

A Distributed Simulation-to-Flight Framework to Support Investigating Trust/Trustworthiness in Multi-Agent Systems

Benjamin N. Kelley* and Matthew P. Vaughan*

As autonomous systems continue to grow both in use and complexity, the necessity for robust and extensible simulation-to-flight methods is paramount for establishing an effective architecture for autonomous systems. A fundamental objective of the ATTRACTOR (Autonomy Teaming and TRAjectories for Complex Trusted Operational Reliability) project was to design and develop a distributed mixed-reality simulation environment to begin establishing a basis for certification of autonomous systems via research into trust and trustworthiness. In this paper, we present an autonomous systems architecture and development framework paired with a persistent distributed modeling and simulation environment for test and evaluation of autonomous systems. The Autonomous Entity Operations Network (AEON) framework enables autonomous system development with an easily extensible collection of libraries and plug-n-play nodes facilitated by the Data Distribution Service (DDS) communication protocol standard. The Baseline Environment for Autonomous Modeling (BEAM) simulation environment is a distributed mixed-reality Unity™-based environment built around the same DDS communication paradigm allowing for easy integration with AEON-based autonomous applications. They were designed under ATTRACTOR in order to measure and establish trustworthiness and trust in single- and multi-agent human-machine systems whether these machines are fixed-wing general aviation, rotary-wing Unmanned Aerial Vehicles (UAVs), ground rovers, or even spacecraft. Together AEON and BEAM enable sim-to-flight with minimal configuration changes. By using AEON and BEAM, source code that runs in simulation ports directly to hardware and has successfully flown in the lab and in the National Airspace System (NAS) at NASA LaRC many times over the lifetime of ATTRACTOR.

I. Nomenclature

<i>ATTRACTOR</i>	=	Autonomy Teaming and TRAjectories for Complex Trusted Operational Reliability
<i>AEON</i>	=	Autonomous Entity Operations Network
<i>BEAM</i>	=	Baseline Environment for Autonomous Modeling
<i>DDS</i>	=	Data Distribution Service
<i>UAS</i>	=	Unmanned Aerial System
<i>P2P</i>	=	Peer-to-peer
<i>Agent</i>	=	actor in the simulation environment

II. Introduction

AUTONOMOUS research is a multi-disciplinary field that is pushing the boundaries for many existing technologies and techniques, as well as a domain for discovering new approaches to the complex challenges that intelligent multi-agent systems pose. This convergence of disciplines leads to software complexity and tends to require rework to accommodate the needs of each specialty. Integrated system test flights are time-consuming and cost prohibitive during early system design and development. Simulation environments can be useful tools to accelerate algorithm development and testing. However, transitions from simulation to flight (sim-to-flight) can be challenging unless systems are designed with this transition in mind and with the necessary capabilities built into the architecture and framework. For these reasons, the need for a robust framework that can isolate and abstract these complex inter-disciplinary dependencies is clear. A fundamental building block of NASA's Autonomy Teaming and TRAjectories for Complex Trusted Operational

*Research Computer Scientist, NASA Langley Research Center, Hampton, VA, 23681, USA.

Reliability (ATTRACTOR) is the design and development of a distributed mixed-reality simulation environment to begin establishing a basis for certification of autonomous systems via research into trust and trustworthiness. ATTRACTOR's objective was to construct computational concepts of trustworthiness and justifiable trust in multi-agent autonomous teams, and to inform future certification of safety-critical and time-critical autonomous systems in aviation. Development of such an autonomous systems framework called the Autonomous Entity Operational Network (AEON) began under the Autonomy Incubator at NASA Langley Research Center [1], but like the ever evolving research in autonomous systems, AEON has undergone significant development efforts over its lifetime.

Throughout much of the initial development of AEON, physical flight tests were used routinely to test the components and overall autonomous system. While flight tests can be indispensable for validating and testing autonomous systems, they can also be burdensome and time-consuming endeavors. The development of a lightweight simulation environment that can easily integrate with AEON was a prominent goal during the ATTRACTOR project [2] to assist in the exploration of trust and trustworthiness in autonomous systems[3]. The same design criteria used for AEON were applied to the ATTRACTOR BEAM (Baseline Environment for Autonomous Modeling) simulation environment, and the resulting Unity™-based environment is a flexible, extensible, modular, and decentralized multi-player simulation that can seamlessly communicate with external algorithms and modules developed using the AEON framework. Much like the development of AEON, BEAM has evolved dramatically since its inception.

This paper presents a complete and current overview of the AEON framework as well as an updated description of BEAM's design and functionality. Lastly, this paper presents the synergy between AEON and BEAM. Together they provide a sim-to-flight framework that enables the developer to perform rapid prototyping, debugging, and testing of their autonomous system in simulation and the ability to transfer the system to physical hardware with minimal effort.

In following sections AEON and BEAM are discussed and shown how they can be utilized together to achieve these goals. Section III will discuss the details, design, and tools of the AEON framework. Section IV introduces the BEAM simulation environment. Section V will provide use cases and examples of how AEON and BEAM can be used together to develop and test autonomous systems. Section VI concludes the paper with lessons learned and future work.

III. Autonomous Entity Operational Network

The AEON framework is an extensible collection of libraries and plug-and-play nodes facilitated by the DDS[4] communication protocol standard. As new algorithms and hardware are developed and released, they can easily be integrated into an AEON node and thus abstracting the interfacing logic from the larger AEON network. This allows the developer to focus on the data relationship between the necessary nodes required by the autonomous system.

The flexibility and extensibility of a framework in the field of autonomous systems are key elements for successful collaboration and integration. Researchers are routinely collaborating and building upon the available research in the field, and the ability to easily adapt the build system, toolchains, and supporting libraries allows for smoother integration with external applications, frameworks, target platform architectures, or hardware. A successful framework should enable the developers to integrate the latest technologies with as much ease as possible with the ability to easily extend a framework with new algorithms, hardware platforms, and sensor payloads which are continually being developed and released at a rapid pace.

A decentralized communication paradigm was another important aspect identified early by the Autonomy Incubator. Having a dynamic and decentralized communication protocol is an enabling feature that allows the framework to be flexible, extensible, and modular while creating opportunities for robustness by not having a central point of failure in a centralized messaging broker.

Having a modular design allows for the creation of a suite of plug-and-play nodes. Using AEON's common set of communication data types over the decentralized communication channels enables the developer to decouple the data consumer from the data producer. This abstraction makes data consumers less dependent on the specific implementation details of the data producers and allows for easy swapping of modules. For example, navigation nodes can be developed to easily be sensor agnostic and support swapping various sensor interface nodes depending on the specific hardware payload of the autonomous platform.

