AIAA SciTech Forum
6-10 January 2025, Orlando, FL
AIAA SCITECH 2025 Forum

10.2514/6.2025-2478

Design of a Multi-Agent Simulation with Heterogeneous Agents, Adversaries, and Targets

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

Abstract

Distributed multi-agent UAV systems motivate fundamental problems in control theory and distributed networking. Although these problems are often studied in isolation, emerging applications of UAV systems require an approach that jointly addresses the control and communication issues of interest. One particular application of interest is the Defend the Republic competition in which teams compete to solve a multi-agent autonomous problem. One major obstacle to this approach is the lack of simulation tools that capture the dynamics of both control and communication so that agents can coordinate between themselves. This paper describes the design of a simulation tool that mirrors the architecture of a real world system to ease the training and testing process of multi-agent systems. The proposed simulator combines Gazebo and ROS2 to simulate a decentralized approach to clustering and communication protocols. A physics engine to deal with the wide array of heterogeneous vehicles used in competition is formulated. Some of the fundamental challenges of this architecture are discussed and resolved.

I. Introduction

Many applications in robotics simplify down to the following problem: a system of agents must autonomously discover and visit targets distributed over three-dimensional space in the shortest possible time. The focus of this paper is on the design of a multi-agent simulator, motivated by the Defend the Republic (DTR) competition. In this competition, two teams of agents compete against each other in a three-dimensional bounded airspace to find, capture, and deliver targets to goals. The agents are lighter-than-air vehicles (LTAs) or "blimps", and the targets are neutrally buoyant spherical objects. Teams aim to outscore each other by delivering more targets to goal locations while simultaneously stopping the opponent from doing the same.

Competing teams equip agents with cameras that capture local field-of-view (FOVs), inertial measurement units (IMUs) for detecting displacement, barometers for measuring height, wireless transceivers for inter-agent communication, and microprocessors (such as Arduinos and Raspberry PIs) for control logic. The microprocessor delivers PWM signals to the motor controllers which regulate propeller-generated thrust. The overall architecture supports coordinated route-planning, path-finding, and target coverage between agents, in addition to the low-level PID control required for consistent trajectories. Complex maneuvers such as flips and rotations (which have been demonstrated manually) can potentially be executed autonomously. However, testing the architecture on actual physical LTAs is difficult. It involves inflating balloons with helium, attaching the inflated balloons to the vehicle frames, charging the vehicle batteries, and deploying the vehicles simultaneously inside a large, closed-roof area. The entire process requires significant time, personnel, and facility usage.

This paper proposes a simulator design that combines high-level multi-agent planning with low-level single-agent control. The simulator models the dynamics of the LTAs and balloons and enables testing of the control stack, without requiring deployment in a real-world environment. It supports heterogeneous vehicles that require varying equations to describe their dynamics. This requirement comes straight from DTR in which vehicles rely on unique attributes to

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

perform their tasks. For example, some LTAs rely on speed and maneuverability to "impale" and incapacitate opposing vehicles. This class of vehicle would need different equations to describe its movement in comparison to a slower moving vehicle designed to capture and score goals. The proposed simulator also allows for development of system policies directing the agents' movement. An optimal system policy should exploit these classes to maximum effect. Simple heuristics or closed-form expressions will not suffice for such a policy. Instead, agents require more complex functions that can interpret the global state.

Recent advances in machine learning make neural networks an attractive option for approximating complicated policies. Although neural networks require extensive training, they are relevant to many modules of the control stack. A simulation allows for data collection on a large scale for this specific purpose. For instance, the object detection module uses neural networks to identify target balloons and goals in the local field-of-view (FOV), whereas the planning module uses neural networks to update an agent's trajectory based on its current understanding of the environment. The proposed simulator enables end-to-end training of all neural networks in the stack to achieve approximately optimal performance.

