The OpenUAV Swarm Simulation Testbed: a Collaborative Design Studio for Field Robotics

Harish Anand, Zhiang Chen, Sarah Bearman, Prasad Antervedi, Devin Keating,
Stephen A. Rees, Jnaneshwar Das

***Abstract*— In this paper, we describe our OpenUAV multi-robot design studio that enables simulations to run as browser accessible Lubuntu desktop containers. Our simulation testbed, based on ROS, Gazebo, and PX4 flight stack has been developed to facilitate collaborative mission planning, and serve as a sandbox for vision-based problems, collision avoidance, and multi-robot coordination for Unmanned Aircraft Systems (UAS). OpenUAV testbed saves students and researchers from the tedious and complex software setup by providing user-friendly Lubuntu containers equipped with PX4 flight controllers and swarm capabilities. The OpenUAV architecture is built around TurboVNC and noVNC WebSockets technology to seamlessly provide real-time web performance for 3D rendering in a collaborative design tool. We have built upon our previous work that leveraged concurrent multi-UAS simulations, and extended it to be useful for underwater, aerial and ground vehicles. The OpenUAV testbed also enables vehicles in gazebo to have visual realistic camera feeds from Unity Game Engine. Three case studies presented in this paper illustrate the scalability, extensibility, and usability of the latest OpenUAV testbed. The first study focuses on interactive performance analysis of Gazebo running in browser sessions with respect to number of consumers. The second case study demonstrates an extended OpenUAV architecture for swarm simulations through multi-host networking using docker swarm. The third study focuses on student engagement of OpenUAV testbed as part of their robotics course work.**

I. Introduction

The increasing number of use-cases for Unmanned Aircraft Systems (UAS) is accentuating the need for robust controllers and mission planning algorithms. In recent years, model-based techniques have shown impressive aerial robot capabilities such as navigation through dynamic and unknown environments [1]. The AI community has demonstrated promising results in this area by utilizing simulations to learn controllers, often from a model-based reference such as a model-predictive controller [2]. However, single-UAS based methods encounter limitations in tasks such as wide-area mapping, freight transportation, and search and rescue operations. This highlights the need for a simulation framework that provides requisite flight software, photorealism and communication network to accelerate the development of efficient swarm algorithms.

There is also a growing demand for small UAS (sUAS)-based imaging technology to provide high resolution spatial context for data analysis [3, 4].

*This work was supported by NSF award CNS-1521617

Authors are with the School of Earth and Space Exploration, Tempe, AZ USA hanand4,zchen256,sbearman,lanterve,dkeatin2,jdas5@asu.edu and Stephen is a staff at Vanderbilt University, Nashville, Tennessee stephen.a.rees@vanderbilt.edu

Applications like autonomous indoor and outdoor mapping, disaster response, sensor deployment, and environmental monitoring also require software that demonstrates a semantic understanding of the environment. Another application of sUAS-based imaging technology in geology involves generating an orthomosaic map of the region and estimating rock traits such as diameter and orientation [5]. This contributes to the growing demand for sUAS-based imaging technology to provide high resolution spatial context [6] [7]. To extend such UAS based sampling methods to other domains, we need simulation and visually realistic rendering software that help the process of algorithm development and testing.

To address some of the aforementioned needs, the OpenUAV testbed was developed. The purpose of the earlier version of OpenUAV testbed (referred as OpenUAV1) was to reduce the number of field trials and crashes of sUAS by implementing a simulation environment that allows extensive testing and mission synthesis for a swarm of aerial vehicles [8]. In this paper, we describe improvements to OpenUAV1 by enabling interactive use of cloud resources through browsers and visually realistic rendering through Unity [9]. In addition to aerial systems, our testbed can model underwater and ground vehicles. A comparison of OpenUAV1 and the new OpenUAV testbed is mentioned in Table 1.

We believe that hardware abstraction and end-to-end simulation tools will accelerate innovation and education, so we have made the OpenUAV simulator to be globally accessible and easy to use.

II. Related Work

A rich ecosystem of tools exists for UAS hardware and software development. With improved on board computational and sensing capabilities, heterogeneous swarms of ground, aerial, and underwater vehicles will enable efficient exploration missions leveraging diversity and heterogeneity [10]. A variety of tools are available to support design and deployment of single as well as multi-robot systems.