A. Design Approach

1. Conceptual Organization

As stated above, the key aspects to AEON's design are its flexibility, its extensible structure, its decentralized communication protocol, and its plug-and-play modular node paradigm. As seen in Fig. 1, AEON's nodes are

conceptually separated into 3 layers: hardware interfacing nodes, behavior nodes, and mission management nodes.

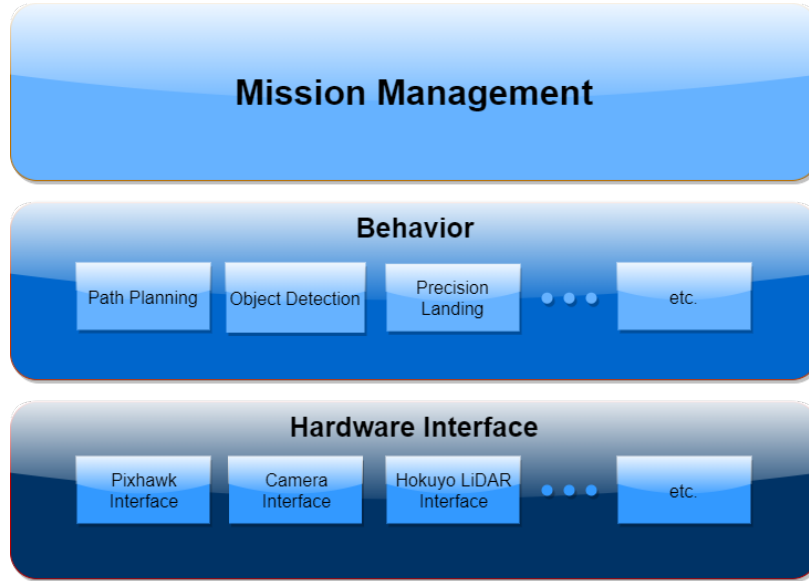


Fig. 1 Conceptual layers in AEON.

The hardware interface nodes handle the device specific serial communication with the physical hardware. These nodes may provide a simple one way communication, such as publishing a data stream from an Inertial Measurement Unit (IMU) or frames from a camera, or they may establish a bidirectional communication link with an autopilot device by publishing state information obtained from the autopilot and accepting commands that will be forwarded on to the autopilot device.

The AEON behavior nodes comprise a broad category that contains nodes that may implement completely autonomous behaviors, such as precision landing, or provide services helpful to other behavior nodes or mission management applications, such as path planning and object detection.

Mission management nodes control the configuration and execution of other AEON nodes to accomplish higher order tasks. The AEON framework provides tools and interfaces to develop these management nodes to be as simple or complex as the mission requires. They can employ simple state machines, hierarchical state machines, complex behavior trees, or any other number of decision making paradigms.

AEON nodes are compiled as self-contained standalone applications and inter-node communication is facilitated by the publish/subscribe DDS middleware. This modular node structure allows for decoupling of data producers from their consumers, thus giving the developer flexibility during development and at runtime to swap compatible modules out. For example, if a developer is working on an Extended Kalman filter (EKF) node for pose estimation, the IMU input can be easily swapped by using different AEON IMU interface nodes if the specific IMU hardware changes. These IMU nodes are responsible for the device specific serial communication with the IMU hardware and publish the data to consumers in the rest of the AEON network. This data decoupling is also beneficial during early system development when the developer may be unsure of the specific hardware going to be used on the final platform. A simple emulation node can quickly be developed and used only to be swapped out once the specific hardware is acquired and the interface node is developed.

This DDS-based modular design also facilitates the investigation of trust and trustworthiness by enabling the analysis of both individual components and their interactions. The DDS data abstraction results in discrete nodes that are easier to test and validate independently while the inter-module interactions can easily be observed and examined through external tools in real-time. The DDS publish/subscribe communication layer allows for custom developed or commercial DDS monitoring tools to subscribe to data topics of interest for real-time analysis or easy data logging for post flight examination.

2. AEON Libraries

As shown in Fig. 2, there are 10 major component libraries in the AEON framework, conceptually broken into 3 groups: the core AEON libraries, node interfaces, and utilities.

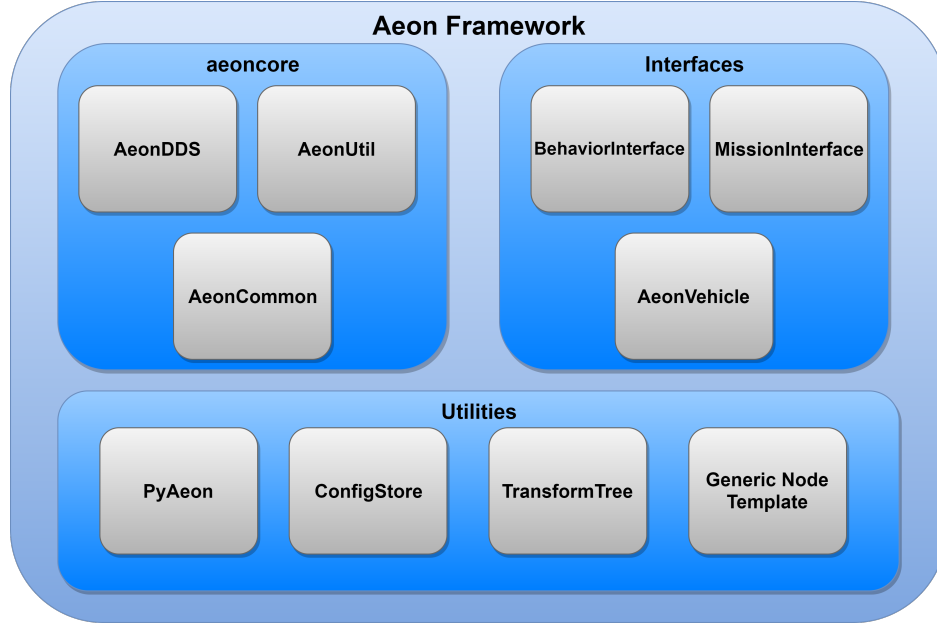


Fig. 2 Libraries in the AEON Framework.

The AeonDDS library contains convenient classes that abstract the most complex details of creating and establishing DDS connectivity between nodes. Because DDS is an open standard with a large selection of available vendors and implementations, this library also serves to isolate any vendor specific details from the AEON nodes. This allows for an easy place to develop extensions for different DDS vendors, while keeping the upstream AEON nodes unchanged. The *DDSManager* and *DDSEntityBase* classes contained in the library do exactly this. They offer a concise yet flexible API for managing the underlying DDS classes, either programmatically or via a runtime XML configuration file.

