Special Issue: Dynamic Large-Scale Swarm Systems in Urban Environments: Results from the DARPA OFFSET Program

Field Report

From Warfighting Needs to Robot Actuation: A Complete Rapid Integration Swarming Solution

Eugene M. Taranta II[1], Adam Seiwert[2], Anthony Goeckner[3], Khiem Nguyen[2] and Erin Cherry[4]
[1]Northrop Grumman Corporation, Mission Systems, Orlando, FL 32817
[2]Northrop Grumman Corporation, Mission Systems, Aurora, CO 80017
[3]Northrop Grumman Corporation, Mission Systems, Annapolis, MD 21401
[4]Northrop Grumman Corporation, Mission Systems, McLean, VA 22102

Abstract: Swarm robotics systems have the potential to transform warfighting in urban environments but until now have not seen large-scale field testing. We present the Rapid Integration Swarming Ecosystem (RISE), a platform for future multi-agent research and deployment. RISE enables rapid integration of third-party swarm tactics and behaviors, which was demonstrated using both physical and simulated swarms. Our physical testbed is composed of more than 250 networked heterogeneous agents and has been extensively tested in mock warfare scenarios at five urban combat training ranges. RISE implements live, virtual, constructive simulation capabilities to allow the use of both virtual and physical agents simultaneously, while our "fluid fidelity" simulation enables adaptive scaling between low and high fidelity simulation levels based on dynamic runtime requirements. Both virtual and physical agents are controlled with a unified gesture-based interface that enables a greater than 150:1 agent-to-operator ratio. Through this interface, we enable efficient swarm-based mission execution. RISE translates mission needs to robot actuation with rapid tactic integration, a reliable testbed, and efficient operation.

Keywords: human robot interaction, swarm robotics, tactics

1. Introduction

Coordinated, collaborative robotic swarms have the potential to offer transformational societal impacts. For example, swarms are expected to significantly improve disaster response activities such as search and rescue and damage assessment, in addition to providing a communication network and even delivering supplies. Swarming may also benefit industries like agriculture and farming through autonomous operations that reduce cost, save time, and improve precision (Albani, IJsselmuiden, Haken, & Trianni, 2017). In our work, we focus on the military application of large-scale swarms

Received: 31 March 2022; revised: 15 October 2022; accepted: 1 February 2023; published: 28 March 2023.

Correspondence: Eugene M. Taranta II, Northrop Grumman Corporation, Mission Systems, Orlando, FL 32817,
Email: eugene.taranta@ngc.com

DOI: https://doi.org/10.55417/fr.2023015

through the Defense Advanced Research Projects Agency (DARPA) OFFensive Swarm-Enabled Tactics (OFFSET) program. The employment of a robotic swarm increases a military unit's span of control from 1:1 (Soldier:Platform) to 1:150+. In this way, a swarm may increase mission effectiveness by several orders of magnitude. Swarming is considered a nonlinear dispersed (NLD) operation with a RAND report by S. J. A. Edwards (2005) positing that swarming is the most aggressive form of NLD and that it will play a central role in future military operations when heavy forces are unavailable.

Advances in low-cost robotics hardware, network solutions, sensing algorithms, and artificial intelligence help bring practical large scale robot swarms closer to reality (Chung et al., 2016). These advancements are typically fragmented across unique and isolated problem domains. However, robot swarming is interdisciplinary, spanning several interrelated fields. Autonomy, distributed perception, logistics, mobile *ad hoc* networking, and human-swarm teaming (Chung et al., 2016; S. J. Edwards, 2000) are example domains where advances in one area may have positive compounding effects in another. Consequently, to understand the impact of new capabilities on swarm mission outcomes, one must integrate and evaluate their technologies into a swarming ecosystem. This in itself is problematic as few holistic robotic swarming systems exist for large scale swarm research, development, and deployment, especially those that have been fielded and proven to operate with a sufficient technical readiness. Our technology, the Rapid Integration Swarm Ecosystem (RISE), is one such system that addresses this problem.

More importantly, we address the question: can a multi-robot ecosystem designed for rapid integration and one-to-many control can be fielded, where a single operator commands 150+ autonomous vehicles in tactical maneuvers? Large scale swarms have yet to be deployed outside of simulation, and it remains unclear what software and hardware architecture can support single operator command at this scale. Earlier efforts in this direction have been made by (Chung et al., 2016) and (Clark, Usbeck, Diller, & Schantz, 2021), though they are limited in either scope or scale. For the first time, we demonstrate with RISE that large scale swarm control for offensive tactical operations is possible. Further, in this work we discuss our architecture design choices and the requirements that informed our design, which practitioners can use to guide their own work.

In general, a swarming ecosystem requires a specialized command and control (C2) interface for swarm control. When we designed RISE, it was our goal that our C2 system would enable a single operator to control hundreds of platforms by specifying only high level commands, and enable the operator to build tactics without having to program robot behaviors. To this end, we were inspired by playbook frameworks and decided to leverage the hierarchical swarm framework of Tactics, Primitives, and Algorithms that were established for the DARPA OFFSET program shown in Figure 1. This framework is similar to the work of Giles and Giammarco (2017), and both drew inspiration from swarm experiments at the Naval Postgraduate School.

The main contribution of this work is RISE,[1] an end-to-end robot swarming ecosystem that enables rapid development, research, testing, and deployment of large scale heterogeneous robot swarm technologies. The RISE contribution further comprises as follows.

1. A human-swarm team interface that enables a single operator to command large scale swarms.
2. A platform for swarm tactic development and execution that bridges the operator to robots and enables practitioners unfamiliar with robotics software to author tactics.
3. A lightweight, mobile *ad hoc* network solution that enables reliable communication at scale between hundreds of agents and other devices.

In addition to the RISE software, we customized, configured, and maintained a swarm testbed of over 250 physical, small, unmanned air and ground vehicles. We demonstrated RISE on this swarm testbed at five military ranges over the last four years. The most recent event in November 2021

[1] RISE software is maintained by Northrop Grumman Corporation and the US Government. Contact Erin Cherry (erin.cherry@ngc.com) for more information on procedures for gaining access.

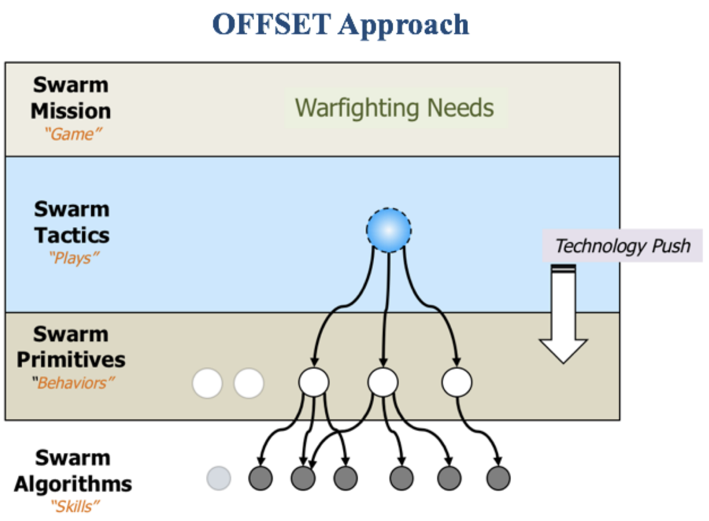


Figure 1. The DARPA OFFSET view on how warfighting needs may be hierarchically decomposed into algorithms that interact with robot hardware. A human operator translates mission objectives into tactical swarm maneuvers that in turn decompose into robot primitives utilizing algorithms. Each layer is present in RISE via a unique software architecture designed to facilitate rapid development and integration. Image adopted from (DARPA, 2017).

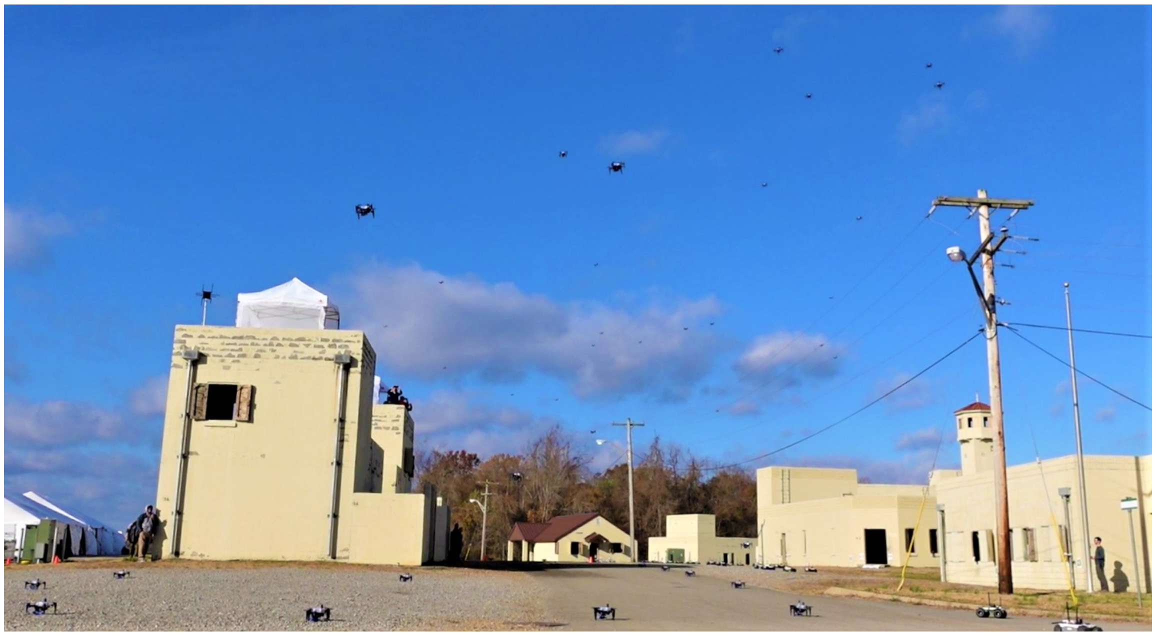


Figure 2. Large-scale unmanned aerial vehicle (UAV) launch for field experiment FX-6 at Ft. Campbell, TN.

(Figure 2) was the culminating OFFSET swarm field experiment, where we demonstrated one user controlling 174 platforms (84 air and 90 ground).

2. Related Work

In this section, we discuss existing work in several fields relevant to RISE. Where possible, we describe the major papers in those fields. Throughout this section, we also refer the curious reader to existing surveys for further information. We also note that swarms consist of many intelligent "agents," individual entities which are capable of autonomous sensing, decision-making, and actuation (Van der Hoek & Wooldridge, 2008). We use the words "agent" or "swarm agent" to refer to the robotic hardware platform together with the software which runs upon it. Further, we sometimes make a distinction between *simulated agents* and *physical agents*, where the former uses simulated hardware and the latter uses real hardware.

2.1. Multi-Robot Systems

In the literature, "swarm robotics" typically refers to homogeneous systems that have a biological inspiration and which communicate using pheromones or other nonswitched networking methods. While RISE is a "swarming ecosystem," it is also more accurately a "multi-robot system" (MRS) or more generally a "multi-agent system" (MAS). This distinction is highlighted by (Rizk, Awad, & Tunstel, 2019), who perform an extensive and high-quality survey of multi-agent and multi-robot systems. Additional surveys of interest include Dorigo, Theraulaz, and Trianni (2021), which focuses on homogeneous swarm robotics, and Schranz, Umlauft, Sende, and Elmenreich (2020), which provides a high-level overview of swarm and multi-robot projects. Rather than reiterate the results of these surveys here, we will only highlight especially relevant works. Note that while we refer to RISE as both a "swarm robotics" and a "multi-robot" systems in this paper, we limit our discussion to that of multi-robot systems, since they are most similar to RISE.

In particular, we first focus on multi-robot systems which function as testbeds for further research. A survey of multi-robot testbeds is given by Jiménez-González, Martinez-de Dios, and Ollero (2013), although several advances have occurred since its publication. Significant early testbeds included the Mobile Emulab (D. Johnson et al., 2006), HoTDeC (Stubbs et al., 2006), and RAVEN (How, Behihke, Frank, Dale, & Vian, 2008). The Mobile Emulab and HoTDeC were characterized by homogeneous robotic platforms and constrained operating environments, only capable of operation in a lab. Both are designed to provide access to remote users who may control the systems' operations via the internet. In both testbeds, robots are extremely simple with limited localization or sensing capabilities of their own, instead relying on centralized camera systems. Importantly, neither study develops abstract swarm concepts such as tactics or primitives, instead providing only a basic computing platform on each agent without a swarm software framework.

Of these early multi-robot testbeds, perhaps the most similar to RISE is RAVEN, a heterogeneous air and ground swarm testbed designed for experimentation with a variety of multi-robot scenarios. While the software architecture is similar to that of RISE, RAVEN is limited to ten vehicles and may only be used in a lab with some external sensing apparatus. Additionally, RAVEN's C2 paradigm is mostly focused on direct control of individual vehicles, whereas RISE focuses on command of the swarm as a whole to maximize the agent-to-operator ratio.

In Pickem, Lee, and Egerstedt (2015) and later in Pickem et al. (2017), the authors showcase a multi-robot testbed created by their GRITSBot platforms. More recently, the "ARGroHBotS" testbed was published by Ospina, Mojica-Nava, Jaimes, and Calderón (2021) and "Crazyswarm" was published by Preiss, Honig, Sukhatme, and Ayanian (2017). Again, these testbeds are limited to lab use and have no or limited command and control functionality.

The Swarmanoid project developed a heterogeneous swarm or multi-robot system consisting of both unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) (Dorigo et al., 2013). As with other projects, the Swarmanoid authors do not clearly define a framework for swarm tactics and overall behavior, and do not clearly state how additional behaviors may be integrated with their system. However, they provide an advanced multi-robot simulator, ARGoS, which remains in use for multi-agent research to this day (Dorigo et al., 2013).

The COMRADE multi-robot system is an advanced multi-robot system intended for clearing mines and explosive devices in conflict zones. COMRADE provides multi-robot search, data fusion, task allocation, and a basic C2 interface, and is designed for real-world use (Dasgupta, Baca, Guruprasad, Muñoz-Melendez, & Jumadinova, 2015). However, the authors do not expand on the concept of tactics or other abstractions for the design of new swarm behaviors; rather, the system is single-purpose. Additionally, the C2 system is extremely rudimentary, providing only the barest of controls to direct individual agents and apparently lacking the ability to manually command the swarm as a whole by issuing new tasks for allocation.

The RUTA heterogeneous swarm (Abukhalil, Patil, Patel, & Sobh, 2016) is developed primarily for research on dynamic task allocation. However, it takes a similar approach to robotic hardware abstraction as RISE, allowing for extreme heterogeneity. Some definition of the different layers of software execution for swarm behaviors is provided, with RUTA's "actions" being similar to RISE's agent-level "primitives" or "algorithms" (Abukhalil et al., 2016). However, the swarm is small and not designed for scale with only five robots. Additionally, the system primarily focuses on task allocation and does not include any concepts related to C2.

More recently, Chamanbaz et al. (2017) developed a hardware and software package to enable simple integration of robotic platforms into a multi-robot system. It consists of computing and networking hardware which can be integrated with existing platforms, along with a software library to enable creation of new behaviors using those platforms. This platform is tested on two separate swarm systems. However, the authors do not differentiate between different layers of the behavior stack (tactics, primitives, and algorithms) and do not consider any human-computer interaction factors. In fact, the authors make no mention of any type of user interface or control.

A significant multi-robot system is presented by Chung et al. (2016), in which fifty UAVs are simultaneously controlled by a team of two. This work may be thought of as a direct predecessor of OFFSET with the same program manager, Dr. Timothy Chung, with similar goals of enabling multi-robot systems to perform complex coordinated actions while under the control of a minimal number of operators. This work is expanded upon by Davis, Chung, Clement, and Day (2018), in which two of the UAV swarms created by Chung et al. (2016) battle for air dominance. RISE expands upon this work with a far greater number of agents, innovative new C2 concepts, a software architecture designed for integration of new tactics and behaviors, the introduction of heterogeneity, and more.

Recently, a number of tools which ease the development of multi-robot systems have been created. One such tool is ROS2swarm, a hardware and package for the Robot Operating System 2 framework (ROS 2) that can be retrofitted to existing platforms. ROS2swarm provides multi-robot programming capabilities using a similar breakdown of swarm software to our tactics, primitives, and algorithms concept (Kaiser et al., 2022). However, ROS2swarm does not enable dynamic transitions between tactics. In other words, the agents may only perform one tactic in a single execution run without a complete reboot of the robot software. ROS2swarm also does not provide any command and control capabilities, unlike RISE. Additionally, ROS2swarm currently uses a centralized network (access point-based) topology, while RISE uses a decentralized topology. While the ROS2swarm is tested with just six agents, we are quite certain that the ROS2swarm system is actually incapable of supporting large numbers of agents (150+), given its use of ROS 2's built-in networking mechanism. RISE replaces ROS 2's default networking mechanism with a custom solution for inter-agent communication. See Section 4.3 for further details on RISE's networking solutions.

An alternate platform, SwarmUS, is provided by Villemure et al. (2022), which includes both a hardware retrofit board and a software package, similar to Kaiser et al. (2022). This provides necessary services for a multi-robot system such as communication, coordination, and localization. An Android-based C2 interface is also provided, but appears to be rudimentary and is not designed for control of a large-scale swarm by a single operator. In fact, SwarmUS was tested on no more than six robots, as opposed to RISE's hundreds (Villemure et al., 2022). Importantly though, SwarmUS appears to focus primarily on hardware design, whereas RISE primarily focuses on software design. Thus the two are difficult to compare further.

RISE is contemporaneous with "CCAST", developed by our counterparts on the DARPA OFFSET program (Clark et al., 2021). As CCAST and RISE solve the same problems and are designed for the same program and experiment scenarios, most of their capabilities are quite similar. However, RISE differs from CCAST in its use a decentralized mesh network for communication, as opposed to CCAST's access point approach. Additionally, RISE makes use of innovative common gesture commands for easy cross-platform use on laptops, touchscreen devices, and virtual and augmented reality, whereas CCAST uses a point-and-click virtual reality interface. CCAST shares the behavior decomposition of tactics, primitives, and algorithms with RISE.

2.2. Robotic Platforms

Since RISE is a complex system of software which may be used with a variety of hardware platforms (see Figure 25), we do not elaborate on prior hardware platforms used for swarm robotics or multi-agent research. Surveys on this area may be found in the literature, such as the one by Schranz et al. (2020) which lists a number of platforms.

2.3. Swarm Networking

Swarm robotics systems in the literature may be broadly divided between those which use biologically inspired "pheromone" systems, directional communications such as in Kornienko, Kornienko, and Levi (2005), and nondirectional radio-based networks. We will focus here on undirected digital radio-based networks, since we believe that the former methods are so different from our approach as to be nearly incomparable.

Three types of radio network architecture are commonly used: central access points, local wireless broadcasts, and mobile *ad hoc* networks (MANETs). In traditional wireless networks, agents communicate only with fixed infrastructure (known as an access point). Direct agent-to-agent communication is impossible. Instead, agents must first send data to the access point, which then retransmits the data to the destination agent. By contrast, a MANET allows both traditional agent-to-infrastructure communication and direct agent-to-agent communication (Hoebeke, Moerman, Dhoedt, & Demeester, 2004).

A central access point approach is used by the CCAST system (Clark et al., 2021). The authors of that study found that agent connection to the access point became a major concern because agents were unable to communicate directly with their neighbors, thus increasing collision risk. For two neighboring agents to communicate, each needed an active connection to the access point, something not always possible in urban environments. This approach is also found in Stubbs et al. (2006) and numerous other implementations.

Local wireless broadcast architectures are more common. While they often do not require a central access point, these architectures typically do not support relaying of data, distinguishing them from true "MANETs" by the definition in Hoebeke et al. (2004). In Konolige et al. (2006), an agent-to-agent local wireless broadcast is used. A similar system is employed by Rubenstein, Cornejo, and Nagpal (2014), where data is not relayed to agents beyond reception range of the transmitting agent, and only "local" communication with nearby neighbors is possible. Similar methods are implemented in Caprari and Siegwart (2005); Dorigo et al. (2013).

Cianci, Raemy, Pugh, and Martinoli (2007) implement a true MANET. Their approach makes use of the ZigBee mesh protocol. ZigBee is also used to implement a MANET by Fernandes, Couceiro, Portugal, Machado Santos, and Rocha (2015), Jevtić, Gazi, Andina, and Jamshidi (2010), and Zahugi, Shabani, and Prasad (2012), among others. By far, the ZigBee system is the most popular MANET implementation in the literature. A MANET solution by Rajant Corporation is also used with success in an unmanned aerial vehicle swarm application by Engebråten, Nummedal, Gilbreath, Yakimenko, and Glette (2019), and in a mine-exploration swarm in I. D. Miller et al. (2020).

2.4. Simulation

As with any robotic development, simulation is a critical tool, but especially so with multi-agent and swarm systems. Simulations dealing with multiple agents vary widely in implementation, but even more so in their ability to translate from simulation to actual hardware. A large majority of swarm simulations typically consist of custom-built simulation environments to do evaluation on particular algorithms (Hamer & Ortega-Sanchez, 2010). While this is appropriate for those particular evaluations, a setup of this nature would not suffice for a rapid integration swarming ecosystem such as RISE. Given this, we focused more on simulation environments that emphasize one-to-one translation to hardware, such as Gazebo (Aguero et al., 2015). We leveraged lessons learned and best practices of simulations of this nature and then focused on making the necessary adjustments to maintain the real world translation while still reaching the necessary number of simulated agents (250+ per the OFFSET program). In Section 9, we go into more detail of this approach and additional acknowledgments of other relevant simulations.

2.5. Human-Swarm Interface

Human-swarm interaction specifically and human-robot interaction generally is an active area of research that includes interface design, control, communications, autonomy, and human factors such as situation awareness and cognition load, among others (Drew, 2021; Chen & Barnes, 2021; Kolling, Walker, Chakraborty, Sycara, & Lewis, 2015; Hocraffer & Nam, 2017). However, in this section, we focus on work that enables a one-to-many operator control over an offensive swarm, beginning with command complexity and ending with user interface customization.

Command complexity draws parallels with computational complexity in that total effort is bound by the number of decisions and actions an operator must take to complete a task (Lewis, Wang, & Scerri, 2006). When robots work independently, effort grows linearly with the number of robots n an operator must command, $\mathcal{O}(n)$. Conversely, tasks requiring careful coordination between robotics may result in superlinear complexity $\mathcal{O}(> n)$. Lewis provides by way of example the coordination of two unmanned vehicles pushing on the corners of a box, where the operator oscillates between each robot to move and straighten the box.

Although command complexity enables practitioners to compare the bounded efficiency of different command strategies, fan-out is another useful measure proposed by Olsen and Goodrich (2003) that one uses to estimate how many homogeneous robots an operator can effectively operate at one time. Fan-out is defined mathematically as $(NF + IT)/(IT)$, where neglect time NT is the expected duration of time a robot can be ignored without degrading beyond a minimum performance threshold, and interaction time IT is the duration of time required to interact with a robot (Crandall, Goodrich, Olsen, & Nielsen, 2005). Intuitively, the more a robot can do without requiring operator attention and the faster an operator can set up commands, the more total robots an operator can command without falling below a required level of performance. Prior work has shown an operator can command between eight and twelve robots simultaneously (H. Wang, Lewis, Velagapudi, Scerri, & Sycara, 2009), depending on a variety of factors. This implies we must move away from direct robot control toward a higher level of abstraction in order to support large scale offensive swarming.

One solution is to increase swarm autonomy (Mi & Yang, 2013). Command complexities moving toward order $\mathcal{O}(1)$ can be achieved in part through the use of planners (Lewis, 2013). Consider that without path planners and obstacle avoidance, it has been found that operators are unable to supervise more than a few UAVs (Cummings, Nehme, Crandall, & Mitchell, 2007). Delegating certain tasks to an autonomy can further improve command complexity and fan-out. For example, McLurkin et al. (2006) designed a swarm interface inspired by the poplar real-time strategy games WarCraft and StarCraft. These games enable a player to control individual units, groups of units, or an entire army with simple high level commands such as mine, build, and attack. By distributing a single command to multiple robots, amortized interaction time can be significantly reduced and fan-out improved. Similarly, command complexity becomes sublinear when the operator can command multiple robots with only a single command when he or she is able to rely on the underlying automation. Another example called Playbook (C. Miller, Pelican, & Goldman, 2000; Calhoun, Ruff, Behymer, & Frost, 2018) provides operators with a library of play templates that the operator can modify and which automation adapts to the situation. Playbook is analogous to a sports team's playbook, where a leader selects a play that the team executes, and several studies have shown the benefits of templated plays.