A. RotorS

RotorS is a modular Micro Aerial Vehicle (MAV) simulation framework developed by the Autonomous Systems Lab at ETH Zurich which enables a quick start to perform research on MAVs [12]. RotorS provides virtual machine images that have pre-installed rotorS packages for easy setup and access to the simulator [13]. The OpenUAV testbed has

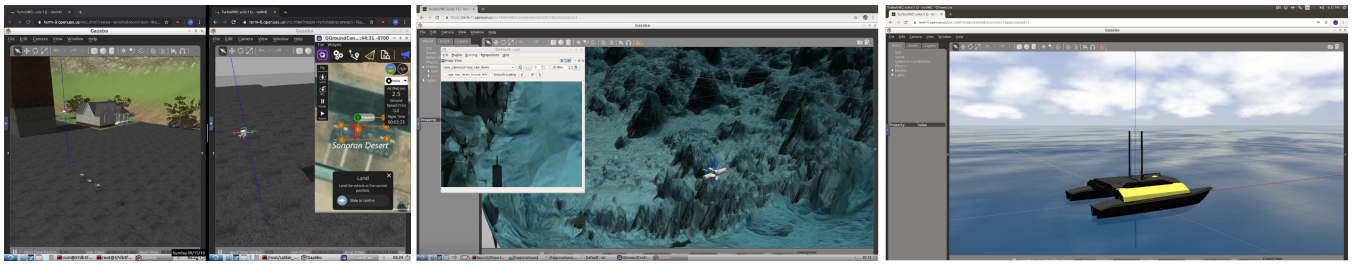

Fig. 1: Example of simultaneous OpenUAV simulation containers with versatile vehicle configurations and worlds. On the left panel, an OpenUAV browser session shows multi-UAV leader follower simulation (left), and a single UAV system controlled using QGroundControl [11] that is running inside the container (right). In center panel, we have an underwater coral reef world, and an underwater quadrotor vehicle. Right panel shows an autonomous surface vessel world.

goals that are very similar to RotorS, but provides these capabilities in a containerized desktop environment. The OpenUAV docker image has additional features like remote web access to a desktop session, support for orchestration tools like docker swarm and ground control software such as QGroundControl.

B. FlightGoggles

FlightGoggles is capable of simulating a virtual-reality environment around autonomous vehicle(s) in flight [14]. When a vehicle is simulated in the FlightGoggles, the sensor measurements are synthetically rendered in real time while the vehicle vibrations and unsteady aerodynamics are captured from the natural interactions of the vehicle. FlightGoggles real time photorealistic rendering of the environment is produced using the Unity game engine. The exceptional advantage of the FlightGoggles framework is the combination of real physics with the Unity-based rendering of the environment for exteroceptive sensors.

Photorealism is an influential component for developing autonomous flight controllers in outdoor environments. Traditional robotics simulators employ a graphics rendering engine along with the physics engine. Gazebo uses Object-Oriented Graphics Rendering Engine (OGRE) and Open Dynamics Engine (ODE) or Bullet physics engine. OpenUAV containers provides users with Unity and OGRE graphics visualization capability with physics simulated through ODE or Bullet physics. OpenUAV testbed aims to be a cloud infrastructure that allows students and researchers to easily setup and simulate aerial, ground and underwater flight dynamics and Unity visualization.

C. AirSim

AirSim is an open-source, cross-platform simulator built on Unreal Engine that offers similar visually realistic simulations for drones and cars [15]. It supports hardware-in-the-loop simulations with flight controllers like PX4 and is compatible with popular communication protocols (e.g. MavLINK). AirSim provides a realistic rendering of scene objects such as trees, lakes and electric poles, which is useful for developing perception algorithms, especially in outdoor environments. However, unlike Gazebo, AirSim requires significant code alterations when transitioning from the simulation environment to real world deployment. Hence, a ROS-Gazebo-PX4 stack is preferred over an Unreal Engine based simulation APIs and network infrastructure. Moreover, the Gazebo simulator uses the more straightforward Unified Robot Description Format (URDF) to create robot models, which can then be generated in Unity using the ROS-Sharp plugin [16]. Hence, we find that our architecture with ROS-Sharp and Unity support can simulate more robots than AirSim's fixed collection of vehicles.

Tools	OpenUAV1 [8]	New OpenUAV
Visualization	GZWEB-DJANGO	Lubuntu VNC
Ground Control Support	No	QgroundControl
Photo-realism	No	Unity
Swarm simulations	Single host	Multi-host
Remote Software development	No	SSHFS support

TABLE I: Comparison of OpenUAV1 and New OpenUAV.