While DDS is a powerful tool that enables robust and configurable communication between AEON nodes, sometimes the publish/subscribe paradigm does not lend itself to the communication needs between certain nodes. To solve this, the AeonDDS library provides a convenient request/reply module. This module contains the *DDSRequester* and *DDSReplier* classes that still utilize the DDS publish/subscribe, but offers a flexible API for a more traditional request/reply communication paradigm. The module ensures requests are acknowledged, that their status can easily be queried by the requester and offers a flexible API for queuing or parallel processing of requests at the receiving node.

The last major component of the AeonDDS library is a collection of data structures for use both internally by nodes and for their DDS communication. The breadth of this collection is wide and many applications will find data types to fit their needs. It contains generic geometrical structures for computation or state estimation, vehicle commands, sensor data payloads, and many other data structures commonly used in autonomous system applications. While this library of data types tries to be as comprehensive as possible, unique needs do arise in some applications and the library's data types are extensible by AEON nodes.

AeonUtil is a collection of utility classes, interfaces and functions for a wide variety of use cases. Most notable are geometry and coordinate transformation functions and a behavior tree module. Manipulating geometrical data and moving between coordinate frames are such common tasks in autonomous system applications that these utility functions are indispensable to many AEON nodes. They help ensure consistency throughout the growing ecosystem of AEON nodes and support both Cartesian and non-Cartesian geodetic frames.

The behavior tree module offers an API for building and using complex behavior trees for nodes' internal decision making processes. A behavior tree is a model for constructing and executing reactive, composable tasks. The tree consists of leaf nodes representing either actions or Boolean conditions and internal nodes representing a parallel, sequential, or contingency control flow for running child nodes [5]. Behavior execution occurs by propagating a timing signal recursively from the root of the tree according to the control flow nodes, and each node reports back to its parent

a status of success, failure, or running.

AeonCommon is a collection of third-party libraries, that are either dependencies of AEON framework itself or numerous AEON nodes, that seemed prudent to manage and make available in one location. These libraries include the underlying DDS libraries, the Boost C++ libraries [6], Eigen [7], logging, and utility libraries. The primary reason AeonCommon is maintained is to standardize the external dependencies used by the framework core and the downstream nodes. Using the same version of these external libraries to build all AEON nodes helps ensure consistency and interoperability among them.

AeonVehicle is a collection of abstract classes to be used by AEON nodes that offer an interface to a vehicle or autopilot device. These base classes help ensure consistency among vehicle interface nodes. The common interfaces reinforce the modularity of the AEON network allowing behavior and mission management nodes to operate correctly with a wide variety of vehicle hardware.

The BehaviorInterface library implements an API for easily establishing client-server mechanisms that manage the execution of abstract BehaviorInterface instances through DDS request/reply interactions. The interface establishes and manages the request/reply DDS entities for the user while the developer simply extends the interface to provide their implementation of the request handling functions. The library allows the developer to focus less on how the behavior commands and data are shared and managed and more on their node's unique algorithms for processing the data to accomplish the request. The use of a standard behavior command interface also produces a more flexible network where behavior nodes can more easily be swapped out for a given deployment.

The MissionInterface library provides a common interface for implementing mission execution and management applications. Like the BehaviorInterface, the MissionInterface provides a standard DDS request/reply interface for mission management nodes. The developer only needs to provide implementations of request handler functions. By abstracting the underlying DDS request/reply communication the library enables the developer to focus on the management algorithms and to quickly prototype and develop complex mission management nodes, especially when used in conjunction with the other AEON utility libraries.

TransformTree provides a distributed utility for defining coordinate frames across the DDS network and transforming between them. The TransformTree library provides a wrapper around DDS end-points for communicating with other applications using the TransformTree library and an API to share coordinate frames and their relationship among other AEON nodes. The underlying DDS entities created and managed by the library ensure coordinate frame relationships are properly propagated through the network and an API is available to easily traverse the tree and transform coordinate information between disparate frames. The library supports Cartesian and non-Cartesian geodetic frames and, if provided reference points, can transform between them.

ConfigStore provides a networked configuration storage service over DDS. AEON nodes using the library can create configuration properties that are either public or private and mutable or immutable at runtime. The ConfigStore creates and manages DDS end-points for sharing and setting configuration values between nodes at runtime.

Although the AEON framework is designed with development ease in mind, C++ is not always the most welcoming language to researchers with little or no prior experience with it. Python has become an increasingly popular platform for autonomous systems and machine learning research and PyAeon seeks to connect AEON's high performance C++ nodes with Python applications. PyAeon is a Python module that offers a similarly structured interface to the DDS communication interfaces used throughout the AEON nodes. With PyAeon, researchers are able to easily integrate their Python-based algorithms and research code with the larger AEON node ecosystem.

B. Build, Utility, and Deployment Tools

Another important aspect of AEON's usefulness to the developer is the development life cycle and utility tools created along with it. This subsection will discuss four useful build, utility and deployment tools provided by the AEON framework.

1. Build System

AEON's default build system is a lightweight Python wrapper around a standard CMake project. The build system is intentionally lightweight to allow for maximum flexibility for each node's individual building requirements. This wrapper and underlying CMake project are sufficient as-is for the vast majority of nodes, but is easily extended and enables easy integration with external build pipelines. The Python build script is well organized into pre-build, build and post-build sections with examples and documentation on how to extend the script to incorporate the build systems of external dependencies or cross-compiling tool chains.

2. *GenericNodeTemplate*

The *GenericNodeTemplate* is a stub AEON node project that is fully functional with an example DDS reader and writer and integrated with the AEON libraries and build system. The template can easily be forked and quickly reworked for most new node development efforts and alleviates the tedious boilerplate configuration that comes with setting up a new development project. The developer can quickly replace the example DDS reader and writer with ones for the appropriate data types and topics for their new node and promptly begin focusing on main run loop and algorithms for the application.

3. *Process Manager*

While AEON's modular design offers many advantages in versatility, a deployed system may have a large number of node applications. Many of these applications may be inactive or unnecessary for a given task during a mission and this wasteful overhead may be unacceptable for the small single board computers or embedded systems that autonomous systems often rely upon. AEON offers a *Process Manager* node to help alleviate this overhead and to simplify node execution. The *Process Manager* uses the request/reply library for mission management nodes or user control station applications to interact with and safely start and stop AEON nodes as needed.

