AIAA Aviation Forum and ASCEND co-located Conference Proceedings
21 - 25 July 2025, Las Vegas, Nevada
AIAA AVIATION FORUM AND ASCEND 2025

10.2514/6.2025-3804

Simulation for Multi-Agent Autonomy: LTA UAVs in Adversarial Environments

Andrew Meighan[*], Caeden Taylor[†], Matthew Dempsey[‡], and Or D. Dantsker[§]
Indiana University, Bloomington, IN 47408

In this paper we present a simulation focused on multi-agent autonomy for lighter-than-air unmanned aerial vehicles in adversarial environments. While autonomous agents have seen increased use in performing real-world tasks, the simulation software for testing these agents has lagged behind. One major difficulty in testing algorithms for multi-agent autonomy is inability to bring software and hardware directly from simulation to the real world. Realistic environments, physical simulators, hardware and software integration, and communication networks are a few of the many requirements for this type of simulation. Originally inspired by the collegiate competition Defend the Republic, this simulation provides users with an interface between Unreal Engine 5 and Python via a TCP connection that allows for software-in-the-loop testing. Hardware-in-the-loop testing will be provided in future updates. Unreal provides photorealistic graphics, in-built physics, and simulated sensors. Data from the environment is sent to Python for processing and control logic and then returned to Unreal so the agent positions can be updated. The developed architecture is intentionally mirrored with the real-world systems used in competition by Indiana University to facilitate transfer between simulation and real-world.

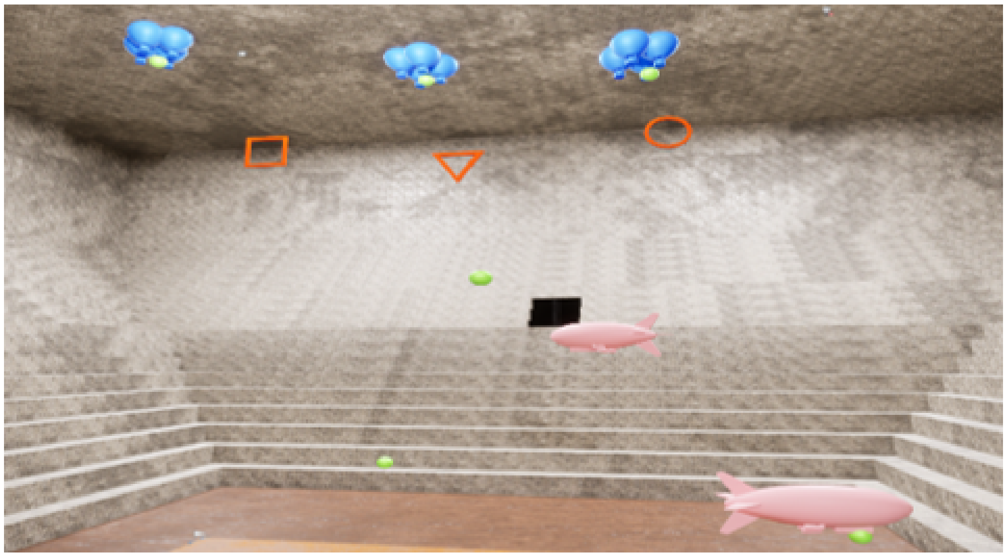
I. Introduction

As the field of robotics continues to evolve alongside recent advancements in machine learning, autonomous systems become more and more relied upon to solve real-world problems. The increased presence of these systems makes the development and testing of software and hardware prior to deployment all the more important. Highly realistic simulations that allow integration of hardware and software are one such solution to this testing problem. Simulations like this are critical due to the inherent difficulty of training and testing algorithms in the real-world. The inherent unpredictability of robotic actors causes safety risks and requires ample facilities, hardware, and personnel, which all contribute to the complexity and cost. When training models that require large amounts of data, the problem becomes even more unrealistic to solve with only real-world data.

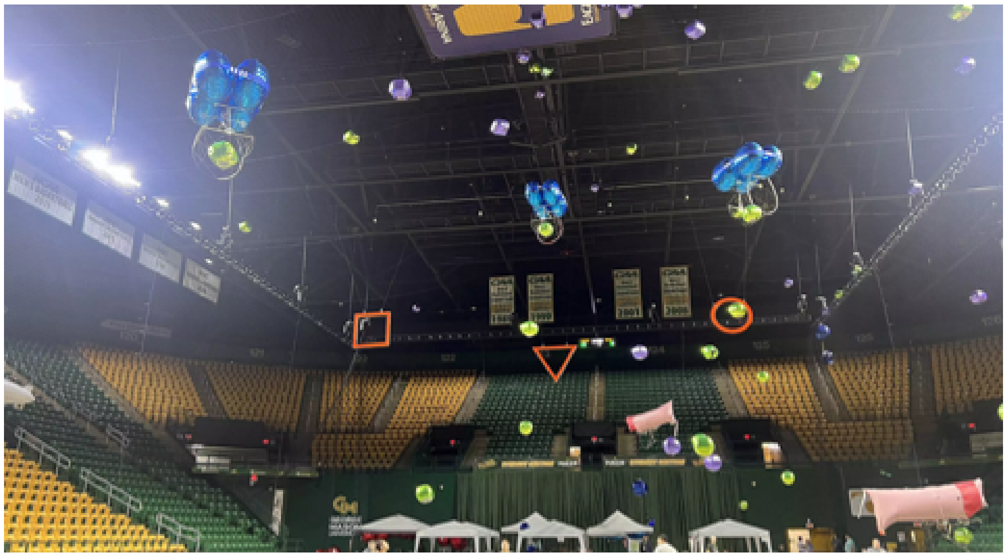
The collegiate Defend the Republic (DTR) competition is one such application that pits teams of autonomous lighter-than-air (LTA) unmanned aerial vehicles against each other in contested airspace.[1–13] Because of the LTA restriction, vehicles face unique weight and computing constraints. LTA restrictions combined with difficulties of robotics mentioned above makes consistent testing of hardware and software on LTA vehicles unsustainable.