A key component of the simulator is a decentralized implementation in ROS2 of hierarchical multi-level clustering. A cluster is a coordinated unit of agents that shares similar trajectories and/or objectives. Each cluster elects a designated cluster-head, which coordinates activity within the cluster and schedules intra/inter-cluster communication. In heterogeneous systems (i.e., with different vehicle classes), cluster-heads could default to vehicles with more computing power, while more agile vehicles with less computing power could serve as cluster members. Cluster-heads form higher-level clusters and elect higher-level cluster-heads until all agents belong to a single clustering tree. The clustering tree is dynamic, adapting to the trajectories of the agents in real-time to ensure reliable coordination. Agents can leave and join other clusters as their trajectories evolve. Similarly, cluster-heads can release their responsibilities to a designated second if they stray too far from their cluster members. A more in-depth description of hierarchical clustering can be found in the author's previous work.[1]

Since this process is decentralized, agents must periodically enter into cluster formation and maintenance phases in which they elect new cluster-heads and update membership. Even when local clocks are appropriately synchronized, the process is still event-driven; changes in cluster membership occur when agents drift away from their current cluster-members towards other clusters. ROS2 supports the simulation and implementation of decentralized multi-agent coordination schemes. More details about the required nodes and topics appear in subsequent sections of this paper.

The clustering algorithm in the design of the proposed simulator is one way to enable coordinated planning in multi-agent systems. Another popular approach in the reinforcement learning (RL) literature is centralized-training-decentralized execution (CTDE).[2,3] Many variants of CTDE exist, but the key idea is for agents to implicitly coordinate their activity through local "value functions" (i.e., estimates of the long-term utility of executing actions in the current time slot) that approximate the ground truth of the global system. The issue with this approach is that agents never know with certainty how their neighbors will behave, even if all agents share common views of the global state. The extra functionality introduced to address this uncertainty, usually applies more complicated reasoning to predict the actions of neighboring agents.[4–7] By contrast, the proposed simulator allows the designated cluster-head to suggest, or, if necessary, plan actions for its cluster-members. This approach supports locally-centralized (instead of distributed) planning, which provides a more reliable and predictable foundation for coordination. Locally-centralized planning is effective in systems where agents require precise coordination with their neighbors but higher-level aggregated coordination with agents located remotely. The target coverage problem described at the onset, and DTR in particular are examples of systems that exhibit this locally-interactive behavior.

This paper is organized as follows: In Section II, simulators used for similar applications are discussed. Then, Section III expands upon the specific requirements for a simulator inspired by Defend the Republic. Afterwards, Section IV describes the proposed simulator's architecture and design, including the proposed integration with existing tools. The paper concludes in Section V with a summary and statement of future work.

II. Existing Simulators

We review a number of simulators found in the literature. While these simulators address a number of the requirements for the proposed simulator, they each lack a number of requirements specific to Defend The Republic. Microsoft's AirSim[8] is an open-source simulator widely used in the literature for aerial robotics testing. AirSim uses Unreal Engine to create photo-realistic environments and PhysX to simulate the physics of aerial vehicles. This engine allows for the accurate simulation of forces like gravity, friction, aerodynamics, and vehicle handling for quadcopters. Dosovitskiy's[9] CARLA simulation, developed in 2017, focuses specifically on autonomous driving for ground-based vehicles. Both AirSim and CARLA support multi-agent systems which allows for complex studies of path-planning. Both platforms serve as vital tools for benchmarking algorithms, and continue to be used in current research[10].[11]

Flightmare[12] builds on AirSim's ability to deploy multiple vehicles in photo-realistic environments. Flightmare is composed of a configurable rendering engine built on Unity and a flexible physics engine for dynamics simulation. Unlike Airsim, Flightmare separates these two components allowing them to run independently from one another. This can be incredibly useful when training tasks that don't require the usage of the physics engine. All three of the simulators discussed above struggle to simulate vehicles that are not rigid body systems such at LTAs. The proposed simulation allows the rendering to run without input from the physics engine so that real-world flight data can be used.