III. Design Goals

The OpenUAV testbed is an on-premise and cloud framework for developing and testing dynamic controllers and swarm algorithms for UAS, Remotely Operated Vehicles (ROV), and Autonomous Underwater Vehicles (AUV) as shown in Fig. 1 center and right panel.

The cloud-based simulation framework of OpenUAV provides remote access to the GPU-enabled machine, thereby facilitating collaborative development and remote demonstrations without the need for separate third-party conferencing software. In order to render the visualizations from Gazebo, a GPU enabled machine is necessary, which would require the researcher to be physically present with machine. This restricts them from providing a live demonstration of their work to remote users, for which they might have to depend on other video conferencing applications. Another disadvantage of this setup is that multiple researchers cannot work on simulations in the same GPU machine simultaneously.

In the following section, we outline the design goals of the improved software architecture for OpenUAV.

A. Goals

As a remotely accessible open source simulation platform, OpenUAV's main purpose has been to lower the barrier to

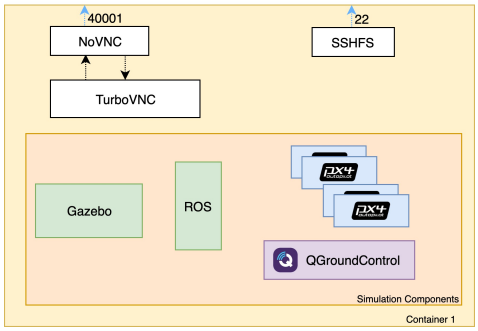

Fig. 2: Components of an OpenUAV container. Softwares like TurboVNC, NoVNC and SSHFS are added to create interactive containers.

entry in research and development of autonomous vehicles. To this end, we aimed to satisfy the following requirements:

- Enable remote desktop accessibility through browsers without compromising on performance.
- Provide an easy to use, software development environment with support for remote code execution.
- Replicate actual resource constraints of vehicles in the simulation containers by having similar memory constraints and computational capacity.
- Minimize the risk of data breach through built-in encryption between client and server. There should not be any additional data encryption and protection layers such as Virtual Private Networks (VPN).
- Provide mechanisms for maintenance through daily build images, regular update releases and data recovery mechanisms.
- Provide visual realistic environments for simulations in on-premise systems.

IV. System Architecture

This section presents the software components of OpenUAV and the interaction between the user and the system. The updated system design was inspired from the work of Will Kessler and the Udacity team [17]. We classified the software components into three categories: simulation components, virtualization components and interactive components. An overview of the single OpenUAV simulation container is shown in Fig. 2.

A. Simulation

Simulation has become a necessity to solve real world problems in a safe and efficient manner. The software packages that enable simulation in the OpenUAV framework are the following.

1) Gazebo: Gazebo is an open-source robotics simulator that is used to design robots, environments and perform realistic rigid body dynamics [18]. Simulated objects have mass, velocity, friction, and other physical properties that enable simulating more realistic behaviors.

2) ROS: The Robot Operating System is a robotics message passing framework that is designed to simplify programming for various robots [19]. ROS is widely used in the robotics community and has several open source packages for sensors and actuators. OpenUAV uses ROS instead of a separate message passing API like in AirSim [15], which enables users to take advantage of the community-developed packages and minimize codes changes when transitioning from simulation environment to real world deployment.

3) PX4: PX4 is an open-source flight control software for UAVs and other unmanned vehicles [20]. It is used to provide basic navigational functionalities like way point navigation, landing, taking off, and hovering. Additionally, the latest version of PX4 provides higher level capabilities like collision avoidance in UAVs. Communication and control between the user's ROS code and the UAS is simplified by MAVROS, which is a ROS package that enables communication between user's ROS code and UAS with MavLINK communication protocol. This provides the user with perception and motion control capabilities of the UAV through ROS topics.

4) QGroundControl: QGroundControl is a software package used for monitoring and mission planning for any MavLINK enabled drone [11]. QGroundControl communicates with PX4 inside the simulation and provides data-logging capabilities which are invaluable for analysing and evaluating the simulations.

B. Unity Game Engine

Unity is a cross-platform game engine developed primarily for the development of 2D, 3D and VR games [9]. Scenes rendered in Unity are photo-realistic because it has the necessary art assets to simulate material properties like shadows, specularity, emissivity, shading, and texture to fine-tune objects in the scene. Unity also has algorithms for occlusion culling, which disables the rendering of objects that are not currently seen by the camera. Our need for these capabilities inspired us to support Unity in the OpenUAV testbed but without replacing the Gazebo physics engine.