We adopt a similar approach to these delegation strategies. In RISE, an operator controls the swarm by using mission level tactic commands that appropriate resources and coordinate the actions of multiple robots. An operator may combine tactics to form a course of action (COA) and modify the plan in real time as the situation changes. During mission planning with human warfighters, commanders communicate tactics through COA diagrams that include tactical control measures to establish responsibility and constrain operations that prevent units from interfering with each other (United States Army, 2015). This concept is easily extended to swarm command using a sketch and gesture based interface to simulate the natural use of pen and paper. Hammond et al. (2010), for example, developed a system for free-hand COA diagram sketch recognition with support for military symbology, achieving high accuracy on 485 symbols.

One issue is that COA diagrams are drawn with standardized military symbology, such as those defined by MIL-STD 2525D (United States Army, 2020). However, RISE swarm tactics and control measures are constantly evolving and for some time will remain unstandardized. For this reason, practitioners who develop new tactics and control measures require rapid interface integration for development, test, and deployment. This can be achieved through gesture customization using rapid prototyping gesture recognizers (Taranta et al., 2017). In this way, we are able to expand our tactics library without requiring direct interface developer support or cluttering the interface, all while supporting a one-to-many low command complexity relationship.

3. Background

3.1. On the Decomposition of Warfighting Needs into Robot Actuation

We illustrate the hierarchical relationship between warfighting needs and robotics algorithms as envisioned by DARPA's OFFSET program in Figure 1. This decomposition leads to a multilevel classification system that maps warfighting into operator, swarm, robot, and hardware component objectives. Each level corresponds to a unique perspective, world view, and development approach. Although we did not intentionally design RISE's architecture to reflect this hierarchy, it nevertheless organically evolved into a distributed system comprising four components that directly correlate to the hierarchy: a user interface for translating warfighting needs into tactics, a system for tactics development, a platform composition of primitives, and a framework for algorithm design and integration. In further detail, top-down.

- **Warfighting Needs**. Based on mission objectives, warfighters generate mission plans that evolve through time and drive swarm command. One must therefore translate mission plans into swarm tactics that satisfy mission objectives. However, one must maintain situation awareness (SA) in order to adapt the mission plan to new information. This requires that the swarm reports relevant information to the command and control (C2) team. Our C2 interface for mission planning, tactics execution, and SA is described in Section 5.
- **Tactics**. Tactics encompass the ordered arrangement and maneuver of forces on or near the battlefield (LeFavor, 2020). Whereas primitives are robot-centric, tactics herein are swarm-centric, being software that organizes and employs agents to achieve mission level objectives. An "overhead scan" is an example tactic that partitions an aerial space into regions, each of which requires reconnaissance. Overhead scan may then utilize multiple agents by issuing multiple move-to primitives to complete its objective. Like primitives, one may construct tactics hierarchically. A "breach" tactic may incorporate tactics for persistent surveillance and building entry, followed by exploration and securement. In RISE, we enable tactics development through a PYTHON-based software application called PyC2 (see Section 6).
- **Primitives**. Primitives are software components that utilize algorithms to achieve robot-centric objectives. One example primitive is the "move-to" behavior, where an agent travels to a specific battlefield location while avoiding threats. Move-to employs algorithms for localization, path

planning, and obstacle avoidance. In parallel, a second continuously running primitive enables the agent to maintain situation awareness via sensor input analysis and communication with other agents. Although move-to works to avoid threats, a third primitive may recognize an immediate inescapable danger, preempt move-to, and engage the threat. Primitives are therefore compositional and hierarchical. Our interface for primitive development and integration is described in Section 7.

- **Algorithms**. Algorithms are the software components that interface with hardware by analyzing sensor input and driving actuator output. Example algorithms include computer vision techniques like YOLO (Redmon, Divvala, Girshick, & Farhadi, 2016) that recognize objects embedded in video frame data, localization techniques that perform dead-reckoning from continuous inertial measurement unit (IMU) data (Brossard, Barrau, & Bonnabel, 2020), and actuation of motor control systems from velocity commands. Single algorithms enable platform capabilities; they are also known as skills. Our interface for algorithm development and integration is described in Section 8.

3.2. The Human-Swarm Team

Over the course of five large-scale field experiments and several intermediate integration tests, we self organized our human team into five disparate roles that we refer to as the swarm commander, swarm operator, swarm health engineer, field operations officer, and field support personnel. Each role not only accounts for one aspect of running a successful mission, but also correlates with the amount of responsibility we found one person could handle without overloading the individual. We envision that these roles will further evolve as robotics technologies, swarming, and RISE continue to mature, and that they can be adopted to a variety of organizational structures. The roles in detail are as follows.

- **Swarm Commander**. A commander leads the C2 team by defining mission objectives, converting those objectives into a mission plan, modifying the plan as the mission progresses, and coordinating with C2 team members. A commander interacts with C2 software to maintain situation awareness but does not directly command the swarm. In this way, the swarm commander's interaction with C2 software correlates with how a platoon or company leader would use the tool.
- **Swarm Operator**. An operator interfaces with the swarm by converting the mission plan into swarm tactics using C2 software. An operator is also responsible for maintaining situation awareness, reporting intelligence information to the commander, and making tactical recommendations based on evolving circumstances and swarm capabilities. The operator may also request support from the swarm health engineer to command individual agents or investigate error conditions. While there are currently no military occupational specialties (MOS) for swarm operators, we expect this role will map to several MOS categories, including the 15W, which is a single UAV operator.
- **Swarm Health Engineer**. A health engineer monitors the communications network as well as individual agents' health. An engineer is a technician who is able to diagnose and remotely resolve robotics issues. In an operational setting, this role could fit into a variety of engineering MOS specialties.
- **Field Operations Officer**. The field operations officer is responsible for logistics involving agent deployment, recovery, physical maintenance, field personnel coordination, and operational safety. An operations officer coordinates with the commander and may lead a small ground team of field support personnel. The field operations officer role is an artifact of the logistical construct of the field experiment events, the swarm hardware utilized (e.g., limited battery life), and safety requirements for system testing at scale.
- **Field Support**. Field support personnel report to the field operations officer and provide the logistical power needed to deploy and maintain the swarm. Given the known limitation

of hardware and the need for sustained operations over extended periods of time, this team provides the means to recover vehicles and prepare them for redeployment. We expect that as the technology continues to mature, the need for this role will subside.

In line with DARPA OFFSET program objectives, we employed a heterogeneous swarm of low-cost ground and aerial vehicles. For aerial operations, we leveraged quadcopter platforms and vertical take-off and landing (VTOL) fixed wing aircraft. Each platform type offers unique operational benefits, but also comes with its own limitations. Further details on the platforms we leverage and their tradeoffs are presented in Section 11.2.2.

3.3. Live, Virtual, and Constructive

Time, logistics, and cost constraints prohibit frequent testing of single agent and large-scale swarm solutions. To support rapid development and integration, it is therefore critical that engineers are able to iterate their software designs without having access to physical hardware. Simulation facilitates this goal, but only when the simulation and deployment environments are uniform. Otherwise, engineers lose time supporting multiple related but incompatible interfaces. In other words, software developed in simulation should transition directly to the real world without concern over what is real or virtual. For this reason, we take care to ensure our software is abstracted from an underlying reality, such that user interface features, tactics, and primitives are unaware of the underlying platform's true nature. We further discuss our Live, Virtual, and Constructive (LVC) approach in Section 9.

4. The Rapid Integration Swarming Ecosystem (RISE)

4.1. Requirements

The design of our swarming architecture is informed by program requirements as well as our experiences as an OFFSET systems integrator. We present some high-level system requirements below:

Req. 1 *Capable of one-to-many (150+) operator-to-agent ratio for control of swarm agents.*

In reviewing prior systems, we found that operators were unable to effectively command more than a handful of individual platforms (see Section 2.5). Thus one of our primary goals for RISE is to enable the operation of hundreds of vehicles simultaneously by a single operator. This necessitates the concept of "swarm operation" as opposed to individual agent operation, whereby the operator directs the swarm as a single unit rather than providing commands to individual agents.

Req. 2 *Capable of supporting 250 networked agents.*

As part of the DARPA OFFSET program, RISE is required to support at least 250 networked agents in simultaneous operation, although this number was not reached in any official field experiments (see Section 11) due to logistical (not technical) limitations. This requirement necessitated the creation of advanced networking protocols and techniques (see Section 4.3).

Req. 3 *Capable of operation in a real-world, uncertain environment.*

The RISE swarm must be capable of operating in real-world, nonlab environments. To enable intelligent decisions and effective C2 in the field, this requires agent-level sensing and autonomy along with robust networking and communication.

Req. 4 *Tactics development abstracted from robotics software.*

We found that many involved in swarm-based research are not roboticists, especially those focused on tactics development. These individuals prefer to view robots as entities capable of autonomous navigation through the environment with which they have only occasional interactions. Their concerns primarily lie in the organization and coordination of independent agents that serve an operational objective, such as to intelligently acquire and maintain situation awareness. In other words, tactical engineers typically take an exocentric world view, solving problems at a level of abstraction above the robot. An architecture serving tactical engineering needs will then separate tactics from behaviors, enabling the engineer to focus on converting a mission level objective into a set of robot commands. Therefore RISE must provide such an abstraction to allow these engineers to easily develop new tactics.

Req. 5 *Robotic development for primitives and algorithms to support tactic creation.*

While tactic development was abstracted in such a way that robotics expertise is not needed, there is still the need for capable robotic platforms. Most off-the-shelf platforms do not come setup with the capabilities required to perform the tactics that were created within RISE. The tactics have a dependency upon the robots being capable of autonomous navigation through the environment and environmental interactions. These dependencies drive the development described in Section 7 and Section 8.

Req. 6 *Flexible swarm command interface for tactical coordination.*

Tactics vary in complexity and oversight. While a tactical engineer may not be concerned with the underlying task management system, support for allocating robot cohorts, initiating direct and indirect command, synchronizing command execution, canceling outstanding tasks, and receiving command status is required.

Req. 7 *Interface for exposing new tactics, tactic parameters, and tactical control measures.*

To facilitate rapid development and because tactical engineers often do not use end-user interface software, an engineer must be able to easily define a new tactic description, means of invocation, associated parameters, and required tactical control measures, which are then immediately available to the operator.

Req. 8 *Support for virtual hardware in real experimentation.*

While the RISE software is hardware-agnostic, research programs such as DARPA OFFSET often have funding limitations and use off-the-shelf platforms (see Figure 25) with limited capabilities. Thus not all desired experiment scenarios may be feasible with the physical hardware. For this reason, RISE is required to provide support for simultaneous operation of simulated and physical agents.

Req. 9 *Low fidelity simulation for rapid development.*

While high-fidelity simulations are useful for development of robotic behaviors and for testing, we found that tactic developers often did not have the computing resources necessary for a full simulation. Therefore one of RISE's earliest requirements is to enable low-fidelity simulation that can be run on a basic laptop computer. This allows rapid iteration on tactics development by our third-party collaborators (see Section 10).

Req. 10 *Simulation for robotic development and evaluation.*

While the low fidelity simulation is primarily utilized for tactic development, there is still a need for a simulation that supports primitive and algorithm development. Without a simulation of this nature, robotic development is tedious and always requires on platform testing. Therefore the simulation must support both mock sensor feeds and actual hardware endpoints.

4.2. The RISE Architecture

The RISE architecture presented in Figure 3 conforms with the decomposition of warfighting needs into robot actuation, as discussed in Figure 1. One of the major components is called Swarm Engine™

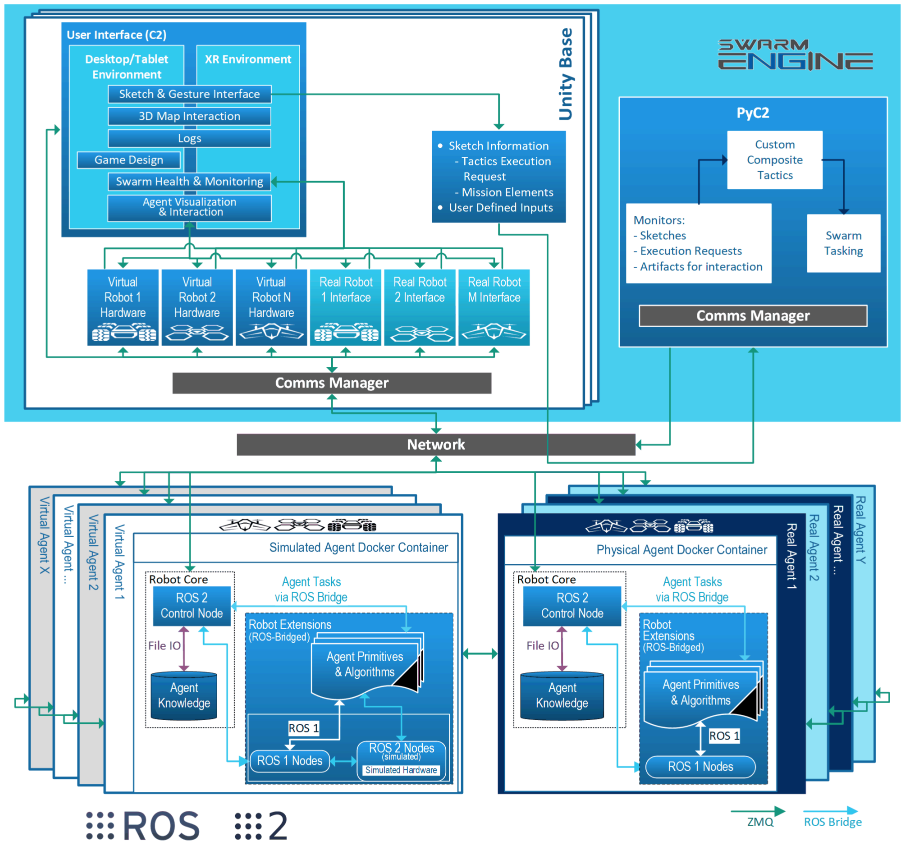


Figure 3. The RISE architecture. Consisting of four main components split between agent side computation and command and control computation.

and houses the command and control solution, which includes C2 and PyC2. The former (C2; referred to as "Unity Base" in Figure 3) implements our user interface software for environment visualization, situation awareness, and sketch-based swarm command. We implemented C2 using the Unity game engine[2] because of its cross-platform compatibility; ability to support advanced desktop, VR, and AR interactions; and built-in simulation support, in addition to having a large active community. PyC2 is a PYTHON-based software application designed for rapid tactics development. It comprises a number of tools designed to assist developers with writing tactics that interact with sketch input, query the environment, and generate robot primitive commands. PyC2 implements a task allocation system, and provides low fidelity simulation support so that engineers can iterate tactic designs without having to run robot software, as specified by Requirement 4. We chose to build PyC2 in PYTHON because of the language's ease of use, extensive library, and wide use, thus fulfilling Requirement 7.

Agent software is divided into "Robot Core" and "Robot Extensions" modules, both encapsulated by a Docker container. Together, these components fulfill Requirement 5. Robot Core is a platform-agnostic compilation of ROS/ROS 2 nodes that handle key functions for task bidding, task execution, health monitoring, swarm communication, and situational reasoning and reporting. All external communication from agent to agent or agent to swarm is also facilitated through the Robot Core system. This creates a layer of abstraction that enables any agent running the Robot Core software stack to become part of the swarm. Robot Core interacts with agents via their given

[2] https://unity.com

Robot Extensions codebase. The Robot Extensions codebase contains all the ROS nodes responsible for sensor or hardware interaction, as well as onboard agent algorithms (see Section 8). Interaction between Robot Core and Robot Extensions is facilitated by standard ROS messaging, using a bidirectional ROS 1 to ROS 2 bridge[3] when appropriate.

Note that in the architecture diagram (Figure 3), there are two variants of the agent software listed—one representing the software running on a physical agent and the other representing a simulated agent. The only differences between the two are the Docker images used and the ROS 1 sensor data sources. The simulation environment, which is the same Swarm Engine environment used for commanding real platforms, generates sensor information in place of real sensors (see Section 9.1). This distinct abstraction of sensor feeds works to facilitate simulation to hardware translation and fulfill Requirement 10. All aspects of the agent software are contained within Docker containers for easy deployment and simulation testing. We also make use of Ansible[4] for fleet management and deployment using this Docker based system (see Section 11.2.2).

4.2.1. ROS

ROS[5] is a key component of the RISE architecture. Everything was developed with ROS in mind for ease of integration and development. The agent software consists of a mixture of ROS and ROS 2 nodes. Due to a lack of hardware driver support in ROS 2 at development time, the Robot Core modules use ROS 2 and the Robot Extensions modules currently use ROS, although Robot Extensions may be upgraded to ROS 2 at a later date when all required drivers are available in ROS 2. All inter- and intra-agent messages are transmitted using either native ROS 1/2 messages using standard ROS networking middleware or ROS 2 messages using a custom ZeroMQ middleware (see Section 4.3). All communication between components of Swarm Engine and varying instances of the Unity environment uses ROS messaging. The simulation environment broadcasts ROS sensor messages to integrate with the agent software for development and testing. ROS messaging is how all algorithms and primitives communicate, and is used at every level of the system, from tactic tasking down to agent-level actuation. ROS's MAVLink/MAVROS package handles all sensor and actuator interaction on each agent. During the course of development for RISE, several releases of ROS and ROS 2 were utilized, but the program currently uses ROS Melodic and ROS 2 Eloquent.

4.3. Swarm Networking

This section describes the network used to enable communication amongst agents and between agents and the operators. We start by providing a summary of early prototypes and challenges and then describe the final topology and protocols.

4.3.1. Initial Prototypes

In the early stages of RISE development, extremely simple off-the-shelf mechanisms were used for swarm networking. Due to availability of components, RISE initially used standard 802.11 Wi-Fi with all agents connected to centralized access points. However, field experiments (see Section 11) quickly exposed the downsides of this approach. Operators needed to pre-position the communication infrastructure before mission execution could begin. This was a difficult process requiring careful site surveys to ensure maximum coverage and establishment of a power supply and network backhaul link at the chosen access point location. Obviously, such preparatory work was deemed impractical for real-world tactical exercises. Additionally, even with site surveys and careful access point placement, the dense concrete urban environment used for field experiments (see Section 11) yielded large "dead zones" in which agents were unable to communicate with the pre-installed access points.

[3] https://index.ros.org/p/ros1_bridge/
[4] https://www.ansible.com/
[5] https://ros.org/

To alleviate these issues, and with the addition of further funding, we conducted an evaluation of commercial off-the-shelf networking solutions and purchased equipment for a mobile *ad hoc* network (MANET) from Rajant Corporation. See Section 4.3.2 for more information on the benefits of this approach.

However, after switching to a MANET architecture, we encountered further problems. While initially we had been using the default ROS 2 messaging mechanism for interagent communications, we found that this did not scale well to more than a few tens of agents when using a MANET. ROS 2 messaging is based on the Data Distribution Service (DDS). Thanks to the MANET's shared medium (2.4 GHz and 5 GHz radio frequency bands), large numbers of radios attempting to transmit simultaneously caused significant data loss and retransmission, rendering the network practically unusable. We worked extensively with our DDS vendor's engineers to resolve this problem, but were still unable to operate hundreds of agents simultaneously. At that point, we began implementation of our own protocol solution for inter-agent communications. See Section 4.3.4 for more information.

4.3.2. Mobile **Ad Hoc** *Network*

One of the critical challenges in fielding a multi-agent robotic system is in the design of a robust communication network. While several architectures were initially considered, we employed a mobile *ad hoc* network (MANET), commonly referred to as a mesh network, in order to facilitate all communication between agents and all nonemergency communication with C2 operators.

We selected the MANET architecture for a variety of reasons, which we list below.

- Urban environments typically comprise structures of varying materials and geometric complexity that obstruct radio communications, preventing access-point-based networks from effectively communicating with agents inside of buildings or agents in-between structures. We found this to be especially true in the environments used for our field testing (see Section 11). A MANET alleviates this problem by allowing other agents in the vicinity to act as network relays, a capability that RISE's tactics exploit. For example, see the building clearing capability developed by SoarTech in Section 10.3.
- By using a MANET, we make optional the prepositioning of network equipment inside the operational area, a task that is required when using networks designed around central access-points.
- A MANET architecture eliminates any single-points-of-failure in the system, allowing agents to continue operation even if one or more of the ground radios fail. The agents may communicate directly with any other devices in the network without routing messages through any centralized infrastructure.
- The MANET architecture allows for easy swarm deployment, a critical capability when operating large-scale swarm systems. Agents do not need to be reconfigured for different deployment strategies, and the ground infrastructure can be changed on-the-fly without any network reconfiguration.

For these reasons, we chose a commercial off-the-shelf MANET solution from Rajant Corporation, utilizing a number of different radios from their product line in our system. Air vehicles are equipped with Rajant DX2 or DX4-series radios, while ground vehicles are equipped with the ES1. The ME4 and Peregrine-series radios are used for ground/base communications, connecting the operators' Ethernet network to the MANET. The mounting of these radios are shown in Figure 4. All radios operate on the 2.4 GHz band. Except for the DX2s, all radios also utilize a second channel on the 5 GHz band. Rajant software manages the use of these bands to allow for efficient transmission with many radios in close vicinity.

4.3.3. Network Overview

While the MANET described above is critical for agent communication, it cannot handle the high volumes of data necessary for LVC sensor simulation (see Section 9). For this reason, we add a

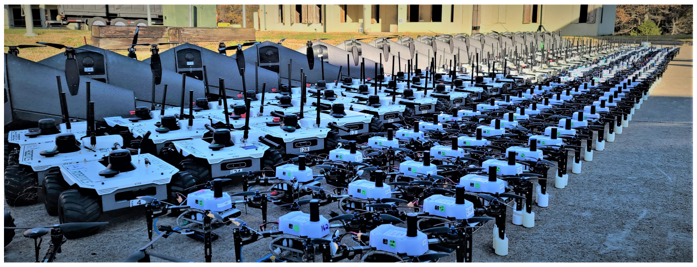
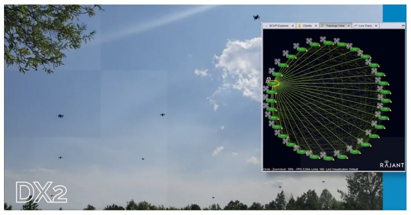



Figure 4. (Left) Rajant ES1 radios are seen mounted to ground vehicles. Aerial vehicles have Rajant DX2 radios mounted underneath. (Right) Mesh radios are shown communicating between IFO platforms during a small-scale field test, as highlighted by Rajant's network management software.

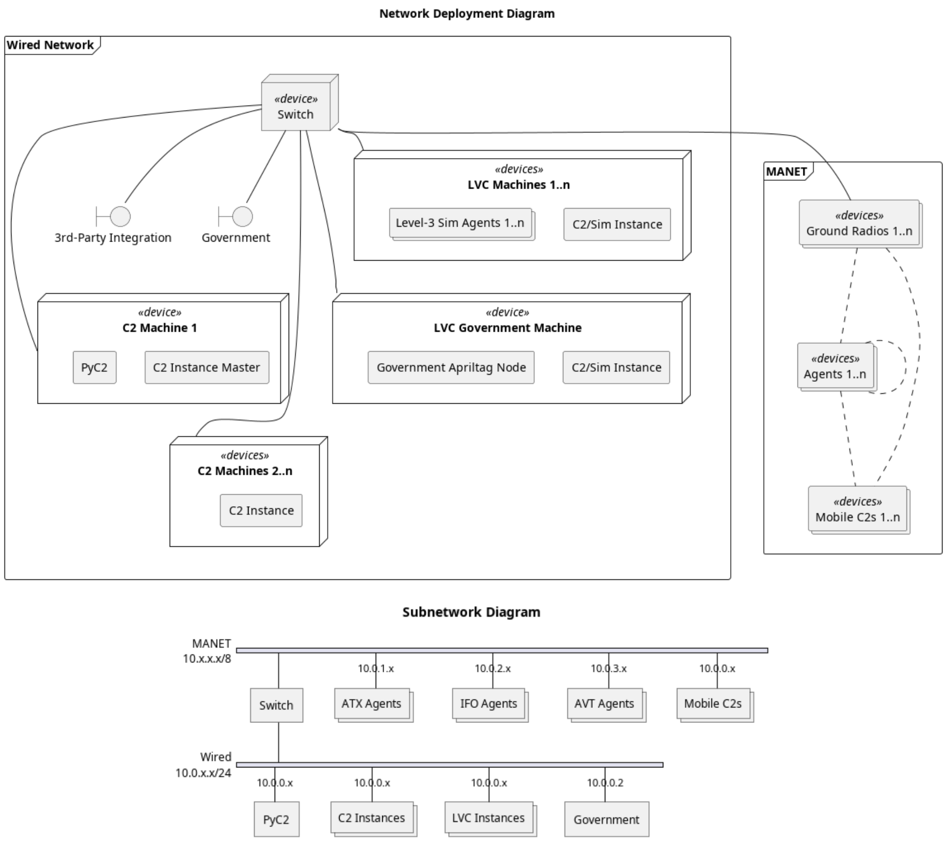


Figure 5. Above, a deployment diagram of the overall RISE network. Items suffixed by "1,..., n" may be of arbitrary quantity. Below, a subnet diagram of the RISE network. In both diagrams, "Government" refers to the government experimentation infrastructure (see Section 11).

