Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

DESIGN AND SIMULATION OF A WIDE AREA SEARCH MISSION: AN IMPLEMENTATION OF AN AUTONOMOUS SYSTEMS REFERENCE ARCHITECTURE

David King
David Jacques
Jeremy Gray
Katherine Cheney

Department Of Systems Engineering & Management
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson Air Force Base, OH 45433, USA

ABSTRACT

The implementation and testing of autonomous and cooperative unmanned systems is challenging due to the inherent design complexity, infinite test spaces, and lack of autonomy specific measures. Simulation provides a low cost alternative to flight tests, allowing researchers to rapidly iterate on the design before fielding. To expedite this process, an Autonomous System Reference Architecture allows researchers to utilize existing software modules to rapidly develop algorithms for autonomous systems and test them in included simulation environments. In this paper, we implement ASRA on a cooperative wide area search scenario as a test bed to study ASRA's utility for rapid prototyping and evaluation of autonomous and cooperative systems. Through a face centered cubic design of experiments, selected autonomy metrics are studied to provide a response surface model to characterize the system and provide a tool to optimize mission control parameters and maximize mission performance.

1 INTRODUCTION

A major technology that has spread rapidly in both the consumer and defense industries is small unmanned aircraft with autonomous capabilities. There is great potential value in further developing autonomous capabilities to perform missions of higher complexity, but this higher complexity comes with a requirement for more rigorous testing. Here we present and implement an Autonomous Systems Reference Architecture (ASRA) to provide a development environment addressing the complexity of designing, simulating, and testing autonomous systems.

ASRA provides autonomy researchers with the tools necessary to quickly develop new autonomous systems, by reusing existing autonomous components and utilizing model based systems engineering approaches to design and describe the system. Additionally, ASRA will provide the flexibility to utilize many different simulation environments along a spectrum of efficiency and fidelity. The modularity of ASRA allows testing to seamlessly progress from tests run in simulation to live tests by swapping simulated and physical components. ASRA also provides test modules to aid researchers in designing and carrying out rigorous automated testing that evaluates the system's ability to achieve mission and autonomy objectives.

A wide area search (WAS) mission with multiple cooperative agents forms the test case for this research and a layered autonomous architecture is designed and implemented to achieve the mission. A face centered cubic design of experiments (DOE) is performed over four scenarios of varying cooperation levels. Test results created a response surface model that characterized the system, providing prediction and optimization of mission performance based on system configuration.

2 BACKGROUND INFORMATION

2.1 Cooperative Wide Area Search

One of the many applications of unmanned aerial systems (UAS) is wide area search with the goal of efficiently searching a large area to detect and identify targets in an unknown environment. There are multiple characteristics that define these scenarios such as the search area size and search pattern, as well as target density, distribution, and movement. Two metrics commonly used to evaluate wide area search and attack scenarios are area coverage rate and false target detection rate (Jacques and Pachter 2004), with the latter being largely dependent on sensor performance. These sensors can be modeled with a confusion matrix which uses empirically derived probabilities to determine how encountered targets are sensed. For a binary confusion matrix, the designer defines the sensor's probability of true target recognition, P_{TR}, and probability of false target recognition, P_{FTR}. Probabilities of 1 describe a perfect sensor and their complements determine the probability of incorrect target recognition, simulating Type I and Type II errors. This modeling approach is readily expandable to scenarios with more than two types of targets.

The WAS problem's main limitation of resources can be addressed through the application of multiple agents assigned to the same mission area. Previous research applied to the munition problem has shown that the application of cooperation to a problem does not guarantee improved results and thus should be thoughtfully applied (Dunkel 2002). It has been found that cooperation yields the greatest improvement when sensor performance is relatively poor (Dunkel 2002) and when the number of deployed agents is scaled according to the target density (Park 2002).

When designing a cooperative system, the distribution of decision making is a key component that defines the performance during mission anomalies such as loss of agents or communication. A decentralized control structure allows agents to share decision making responsibilities, making the system more robust to the loss of an individual agent. With decentralized control, each agent must run decision making logic such as an algorithm that calculates the utility for each agent to go confirm a target or alternatively to continue searching for undeclared targets, choosing the task with the highest utility (Gillen 2003).

2.2 Autonomous Architectures

Autonomous system architectures have progressed from deliberate "sense, plan, act" loops to reactive architectures capable of faster execution, but lacking higher level planning (Murphy and Arkin 2000). While each of these architectures have their applications, the common standard first proposed by Gat (1998) combines them into a 3 layer architecture with a deliberator, sequencer, and controller to provide a system that can keep up with dynamic environments but still achieve high level goals. The deliberator layer develops a world model to drive high level planning and sends associated goals to the sequencer. The sequencer executes these goals by selecting the appropriate controller behavior, or sequence of behaviors that achieves the current goal, monitoring the behavior for anomalies, and then reporting to the deliberator the status of the current goal, which can be considered in the deliberator's planning. The controller executes the specified behaviors, converting sensor inputs to actuation outputs. A fourth layer, the coordinator, can be added for multi-agent cases above the deliberator to act as a mediator between agents, processing information on all agents, determining cooperation level utilities, and passing necessary information to the deliberator, enabling high functioning, multi-agent integration (Hooper and Peterson 2009).