4. *Node Deployer*

AEON provides tools, not just for the development of autonomous system, but also for the configuration and deployment to remote computers that may be onboard the autonomous agents. The *Node Deployer* provides a utility for installing AEON libraries and node executables to both local and remote targets. This tool helps ensure the target machine's AEON software is installed properly, and it can install builds that are maintained by a continuous integration (CI) server, or locally built nodes that the user may wish to test before committing to the software repository. The *Node Deployer* is especially valuable when building large multi-agent autonomous systems or swarms with tens or hundreds of onboard computers to configure and ensure the proper software is installed.

C. Impact

The AEON framework has resulted in a development environment that enables rapid prototyping and exploration. The collection of standard data types and utility libraries allows for new research to quickly be integrated with the larger AEON node ecosystem. The modular nature also results in applications that are more focused and easier to test, verify, and maintain. By abstracting internal dependencies in inter-module connectivity, AEON reduces overall system complexity and promotes code reuse and portability. These aspects enable versatile system design, risk reduction, easy testing, and integration [8].

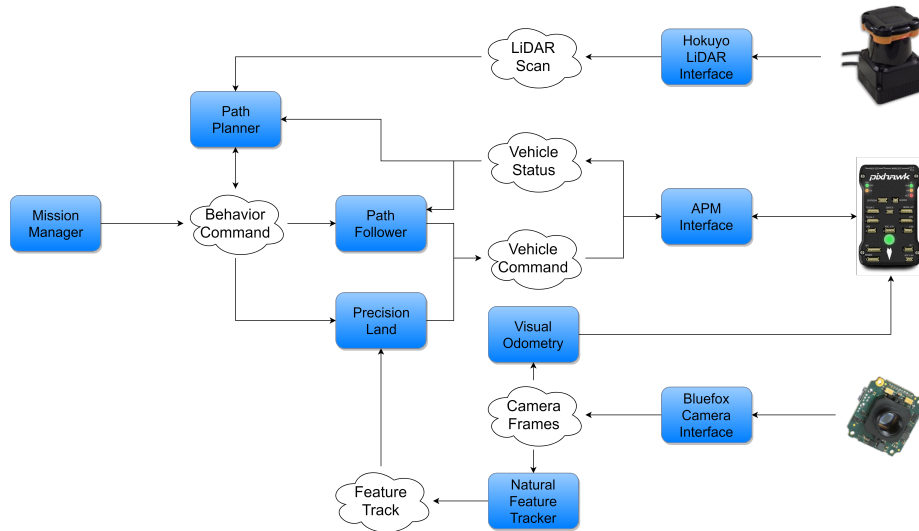


Fig. 3 Example deployment of an autonomous AEON system.

Numerous AEON network configurations have been flown, and Fig. 3 shows one such AEON-based autonomous system that was previously flown at NASA Langley. This deployment was used during an Autonomy Incubator flight demonstration for remote sensor package retrieval in a GPS-denied or degraded environment. The use of AEON during the development of this demonstration reduced complexity of individual components and allowed for risk reduction by allowing previously developed, tested, and trusted nodes to be used in the deployed system.

IV. Baseline Environment for Autonomous Modeling

During the initial development and use of AEON, the primary testing method was physical hardware flights using both custom and commercially available UAV platforms. While physical flights are an unmatched benchmark for testing the performance of an autonomous system, they are also demanding tasks that can negatively affect the development cycle. Physical flight tests require both extra personnel and hardware maintenance support with the need for safety pilots, safety observers, and vehicle technicians. The number of test flights that can be performed in a day is also limited by the battery life and recharge time of the target platform. One paramount goal when the ATTRACTOR project was established was the development of a lightweight simulation environment that integrated with AEON to help facilitate testing, validation, and exploration of autonomous systems research.

A. Design Approach

The design approach to BEAM reflects the core principles in AEON's design. By using gaming industry technologies, BEAM is designed to be a flexible, extensible, decentralized, multiplayer simulation that utilizes DDS as its communication middleware to enable easy integration with AEON based applications. While BEAM is not unique in building on a gaming engine [9][10], its incorporation of DDS and an extensibility pipeline has enabled a distributed simulation that not only blurs the line between the digital and physical worlds but can also stay up-to-date with new vehicles, sensors, and environments without recompiling. It utilizes many of the same data types and topics used throughout the AEON framework, so simulated vehicles and sensors can communicate to the same external AEON nodes that would interact on a physical system. The development of an extensibility pipeline is also an important design focus to allow the developer a method to extend the base functionality of the simulation environment with additional vehicle platforms, sensor payloads, environments, obstacles, user interfaces, etc.

1. *Unity™*

BEAM is built upon the Unity™ game engine. The decision to make use of a gaming engine was made because it offers an affordable development environment with an active community and cross-platform support. There already exist numerous high fidelity physics simulation tools and BEAM was not designed to compete with them. Instead, the use of a gaming engine offers lightweight physics simulation that can run on a wide variety of consumer hardware and offer adequate fidelity to allow the users to focus more on autonomous algorithm testing and development.

2. *DDS Based Multiplayer Simulation*

While Unity™ does offer an API for establishing a traditional client-server multiplayer game, BEAM instead utilizes DDS to facilitate dynamic discovery and peer-to-peer communication to offer a decentralized multiplayer simulation. Separate instances of the BEAM simulation will share simulated agent state information and world events. Simulated agents and sensors can only be owned by one BEAM instance and it is the responsible entity for performing the necessary physics calculations for those agents, responding to any agent commands, and publishing agent state information and sensor data during the simulation's update loop. Updates received for remotely owned agents are reflected in the simulation's respective game objects. Agent ownership can be exchanged between BEAM instances allowing agents to persist for longer durations even after the original owner may no longer be online.

The use of DDS as the communication layer also enables mixed-reality simulations with extensive interoperability with physical AEON-based platforms and simulated agents. AEON-based hardware platforms and simulated BEAM agents both communicate using the same DDS data types and topics allowing for interaction between the two. A physical asset's status information can be received by a BEAM simulation and represented in the digital environment as a remote agent. Likewise, simulated agents can communicate with external AEON-based nodes. In fact, due to the status and control message standardization between the two, behavior and mission management nodes that communicate with vehicle interfaces can seamlessly interact with both physical and simulated agents. This allows for complex mixed-reality flights where simulated agents can augment physical fleets to address safety or hardware limitations. For example, a

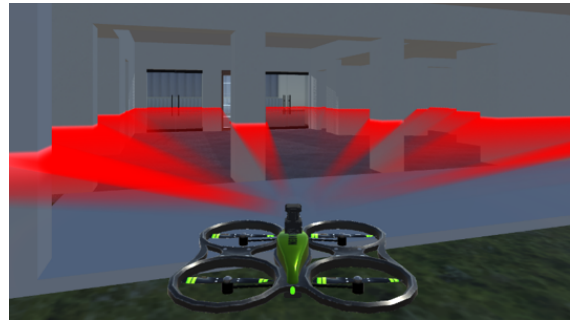
small fleet of five UAS can be transformed into a swarm of tens or hundreds with the addition of simulated agents to test the scalability and robustness of the multi-agent system.