1000BASE-T Ethernet network, connected to the MANET via a switch. This Ethernet network serves multiple LVC simulation machines, plus various C2 computers, and facilitates connections with 3rd-party ground systems (see Section 10). The overall network is shown in Figure 5.

4.3.4. Network and Transport Protocols

Initially, we attempted to use ROS 2's built-in network stack, which is based on the Data Distribution Service (DDS). However, we quickly found that DDS was not suitable for our large-scale MANET due to DDS's discovery phase overhead (Object Management Group, 2015). We instead implemented a custom solution using ZeroMQ (ZeroMQ, 2021) as a socket library to allow for reliable multicast communication between all networked systems.

We use the Internet Protocol (IP) (*Internet Protocol*, 1981) as the basis of our swarm network. Every agent has an IP address and is joined to one or more IP multicast groups using the Internet Group Management Protocol (IGMP). In multicast, a device must only transmit one message, which is addressed to a group and will be received by all members of the group (Deering, 1989). This is advantageous in our architecture, since it reduces the amount of air time that a vehicle's radio must use for message transmission. Transmitting a unicast message once for every agent that must receive it (often 200+) requires a significantly larger amount of airtime than transmitting a single multicast message. In a swarm with potentially hundreds of radios in proximity, this is a critical consideration. We take advantage of Rajant's "Tactical Multicast" feature to prevent multicast messages from "echoing" around the network. This feature allows multicast messages to reach all vehicles with minimal retransmission throughout the MANET (Acker, Hellhake, Jordan, & Parks, 2016).

We divide our messages into two transport classes: reliable and unreliable. Both use the IP network protocol, but have different transport protocols. For unreliable messages, standard User Datagram Protocol (UDP) datagrams are employed to transport messages. This protocol ensures message correctness, but does not guarantee delivery or reception order (*User Datagram Protocol*, 1980). Such a protocol is ideal in a dense transmitter environment, since it minimizes transmissions and does not waste air time by attempting to retransmit unimportant data.

For messages that require delivery guarantees, we use the Pragmatic General Multicast (PGM) protocol. PGM uses negative acknowledgments to enable retransmission of lost or corrupted messages. That is, each device maintains a sequence number, which is periodically transmitted in a heartbeat message (SPM packet) to other devices. This allows receiving devices to recognize message loss and request retransmission of those messages using a negative acknowledgment (NAK) (Speakman et al., 2001). We changed several PGM parameters from their default values to allow for more efficient use of our limited network resources. Information on these parameter changes is included in Appendix B.

4.3.5. Application Protocols

As all agents in the system use ROS 2 for their high-level logic and control, integration of the network stack with ROS 2 was a crucial requirement. We created a "Communication Handler" ROS 2 node and package for this purpose. The node accepts a user-specified list of ROS 2 topics (see Section 4.2.1) to be transmitted to or received from the network.

The application protocol was designed to be straightforward and easy to implement. We first encode the topic name as a 32-bit value using the DJB2 hash algorithm (Bernstein, 1991). Then, we use the MessagePack library to encode each field of the message using MessagePack's efficient encoding scheme (Furuhashi, 2021). Each field of the ROS 2 message is automatically decoded and then encoded as a MessagePack field, and the whole message is packed into a structure described by Figure 6. The message is then transmitted using the ZeroMQ library over one of the transport mechanisms described above.

To receive a message, the reverse occurs. The message is received by ZeroMQ, the topic hash compared against a dictionary of known topics (and the message dropped if unknown), and then each field is decoded by MessagePack and re-encoded as a ROS 2 message. The ROS 2 message is then published using the standard ROS 2 publishing mechanism.

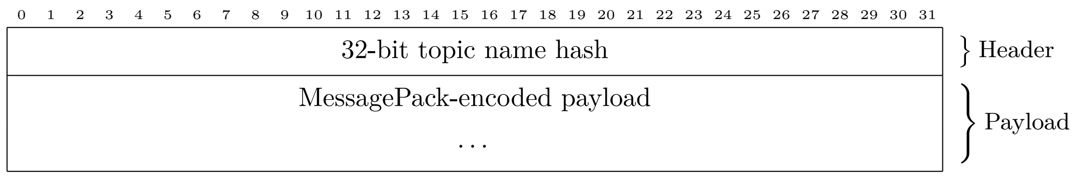


Figure 6. The network application protocol.

ROS 2 publications, subscriptions, services, and service clients are supported with automatic message translation in both directions. See Section 4.4 for more information on topics used in the swarm. The ROS 2 project originally investigated the use of ZeroMQ, but chose DDS for ease of development at the cost of messaging customization (Woodall, 2014). We have implemented ROS 2 messaging over ZeroMQ and have realized many of those performance and customization gains. While we have lost some of ROS 2/DDS's flexibility (such as automated topic discovery), these changes allow the system to function under the heavy constraints imposed by our 200+ vehicle mesh network.

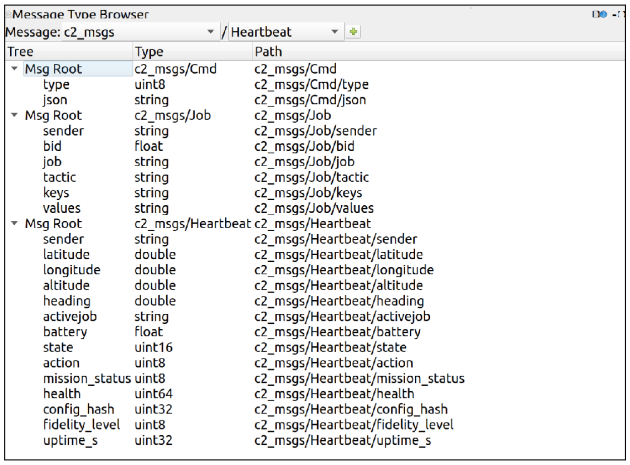
4.3.6. Emergency Control Channel

We also maintain an emergency out-of-band control channel in the 2.4 GHz range. The Swarm Operator may command UAV emergency stop via this channel, which will immediately cause all UAVs to land and disarm.

4.4. Swarm API

As mentioned in Section 4.2.1, ROS is utilized throughout RISE and is the common API interface. As with any ROS-based system, the interfaces between nodes are well defined by ROS messages, publishers, and subscribers. Agents in the swarm primarily interact through three ROS 2 messages which we define: the Heartbeat, Cmd, and Job messages. Figure 7 shows the contents of these three primary messages utilized by swarm agents and Swarm Engine. Individual agents of the swarm must populate the elements of the Heartbeat message and send that information repeatedly on the /a2c/heartbeat topic. As agents receive tasking or other factors occur, they must update the respective fields in the message. All Swarm Engine entities listen to that message topic to automatically discover swarm agents and add them to their agent table. Likewise, Swarm Engine instances will broadcast the same Heartbeat message information on a /heartbeat topic so that swarm agents are aware of all the Swarm Engine instances that are online and other Swarm Engine instances are also aware of each other.

The Job and Cmd message are used as part of the bidding process that makes up the swarm task allocation. Figure 8 demonstrates how tasks are generated from tactic execution and are used to prompt swarm agents to bid via the Job message and bidding topics. After task evaluation has been completed based on bid responses, a command message is sent out to the swarm, instructing agents to complete the requested tactic. This interaction is facilitated by the Cmd message, with agents subscribing to the /command topic to determine if they have had a task assigned to them. If an agent has been tasked, logic flows into the agent primitive part of the architecture. Otherwise,



Message Type Browser

Message: c2_msgs / Heartbeat

Tree	Type	Path
Msg Root	c2_msgs/Cmd	c2_msgs/Cmd
type	uint8	c2_msgs/Cmd/type
json	string	c2_msgs/Cmd/json
Msg Root	c2_msgs/Job	c2_msgs/Job
sender	string	c2_msgs/Job/sender
bid	float	c2_msgs/Job/bid
job	string	c2_msgs/Job/job
tactic	string	c2_msgs/Job/tactic
keys	string	c2_msgs/Job/keys
values	string	c2_msgs/Job/values
Msg Root	c2_msgs/Heartbeat	c2_msgs/Heartbeat
sender	string	c2_msgs/Heartbeat/sender
latitude	double	c2_msgs/Heartbeat/latitude
longitude	double	c2_msgs/Heartbeat/longitude
altitude	double	c2_msgs/Heartbeat/altitude
heading	double	c2_msgs/Heartbeat/heading
activejob	string	c2_msgs/Heartbeat/activejob
battery	float	c2_msgs/Heartbeat/battery
state	uint16	c2_msgs/Heartbeat/state
action	uint8	c2_msgs/Heartbeat/action
mission_status	uint8	c2_msgs/Heartbeat/mission_status
health	uint64	c2_msgs/Heartbeat/health
config_hash	uint32	c2_msgs/Heartbeat/config_hash
fidelity_level	uint8	c2_msgs/Heartbeat/fidelity_level
uptime_s	uint32	c2_msgs/Heartbeat/uptime_s

Figure 7. RQT Message view showing some of the primary API messages. Heartbeat, Cmd, and Job.

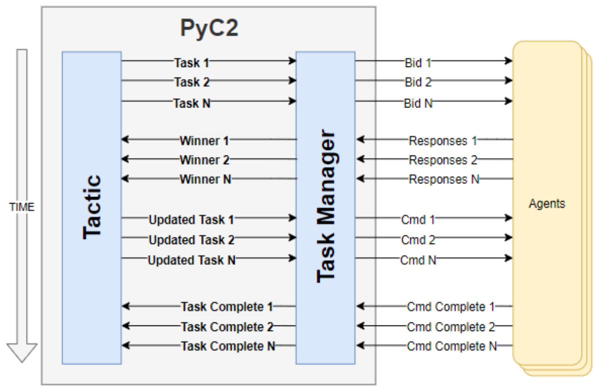


Figure 8. Bidding command flow.

agents continue to respond to task requests on the /bidding topic as additional tactics are executed over the course of the mission. A sequence diagram showing the swarm in operation is available in Figure 9.

4.4.1. Command API

Agent task assignment within RISE is handled via a centralized bidding system. While research on decentralized task allocation is quite common (L. Johnson, Ponda, Choi, & How, 2010) and RISE has even been integrated with third parties that have implemented decentralized task allocation (see Section 10.1), we found centralized tasking to be the ideal solution for RISE's needs. First, centralized task allocation provides a much higher level of control of the swarm. It is abundantly clear when agents will take tasking and who the anticipated agents will be in a centralized system. Another primary reason for the centralized tasking is network load. With the numbers of agents in our use case, we need to limit network traffic wherever possible, and a centralized system allows for this.

The bidding system works through the use of the job and command messages described in Section 4.4. As a swarm operator provides tactic input, a series of tasks or "jobs" are created within our system. These jobs are then sent out via the /bidding topic and agents respond with a heuristic-based bid. The agent heuristic looks at the job being requested, the location of the job relative to itself, battery life, vehicle health, and a few other factors, before it responds with a bid. PyC2 then evaluates all job responses received and eventually assigns tasking for all the jobs in a respective tactic. This final tasking is sent out via the /command topic and message. Figure 8 showcases the standard interaction over time for this overall process. Multiple tactics and jobs can be simultaneously bid out.

4.4.2. Intelligence API

All agents in the swarm are constantly searching for intel, regardless of the task they are performing. This intelligence is primarily gathered through the various platforms' RGB cameras. While YOLO or some other variation of object detection has been utilized during RISE, intel in DARPA OFFSET field experiments was primarily represented by AprilTags[6] (See Section 11.3.1). Agents utilize an AprilTag detection algorithm to report to the swarm what they saw via a common detection message which contains information such as who saw the artifact, what does it represent in its current context, where was the artifact, and so forth. All agents and Swarm Engine instances listen for all reported detections from each other. As detections are reported, agents record the events in another subcomponent of Robot Core called the central data store. This information is stored and queried by certain primitives as any prior or dynamic intel is required for that primitive execution. Section 5.5.1 discuses how Swarm Engine visually represents reported intelligence from swarm agents.

[6] https://github.com/AprilRobotics/AprilTag

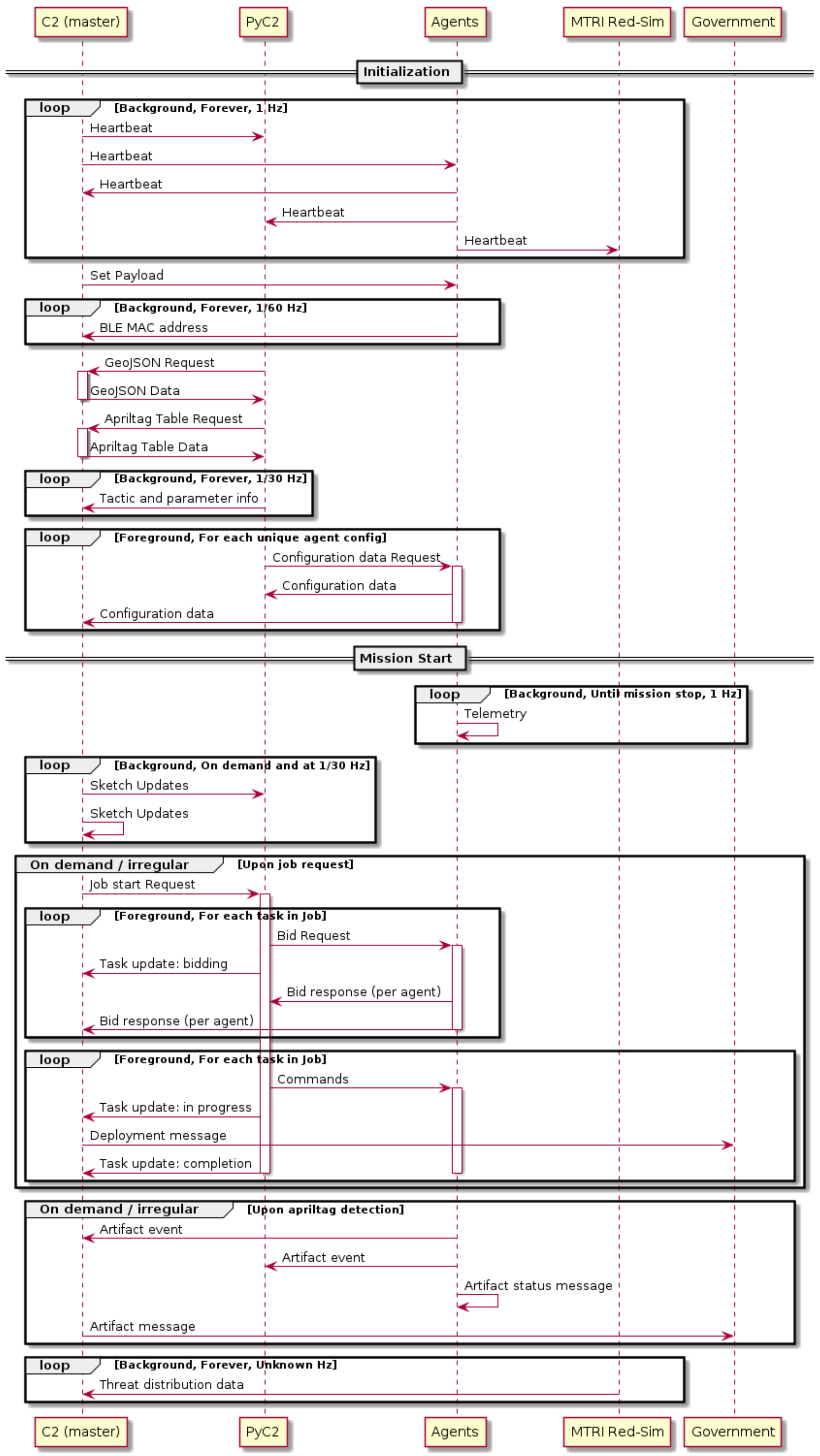


Figure 9. Sequence diagram showing messages sent over the network during swarm operation.

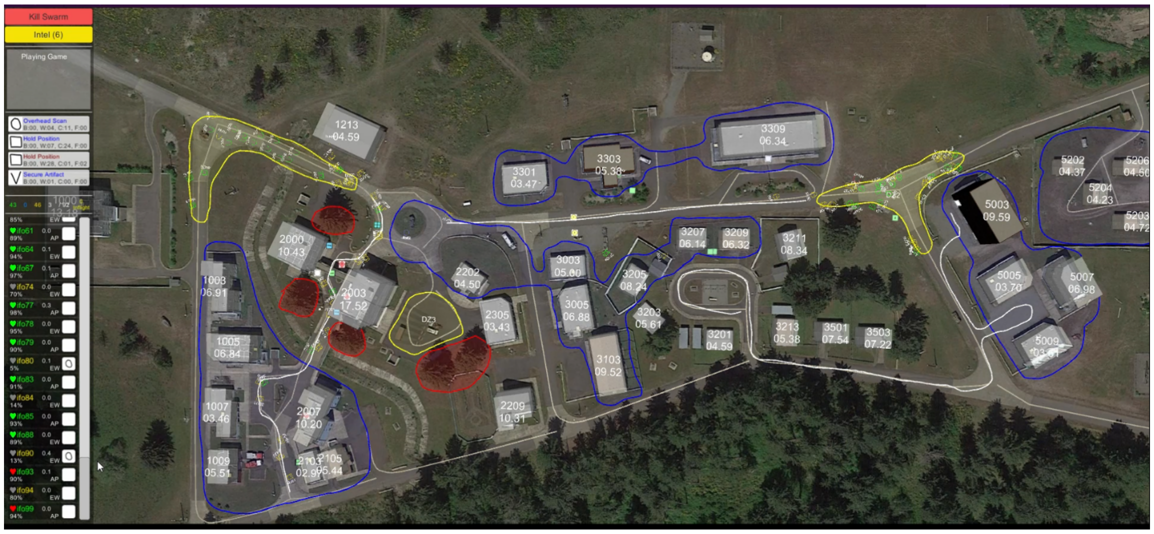


Figure 10. Mission plan sketch drawn by the swarm operator during the fourth DARPA OFFSET field experiment at Joint Base Lewis-McChord.

5. RISE: User Interface

We built our C2 user interface to balance the needs of swarm commanders, operators, and health engineers. Swarm commanders are primarily concerned with situation awareness (SA), one aspect of which is "the perception of elements within a volume of time and space" (Endsley, 1995). Elements of interest include blue and red force positions, terrain layout, and mission plan data. Although operators must similarly maintain SA, they are also responsible for swarm command—the issuance of tactics that effectuate the commander's plan. Given that robotic swarms interact with the environment, tactics are inherently grounded in a temporal and spatial context. The nature of SA and tactic invocation therefore implies that both the commander and operator primarily communicate with the swarm through the environment. On the other hand, swarm health engineers, as do researchers, practitioners, and developers, require access to detailed information that commanders and operators do not necessarily require. Agent task status, logs, component-level health information, and trajectory information are example data that engineers use to diagnosis issues occurring during a mission. Although, health engineers utilize SA and may aid swarm command, being able to quickly locate and interact with specific agents is critical to their success. Figure 10 illustrates how we balance these requirements.

The proposed cross-platform compatible interface was refined through an iterative design approach spanning four years. Our experiences participating in five large-scale field experiments across five Combined Arms Collective Training Facilities (CACTFs), along with periodic field integration testing and regular development, helped inform our design. As shown, the interface is divided into an interactive panel (left) and scene view (right). Given that the environment is our primary communication channel for SA and swarm command, we appropriate significant screen real estate to the scene view. Bandwidth is further maximized by abandoning traditional WIMP (windows, icons, menus, and pointers) design patterns. Instead, we implement a context-sensitive gesture-based user interface, whereby users interact with C2 via sketch commands. This decision also serves our desire for cross-platform capability, as we are able to preserve limited mobile device screen space. In the remainder of this section, we expand on these ideas and describe our interface in detail.

5.1. Cross Platform Compatibility

We designed a cross-platform compatible C2 by implementing techniques that utilize only 2D input, which we regard as the least common denominator among all input devices. Mouse, stylus, and touch input are inherently 2D. Three-dimensional controller and hand pose data can be made 2D via planar projections. For this reason, baseline C2 techniques learned on one system transfer to all supported systems. To date, we have ported our software to Windows, Linux, Android, HoloLens 2, and HTC Vive. It is important to note, however, that although we implement a common cross-platform interface, we are still able to exploit affordances offered by more capable systems. Additional information on alternative HMI interfaces supported by RISE can be found in (Williamson, Taranta II, Moolenaar, & Laviola Jr., 2023).

5.2. Gesture Interface

Human motion that intentionally conveys information is a gesture. In the context of this work, gestures are motion patterns captured by an input device that map to software commands—when we recognize a known pattern, we invoke the associated function. To recognize input patterns, we employ Jackknife (Taranta et al., 2017), a device-agnostic custom gesture recognizer.

Being device-agnostic means we are able to recognize mouse, touch, stylus, hand, and 3D controller gesture input, among other modalities using the same recognizer, which enables rapid integration of new input modalities. Being customizable means the recognizer learns from a small set of example input patterns. Jackknife specifically achieves high accuracy ($> 90\%$) with only one training sample loaded and improves with more training data.[7] Since we only require minimal training data, users can customize the interface according to their preference, which has potential to increase learnability and memorability (Nacenta, Kamber, Qiang, & Kristensson, 2013). Further, because Jackknife uses a nearest neighbor pattern matching strategy, we can train the recognizer online in real time, enabling us to define new gesture classes on demand. When RISE implements a new feature, but the associated invocation gesture is unknown, our interface prompts the user for training data, after which the recognizer is retrained and the new feature is immediate accessible. This enables PyC2 to employ new tactic and sketch parameter types without direct C2 support.

All interface functions are accessed via gesture commands, including system commands to start and stop missions, agent commands to access logs and video feeds, and tactic and tactic parameters commands. Besides preserving screen space, an additional advantage in using gestures over WIMP design patterns is that C2 and PyC2 developers can add new functionality without having to reorganize menu hierarchies or toolbars, thereby accelerating development time. More importantly, however, gestures reduce mode switching and allow the user to interleave commands, potentially increasing command throughput.

To aid swarm operators learn the user interface, we developed an interactive training module that walks users through the basic agent, command and control, and navigation interactions. Separately, all available gesture commands can be found through an online help system shows the user how to draw each symbol and informs them on their usage.

5.3. Agent Interface

More so than commanders or operators, swarm health engineers are concerned with individual agents. We therefore provide access to essential agent interactions through the left panel shown (Figure 11). In this panel, we generate a list of all known agents—those for whom we have received one or more heartbeat messages. Each list entry communicates the agent's name, payload, status, and tactic icon.[8] The agent name's color encodes its status, which may be idle (green), killed (orange), manually disabled (red), tasked (blue), or unknown (orange). This later state indicates that C2 has lost communication[9] with the agent.