This paper lays the foundation for a multi-agent simulation that allows users to perform research on control, perception, and communication algorithms without requiring a physical LTA vehicle. We leverage the graphics and physics of Unreal Engine 5 to simulate image and sensor data similar to what a LTA vehicle would receive in real-world conditions. Building upon previous work,[14] the simulator framework enables agents to receive image and state data, and return actions to a graphics engine in closed-loop cycles. The simulator's transfer to real world environments is validated through a performance test of a YOLOv8 model used in the 2025 Spring DTR competition held at George Mason University (see comparison in Fig. 1). While this simulation has been developed with DTR in mind, it still has numerous applications in the broader autonomous multi-agent space. Examples include wildfire containment[15],[16]

[*]Graduate Student, Department of Intelligent Systems Engineering. ameighan@iu.edu. AIAA Student Member.
[†]Graduate Student, Department of Intelligent Systems Engineering. caedtayl@iu.edu. AIAA Student Member.
[‡]Undergraduate Student, Department of Intelligent Systems Engineering. mattdemp@iu.edu.
[§]Assistant Professor, Department of Intelligent Systems Engineering. odantske@iu.edu. AIAA Member.



(a) Simulated scene in Unreal Engine mirroring the 2025 Spring Defend the Republic competition.



(b) Real-world scene from the 2025 Spring Defend the Republic competition hosted by George Mason University.

Figure 1. Comparison of mirrored scenes in both Unreal Engine and real life.

disaster relief,[17] and satellite internet constellations.[18] This simulator is one step towards safe and efficient deployment of autonomous multi-agent systems.

The following sections are ordered as follows. Section II delves into related work including currently available simulations, software- and hardware-in-the-loop testing, and recent advancements in 3D mesh creation. Section III details the specific requirements imposed by DTR and the resulting architectural design choices. Section IV discusses the design of the simulator environment which includes the graphics engine, networking agents, TCP interface, and python control logic. Section V validates the transfer of perception models between the real-world and simulation and tests a full scoring loop of a LTA autonomous vehicle. Section VI summarizes the simulation and proposes future work.

II. Background

Robotic simulators play an important role in the development of autonomous systems because of the inherent difficulties surrounding real-world testing. In this section, we categorize and discuss existing simulators and recent work in 3D mesh creation.

A. Simulation Environments

Several simulation platforms have been developed to support autonomous behavior for individual agents. Gazebo[19] is one such simulation platform that seems to works best with a single agent. Multiple papers focus on single agents in Gazebo including ground vehicle modeling by Rivera, and TurtleBot navigation by Tamang[20].[21] Attempts have been made to perform autonomous navigation in Gazebo for multiple robots as well,[22] but other platforms seem better suited to this problem. During initial research for the paper, we found Gazebo became significantly slower and less stable compared to other simulations as the number of agents scaled.

There are simulators with direct support for multi-agent autonomy. Notably, CARLA[23] can support multi-client and multi-agent systems. CARLA is built on top of Unreal Engine and focuses on ground-based autonomous driving systems. Microsoft AirSim[24] is another widely used simulator designed for aerial vehicles that utilizes Unreal Engine to provide photo-realistic environments with realistic physics modeling. Both of the simulators utilize older versions of Unreal Engine which makes them less appealing for long term use. AirSim in particular requires older versions of Linux which makes compatibility with certain machine learning packages and libraries difficult. Flightmare[25] builds upon AirSim's strengths with photorealistic rendering and a physics engine. Flightmare uses Unity instead of Unreal, but the benefits of both are similar. The particular success of these simulators gave us further confidence that as agents scaled, Unreal Engine would have no issue handling DTR's distinct requirements.

Others, like the Starcraft Multi-Agent Challenge[26] (SMAC) and Google Research Football[27] (GRF) are less focused on general application, but offer good insight into how multi-agent simulations can be successful for specific problem spaces. SMAC and GRF were both built to train reinforcement learning algorithms. While this type of algorithm is not currently used on the LTA systems in DTR, further research may be done in this area with hopes that trained moedls could see deployment on physical LTAs.

B. Software- and Hardware-in-the-Loop Simulations

Simulation platforms designed for more general use cases often neglect having the ability to test various software or hardware within training loops. Fayad[28] is one early example of hardware-in-the-loop (HIL) testing for missiles. Theile and Dantsker[29, 30] integrated vehicle power awareness for hardware-in-the-loop (HIL) testing of fixed-wing aircraft and flight control software. Mihalič[31] summarizes the importance of hardware-in-the-loop (HIL) as a critical component to testing physical systems. The LTA vehicles used in DTR are one such system that can benefit from the testing of hardware and software in a simulated environment. Since the LTA vehicles are a research platform, it can be incredibly helpful to test control logic, computer boards, and other components before having to fly them in the real world. Unreal Engine has proper tools to interface between itself and API based servers which can connect to different hardware and software.

