Astralis: A High-Fidelity Simulator for Heterogeneous Robot and Human-Robot Teaming

Gong Chen*, Duong Nguyen-Nam*, Malika Meghjani*, Phan Minh Tri, Marcel Bartholomeus Prasetyo, Mohammad Alif Daffa, Tony Q. S. Quek

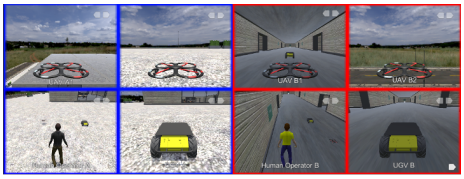
***Abstract*— We introduce Astralis simulator, a high-fidelity robot simulation platform for developing multi-robot and human-robot coordination algorithms that can be translated to real-world environments. It features dynamically initializing virtual environments with real-world 3D point cloud data and virtual random obstacles to create multiple scenario variants for algorithm validation. The simulator controls Unmanned Aerial Vehicles (UAVs), Unmanned Ground Vehicles (UGVs), and human avatars. The simulated robot models are equipped with high fidelity control and navigation capabilities that can be deployed on real robot platforms. We use the simulator to analyze human-robot coordination algorithms for search and rescue missions.**

I. Introduction

Multi-robot system validation with virtual robots in simulated environments is an essential step for the design and development of intelligent systems. Specifically, robot simulators alleviate the expensive hardware costs and logistics of deploying physical robots by providing a simulated environment for safely validating interactions between robots, environmental features, and even human operators [1]. We propose a real-world-based simulator, which can incorporate scanned 3D models of environments and physical robots' dynamic models. For strategic planning and validating the robustness and consistency of the user's algorithms, the simulator can generate random and systematic environmental features (e.g., static and dynamic obstacles) to be placed in the 3D virtual environment.

The primary objective of our simulator is the coordination of heterogeneous robot teams and human-robot teams in dynamic environments. Hence, we consider the physical navigation constraints for two types of robots, the non-holonomic constraints on UGVs and omni-directional motion control for UAVs. For ease of commanding the robots, each robot agent is equipped with fundamental actions to navigate and interact with the human and the environment. A combination of such fundamental actions forms a complex robot behavior, such as a strategic search. The simulator provides a default urban environment consisting of two buildings and an urban outdoor environment illustrated in Fig. 1(a).Each human-robot team consists of 2 UAVs, 1 UGV and 1 human operator as illustrated in Fig. 1(b). The human operator can be controlled by keyboard commands, whereas the robots can be controlled with our highly accessible API or directly via a chat command interface.

(a) Simulated urban environment representing a physical environment with an estimated area of $38\,000\,\mathrm{m}^2$



(b) Human-robot team

Fig. 1: Default environment for Astralis simulator

Our contributions in this paper are: (a) a high-fidelity robot simulator providing realistic graphics, heterogeneous robots, and human avatars for validation and testing of multi-robot and/or human-robot teaming algorithms, (b) a generalized virtual environment initialization, which can incorporate real-world LIDAR point cloud data for generating a digital twin, (c) randomized generation of static and dynamic obstacles for strategic planning and robust validation of both multi-robot and human-robot teaming algorithms, (d) development of fundamental and modular robot commands for building complex behaviors for multi-robot and human-robot collaboration, (e) evaluation of mixed-rule based agent assignment for a simulated target search mission.

II. Simulator Design

The proposed simulator for multi-robot system validation uses Unity, a commercial game engine software, ROS, an open-source platform for robot control, and a custom-designed ROS-Unity bridge. The simulator's design is presented in Fig. 2. Unity was chosen for its easy-to-use 3D environment construction and customization features, while ROS provides realistic robot dynamics models and a navigation stack for seamless algorithm validation. The whole system is containerized with Docker, allowing for cloud deployment.

* These authors contributed equally.