III. Requirements

At the Defend The Republic (DTR) competition, collegiate teams strive to use their autonomous lighter-than-air vehicle (LTA) vehicles (i.e., autonomous blimps) to autonomously capture helium balloons and score them into the opponent's goals——effectively Robotic Quidditch. Fig. 1 shows an example of the Indiana University's LTA vehicles navigating the game space during a DTR match. Specifically, the goal is for the LTA vehicles to autonomously capture green and purple helium balloons and score them into the opponent's fluorescent yellow or orange goals. Fig. 2 shows a two-dimensional birds-eye view of a DTR match. The scope of the challenge includes the physical architecture, sensor payload and design, software implementation, cyber-physical development of the entire system. Designs are constrained by total buoyancy, which in turn limits complexity, sensing capabilities, and maximum compute power. Typical implementations include camera-based computer vision navigation with brush-less motor propulsion. Basic nets are used for ball capturing and mylar balloons provide the lift for the vehicles.[13–21]

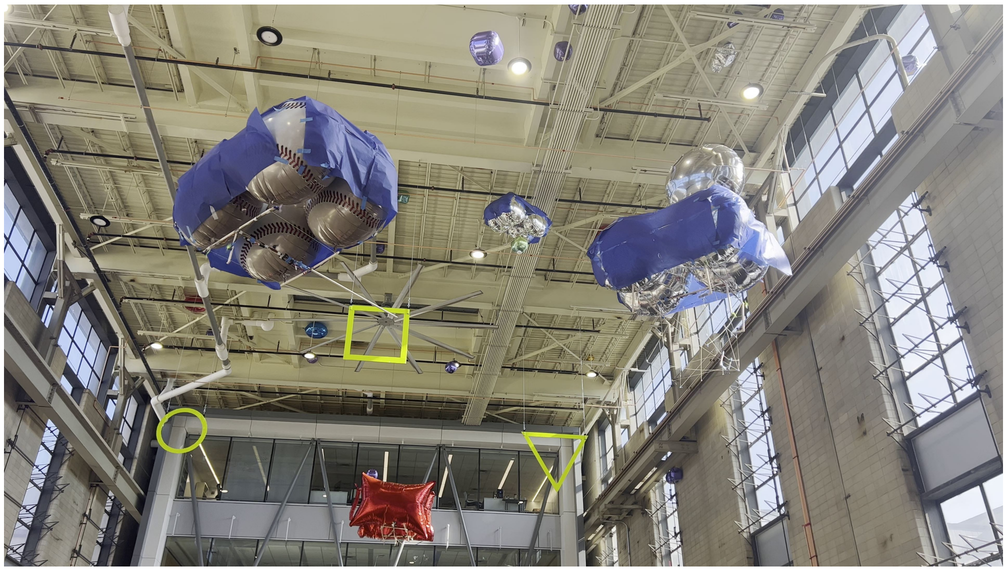


Figure 1. Indiana University DTR team's LTA vehicles (blue) navigating the game space during an match against Baylor University (red) with goal posts (yellow) and game balls (green and purple) in the background.

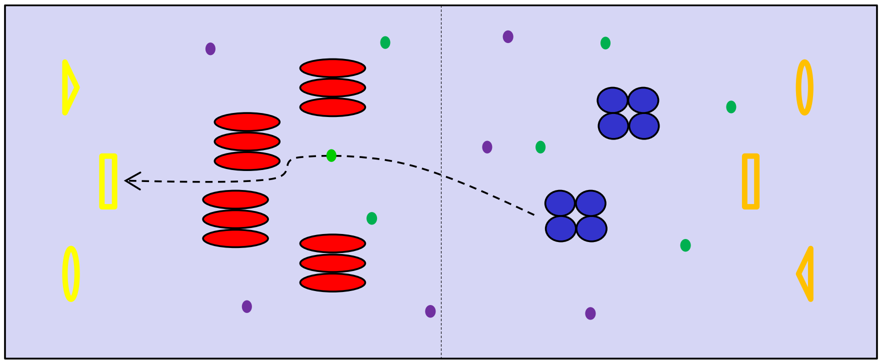


Figure 2. Birds-eye view representation of a DTR match layout.