2.2.1 Unified Behavior Framework

The Unified Behavior Framework (UBF) (Woolley and Peterson 2009) standardizes the behaviors and their interfaces that run in the controller. The UBF structures behaviors in a composite pattern which allows for behavior reuse, where composite behaviors can be built from a set of atomic leaf behaviors. By arbitrating multiple behavior outputs into a single output, composite behaviors achieve goals that the individual behaviors on their own do not. Behaviors are encapsulated in a standard form such that each

behavior is externally handled identically, given a perceived state input and returning an action output. The UBF offers a flexible environment to adjust behavior control structures during execution as the appropriate structure or combination scheme for different tasks may vary.

2.3 Autonomous Systems Reference Architecture

The Autonomous Systems Reference Architecture aims to provide an environment for autonomy researchers to quickly develop complex autonomous systems in a variety of domains using reusable and modular components (Gray and Jacques 2019). The reference architecture is modeled in SysML using the model based systems engineering tool, Cameo System Modeler. This approach provides multiple levels of abstraction and brings out details of the architecture, interfaces, and concepts.

At the highest level of abstraction, the autonomous system interfaces with its environment through its action outputs, communication with other agents, and environmental precepts. The next level, shown in Figure 1 models the three levels of an agent system. The autonomy layer consists of a software agent, where the autonomy architecture resides, which interfaces with the hardware layer through the hardware interface layer. This interface layer moderates their interactions with a standardized messaging structure which allows for modularity of hardware and autonomy layers. For the hardware layer, ASRA has existing modules to interface with the Ardupilot Software in the Loop (SITL) autopilot and a lightweight point mass simulator, referred to herein as the particle simulator. Researchers can transition from a simulated environment to a real world environment by simply swapping out the hardware layer. For example, Ardupilot SITL is a software representation of the Ardupilot autopilot in a simulated world environment, so researchers can make this change and transition from simulation to flight testing without any major modifications to the rest of the software. Additionally, this provides a digital twin capability, where the autonomous software agent can run simultaneously on a physical and simulated agent, with the main difference being what is running in the hardware layer. The next lower abstraction level in Figure 2 models an instantiation of the software agent. This instantiation utilizes a four layer architecture with complex perceptors to develop world perception from sensor information; this level could be implemented with other architectures such as a simple reactive controller, or reinforcement learning.

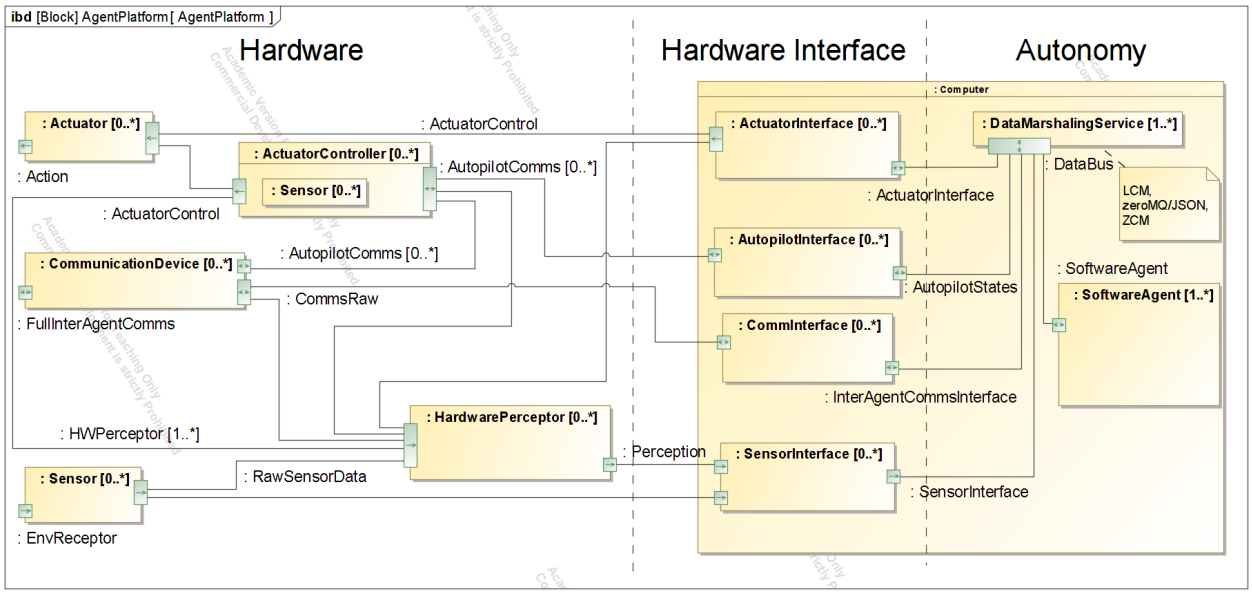

Figure 1: Model of an agent system architecture with the Hardware Interface Layer handling the interface between modular Hardware and Autonomy Layers.