The authors are with Singapore University of Technology and Design (SUTD), Singapore. {chen_gong,mohammad_alif}@mymail.sutd.edu.sg {namduong_nguyen,malika_meghjani,marcel_prasetyo, minhtri_phan,tonyquek}@sutd.edu.sg

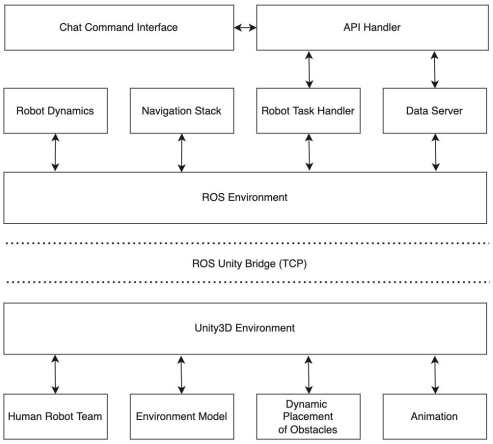


Fig. 2: System architecture of proposed simulator

A. Feature-Rich Unity Environment

Unity allows us to design custom features suitable for simulation purposes such as obstacle placement and modeling of sensors. The human-robot team is initialized in Unity for receiving simulated sensor data from the virtual environment. With a wealth of openly shared assets, we have selected a few human avatars and common objects to be part of the default environment initialization.

1) Simulated Sensors: To allow navigation packages to receive perception data from the simulated environment, we have modeled simulated sensors in Unity. The simulated sensors consist of RGB camera, depth camera, 2D lidar, and IMU. Mesh colliders are implemented on the virtual robots and other objects to ensure collisions can be accounted.

2) Generalized Environment Initialization: To reconstruct a digital twin of real-world scenes in simulated environment, we developed a generalized world initialization pipeline for users to import real-world scanned data (point clouds, laser scan, photogrammetry, etc.). This feature also supports the conversion of CAD models and other 3D model file formats (.igs, .skp, .sldprt, .obj, .glb, etc.) to be imported into Unity as illustrated in Fig 3.

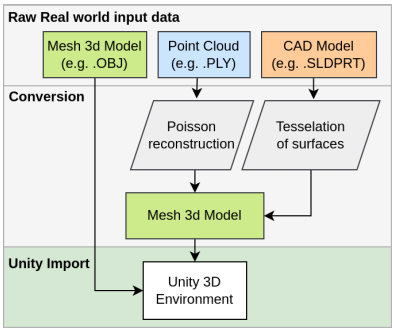


Fig. 3: Real World Data Import Pipeline

3) Dynamic Placement of Obstacles: For robust validations of multi-robot and human-robot coordination algorithms, we develop a dynamic obstacle placement algorithm where non-overlapping obstacles are placed in the free space. This process involves two stages: analyzing the floor normal of the objects to detect the free space regions in the environment and wall tracing for detecting the boundaries of free-space. Users can customize the messiness in the environment and visual attributes of human avatars using a JSON file format. The current list of dynamically placed objects consists of: chairs, cabinets, tables, potted plants, computer desks, and human avatars. Fig. 4 shows examples of messiness index of 0, 0.5 and 1 in a $7\,\mathrm{m}$ x $4\,\mathrm{m}$ office room.

Table I: Fundamental Robot Actions

Agent	Action	Description
UAV	move	move to a place
	hover	hover in place
	search	inward and outward spiral search over a given space
	sweep	vertical and horizontal search for human target
UGV	move	move to a place
	sneak peek	peek into a room
	observe	observe along a corridor

Fig. 4: Examples of different messiness index

B. High-Fidelity ROS Backend

1) High-Fidelity Robot Model: For the simulated UAV and UGV control, we use Firefly [2] and Husky [3] dynamic models, respectively, in ROS. The integration between the dynamic models of the robots with the interactive environment in Unity is generalized in such a way that other wheeled robot models or aerial vehicles for which the ROS packages are available, can also be simulated easily.

2) Fundamental Robot Actions: Since the UAVs and UGVs have their unique navigation capabilities, we develop separate sets of fundamental actions for them to interact with the environment. These fundamental robot actions are described in Table IV.

3) Robot Navigation: We provide a local planner for dynamic obstacle avoidance and a global planner for the obstacle-free path toward a desired goal location for the robots. The UGV uses the open source UGV navigation stack [4]. The UAV uses *voxblox* [5] tool and Skeleton Sparse Graphs Planner (SSGP) [6] for 3D navigation. Our simulator can be easily customized to support other motion planners as long as the respective control packages for the customized dynamic robot models are available.