C. Text-to-Mesh 3D Modeling

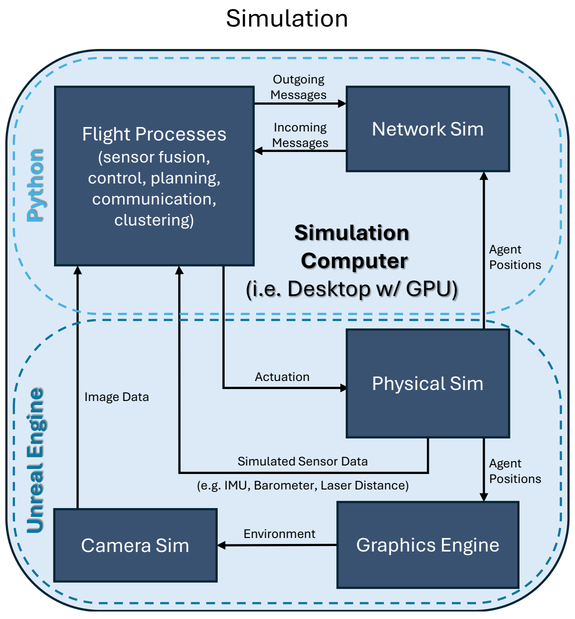
While Unreal does have the capability to interface, one common concern is that creation of handmade environments with custom 3D meshes are time-consuming and costly to produce. With the increasing prevalence of text-to-image and text-to-video models in the past few years, custom 3D mesh generation has become much easier. Magic3D[32] is one such text-to-image diffusion model that allows users to create high quality 3D meshes in less than one hour. While software of this type is not utilized for the environments created in this paper, the option for future development remains if more diverse environments are required for testing.

III. System Architecture

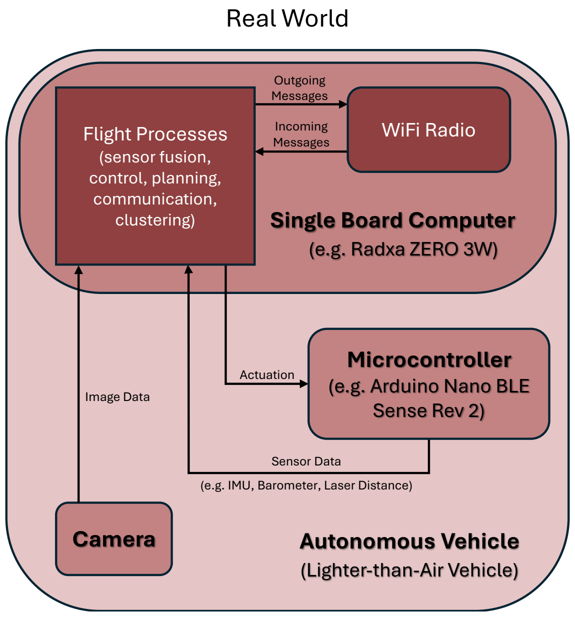
In Defend the Republic, two teams compete within a shared airspace in a game of what is effectively robotic quidditch. LTA vehicles must capture neutrally buoyant balloons and score them in floating nets while simultaneously defending against opponent scoring. This game structure leads to a number of competition specific requirements. The simulation must support a wide variety of vehicle classes. Indiana University alone has designed numerous vehicle types for different strategic purposes including scoring,[33] defense,[34] and deception. The simulator must also interface and provide realistic sensor data for the Python software currently deployed in competition. These requirements are imposed on top of the more general requirements for simulating autonomous multi-agent systems.

To meet both types of requirements, a simulation architecture that mirrors typical pipelines for real-world vehicles was constructed. Figure 2 shows the mirrored architectures for a single agent. In simulation, processes are either performed in Unreal Engine or Python. The transfer of data between the two is performed by a TCP connection.

The process loop of both simulation and real world LTAs goes as follows. First, virtual sensors collect data simulated by the environment within Unreal's graphics and physics engine. This could include data from a wide variety of sensors such as cameras, accelerometers, barometers, gyroscopes, and magnetometers. These are the same sensors that a physical LTA UAV would use in competition. Then, the information gets passed to the Python software-in-the-loop which performs processes like sensor fusion, control, planning, communication, and clustering to generate messages and actuation commands. During competition, messages are transmitted over WiFi to a ground station located outside the field of play for real-time game monitoring. In simulation, these messages are sent directly to Python. Actuation commands are sent either to the micro-controller or the physical sim. Unreal receives these commands, then modifies the positions of the vehicles within the graphics engine. Once the environment is updated, the loop restarts, creating a continuous feedback cycle between Unreal and Python.


Simulation
Python
Unreal Engine
Flight Processes (sensor fusion, control, planning, communication, clustering)
Outgoing Messages
Incoming Messages
Network Sim
Simulation Computer (i.e. Desktop w/ GPU)
Agent Positions
Actuation
Physical Sim
Image Data
Simulated Sensor Data (e.g. IMU, Barometer, Laser Distance)
Agent Positions
Environment
Camera Sim
Graphics Engine

(a)


Real World
Flight Processes (sensor fusion, control, planning, communication, clustering)
Outgoing Messages
Incoming Messages
WiFi Radio
Single Board Computer (e.g. Radxa ZERO 3W)
Actuation
Microcontroller (e.g. Arduino Nano BLE Sense Rev 2)
Image Data
Sensor Data (e.g. IMU, Barometer, Laser Distance)
Camera
Autonomous Vehicle (Lighter-than-Air Vehicle)

(b)

Figure 2. Depiction of Process Architecture: (a) in Simulation, (b) in Real World; squares represent processes, rounded corners represent devices.