3 COOPERATIVE WIDE AREA SEARCH CONCEPT

A simplified version of the WAS concept was implemented, where one or more agents searching a defined area attempted to find all targets in search mode to get an initial classification, and refine that classification with a confirmation at a lower altitude (notionally, to improve sensor resolution). The confirmation could be performed by either the original searching agent in a single agent case or another agent in the cooperating cases. This provided a mission with competing goals: to efficiently find and accurately classify as many targets as possible in a given search area.

All homogeneous multi-rotor agents had identical binary confusion matrix sensor models with circular field of views. The confusion matrix probabilities were degraded based on a chosen best case Ground Sample Distance or minimum altitude. This resulted in an altitude where flying higher increased area coverage rate but degraded sensor performance, but flying lower than this minimum altitude did not improve sensor performance any further than the best case. The minimum altitude was selected as the agents' confirm altitude, so the higher search altitude diminished sensor quality by some factor. Only the first target appearance in the field of view prompted a detection or classification declaration.

A distribution of real and false targets was implemented to test the sensor's false alarm errors (Type I) and missed detection errors (Type II). These targets were uniformly distributed across the search area for each trial and kept static throughout the simulation. The agents searched in a "lawnmower" pattern with spacing determined by the sensor field of view and search altitude to provide maximum coverage of the search area in a minimum number of passes. This put classification accuracy and area coverage rate at odds as both could not be maximized for the same mission parameters. Additionally, each agent had a fuel usage model that limited endurance and would trigger a return to launch (RTL) condition to ensure that the vehicle returned home with a 20 percent fuel reserve.

The scenario was run at four levels of cooperation to study their effects on system performance. First, the single agent case provided a baseline to test the basic autonomy and ASRA's performance. Then, three more levels of cooperation were run to provide a spectrum of cooperation. An additional case was run on the final, moderate cooperation case that specifically tested the WAS system's ability to compensate for an agent falling offline. To test this, one agent was initialized with a low fuel capacity, causing it to RTL early in the mission. The remaining agent would then continue searching the total area, taking over the area left by the other agent. The RTL agent could then return to the mission but could only confirm already found targets.

Tested Cooperation Levels:

1. Single Agent Case - One agent searches and confirms the entire search area.
2. Complementary Agent Case - Two agents split the search area and individually search and confirm their halves.
3. Extreme Cooperation Case - Two agents split the search area, search their halves, and immediately confirm any target the other agent finds.
4. Moderate Cooperation Case - Two agents split the search area, search their halves, and use a utility function to determine the value of confirming any target or continuing search.

The scenario and cooperation levels were designed to tax the system in multiple ways, providing a means to analyze how implementing cooperative agent interactions on an implementation of autonomy can alter mission effectiveness. Many elements could have been added to the scenario such as multiple target types and target priorities, persistent surveillance tasks, more agents, moving targets, or a more advanced simulation environment. Variations such as these are subjects of planned future research.

4 ASRA IMPLEMENTATION

4.1 Software Design

This research implemented ASRA with a three layer architecture for the single agent case initially. This was expanded to the four layer architecture to provide the additional coordinator layer needed for the cooperation cases. The software instantiation of this architecture was written in Python and the communication interface between the discrete software modules forming each layer was provided by Lightweight Communications and Marshalling (LCM), a publish/subscribe messaging model utilizing UDP multicast. The connections and 12 LCM messages sent between modules are shown in Figure 2.

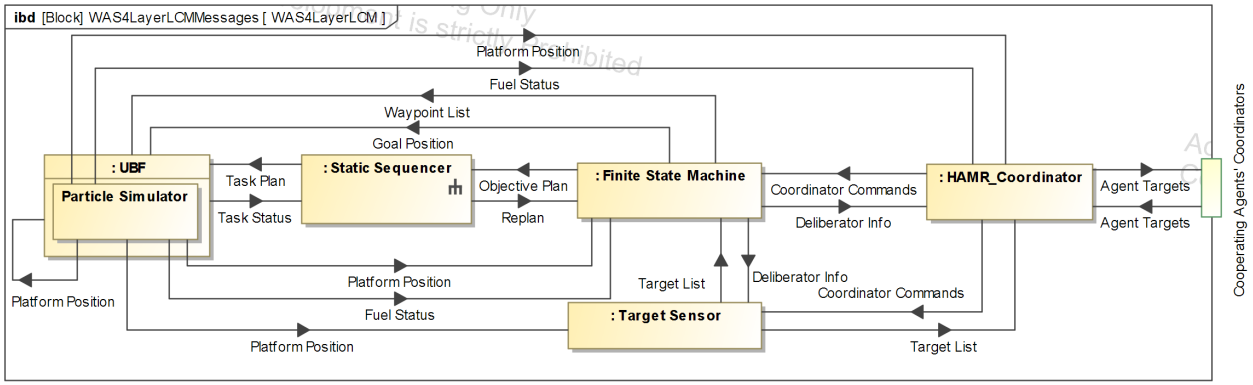

Figure 2: The four layer architecture required 12 LCM message types to communicate between its 5 discrete software modules in each agent as well as between agent coordinators.