The OpenUAV uses ROS-Sharp to add URDF robot models in Unity, and to communicate between the container and the Unity game engine. A ROS connector plugin inside the container publishes the Pose and Twist of the gazebo models and which is then rendered in Unity. The robot models in Unity have kinematics, but lack any Unity based physics or collision effects. The OpenUAV testbed doesn't currently have docker support for Unity, and therefore Unity runs on the host machine and communicates with the simulation via port 9090 as shown in Figure 4. In figure 3, we demonstrate the Unity support to the OpenUAV testbed by rendering volcanic plumes as seen by the two vehicle cameras.

C. Virtualization

In OpenUAV, we have operating-system-level virtualization to deliver software as packages called containers [21]. Fig. 4 shows how each of these components interact with each other.

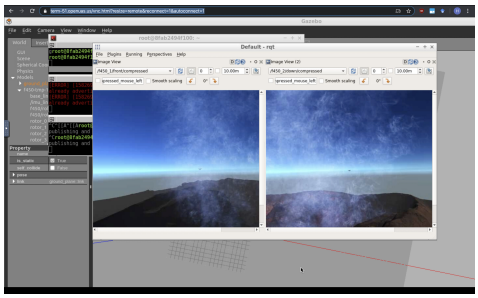

Fig. 3: A team of two UAS vehicles orbiting around a volcanic plume. The camera topics inside the container are generated from Unity scene, and the two UAS has the physics simulated from Gazebo's ODE engine. This simulation is being used to test mapping and sampling strategies for a volcanology study.

1) Docker: Container technology is rising as the preferred means of packaging and deploying applications. Docker can package software and its dependencies into a lightweight container. Out of the box, Docker provides isolation from the host as well as from other containers. It also improves the security of the application by restricting the possible host system calls, providing network, process, file name-space and running applications in the least privileged mode [22, 23]. Our architecture is not dependent on which orchestration tools we use, like LXC or docker, as these are runc and NVIDIA containers.
Another advantage of containers is its ability to configure resources allocated to each container. These restrictions are useful for replicating actual vehicle compute power in the simulation containers by having similar memory constraints and compute capacity.

2) Docker Swarm: Docker swarm is an open-source container orchestration software that allows users to manage a group of docker daemons running on multiple host machines. Docker swarm achieves the above capability through its overlay network [24]. OpenUAV architecture can allocate a container with PX4's software-in-the-loop and MAVROS ROS nodes in one host machine, and use overlay network to communicate across a GPU enabled host running Gazebo simulator. This provides the capability of distributing the SITL and MAVROS swarm workload across different machines in cloud while a single host machine displays Gazebo work-space. The disadvantage of using this architecture is that there is a high latency between the SITL and Gazebo communication in comparison to running them on same machine.

3) Kubernetes: Kubernetes is an open-source software for automating deployment, scaling, and management of containerized applications [25, 26]. OpenUAV architecture was tested in both Docker Swarm [27] and Kubernetes. One of the architectural goals of OpenUAV is to seamlessly provide access to multiple simulation containers in cloud or local GPU-enabled machine. Thus, sharing of GPU resources among the pool of deployable containers (pods) is necessary. Kubernetes pods are advantageous when compared to individual containers since a group of containers (pods) working together is capable of achieving complex tasks [28].

4) Nginx: Nginx is a web server which can also be used as a reverse proxy, load balancer and HTTP cache [29]. OpenUAV utilizes Nginx as a reverse proxy and load balancer for accessing the remote desktop sessions. Each remote desktop session running inside a container is given a simulation ID (whole number), and the access to that session uses the following URL pattern `term-<ID>.openuas.us`. Nginx is used as a TCP streaming proxy for the openssh server running inside the container. Streaming ports are opened on the OpenUAV server based on user requirements. This feature enables users to remotely develop software and execute code.

D. Interactive components

In the following section, we describe the software components that enable the users to interact with the simulations running in containers. Some of the technologies are used in remote computing and High Performance Computing (HPC) simulation services [30, 31].

1) TurboVNC: When used with VirtualGL, TurboVNC provides a high performing and robust solution for displaying 3D applications over all types of networks [32, 33]. OpenUAV utilizes TurboVNC's 3D rendering capability to display remote frame buffer (Lubuntu desktop session) associated with the container to any connected viewers.

