2018 9th ACM/IEEE International Conference on Cyber-Physical Systems

OpenUAV: A UAV Testbed for the CPS and Robotics Community

Matt Schmittle[1], Anna Lukina[2], Lukas Vacek[3], Jnaneshwar Das[3],
Christopher P. Buskirk[4], Stephen Rees[4], Janos Sztipanovits[4], Radu Grosu[2], and Vijay Kumar[3]

***Abstract*— Multirotor Unmanned Aerial Vehicles (UAV) have grown in popularity for research and education, overcoming challenges associated with fixed wing and ground robots. Unfortunately, extensive physical testing can be expensive and time consuming because of short flight times due to battery constraints and safety precautions. Simulation tools offer a low barrier to entry and enable testing and validation before field trials. However, most of the well-known simulators today have a high barrier to entry due to the need for powerful computers and the time required for initial set up. In this paper, we present OpenUAV, an open source test bed for UAV education and research that overcomes these barriers. We leverage the Containers as a Service (CaaS) technology to enable students and researchers carry out simulations on the cloud. We have based our framework on open-source tools including ROS, Gazebo, Docker, PX4, and Ansible, we designed the simulation framework so that it has no special hardware requirements. Two use-cases are presented. First, we show how a UAV can navigate around obstacles, and second, we test a multi-UAV swarm formation algorithm. To our knowledge, this is the first open-source, cloud-enabled testbed for UAVs. The code is available on GitHub: https://github.com/Open-UAV.**

I. INTRODUCTION

Unmanned aerial vehicles (UAVs), specifically the multirotor platform, have been rapidly growing in popularity in robotics and cyber-physical systems research. Multirotors are UAVs with four or more rotors enabling hovering and maneuvering similar to a helicopter, but with added stability and a simplified electro-mechanical configuration [1]. While fixed wing and helicopter UAVs do not have the precision or footprint necessary for tasks such as close-range inspection, or smooth videography, multirotors are compact, allow high precision control, and can hover, enabling execution of such tasks. With improved on board computational and sensing capabilities, these platforms are being used for increasingly complex missions, and hardware abstraction and end-to-end simulation tools will accelerate innovation and education as they allow for more time to be spent designing the algorithms than on implementation.*

*Supported by the NSF CPS Virtual Organizations Active Resources initiative

[1]M. Schmittle, Department of Computer Science, University of Delaware

[2]A. Lukina and R. Grosu, Cyber-Physical Systems Group, Technische Universitat Wien

[3]J. Das, L. Vacek, and V. Kumar, GRASP Lab, University of Pennsylvania, Philadelphia, PA 19104, USA.

[4]Christopher P. Buskirk, Stephen Rees,and Janos Sztipanovits, Vanderbilt University

*http://usblogs.pwc.com/emerging-technology/lowering-the-barrier-to-innovation-in-robotics/

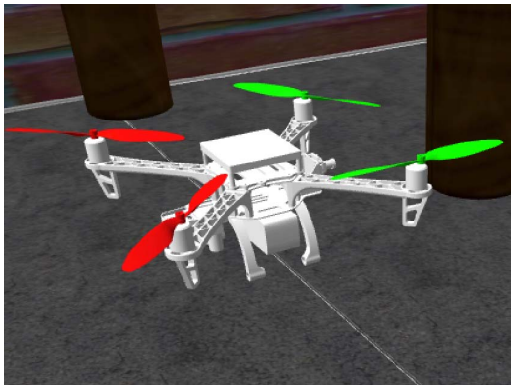


Fig. 1: A simulated DJI F450 multirotor as seen on a browser, enabled through Gzweb on the OpenUAV simulator.

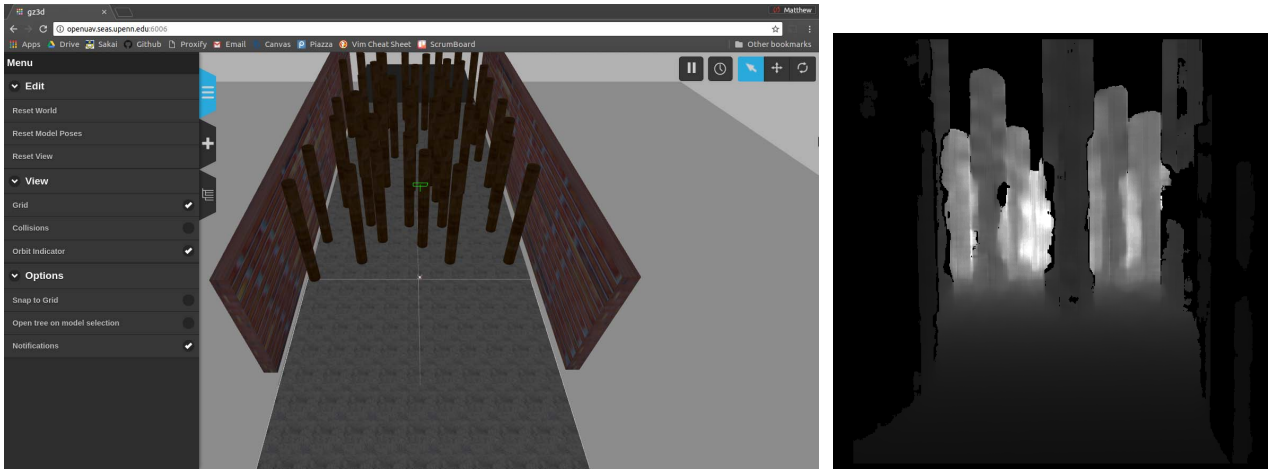
Within the robotics community, the Robot Operating System (ROS)* has achieved extensive adoption. ROS simplifies control through a message passing interface, which helps reduce errors and accelerates innovation. Simulators, such as Gazebo,* are also essential for developing safety-critical robotics systems. Gazebo provides an open-source simulator to test algorithms and controls before moving to a real robot.