This unique game design results in a number of DTR-specific requirements for a simulation. The heterogeneity of LTAs used in competition means that each class of vehicle will interact with the environment in different ways. To properly train LTAs to perform maneuvers in the real world, they must interact with the simulated environment in a similar fashion to how actuation outputs would impact an environment in real-life. An LTA physics simulation must account for aerodynamic, buoyancy, thrust, power, and inertial properties for multiple unique vehicle designs within a single environment.

In order to effectively simulate control feedback and train navigation algorithms, the simulator must also be able to interface with sensor packages that can emulate the readings the sensors will produce in real-life. This also relates to the training of object detection and path planning as inputs must be comparable during training to the testing readings. It should also be possible to add noise to these sensors to better simulate real-world environments. Post-processing can also be used to minimize the loss between unrealistic environments by transforming image data into other data forms.

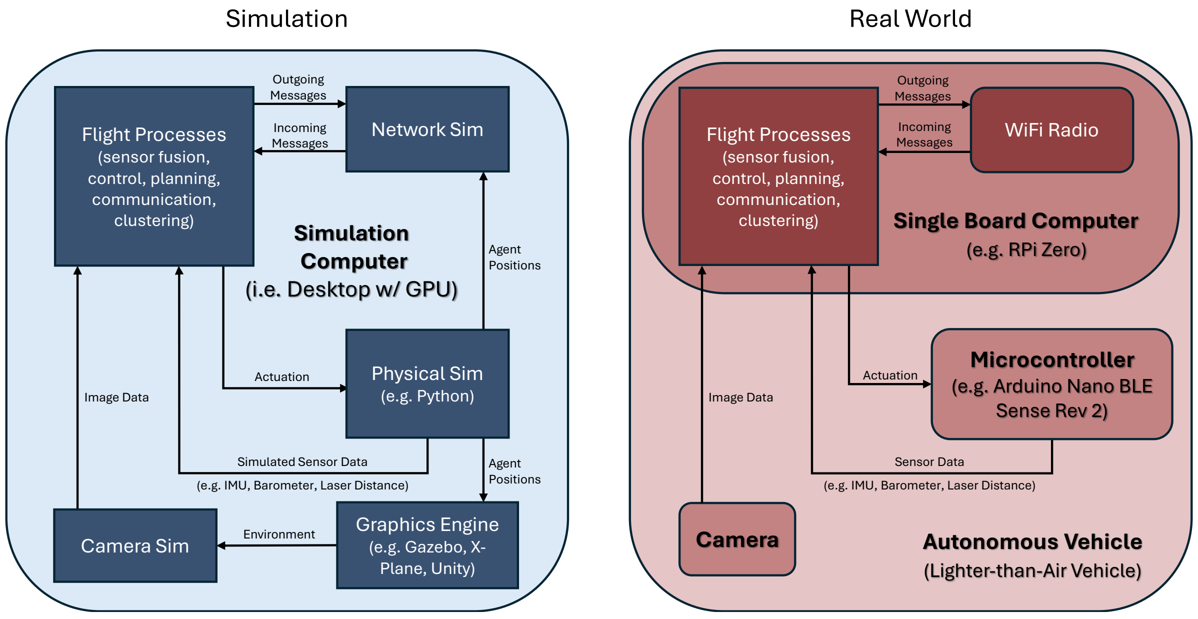
To effectively train vision algorithms, it is critical for the simulator not only to incorporate realistic environmental dynamics and physics models, but to properly render the perspective of each LTAs camera. The simulation's rendering engine should account for certain key properties of real-world objectives and environmental changes, such as the metallic reflection from the game balls. The impact of dynamic lighting on the perceived colors of goals and game balls significantly impacts the performance of most computer vision algorithms based on color filtering.