3. Extensibility

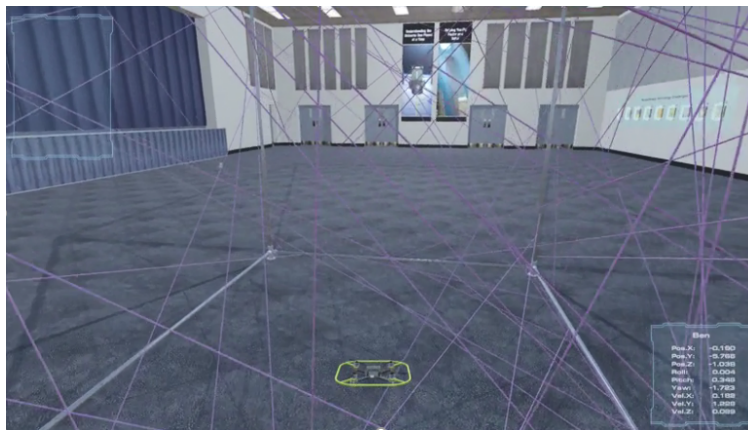
A successful simulation environment for autonomous systems research needs to be easily extensible, as it would be impossible to adequately predict the unique vehicle hardware, sensors, environments, obstacles, or user interface elements the researchers may desire to simulate. While BEAM offers a built-in level of core functionality, it also provides utility libraries and tools to help the developer provide runtime modifications to the BEAM simulation. This *Mod* pipeline allows the developer to add a wide range of additional content to the simulation as seen in Fig 4.



(a) Agent platforms added to BEAM via mods.



(b) A LiDAR sensor added to BEAM through a mod.



(c) A random wire maze environmental obstacle added to BEAM by a mod.

Fig. 4 Extensions to BEAM provided through mods.

Mods consist of a collection of Unity™ asset bundles and standalone C# dynamically-linked libraries (DLL) so that they can be loaded at runtime by BEAM's *ModManager*. BEAM's utility libraries offer APIs for ensuring the mod is registered and instantiated properly. The APIs and tools provided to the mod developer are discussed in further detail in the following subsections.

B. Build, Utility, and Deployment Tools

1. AeonUnity

AeonUnity is a C# DLL that contains the utility functions, DDS framework, and management infrastructure for most of BEAM's core functionality. It is developed and maintained as a standalone DLL, as opposed to being compiled into the Unity™ project directly, to facilitate the extensibility pipeline where mods need access to the modding framework classes in the library. A brief overview of the eclectic contents of the AeonUnity library can be seen in Fig. 5.

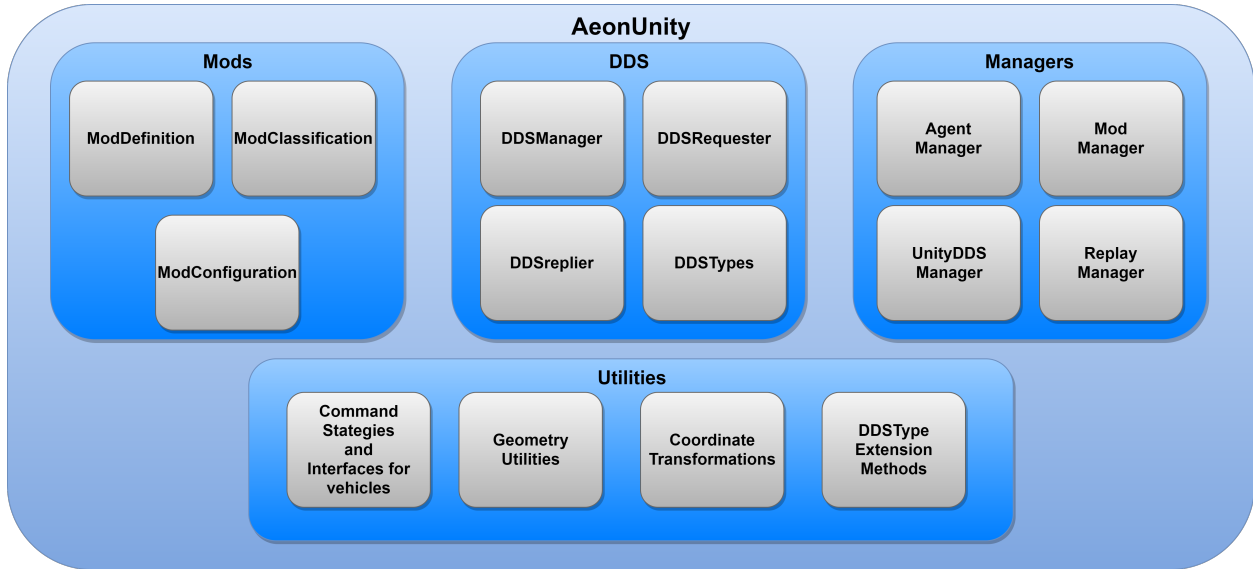


Fig. 5 Contents of the AeonUnity library.

2. *GenericModTemplate*

Similar to AEON’s *GenericNodeTemplate*, BEAM offers a complete template project for the creation of a BEAM extension mod. This template project comes completely configured with the two necessary subprojects for constructing a BEAM mod: a Unity™ project and a standalone C# DLL project.

The Unity™ development subproject is a standard Unity™ development environment that the user can use to develop and test their Unity assets. These asset prefabs must be compiled into Unity™ asset bundles for deployment to the BEAM software so that they can be dynamically loaded and referenced at runtime.

The C# DLL subproject is a standard C# library project preconfigured to link with AeonUnity and the UnityEngine libraries, and it is where all the mod’s source code must be compiled. Because BEAM mods are loaded dynamically at runtime, all necessary Unity™scripts must be compiled into this library. The developer uses the library to provide their implementation of one or more *ModDefinition* classes, any *ModConfiguration* classes, and other Unity™*MonoBehaviour* scripts necessary for the mod’s functionality.