2) NoVNC: NoVNC is a JavaScript VNC application that provides VNC sessions over the web browser [34]. NoVNC follows the standard VNC protocol and has support for persistent connection through WebSockets. We use the NoVNC client to connect with the TurboVNC server running inside the containers. The NoVNC displays the VNC session over port 40001, which docker exposes at port 40xx, where xx is the simulation ID. Nginx proxies 40xx to the outside world over unique terminals with same simulation ID `term-<ID>.openuas.us`. NoVNC also provides built-in encryption between client and server, so it does not require additional layers of protection such as Virtual Private Network (VPN). A user with a VNC client can also connect directly to the TurboVNC VNC session. This requires exposing the VNC session over Nginx as a stream proxy. Another advantage of NoVNC is that it enables users to easily switch between machines while still being presented with the same desktop (i.e. each application stays exactly as they left it).

3) Secure Shell FileSystem (SSHFS): Secure Shell FileSystem allows you to mount a remote file system using Secure Shell File Transfer protocol (SFTP) [35]. Through SSHFS, the OpenUAV testbed provides an easy remote file access mechanism to simulation directories by having it mounted as a remote filesystem in your local machine. An SSH server running inside the container can also act as a remote code execution environment, which is useful for Integrated Development Environments like PyCharm [36].

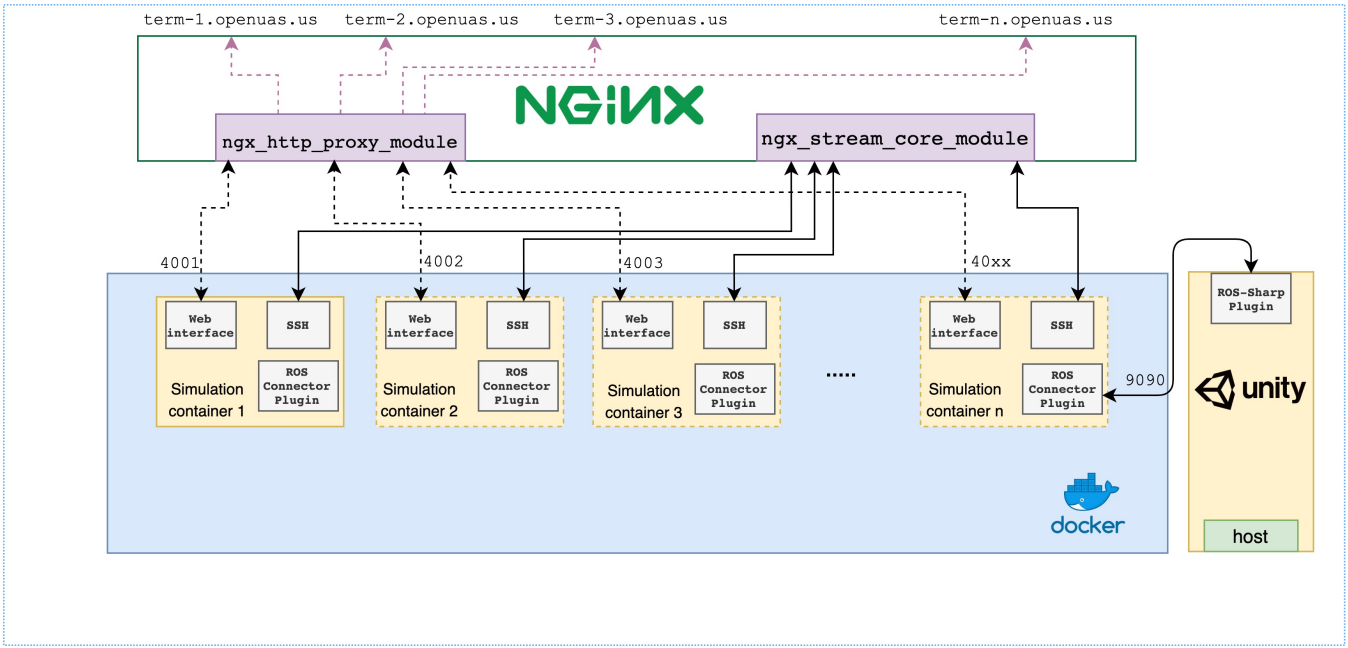

Fig. 4: The virtualization components of the OpenUAV testbed. Docker forwards each container's web interface as 40xx ports where xx denote the simulation ID. Nginx proxy and stream modules are used to proxy multiple HTTP sessions and stream SSH session to outside users. Unity game engine runs on the host and communicates to docker container via port 9090.

