

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

Harish Anand, Zhiang Chen, Jnaneshwar Das

Abstract—In this paper, we showcase a multi-robot design studio where simulation containers are browser accessible Ubuntu desktops. Our simulation testbed, based on ROS, Gazebo, PX4 flight stack has been developed to tackle higher-level challenging tasks such as mission planning, vision-based problems, collision avoidance and multi-robot coordination for Unpiloted Aircraft Systems (UAS). The new 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, airship and ground vehicles. This opens up the possibility for both rigorous Monte Carlo styled software testing of heterogeneous swarm simulations, as well as sampling-based optimization of mission parameters. The new OpenUAV architecture has native support for ROS, PX4 and QGroundControl. Two case studies in the paper illustrate the development of UAS missions in the latest OpenUAV setup. The first example highlights the development of a visual-servoing technique for UAS descent on a target. Second case study referred to as terrain relative navigation (TRN) involves creating a reactive planner for UAS navigation by keeping a constant distance from the terrain.

I. INTRODUCTION

With an increasing number of use-cases for unpiloted aircraft systems (UAS), the need for robust controllers and mission planning algorithms are far exceeding available engineer or scientist time. The AI community has demonstrated promising results in this arena by utilizing simulations to learn controllers, often from a model-based reference such as a model-predictive controller. [1] In past years, model-based techniques have successfully shown impressive aerial robot capabilities like quick navigation through unknown environments. To achieve real autonomy in applications such as indoor or outdoor mapping, disaster response, sensor deployment and environmental-monitoring, UAS software must demonstrate a semantic understanding of the environment. [2] [3]

The OpenUAV testbed was developed to reduce the number of field trials and crashes of sUAS systems by implementing a simulation environment that allows extensive testing and mission synthesis for a swarm of aircraft [4]. In this paper, we describe improvements to this testbed, enabling interactive use of cloud resources on a browser. In addition to aerial systems, OpenUAV framework can model underwater and ground vehicles.

There is a growing demand for sUAS-based imaging technology to provide high resolution spatial context for

data analysis [5, 6]. An example application of sUAS-based imaging technology in geology involves generating orthomosaic of a region and estimating rock traits such as diameter and orientation. To extend such UAS based sampling methods to other domains, we need software tools that help in algorithms development and testing. Single UAS based methods encounter limitations in tasks such as wide-area mapping, freight transportation, and cooperative search and recovery. Therefore, there is a need for a simulation framework that provides flight stack and communication network to develop efficient swarm algorithms.

With our belief that hardware abstraction and end-to-end simulation tools will accelerate innovation and education, 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 [7]. A variety of tools exist to support design and deployment of single as well as multi-robot systems.

A. RotorS

RotorS [8] is a Micro Aerial Vehicles (MAV) Gazebo [9] simulator developed by the Autonomous Systems Lab at ETH Zurich. It provides UAV models such as the AscTec Hummingbird, the AscTec Pelican, or the AscTec Firefly. There are sensors such as an inertial measurement unit (IMU), odometry sensor and the visual-inertial-sensor, which can be mounted on these models.[10] RotorS provide virtual machine images that have pre-installed rotorS packages for easy setup and access to the simulator. OpenUAV framework has goals that are very similar to rotorS. However, it provides similar capabilities in a containerized desktop environment. The improved OpenUAV docker image has additional features like remote web access to a desktop session, support for orchestration tools like Kubernetes and mission planning software QGroundControl.

B. FlightGoggles

FlightGoggles [11] is capable of simulating a virtual-reality environment around autonomous vehicle(s) in flight. When a vehicle is simulated in FlightGoggles, the sensor measurements are synthetically rendered in real time while the vehicle vibrations and unsteady aerodynamics

*This work was supported by NSF award CNS-1521617

Authors are with the School of Earth and Space Exploration, Tempe, AZ USA hanand4,zchen256,jdas5@asu.edu