[7] On 2D gestures specifically, Jackknife achieves 95% accuracy in a writer-independent recognition scenario with two templates loaded.
[8] Although agents execute primitives, we are able to trace its work back to the tactic that generated it.
[9] Communication is lost when C2 has not received a heartbeat in seven seconds

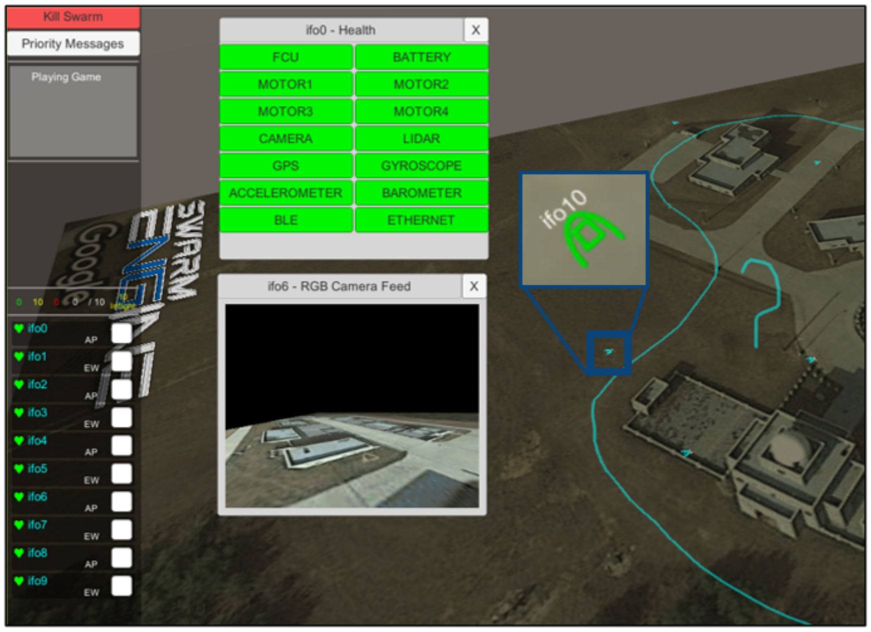


Figure 11. User interface elements illustrating the agent list (left), video stream, component health panel, and agent icons within a map view simulation.

A user may access agent-specific data by gesturing on the agent's button, where one's initial contact point selects the agent. A right-swipe causes C2 to position the camera over the agent in the scene view. An *h* gesture opens the agent's health information window wherein all relevant hardware components are listed with color coded status, those being nominal (green), missing (gray), intermediate (yellow), or critical (red). Color and depth camera streams from the agent can be viewed by *h* and *)*, respectively. Finally, *L* displays the agent's logs. Although we are able to provide access to additional features, these were found to be most relevant for initial agent SA and diagnosis. A health swarm engineer will have access to additional information via SSH remote terminal access.

We present each agent as an icon within the scene view using a platform-specific symbol. Its color reflects the agent's status using the same scheme described above. Within the agent icon, we render a tactic symbol that reflects ongoing or previously completed work in which the agent is or was engaged. In an early user interface iteration, we instead rendered agents as 3D models. However, other than being more aesthetically satisfying, they provided no clear advantage, which we opted for a less resource intensive variant.

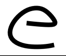
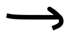
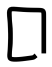
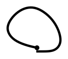
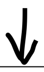
5.4. Command and Control (C2)

The heart of swarm command in RISE comes down to specifying tactic commands, their inputs, and their execution order. These three elements are brought together through our sketch-based interface, as described next.

5.4.1. Tactic Invocation

To invoke a specific tactic, an operator must draw its associated gesture into the environment. When the gesture is recognized, C2 inserts an interactive tactic icon at the stroke's centroid. The operator may then click on the tactic to open its associated pop-up window. This window comprises an input parameter list and two interaction buttons, one that immediately issues the tactic to PyC2 for further processing, and another that simply closes the window to enable mission planning. Table 1 shows five example tactics—their invocation gesture and required input parameters. The "Context" parameter specifies which sketch input parameter type a tactic will act upon.

Table 1. Subset of PyC2 tactics an operator may invoke to fulfill mission objectives. The operator draws a tactic's gesture into the environment to access the command. Each tactic requires zero or more parameters, where context is the sketch input type on which the tactic will execute. The operator can modify remaining parameters through a pop-up window before submitting the command to PyC2. Note, this LaTeX table was generated with PyC2's automated documentation system.

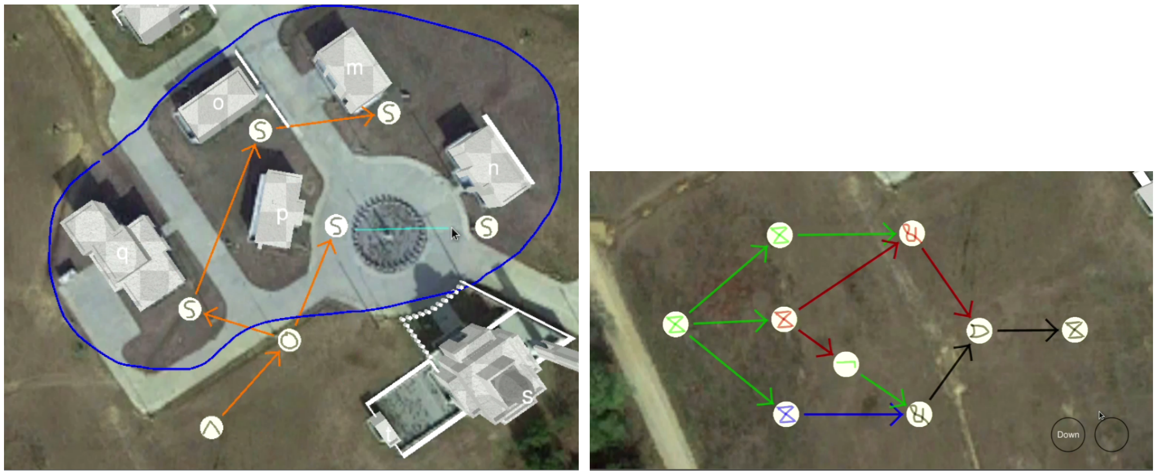
Tactic Name	Gesture	Parameter	Type	Description
Examine Object		Context	POI	Use UAV to scan an object of interest.
		Radius	float	Radius of sphere around object.
Follow Route		Context	Path	Request an agent to traverse the nearest path.
		Altitude	float	Height in meters that UAV will assume.
		Distance	float	Distance between points. Zero to force simplification.
		Use Chaining	bool	Issue one point at a time, for testing only.
Hold Position		Context	Path	Move a set of agents to points along the perimeter and hold.
		Altitude	float	Height in meters that UAV will assume.
		Duration	float	How long to hold.
		Agent Count	int	Number of agents to place along perimeter.
Overhead Scan		Context	Explore	Fly UAVs over area to find artifacts.
		Altitude	float	Height in meters that UAV will assume.
		Cell Size	float	Minimum linear distance between waypoints in meters.
		Agent Count	int	Number of agents used to scan area.
Safe Land		Context	None	For a given air vehicle, find nearby safe location to land.

The location of a tactic icon within the environment is relevant in that tactics optionally use position information to infer operator intention. To illustrate, our scan building tactic infers an operator wishes to scan that building which is closest to the icon. Tactics make similar inferences on sketch parameter input. Our overhead scan tactic, for example, operates on the closest explore area sketch (an input parameter type discussed below). For this reason, an operator may drag their tactic icons through the environment in order to precisely assign position.

Once invoked, tactic status is encoded into the icon's color, being one of the following: pending (black), in progress (blue), failed (red), or successfully completed (green). We also add a tactic button to the left panel in order to provide a quick enumeration of ongoing work as well as provide access to additional developer and swarm heath engineer data. This includes access to children tactic and primitive data and agent waypoint lists when applicable. As the list grow long, it becomes difficult to correlate specific tactic buttons with ongoing field work. However, utilizing spatial memory, one may also return to the tactic icon within the scene view at any time in order to access the same information via a pop window. To cancel a tactic as well as its children tactics and primitives, one may simply scribble-erase the icon.

5.4.2. Tactic Chaining

An operator rarely performs only a single tactic. Rather, he or she deploys multiple tactics in a specific order to advance particular mission objectives. The coordination of multiple tactics is therefore an important feature that we support via tactic chaining. As illustrated in Figure 12, an operator is able to link two tactic icons together via a sketch gesture. The combination of multiple links forms a directed acyclic graph. When an operator issues the root node tactic, C2 submits the entire graph structure to PyC2 for further processing. By default, children tactics execute only after their parent tactics successfully complete. In other words, parent node status propagates to children nodes, and if any parent tactic fails, the child tactic similarly fails. However, this propagation behavior can be modified via the use of logic gate tactic nodes.



Gesture Legend

&	Conjunction	Λ	Create agents	D	Disjunction	¬	Negation
◯	Overhead scan	S	Scan building	⧖	Timer		

Figure 12. Tactic chaining. (Left) Operator drawing a link between two nodes to form a tactic chain link. (Right) Example tactic chain including logic gate-like nodes. Tactic icons and links are color coded where blue indicates the tactic is in progress, black means pending, green denotes successful completion, and red signifies failures.

Logic gate tactic nodes are identical to standard tactics in implementation. One difference is that they override the default propagation behavior (see Section 6). PyC2 presently implements negation, conjunction, and disjunction. The negation gate inverts the status of its parent tactic, which can be useful for planning contingencies. Disjunction gates output a success response if any parent tactic is successful, whereas conjunction gates requires that all parent tactics are successful. Logic gate tactic nodes enable an operator to express sophisticated mission plans in sketch input form.

5.4.3. Tactic Parameters

Tactics require input data that specify their precise behavior on invocation, e.g., a persistent surveillance loitering altitude or safe building standoff distance. Most parameters are typically determined empirically or resolved through automation, though during development or in special cases, an operator may need to specify alternative values. For this reason, certain input data can be modified by an operator through the pop-up window interface. Data types we found to be suitable for the pop-up window include numeric, text, and boolean input. However, other data types are more easily described via sketch-based interactions. Throughout the course of the DARPA OFFSET program in which we developed and integrated numerous primitives and tactics, we encountered a consistent demand for only three sketch interaction types: selection groups, points, and polylines.

- **Selection Groups**. It is sometimes necessary, especially for developers and swarm health engineers, to be able to select individual agents or a group of agents upon which subsequent commands are assigned. To support this operation, we implement a lasso selection technique, whereby the operator may draw a stroke around those objects he or she wishes to group. The operator may continuously lasso select objects until satisfied. Those objects within each new stroke are combined with previous groups using disjunctive union logic, thereby allowing deselection. Any tactic issued by the operator will then be restricted to just those agents in the final selection group. Should the specified tactic generate children tactics, those tactics too will adhere to the same restriction.

- **Points**. Points are geospatial location markers that specify a certain position within the environment, e.g., points of interest, rendezvous points, and breach points. Points are mapped to gestures such that when drawn, the associated point appears at the gesture's centroid within the scene view. The point can then be drag around to a precise location in three-dimensional space, though through customization, axes can be locked to reduce error. For instance, it is common practice to lock points to the ground plane.
- **Polylines**. A polyline is a curve specified by an ordered sequence of geospatial locations that may be open, or closed to form a loop. Sketch strokes serve as the mechanism an operator uses to input polyline data, as strokes are a natural and fluid form of communication. One can therefore use polylines to specify trajectories including preferred routes and boundaries (no-go zones, explore areas, and deployment zones). An operator enters a gesture into the scene view to select a polyline type. The operator then inputs a stroke that defines the initial polyline, after which he or she can modify by sketching directly onto the polyline. The modification stroke acts as a magnetic tool that pulls the polyline in the direction of the cursor. When the polyline type is a closed loop, self loops and intersections are removed. Example routes and boundaries are shown in Figure 10.

To aid with perception and comprehension, tactic developers may customize point and polyline types with unique gesture invocation and rendering properties. See Section 6 for additional information.

5.5. Situation Awareness

C2 provides several tools to aid in the perception and comprehension of swarm intelligence. Swarm commanders can use this information to estimate future status and decide next steps.

5.5.1. Artifacts

We define an artifact as an object of interest. C2 visualizes relevant artifacts detected by the swarm using military iconography as shown in Figure 13. We use standardized iconography to aid with comprehension because of their widespread use and ability to communicate essential information. Two DCRI levels (Self, Miller, & Dixon, 2005) are supported, where C2 renders recognized artifacts using general class icons and identified artifacts using specific icons, e.g., person versus noncombat, medic, or hostile. To gather more information, an operator may view an agent's video stream or command the swarm to surveil the artifact. We further render a red sphere around artifacts that pose a threat. Once neutralized, we remove the sphere.

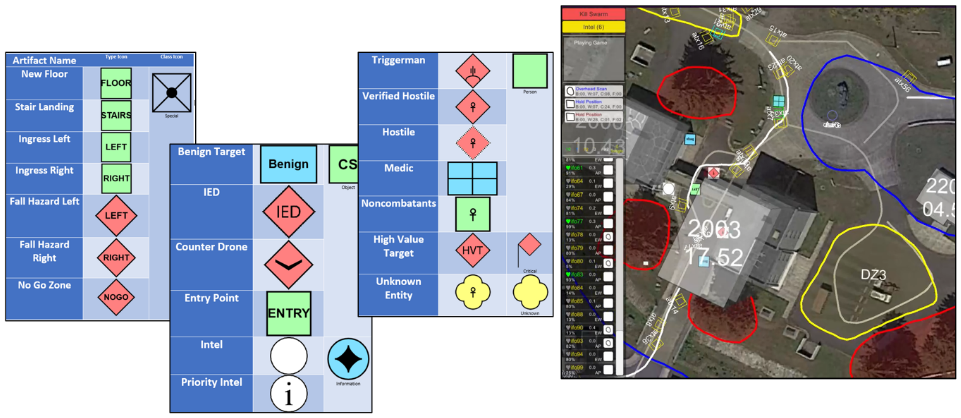


Figure 13. Example iconography used to communicate artifact information during the DARPA OFFSET Joint Base Lewis-McChord field experiment.

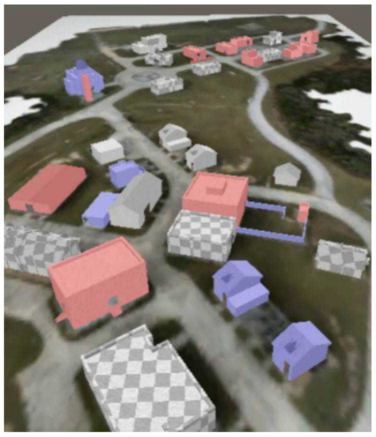


Figure 14. Buildings are color coded in accordance with swarm gathered intelligence information. We render buildings in a checkerboard pattern when the building is known to exist via *a priori* knowledge, which translations to solid color when discovered by the swarm. A building known to contain threats inside is represented by a red shade, and buildings known to contain intelligence information as blue.

5.5.2. Intelligence Information

As an alternative to the tactic and agent list view, a commander may opt to use our intelligence messages view. In this mode, swarm intelligence messages are listed in the left information panel. Unread messages are organized by priority level and require commander feedback to ensure they have been read. Messages referencing scene view data include hyperlinks that move the camera to the associated location. For example, "HVT reported in building 10" embeds a clickable link that moves the camera over the building.

5.5.3. Building Visualizations

C2 visualizes swarm intelligence gathered on building state information as shown in Figure 14. Specifically, we render *a priori* known building geometry in a checkerboard pattern. This indicates that the swarm expects to find a building, although its status is unconfirmed. Once the swarm verifies its state, we render the building solid. Additionally, buildings that house threats or intelligence are shaded with an oscillating red or blue tint, respectfully, and a quick glance of the scene view quickly reveals areas where additional attention may be required.

5.5.4. Grid Data Visualization

Swarms may track data that is well represented as a grid overlay. For this reason, we provide a mechanism by which swarm entities (any networked components including agents or third-party programs) may communicate custom grid data to C2. Figure 15 illustrates two examples born out of our collaborations with Michigan Tech Research Institute. The left grid visualization is a probability threat distribution where denser probabilities (brighter red) correlate with larger threats. Initially, initial uniform distribution evolves into a multimodal distribution as the swarm gathers intelligence and red forces are isolated. The right grid visualization illustrates an acoustic map. This is the area where deceptive firefight sounds would be heard if projected from under the operator's cursor. One can see how grid data visualizations may be useful in several contexts; for example, instead of visualizing acoustic zone data, the swarm could report on network communication ranges, among other data.

5.5.5. Temporal Coverage

In order to visualize coverage over time, RISE C2 tracks agents as they move through the environment. Specifically, for each agent, across each frame, we cast rays from each camera into the virtual environment. The resulting collection of hit points yield a coverage estimate for the given frame.

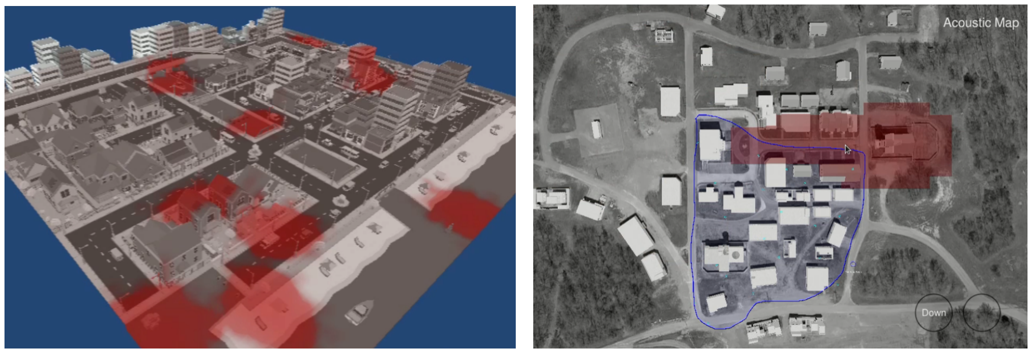


Figure 15. Example grid data visualizations. (Left) Threat distribution map. (Right) Acoustic zone map correlated with the opreator's cursor position.

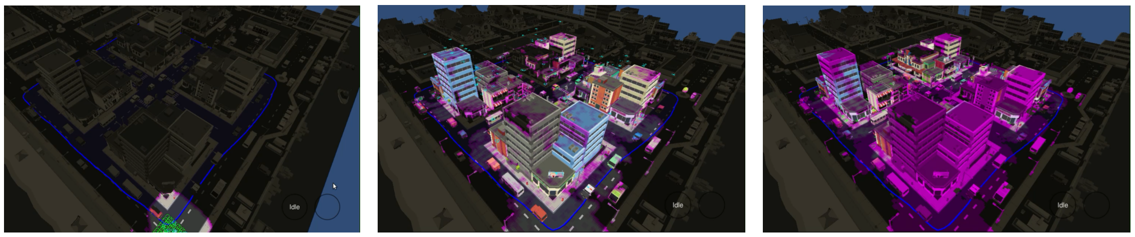


Figure 16. Temporal coverage visualization as time progresses from left to right. Unexamined areas initially render in a dark sepia tone. As agents navigation through the environment, explored scene geometry comes into full color. Finally, as time progresses and we lose situation awareness in particular areas, color fades to magenta.

All hit points are then cached in a GPU-based voxel hash table representation of the environment that a custom temporal coverage shader uses to render the environment. Each entry stores a voxel ID and timestamp. When the said shader renders a fragment, it queries the hash table to determine when the voxel was last seen by an agent. As shown in Figure 16, if the associated voxel is absent from the hash table, we assume the voxel has never been observed, and so we render the fragment in a dark sepia tone. Otherwise, the fragment is colored based on a linear interpolation between full color to magenta, based on the time passed since the voxel was observed.[10]

This visualization has several practical applications. Practitioners, for example, can use temporal coverage data to quickly validate reconnaissance and persistent surveillance tactics work according to expectation. Swarm operators can maintain situation awareness to verify where within the mission area knowledge is absent or outdated. Figure 17 shows the temporal coverage visualization in use at the DARPA OFFSET sixth field experiment during a Northrop Grumman and BBN joint run scenario. The swarm commander was able to quickly measure progress against mission objectives and modify the mission plan accordingly to increase coverage.

5.6. Supported Environments

C2 supports three environment types of varying complexity. The simplest environment we support is an image based top-down map view that C2 loads from a public repository based on given GPS coordinates. This environment is useful for *ad hoc* testing that may occur during travel or when 3D models are unavailable. Second, C2 can load GEOJSON encoded data, which can be useful for randomized as well as custom scenario testing. However, when more complexity is required, developers can export custom Unity scenes that C2 can load. Finally, a number of custom scenes

[10] We fade to full magenta after twenty minutes.

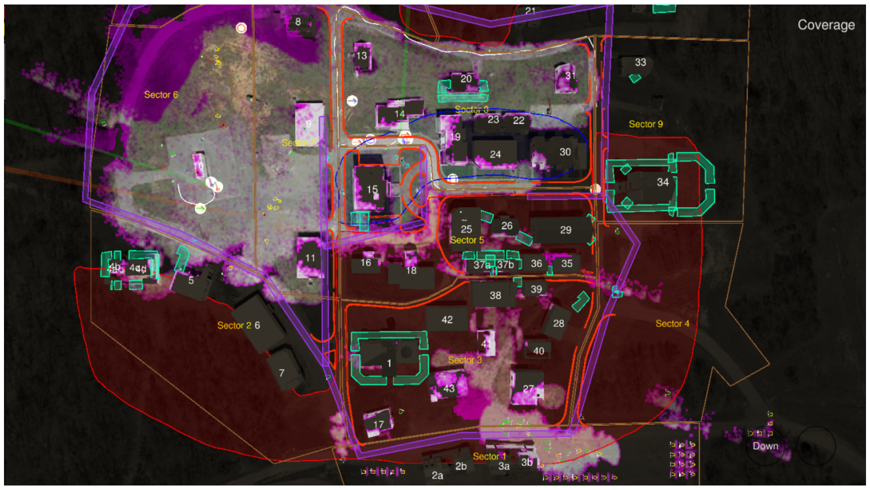


Figure 17. Temporal coverage visualization captured in the course of a joint Northrop Grumman and BBN exercise during the DARPA OFFSET sixth field experiment at Fort Campbell. Dark areas have not yet been examined. Full color areas have current situation awareness. Magenta-colored areas have been previously examined, but the situation data is now stale.

are also built into RISE, namely, the Joint Base Lewis-McChord, Camp Shelby, and Fort Campbell CACTF sites.

6. Tactics Development Interface

PyC2 provides a straightforward interface for rapid tactic development. To implement a new tactic, one derives a new class from the PyC2 Tactic base class and specifies a tactic definition as shown in Listing 2. All information required by C2 to invoke the tactic is included in this definition, including the command gesture, required input parameters, as well as human-readable descriptions of the tactic and its parameters. When PyC2 discovers a new C2 device via its heartbeat, we transmit all tactic definitions over the network. When C2 receives a new tactic definition, it will retrain the gesture recognizer (collecting new samples if required) so that an operator can immediately invoke the tactics. The associated tactic pop-up window (see Section 5.4.1) is automatically populated with numeric, boolean, and text parameter data entries. This design enables one to focus on core tactic logic without having to concern oneself with user interface integration, e.g., GUI layout design, arrangement, and embeddings within hierarchical menus.

When C2 invokes a tactic, PyC2 receives a network message with all nonsketch parameter data encoded into it. PyC2 parses the message, verifies data validity, checks for errors, and resolves context. That is, since one can write a tactic that works under different contexts, PyC2 will additionally examine all available context options and select that which is closest to the gesture's position. For example, our overhead scan tactic operates over either an explore area or sector. If the closest explore area polyline point is less than that of the closest sector point, Pyc2 will select the explore area. Errors are reported back to C2, otherwise, all parameter data including lasso-selected agents are copied into a task object. PyC2 then instantiates the associated tactic class, passing in the task object as its only parameter. In this way, all boilerplate processing is handled automatically, and again one can focus primarily on tactic logic.

Any tactics may invoke other tactics or robot primitives, each of which are handled as individual children tasks. Therefore each child task encodes a tactic or primitive name, along with its required input parameter values. PyC2 implements a wrapper task class for each primitive that contains parameter data selection, manipulation, and transformation routines specific to the primitive. In most cases, specialized logic is not required because the encoded primitive is a direct copy of

```
# SketchInputTypes.PointOfInterest
{
    'basic_type' : PARAM_CUSTOM_POINT,
    'type_id' : int(SketchInputTypes.PointOfInterest),
    'gname' : 'lowercase_p',
    'parameters' :
        json.dumps(
            PointProperties(
                display_name = 'Point of Interest',
                sphere_color = [0.0, 1.0, 0.0, 1.0],
                beacon_color = [0.0, 1.0, 0.0, 0.1],
                radius_m = 1.0,
                lock_axes = 2).__dict__)
},
```

Listing 1. Dictionary entry for a point of interest type in PyC2. The type, id, gesture command name, and rendering properties of this custom point type are automatically encoded and transmitted over the network whenever PyC2 recieves new C2 heartbeat.

```
class OverheadScan(Tactic):

    # Tactic description data that is sent to C2 during registration.
    TacticDefinition = TacticDefinition(
        'Overhead Scan',
        'Fly agents over an area to find artifacts.',
        'circle',
        context = [
            SketchInputTypes.ExploreArea,
            SketchInputTypes.Sector],
        parameters = [
            TacticParameter(
                'Altitude',
                'Height in meters that UAV will assume. Use -1 for automatic selection.',
                float,
                -1),
            TacticParameter(
                'Cell Size',
                'Minimum linear distance between waypoints in meters.',
                float,
                10),
            TacticParameter(
                'Agent Count',
                'Number of agents used to scan area.',
                int,
                5)])
```

Listing 2. Tactic definition for Overhead Scan, which subdivides a given area into a number of smaller regions based on the input parameters and generates one primitive move-to command for each region. A tactic definition comprises a tactic name, short human-readable description, gesture, context (which can be an explore area *or* a sector in this example), and parameter list where each entry specifies a human-readable name, description, data type, and default value. Each tactic definition is encoded and broadcast over the network whenever PyC2 receives a new C2 heartbeat.

key-value (parameter-value) pairs. However, in more complex cases, primitive encoding may be situation-dependent. Each child task is queued via the tactic base class's queuing mechanism, and PyC2's task management system handles task bidding, assignment, and invocation as described in Section 4.4.1. Appendix A presents a complete tactic example. In the remainder of this section, we discuss how tactic behaviors can be customized and what tools PyC2 provides to enable efficient tactic development.