4.1.1 Controller

The controller was structured following the UBF and consisted of a behavior hierarchy, an executive, and behavior controller. The behavior hierarchy was manually created in the setup script as a single level of leaf behaviors from the behavior library because composite behaviors were not required to provide the necessary functionality. The implemented leaf behaviors were Takeoff, Land, FlySearchPattern, FlyConfirmPattern, GoToWaypoint, and HoldXYZ.

While not used in the experiment trials, composite behaviors were developed and tested to study the process of building a multi-level behavior hierarchy to achieve more complex agent behaviors. With ASRA and the UBF, building and testing composite behaviors by combining multiple leaf behaviors was a straightforward task. For example, a SearchAvoid composite behavior was created to fly the search pattern while avoiding a position in the middle of the search area. The composite behavior combined these two behaviors with the vector sum arbiter to smoothly route around the avoid location during the search pattern, effectively bending the search pattern passes away from the avoid point. By combining individually tested leaf behaviors, advanced composite behaviors can be built up in just a few minutes in ASRA.

The second piece of the controller layer is the executive which handles the higher level interface with other layers. The executive determines the behavior that should be active based on the task plan it receives from the sequencer. The executive then calls the behavior controller to generate an action by sending the current behavior a stateblock of the necessary sensor inputs. These behaviors take in position data, determine where the agent is and where it needs to go, and decide on the appropriate actuation required to get there. The behavior then outputs an action which includes a weight, an action complete boolean, and an actuator command containing motor controls such as 3D velocity. This action gets passed to the hardware interface layer shown in Figure 1, which handles the transfer to the hardware layer so that behaviors require no change when running in simulation or with physical hardware.

The particle simulator and Ardupilot SITL were interchangeable and interfaced with the behavior controller. The controller's execute action callback determines what software module handles the behaviors' action outputs in the execute action method, so the simulator of choice was set as this execute action callback. When using Ardupilot SITL, the callback is set as an interface that converts the action output's motor commands to control messages understood by SITL. When using the particle simulator, the callback was set as the simulator's update position function. This function takes in the action output's motor commands and steps forward one simulation step which updates the agent position and agent position LCM message.

4.1.2 Sequencer

The sequencer handles the conversion of the deliberator's objective plans to individual tasks sent to the controller. For this research, tasks and behaviors were mapped one to one so objectives were only broken down to tasks, not needing to be broken down further to separate behaviors. The objective plans could include one to many objectives, each with a sequence number and activation priority. From the objectives, the sequencer creates a list of tasks in the order of each objective's sequence number and activation priority.

When a task plan is created, the sequencer commands the controller to start the first task and then monitors its status. When the controller notifies the sequencer that the current behavior has completed the task, the sequencer commands the next task to begin. If it was the final task, the sequencer notifies the deliberator that the current objective plan is complete, so the deliberator can send the next objective plan.

4.1.3 Deliberator

The deliberator provides the high level decision making for the WAS agent and was implemented as a finite state machine. For the single agent case, the main flow of the state machine consisted of Ingress, Search, Confirm, and Egress with transitions triggered by the sequencer notifying the deliberator that the current objective plan is complete. Deviations from this flow were only caused by a return to launch condition due to a low fuel status, or the decision to go confirm an unconfirmed target in the cooperation cases. Additionally, the deliberator performed functions such as search pattern generation, fuel status monitoring, target sensor control, and cooperative utility calculations. The deliberator interfaces with every other software module in order to receive the necessary information to perform these functions.

For the multi-agent cases, the deliberator received information on other agents from the coordinator and considered that in its decision making process. In the moderate cooperation case, a cooperative utility function was run by the deliberator to determine the utility for all agents to confirm any unconfirmed target. All agents ran identical cooperation algorithms based on global agent information; information loss and limited transmission range are subjects of ongoing research. With the assumption that all agents perform identical utility calculations, the agent with the highest utility is free to go confirm a target without requiring acknowledgement from others. By removing this step, agents respond to the environment faster by immediately making decisions instead of waiting for responses from all other agents.

Agents were limited to claim three targets at a time to confirm in order to distribute the work evenly throughout the group. Agents published these to the group and other agents simply excluded these targets from their consideration which did not require an additional deconfliction step. While the utility function was designed to accommodate many agents, it was only tested on two agent missions as shown in Figure 3. This confirm utility function was based on the following equally weighted, individual values and was finally compared to a threshold to ensure the utility was great enough before confirming a target.

- Distance from agent to target: When the distance is shorter, the utility is higher, prioritizing agents closer to the given target.
- Agent's fuel status: Agents with more fuel are assigned a higher utility value because the probability of reaching the target and returning to their search region to continue doing usefull work without running out of fuel is higher.