Even as more agents are added into the simulation, the overall flow remains the same. Agents, with their own sensors, continue to receive environmental data. Once they process the data, they use it as input to generate actuation commands and messages that they can pass to both other agents and a centralized ground station. Figure 3 highlights

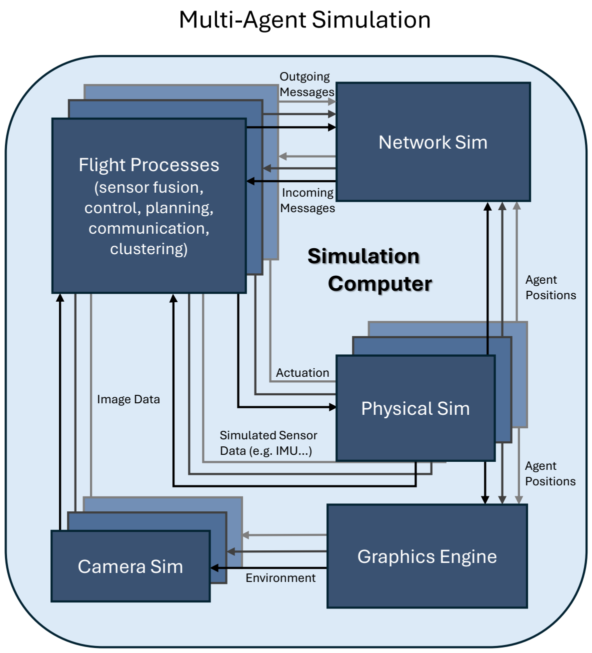


Figure 3. Depiction of Multi-Agent Process Architecture

this group of processes. Each individual agent has their own initialization of flight processes, camera sims, and physical sims. The graphics engine and the network sim are shared between agents. This allows agents to interact with each other in the same environment. Without this, actors could move freely without fears of running into each other. The shared network simulation also allows agents to communicate on the same network, which enables coordination.

IV. Methodology

Unreal Engine 5 and Python are linked together with a TCP connection initialized through Python. Unreal Engine controls the physical, graphical, and sensor components of the simulation while Python code used in the DTR competition generates control logic. To actually do anything in one of Unreal's environments, an actor must be created. The Actor class represents all objects placed or spawned in a particular level. We focus on the Pawn subclass which denotes any actor that can be controlled by either a human player, or an AI entity.

A. UE5 Actors

Unreal initializes the creation of pawns in two ways. The first utilizes a graphical programming language called Blueprints which is much simpler than standard C++ coding. This allows users with little coding experience to more intuitively create actors. Instead of Blueprints, pawns can also be created from scratch with C++ code. This approach offers more flexibility in the functions you can write for the pawn. The main functions of any pawn can be synthesized into three parts. The first, BeginPlay(), is called as soon as the pawn is loaded into a running environment in Unreal. This function creates initial states and variables, and starts reoccurring events. The second, Tick(), is called every frame during a simulation which allows for reoccurring events like data transfer and reception to occur continuously. The third, EndPlay(), is called at the termination of the game or when the pawn becomes inactive. EndPlay() performs any end of game events and removes the pawn from the environment. This paper utilizes two classes of pawns created in C++. One moves in predetermined trajectories while the other interfaces with Python to decide on controls.

WaypointPawn was created to simulate adversarial LTA vehicles and non-static targets present in the environment. Within WaypointPawn, a trajectory can be created with waypoints given upon initialization. This allows users to generate realistic flight paths for adversarial vehicles and targets based on competition data. Because this class moves on a predetermined trajectory, there is no need to connect additional sensors to the pawn. Instead, actors can step along the trajectory at a fixed velocity. While the control logic of WaypointPawn objects is simplistic, it could be further enhanced to chase or avoid other pawns based on their locations.

NetworkPawn on the other hand uses Python software in the control loop to allow for testing of code used in the DTR competition. Thus instead of following waypoints, NetworkPawn can dynamically react to objects in its local view based on real-time control inputs received from Python. To share and receive data between Unreal and Python the simulation utilizes a TCP connection.

B. TCP Interface

Transmission Control Protocol (TCP) is a communication method that guarantees reliable and errorless data transfer. TCP enforces a number of protocols including acknowledgments and retransmission to ensure data passes properly between nodes. This is quite important when it comes to real-time control as incomplete messages could result in the system not taking any actions and ending up in a dangerous position. TCP can be simplified down to a conversation between friends in which an introduction must occur before any actual sharing of information. During transmission, the receiving friend must acknowledge and confirm they understood the message, or request a retransmission. Once all the information is shared, both friends say goodbye and end the connection.

In simulation, the server to share messages in is created before actors are initialized in the environment. Once the server has been opened on a specific port, actors initialize a client socket for Unreal. During each tick of simulation, game data is transformed into binary for easy transmission over the TCP connection. Python receives the data, computes a binary control output, and returns it back to the Unreal client socket. While TCP is great for its consistency, the communication checks can cause slowing with large numbers of agents. In the future, we will also consider using User Datagram Protocol (UDP). UDP skips the acknowledgment phase, which can speed up the overall process at the cost of guaranteed transmission.