4) Robot Task Handler: Robot task handler manages and sends high-level commands to each robot. The execution queue follows a first-in-first-out sequence. This handler can also be extended to define different types of tasks. It plays an important role in the system because it helps high-level algorithms to command multiple robots at the same time.

5) Data Server: The data server is built for external applications to get information not only from ROS but also from Unity environment. Users can access the information in the data server via our API handler or ROS service calls.

6) API Handler: For supporting user-defined applications to access data and communicate with internal software components of our simulator, i.e. "Task Handler" and "Data server", the API Handler is designed as a standardized request and response server using ZeroMQ [7]. The information that can be queried via the API is shown in Table II.

Table II: Data Attributes for API Queries

	Attribute	Value	Type
Human Avatar	"activity"	"sitting", "laying", "patrolling", "idle"	string
	"color"	"yellow", "red", "purple", "blue"	string
	"gender"	"male", "female"	string
	"height"	"tall", "medium", "short"	string
	"tiedToChair"	true, false	bool
Location	"location"	[x, y , z]	array
	"messiness_level"	[0,1]	float
	"risk"	[0,1]	float
	"sound"	"minimal", "moderate", "loud"	string
Robot	"type"	"robot_id"	string
	"pose"	[x, y, z]	array
	"command"	user defined action command	string

7) Chat Command Interface: We have also designed a chat interface for users to explicitly give commands to the heterogeneous robot team. The interface can be used to query status and assign navigation or search tasks for each robot as shown in 5. This chat interface is particularly important for sending high-level instructions to coordinate the robot team.

Fig. 5: Usage of chat command interface

C. ROS-Unity Bridge

The ROS-Unity Bridge, which is built using ZeroMQ [7], allows Unity and ROS environments to communicate with each other. The bridge uses publish/subscribe features from the ZeroMQ library. For the implementation of ROS-based coordination algorithms, users can call the ROS-Unity Bridge service to query the robots' state information or assign high-level commands to them.

D. Simulator Capability Benchmarking

We benchmark our Astralis simulator with Nvidia IsaacSim [8], FlightGoggles [9], AirSim [10] and Carla [11]. We qualitatively examine the simulator features in Table III. The key design highlights of our simulator are: a highly customizable environment and dynamic obstacle generation. We simplify the importation of real-world data by presenting a pipeline for users to import custom real-world scans into the simulator directly. The dynamic obstacle generation feature allows us to progressively increase the environment complexity and robustly validate multi-robot and human-robot coordination algorithms. In addition, we can vary the visual attributes of dynamically generated human avatars

Table III: Simulator Capability Benchmarking

Feature	Nvidia IsaacSim	FlightGoggles	AirSim	Carla	Astralis Simulator
High-Fidelity Dynamic Models	Yes	No	Yes	No	**Yes**
Multi Robot Support	UGV	UAV	UAV	UGV	**UGV/UAV**
Environment Types	Indoor	Indoor	Outdoor	Outdoor	**Indoor Outdoor**
Environment Customization	Medium	Low	Medium	Medium	**High**
Real-World Based Environment	No	Yes (limited to pre-existing assets)	No	Yes	**Yes (For any scanned real world data)**
Dynamic Obstacle Generation	No	No	No	Yes	**Yes**
Human Robot Interaction	Keyboard Joysticks	Motion Capture Joysticks	Keyboard Joysticks API	Keyboard API	**Keyboard Joysticks API Chat Interface**
Sensor Support	RGB/D Camera 2D/3D Lidar IMU	RGB/D Camera 1D Lidar IMU	RGB/D Camera 2D/3D Lidar IMU Barometer GPS Magnetometer	RGB/D Camera 3D Lidar IMU GNSS Radar	RGB/D Camera 2D Lidar IMU
Physical World Implementation	Yes	Yes	Yes	No	Yes
3D Environment Engine	Unity3D	Unity3D	Unity3D	Unreal Engine	Unity3D
Containerization	Supported	Supported	Supported	Supported	Supported
Software Licence	Commercial	Open Source	Open Source	Open Source	Open Source

similar to Carla simulator. We note that, though Nvidia IsaacSim and AirSim both have the physical equivalents for robot dynamic models, they either tailor only for UGVs or UAVs. In contrast, our simulator supports heterogeneous robots i.e., both UGVs and UAVs. This fills the gaps not fully addressed by other simulators that we bench-marked against.