Lastly, due to the complexity of managing two subprojects for the construction of BEAM mods, the Generic Mod Template provides Python utility scripts for a variety of tasks. These utility scripts help ensure the subprojects stay

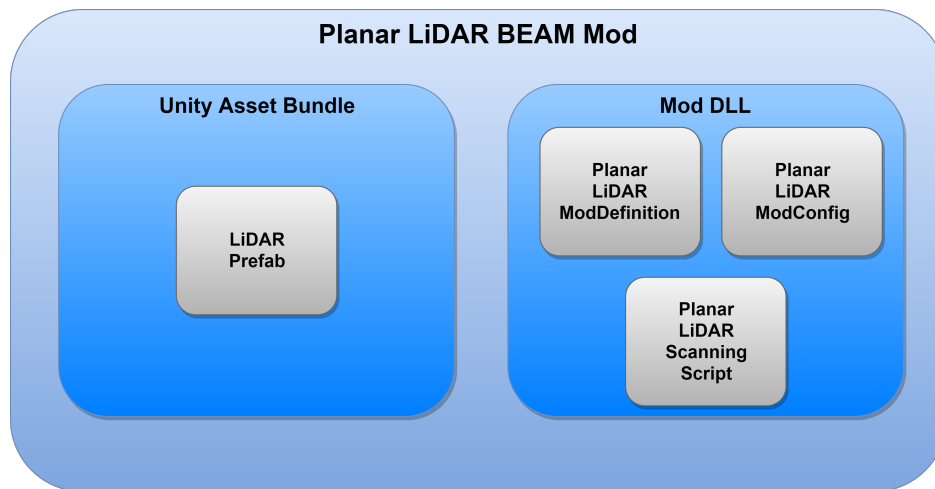


Fig. 6 Example structure of a planar LiDAR BEAM mod.

synchronized and assist in the building, packaging, and installation of the mod. Fig. 6 shows an example structure of a planar scanning Light Detection And Ranging (LiDAR) sensor mod built using the Generic Mod Template.

C. Impact

BEAM is a lightweight, general purpose, multiplayer unmanned system simulator. Its real strength and impact lies in its DDS-based interface for external system integration and the tools it provides to be easily extended. While BEAM was initially developed for interconnectivity with AEON, it is fully capable of integrating with other DDS-based applications and development frameworks with minimal effort. For example, as the Robot Operating System (ROS) moves towards utilizing DDS as its underlying communication protocol with the development of ROS2, BEAM can also easily be used as a simulation environment for the development and testing of ROS2 based applications.

The incorporation of BEAM into the development cycle of AEON nodes provides both risk reduction and time saving opportunities, while encouraging iterative system integration by removing the complexities that come with physical flight tests. The combination of AEON and BEAM creates a development environment that facilitates and encourages rapid prototyping and integration. This flexible development cycle is discussed through example in Section V.

V. Leveraging BEAM and AEON for Rapid Prototyping, Integration, and Testing

The previous sections discussed details on both AEON and BEAM's individual designs and capabilities. This section focuses on two examples that demonstrate how AEON and BEAM can be used together to facilitate an agile development cycle. The first one addresses the design, evaluation, and integration of a path-following algorithm. The second demonstrates how BEAM can be used to help with the analysis and evaluation of hardware in the loop testing.

In the first example, a researcher was developing and tuning a new path-following algorithm. By developing the algorithm into an AEON-based node, the developer was able to easily integrate this new path-following behavior with the necessary vehicle interface topics, sensor data topics, and mission management nodes. Because BEAM uses the same DDS interfaces for agent interaction and sensor data as AEON, the developer was also able to leverage BEAM as an early testing tool as shown in Fig. 7. Indeed, Fig. 7a shows a virtual laboratory environment with a UAS equipped with a LiDAR sensor. The vehicle is capable of tracking a safe path designed to explore the environment, and the path-following algorithm aforementioned is being evaluated. Figure 7b depicts the set of nodes that make this mission possible. Note that due to AEON and BEAM's distributed design, the Path Controller and Mission Manager nodes can be launched manually or by the ProcessManager and be executed on either the same workstation running BEAM, a remote workstation, or even the onboard computer to be used during physical flights as long as network connectivity is available.

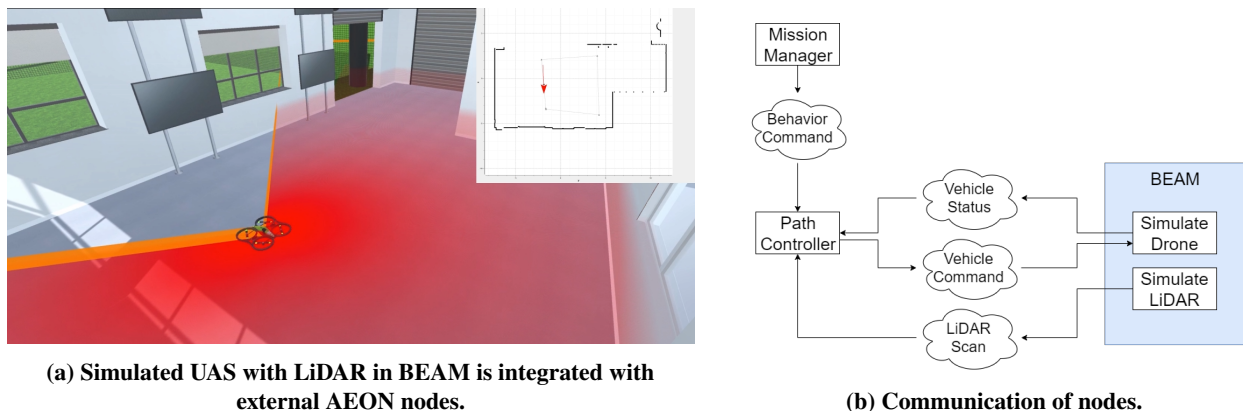


Fig. 7 Using BEAM in the AEON node development cycle.

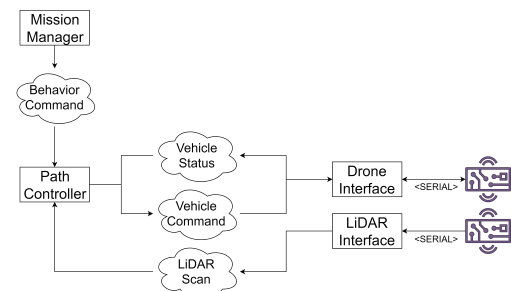
Due to AEON's modular design, the developers are able to layer the complexity in the development cycle. In fact, the developer that worked in the example depicted Fig. 7 benefited from the AEON-BEAM ensemble in three ways. First, establishing communication with real vehicles and BEAM-simulated agents was standardized through the Vehicle Status and Vehicle Command topics. Second, the expansion of the available nodes to exchange new data types, such as LiDAR information, is standardized and facilitated by the modularity of this architecture. Last, the developer used the

BehaviorInterface library to quickly integrate with AEON's existing mission management nodes.