- Number of targets agent already found: Agents that have already found many targets should shift to confirm those targets. Their utility value increases as they find more targets, encouraging them to stop searching and go confirm.
- Agent's search area completion status: Agents that are finished searching their region are assigned a higher value than agents still searching. This is to encourage agents who finished searching to go confirm targets over agents still searching.
- Number of agents finished search: When many agents have completed searching, the remaining searching agents should be discouraged from confirming. By adding less value to all agents when many agents have completed search, the threshold is effectively raised to keep the remaining searching agents from stopping search to confirm.
- Agent's loiter status: Similar to the search completion status value, agents in the loiter state are assigned a higher confirm value to encourage loitering agents to go confirm a target over an agent already confirming a target.

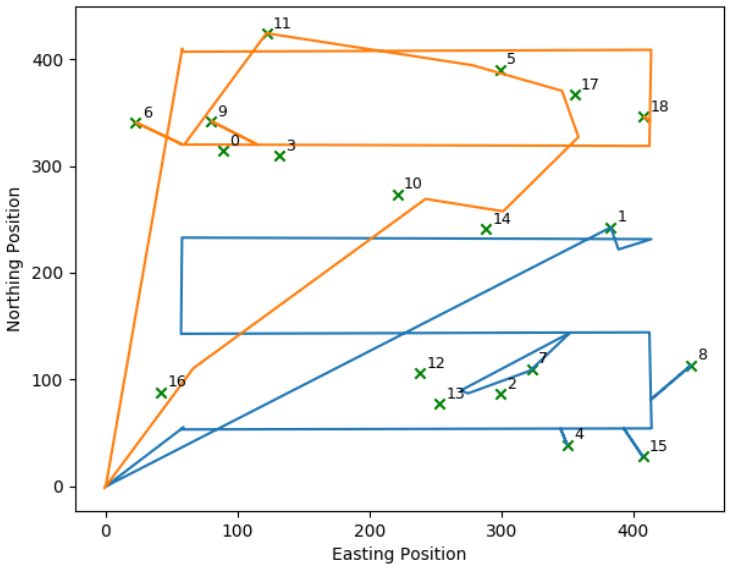

Figure 3: A top view of the moderate cooperation mission with a utility function driving confirm decisions. By tuning utility parameters, immediate confirmations can be balanced with post-search confirmations.

4.1.4 Coordinator

The coordinator exchanged appropriate information between agents and provided the deliberator with information on other agents as well as cooperative utility values for the multi-agent cases. The coordinator was initialized with the number of agents to expect during the mission. It used this information to split the search area upon initialization and then send the new area to the deliberator to generate a search pattern.

During the mission, the coordinator continuously updates and sends its global agent message to other agents which contains information on its current state, target list, current confirming list, and fuel status so other agents can use the information to calculate confirm utilities. It also determines cooperative utilities to send to the deliberator such as the utility of loitering after finishing search to wait to confirm any new targets yet to be found. The global agent message also forms a heartbeat message that the coordinator uses to detect if any agents left the mission. The coordinator shares this information as well as that agent's last search position with the deliberator to take over any unsearched area in the robustness test case. Finally,

the coordinator sends a target list compiled from the other agents, a list of targets currently being confirmed by other agents, and the calculated cooperation information to the deliberator.

4.1.5 Perceptor

The perceptor module, shown in Figure 2, was the simplest of the five, as it only ran a target sensor algorithm upon the deliberator's request and returned a list of targets and their attributes after sensing. The perceptor was initialized with a list of targets, each having a true position and type. When the targets are sensed, the perceptor adds sense results to each target, such as sensed position, sense type, and sense mode, either search or confirm.

Each iteration the perceptor cross checks the agent's current position with the list of true target positions to determine if any targets are in the field of view. When a new target is in view, a number is pulled from a uniform, random distribution between zero and one. If this value is below the diagonal value of the confusion matrix, the target reading is recorded correctly; if not, the sensor incorrectly classified the target and a Type I or II error is assigned to the target. Sensed target location error was also assigned which scaled with distance. The perceptor then sent the updated target list to the coordinator and deliberator.

While truth data being included in the perceptor here is not conducive to live tests, the decision was made to package them together for this simulated perceptor. A perceptor used for live tests would simply swap this simulated perceptor for one that worked off of purely sensed data. This live test perceptor would ultimately hand out the same target list to the other autonomy layers but would use a sensor such as a camera with an object recognition algorithm to determine target positions and types.

4.1.6 Simulator

A particle simulator and the Ardupilot Software in the Loop simulator are both available simulators within ASRA. SITL's higher fidelity model but longer run time had to be weighed against the faster running particle simulator. The particle simulator was the lighter weight option, as it started up and ran quickly but provided a low fidelity model. This simulator simply took in velocity commands and stepped the vehicle in that direction an amount based on the time step and velocity magnitude. This simulator does not include a vehicle dynamics model so it simulates a massless particle that can move in any direction. This provided a reasonable model of a small multi-rotor vehicle when driven by velocity commands because these vehicles can hover and move in any direction. This simulator lacked the effects of environmental factors such as wind or air density, so an analysis had to be performed to determine if the low fidelity impacted results.