Furthermore, integrating software-in-the-loop capabilities to the simulation suite enables the software to interface directly with the virtual environment, allowing developers to test algorithms before seamlessly porting it to real-world hardware.

IV. Simulation Design

To meet the requirements set out above, the proposed simulator bases its structure off of the real world implementation. At its core, the simulation depends on the computer to maintain a number of activities during run-time. The simulator is composed of four main parts: the processing unit, the physics engine, the graphics engine, and the network simulator which is primarily used for multi-agent scenarios. Integrated hardware serves as an intermediary between the physics and graphics engine so that agents can receive data that mirrors sensor output in real life.

Fig. 3 shows the similarities and differences between simulation and real world architecture for a single agent. In simulation, there is no real world environment. Instead, this environment must be generated by a graphics engine with input from the physical sim. Because of this, all processes in simulation are run on a single computer. This differs from the real world implementation that uses multiple pieces of hardware to create the connected system. The architecture of a multi-agent system, shown in Fig. 4 shows how a single graphics engine and network sim take responsibility for the input of all agents.


Simulation
Real World
Flight Processes (sensor fusion, control, planning, communication, clustering)
Outgoing Messages
Incoming Messages
Network Sim
Simulation Computer (i.e. Desktop w/ GPU)
Agent Positions
Image Data
Actuation
Physical Sim (e.g. Python)
Simulated Sensor Data (e.g. IMU, Barometer, Laser Distance)
Camera Sim
Environment
Graphics Engine (e.g. Gazebo, X-Plane, Unity)
WiFi Radio
Single Board Computer (e.g. RPi Zero)
Microcontroller (e.g. Arduino Nano BLE Sense Rev 2)
Sensor Data (e.g. IMU, Barometer, Laser Distance)
Camera
Autonomous Vehicle (Lighter-than-Air Vehicle)

(a) (b)

Figure 3. Depiction of Process Architecture: (a) in Simulation, (b) in Real World; squares represent processes, rounded corners represent devices.

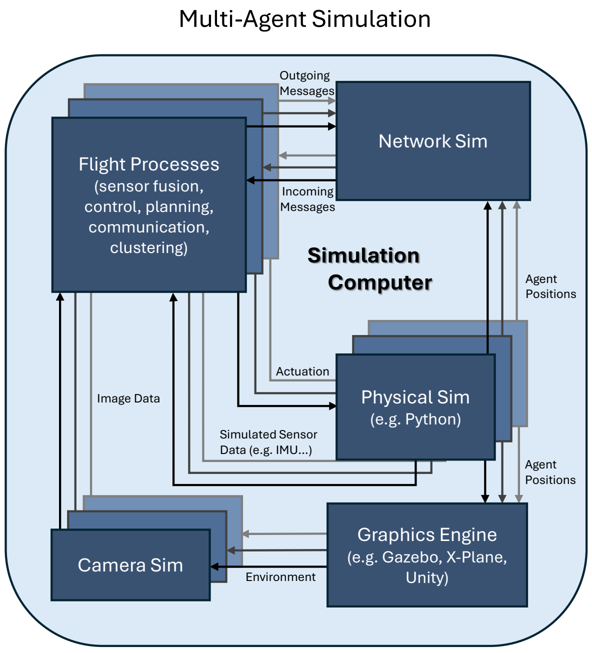

Multi-Agent Simulation
Outgoing Messages
Network Sim
Flight Processes (sensor fusion, control, planning, communication, clustering)
Incoming Messages
Simulation Computer
Agent Positions
Actuation
Physical Sim (e.g. Python)
Image Data
Simulated Sensor Data (e.g. IMU...)
Agent Positions
Graphics Engine (e.g. Gazebo, X-Plane, Unity)
Camera Sim
Environment

Figure 4. Depiction of Multi-Agent Process Architecture

A. Flight Processes

In flight processes, sensor data is fused with previous knowledge to provide an accurate estimate of the system state. A Kalman filter is one possible way to estimate an agent's state. Kalman filters are an iterative process in which the state is first predicted based on actuation inputs to the system, then updated based on sensor readings.[22] The research community studying robotics has done extensive review on the applications of Kalman filtering for sensor fusion. Chen[23] summarizes previous uses of Kalman filtering as a solution for robotic vision problems. Fig. 5 shows the image and sensor data inputted into the sensor fusion block which generates a feature set representative of the agent's position and its understanding of the local environment.