Using BEAM, the developer was able to perform hundreds of simulated flights from their development workstation in a matter of a few days as they iteratively developed their path-following node with more complexity. Figure 8 shows that the software developed through this workstation testing was easily transitioned to performing real flight tests with no significant changes.



(a) UAV indoor flight using same AEON node as Fig. 7.



(b) Communication of nodes and hardware.

Fig. 8 Indoor flight of UAV using the same code tested using BEAM.

The second use case demonstrates how BEAM can be used as a mixed-reality tool for visualizing and testing complex distributed autonomous systems. Under ATTRACTOR, a researcher was developing a potential field-based distributed area coverage algorithm for use in search and rescue contexts[11]. The algorithm was being developed in Python, but by using the PyAeon library, it was quickly developed into a functional AEON node capable of connectivity with BEAM and other AEON nodes.

BEAM was used as a mixed-reality visualization and analysis tool during physical hardware testing of the potential field algorithm with a fleet of small ground-based agents. As shown in Fig. 9, the algorithm receives state information from a collection of ground rovers, which it uses to produce vehicle commands to have the agents continually cover the search area as much as possible. Status messages for the vehicle and potential field's control points can be received by BEAM and reflected in real-time as remote agents in the simulation environment. This can be seen in Fig. 10.

VI. Conclusion

AEON and BEAM have proven to be versatile tools that together have created an agile development cycle. The modular design promotes isolation and abstraction of inter-module dependencies. These qualities facilitate the creation of nodes that are easier to interchange, simpler to test and validate, and less coupled with constituent modules, resulting in reduced complexity of individual applications and the overall system. AEON and BEAM's extensibility has also enabled quick integration with new algorithms, hardware, and external frameworks.

While AEON and BEAM were initially conceived to assist research within the aeronautics domain, their adaptability is enabling ongoing and future research to support in-space autonomous systems technologies. For example, Fig. 11 shows an ongoing effort leveraging the versatility of BEAM and AEON that supports in-space assembly research [12][13]. The combined software ecosystem highlights the capability to rapidly prototype and test new algorithms and facilitate research across multiple domains.

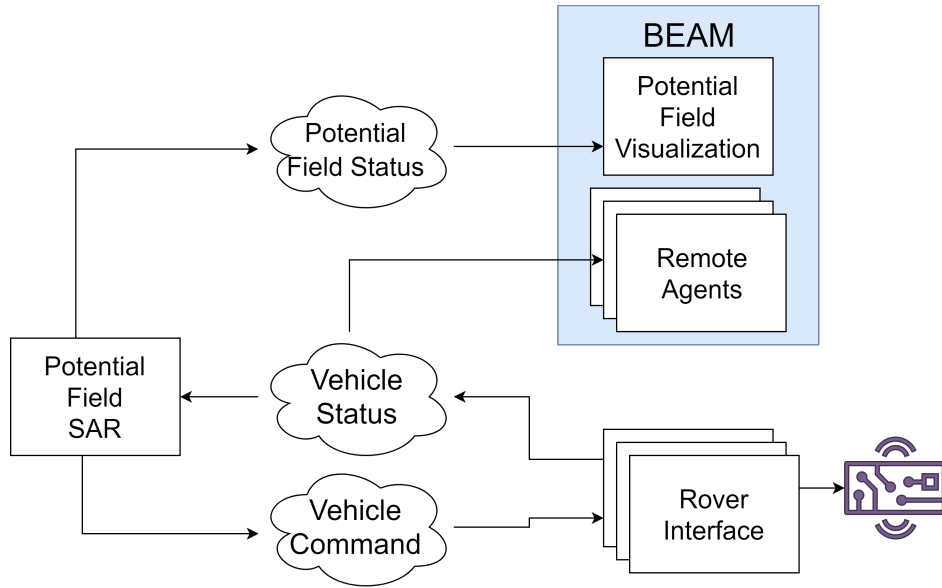


Fig. 9 Communication of nodes used to test the potential field algorithm.

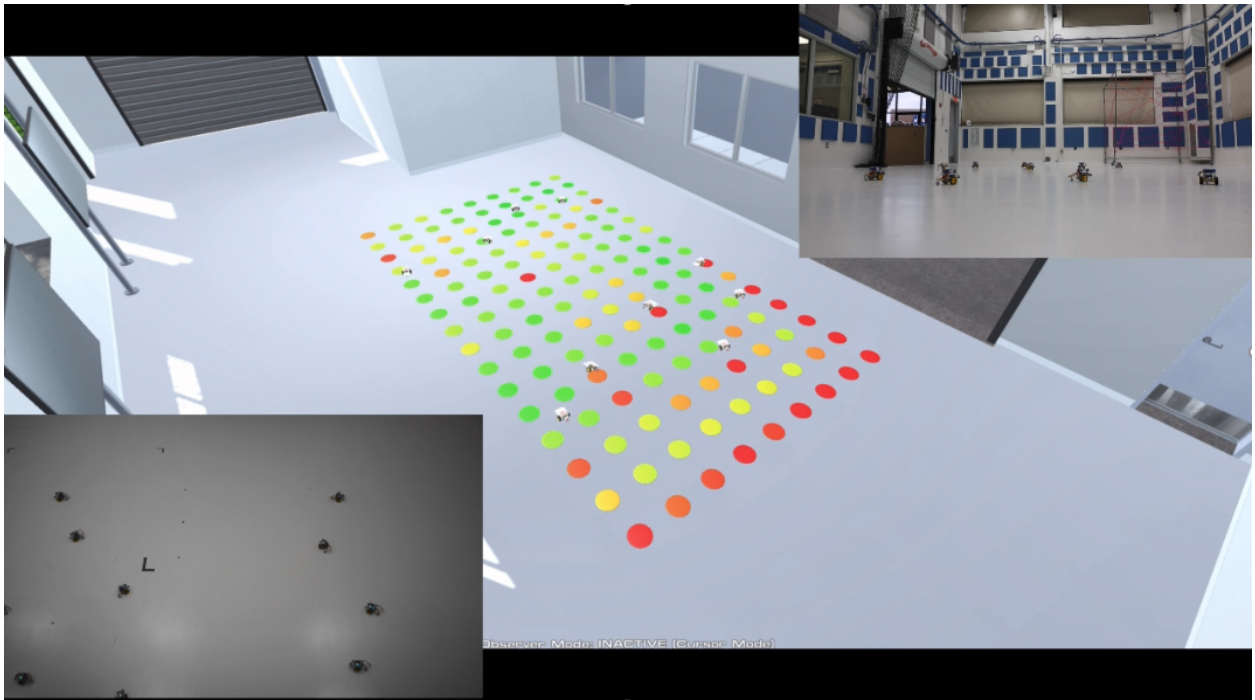


Fig. 10 Potential field algorithm controlling several physical ground vehicles. Their status and the status of the potential field control points are reflected in BEAM in real-time.