C. Flight Processes

While the actual transmission of data is done through the TCP interface, the computation of control outputs for each actor is done in Python. These processes govern how the simulated LTAs interpret sensor inputs and make planning decisions. The flight process diagram is shown in Figure 4. During each tick cycle, the Python server receives the vehicle's positional data and camera imagery. From there, the data passes through several critical steps including sensor fusion, data aggregation, and planning before it outputs a control action.

The first subprocess combines perception and sensor fusion. Camera images are processed using a YOLOv8 model trained on real-world competition data. Then data us fused together using Kalman Filtering to reduce noise coming from the multiple inputs. For example, camera data alone might identify a nearby object, but without position and orientation information, the relative location of that object cannot be properly determined. One potential method to calculate the relative location is triangulation. By using the camera specifications and comparing the size of the object in real life to the bounding box size, an approximation of the objects location can be done. The fused state serves as the base for later planning and control decisions.

If multiple agents are present in the environment and communication is enabled, agents can receive messages about the global environment from other agents. By aggregating this new information with their own, agents can perform basic collaborative behavior. Once incoming messages have been combined with local knowledge, the planning stage can begin.

Planning determines an efficient trajectory for the vehicle to follow. This process begins by checking the current mission objectives of the agent. For example, if the agent currently is attempting to score a goal, then planned trajectories should result in the agent moving towards a goal of the correct color. Currently, trajectory planning is as simple as

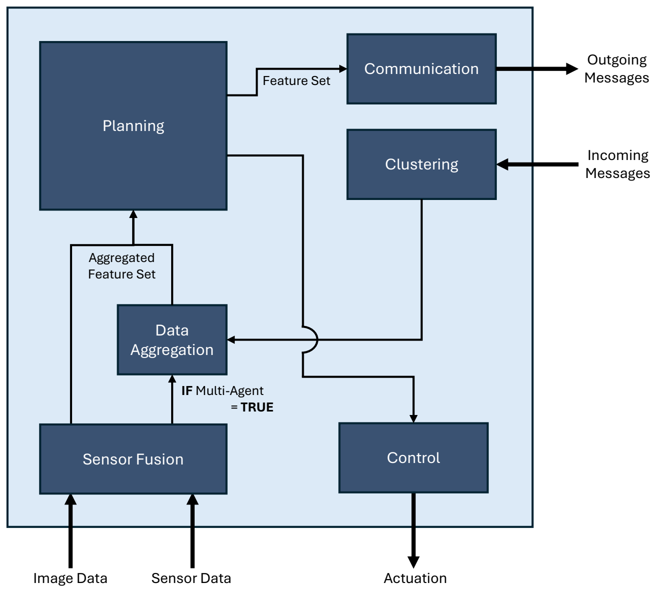


Figure 4. Diagram of Flight Processes for a Single Agent

choosing the object that corresponds to the current mission to keep in the center of the camera frame. While this does not fully utilize the collected data, we are working on implementing segmented trajectories based on locational data. These trajectories would generate a spline from a number of waypoints that concludes with the goal position. Constraints can be applied to the calculated trajectory based on velocity, acceleration, or turn radius to ensure realistic path generation.

Once a trajectory or objective has been selected, the system must compute actuation commands to send to Unreal Engine. These commands translate the high-level trajectory into a low-level control signal. Competition blimps use a PID controller for changes in movement in the forward, up-down, and left-right directions. In the current setup, the Python server encodes these commands into a binary format and sends them back through the TCP socket to the appropriate Unreal client. This handshake completes the control loop. The simulated vehicle applies the commands, moves to a new position, and the next cycle begins. Through this continuous process of sensing, planning, and acting, the simulated vehicle is able to navigate its virtual world autonomously. By performing all computational steps in Python, the system remains flexible and extensible—allowing for rapid prototyping of new algorithms while retaining a reliable simulation interface through Unreal. While flight process testing only included a single agent for this paper, the system architecture is designed to scale to multiple agents with minimal modification. In the next section, we discuss experimental results with the perception model and control logic used for DTR within the simulation environment.

V. Experimental Results

One of the main reasons for creating this simulation pipeline was to perform software- and hardware-in-the-loop testing. If the environment in Unreal was not realistic enough, then the perception models used in competition would not transfer and the pipeline would break down. To ensure we were headed in the right direction, we tested multiple tasks that an LTA vehicle might perform during a DTR match. First, we tested the reliability of the YOLO model used in 2025 Spring DTR competition. We also tested software of a new vehicle that tracks and predicts moving objects within the camera frame. Finally, we tested a normal mission cycle of Indiana University's primary scoring vehicle.