For UAVs, tools such as the open-source PX4 project and QGroundControl have already created autopilots that can simplify control and basic flight of a multirotor. To make programming easier, the ROS package MAVROS* has been created to communicate with PX4. MAVROS allows the user to control the multirotor at a higher level, so the programmer can focus on algorithmic implementation and not basic flight. To develop automated control, task-specific multirotors, and additional future development of UAVs, new tools and simulators are needed [2].

Currently, to develop UAV software or conduct UAV experiments, the developer or researcher starts with simulation and then moves to real robots. Figure 2 shows an example simulation environment. For simulation, the user (i.e., developer or researcher) usually works with the popular tools, ROS and Gazebo, with ROS to communicate with the simulated robot(s) in Gazebo. To run visualization, the user could use Gazebo or rviz.* ROS is ideal for simulation as it can port directly to a real robot, requiring little change to run real life testing. The complexity of the simulated

*http://www.ros.org/about-ros/
*http://gazebosim.org/
*http://wiki.ros.org/mavros
*http://wiki.ros.org/rviz

0-7695-6378-3/18/$31.00 ©2018 IEEE
DOI 10.1109/ICCPS.2018.00021



(a) UAV at center of axes pointing forward. The world is for an obstacle avoidance machine learning task. Visualization is through Gzweb. The left side is a tool bar of actions that the user can take to change the world or camera.

(b) Depth map computed from the UAV's stereo camera images. Obstacles shown at varying depths correspond to trees in (a).

Fig. 2: Snapshots of OpenUAV simulation interface.

environment (e.g., number of objects, lighting, collision checking) and the number of UAVs used, will determine the power of the computer necessary for simulation.

The process of setting up these tools requires proficiency in UNIX (or Linux) systems and access to a powerful desktop computer.* This barrier to entry can inhibit researchers and stagnate innovation of field systems. In education, these barriers are more prevalent. Students interested in learning about UAVs may lack Linux knowledge, and there may be limited access to a powerful computer for an entire class. Thus, these barriers to entry often slow research and limit education.

In this paper, we describe OpenUAV, an open-source, web-based simulation testbed designed specifically for UAVs. To our knowledge, this is the first open-source, cloud-enabled testbed for UAVs. By using a cloud-based simulation as seen in Fig. 2, automated by Ansible,* we relieve the users of prior Linux knowledge for setup and expensive computer hardware requirements, which together reduce the barrier to entry. The simulation testbed is based on the popular PX4 autopilot. Through creating an open source simulation testbed with little barrier to entry, we seek to increase UAV research and education. To demonstrate effectiveness of our testbed, we showcase state-of-the-art machine learning and multi-UAV simulations by using tools from the ROS and the Open Source Robotics Foundation (OSRF) community. OpenUAV can be used for a variety of UAV applications, such as aerial phytobiopsy [3], or sensor probe deployment and recovery [4].

Specifically, this paper makes the following contributions:

- the design and implementation of OpenUAV, a cloud-based simulation testbed for UAVs with low barrier to entry,
- two case studies demonstrating the use of OpenUAV for machine learning and multi-UAV swarms, and
- an observational study of the use of OpenUAV with lessons learned and opportunities for future improvement.

II. UAV Research and Education Needs

UAV research has exploded in the past 10 years, evidenced by the increase in publications at conferences including the International Conference on Intelligent Robotics and Systems (IROS) and International Conference on Robotics and Automation (ICRA), and increased commercial startups, including Exyn Technologies* and Skydio.* Specifically, multirotors have seen massive growth in development due to their applicability to a broad range of tasks such as search and rescue [5], inspection [6], photography [7], and monitoring [8]. Progress has been made in UAV research also. High speed navigation [9], multi-UAV swarms [10], and aerial tracking [11] using simulation and robotics tools have all seen progress. The necessary simulation and robotics tools require new researchers to install ROS and Gazebo, have a strong understanding of Linux, and have a computationally capable computer to get started in research. These requirements raise the barrier to entry, and, more importantly, slow research.

While almost every computer science department offers a robotics course today, there are only a few examples of known courses in UAVs. For example, the University of Nevada offers a course called Introduction to Aerial Robotics [12]. Of the specialized robotics courses offered, it

*http://www.gazebosim.org/tutorials?tut=guided_b1&cat=#Systemrequirements
*https://www.ansible.com/
*https://exyntechnologies.com/
*https://www.skydio.com/

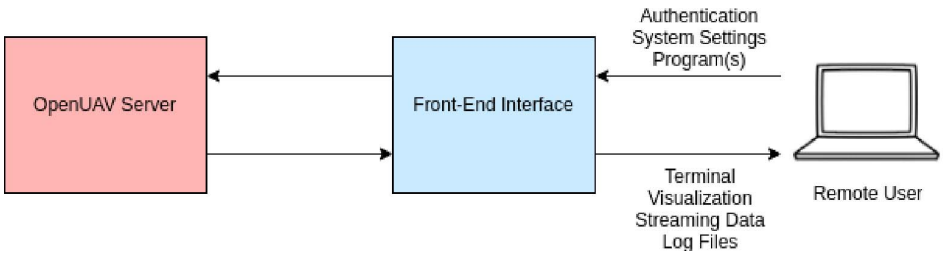


Fig. 3: An overview of the OpenUAV testbed. The user (white) interacts with the front-end interface (blue), which calls the necessary scripts to run the program with the user settings on the back-end OpenUAV server (red).

is common for them to be only at the graduate level, directed toward training researchers.