V. Case Studies

A. *Performance evaluation of large simulations in OpenUAV*

The OpenUAV architecture leverages the docker containerization tool to generate a large number of simulations. A user interacts with the OpenUAV simulation through a NoVNC configuration, which has a higher latency of visual throughput when compared to actual display device. Interactive response time is the amount of time it takes for a recorded change in mouse and keyboard to be displayed on the user interface. Since OpenUAV is a VNC based desktop environment, we evaluate the performance of the interactive response time instead of throughput, as throughput is not an appropriate measure of GUI or user desktop performance. Similar performance analysis have been done for other GUI applications like Microsoft Powerpoint and multimedia applications [37, 38].

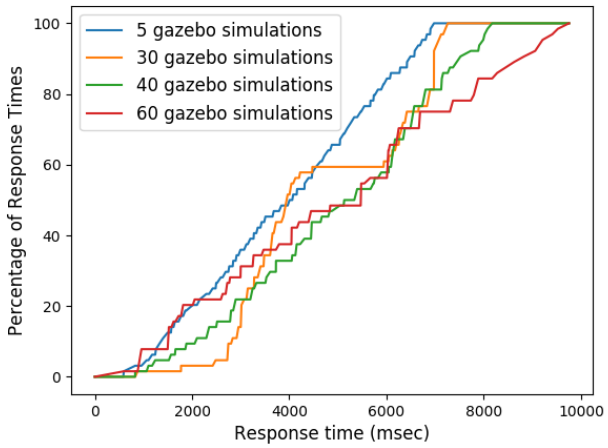

Fig. 5: A cumulative distribution function plot of interactive response time of an empty Gazebo world in OpenUAV container under different conditions.The lines shows the number of empty world gazebo simulations that were active during the test.

We evaluate the performance of the OpenUAV architecture on a 64 core Intel(R) Xeon(R) CPU E5-2683 v4 2.10GHz and RTX 2080 Ti GPU on-premise desktop. We evaluated the interactive performance of Gazebo empty world using vncplay software [37]. The origin of the empty world in gazebo was translated and rotated using mouse movements as part of the GUI performance tests. These tests were then replicated across different loads on the server. For instance, the blue line in the Fig. 5 represents the case when 5 gazebo empty world simulations were running, and for 70% of the time the interactive response was within 5224 ms. An important thing to note here is that the interactive response time is different from a computer throughput, and is dependent on the user's mouse speed during the recording. We find there is a gradual increase in interactive response time as the number of simulations increase (when y axis is 100% in Fig. 5). The large response time for 60 gazebo simulations indicates that some frames were skipped.

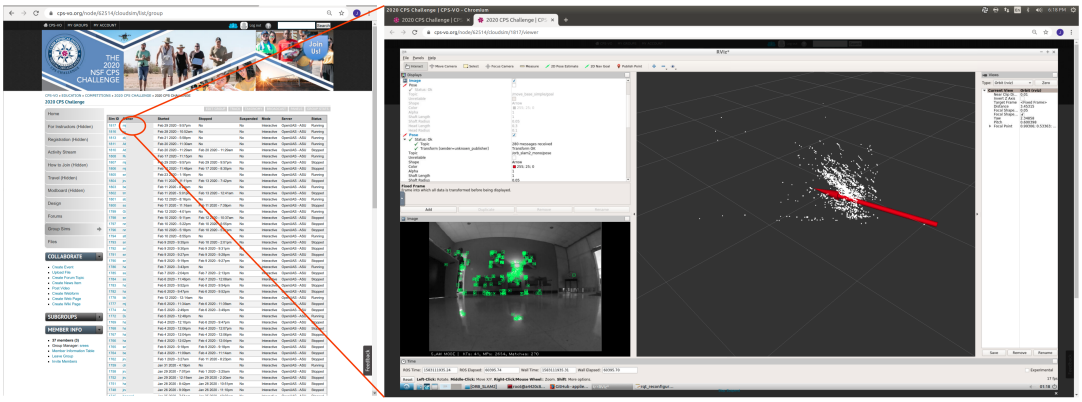

Fig. 6: Student engagement of OpenUAV testbed deployed on CPS-VO, for a course assignment on Simultaneous Localization and Mapping (SLAM) method evaluation on benchmark datasets. Login names have been partially masked in the above figure to maintain user anonymity.