Acknowledgments

The authors would like to thank the entire ATTRACTOR team, led by Dr. B. Danette Allen and Dr. Natalia Alexandrov, and the Convergent Aeronautics Solutions (CAS) Project of the NASA ARMD's Transformative Aeronautics Concepts Program (TACP) for supporting this work via ATTRACTOR.

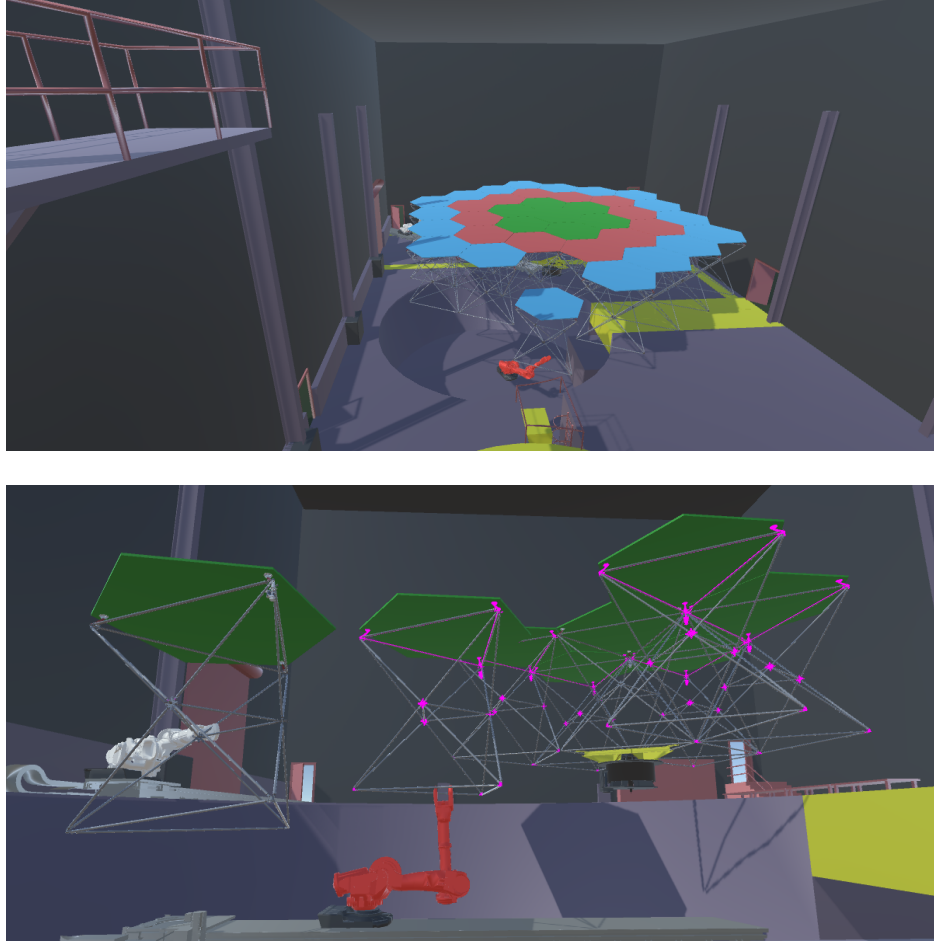


Fig. 11 BEAM and AEON extended to support in-space assembly research.

References

- [1] Cross, C. D., Motter, M. A., Neilan, J. H., Qualls, G. D., Rothhaar, P. M., Tran, L., Trujillo, A. C., and Allen, B. D., "Towards an Open, Distributed Software Architecture for UXS Operations," 2015.
- [2] Kelley, B. N., Williams, R. A., Holland, J. L., Schnarr, O. C., and Allen, B. D., "A persistent simulation environment for autonomous systems," *2018 Aviation Technology, Integration, and Operations Conference*, 2018, p. 4015.
- [3] Allen, B. D., "Real-World and Mixed-Reality Applications of Trustworthiness and Trust in Autonomous Cyber-Physical-Human Systems," 2021.
- [4] Pardo-Castellote, G., "Omg data-distribution service: Architectural overview," *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, IEEE, 2003, pp. 200–206.
- [5] Colledanchise, M., and Ögren, P., *Behavior trees in robotics and AI: An introduction*, CRC Press, 2018.
- [6] "Boost C++ Libraries," <http://www.boost.org/>, 2020.
- [7] Guennebaud, G., Jacob, B., et al., "Eigen v3," <http://eigen.tuxfamily.org>, 2010.
- [8] Vaughan, M. P., and Kelley, B. N., "Bootstrapping Multi-Agent Unmanned Aerial Vehicle (UAV) System Integration Using Ground-Based Assets: Lessons Learned," 2021.
- [9] Shah, S., Dey, D., Lovett, C., and Kapoor, A., "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," *Field and service robotics*, Springer, 2018, pp. 621–635.

- [10] Meng, W., Hu, Y., Lin, J., Lin, F., and Teo, R., "ROS+ unity: An efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments," *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2015, pp. 002562–002567.
- [11] Cooper, J. R., "Optimal Multi-Agent Search and Rescue Using Potential Field Theory," *AIAA Scitech 2020 Forum*, 2020, p. 0879.
- [12] Belvin, W. K., Doggett, W. R., Watson, J. J., Dorsey, J. T., Warren, J. E., Jones, T. C., Komendera, E. E., Mann, T., and Bowman, L. M., "In-space structural assembly: Applications and technology," *3rd AIAA Spacecraft Structures Conference*, 2016, p. 2163.
- [13] Doggett, W., Dorsey, J., Jones, T., Mikulas, M., Teter, J., and Paddock, D., "TriTruss: A New and Novel Structural Concept Enabling Modular Space Telescopes and Space Platforms," 2019.