The lack of UAV courses is mainly due to the cost of multirotors and replacement parts, and the time required to setup and become familiar with a multirotor programming interface. Using simulators for teaching is an obvious choice to mitigate costs. If students can learn about multirotors through simulation, then they are free to make mistakes without catastrophic costs. However, current simulators have a high barrier to entry. Students taking such a course will be in principle less familiar with robotics than researchers, but need to work with the same complicated tools that researchers need to master. One of the goals of our OpenUAV work is to minimize the problems associated with current simulation environments to make exploring UAV development more accessible to a wider audience and to encourage the development of more courses on UAVs.

III. Related Work

Gzweb was developed to provide cloud connectivity to the popular Gazebo simulation.* Gzweb is a WebGL client for Gazebo. With Gzweb, users can visualize and interact with the simulation in their browser. While this is a useful tool, Gzweb and Gazebo do not provide enough support for a full testbed for UAV research and development.

Robot Web Tools [13] offers a suite of open source visualization and robot interaction tools. It is intended to allow the user to create web-based robotics applications through its libraries and tools. This is a good platform to develop web-based robotics, but it is not a full development and simulation environment in itself. While each of these tools are good for general robot development, OpenUAV is greater than the sum of its parts by integrating its tools in an easy to use manner.

Another similar commercial tool has been created for ROS education and development called The Construct.* The Construct is a web development environment for ROS-controlled robots that offers a shell, ROS Tools, simulation environment, instructional lessons, and the capability to connect to a real robot. While an effective tool, The Construct limits free users to slow machines and is not open source. Also, it focuses on ROS education specifically, and thus is better for general robotics education than UAV education.

AirSim [14] is a UAV and car simulator based on Unreal engine. AirSim is similar to Gazebo in that it is a simulation environment and not a testbed. It also integrates with PX4 and is based off of Unreal Engine which provides a photo-realistic simulation. Some limitations of AirSim are its lack of ROS support and lack of cloud connectivity making Gazebo the prefered simulator for OpenUAV. While AirSim is not currently implemented into the OpenUAV platform it is a possible direction for future work.

IV. The OpenUAV Testbed

We designed OpenUAV to meet the following requirements:

- The code, visualization, tools, and models/environments must all be accessible from the cloud.
- The testbed should be a deployable system for system independence and future cloud computing.
- The back-end simulation should provide communication to a user through a front-end interface.
- The system should support multiple simultaneous users, where each user simulation should be contained to prevent interference between simulations.
- Each simulation should be controllable in an easy to manage manner.
- The simulation should require little to no setup for the user beyond experimental parameters.
- The user should experience simulation speeds and responsiveness similar to running their simulation locally.

A. System Architecture

Figure 3 depicts the overall system architecture of OpenUAV. The three major components are the OpenUAV server that hosts the simulation, the front-end interface hosted on a separate computer, and the communication between the user, front-end interface, and OpenUAV server. We describe each of these three components and their technologies in the following subsections.

B. OpenUAV Server Component

The OpenUAV Server component is responsible for running the simulations and reporting back the visualization and

*https://www.docker.com/what-docker
*http://gazebosim.org/gzweb
*http://www.theconstructsim.com/

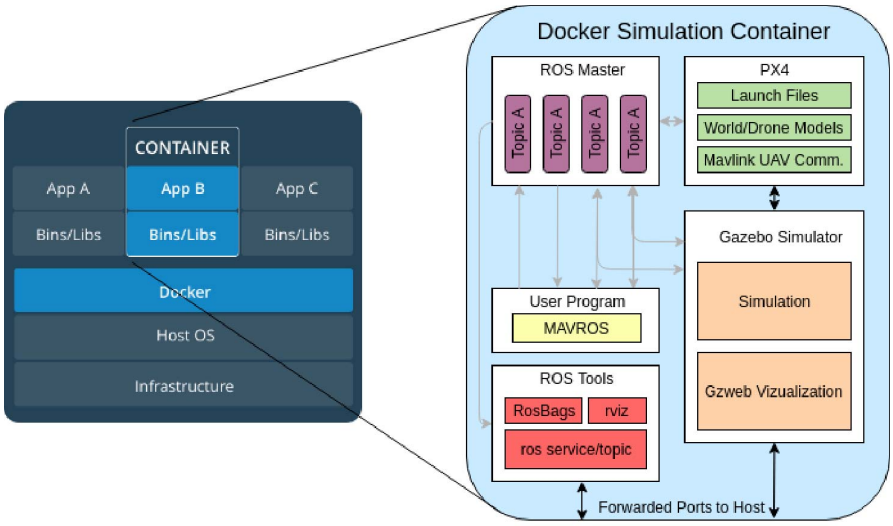


Fig. 4: A visual representation of containers: A, B, and C. Docker is more lightweight than a virtual machine because it only copies the files necessary to run the application, not a whole operating system into each container.*The right shows an instance of the OpenUAV docker image. ROS Master is the central point for all internal communication through ROS topics. Gray arrows represent publishing and subscribing to ROS topics. Also shown are the ROS tools and simulator connecting to the host machine for cloud access through forwarded ports.

simulation data to the front-end interface. The server component addresses the requirements of deployability, support for multiple users, ease of use, speed, and little setup. These requirements are primarily satisfied by the use of Docker and Ansible.