B. Extending swarm simulations using Docker swarm

The OpenUAV testbed has swarm simulations setup that were developed as part of the initial OpenUAV1 project [8]. These swarm simulations include leader-follower, formation and motion control of multiple aerial vehicles. The swarm simulation creates ROS nodes for PX4-SITL, Gazebo and MAVROS for each individual UAS. By leveraging the docker swarm's overlay multi-host network, OpenUAV can now distribute each vehicle's SITL simulation across multiple host containers. This is suitable for cloud infrastructure services like Google Compute Engine and Amazon EC2 instances where multiple simulations are performed in the local area network.

The distributed swarm simulation requires a single gazebo session running on GPU enabled host0 as shown in Fig. 7. ROS master and vehicles are spawned in host0. Their network is shared through docker swarm overlay network across multiple hosts. This modified simulation architecture is more suited for cloud infrastructure than on-premise machine as the latency between multiple hosts can result in frequent vehicle crashes. An obvious advantage of using this architecture is its capability to simulate a large number of vehicles in a distributed compute platform.

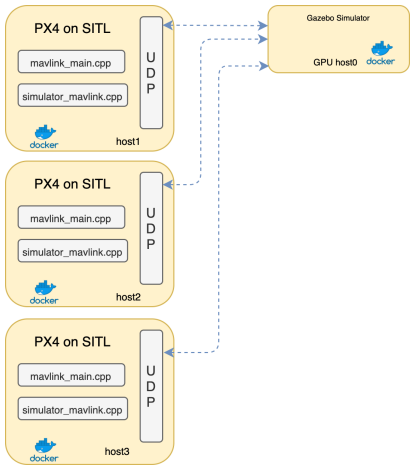

Fig. 7: Architecture on distributed PX4-SITL ROS nodes across multiple machines through docker swarm's overlay network (Blue dotted lines).

C. CPS-VO integration for education

The OpenUAV testbed is used by 30 students as part of a Robotics course in Arizona State University (ASU) [39]. Cyber-Physical Systems Virtual Organization is a collaboration among CPS professionals in academia, government, and industry [40]. Each student creates an account in CPS-VO page and uses it as the authentication to create simulation containers in the OpenUAV testbed. The testbed has an average of 10 active gazebo simulations every day. In order to maintain a judicious usage of CPU/GPU resources on the machine, the CPS-VO lets students suspend a simulation and resume back from the saved simulation. 'Suspend' and 'Resume' functions in CPS-VO are implemented using the docker's pause and unpause feature. A single OpenUAV server can natively handle multiple CPS-VO like integrations through different Nginx server blocks. Moreover, the OpenUAV testbed saves the student's container images every night as a backup to avoid loss of data. Fig. 6 right demonstrates a student's engagement of OpenUAV testbed as part of the coursework. Instructors can view every student's simulations through CPS-VO dashboard in Fig. 6 left.

VI. Conclusion

In this paper, we presented OpenUAV as a scalable, extensible, and user-friendly simulation testbed that remedies users from time-consuming and complex software setup by providing a containerized simulation stack. We described the OpenUAV testbed's architecture and performance analysis on robotic tools like Gazebo that run inside the containers. We extended OpenUAV's architecture to incorporate distributed computation capability for swarm simulations and presented academic use cases through CPS-VO integration.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation award CNS-1521617.

REFERENCES

[1] Antonio Loquercio et al. "Dronet: Learning to fly by driving". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1088–1095.

[2] Aviv Tamar et al. "Learning from the hindsight plan—episodic MPC improvement". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 336–343.

[3] Philipp Lottes et al. "UAV-based crop and weed classification for smart farming". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3024–3031.

[4] Nassim Ammour et al. "Deep learning approach for car detection in UAV imagery". In: *Remote Sensing* 9.4 (2017), p. 312.

[5] Zhiang Chen et al. "Geomorphological Analysis Using Unpiloted Aircraft Systems, Structure from Motion, and Deep Learning". In: *arXiv preprint arXiv:1909.12874* (2019).

[6] *ICRA Workshop 2019 on Algorithms And Architectures For Learning In-The-Loop Systems In Autonomous Flight*. 2019. URL: https://uav-learning-icra.github.io/2019/ (visited on 09/14/2019).

[7] Lukas Vacek et al. "sUAS for deployment and recovery of an environmental sensor probe". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 1022–1029.

[8] Matt Schmittle et al. "Openuav: A UAV testbed for the CPS and robotics community". In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. 2018, pp. 130–139.