III. Mixed Heuristic Task Assignment

We proposed a target search scenario where a team of 2 UAVs, 1 UGV and 1 human operator will be sent into a 3 storey building to search for a potential target while considering several partially known contextual information's of the environment. The building is abstracted into 6 building segments and each segment consists of 3 rooms. For each room to be searched, we define a set of contextual information as risk value and messiness of a room. Risk value, E_{risk}, refers to how safe it is for a human operator to enter and messiness index, $E_{messiness}$, refers to how cluttered a room is based on the randomized furniture generation.

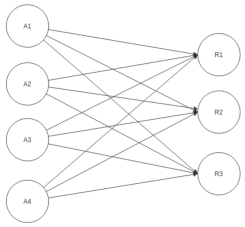


Fig. 6: Bipartite Graph Representation

A. Bipartite Agent-Location Assignment

The agent-to-location pairing is abstracted into a bipartite assignment graph. The effectiveness of the assignments could be quantified as the weighted edge between the vertices in the graph shown in Fig. 6. The reason to choose a bipartite assignment is to avoid repeated pairing.

Given a unidirectional graph $\mathcal{G} = ((A, R), E)$ as a weighted bipartite graph with real weights $w : E \to R$. The agents (A) and rooms (R) are represented by the nodes in the graph. The edge, E, are the weights calculated based on

the contextual information associated with each A_i and R_j. E_{dist} is the normalized distance for the agent to a particular room with respect to the average distance to all the rooms. For the UAVs, $l2$ distance is used. For the UGV and the human, Manhattan Distance is used to compensate for their navigation constraints. E_{risk} is the associated risk value with a room. $E_{messiness}$ is the level of the messiness of a room. Both E_{risk} and $E_{messiness}$ are predetermined with a range of $[0, 1]$. After E is computed, we use a minimum weight full matching in which a matching $M \subseteq E$ with cardinality $|M| = \min(|A|, |R|)$ which minimizes the sum of the weights of the edges included in the matching.

To prevent UGV from going into a highly cluttered room and human from going into a high-risk area, tunable weights, $\mathcal{W}_1$ and $\mathcal{W}_2$, are introduced as a scaling factor on the weight for pairing edges, $\mathcal{W} = [w_{human}, w_{UAV}, w_{UGV}]$. Overall, the total weighted cost, $E^{(i)(j)}$, for each agent, A_i, and room, R_j, pairing is defined as below:

$$E^{(i)(j)} = \frac{E^{(i)}_{dist}}{\frac{1}{n}\sum_{k=0}^{n} E^{(k)}_{dist}} + E^{(i)}_{risk}\mathcal{W}^{(i)}_1 + E^{(i)}_{messiness}\mathcal{W}^{(i)}_2 \quad (1)$$

B. Rule-based Agent-Action Assignment

With reference to the fundamental actions shown in Table IV, we used a rule-based method to determine which action the UAV or UGV should take while performing a search.

Risk is the top priority to consider the action in the room. If the risk is high in the room, UAV will do an inward spiral, as the inward spiral provides a strategic path for the UAV to spiral into the center of the room. If the risk is low for UAV, it will do an outwards spiral to minimize the search time in a room with a low threat. High-Risk means risk value ≥ 0.5, and High-Messiness means messiness index ≥ 0.5. The overall action decision is summarized in the table below

Table IV: Rule-Based Action Assignment

	Low Risk	**High Risk**
Low Messiness	UAV: Outward Spiral UGV: Sneak Peek	UAV: Inward Spiral UGV: Observe Corridor
High Messiness	UAV: Horizontal Sweep UGV: Observe Corridor	UAV: Inward Spiral UGV: Observe Corridor

IV. Experiments and Results

A highlight of our Astralis simulator capabilities with the default two building environment is presented in the following video link: https://youtu.be/jarLqzwSdi0. A highlight of our results for a more complex environment is presented in our open-source repository: https://gitlab.com/marvl-astralis/astralis.