1) Simulation Core: This component is the core of the testbed. It was built using several technologies, including ROS, Gazebo, PX4, and MAVROS. The main components of the simulation core are ROS and Gazebo. ROS is a robotics message passing framework that is designed to simplify programming various robotics platforms. Each part of a robotics system has a ROS node. From that node, it can post and receive messages labeled by their topic. So a program can post to different topics instead of connecting to each part through a different API. ROS is used in the user program for control and sensing of the robot and simulation environment. It also provides the log data and additional live monitoring for all messages being passed.

Gazebo is a robotics simulator that can be used for indoor and outdoor applications. It comes with a physics engine, simulating the dynamical properties of objects as well as inter-object collisions. Gazebo simulates aerodynamics through its LiftDragPlugin.* Due to the computational limitations of solving complex fluid dynamics for each model, Gazebo applies all of the forces to the object links directly in accordance to the physics of lift and drag. It also provides a cloud visualization tool, Gzweb, and its own ROS topics. We use Gazebo to simulate the robot and environment, and Gzweb to visualize and manually interact with the simulation environment. Synchronization between Gazebo and ROS is mangaged through the shared ROS clock. Together ROS and Gazebo make the basic framework of a robotic simulation.

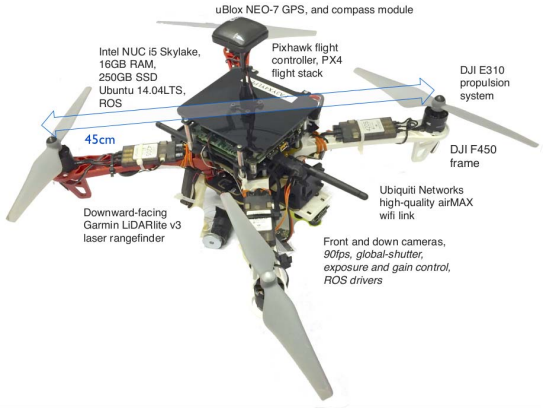


Fig. 5: The DJI F450 airframe and Intel NUC i5 based multirotor UAV model used in the OpenUAV simulator.

PX4* and MAVROS add to the ROS and Gazebo core by incorporating a UAV-specific interface. PX4 is an open source autopilot and hardware platform. It is used to provide basic functions such as hovering, way point following, landing/takeoff, and reading sensor data. MAVROS is a ROS package that simplifies the communication and control between the user and the UAV by converting ROS messages to MAVLink protocol.* This allows a user to interact with the UAV through ROS topics.

*http://gazebosim.org/tutorials?tut=aerodynamics&cat=plugins

*http://px4.io/about-us/

*http://qgroundcontrol.org/mavlink/start

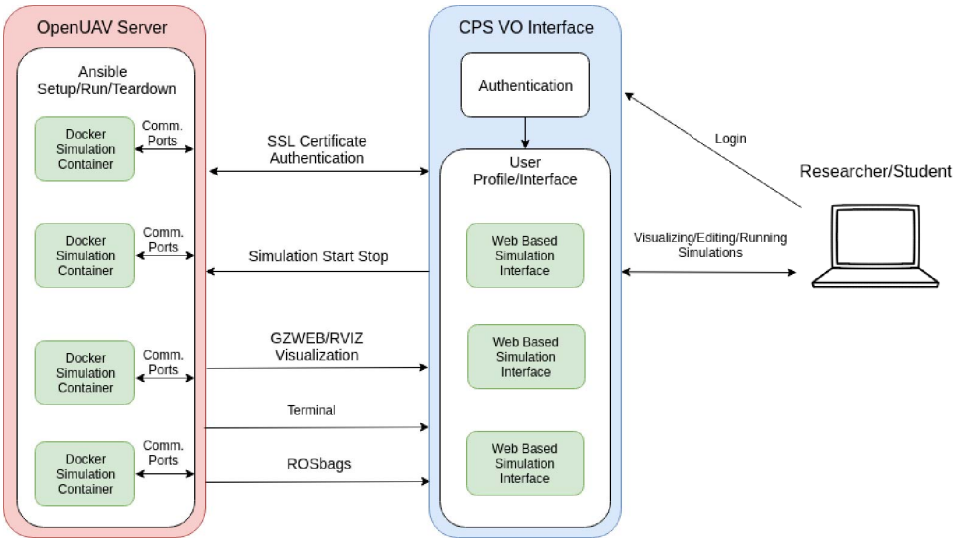


Fig. 6: The communication between the OpenUAV server, CPS-VO webpage, and Researcher/Student. As shown, Ansible is managing each Docker container, and the containers are connecting to the host machine through the forwarded ports. The CPS-VO page is controlling communicating with the OpenUAV server through calling Ansible scripts and the server is sending back visualization, ROSbags, and a terminal connection.

PX4 and Mavros reduce the gap between simulation and real world testing by providing software in the loop (SITL) and hardware in the loop (HITL) simulations that work with Gazebo. SITL allows the user to test with code that would work the same on a real UAV, and HITL runs that code on the UAV computer, but simulates on the testing computer so as to test computation time.

This work is focused on SITL as the user is on a remote machine, and the testbed would not be scalable if we had to have a series of HITL computers connected to the simulation machine. We were able to add to the basic simulation our own DJI Flame Wheel F450 frame based multirotor as seen in Figure 5 that is used in the annual NSF CPS UAV Student Challenge 2016* and expand the system for our own testing described later.

2) Containing Simulations Using Docker and Ansible: We designed OpenUAV to isolate each user simulation using the Containers as a Service (CaaS) paradigm. To do so, we use Docker, a popular CaaS platform.* Simulations using a CaaS platform offers multiple advantages. First, it eliminates interference between simulations, allowing multiple users to run multiple scenarios in parallel. Second, by provisioning powerful servers, simulations can scale through an arbitrary number of containers constrained only by the resources of the server. Finally, using tools such as Ansible, complex simulation (i.e., container) orchestration can be carried out.