[9] *Unity Game Engine*. 20202. URL: https://unity.com/.

[10] Amanda Prorok, M Ani Hsieh, and Vijay Kumar. "Formalizing the impact of diversity on performance in a heterogeneous swarm of robots". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 5364–5371.

[11] E Zurich. *Qgroundcontrol: Ground control station for small air land water autonomous unmanned systems*. 2013. URL: http://qgroundcontrol.com.

[12] Fadri Furrer et al. "RotorS—A modular Gazebo MAV simulator framework". In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[13] ETHZ-ASL. *RotorS is a MAV gazebo simulator.* 2018. URL: https://github.com/ethz-asl/rotors_simulator.

[14] Winter Guerra et al. "FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality". In: *arXiv preprint arXiv:1905.11377* (2019).

[15] Shital Shah et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[16] ROS Sharp. *ROSSharp is a set of open source software libraries and tools for communicating with ROS to Unity.* URL: https://github.com/siemens/ros-sharp (visited on 03/01/2020).

[17] Will Kessler. *Building a GPU-enhanced Lubuntu Desktop with nvidia-docker2*. 2018. URL: https://github.com/willkessler/nvidia-docker-novnc.

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

[19] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[20] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 6235–6240.

[21] Carl Boettiger. "An introduction to Docker for reproducible research". In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 71–79.

[22] Paolo Di Tommaso et al. "The impact of Docker containers on the performance of genomic pipelines". In: *PeerJ* 3 (2015), e1273.

[23] Docker. *What is a Container?* URL: https://www.docker.com/resources/what-container.

[24] Jouni Mäenpää and Gonzalo Camarillo. "Analysis of delays in a peer-to-peer session initiation protocol overlay network". In: *Proceedings of the 7th IEEE conference on Consumer communications and networking conference*. 2010, pp. 16–21.

[25] David Bernstein. "Containers and cloud: From lxc to docker to kubernetes". In: *IEEE Cloud Computing* 1.3 (2014), pp. 81–84.

[26] Kubernetes community. *Kubernetes*. URL: https://kubernetes.io/ (visited on 09/14/2019).

[27] Docker community. *Docker Swarm mode overview*. URL: https://docs.docker.com/engine/swarm/ (visited on 09/14/2019).

[28] CoreOS community. *CoreOS*. URL: https://coreos.com/kubernetes/docs/latest/pods.html (visited on 09/14/2019).

[29] Will Reese. "Nginx: the high-performance web server and reverse proxy". In: *Linux Journal* 2008.173 (2008), p. 2.

[30] Karin Meier-Fleischer, Niklas Röber, and Michael Böttinger. "Visualization in a Climate Computing Centre". In: *EGU General Assembly Conference Abstracts*. Vol. 16. 2014.

[31] University of Cambridge. *Connecting to CSD3 via TurboVNC*. URL: https://docs.hpc.cam.ac.uk/hpc/user-guide/turbovnc.html (visited on 09/14/2019).

[32] Lien Deboosere et al. "Thin client computing solutions in low-and high-motion scenarios". In: *International Conference on Networking and Services (ICNS'07)*. IEEE. 2007, pp. 38–38.

[33] TurboVNC. *A Brief Introduction to TurboVNC*. URL: https://turbovnc.org/About/Introduction (visited on 09/14/2019).

[34] Joel Martin et al. *noVNC: HTML5 VNC Client*. 2015. URL: https://novnc.com/info.html.

[35] Matthew E Hoskins. "Sshfs: super easy file access over ssh". In: *Linux Journal* 2006.146 (2006), p. 4.

[36] JetBrains. *PyCharmThe Python IDE for Professional Developers*. URL: https://www.jetbrains.com/pycharm/.

[37] Nickolai Zeldovich and Ramesh Chandra. "Interactive Performance Measurement with VNCPlay." In: *USENIX Annual Technical Conference, FREENIX Track*. 2005, pp. 189–198.

[38] Bert Vankeirsbilck et al. "Platform for real-time subjective assessment of interactive multimedia applications". In: *Multimedia tools and applications* 72.1 (2014), pp. 749–775.

[39] CPS-VO. *2020 NSF Student CPS Challenge*. URL: https://cps-vo.org/group/CPSchallenge.

[40] Cyber-Physical Systems Virtual Organization. *Cyber-Physical Systems Virtual Organization*. URL: https://cps-vo.org.