Testing the YOLO model on simulated data from Unreal Engine was done by manually flying a NetworkPawn around the environment and taking screenshots of various objects at different distances and orientations. Figure 1

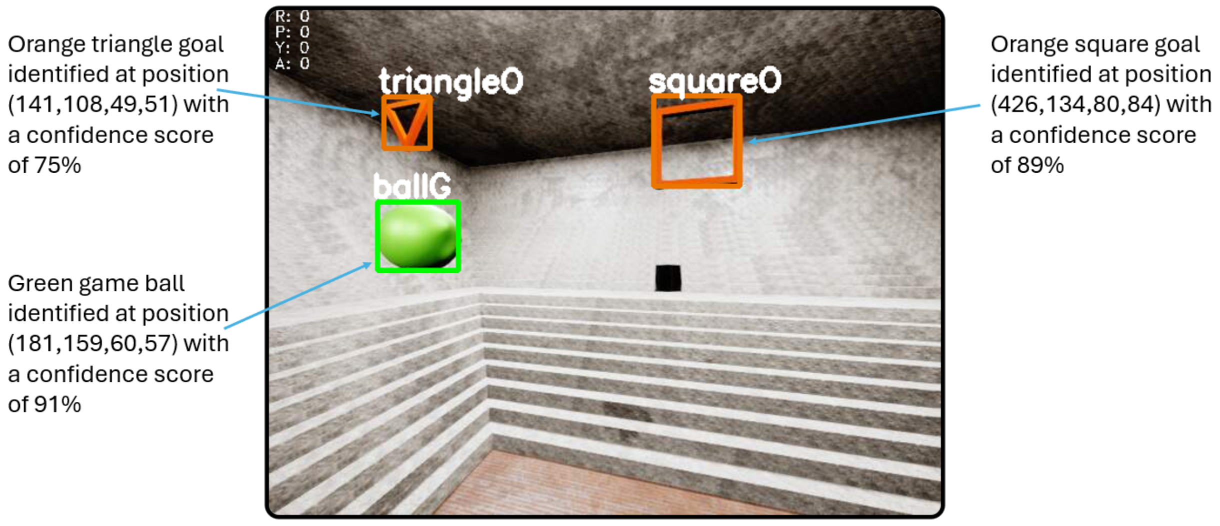

Orange triangle goal identified at position (141,108,49,51) with a confidence score of 75%
Orange square goal identified at position (426,134,80,84) with a confidence score of 89%
Green game ball identified at position (181,159,60,57) with a confidence score of 91%
R: 0
P: 0
Y: 0
A: 0
triangleO
squareO
ballG

Figure 5. Image labeled by YOLOv8 perception model sent to a groundstation from simulation. Each object is defined by the center x and y of the bounding box, the bounding box's width and height, and the confidence of the model in the prediction. Image resolution and FOV intentionally mirrors the camera used onboard of a scoring vehicle in competition.

shows the comparison between a live scene from competition and a simulated image with an intentionally mirrored scene. Objects were shifted multiple times to simulate real-world game conditions in which adversarial LTA vehicles and neutrally buoyant game balls move around the playing field. Once the images were captured, they were manually labeled and added to the YOLOv8 model's test set to determine its accuracy on simulated objects. Results in Table 1 show the accuracy on simulated data is almost equivalent accuracy on real world data. The model did show difficulty with objects very far away, and mislabeled the light colored walls as yellow goals multiple times. Even still, these results showed that the YOLO model could recognize objects in simulation even though it was only trained on real-world data. We believe that the performance of the model in simulation would increase dramatically if a small number of simulated photos were added to the training set. This is because the simulations lighting is constant. This is something to keep an eye on because a model that is too accurate in simulation will not mimic the difficulties faced by the LTA vehicles when lighting is dynamic. In Figure 5 shows the camera image from an LTA vehicle moving towards an orange goal frame. Other objects in the frame are also recognized by the model. This is a common scene that an autonomous vehicle would find itself in as it attempts to score.

We also test the target prediction system which estimates the future position of a detected object by analyzing its recent motion history using a quadratic polynomial fit. Once a target is detected with sufficient confidence, its current position is stored in a trajectory buffer that maintains the last 10 positions, ensuring that only relevant data is used for prediction. These polynomial models are then used to predict the target's position n frames into the future. In Figure 6, the green game ball is shown within a green bounding box, while the cyan trail indicates its past positions.

Table 1. Perception Model Performance on Simulated Images

Metric	Count
Correct Detections	132
Incorrect Labels	22
Missed Detections	14
Total Objects	150
Accuracy on Simulated Data	88.0%
Baseline Accuracy (Test Set)	89.0%

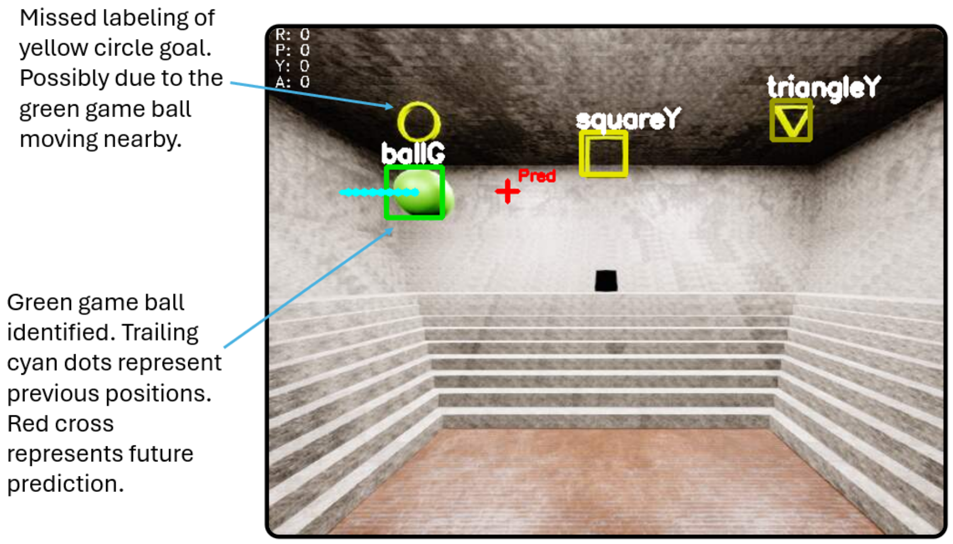

Missed labeling of yellow circle goal. Possibly due to the green game ball moving nearby.
Green game ball identified. Trailing cyan dots represent previous positions. Red cross represents future prediction.
R: 0
P: 0
Y: 0
A: 0
triangleY
squareY
ballG
Pred

Figure 6. Image labeled by YOLOv8 perception model. A NetworkPawn tracking a green game ball with yellow goals in the background records previous positions and predicts future location.