During the simulation of multi-agent systems, agents communicate their feature set through ROS2 nodes that allow them to subscribe and publish to a message board called a topic. This process emulates how a WiFi radio would function in real life. As agents receive incoming messages through the clustering block, they aggregate their local feature set with the other members of their cluster. Agents determine their neighbors by the hierarchical clustering protocol described earlier in this paper. As agents receive messages from their clusters, they can further filter messages that lack a steady signal strength.

Only after the aggregation of feature sets sent from nearby neighbors is the aggregated feature set used for path and trajectory planning. After the agent's trajectory has updated, it is sent to the control block which calculated the actuation commands used by the physical sim to update the agent's position.

The flight processes used in simulation deliberately replicate those used by LTAs during real world missions. This guarantees easy transfer of algorithms and other functions from the simulated environment to real world applications like DTR. This mirroring makes the development and deployment of our autonomous systems more efficient.

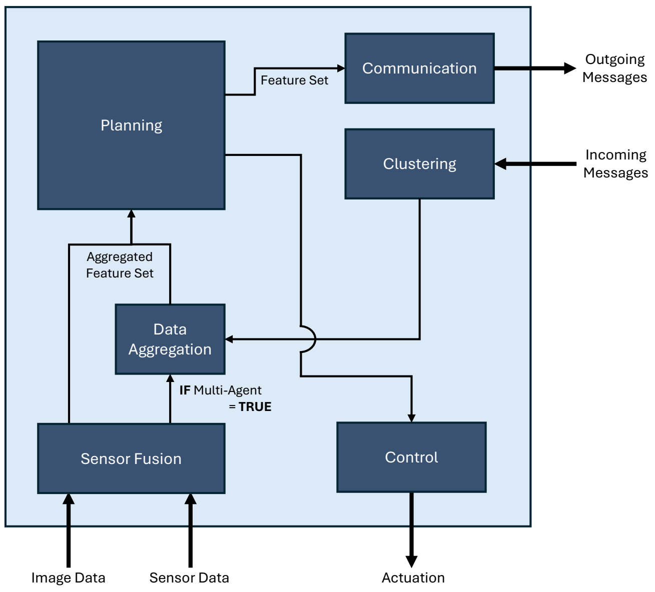


Figure 5. Diagram of Flight Processes for a Single Agent

B. Physical Sim

The physical simulator is used to artificially recreate the movement and control response of the vehicles and other objects (e.g. target balloons) to the environment. The resultant positional data of all objects are then used by the graphics engine and network simulator, which is described more in the following subsections. The physical simulator is broken down into two portions: (1) an environment modeling of the conditions the objects are moving in and (2) an object modeling of the each of the vehicles and other objects based on the environment and control commands, yielding movement. The physical simulation would be done mathematically using Python or similar high-level programming language, applying the two aforementioned modeling methods described in greater detail below.

A key component to physical simulation of aerial vehicles is their operational environment . Movements of air, i.e. horizontal and vertical wind, greatly affect the position and motion of UAVs, especially smaller fixed wing and LTA vehicles. There has been significant research into simulating wind vector fields in and outside of buildings. Additionally, air density changes due to pressure and temperature changes must also be taken into account as the buoyancy of LTA vehicles will change as a result. Both of these environmental factors may be modeled as fixed or dynamically changing over time.

Once the environment is set up, individual vehicles and other objects can be modeled using 1st-order principles. For example, the primary LTA vehicle used in DTR[24] is idealized in the freebody diagram presented in Fig. 6 diagram. The forces and moments include buoyancy (B) from the helium balloons, thrust (T) produced by the 4 propellers, drag (D) generated by the helium balloons in forward movement and rotation, and weight (W) from all the LTA vehicle components.