The left side of Figure 4 shows a visual representation of docker containers. Docker provides a lightweight virtual environment separate from the host machine (in our case, the OpenUAV server) for running simulations. It can connect to the host machine through forwarded ports that can be forwarded to other servers. Docker images are blueprints of the environment that can be instantiated into containers. Images are created through scripts called dockerfiles.

The right side of Figure 4 shows an instance of our docker image. We created a docker image by developing a custom dockerfile for the OpenUAV simulation that has all of the packages, tools, and dependencies pre-installed. This docker image with pre-installed software achieves the isolation of the simulation testbed. From there, we forwarded ports for visualization and monitoring.

Each container instance created from the docker image is used for one simulation and then destroyed. The creation, running, and tear down of each simulation is automated by Ansible. Ansible is an automation system that has been designed to work well with docker containers. Similar to shell scripting, Ansible utilizes the YAML programming language to make scripts easy to run and read. Ansible allows the front end interface to interact with the containers through running simple scripts.

C. Front-End Interface

The front-end interface is responsible for user authentication and clean interfacing with the simulations. The front-end interface satisfies the cloud accessibility requirements of the testbed as it is hosted on the Cyber-Physical Systems Virtual Organization (CPS-VO)* website. The CPS-VO is a virtual organization from academia and industry with the goal of growing the knowledge about cyber-physical systems. Fig 7 is a view of the visualization through the front-end interface.

*https://cps-vo.org/group/CPSchallenge
*https://www.docker.com/what-docker

*https://cps-vo.org/

D. Communication Architecture and CPS-VO Interface

The main novelty of this UAV testbed is that it is cloud enabled. This allows the user to design and test on any platform from anywhere. The communication architecture is responsible for satisfying the requirement of providing a connection to the back-end interface through the CPS-VO. While the back end can be accessed over ssh, ssh is not a scalable secure form of access. It also adds a learning curve to students who are new to Linux. To add security, scalability, and simplicity, we are in the process of connecting the back-end simulation with the CPS-VO interface.

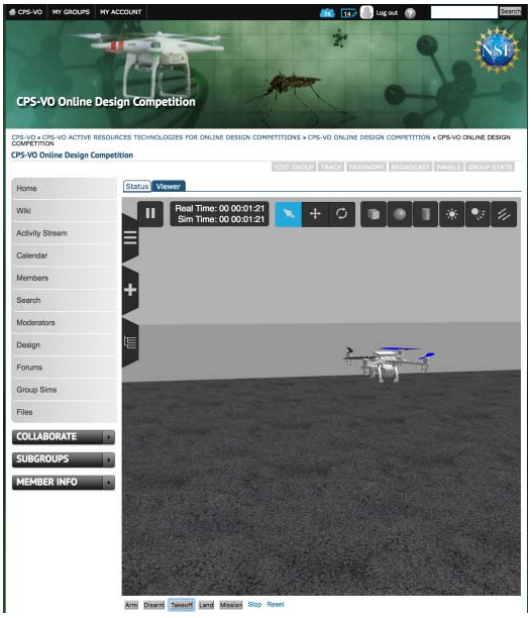


Fig. 7: The CPS-VO allows users to access the cloud-based OpenUAV simulation stack after authenticating into the VO website.

Fig 6 shows the communication between the back-end OpenUAV server and the front-end interface. The communication occurs in two stages. First, there is communication from a Docker container to the host machine, OpenUAV server, running Ansible. Second, there is communication that occurs from the host machine, OpenUAV server, to the CPS-VO interface and back. To provide security, SSL encryption is used with user authentication on the front end. The simulation is controlled through a call to an Ansible script by the VO interface. Visualization during the simulation is sent back through Gzweb. Post simulation, ROSbags, which are ROS logs, will be sent to the user for evaluation. Future work will include adding tools similar to rviz and a shell in the interface for debugging.

V. System Testing Through Case Studies

To test a simulation testbed, the important factors to consider are computational load, user interface, simulation limitations, and popular use cases. To evaluate the effectiveness of OpenUAV meeting our requirements, we chose computationally demanding test scenarios, among the many potential use cases for OpenUAV. Specifically, we implemented a machine-learning based UAV navigation test scenario and a multi-UAV swarm formation test scenario. We chose these scenarios as they were more computationally challenging, current research areas in our lab, and were more likely to find flaws than a simple UAV task. In addition, the multi-UAV test was done by a user not involved with the OpenUAV simulator development, further showing its usability.

Specifically, with the multi-UAV task, we were able to determine how easy it is to setup and control multiple UAV's with our modified PX4 control. With the machine learning task, we were able to test how effective Tensorboard is as a cloud-enabled monitoring tool and the impacts of machine learning on simulation responsiveness to the user. We report on our observations from these two implemented test scenarios using OpenUAV in the following section.

A. UAV Machine Learning Challenge

Due to machine learning's growing popularity and applications, we wanted to test the simulation stack on a set of machine learning tasks. We decided to focus on neural networks specifically because they present unique challenges that a simulator must be able to address, and we wanted to explore performing a demanding machine learning task for the simulator. In addition, we could leverage current work on UAV navigation [15].

Machine learning is used for solving problems that may not necessarily have a clear answer but have access to training data [16]. There are many different machine learning algorithms for solving different problems such as classification or image segmentation. Neural networks specifically are good at generalizing from training data for a diversity of tasks [16]. The network used is a deepQ network, which means the network is the Q-function in Q-learning. It takes in some state and outputs Q values, or expected benefit, for each of the potential actions. We are able to implement this neural network through the TensorFlow [17] library.