The predicted position, 10 frames ahead in this example, is marked with a red cross. This testing provided further reassurance that the YOLO model could perform in the simulation environment without additional training data.

After the YOLO model was shown to perform adequately, the vehicle's mission cycle was tested. Indiana University's main scoring vehicle has a mission cycle that can be broken up into four main pieces: find a game ball, capture a game ball, find a goal, and score a goal. It was demonstrated that a simulated LTA could repeat successful goal scoring when given environment states similar to what the autonomous vehicle would score on in competition. This confirmation also opens the door for testing of improved cycles that include localization, different perception models, and obstacle avoidance. Before, testing new algorithms and scoring cycles would require a physical LTA vehicle to be set up and maintained for extensive periods of time which was not feasible with limited resources. Now, initial testing can be done in simulation with physical vehicles only being needed for validation of the simulation results.

VI. Summary and Future Work

This paper presents a simulation for multi-agent autonomy that enables testing of software used in the Defend the Republic competition. The simulation uses Unreal Engine 5 for graphics, physics modeling, and sensors which are used by Python via a TCP interface. The developed architecture allows for further development of localization, communication, and planning protocols that will influence the next iteration of Indiana University's autonomous LTA vehicles.

Future work will include the integration of hardware-in-the-loop testing. In particular, we would like to send the data from Unreal directly to the platforms in Figure 2(b) to perform flight processes and communication protocols. The physics used in the simulation environment should become more realistic. We are also interested in adding a discrete-event network simulator like ns-3.[35] This would allow us to properly model communication and clustering events in real-time with the goal of inducing cooperative behavior between autonomous agents. As robotics continues to gain a foothold in real-world activity, the communication between agents will become more and more important. The ability to test these interactions in simulation before deployment is critical, and we hope this work acts as a step in the right direction towards safe deployment of autonomous agents in the real world.

References

[1]Messinger, S., *Modeling, Adaptive Control, and Flight Testing of a Lighter-than-Air Vehicle Validated Using System Identification*, Master's thesis, The Pennsylvania State University, 2022, Master's Thesis.

[2]Mathew, J. P., Karri, D., Yang, J., Zhu, K., Gautam, Y., Nojima-Schmunk, K., Shishika, D., Yao, N., and Nowzari, C., "Lighter-Than-Air Autonomous Ball Capture and Scoring Robot – Design, Development, and Deployment," *arXiv preprint arXiv:2309.06352*, 2023.

[3]Mendoza, A., Lovelace, A., Potter, S., and Koziol, S., "Sensor Fusion Image Processing for Autonomous Robot Blimps," *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2023, pp. 312–316.

[4]Simmons, J., Lovelace, A., Tucker, D., Mendoza, A., Coates, A., Alonzo, J., Li, D., Yi, X., Potter, S., Mouritzen, I., Smith, M., Banta, C., Hodge, R., Spence, A., and Koziol, S., "Design and Construction of a Lighter than Air Robot Blimp," *2023 ASEE GSW*, No. 10.18260/1-2-1139-46327, ASEE Conferences, Denton, TX, June 2024, https://peer.asee.org/46327.

[5]Xu, J., D'antonio, D. S., Ammirato, D. J., and Saldaña, D., "SBlimp: Design, Model, and Translational Motion Control for a Swing-Blimp," *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2023, pp. 6977–6982.

[6]Li, K., Hou, S., Negash, M., Xu, J., Jeffs, E., D'Antonio, D. S., and Saldaña, D., "A Novel Low-Cost, Recyclable, Easy-to-Build Robot Blimp For Transporting Supplies in Hard-to-Reach Locations," *2023 IEEE Global Humanitarian Technology Conference (GHTC)*, IEEE, 2023, pp. 36–42.

[7]Santens, L., D'Antonio, D. S., Hou, S., and Saldaña, D., "The Spinning Blimp: Design and Control of a Novel Minimalist Aerial Vehicle Leveraging Rotational Dynamics and Locomotion," 2025.

[8]Xu, J., Vu, T., D'Antonio, D. S., and Saldaña, D., "MochiSwarm: A testbed for robotic blimps in realistic environments," 2025.

[9]Nojima-Schmunk, K., Turzak, D., Kim, K., Vu, A., Yang, J., Motukuri, S., Yao, N., and Shishika, D., "Manta Ray Inspired Flapping-Wing Blimp," *arXiv preprint arXiv:2310.10853*, 2023.

[10]Dantsker, O. D., "Design, Build, and Fly Autonomous Lighter-Than-Air Vehicles as a Project-Based Class," AIAA Paper 2024-4375, AIAA Aviation Forum 2024, Las Vegas, NV. 2024.

