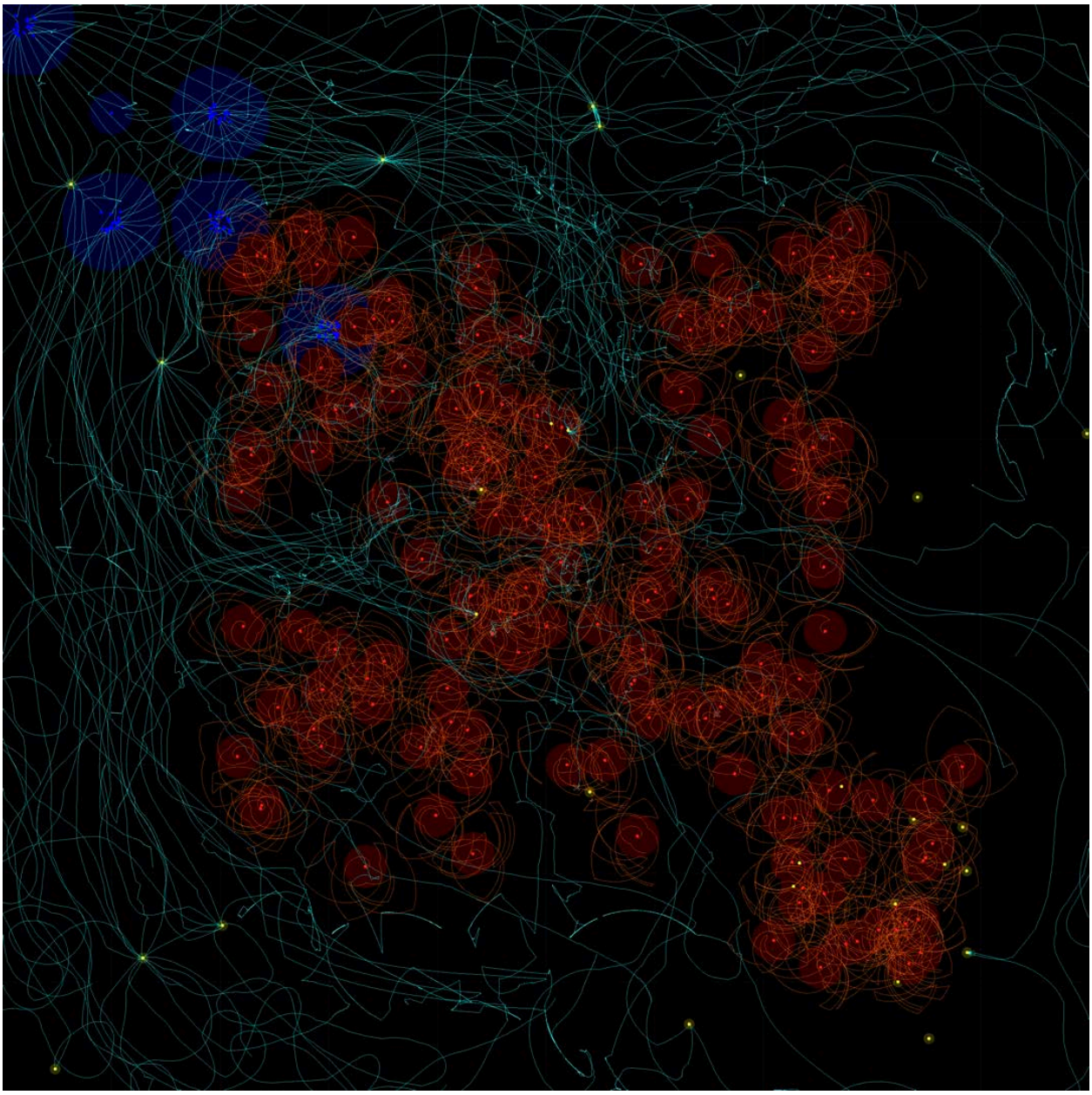

Figure 11. Visualization of Blue and Red Force Movements for a Mission

6. PUBLICATIONS

DASC 2017: “Multi-Layer Model of Swarm Intelligence for Resilient Autonomous Systems”, Clifford, Towhidnejad, Neighbors, Verenich, Staskevich

SPIE 2018: “Integrated Swarm Intelligence Simulation for Resilient Autonomous Systems”, Clifford, Neighbors, Towhidnejad

DASC 2018: “Highly Flexible Swarm Intelligence Algorithm for Resilient Autonomous Systems”, Clifford, Towhidnejad, Neighbors, Verenich, Staskevich

“Implementing Social Behavior within a Resilient Autonomous Systems Simulation”, Colborn, Garfield, Clifford, Towhidnejad

“Using Case-Based Reasoning Algorithm to identify Training Secenario”, Fu, Towhidnejad, Clifford, Neighbors

7. LIST OF ACRONYMS

A2AD	Anti-Access Area Denial
AA	Anti-Aircraft
Alife	Artificial Life
AOI	Area of Interest
API	Application Programming Interface
EW	Electronic Warfare
GA	Genetic Algorithm
HWIL	Hardware-in-the-Loop
IAD	Integrated Air Defenses
ISR	Intelligence, surveillance, reconnaissance
JSON	JavaScript Object Notation
RAS	Resilient Autonomous Systems
REST	Representational State Transfer
RPC	Remote Procedure Call
UAS	Unmanned Aircraft System