6.1. Tactic Execution Customization

The simplest tactics are those that generate children tasks and nothing more. Our tactic base class provides default behaviors for all task processing stages and error handling. However, there are often situations where one must override default tactic behaviors in order to extend its capabilities. A tactic may wish to issue several primitives to a particular agent over time, such as to move to a particular location and then secure a threat. In another case, a tactic may encode certain parameters based on which agent won a bid. Or a tactic may provide custom error handling logic, and so forth. For this reason, one may override the default bid completion, bidding failure, task completion, task cancelled, task failure, and tactic completion callback handlers.

One may also override a tactic's prerequisite check. As described in Section 5.4.1, an operator may chain commands together using gate-like tactic nodes. By default, we only invoke a child tactic after all parent tactics on which the child depends have completed successfully. Otherwise, we fail the child tactic. When this behavior is inappropriate, one can employ a custom prerequisite check, as we have done with the conjunction, disjunction, and negation tactics.

6.1.1. Sketch Input Customization

As illustrated in Listing 1, one may generate new custom sketch input types based on C2's generic point and polyline classes. In defining a new type, one specifies its underlying type name, unique identifier, command gesture, and render properties. Once defined, custom parameter types can be used as context that PyC2 resolves when a tactic is invoked. Two benefits arise from this approach. First, by having unique parameters types, users can more easily distinguish their presence and purpose in C2. Second, it helps resolve ambiguity when a gesture is close to multiple different sketch types—there are fewer chances for operator error.

Custom parameters are also quick to implement and put into practice. For example, during our final field experiment at Fort Campbell, we decided that agents should return to a recovery point to facilitate field operations. Our safe land tactic previously landed agents at unobstructed locations when their battery level fell below a certain threshold. Thus we created a new input parameter type and updated our tactic such that if a recovery point was specified somewhere within the scene, we would instead command agents to a location near the recovery point. This tactic was implemented and validated in simulation within two hours and fielded the same day using more than seventy air agents.

6.2. The Tactic Development Toolkit

PyC2 comes equipped with a variety of features and utilities that enable intelligent tactic design. The most relevant tools follow.

- **Scene Geometry**. C2 transmits *a priori* scene geometry to PyC2 during initialization. This information includes GEOJSON encoded building label, boundary, wall, and height data. Tactics can use this information to inform reconnaissance, surveillance, and other maneuvers that interact with buildings. Tactics can also use building data as context, e.g., to select which building an operator intends to scan.
- **Artifact Support**. PyC2 tracks artifacts with which the swarm may interact, along with basic state information such as whether the artifact has been recognized or detected, and whether the artifact is a threat or has been neutralized. Like sketch input and buildings, tactics may use artifacts as context. Secure artifact is one example tactic where the operator may manually command the swarm to neutralize a threat. In this case, the tactic chooses that artifact which is closest to the operator's secure tactic gesture.
- **Path Planner**. We include a jump point search (JPS) (Harabor & Grastien, 2011) based path planner implemented as a multilevel grid. The underlying grid structure encodes known obstacles, including buildings and no-go zone sketches. A tactic can use the JPS planner to recommend optimal paths through the environment for air and ground agents. An agent who receives this path information as a primitive input parameter may optionally follow the

prescribed path to a specified destination using its own local path planner to avoid dynamic obstacles while coordinating airspace maneuvers with other agents.
- **Sketch Input**. All operator sketch parameters are transmitted over the network when they are created and as they are modified. PyC2 stores each parameter in a local database with which tactics may interact. Utilities for querying sketches based on type and distance, analyzing geometric properties such as length and winding, and transforming sketches by spatial resampling and line simplification are included. These tools enable developers to efficiently analyze context and generate work based on free form sketch-based user input.
- **Sketch Generation**. We also provide mechanism that enable developers to generate and publish new sketch data. In advance of the DARPA OFFSET sixth field experiment, the Naval Information Warfare Center Pacific (NIWC) provided curb, wall, and power line footprint data. We encoded this information as custom sketch data that PyC2 treated as no-go zones. As such, our path planner was able to route around these obstacles.

7. Primitives Development Interface

All tactics are made up of primitives, just as all words are made up of letters. Taken as a group, primitives can spell out a vastly more complex action. In RISE, we define primitives as basic actions, individual to a specific agent. They are combined into tactics, which encompass the entire swarm (see Section 6). A single tactic may involve running multiple primitives on the same agent, or even running different primitives on separate agents simultaneously.

Primitives combine high-level individual agent logic with low-level robotic sensing and control algorithms to effect desired behaviors.

We provide a highly modular system for integrating new primitives by making use of the FLEXBE behavior engine. FLEXBE[11] is a ROS package providing state machine creation and execution capabilities, with simple ROS integration and an easy-to-use visual programming interface (Schillinger, Kohlbrecher, & von Stryk, 2016). See Figure 18 for an example of primitive creation using the FLEXBE state machine editor. These FLEXBE state machines are tightly interwoven with the basic operation of each agent, providing many core functionalities. This is important, since it allows primitives to make use of existing agent behaviors using FLEXBE's state machine nesting and concurrency features. Rather than rewriting code to, for example, move the agent to a waypoint, the existing waypoint movement primitive may be nested in the new primitive. In this way, we are able to quickly integrate new primitives with the agent codebase and unlock new behaviors. Each agent's unique set of primitives are contained within platform specific FLEXBE State Libraries.

While the agent primitive system is designed to allow for easy integration of third-party primitives, we provide a number of primitives with RISE. These include basic movement, mission interaction, building breach and entry, building interior exploration, and more.

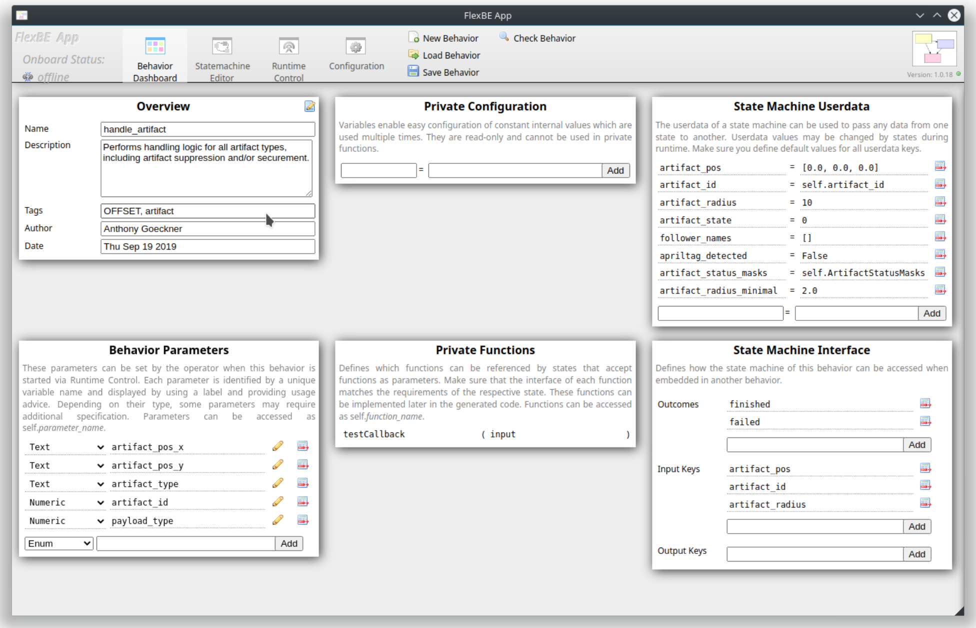
8. Algorithms Development Interface

Low-level robotic algorithms form the basis of agent capabilities. We implement many of these from the ROS ecosystem, but a number are our own creations for RISE. The majority of these algorithms are implemented as ROS nodes on each agent's main compute board, although some reside on vehicle flight control units (FCUs), which are based on the Pixhawk standard.

8.1. Sensing

Each agent in the swarm collects data about its local environment. Depending on the class of agent, different sensors might be available. For example, IFO vehicles have a forward-facing stereo camera, downward-facing camera, downward-facing single-point LIDAR, Global Positioning System

[11] http://philserver.bplaced.net/fbe/index.php



Overview

Name	handle_artifact
Description	Performs handling logic for all artifact types, including artifact suppression and/or securement.
Tags	OFFSET, artifact
Author	Anthony Goeckner
Date	Thu Sep 19 2019

Private Configuration

Variables enable easy configuration of constant internal values which are used multiple times. They are read-only and cannot be used in private functions.

State Machine Userdata

The userdata of a state machine can be used to pass any data from one state to another. Userdata values may be changed by states during runtime. Make sure you define default values for all userdata keys.

```
artifact_pos            = [0.0, 0.0, 0.0]
artifact_id             = self.artifact_id
artifact_radius         = 10
artifact_state          = 0
follower_names          = []
apriltag_detected       = False
artifact_status_masks   = self.ArtifactStatusMasks
artifact_radius_minimal = 2.0
```

Behavior Parameters

These parameters can be set by the operator when this behavior is started via Runtime Control. Each parameter is identified by a unique variable name and displayed by using a label and providing usage advice. Depending on their type, some parameters may require additional specification. Parameters can be accessed as self.*parameter_name*.

```
Text      artifact_pos_x
Text      artifact_pos_y
Text      artifact_type
Numeric   artifact_id
Numeric   payload_type
Enum
```

Private Functions

Defines which functions can be referenced by states that accept functions as parameters. Make sure that the interface of each function matches the requirements of the respective state. These functions can be implemented later in the generated code. Functions can be accessed as self.*function_name*.

```
testCallback        ( input        )
```

State Machine Interface

Defines how the state machine of this behavior can be accessed when embedded in another behavior.

Outcomes	finished failed
Input Keys	artifact_pos artifact_id artifact_radius
Output Keys	

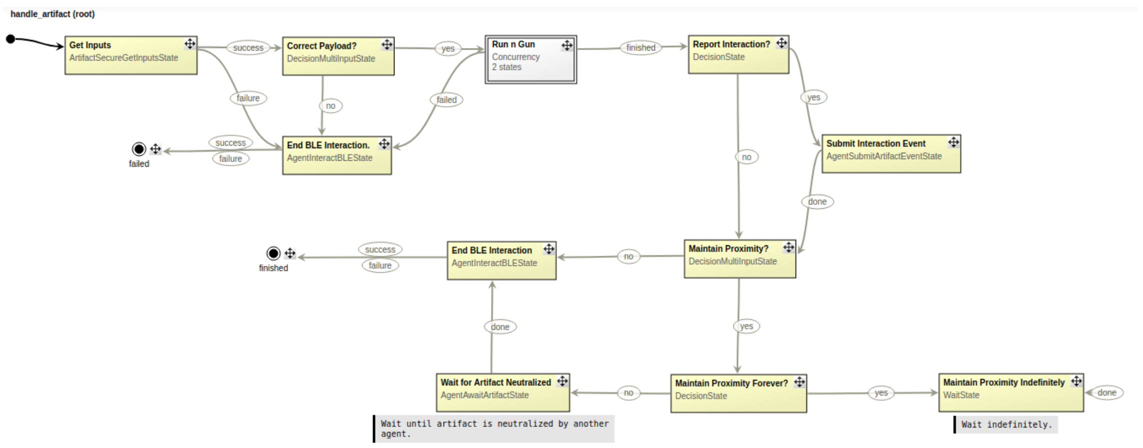


Figure 18. Images of the FLEXBE APP utilized to create agent primitives. At bottom, a high-level mission logic primitive handles interaction with artifacts found in the environment.

(GPS) receiver, magnetometer, altimeter, and accelerometer. ATX ground vehicles do not have downward-facing sensors, but instead have a 360° circular LIDAR.

We implement drivers to translate raw sensor data into useable ROS messages, which are made available to other ROS nodes running on the agent.Processing of magnetometer, altimeter, accelerometer, and GPS data occurs on the vehicles' FCUs, using both the PX4 and Ardupilot systems depending on the platform. However, processing of camera and LIDAR data occurs on the main compute board of the vehicle and is implemented in ROS nodes. We use a simple speckle filter to remove strands of grass and other nonobstacles from the LIDAR data. For stereo cameras, the driver calculates a disparity image, which it makes available to other ROS nodes for perception.

8.2. Perception

To implement visual obstacle detection on ground vehicles, we take advantage of our stereo cameras to perform binocular disparity-based detection. Our implementation is based on the work of

Oleynikova, Honegger, and Pollefeys (2015) and is highly effective at detecting obstacles while ignoring environmental artifacts such as stems of grass which do not pose a collision risk to the vehicle.

For aerial vehicles, we implement the PX4 Avoidance system, which detects and then avoids obstacles using the forward-facing depth camera (see Section 8.3 below) (Baumann, 2018).

In addition to local data, our agents may also use the telemetry information of other vehicles to avoid collisions. Telemetry information is shared between all vehicles using the MANET. Future work may extend this feature to include prediction and make it more useful.

8.3. Motion Planning and Control

We use data described in the Sensing and Perception sections above to form two costmaps for each ground agent. A two-dimensional local costmap stores detailed information about obstacles close to the agent. This costmap features a decay function such that obstacles not seen for a certain period of time will be removed from the costmap. This allows for accurate tracking of dynamic obstacles. We also maintain a second, global costmap which is used to store obstacle information for the entire mission area. This costmap has no decay function, meaning that obstacles will remain in the costmap until updated by new local sensor data. This two-dimensional global costmap is used for global path planning.

Ground vehicle movement planning is performed by ROS's move_base system. This involves a global planner for long-distance goals and a local planner which is reactive to obstacles blocking the global path. The move_base system also handles high-level actuator control, issuing velocity commands to the FCU.

For IFO vehicles, PX4 Avoidance controls all obstacle avoidance using its own internal costmap. This involves direct control of vehicle velocity and heading. We have not made significant modifications to the publicly-available PX4 Avoidance library, other than to allow the avoidance function to be switched off temporarily. This allows agent primitives to control the vehicle heading for in-place rotation, something impossible with standard PX4 Avoidance.

8.4. Agent Logic

To enable agent primitives that further mission objectives, we provide algorithms for high-level agent logic. These include automatic detection and recognition of AprilTags using the ROS AprilTag library (J. Wang & Olson, 2016), interaction with the environment and mission scenario using Bluetooth Low-Energy beacons, and more. See Section 11 for more information on how we enable agents to interact with the environment.

9. Live, Virtual, and Constructive

We treat agents as logical entities whose underlying components may be centralized or distributed, as well as physical or virtual. This is possible because ROS implements a publishsubscribe messaging pattern over network-based named buses for inter-process communication (see Section 4.3). As such, ROS packages and libraries that use ROS are self-contained in a way that enables robotics modules to run as independent processes. By exploiting this feature, it is possible for one module to publish sensor data while another module analyzes the data and generates locomotion commands that a third module actuates; and although all processes belong to the same logical entity, they may in fact be distributed across disparate systems. We therefore think of a logical entity as being the set of all namespace topics that effectuate a robot instance.

As OFFSET progressed, it became apparent we would need to design a method to allow for varying levels of fidelity within our simulation environment. Typical robotic simulation environments, such as Gazebo (Aguero et al., 2015), provide a robust development environment, but typically only for single agent use. While we initially attempted to utilize Gazebo for swarm development, it became clear that the system would be computationally constrained at around ten platforms. We observed similar issues across other various high level fidelity simulation environments. This pushed us to develop the concept that we call in our system Fluid Fidelity (Section 9.3).

Another implementation choice that we determined to be critical was the ability to have virtual agents working with physical ones. From a logistical perspective alone, it often becomes unfeasible to transport hundreds of platforms to various real world environments for intermittent testing. So while we wanted to test out a new swarm tactic on physical platforms, we needed a way to have a few physical agents work alongside virtual agents to quickly evaluate new capabilities without bringing the whole swarm with us. Due to the utilization of a singular application for simulation and command and control (Section 4.2) as well as our ROS interfacing (Section 4.2.1), we were able to integrate this capability.

9.1. High Fidelity Simulation

For our high fidelity simulation environment, we took inspiration from many industry standard simulations, such as Gazebo and AirSim (Shah, Dey, Lovett, & Kapoor, 2017). Still, our high fidelity simulation does have several characteristics that cause it to vary from many common simulations. Two of the key differentiators is that there is no active physics in the high fidelity simulation and agent bodies consist of simple relative sized cube objects. In place of a physics environment, each platform type has a geometry based PID controller to actuate motion. Agents send actuation commands via ROS Twist messages or setpoint position commands just like on the real platforms. Comparable to other simulators, individual agents receive sensory information from ROS messages. All ROS messages pertaining to sensor sources that would typically be received from a physical sensor are instead generated from the Unity Game Environment. The ROS nodes simulated are TF information, MAVROS data, LIDAR, RGB, and depth, among others. Each of the sensor streams per agent are organized through ROS namespaces. All data are generated as ROS 2 messages, but since the agent codebases expect this information as ROS messaging, we utilized the ROS 2 bridge to translate between them. This ROS 2 translation and a centralized ROS master is handled through an application we call the ROS Simulation Server. The ROS Simulation Server allows us to run multiple high fidelity agents on a single system and has pairing with the simulation system to allow for Fluid Fidelity (Section 9.3). Although Unity now has official support for ROS integration,[12] we needed ROS support before this system existed, and we instead implemented our own method utilizing our ZMQ message system (see Section 4.3).

With the data streams being facilitated, actual agent primitives and algorithms are tested through the simulated Docker containers. The underlying codebase is exactly identical to that which the real swarm agents are running when utilizing high fidelity simulation. There are only a few ROS nodes that need minor adjustments, which is determined via a runtime ROS parameter to make networking configuration changes. ROS nodes that interact directly with the hardware sensors are disabled, as they are no longer needed given the data is being published from Unity. Each individual agent has a corresponding Docker instance, differentiated only by the hostname of the container and the mounted file directory codebase. Figure 19 showcases both a high fidelity simulated IFO and ATX utilizing this system and rendering their sensor streams in RViz as their navigation stacks run.

9.2. Low Fidelity Simulation

Our Low Fidelity simulation capability is vastly different from standard robotic simulators. There are no robotic algorithms running in this configuration, and the whole concept revolves around mimicry. We found when using our high fidelity simulator that around a maximum of ten agents was all that could be achieved on a single machine. While distributing the computational load

[12] `https://github.com/Unity-Technologies/ROS-TCP-Connector`

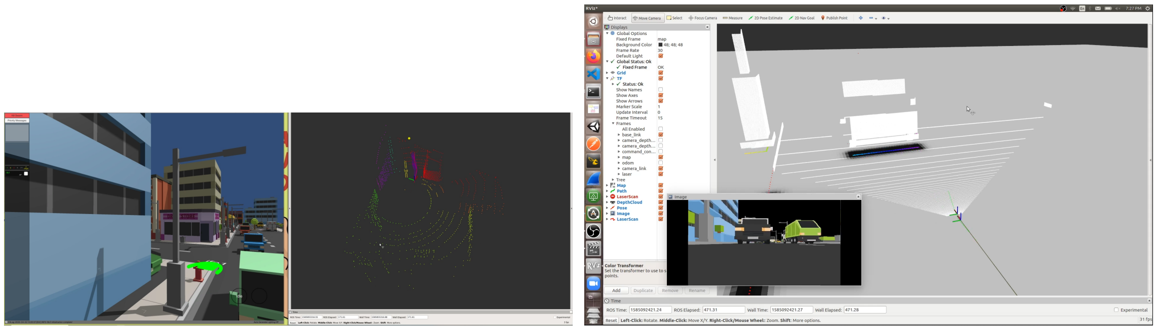


Figure 19. (Left) UAV navigating around a street lamp in simulation and its associated sensor feed as visualized in RViz. (Right) UGV navigating a simulation environment and its associated depth camera sensor feed, also as it appears in RViz.

across multiple systems is possible with our high fidelity simulation, we found it to be not feasible to achieve the scale we were looking for, especially since we lacked cloud compute resources. Tactic development is also sometimes performed rapidly in the field, where additional compute resources or an internet connection are not always guaranteed. These limitations led to the creation of our low fidelity simulator. In this setup, there are no simulated agent Docker containers running, and the configuration consists solely of the Unity Base and PyC2 application (see Figure 3). Unity is rendering the individual game objects for the simulated agents, similar to the high fidelity simulator, but it is not generating any simulated sensor sources except for positional information. In place of generating the raw sensor information, there are components within our Unity environment that are performing rough estimations for things such as agent attrition or agent detection, essentially mimicking the response of the full sensor pipeline in the high fidelity simulation. Another aspect is PyC2 mimicking agent primitives. Since there are no actual agent primitives or algorithms running, PyC2 is in charge of running agent instances that run mock algorithms that simulate the relative agent responses for certain tactic executions. These mock algorithms still send commands via the setpoint position or twist API just like the high fidelity simulation environment. It would be unmaintainable to have PyC2 have mock duplicates of every single agent primitive, so it's important to note that only the most common primitives have direct low fidelity mappings. To handle the rest, that is where the concept of Fluid Fidelity comes in (see Section 9.3).

Figure 20 showcases 100+ low fidelity agents running on a single machine. It is worth noting that regardless of fidelity level or whether the agent is real or simulated, the interface to the end user is always the same. This low fidelity simulation is not particularly useful when doing agent specific development, but is very powerful when performing tactic development (see Section 6). Without the creation of low fidelity, not nearly as many tactics would exist. This lower fidelity also allows for further concepts such as faster than real-time performance, which allows for fast tactic analysis and opens the door to concepts such as optimal course of action generation.

9.3. Fluid Fidelity

As low fidelity simulated agents are unable to simulate some primitives required by certain tactics, we need high fidelity simulated agents to be able to step in and execute those primitives. As described in Section 9.2, low fidelity agents merely have mock algorithms that mimic actual agent primitives, but the collection is not complete. This is where the concept of fluid fidelity comes in. As an operator is running a mission, there may arise situations where tactics are called that low fidelity agents cannot perform. We needed a way to perform a seamless transition to convert a low fidelity agent to a high fidelity agent, so that tactic could still be executed without restarting the runtime environment. The heartbeat message, as described in Section 4.4, contains two properties called "fidelity level" and "config hash." Both of these fields are utilized by our simulation environment to realize what

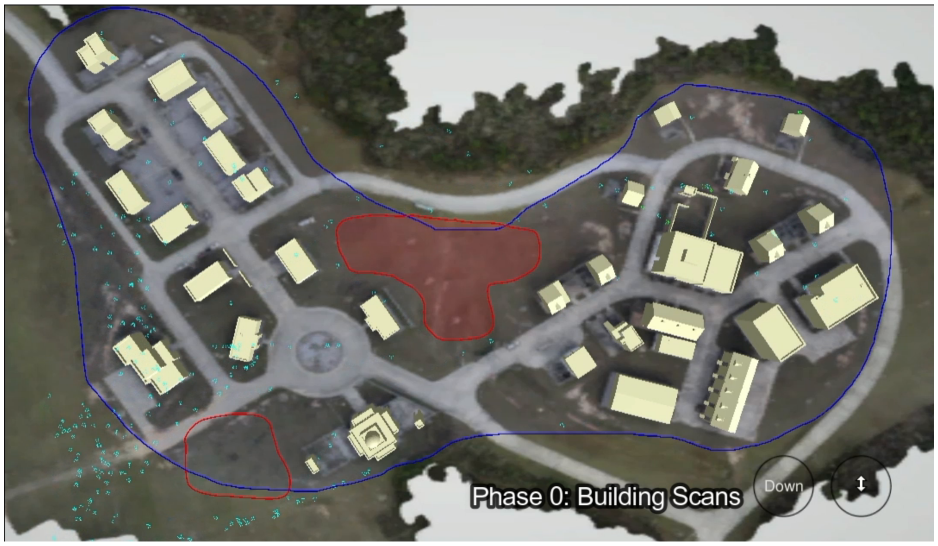


Figure 20. Low fidelity simulation running over 100 agents in the Camp Shelby, MS environment, using a single machine.

kind of agent should be spawned, and the agent's onboard sensors. By adjusting the values in these fields from both the low level simulation and the high level simulation, we are able to make on the fly changes as to what level of agent is running.