The first challenge of neural networks is that they must have a large set of training data. This means that the simulator must run continuously for long periods of time. The second challenge is that a diverse training set is needed to generalize well. To create this training set, the environment must change dynamically within each episode [18].

To examine machine learning-based UAV simulations under different scenarios, we tested three different scenarios: (1) a homing task where the agent was told where the target is, (2) a visual servoing task where the agent had to home in on an apriltag below it, and (3) an obstacle avoidance task where the agent had to navigate a forest-like environment. To set up a diverse environment in scenarios (1) and (2), the target moved after each episode. In scenario (3), the network was saved to switch to a new environment once it completed one environment.

We were able to monitor the agents' progress continuously

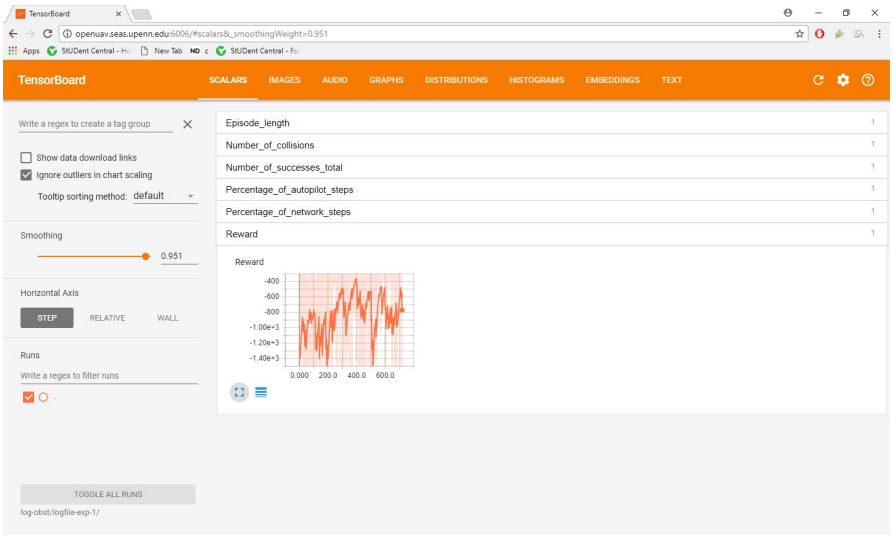


Fig. 8: The Tensorboard browser interface from the obstacle avoidance simulation.

over the Internet via TensorBoard.* Tensorboard allows constant live monitoring of learning statistics. In addition, it allows users to view raw data and network structures. Tensorboard is hosted inside the simulation container, and gets data by reading from a continuously updated log file. The main challenge of Tensorboard was to make sure it was accessible from outside the container. We were able to achieve this through a forwarded port to the host machine. Figure 8 shows the Tensorboard browser interface from the obstacle avoidance simulation.

Our focus was on creating an effective training simulator and then a good policy for the task. We focused on what was required of the simulator for machine learning. The most important thing we found was the ability to reset upon failure or success. On the simpler tasks, such as servoing, only the target had to be reset. We found resetting on the obstacle avoidance task to be much more challenging, in part, because PX4 was not primarily designed to work in simulated environments. The PX4 position estimator had no way of being reset which caused a multirotor position reset to the origin make the multirotor move aggressively towards its old waypoint before the position estimator caught up with the actual location. The position estimator uses a Kalman filter [19], which, for a large value change, takes a series of samples before it will shift to the correct value. We overcame this by landing prior to a position reset, then waiting for the position estimator to catch up with the actual location before taking off. Task (1) was successfully learned to home on a target. The visual servoing and obstacle avoidance tasks needed more work on the learning side to perform well. A video of the basic obstacle avoidance can be found here: https://youtu.be/hQ5Pend41VM. Future work is needed to speed up the simulation beyond real time and early error detection via continuous integration.

*https://www.tensorflow.org/get_started/summaries_and_tensorboard

B. Multi-UAV Swarm Formations

The goal of the multi-UAV case study was to determine how easy it is to setup and control multiple UAV's with our modified PX4 control. PX4 was not created to control multiple UAV's. We modified PX4 to simulate multiple UAV's, each controlled independently. We chose a use case for plan optimization with multiple UAVs. In this case, MATLAB is used to synthesize a plan that brings n agents from an arbitrary initial configuration to a so-called V-formation. This formation has been proven to be energy efficient for birds on long-distance flights. The vortex generated near the wingtips of a bird offers an upwash benefit for the other following birds but only given they avoid downwash formed directly behind the bird as depicted in Figure 9. The same principle has been exploited by aircraft for fuel conservation on military flight missions. In [20], the authors model bird-flocking behavior as a stochastic process and propose an approximation algorithm for plan synthesis. The approach combines model-predictive control with particle swarm optimization to find a sequence of best actions at each step.

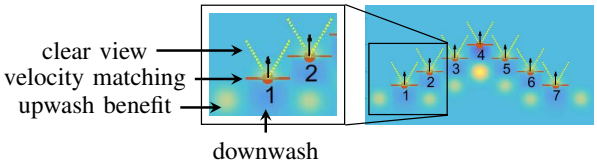


Fig. 9: Optimal positions in V-formation: clear view, velocity matching, and upwash benefit for each bird, except for the leader of the flock which does not receive the upwash benefit.

Field experiments with Crazyflie 2.0 Nano Quadcopters*

*https://www.bitcraze.io/crazyflie-2/

using Vicon Motion Capture System* highlighted that testing of the planning algorithms for the drone swarms has to be performed in a simulation environment prior to deployment due to the challenges posed by the real-world environment, as well as sensor and actuator noises. However, achieving a formation of drones requires emulating the dynamics as close as possible to reality to identify the risks of collisions or interferences between multiple UAVs.