A. Validation of Mixed Heuristic Task Assignment Algorithm

In the experimental analysis section, we primarily examine how to allocate the UGV and the human operator. Our aim in this assignment is twofold: firstly, to reduce the likelihood of assigning the human operator to high-risk areas, and secondly, to minimize the deployment of the UGV in areas with high levels of messiness. An example of the action assignment can be viewed in Fig. 7. With reference to Table IV, as the two UAVs are assigned to high-risk areas, they are tasked with an inward search action. The human operator is not assigned to any location as the nearby UGV is better suited to investigate the area with very low levels of messiness and is thus tasked to perform a sneak peek.

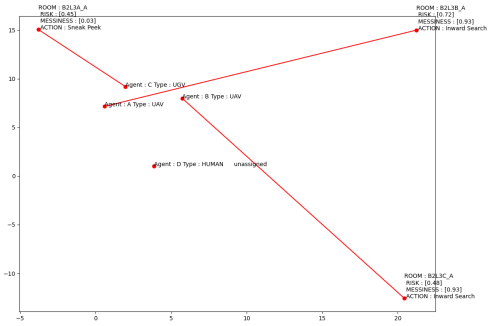


Fig. 7: Bipartite Graph Representation

We generate 1000 proposed scenarios with random initialization of agent position, risk value, and messiness value for each room. $\mathcal{W}_1$ and $\mathcal{W}_2$ from Eqn. 1 empirically tuned to be $[0, 5, 10]$ and $[5, 0, 10]$. Given 4 agents are one-to-one matched to 3 rooms, one agent will always remain unassigned. In our analysis, this unassigned agent could be either the UGV or the human operator. To evaluate the effectiveness of our proposed assignment algorithm, we compare it with a naive assignment strategy that only considers the distances in Table V. By introducing tunable weights $\mathcal{W}$, we significantly reduce the likelihood of assigning a human operator to a high-risk area, from 39.3% to 0.0%, and reduce the likelihood of assigning the UGV to high-messiness areas, from 36.0% to 5.9%. However, it is worth noting that the UGV may still be assigned to high-messiness areas since these areas can also be associated with high risk. As our priority is to minimize the pairing of human operators with high-risk areas, we assign the UGV to such areas regardless of their messiness value.

Table V: Results for 1000 randomly initialized assignments

	Naive Baseline	**Mixed Heuristic Assignment**
Human Assigned to High-Risk Area	39.3%	0.0%
UGV Assigned to High-Messiness Area	36.0%	5.9%

A video of the assignment strategy leading to a target discovery tested in Astralis simulator can be seen in "Scene Two" from https://youtu.be/jarLqzwSdi0.

V. Conclusion

This paper introduces Astralis simulator which is an open-source human-robot teaming simulator. It achieves a functionality comparable performance when compared to the state-of-the-art simulators and has an edge in simulating heterogeneous robot teams, robust algorithm validation and strategic planning. In addition to a default urban environment consisting of indoor and outdoor spaces, we present a novel way for users to import the real-world data into the simulated environment with the functionality of being able to add virtual static and dynamic obstacles. We demonstrated a mixed heuristic task assignment for a simulated heterogeneous robot search mission.

REFERENCES

[1] Sarah Al-Hussaini, Jason M. Gregory, Neel Dhanaraj, and Satyandra K. Gupta. A simulation-based framework for generating alerts for human-supervised multi-robot teams in challenging environments. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 168–175, 2021.

[2] Fadri Furrer, Michael Burri, and Roland Achtelik, Markusand Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[3] Tony Baltovski. Husky: Common packages for the clearpath husky. Available online at: `https://github.com/husky/husky`.

[4] ROS Navigation stack. Available online at: `https://github.com/ros-planning/navigation`.

[5] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373, 2017.

[6] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[7] ZeroMQ. Available online at: `https://zeromq.org/`.

[8] *Nvidia Isaac Sim*. Available online at: `https://developer.nvidia.com/isaac-sim`.

[9] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6941–6948, 2019.

[10] S. Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *FSR*, 2017.

[11] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.