9.4. Cross Simulation Support

Over the course of our development, we realized that there are unique situations in which our simulation was missing a required feature we needed. Most of the time, this arises during the creation of very specific robotic algorithms. Whether we needed a detailed physics model, higher quality visuals, or a unique platform with accurate model description, these are gaps within our simulation. Thankfully, there are many other simulation environments that specialize in these areas. Due to the nature of our minimum API required for agent integration Section 4.4, we are able to run Swarm Engine, simulated Docker instance(s), and a separate simulation at the same time. In this configuration, Swarm Engine interprets the simulated agent as a physical platform instead and does not generate any simulated sensor sources. Instead, the separate simulation instance is in charge of generating the simulated sensor sources via ROS messaging. The simulated Docker instance is expecting to receive ROS information, so as long as the simulator is configured with correct topic names and namespacing, we are able to perform this swap. Both Swarm Engine and the separate simulation environment must be able to localize within the same environment, however, most simulators have the ability to configure GPS information. We utilized this configuration to quickly spin up our recently integrated VTOL platforms, the AVTs. Figure 21 showcases an example where we had two simulated AVTs being tasked by Swarm Engine but utilizing the ArduPilot simulation for sensory information. This allowed us to quickly develop and verify the AVT primitives and algorithms that we used on the real platforms without the need of supporting an additional vehicle type in our simulation environment. This type of simulated environment setup was also utilized with Gazebo during integration with the Hive (see Section 10.4).

10. RISE Extensibility: Third Party Integration

In this section, we highlight various third party capabilities that were integrated into the RISE system as part of the DARPA OFFSET program. These third parties, known as "sprinters," comprise small

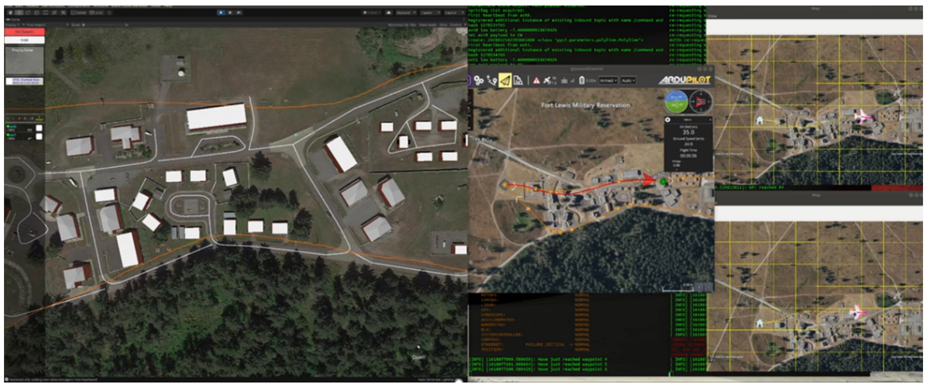


Figure 21. Swarm Engine running alongside the Ardupilot simulator with two simulated AVTs.

business and university research groups that proposed capabilities for specific thrust areas that were developed and integrated during a "sprint." DARPA scheduled five sprints during the program's life that aligned with potential showcases of sprinter capabilities at the regular Field Experiments, or "FX" (see Section 11).

10.1. Heron Distributed Task Assignment System

Heron Systems focused on developing an autonomous task assignment framework via decentralized asynchronous auctioneers to handle the distributed task assignment problem. This would function in place of the existing bidding task allocation system and provide an alternative that offered additional features and capabilities. We closely collaborated with Heron to integrate their capabilities into RISE and leveraged the ROS-based design to handle data passed between Heron's core modules and RISE. Heron's task assignment framework was fully integrated and was demonstrated at a field experiment in Ft. Benning, GA. This capability performed in place of the prior existing bidding system and allowed for a decentralized solution that allowed agents to take on tasks, even if they were not directly in communication with RISE C2 but were communicating with other agents. This provided the ability for tasking assurance and mission persistence, utilizing indirect communication between the intended swarm agent and C2. The modularity in the RISE design and its foundation in ROS allowed for a critical component in the overall swarm operation to be replaced with ease.

10.2. MTRI-SoarTech Synthetic Scan and Dismount Detection

Michigan Tech Research Institute (MTRI) participated in a number of sprints on the DARPA OFFSET program and were seasoned in integrating with RISE. In Sprint 4, MTRI focused on the virtual testbed thrust area to utilize a synthetic technology that may not exist in the real world but could be simulated in a virtual environment. MTRI, in collaboration with SoarTech, aimed to develop the Structure Situation Awareness for Swarms (SSAS) capability, which provided synthetic through-wall floor plan generation and dismount detection via software that was integrated with RISE. We worked with both MTRI and SoarTech to fully integrate SSAS and showcased this capability using physical platforms at the field experiment at Joint Base Lewis-McChord, WA. Using the RISE Swarm Engine C2, we were able to task UAVs to perform flight tactics around buildings and physically perform scanning operations. The physical scans employed the SSAS behaviors to perform a data collection on simulated info of those real physical buildings to then generate the through wall floor plan of that building and also detect simulated dismounted soldiers. The data product they produced included uncertainty information that would closely resemble real life data collects if this technology existed in reality. From the physical flight and the generated data product, MTRI was able to produce a visualization of that data product and show the generated floor plan and dismount detections performed. By integrating with RISE, a virtual synthetic technology and application was demonstrated using actual physical platforms. RISE was able to support the entire workflow without requiring MTRI and SoarTech to develop the whole end-to-end pipeline, allowing them to focus on developing out their specific capabilities while also being able to apply those capabilities in a larger robotics system.

10.3. SoarTech Interior Building Clearing

SoarTech is a seasoned Sprinter that has participated in numerous sprints and has worked closely with us on integrating various capabilities into RISE. During Sprint 3, SoarTech's proposed capability was focused on the indoor environment and the aspect of searching and clearing an interior of a building. Their proposed approach would provide an increase in effectiveness and efficiency for clearing a building floor's interior by implementing a continuous and simultaneous search approach that leverages the multi-agent feature of the swarm. SoarTech developed this capability and wrapped it into a FLEXBE state that was integrated into our existing building exploration tactic. As their focus was solely indoors, they leveraged RISE's existing outdoor navigation capabilities to initially maneuver to the buildings. SoarTech ran their own execution environment as a standalone servlet within the RISE agent Docker containers to perform their capability when it was executed in the FLEXBE behavior. SoarTech utilized RISE's network to share generated building maps between agents. They maintained communication in large buildings by use of dynamically assigned "relay" agents automatically placed at strategic locations to ensure that data could be routed using RISE's MANET (see Section 4.3.2) out of the building and to the rest of the swarm. Through extensive collaborative development and iterative testing, we were able to conduct a full integration of their capability with RISE and our ground vehicles. We demonstrated the execution of their capabilities during FX-3 at Camp Shelby, MS.

10.4. HIVE

For the hardware thrust area of Sprint 5, Sentien Robotics proposed an automated UAV ground management system that provided storage, charging capabilities, and launch/recovery logistics for a larger fleet of agents than their previously developed systems. This system is called Hive, and they developed two variations: HiveXL and HiveISO, which were essentially the same but were just aesthetically disparate due to their different manufacturing processes. Their system was fully housed in a trailer that contained storage bays for a large capacity of drones with a retrieval system that can transport an integrated UAV to and from its charging bay to one of two rooftop launch/landing pads. The system also consisted of maneuvering gantry arms for each launch/landing pad to manipulate the UAV to be in the necessary position for take off and landing retrieval. RISE and the Sentien team worked closely with each other to fully integrate a UAV operation workflow in which RISE C2 could task agents in the Hive, and the Hive would retrieve those agents from their charging bays and bring them to the rooftop launch pad for takeoff. Upon completing their flight operations, the Hive UAVs would return to the Hive and perform a precision landing onto one of the two bays (see Figure 22). Once landed, the Hive would reposition the UAV to the appropriate orientation and then place the UAV onto a charging tray through a bay door. The orientation is important as there is a charging mechanism that connects the drone to the tray for charging in its bay that does require a specific orientation for proper interfacing. Once the vehicle is on the tray, the interior retrieval system will return the vehicle to a charging bay, thus completing a full flight operations cycle. RISE and Sentien were able to demonstrate this full flight operations cycle at the final field experiment at Ft. Campbell, TN. The Hive does provide a significant logistical solution for transportation, charging, deployment, and retrieval of swarm agents but does encounter some limitations in supporting simultaneous launch of numerous platforms. It is better suited for persistent operations of lower numbers of UAV or more consecutive launches, as it would start to encounter congestion of the launch and landing pads. The Hive integration with RISE demonstrates

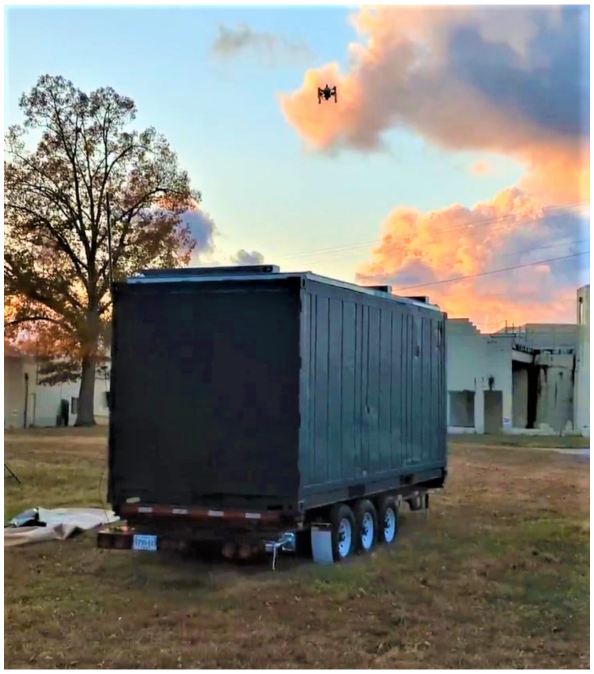


Figure 22. RISE UAV agent landing onto HiveISO at Field Experiment 6.

the ability of RISE to incorporate other technologies aside from robotic platforms that support swarm operations, in this case a logistical technology.

10.5. Integration Summary

The integrations of capabilities from Heron, MTRI, SoarTech, and Sentien Robotics serve as case studies representing diverse examples of some sprinter integration efforts from the DARPA OFFSET program. The various sprints with their different focus areas attracted a large range of capabilities. We were able to work with various sprinters to establish integration paths for their capabilities into RISE and provide a swarming robotics framework to realize the sprinters' capabilities, which ranged from swarm autonomy, swarm tactics, robotics hardware, human-machine interfaces, AI, and more.

11. Field Experimentation

As part of the DARPA OFFSET program, field experiments were conducted to test and demonstrate the program's technological progress. These were conducted approximately every six months and were orchestrated by an experimentation team from Naval Information Warfare Center Pacific (NIWC). In total, RISE was tested iteratively at five urban training ranges over four years. Table 2 summarizes all OFFSET field experiments, and the locations are shown in Figure 23. Whereas FX-0 was an indoor only event, the remaining field experiments took place outside. FX-1—FX-3 gave us the opportunity to test our system in a representative environment, and lessons learned from these experiments informed the design of our architecture as well as our user interface. FX-3 gave us additional insights into scalability issues stemming from our original network solution (see section Section 4.3.1), allowing us to achieve greater scale and operational capabilities in subsequent experiments. Upon reaching FX-4 and FX-6, we had gained the relevant experience and information to consistently field large scale swarm deployments based on mission level objectives.

In the remainder of this section, we discuss the penultimate field experiment, FX-6. This experiment enables us to answer the question: can a multi-robot ecosystem designed for rapid

Table 2. RISE has been tested at one indoor and five outdoor field experiments over the last four years. Note, FX-5 was cancelled due to COVID-19.

Experiment	Date	Location	RISE Objectives
FX-0	March 2018	FDNY, NYC	Test of robotics codebase and architecture. Also, rehearsal for field robotics.
FX-1	October 2018	Camp Roberts, CA	Implementation of multi-agent framework and command and control interface
FX-2	June 2019	Ft. Benning, GA	Swarming operations for primitive environmental Intelligence, Surveillance, and Reconnaissance (ISR) using next C2 iteration, mobile commanding with tablet C2 modality, and a Virtual Reality (VR) interface
FX-3	December 2019	Camp Shelby, MS	Large scale swarm networking and operations with the next-generation gesture-based iteration of the C2
FX-4	June 2020	Joint Base Lewis-McChord, WA	Enhanced swarm tactic execution and improved C2 with Augmented Reality (AR) modality
FX-5	Cancelled	N/A	N/A
FX-6	November 2021	Ft. Campbell, TN	Full scale swarm testing with latest evolution of C2 to include Live, Virtual and Constructive feature and improved AR interface to include commanding

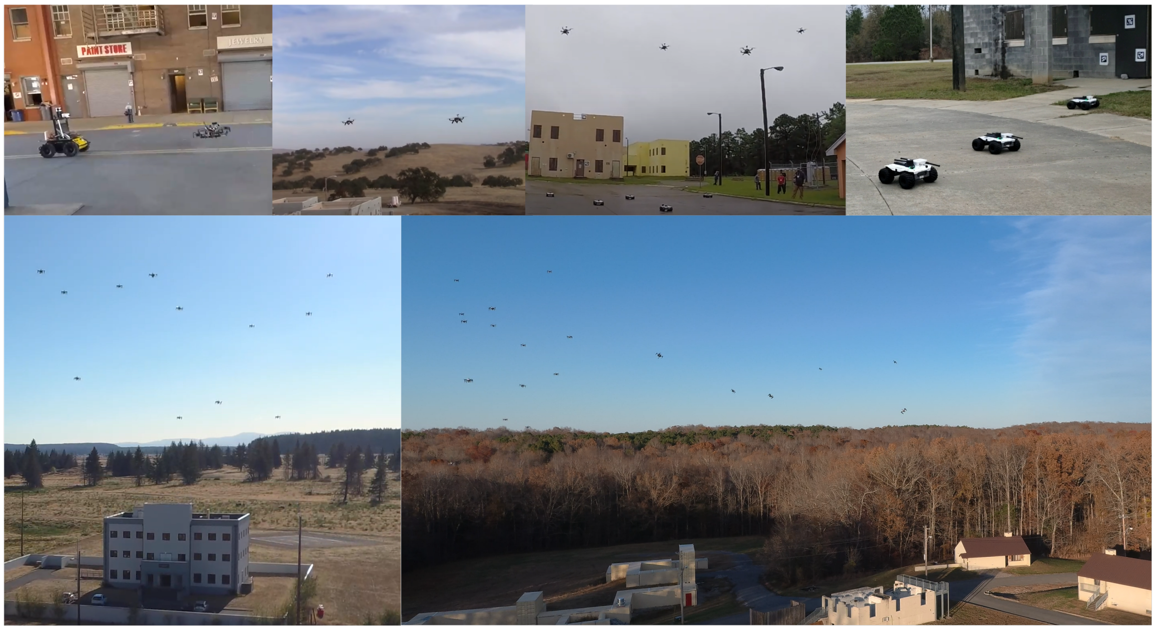


Figure 23. The six field experiment locations corresponding with Table 2. (Top) FX-0–FX-3. (Bottom) FX-4 and FX-6.

integration and one-to-many control can be fielded, where a single operator commands 150+ autonomous vehicles in tactical maneuvers?

11.1. Participants

The Northrop Grumman RISE team filled all primary roles including that of one swarm commander, one swarm operator, one health engineer, and one field operations officer. These individuals were all

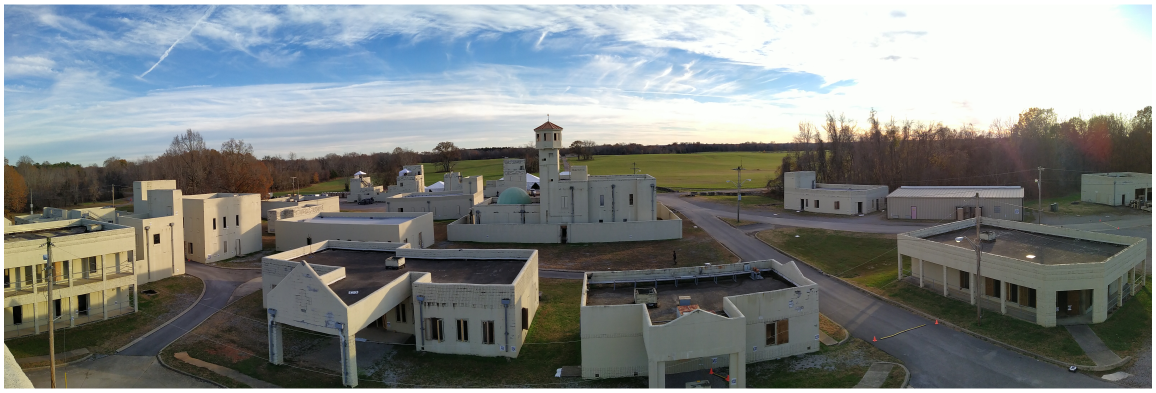


Figure 24. Rooftop view of Cassidy CACTF in Ft. Campbell, TN.

RISE experts with advanced knowledge of the C2 interface and platforms used in the experiment. Further, all except for the health engineer had participated in prior field experiments. The core team was supported by 8–9 field support personnel who assisted with logistics, deployment, and safety. Finally, a Rajant field engineer participating as an auxiliary team member assisted with network infrastructure.

11.2. Facilities and Equipment

11.2.1. Environment

FX-6 was conducted at Fort Campbell in Tennessee on the Cassidy CACTF, see Figure 24. Cassidy is approximately 305 m in diameter and comprises an urban road network connecting approximately forty-five buildings, including a church, embassy, town homes, and mosque, among others. In addition to being densely packed, power lines, curbs, trees, and other obstacles permeate the CACTF.

NIWC augmented the CACTF to support experimentation by distributing scenario-based artifacts throughout the environment. These artifacts are AprilTags (J. Wang & Olson, 2016) (a type of fiducial marker) that represent complex objects. For example, AprilTags are used to represent high value targets, hostiles, improvised explosive devices (IEDs), medics, benign objects, building labels, and priority intelligence information, among others. Certain AprilTags are coupled with field nodes that house a Raspberry Pi and Bluetooth Low Energy (BLE) device that enable interactions between robots and artifacts, e.g., an IED can disable a robot via a Bluetooth interaction when the robot comes within range of the IED. The specific AprilTag family used for this experimentation was 48h12. This tag family was used for its nested tag capabilities. The outer tag could be detected by agents from a further distance and typically only contained generalized information such as person or object. Only the inner tag revealed the true nature of a particular fiducial, revealing whether it was benign, hostile, or friendly. Within RISE, most of the inner versus outer tag interaction was handled autonomously by onboard agent logic (see Section 7). In total, over 8000 m of Ethernet cables, 300 m of optical fiber, 1901 unique AprilTags (26 dynamic), 136 Raspberry Pi nodes, and 46 network switches were employed to generate the test scenario.

11.2.2. RISE Swarm

The mission scenario and the venue selected aimed at necessitating a larger swarm from one field experiment to the next. Therefore, by FX-6, we not only required a large swarm capable of sustaining a 3.5 hour mission, but one with diverse capabilities to meet varying mission objectives. With this in mind, we brought a large heterogeneous swarm of 274 platforms as shown in Figure 25. This composition enabled us to support three operational levels: ground level with ATX platforms, low

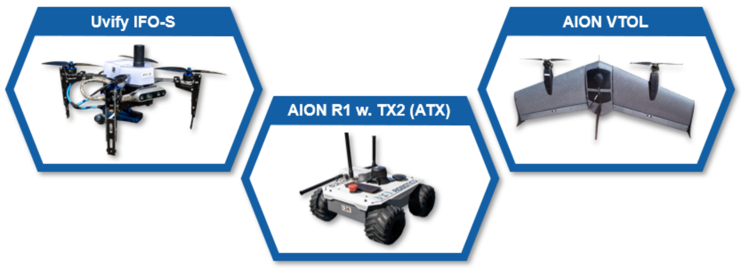


Figure 25. The 274 robotics platforms utilized in FX-6 from left to right: 158 Uvify IFO-S (IFO), 97 Aion R1 (ATX), and 19 Aion (AVT) vertical take-off and landing (VTOL) platforms.

to mid-altitude air with IFO platforms, and high altitude overhead with AVT platforms. Additional information about each platform type follows.

- **Ground**. Ground vehicles provide long mission endurance and are more power-efficient than their aerial counterparts. They can support more payloads and additional hardware without major impacts to their overall performance. They provide a ground-level operational advantage with decreased risk of damage during operations. They can better withstand collisions and/or harsher operations in comparison with aerial platforms. These platforms also have an additional sensor which is an omnidirectional range scanning lidar (RPLidar A2) which was used for mapping and localization. The lidar along with wheel encoders allow these platforms to also perform in GPS denied or restricted environments unlike the other platforms. However, they have limited traverse-ability and must avoid more obstacles than aerial vehicles, which increases the time it takes for them to reach their destination. We initially used Kobuki Turtlebots and custom-built skid-steer rovers. However, to support larger operations, we integrated the AION R1 rover with our system, where it became the primary ground asset. The R1 rovers use Jetson TX2 companion boards and are referred to as "ATX" (see Figure 25).
- **Quadcopter**. Quadcopters provide more targeted control of aerial vehicles and do not require lateral movement to maintain flight. They are highly maneuverable and can hover as needed, allowing them to operate in lower altitudes where there are more obstacles. Quadcopters are capable of traversing the environment in three dimensions, with an ability to go over as well as under various obstacles, and can navigate to goal positions faster than ground vehicles. However, there are limitations to the platform, such as the importance of its power:weight ratio to achieve and sustain lift. Flight time and efficiency are influenced by vehicle weight and payload. There is also an increased risk of damage, especially when flying in dense environments. We previously integrated custom-built platforms, as well as the Intel Aero. To support the OFFSET program and its scope, we integrated and heavily utilized the Uvify IFO-S platform, referred to as the "IFO" (see Figure 25).
- **Fixed Wing**. Additionally, we integrated the AION VTOL Tailsitter (AVT) shown in Figure 25. While this vehicle is more energy-efficient than quadcopters and thus may remain airborne for longer periods, it must also maintain a minimum airspeed. VTOL type fixed wing aircraft remove the need for launching mechanisms as well as long take-off runways, and offer the hovering capability of a quadcopter, but are more susceptible to wind. Utilization of non-VTOL fixed wing platforms with swarming has led to constraints on the number of platforms one can launch simultaneously (Chung et al., 2016). However, while VTOL platforms have limited rotational agility, requiring large open areas to turn, this is countered by their increased flight time as opposed to quadcopters. Therefore VTOLs are well suited for high altitude operations such as overhead surveillance.

Every platform further carried a Bluetooth emitter that was used to simulate a payload. Potential payloads were electronic warfare (EW), antipersonnel (AP), acoustic spoofing (AS), and covering fire

(CF). These payload types affected agent capabilities, what field node artifacts they could interact with, and impacted how they were deployed in a mission.

Given the mission duration, maintaining and updating a large swarm is an intensive effort. Aside from the logistics of maintaining the hardware and performance aspect of the agents, it also requires the ability to update all agents to the latest software before execution of their missions. RISE's architectural design allows for the utilization of Ansible to effectively and efficiently deploy software updates to the swarm. A variety of playbooks were developed for deployments based on update requirements. These updates can be as discrete as deploying an updated Docker container, robot extensions codebase, or other components of the platform codebase.

11.2.3. Command and Control

The swarm operator ran RISE C2 software on an Alienware laptop comprising a 2.20 GHz IntelCore™i7-8750H CPU with 32 GB of 2667 MHz DDR4 memory and an NVIDIA® GeFore® RTX 2080 mobile graphics processor. A mouse was used for all swarm command and control, while keyboard arrow keys were used to toggle between visualization modes. The swarm health engineer and swarm commander used a similar system, though the commander only used C2 for situational awareness. The commander, operator, and engineer operated from the roof of building 3b, shown at the south side (bottom) of Figure 26.