We chose V-formation for the multirotors as planning of the close formation flights for them should be downwash-aware as well as for winged vehicles. The MATLAB computations were performed offline and passed to the low-level controller as a set of waypoints. The current state of the art relied heavily on the environmental conditions and smoothness of the trajectories, which limited the experiments to three drones and wide formations. In contrast, OpenUAV allowed us to simulate larger flocks and study the performance of the algorithm on more challenging starting configurations.

Experiments in [20] were executed using a MATLAB setup that produced a sequence of matrices as an optimal plan. Each matrix consists of positions for each agent at each step of the plan. Such matrices can be fairly easily passed to an OpenUAV controller in Python and serve as waypoints for the drones. Based on a given set of points the multi-UAV control further computes accelerations for all robots to follow the plan. To use the OpenUAV stack, we wrote a node that subscribes to a multi-UAV pose array published by the Matlab code. Then, the OpenUAV ROS node publishes the desired position and yaw (set as a fixed value) to the PX4 position controller on each UAV. The simulation can be monitored during a mission on Gzweb (Figure 10), and a rosbag file can be collected afterwards for analysis. Naturally, since aerodynamic considerations are not addressed in the OpenUAV stack, it serves as a way for testing and debugging the system before actual field trials.

Visualizing a realistic course of events in case the drones are at risk of colliding, the OpenUAV simulator helps detect collisions that can be caused by close formations, which may not appear risky in offline Matlab calculations. Additionally, injecting stochastic external noise as an attack on each robot allows to play controller-attacker games, where the control input is to be synthesized online as the best response to attacks on-the-fly [21]. Future computations will be done in C++ to enable direct publishing to the PX4 position controller. Moreover, it is planned to introduce the downwash generated by each UAV into the simulation model to study its effect on the optimal solutions.

VI. Analysis of Case Study Results

In this section, we report on our observations of lessons learned from the two case studies of OpenUAV testing.

In the machine learning tasks, we found that overall OpenUAV worked effectively to simulate and train a UAV on machine learning tasks. The main roadblocks to machine learning for UAVs that we experienced were the difficulty with resetting the UAV and the time required to train. While it is possible to circumvent PX4's limitations with positional reset, time requirements are harder to fix. We found that speeding up the simulation caused erratic behavior, rendering speedup useless.

*https://www.vicon.com/motion-capture/engineering

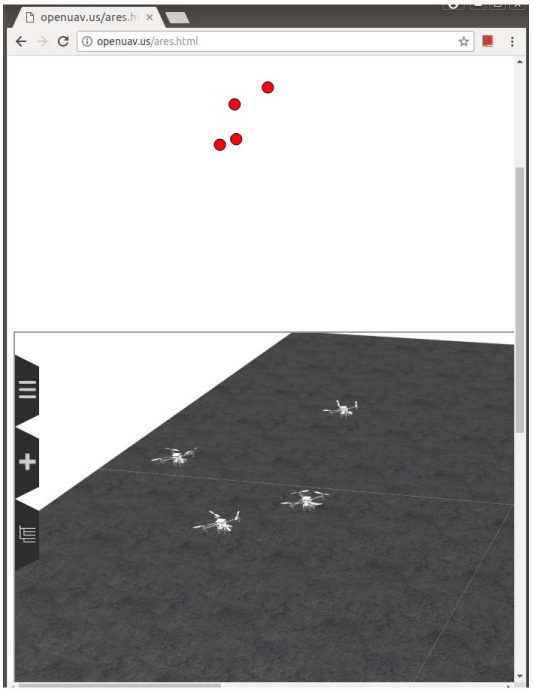


Fig. 10: Screenshot showing a simplified 2D visualization (top), as well as Gzweb view of four UAVs in formation on the OpenUAV simulator.

In the multi-UAV task, we found that OpenUAV was able to use more challenging initial multirotor configurations and simulate on larger flocks than previous approaches. Since the multi-UAV computation was in MATLAB, we were able to test its ability to integrate with OpenUAV. The main limitation to MATLAB was that it is proprietary software, which has two ramifications: (1) We could not integrate it onto the simulation server. (2) The user has to feed the data in or communicate it from their local machine. For the purpose of testing, we chose to feed the data in post calculation. We also found that while the simulation ran smooth, visualization over Gzweb for more than one multirotor was lagged, because Gzweb renders the simulation on the user's machine. Future work to simplify the multirotor model or visualize through RobotWebTools could help speed up visualization for multi-UAV swarms.

Both case studies demonstrated that OpenUAV can serve as an effective testbed for UAV simulation. While the case studies were effective on the testbed, both lacked a simple user interface because the communication to the front-end interface was under development. The users had to connect through ssh while also viewing Gzweb, and Tensorboard on

separate urls. Despite this, users connected remotely, little installation was required, and no local computer requirements for the remote user were necessary for the tests. In addition, multiple simulations could run simultaneously with no interference.

VII. Load Testing for Multiple Users

In order for OpenUAV to be used for education, it must support multiple users simultaneously. The current hardware setup for OpenUAV is a simulator server setup with a 32 core Xeon CPU, with 64 GB DDR3 RAM, and 2 Titan X GPU's. Each user simulation is in its own container and for testing we had UAV's flying in patterns running basic vision algorithms from their onboard cameras. We found that each individual container can efficiently handle up to 6 UAV's and the whole system can handle 30 UAV's with no cameras and 9 with camera's. For future work, we would like to move OpenUAV to a scalable cloud computing solution so as to handle more users and UAV's.