The other simulator option was Ardupilot SITL, which simulates a UAS flight controller and vehicle by running flight controller firmware on a computer with simulated sensors. This allows autonomous systems to move from simulation to flight tests with minimal changes, as the interface with Ardupilot SITL and a physical flight controller is identical. Use of SITL comes at a cost of longer startup and simulation times as well as a more complicated interface with the simulator when compared to the particle simulator. Ardupilot SITL is also limited by the fact that it cannot initialize a vehicle in the air, and thus a complete startup sequence on the ground must be performed, just like real flight controllers. Additionally, because the flight controller firmware is designed to run at real time speed, Ardupilot SITL becomes unstable when running simulations faster than around five times real time.

The Ardupilot flight controller firmware uses the MAVLink messaging protocol to communicate with other devices, making it the communication method used by Ardupilot SITL. This means that to communicate with Ardupilot SITL, the action outputs of the behaviors must be converted to MAVLink messages. ASRA provides an interface layer module to perform this conversion. The action outputs are sent over LCM from the autonomy layer and are received by the Ardupilot Interface. This interface then reads the action from the LCM message, which is in the form of velocity commands, and creates and sends MAVLink messages with the same contents. The Ardupilot interface performs the same operation in the other direction,

converting simulator outputs such as position, battery level, and other sensor readings from MAVLink to LCM messages, sending the information to the autonomy layer.

Ardupilot SITL's higher fidelity model had to be tested against the faster executing particle simulator, so identical single agent scenarios were run in both simulators and compared. To make the two runs comparable, they were run with the same target positions and with perfect sensor accuracy to ensure the same targets were revisited in both cases. It was determined that the difference in scenario outputs was negligible. The difference in area covered was within 0.13%, but the main difference was the execution time. The particle simulator completed the simulation in 8 seconds while Ardupilot SITL took 8 minutes. The minimal difference in area covered was acceptable given the particle simulator's immense time savings and the relatively low fidelity required for this research. For these reasons, the particle simulator was chosen for the experiment runs. An example output of a single agent mission is given in Figure 4.

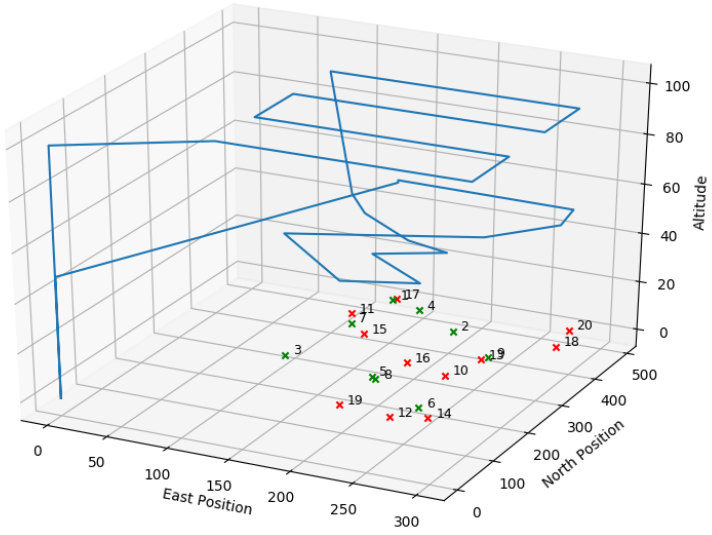

Figure 4: A single agent mission showing the search altitude above the confirm altitude with targets below.

If Ardupilot SITL had been required for the experiment runs, there would still be value in using the particle simulator for development. This ASRA implementation was able to easily switch between both simulators by adjusting a single configuration file parameter. This quick switch conveniently provides the particle simulator's fast execution times for development and Ardupilot SITL's higher fidelity for analysis runs.

4.2 Autonomy Testing

Using the requirements for autonomous systems listed by Brooks (1986) and reactive control systems given in Woolley and Peterson (2009), the following metrics of autonomy were selected to evaluate the system:

- **Responsiveness:** The amount of time the agent requires to respond to external stimuli. This is measured as the amount of time required to act on an objective plan (OP).
- **Robustness:** The degree to which the system can continue the mission using operable vehicles after a vehicle is forced offline due to a failure.
- **Perception Accuracy:** The impact of false perception on actions performed by the agent.

Additionally, the following response variables were used to evaluate the system's mission performance.