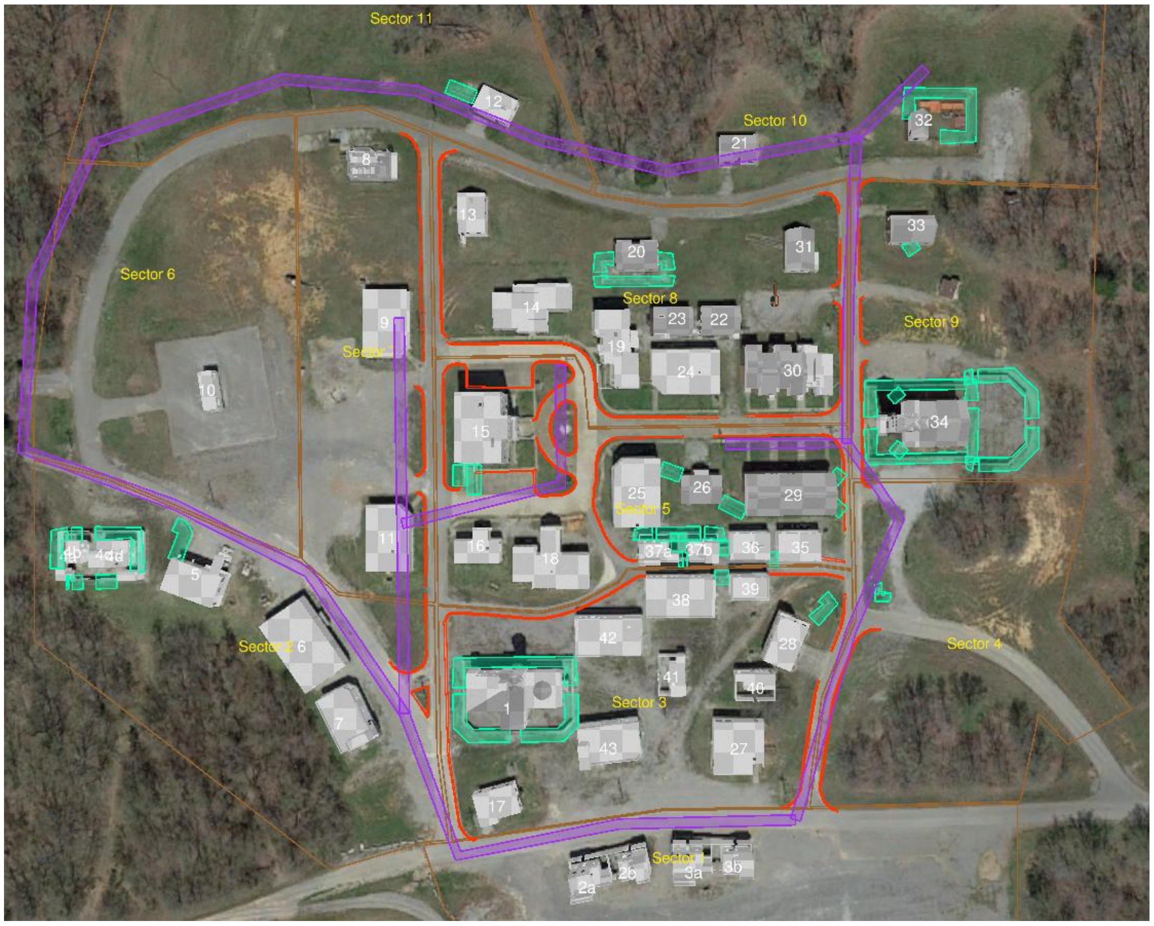


Figure 26. Fort Campbell Cassidy CACTF augmented with *a priori* site intelligence provided by NIWC, including curb (red), wall (green), and powerline (purple) obstacle data as well as sector boundary information. Buildings were modeled prior to FX-6 from floor plan data site visit information, whereas obstacle data was loaded at the start of each mission run. The map is oriented so that north is up.

11.3. Protocol

11.3.1. Scenario

Throughout the OFFSET program, NIWC developed mission scenarios focused on swarm operations in an urban environment that iteratively evolved since the first field experiment. The mission story typically centers around an oppositional force (OPFOR) having established a hold on an urban environment, with primary and secondary High Value Targets (HVTs) spread throughout and protected by various defensive measures. The objective is then for the blue force swarm (BLUFOR) to enter the environment, overcome the defensive measures, and secure the HVTs. For FX-6, the mission statement given in a scenario and experiment guide distributed by NIWC states:

> DARPA HQ has received intelligence pointing to the development of a weaponized contagion that is located in the buildings within the Fort Campbell Cassidy CACTF compound. Blue forces have established a foothold on the south end (Sector 1) of the CACTF. The primary assault will take place starting from the south end (Sector 1).

Further, the mission is divided into four phases as follows.

- Phase 0: Swarm Deployment and Intelligence, Surveillance and Reconnaissance (ISR).
- Phase 1: Combat Action and Isolation of Primary Target Objectives.
- Phase 2: Conduct an Urban Raid.
- Phase 3: Seize Key Urban Terrain.

The objective of phase 0 is to deploy the swarm, gather intelligence information about the environment, maintain surveillance in the area of operation, and maintain situational awareness. Localization and sensor errors prohibited our ground vehicles from autonomously moving with the precision necessary to enter buildings, and for this reason, we primarily participated only in phase 0, as later phases required breaching capabilities.

11.3.2. Mission Preparation

The BLUFOR team was given approximately one hour to prepare for each mission run. During this time, the field operations personnel situated the swarm in their initial deployment zones according to mission requirements. Since assaults came from sector one per the established scenario, IFO and ATX platforms were distributed along the south road, using all available space. For safety, AVT fixed wing platforms were set in an open area east of building 3b, away from most personnel. During this time, the swarm commander worked with the swarm operator to develop a course of action based on mission requirements and platform limitations. Once the plan was finalized and platforms placed, systems were powered on for a communications check. Network issues, when they occurred, were resolved and the swarm was powered off. Thereafter, the swarm commander briefed the experiment team, event organizers, and VIP guests on the mission plan and expected outcomes while the swarm operator separately input the final initial course of action in preparation for the start of the mission.

11.3.3. Mission Execution

After the mission briefing, field operations personnel returned to the deployment zone or took safety spotting positions throughout the CACTF. Those robots required to execute the initial COA were powered on, and once a sufficient percentage of the swarm was operational, the swarm commander began the assault. Throughout a mission run, the swarm commander would monitor the situation, react to intelligence information, and decide on subsequent courses of action, which the swarm operator effectuated through C2. The mission continued until the swarm commander called mission end or the allotted 3.5 hours were exhausted.

11.4. Results

With the field experiments scenario infrastructure in place, RISE was able to field its capabilities and technological innovations in a manner that provided insight into robotic swarm operations.

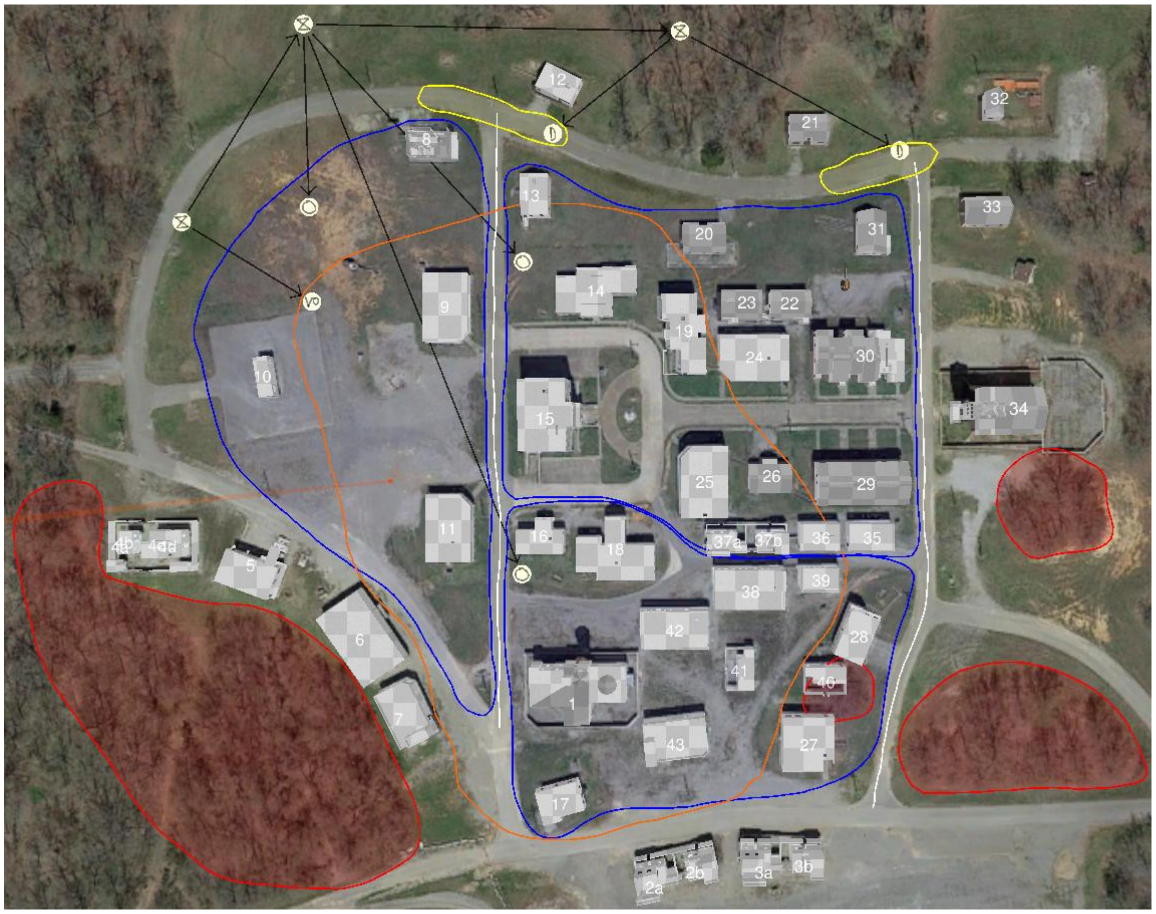


Figure 27. Example starting course of action used at FX-6 with obstacles (curbs, walls, and powerlines) removed for clarity. Details are discussed in Section 11.4.

11.4.1. Command and Control

Figure 27 illustrates one example course of action used at FX-6 used to start a mission run. Since the Cassidy CACTF possessed trees not included in our virtual environment representation, the swarm operator manually setup no-go zones to ensure PyC2's path planner routed around these obstacles. The swarm operator further used a tactic chain to sequence three events. First, a timer (far left hour glass symbol) with zero delay initiates a VTOL scan using three AVTs over the orange sketch area. Next, after a one-minute delay, a second timer initiates three overhead scans, one per blue explore area, which utilizes the IFO quadcopter assets. Finally, after another minute passes, the third timer initiates two deploy tactics intended to move ATX ground agents into the northern yellow deploy zones. Since each deploy zone is connected to a route (white south-to-north sketch lines), agents will traverse the specified path rather than use a possibly shorter path through the CACTF. A recovery point (orange point with beacon near building 11) was also set, identifying where IFO agents should fly to and land when their batteries are low. With this approach, the operator was able to deploy all available assets and achieve high coverage using only 14 tactic sketch parameters (tactical control measures) and 9 tactics.

As the mission progressed, the swarm operator was typically directed by the commander to continue scan operations, including overhead and building scans depending on the situation. Attempts were made to engage hostiles and breach buildings with ground agents, though localization and sensor errors often prohibited these advancements.

In total, 13 unique tactics were used throughout the entire event, two of which employed virtual hardware sensors. These tactics being operationalized with sketch-based tactic parameters provided enough flexibility to serve our needs given our platforms' autonomy limitations. With greater autonomy in obstacle avoidance and navigation, more tactics would have been required as we would have put the swarm to other uses, especially in coordinating assaults and securing buildings.

11.4.2. Operational Sustainment

At FX-6, RISE was able to maintain operations throughout the allotted 3.5 hours per trial run, with the ability to extend beyond that. Given the nature of field experimentation with low-cost robotics in a dense urban environment, attrition such as sensor and system malfunctions, abrupt degradation in flight conditions, prolonged platform life cycle, and other physical inhibitors is to be expected. Yet because of operational prudence and our approach to tactics design, the swarm of 274 assets brought to FX-6 were appropriately provisioned for the two-week event, and despite occasional vehicle collisions with the environment, onsite repairs and revivals of assets allowed for not only swarm sustainment but growth. With each exercise shift, RISE provisioned a greater percentage of its assets toward the mission run such that by the end of the event, RISE was able to utilize 174 assets simultaneously with additional assets on standby to ensure mission sustainment. A key question of the program was if a single operator could control 150+ drones, and this final result proved that it is possible.

11.4.3. Mission Coverage

Figure 28 shows the detections for approximately the first hour and a half of a run using 100 agents. Within approximately the first 20 minutes, the swarm was able to detect 600 artifacts and as the mission went on, even more were detected. Figure 29 shows the mission area coverage over the course of one run, where the top image illustrates the flight paths of a subset of ten swarm agents and the bottom image visualizes swarm activity across the environment. These results highlight not only the full area coverage provided by the swarm, but also that the swarm activities coincide with the

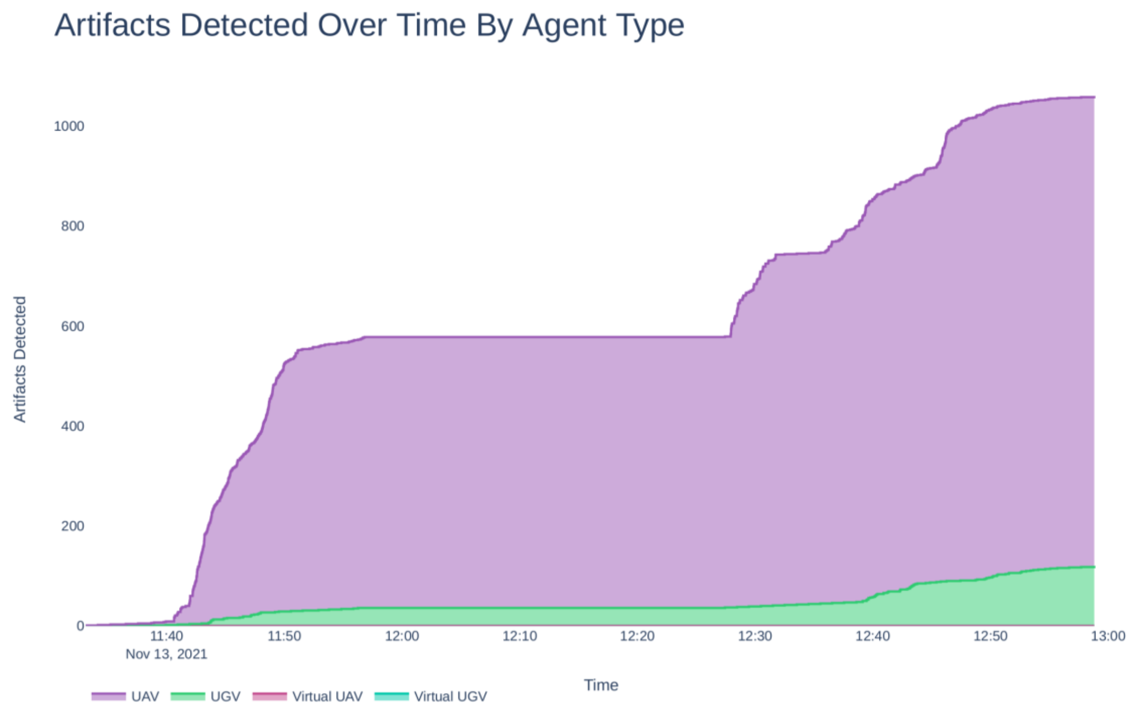
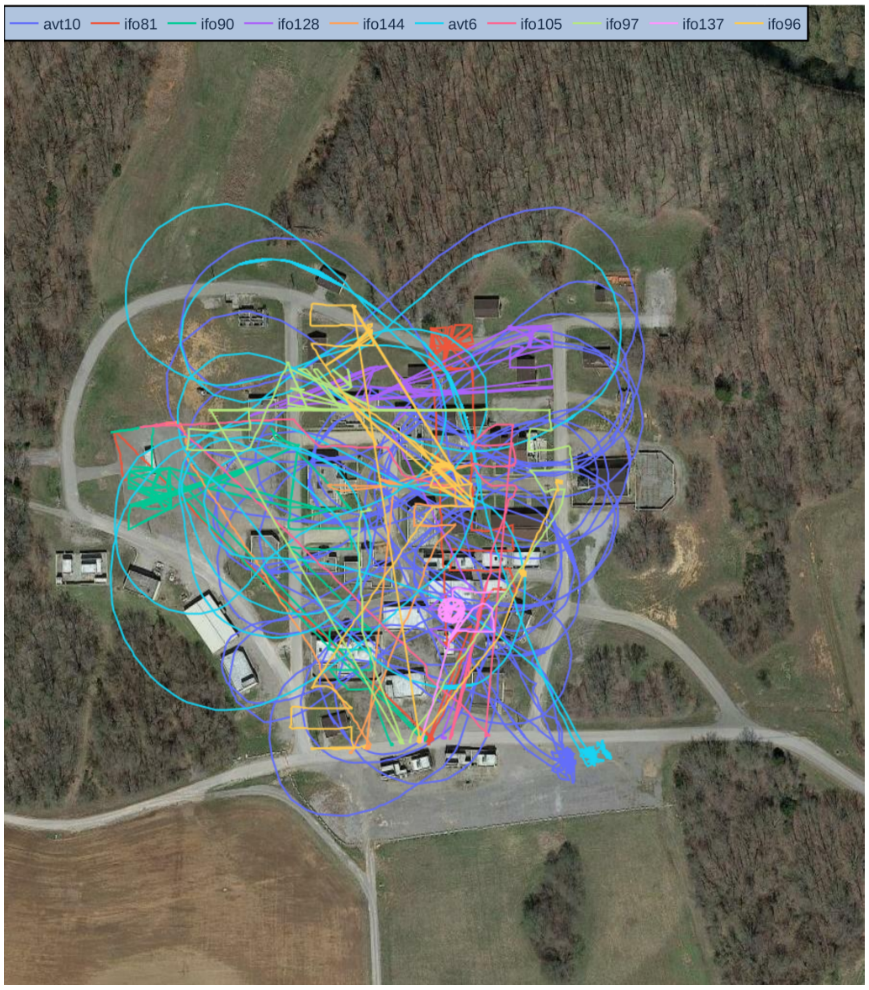


Figure 28. Unique artifact detection events recorded by NIWC's orchestration software over the first part of 100 agent run at FX-6.



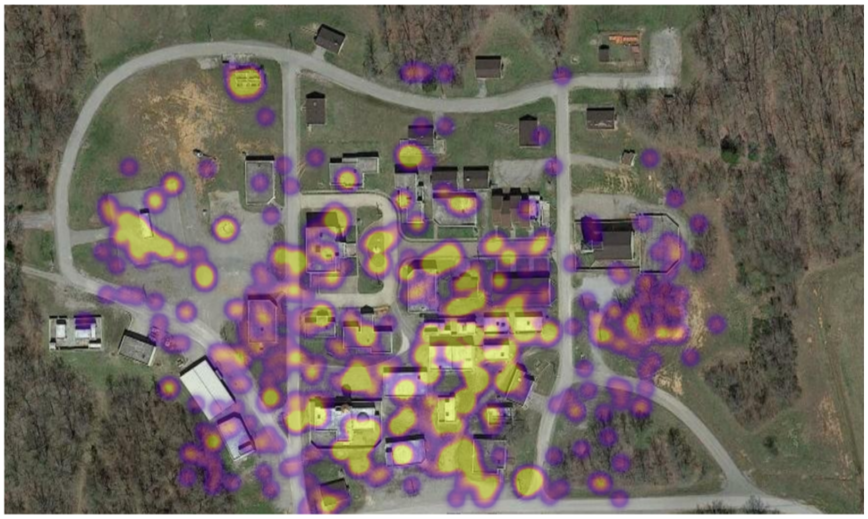


Figure 29. Agent trajectories (top) and coverage (bottom) recorded by NIWC's orchestration software over the course of run at Fort Campbell during FX-6.

building density of the CACTF. Though these charts and images are only a snapshot of a single run of the RISE swarm during FX-6, they are representative of the consistent ability of RISE to deploy its swarm to cover the entire operational area and perform its mission.

12. Discussion, Lessons Learned, and Future Work

With RISE we aimed to demonstrate that a system designed for rapid integration of new technologies and ideas can be used by a single operator to control a large heterogeneous swarm of autonomous agents. To this end, we designed an ecosystem informed by our experiences as an integrator who worked with third parties focused on key features rather than holistic systems. Distilled down to the set of requirements defined in Section 4.1, these experiences combined with OFFSET program requirements led us to develop a solution that separates concerns into four core modules—those pertaining to the user interface, tactics, behaviors, and algorithms. In particular, the separation of tactics into a module that facilities coordination between agents at a level of abstraction above the agent, in combination with a sketch-based interface where operations are described in a sketch language were key in enabling rapid integration and one-to-many swarm control. With PyC2, tactics development time dropped from weeks down to hours. PyC2 also enabled tactics developers to focus on how an operator will operationalize a tactic through the user interface (without being a UI developer), which can then be put into relation with how individual robots are coordinated to serve a specific tactical objective. Once all modules were in place, we were able to quickly write tactics that enabled an operator to input a single command that invoked numerous agents, and the operator did not need to concern their self with individual robot operations, owing in part to behavior level autonomy. As such, RISE successfully enabled a single operator to simultaneously control 174 platforms in the real world. This may effectively increase a Soldier's span of control in the future to 1 Soldier to 150+ platforms, in contrast to the current paradigm today of 1:1. Swarm control for operators is enabled through our C2 interface and tactics development tools.

In contrast to most prior work in swarming, RISE has been operated on hardware in real environments since the very beginning. We have fielded and tested at 5 military ranges, with more field work planned in the future. Field robotics at the scale of swarming brings many challenges, but swarm logistics is also a ripe area for new research.

In this discussion, we highlight the topics of Swarm Engine C2, tactics development, and field robotics, offering lessons learned and future work for each topic.

12.1. Swarm C2

We designed Swarm Engine C2 so that a single operator would be able to command a large scale swarm. We demonstrated through field experimentation that it was possible for an operator to sustain continuous operations over an extended period of time, while maintaining sufficient situation awareness. Difficulties that arose during testing were primarily due to hardware errors and insufficient automation. In these situations, the operator was unable to generate new swarm level work and agents were left idle. These circumstances were due to the nature of the program and experimentation, as they do not reflect normal operations as they would occur in practice. Robot swarm hardware improvements (rather than our RISE platform) in combination with new tactics already under development will overcome these issues, thereby allowing the operator to work as intended. Once these tactics are available, a full simulation-based user study will be warranted.

Our user interface was designed to be customizable so that operators could select memorable gestures based on their personal experiences and associations. We found, however, that our team used those gestures that were defined by the engineer who implemented the associated feature. Because it is known that individually defined developer gestures are less favorable than those proposed by large groups (Morris, Wobbrock, & Wilson, 2010), there were sometimes memorability issues. That is, as expected, individuals are able to learn our interface commands, but the gesture set was not optimal. And although customization enables rapid prototype in the short term, over a longer period of time, as swarm tactics mature and become standardized, we anticipate a need for periodic elicitation studies and agreement analysis (Vatavu & Wobbrock, 2015). As a result, swarm sketch grammars will become codified in the same way that mathematical, circuit diagram, and military iconography have all been standardized.

12.2. Tactics Development Through PyC2

PyC2 was a late development that came about during the second phase of the OFFSET program. Prior to PyC2, tactics were written partly in C2 and partly in robotics code, which proved to be cumbersome, time-consuming, and limiting. Thus PyC2 was born out of a need to accelerate tactics integration and enable nonroboticists with the ability to develop new tactics (see Requirement 4). Once in place, our team rapidly developed and iterated numerous tactics, including those that led to a substantial performance improvement between the third and fourth field experiments. Being able to write new tactics without having to modify C2 and with the aid of sketches and context proved to be a valuable tool. Tactics development only slowed down due to platform limitation issues and the need for our team to turn its attention toward other priorities. With PyC2, we also saw an increase in sprinter integrations, as outlined in Section 10.

One key issue we plan to address near term is that PyC2 is a fracture critical component. This means that if PyC2 fails or is unable to communicate with an agent, we are unable to command the swarm. Our solution is to enable PyC2 to run directly on robot platforms with an improved bidding protocol that better enables collaborative tasking. In this way, PyC2 tactics, integration, and development are unaffected, yet PyC2 becomes a distributed system.