[11]Dantsker, O. D., "Integrating Unmanned Aerial Systems into the Intelligent Systems Engineering Curriculum," Paper 2024-1185, Congress of the International Council of the Aeronautical Sciences, Florence, Italy. 2024.

[12]Taylor, C., Widjaja, M., and Dantsker, O. D., "Remotely-Processed Vision-Based Control of Autonomous Lighter-Than-Air UAVs With Real-Time Constraints," AIAA Paper 2025-1344, AIAA SciTech Forum 2025, Orlando, FL. 2025.

[13]Widjaja, M., Taylor, C., Meighan, A., and Dantsker, O. D., "Viability of NPU-Equipped SBCs for Real-Time Onboard Vision Autonomy on Lighter-Than-Air UAVs," AIAA Aviation Forum 2026, Las Vegas, NV. 2025.

[14]Meighan, A., Taylor, C., Ponniah, J., and Dantsker, O. D., "Design of a Multi-Agent Simulation With Heterogeneous Agents, Adversaries, and Targets," AIAA Paper 2025-2478, AIAA SciTech Forum 2025, Orlando, FL. 2025.

[15]Haksar, R. N. and Schwager, M., "Distributed Deep Reinforcement Learning for Fighting Forest Fires with a Network of Aerial Robots," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1067–1074.

[16]Julian, K. D. and Kochenderfer, M. J., "Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 8, 2019, pp. 1768–1778.

[17]Goodchild, A. and Toy, J., "Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO2 emissions in the delivery service industry," *Transportation Research Part D: Transport and Environment*, Vol. 61, 2018, pp. 58–67, Innovative Approaches to Improve the Environmental Performance of Supply Chains and Freight Transportation Systems.

[18]Katila, C. J., Di Gianni, A., Buratti, C., and Verdone, R., "Routing protocols for video surveillance drones in IEEE 802.11s Wireless Mesh Networks," *2017 European Conference on Networks and Communications (EuCNC)*, 2017, pp. 1–5.

[19]Koenig, N. and Howard, A., "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Vol. 3, Sept 2004, pp. 2149–2154 vol.3.

[20]Rivera, Z. B., De Simone, M. C., and Guida, D., "Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments," *Machines*, Vol. 7, No. 2, 2019.

[21]Tamang, M. T., Maheriya, D., Sharif, M. S., and Sutharssan, T., "Autonomous Navigation for TurtleBot3 Robots in Gazebo Simulation Environment," *2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2024, pp. 568–574.

[22]Pandit, A., Njattuvetty, A., and Mulla, A. K., "ROS-Based Multi-Agent Systems COntrol Simulation Testbed (MASCOT)," 2022.

[23]Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V., "CARLA: An Open Urban Driving Simulator," *Proceedings of the 1st Annual Conference on Robot Learning*, edited by S. Levine, V. Vanhoucke, and K. Goldberg, Vol. 78 of *Proceedings of Machine Learning Research*, PMLR, 13–15 Nov 2017, pp. 1–16.

[24]Shah, S., Dey, D., Lovett, C., and Kapoor, A., "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," 2017.

[25]Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D., "Flightmare: A Flexible Quadrotor Simulator," *Conference on Robot Learning*, 2020.

[26]Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C., Torr, P. H. S., Foerster, J. N., and Whiteson, S., "The StarCraft Multi-Agent Challenge," *CoRR*, Vol. abs/1902.04043, 2019.

[27]Kurach, K., Raichuk, A., Stanczyk, P., Zajac, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., and Gelly, S., "Google Research Football: A Novel Reinforcement Learning Environment," *CoRR*, Vol. abs/1907.11180, 2019.

[28]Fayad, M. E., Hawn, L. J., Roberts, M. A., Schooley, J. W., and Tsai, W.-T., "Hardware-In-the-Loop (HIL) simulation: an application of Colbert's object-oriented software development method," *Proceedings of the Conference on TRI-Ada '92*, TRI-Ada '92, Association for Computing Machinery, New York, NY, USA, 1992, p. 176–188.

[29]Theile, M., Dantsker, O. D., Nai, R., and Caccamo, M., "uavEE: A Modular, Power-Aware Emulation Environment for Rapid Prototyping and Testing of UAVs," 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).

[30]Dantsker, O. D., Theile, M., and Caccamo, M., "A High-Fidelity, Low-Order Propulsion Power Model for Fixed-Wing Electric Unmanned Aircraft," AIAA/IEEE Electric Aircraft Technologies Symposium, Jul. 2018.

[31]Mihalič, F., Truntič, M., and Hren, A., "Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges," *Electronics*, Vol. 11, No. 15, 2022.

[32]Lin, C.-H., Gao, J., Tang, L., Takikawa, T., Zeng, X., Huang, X., Kreis, K., Fidler, S., Liu, M.-Y., and Lin, T.-Y., “Magic3D: High-Resolution Text-to-3D Content Creation,” 2023.

[33]Taylor, C. and Dantsker, O. D., “Lighter-Than-Air Vehicle Design for Target Scoring in Adversarial Conditions,” AIAA Paper 2024-3896, AIAA Aviation Forum 2024, Las Vegas, NV. 2024.

[34]Taylor, C. and Dantsker, O. D., “Lighter-Than-Air-Vehicle Design for Adversarial Defense,” AIAA Paper, AIAA Aviation Forum 2025, Las Vegas, NV. 2025.

[35]Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J., “Network Simulation with the ns-3 Simulator,” SIGCOMM demonstration, 2008.