VIII. Conclusions and Future Work

In this paper, we have presented OpenUAV, the first cloud-enabled, open source simulation testbed for UAV research and education. Based on open source software, we offer this simulator free for public use towards reducing the cost of research and education, and to promote further development of the simulator. By being cloud enabled, we have lowered the barrier to entry to UAV development and research by not requiring specific hardware or complicated setup. By using Docker and Ansible, we have made the simulation deployable and workflow automated, for a potential push to a cloud computing service. We have also demonstrated effective use of this testbed for computationally challenging machine learning and multi-UAV simulations successfully including testing done by a user not involved with development.

Future work will focus on implementing the designed connection between the testbed and the front-end interface, so as to even further reduce the barrier to entry and to provide cleaner authentication to the system. In addition, we seek to expand the basic testbed to encompass more multirotor platforms and environments for a variety of potential tasks. Finally, we plan to take advantage of Ansible to incorporate continuous integration into the system.

ACKNOWLEDGMENT

We gratefully acknowledge NSF grant CNS-1521617, USDA grant 2015-67021-23857 under the National Robotics Initiative, a gift from Microsoft Research, the Doctoral Program Logical Methods in Computer Science and the Austrian National Research Network RiSE/SHiNE (S11412-N23) project funded by the Austrian Science Fund (FWF) project W1255-N23 for supporting this work.

References

[1] G. Shweta, P. Infant Teenu Mohandas, and J. Conrad, "A survey of quadrotor Unmanned Aerial Vehicles," pp. 1–6, 03 2012.

[2] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, 5 2015.

[3] D. Orol, J. Das, L. Vacek, I. Orr, M. Paret, C. J. Taylor, and V. Kumar, "An aerial phytobiopsy system: Design, evaluation, and lessons learned," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2017, pp. 188–195.

[4] L. Vacek, E. Atter, P. Rizo, B. Nam, R. Kortvelesy, D. Kaufman, J. Das, and V. Kumar, "sUAS for deployment and recovery of an environmental sensor probe," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2017, pp. 1022–1029.

[5] S. Waharte and N. Trigoni, "Supporting Search and Rescue Operations with UAVs," in *Proceedings of the 2010 International Conference on Emerging Security Technologies*, ser. EST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 142–147. [Online]. Available: http://dx.doi.org/10.1109/EST.2010.31

[6] C. Deng, S. Wang, Z. Huang, Z. Tan, and J. Liu, "Unmanned Aerial Vehicles for Power Line Inspection: A Cooperative Way in Platforms and Communications," *JCM*, vol. 9, no. 9, pp. 687–692, 2014. [Online]. Available: http://dblp.uni-trier.de/db/journals/jcm/jcm9.html#DengWHTL14

[7] S. Hamilton and J. Stephenson, "Testing UAV (drone) aerial photography and photogrammetry for archeology," Lakehead University, Tech. Rep., 03 2016.

[8] T. F. Villa, F. Gonzalez, B. Miljievic, R. Zoran D., and L. Morawska, "An Overview of Small Unmanned Aerial Vehicles for Air Quality Measurements: Present Applications and Future Prospectives," *Sensors*, vol. 16, no. 7, 2016. [Online]. Available: http://www.mdpi.com/1424-8220/16/7/1072

[9] B. Lopez and J. How, "Aggressive 3-D collision avoidance for high-speed navigation," *IEE/RSJ International Conference on Robotics and Automation (ICRA)*, 07 2017.

[10] M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno, and V. Kumar, "System for deployment of groups of unmanned micro aerial vehicles in gps-denied environments using onboard visual relative localization," *Autonomous Robots*, vol. 41, no. 4, pp. 919–944, Apr 2017. [Online]. Available: https://doi.org/10.1007/s10514-016-9567-z

[11] J. Thomas, J. Welde, G. Loianno, K. Daniilidis, and V. Kumar, "Autonomous Flight for Detection, Localization, and Tracking of Moving Targets With a Small Quadrotor," vol. PP, pp. 1–1, 05 2017.

[12] A. Kostas, "CSE 491/691: Introduction to Aerial Robotics," 2016. [Online]. Available: https://www.unr.edu/Documents/engineering/cse/S16-Syllabi/CS491-Syllabus-S16.pdf

[13] T. Russell, K. Julius, L. David, L. Jihoon, C. J. Odest, S. Osentoski, W. Mitchell, and S. Chernova, "Robot Web Tools: Efficient Messaging for Cloud Robotics," *IEE/RSJ International Conference on Intelligent Robotics and Systems (IROS)*, 2015.

[14] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[15] M. Schmittle and C. Rasmussen, "Exploring DeepQ Learning for Micro UAV Tree Avoidance," 2017, iROS-Abstract Only.

[16] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[17] "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[19] L. Kleeman, "Understanding and Applying Kalman Filtering," 2006. [Online]. Available: http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf

[20] A. Lukina, L. Esterle, C. Hirsch, E. Bartocci, J. Yang, A. Tiwari, S. A. Smolka, and R. Grosu, "ARES: adaptive receding-horizon synthesis of optimal plans," in *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017*, ser. LNCS, vol. 10206, 2017, pp. 286–302. [Online]. Available: https://doi.org/10.1007/978-3-662-54580-5_17

[21] A. Tiwari, S. A. Smolka, L. Esterle, A. Lukina, J. Yang, and R. Grosu, "Attacking the V: on the resiliency of adaptive-horizon MPC," in *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, ser. Lecture Notes in Computer Science, D. D'Souza and K. N. Kumar, Eds., vol. 10482. Springer, 2017, pp. 446–462.