12.3. Field Robotics

rise is developed with a bottom-up approach, so all overall tactic performance depends upon agent abilities to reliably complete tasks. When agents are unable to perform tasks reliably, this leads to a cascade problem of additional agents having to step in to take the tasks that failed or forces the swarm commander to re-think their approach to a mission. Most task failures were caused by individual agents' inability to accurately localize and navigate through the environment. Navigation and localization is an ongoing research area of robotics, especially in urban environments. Pushing forward in these areas will further increase the swarm's effectiveness and allow for even more complex agent behavior in the future.

With the scale of the rise swarm, logistics and overall platform maintenance (both hardware and software) is a large problem. For example, over time, damage occurs to agents' sensors or motors and requires troubleshooting and/or repair. From a software perspective, every platform is running a unique release of its given codebase as well as particular firmware for its flight controller. Both of these systems needed to be up-to-date on all platforms in use to ensure consistent swarm operations between platforms. Also, the platform's FCU (e.g., the Pixhawk) requires calibration, and though this is not necessarily an intensive process, it becomes a high level of effort when it comes to calibrating the numerous platforms in the swarm. Additionally, even though on paper all the platforms subgroups were identical, we noted a lack of consistency from platform to platform, which added complications to platform development. By utilizing Ansible and Docker, we were able to create manageable platform update procedures for agent codebases and could deploy to 50 assets in 5–15 min, depending on the amount of update required. Firmware updates and calibration were still an ongoing pain-point over the course of the program. Future work would potentially involve either more direct involvement in firmware codebases, removing our dependency on these subcomponents or working more closely with manufacturers to help manage these aspects.

By leveraging flexbe, we established a strong foundation given our system's bottom-up approach to tactic development. The reusability of state development and simple drag and drop primitive creation based on underlying algorithms or states allowed for rapid creation of new agent behaviors. In the end, however, it seems as though there has been a large push beyond hierarchical state machines and into behavior trees. While FLEXBE could theoretically support this capability, it does not currently exist within the system. As our system and architecture has evolved over the last 4 years, we discovered that FLEXBE and PyC2 have potential to be duplicative in the future. For example, it is still an open question as to whether certain logic flow should exist within the agent codebases in primitives or at the tactic level within PyC2.

Utilizing ROS was an absolute necessity when designing RISE, and it would be our team's choice again. We do however wish that the ROS 2 transition occurred earlier. We were often a little too early to adopt each new ROS 2 feature. We believe that as ROS 2 matures, the future decentralized concept of no ROS master will be a powerful tool in future swarm algorithm development and swarm development in general.

13. Conclusion

Our Rapid Integration Swarming Ecosystem (RISE) provides a platform for future multi-agent research and deployment. Using both physical and simulated swarms, we demonstrated RISE's rapid integration of third-party swarm tactics and behaviors. Our physical testbed, composed of more than 250 networked heterogeneous agents, has been extensively tested in mock warfare scenarios at five CACTF sites. With our live, virtual, constructive (LVC) simulation capabilities, RISE allows the use of both virtual and physical agents simultaneously. Other simulation advances such as our "fluid fidelity" concept and super real-time simulation enable rapid swarm tactic prototyping. Our gesture-based interface enables an operator ratio greater than 1:150, making RISE a massive force multiplier. Together, these feature allow RISE to translate mission needs to robot actuation and swarm operation with unprecedented ease and flexibility.

Acknowledgments

This work was supported in part by DARPA under Contract N66001-17-C-4066. We thank OFFSET program manager Dr. Timothy Chung for his expert oversight and insightful feedback over the program's four years as well as our collaborators at Northrop Grumman, University of Central Florida, and Intelligent Automation, Inc. We also want to thank the global ROS community and all open source robotic developers.

ORCID

Eugene M. Taranta II https://orcid.org/0000-0001-5615-2597
Adam Seiwert https://orcid.org/0009-0004-2293-5787
Anthony Goeckner https://orcid.org/0009-0001-0350-2546
Khiem Nguyen https://orcid.org/0009-0004-5113-7848
Erin Cherry https://orcid.org/0000-0001-9526-4356

References

Abukhalil, T., Patil, M., Patel, S., & Sobh, T. (2016, April). Coordinating a heterogeneous robot swarm using Robot Utility-based Task Assignment (RUTA). In *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)* (pp. 57–62). doi: 10.1109/AMC.2016.7496328

Acker, D., Hellhake, P. R., Jordan, W., & Parks, J. E. (2016). *System and method for multicast over highly mobile mesh networks.* (U.S. Patent 9319922B2)

Aguero, C., Koenig, N., Chen, I., Boyer, H., Peters, S., Hsu, J., … Pratt, G. (2015, April). Inside the virtual robotics challenge: Simulating real-time robotic disaster response. *Automation Science and Engineering, IEEE Transactions on*, *12*(2), 494-506. doi: 10.1109/TASE.2014.2368997

Albani, D., IJsselmuiden, J., Haken, R., & Trianni, V. (2017). Monitoring and mapping with robot swarms for agricultural applications. In *2017 14th ieee international conference on advanced video and signal based surveillance (avss)* (pp. 1–6).

Baumann, T. (2018). *Obstacle avoidance for drones using a 3dvfh* algorithm* (Unpublished master's thesis). Swiss Federal Institute of Technology Zurich.

Bernstein, D. J. (1991). *Djb2 hash function.* http://www.cse.yorku.ca/~oz/hash.html. (Last accessed 26 January 2022)

Brossard, M., Barrau, A., & Bonnabel, S. (2020). Ai-imu dead-reckoning. *IEEE Transactions on Intelligent Vehicles*, *5*(4), 585–595.

Calhoun, G., Ruff, H., Behymer, K., & Frost, E. (2018). Human-autonomy teaming interface design considerations for multi-unmanned vehicle control. *Theoretical issues in ergonomics science*, *19*(3), 321–352.

Caprari, G., & Siegwart, R. (2005). Mobile micro-robots ready to use: Alice. In *2005 ieee/rsj international conference on intelligent robots and systems* (p. 3295-3300). doi: 10.1109/IROS.2005.1545568

Chamanbaz, M., Mateo, D., Zoss, B. M., Tokić, G., Wilhelm, E., Bouffanais, R., & Yue, D. K. P. (2017). Swarm-Enabling Technology for Multi-Robot Systems. *Frontiers in Robotics and AI*, *4*.

Chen, J. Y., & Barnes, M. J. (2021). Human–robot interaction. *Handbook of human factors and ergonomics*, 1121–1142.

Chung, T. H., Clement, M. R., Day, M. A., Jones, K. D., Davis, D., & Jones, M. (2016, May). Live-fly, large-scale field experimentation for large numbers of fixed-wing UAVs. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1255–1262). doi: 10.1109/ICRA.2016.7487257

Cianci, C. M., Raemy, X., Pugh, J., & Martinoli, A. (2007). Communication in a swarm of miniature robots: The e-puck as an educational tool for swarm robotics. In E. Şahin, W. M. Spears, & A. F. T. Winfield (Eds.), *Swarm robotics* (pp. 103–115). Berlin, Heidelberg: Springer Berlin Heidelberg.

Clark, S., Usbeck, K., Diller, D., & Schantz, R. E. (2021, March). CCAST: A Framework and Practical Deployment of Heterogeneous Unmanned System Swarms. *GetMobile: Mobile Computing and Communications*, *24*(4), 17–26. doi: 10.1145/3457356.3457362

Crandall, J. W., Goodrich, M. A., Olsen, D. R., & Nielsen, C. W. (2005). Validating human-robot interaction schemes in multitasking environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *35*(4), 438–449.

Cummings, M. L., Nehme, C. E., Crandall, J., & Mitchell, P. (2007). Predicting operator capacity for supervisory control of multiple uavs. In *Innovations in intelligent machines-1* (pp. 11–37). Springer.

DARPA. (2017, February 15). *Broad agency announcement, offensive swarm enabled tactics (offset)* (No. HR001117S0011).

Dasgupta, P., Baca, J., Guruprasad, K. R., Muñoz-Melendez, A., & Jumadinova, J. (2015). The COMRADE System for Multirobot Autonomous Landmine Detection in Postconflict Regions. *Journal of Robotics*, *2015*. doi: 10.1155/2015/921370

Davis, D. T., Chung, T. H., Clement, M. R., & Day, M. A. (2018). Multi-swarm Infrastructure for Swarm Versus Swarm Experimentation. In R. Groß et al. (Eds.), *Distributed Autonomous Robotic Systems* (Vol. 6, pp. 649–663). Cham: Springer International Publishing. doi: 10.1007/978-3-319-73008-0_45

Deering, D. S. E. (1989, August). *Host extensions for IP multicasting* (No. 1112). RFC 1112. RFC Editor. Retrieved from https://www.rfc-editor.org/info/rfc1112 doi: 10.17487/RFC1112

Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., ... Vaussard, F. (2013, December). Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms. *IEEE Robotics & Automation Magazine*, *20*(4), 60–71. doi: 10.1109/MRA.2013.2252996

Dorigo, M., Theraulaz, G., & Trianni, V. (2021, July). Swarm Robotics: Past, Present, and Future [Point of View]. *Proceedings of the IEEE*, *109*(7), 1152–1165. doi: 10.1109/JPROC.2021.3072740

Drew, D. S. (2021). Multi-agent systems for search and rescue applications. *Current Robotics Reports*, *2*(2), 189–200.

Edwards, S. J. (2000). *Swarming on the battlefield: past, present, and future.* RAND NATIONAL DEFENSE RESEARCH INST SANTA MONICA CA.

Edwards, S. J. A. (2005). *Swarming and the future of warfare* (PhD Dissertation). Pardee RAND Graduate School, Santa Monica, CA.

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human factors*, *37*(1), 32–64.

Engebråten, S., Nummedal, O. R., Gilbreath, D., Yakimenko, O., & Glette, K. (2019). *Uav swarm with mesh radios: Development update* (Tech. Rep.). EasyChair. Retrieved from https://easychair.org/publications/preprint/XxRx

Fernandes, A., Couceiro, M. S., Portugal, D., Machado Santos, J., & Rocha, R. P. (2015). *Ad hoc* communication in teams of mobile robots using zigbee technology. *Computer Applications in Engineering Education*, *23*(5), 733-745. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.21646 doi: https://doi.org/10.1002/cae.21646

Furuhashi, S. (2021). *Messagepack.* https://msgpack.org/. (Last accessed 26 January 2022)

Giles, K., & Giammarco, K. (2017). Mission-based architecture for swarm composability (masc). *Procedia Computer Science*, *114*, 57–64.

Hamer, M., & Ortega-Sanchez, C. (2010). Simulation platform for the evaluation of robotic swarm algorithms. In *Tencon 2010-2010 ieee region 10 conference* (pp. 1583–1588).

Hammond, T. A., Logsdon, D., Paulson, B., Johnston, J., Peschel, J., Wolin, A., & Taele, P. (2010). A sketch recognition system for recognizing free-hand course of action diagrams. In *Twenty-second iaai conference.*

Harabor, D., & Grastien, A. (2011). Online graph pruning for pathfinding on grid maps. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 25).

Hocraffer, A., & Nam, C. S. (2017). A meta-analysis of human-system interfaces in unmanned aerial vehicle (uav) swarm management. *Applied ergonomics*, *58*, 66–80.

Hoebeke, J., Moerman, I., Dhoedt, B., & Demeester, P. (2004). An overview of mobile *ad hoc* networks: applications and challenges. *Journal-Communications Network*, *3*(3), 60–66.

How, J. P., Behihke, B., Frank, A., Dale, D., & Vian, J. (2008, April). Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, *28*(2), 51–64. doi: 10.1109/MCS.2007.914691

Internet Protocol (No. 791). (1981, September). RFC 791. RFC Editor. Retrieved from https://www.rfc-editor.org/info/rfc791 doi: 10.17487/RFC0791

Jevtić, A., Gazi, P., Andina, D., & Jamshidi, M. (2010). Building a swarm of robotic bees. In *2010 world automation congress* (p. 1-6).

Jiménez-González, A., Martinez-de Dios, J. R., & Ollero, A. (2013, December). Testbeds for ubiquitous robotics: A survey. *Robotics and Autonomous Systems*, *61*(12), 1487–1501. doi: 10.1016/j.robot.2013.07.006

Johnson, D., Stack, T., Fish, R., Flickinger, D. M., Stoller, L., Ricci, R., & Lepreau, J. (2006, April). Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications* (pp. 1–12). doi: 10.1109/INFOCOM.2006.182

Johnson, L., Ponda, S., Choi, H.-L., & How, J. (2010). Improving the efficiency of a decentralized tasking algorithm for uav teams with asynchronous communications. In *Aiaa guidance, navigation, and control conference* (p. 8421).

Kaiser, T. K., Begemann, M. J., Plattenteich, T., Schilling, L., Schildbach, G., & Hamann, H. (2022, May). ROS2SWARM - A ROS 2 Package for Swarm Robot Behaviors. In *2022 International Conference on Robotics and Automation (ICRA)* (pp. 6875–6881). doi: 10.1109/ICRA46639.2022.9812417

Kolling, A., Walker, P., Chakraborty, N., Sycara, K., & Lewis, M. (2015). Human interaction with robot swarms: A survey. *IEEE Transactions on Human-Machine Systems*, *46*(1), 9–26.

Konolige, K., Fox, D., Ortiz, C., Agno, A., Eriksen, M., Limketkai, B., ... Vincent, R. (2006). Centibots: Very large scale distributed robotic teams. In M. H. Ang & O. Khatib (Eds.), *Experimental robotics ix* (pp. 131–140). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kornienko, S., Kornienko, O., & Levi, P. (2005). Minimalistic approach towards communication and perception in microrobotic swarms. In *2005 ieee/rsj international conference on intelligent robots and systems* (p. 2228-2234). doi: 10.1109/IROS.2005.1545594

LeFavor, P. D. (2020). Us army small unit tactics handbook. In (chap. Tactics). Blacksmith LLC.

Lewis, M. (2013). Human interaction with multiple remote robots. *Reviews of Human Factors and Ergonomics*, *9*(1), 131–174.

Lewis, M., Wang, J., & Scerri, P. (2006). Teamwork coordination for realistically complex multi robot systems. In *Nato symposium on human factors of uninhabited military vehicles as force multipliers* (pp. 1–12).

McLurkin, J., Smith, J., Frankel, J., Sotkowitz, D., Blau, D., & Schmidt, B. (2006). Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In *Aaai spring symposium: to boldly go where no human-robot team has gone before* (pp. 72–75).

Mi, Z.-Q., & Yang, Y. (2013). Human-robot interaction in uvs swarming: a survey. *International Journal of Computer Science Issues (IJCSI)*, *10*(2 Part 1), 273.

Miller, C., Pelican, M., & Goldman, R. (2000). 'tasking'interfaces for flexible interaction with automation: Keeping the operator in control. In *Proceedings of the conference on human interaction with complex systems* (pp. 123–128).

Miller, I. D., Cladera, F., Cowley, A., Shivakumar, S. S., Lee, E. S., Jarin-Lipschitz, L., ... Kumar, V. (2020). Mine tunnel exploration using multiple quadrupedal robots. *IEEE Robotics and Automation Letters*, *5*(2), 2840-2847. doi: 10.1109/LRA.2020.2972872

Morris, M. R., Wobbrock, J. O., & Wilson, A. D. (2010). Understanding users' preferences for surface gestures. In *Proceedings of graphics interface 2010* (pp. 261–268).

Nacenta, M. A., Kamber, Y., Qiang, Y., & Kristensson, P. O. (2013). Memorability of pre-designed and user-defined gesture sets. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 1099–1108).

Object Management Group, I. (2015). *Data distribution service.* Retrieved from https://www.omg.org/spec/DDS/ (Last accessed 26 January 2022)

Oleynikova, H., Honegger, D., & Pollefeys, M. (2015). Reactive avoidance using embedded stereo vision for mav flight. In *2015 ieee international conference on robotics and automation (icra)* (p. 50-56). doi: 10.1109/ICRA.2015.7138979

Olsen, D. R., & Goodrich, M. A. (2003). Metrics for evaluating human-robot interactions. In *Proceedings of permis* (Vol. 2003, p. 4).

Ospina, N. I., Mojica-Nava, E., Jaimes, L. G., & Calderón, J. M. (2021, January). ARGroHBotS: An Affordable and Replicable Ground Homogeneous Robot Swarm Testbed. *IFAC-PapersOnLine*, *54*(13), 256–261. doi: 10.1016/j.ifacol.2021.10.455

Pickem, D., Glotfelter, P., Wang, L., Mote, M., Ames, A., Feron, E., & Egerstedt, M. (2017, May). The Robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1699–1706). doi: 10.1109/ICRA.2017.7989200

Pickem, D., Lee, M., & Egerstedt, M. (2015, May). The GRITSBot in its natural habitat - A multi-robot testbed. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4062–4067). doi: 10.1109/ICRA.2015.7139767

Preiss, J. A., Honig, W., Sukhatme, G. S., & Ayanian, N. (2017, May). Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3299–3304). doi: 10.1109/ICRA.2017.7989376

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Rizk, Y., Awad, M., & Tunstel, E. W. (2019, April). Cooperative Heterogeneous Multi-Robot Systems: A Survey. *ACM Computing Surveys*, *52*(2), 29:1–29:31. doi: 10.1145/3303848

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, *345*(6198), 795-799. Retrieved from https://www.science.org/doi/abs/10.1126/science.1254295 doi: 10.1126/science.1254295

Schillinger, P., Kohlbrecher, S., & von Stryk, O. (2016, May). Human-Robot Collaborative High-Level Control with an Application to Rescue Robotics. In *Ieee international conference on robotics and automation.* Stockholm, Sweden.

Schranz, M., Umlauft, M., Sende, M., & Elmenreich, W. (2020). Swarm Robotic Behaviors and Current Applications. *Frontiers in Robotics and AI*, *7*.

Self, M., Miller, B., & Dixon, D. (2005). *Acquisition level definitions and observables for human targets urban operations and the global war on terrorism* (Tech. Rep.). ARMY COMMUNICATIONS-ELECTRONICS COMMAND FORT BELVOIR VA NIGHT VISION AND ELECTRONICS SENSORS DIRECTORATE. Retrieved from https://apps.dtic.mil/sti/citations/ADA442798

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2017). *Airsim: High-fidelity visual and physical simulation for autonomous vehicles.*

Speakman, T., Crowcroft, J., Gemmell, J., Farinacci, D., Lin, S., Leshchiner, D., ... Vicisano, L. (2001, December). *PGM Reliable Transport Protocol Specification* (No. 3208). RFC 3208. RFC Editor. Retrieved from https://www.rfc-editor.org/info/rfc3208 doi: 10.17487/RFC3208

Stubbs, A., Vladimerou, V., Fulford, A., King, D., Strick, J., & Dullerud, G. (2006, June). Multivehicle systems control over networks: A hovercraft testbed for networked and decentralized control. *IEEE Control Systems Magazine*, *26*(3), 56–69. doi: 10.1109/MCS.2006.1636310

Taranta, E. M., II, Samiei, A., Maghoumi, M., Khaloo, P., Pittman, C. R., & LaViola, J. J., Jr. (2017). Jackknife: A reliable recognizer with few samples and many modalities. In *Proceedings of the 2017 chi conference on human factors in computing systems* (pp. 5850–5861).

United States Army. (2015). *Field manual fm 3-90-1 offense and defense volume 1 change 2 april 2015.* CreateSpace Independent Publishing Platform. Retrieved from https://books.google.com/books?id=wJJhrgEACAAJ

United States Army. (2020). *Field manual fm 1-02. 2 military symbols november 2020.* Independently Published. Retrieved from https://books.google.com/books?id=I4gKzgEACAAJ

User Datagram Protocol (No. 768). (1980, August). RFC 768. RFC Editor. Retrieved from https://www.rfc-editor.org/info/rfc768 doi: 10.17487/RFC0768

Van der Hoek, W., & Wooldridge, M. (2008). Multi-Agent Systems. In *Foundations of Artificial Intelligence* (Vol. 3, pp. 887–928). Elsevier. doi: 10.1016/S1574-6526(07)03024-6

Vatavu, R.-D., & Wobbrock, J. O. (2015). Formalizing agreement analysis for elicitation studies: new measures, significance test, and toolkit. In *Proceedings of the 33rd annual acm conference on human factors in computing systems* (pp. 1325–1334).

Villemure, É., Arsenault, P., Lessard, G., Constantin, T., Dubé, H., Gaulin, L.-D., ... Ferland, F. (2022, March). *SwarmUS: An open hardware and software on-board platform for swarm robotics development* (No. arXiv:2203.02643). arXiv. doi: 10.48550/arXiv.2203.02643

Wang, H., Lewis, M., Velagapudi, P., Scerri, P., & Sycara, K. (2009). How search and its subtasks scale in n robots. In *Proceedings of the 4th acm/ieee international conference on human robot interaction* (pp. 141–148).

Wang, J., & Olson, E. (2016, October). AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS).*

Williamson, B. M., Taranta II, E. M., Moolenaar, Y. M., & Laviola Jr., J. J. (2023). Command and control of a large scale swarm using natural human interfaces. *Field Robotics*, *2*, 1-22.

Woodall, W. (2014). *Ros on zeromq and friends.* https://design.ros2.org/articles/ros_with_zeromq.html. (Last accessed 26 January 2022)

Zahugi, E. M. H., Shabani, A. M., & Prasad, T. V. (2012). Libot: Design of a low cost mobile robot for outdoor swarm robotics. In *2012 ieee international conference on cyber technology in automation, control, and intelligent systems (cyber)* (p. 342-347). doi: 10.1109/CYBER.2012.6392577

ZeroMQ. (2021). *Zeromq: An open-source universal messaging library.* Retrieved from https://zeromq.org/ (Last accessed 26 January 2022)

How to cite this article: Taranta II, E. M., Seiwert, A., Goeckner, A., Nguyen, K., & Cherry, E. (2023). From warfighting needs to robot actuation: A complete rapid integration swarming solution. *Field Robotics, 3*, 460–515.

Appendix A. Full Tactic Example

Listing 3 presents the full Hold Position tactic. An operator uses this tactic to move a number of agents onto a sketched path surrounding an object or structure of interest, with all agents facing inward. Detailed comments are provided for comprehension.

```python
class HoldPosition(Tactic):

    # Tactic description data that is sent to C2 during registration.
    TacticDefinition = TacticDefinition(
        'Hold Position',
        'Move a set of agents to points along the perimiter and hold.',
        'square',
        context = SketchInputTypes.Path,
        parameters = [
            TacticParameter(
                'Altitude',
                'Height in meters that agent will assume.',
                float,
                6),
            TacticParameter(
                'Duration',
                'How long to hold in seconds.',
                float,
                10000),
            TacticParameter(
                'Agent Count',
                'Number of agents to place along perimeter.',
                int,
                8)],
        requires=[MoveTo])

    def __init__(self, task):
        """ """
        super().__init__(task)

        # Project onto XZ ground plane
        polyline = task.parameters.context.pts[:, (0,2)]
        count = task.parameters.agent_count
        positions = resample(polyline, int(count + 1))

        # Throw out last point because it duplicates first
        positions = positions[:-1]

        # Get inward heading at each point based
        # on winding of path
        vecs = get_inward_vectors(positions)
        headings_deg = vecs_to_headings_deg(vecs)

        # Create a task for each hold position
        for pt, heading_deg in zip(positions, headings_deg):
            pt = [pt[0], task.parameters.altitude, pt[1]]

            # Use robot MoveTo primitive to command agent.
            child = MoveTo(
                parent = self,
                poses_xyz = [pt],
                headings_deg = [heading_deg],
                durations_s = [task.parameters.duration],
                selected_agents = task.selected_agents)

            self.tasks += [child]

        # Send all children tasks to the task manager
        self.execute()
```

Listing 3. Full tactic implemented in PyC2.

Appendix B. Network Parameters

We changed several PGM parameters from their default values to allow for more efficient use of our limited network resources. First, we reduced the ambient SPM rate to $\frac{1}{30}$ Hz, and only transmit a single end-of-transmission "heartbeat" SPM 30 seconds after the last data transmission. Most importantly, we increased the NAK back-off interval to two seconds, the NAK repeat interval to 10 seconds, and reduced the allowed NAK retry attempts to 3. We further increased the NAK repair data interval to 800 ms and set a maximum repair data rate of 1 KB/s. Together, these adjustments prevent flooding and network breakdown in the case of serious communications disturbances, while still allowing for an acceptable measure of reliability.