- **Percent Area Searched:** The percent area evaluated by the agent out of total assigned area.
- **Mission Time:** The amount of time taken by the agent to finish the mission. This is the actual time to accomplish the mission which is a function of the total number of simulation iterations.
- **Percent Detected**: The percent of targets detected by the agent, both true and false.
- **Percent Correct Detected**: The percent of true targets detected by the agent.
- **Type I Error Detected:** The percent of targets detected as true that were in truth false.
- **Type II Error Detected:** The percent of targets detected as false that were in truth true.
- **Percent Confirmed:** The percent of all targets that were confirmed.
- **Percent Correct Confirmed:** The percent of correct confirmations out of possible confirmations.
- **Type I Error Confirmed:** The percent of targets confirmed as true that were in truth false.
- **Type II Error Confirmed:** The percent of targets confirmed as false that were in truth true.

The factors and their tested levels shown in Table 1 were selected for the DOE to evaluate the system's mission performance. The DOE was conducted with multiple replicates for each cooperation level to estimate experimental error and to detect a lack of fit. Just two replicates were selected to minimize simulation and analysis time and proved appropriate as all response variables passed the multiple comparison statistical test. By running the factorial analysis across all four levels of cooperation and recording the response variables, a statistical model was created to predict the response variables based on Table 1's input parameters.

Table 1: Factors and levels for the face centered cubic design of experiments.

Factor	High	Low
Search Velocity	10 m/s	5 m/s
Detect Real Probability	0.9	0.65
Detect False Probability	0.9	0.65
Search Area Size	490,000 m^2	292,500 m^2
N Real Targets	19	1
N False Targets	19	1

These results provide two major insights on the system. The first allows researchers to predict response variables based on input parameters, the factors given in Table 1. This means the performance of the system can be predicted if the mission parameters are known. The second and perhaps more powerful insight is in finding the optimal configuration to maximize a given response variable. For example, we found that the moderate cooperation case minimized mission time, which was the expected result. Analysis such as these allow for system characterization across various mission environments, allowing users to determine optimum operating conditions for a given mission, based upon expected mission and vehicle parameters.

5 EVALUATION AND USE

5.1 Support for Test

LCM provided more than just messaging during the simulation. The LCM library's LCM Logger recorded all message traffic to a log file. The log files can then be played back to mimic the original traffic, or parsed for analysis. These logs provided all of the information necessary to calculate the metrics given in section 4.2. It was found that the messages and their contents required by the autonomy were sufficient for post analysis but adding additional test messages solely for testing would be a nominal task.

Using LCM logs meant that no changes to the system had to be made to capture all required information. Only an external logging tool had to be run during the simulation. This simplified the transition from development to test. We found that simply capturing all LCM message traffic resulted in large LCM log file sizes which lead to stability issues when recording across multiple unsupervised test runs. To reduce the file size, the simulator could be run at a larger step size, reducing the number of iterations required to

complete the scenario, and thus the number of messages sent. To determine how the particle simulator step size affected fidelity and file size, identical missions were run at step sizes of 0.05, 0.1, and 0.3 seconds. The results shown in Table 2 revealed that increasing the step size from 0.05 to 0.3 results in less than 1% change in area covered and 2.3% change in mission time, but over 80% reduction in file size, making the 0.3 step size the best choice to balance fidelity and file size, leading to reduced analysis times as well.

Table 2: Particle simulator step size test results show an 80% decrease in file size with a minimal impact on performance when increasing step size from 0.05 seconds to 0.3 seconds.

Measure	Step Size		
	0.05 Seconds	0.1 Seconds	0.3 Seconds
Percent Area Covered	62.21%	62.8% $+\Delta 0.11\%$	62.83% $+\Delta 0.99\%$
Mission Minutes	5.62 mins	5.67 mins $+\Delta 0.88\%$	5.75 mins $+\Delta 2.31\%$
File Size (MB)	13.5 MB	6.8 MB $-\Delta 49.62\%$	2.3 MB $-\Delta 82.96\%$

The ASRA test module provides the ability to run automated tests which consist of automatically and efficiently running tests across all selected factor levels, capturing response data for every run, and performing analysis to provide results of system performance across the testing range. The automated testing code read the factor inputs from a table that defined the levels of each factor for all test runs of the face centered cubic factorial experiment design. These factors were then assigned to the agent and the mission was run. During the mission, the LCM message traffic was logged and upon mission completion, the log file was analyzed to calculate the autonomy metrics listed in Section 4.2. These metrics were saved to a results table and the next run was executed.

5.2 Use in Design and Operation

We have discussed how ASRA can provide a rapid development and test environment, but it also has applications once the system is fielded. ASRA's modularity enables components to be modified and upgraded in the field to achieve new mission sets or improve performance. For example, by selecting a different deliberator and perceptor at startup, the agent's mission could completely change from a search and confirm enemy targets mission to a search and rescue mission while using the same behavior library. Furthermore, any components can be upgraded to improve performance throughout the system's lifespan, and the modular design provides for seamless integration without affecting the rest of the system.

The most likely changes to be made to a system such as the UAS studied here, are sensor payloads, to include the use of heterogeneous sensors in cooperative agent scenarios. Various sensors can be modeled in ASRA, tested, and fielded without requiring major changes to the rest of the system, despite the fact that the system was not originally designed for that specific sensor. This enables autonomous systems to be flexible to new environments, missions, sensors, and upgrades.