Each of these forces and moments shown can be easily determined using widely available sources or empirical in-house testing. Buoyancy can either be calculated based on the expected balloon envelope volume, helium purity, and relative air density; or can be measured for each ballon using a scale. Thrust can be determined by mapping propulsion system response to throttle input commands; related efforts are already underway characterizing propulsion system components, including propellers, motors, electronic speed controllers, and batteries, which will not only allow thrust to be determined but also the vehicle to be optimized for its mission.[25] The drag forces can be estimated from balloon envelope using drag coefficient estimates from Horner[26] or by measuring vehicle speeds to known thrust and throttle inputs (per above). The weight of the un-inflated LTA vehicle can easily be measured or estimated from CAD models.

Based on the design of the example vehicle, simulated proportional control of the 2 differential-drive side and vertical fan motors with 0 to 100% of positive or negative thrust enables forward/backward, differential turning, and ascending/descending motion to be achieved. Additionally, the back blower motor will allow internal blowing/sucking of “target” balloons and adverse pitching of the vehicle. Once a “target” balloon is caught, the additional drag can be added to the forces simulated for the vehicle.

Similarly, simpler LTA design or “target” balloon control and motions can be simulated with a sub-set of the presented forces. For example, a ‘target” balloon can be modeled with only buoyancy, weight, and drag; buoyancy and drag depend on the environmental wind and air density. Thus, potential agents and targets and be simulated by modeling the environment and individual obiects.

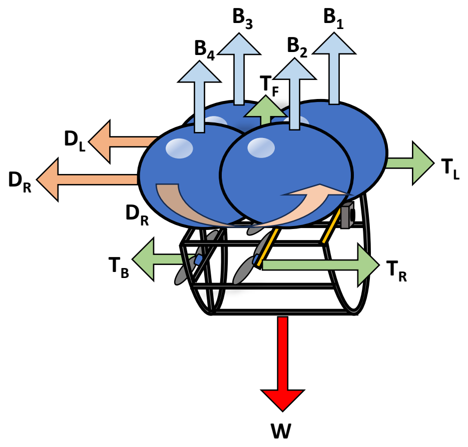


Figure 6. Diagram of 1st-order forces and moments used to model a blimp.

C. Graphics Engine

The graphics engine renders 3D environments so that simulated sensors can produce realistic outputs. When deciding on a particular graphics engine, one must consider the trade-off between graphics quality, speed, and dynamics. Since our proposed physical sim bypasses in-built physics engines integrated directly with the graphics engine, the graphics engine must be able to take in positional data to iteratively update an agents position. All agents share the same instance of the graphics engine which allows them to interact with each other directly. This approach allows agents to experience realistic scenarios in which collision avoidance techniques must be utilized. This further provides a realistic simulation so that agents can experience events that could occur in the real world competition.

Gazebo is one such simulator that supports quasi-realistic rendering and the custom environments. It also integrates with ROS2 which makes the development of multi-agent systems easier. In this implementation, Gazebo receives agents' real-time positional data from the external physical simulation and to iteratively update the environment's state. It also connects with the ROS2-based network sim to allow for communication between clustered agents.

D. Network Sim

In single-agent scenarios the Network Sim is not utilized unless a ground station is required. In this case, information can be sent from the agent to an off-board computer for more processing power. Additional processing power can be utilized for larger machine learning models, high resolution images, and faster inference speeds based on network bandwidth.[21] The ground station also provides a means to receive live data streams from the vehicle during flight.

On the other hand, multi-agent systems can utilize the network sim as a message board for intra- and inter- cluster communication. Agents can receive messages sent to a ROS2 topic based on their cluster status. The network sim has the primary responsibility of filtering these messages so that agents only receive messages that come from agents within a certain radius. As the number of agents in simulation becomes large, the topic can become over saturated with messages. To solve this problem, multiple topics can be constructed for agents based on location or cluster.