6 CONCLUSIONS & FUTURE WORK

In this paper, we presented an implementation of the Autonomous Systems Reference Architecture and how it provides a streamlined development environment for autonomous systems from initial development, through simulation and test, and finally to field operation. ASRA provided an environment to develop the WAS mission autonomy, allowing for model and software component reuse to accelerate the development process. The coordinator effectively handled multi-agent interactions, successfully achieving a cooperative autonomous mission. ASRA's hardware interface allowed for the seamless transition from a low fidelity particle simulator to the Ardupilot Software in the Loop flight controller simulator, enabling a comparison to determine the appropriate simulator to use for the hundreds of DOE test runs.

The face centered cubic DOE test, run across the four cooperation levels, drew insights on mission performance. By building up prediction models, future missions can by improved by selecting an optimal

configuration. As expected, a moderate cooperation approach where agent decisions are based on cooperative decision rules that consider the system of system objectives performed optimally compared to the three other cooperation levels that were explored. The next step for this research is to take the system into live flight tests to determine the accuracy of the mission performance predictions derived from simulation. This transition is a straightforward step with ASRA: the hardware layer is swapped from simulated hardware to a physical flight controller that uses the same communication protocol as the Ardupilot SITL simulator.

ASRA's use carries even further than live flight tests, into fielded systems to aid in managing system upgrades and enabling flexible mission applications for a given fielded system. ASRA provides an all in one solution to system design, development, simulation, test, and field modifications for the growing complexity and applications of autonomous systems.

ACKNOWLEDGMENTS

The authors wish to acknowledge Dr. Trevor Bihl from the Air Force Research Laboratory Sensors Directorate for his financial and technical support of this project.

REFERENCES

Brooks, R. A. 1986. "A Robust Layered Control System for a Mobile Robot". *IEEE Journal on Robotics and Automation* 2(1):14–23.

Dunkel, T. B. 2002. "Investigation of Cooperative Behavior in Autonomous Wide Area Search Munitions". Master's thesis, Air Force Institute of Technology.

Gat, E. 1998. "On Three-Layer Architectures". In *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, edited by D. Kortenkamp, R. P. Bonasso, and R. R. Murphy, 195–210: Palo Alto, California: American Association for Artificial Intelligence Press.

Gillen, D. P. 2003. "Cooperative Behavior Schemes for Improving the Effectiveness of Autonomous Wide Area Search Munitions". Master's thesis, Air Force Institute of Technology.

Gray, J., and D. Jacques. 2019. "Reference Architecture for Development of Autonomous Systems". In *Proceedings of the National Defense Industrial Association*. October 21th-23th, Scottsdale, Arizona.

Hooper, D., and G. Peterson. 2009. "HAMR: A Hybrid Multi-Robot Control Architecture". In *Twenty-Second International Florida Artificial Intelligence Research Society Conference*. May 19th-21th, Sanibel Island, Florida.

Jacques, D., and M. Pachter. 2004. "A Theoretical Foundation for Cooperative Search, Classification, and Target Attack". In *Recent Developments in Cooperative Control and Optimization*, edited by S. Butenko, R. Murphey, and P. M. Pardalos, 175–205: Boston, Massachusetts: Springer US.

Murphy, R. R., and R. C. Arkin. 2000. *Introduction to AI Robotics*. 1st ed. Cambridge, Massachusetts: MIT Press.

Park, S. M. 2002. "Analysis for Cooperative Behavior Effectiveness of Autonomous Wide Area Search Munitions". Master's thesis, Air Force Institute of Technology.

Woolley, B., and G. Peterson. 2009. "Unified Behavior Framework for Reactive Robot Control". *IEEE Journal of Intelligent and Robotic Systems* 55(2):155–176.

AUTHOR BIOGRAPHIES

DAVID KING is a recent graduate from the Systems Engineering & Management Department at the Air Force Institute of Technology where he earned a M.S. in Systems Engineering. He is currently an Autonomous Systems Engineer for the Air Force Research Laboratory. His email is david.king.85@us.af.mil.

DAVID JACQUES is a Professor of Systems Engineering at the Air Force Institute of Technology where he received both a M.S. and Ph.D. in Aeronautical Engineering. His research interests are concept definition and system analysis as well as cooperative behavior and control of semi-autonomous flight vehicles. His email is david.jacques@afit.edu.

JEREMY GRAY earned a M.S. in Systems Engineering at the Air Force Institute of Technology where he works is a staff engineer for the Autonomy and Navigation Technology Center. His work focuses on Small Unmanned Aerial Systems, indoor navigation, and cooperative command and control architectures. His email is jeremy.gray.ctr@afit.edu.

KATHERINE CHENEY recently received a M.S. in Systems Engineering from the Air Force Institute of Technology. She is currently a Systems Engineer for the Air Force Research Laboratory. Her email is katherine.cheney.1@us.af.mil.