V. Summary and Future Work

This paper proposed a simulator design that enables training and testing of multi-agent systems composed of lighter-than-air vehicles in quasi-realistic rendering environments. The discussed pipelines will assist in the eventual deployment of hardware that allows agents to determine their position, and the training of computer vision and sensor fusion algorithms. The architecture shown allows for development of decentralized, multi-agent clustering protocols that can provide agents with the required information to coordinate with each other to find optimal solutions to the unique problems posed by Defend the Republic. It also closely mirrors the architecture used in competition on the vehicles to allow for efficient deployment.

Future work will involve the development of the proposed simulator, specifically applied to the DTR competition. This will include modeling the physical, network, and graphical aspects of the competition using both theoretical and empirical models of various system and environmental properties. As the simulation suite matures to handle multiple systems with unique dynamics, aquatic, aerial, and multi-modal agents can be implemented into the simulation to enable even more complex scenarios.

References

[1]Meighan, A., Ponniah, J., and Dantsker, O. D., "Simulations of Hierarchical Belief Sharing for Multi-Agent Coverage with Heterogeneous Targets," *AIAA SCITECH 2024 Forum*, AIAA, 2024.

[2]Kraemer, L. and Banerjee, B., "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, Vol. 190, 2016, pp. 82–94.

[3]Amato, C., "An Introduction to Centralized Training for Decentralized Execution in Cooperative Multi-Agent Reinforcement Learning," 2024.

[4]Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, T. K. S., Koenig, S., and Choset, H., "PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning," *IEEE Robotics and Automation Letters*, Vol. 4, 2018, pp. 2378–2385.

[5]Wang, Y., Xiang, B., Huang, S., and Sartoretti, G., "SCRIMP: Scalable Communication for Reinforcement- and Imitation-Learning-Based Multi-Agent Pathfinding," *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 9301–9308.

[6]Li, W., Chen, H., Jin, B., Tan, W., Zha, H., and Wang, X., "Multi-Agent Path Finding with Prioritized Communication Learning," *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10695–10701.

[7]Lin, Q. and Ma, H., "SACHA: Soft Actor-Critic With Heuristic-Based Attention for Partially Observable Multi-Agent Path Finding," *IEEE Robotics and Automation Letters*, Vol. 8, No. 8, Aug. 2023, pp. 5100–5107.

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

[9]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.

[10]Li, Q., Jia, X., Wang, S., and Yan, J., "Think2Drive: Efficient Reinforcement Learning by Thinking in Latent World Model for Quasi-Realistic Autonomous Driving (in CARLA-v2)," 2024.

[11]Turco, L., Zhao, J., Xu, Y., and Tsourdos, A., "A Study on Co-simulation Digital Twin with MATLAB and AirSim for Future Advanced Air Mobility," *2024 IEEE Aerospace Conference*, 2024, pp. 1–18.

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

[13]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.

[14]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.

[15]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.

[16]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.

[17]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.

[18]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.

[19]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.

[20]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.

[21]Taylor, C., Widjaja, M., and Deters, R. W., "Remotely-Processed Vision-Based Control of Autonomous Lighter-Than-Air UAVs with Real-Time Constraints," AIAA SciTech Forum 2025, Orlando, FL. 2025.

[22]Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, Vol. 82, No. 1, 03 1960, pp. 35–45.

[23]Chen, S. Y., "Kalman Filter for Robot Vision: A Survey," *IEEE Transactions on Industrial Electronics*, Vol. 59, No. 11, 2012, pp. 4409–4420.

[24]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.

[25]Cox, B., Dantsker, O. D., and Deters, R. W., "Lighter-Than-Air Vehicle Design for Target Scoring in Adversarial Conditions," AIAA SciTech Forum 2025, Orlando, FL. 2025.

[26]Hoerner, S., *Fluid-Dynamic Drag*, Bakersfield, CA, 1965.