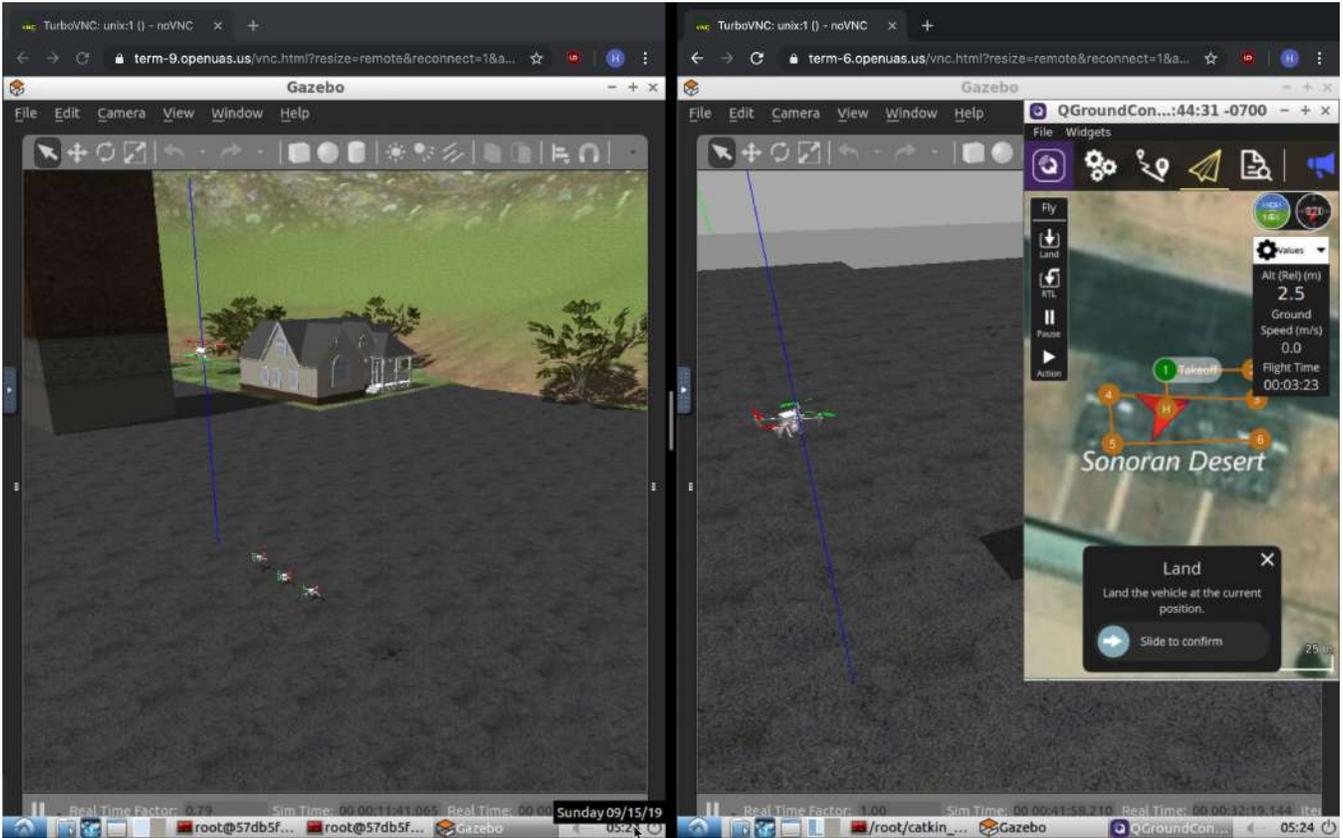


Fig. 1: Example of simultaneously accessing OpenUAV simulation containers. On the left web browser, we have an OpenUAV container with ID 9 accessible over the url `term-9.openuas.us`. Term-9 is executing a multi-UAV leader follower simulation. On the right web browser, we have a similar OpenUAV container with ID 6 accessible over the url `term-6.openuas.us`. Term-6 demonstrates a single UAV system controlled using QGroundControl which is running inside the container.

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 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 or Bullet physics engine. Scenes rendered using Unity are photorealistic because they provide the art assets, material properties like shadows, specular, emissivity, shaders and textures to fine-tune scene objects to be more realistic. Besides object enhancements, Unity has algorithms for occlusion culling which disables rendering of objects that are not currently seen by the camera. Here, we realize a need in Gazebo to develop a photorealism layer between the physics and rendering engine to supplement computer vision algorithms in robotics simulation. Future versions of OpenUAV hopes to provide an option to integrate with the game engines to create photorealistic renderings.

C. AirSim

AirSim is an open-source, cross-platform simulator built on Unreal Engine that offers visually realistic simulations for drones and cars. [12] It supports hardware in the loop simulations with flight controllers like PX4 and support for popular protocols (e.g. MavLink). AirSim provides a realistic rendering of scene objects such as trees, lakes and electric poles. AirSim approach is useful for developing perception algorithms, especially in outdoor environments. One of the goals of OpenUAV system is to have minimum code changes when transitioning from simulation experiment to field trials. Although Gazebo simulator is not capable of photorealistic rendering, it has the advantage of a straightforward simulation description format to create general robots like manipulator arms or legged robots. Future versions of OpenUAV hopes to utilize concepts from game engines like Unity and Unreal Engine to generate visually realistic simulations.

III. DESIGN GOALS

OpenUAV is an on-premise, and cloud framework for developing dynamic controllers and swarm algorithms for UAS, remotely operated underwater vehicles (ROV), and

autonomous underwater vehicles (AUV). OpenUAV utilizes the docker container technology that replaces the tedious installation of simulation and flight control software with its dependencies, by having to download a single pre-built ready-to-run image in Linux machines. [13] OpenUAV simulation containers are built with a fast and lightweight operating system Ubuntu with necessary flight control software, communication protocols and mission planning software. Figure 1 shows a demo of single and multi UAS scenarios in OpenUAV framework.

Currently, to develop UAS software or conduct field experiments, the developer or researcher starts with the simulation and then moves to real robots. For simulation, they usually work using the popular robotics tools like Gazebo, ROS and flight stack such as PX4 and QGroundControl. The researcher would realize the need for GPU enabled machine to render the visualization from Gazebo. An apparent difficulty they face is to be always at the GPU enabled computer to work on the simulation. This approach also indirectly restricts them on providing a live demonstration of their work to remote users, 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. [4] In the following section, we outline how these necessities guided the design 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 this barrier to entry in research and development of autonomous vehicles. Therefore for OpenUAV to achieve its goals, it has to implement 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 devices in the simulation containers, by having similar memory constraints and compute capacity.
- Minimize the risk of data breach, through built-in encryption between client and server. The data protection mechanisms should not require additional layers of protection such as virtual private networks (VPN).
- Provide mechanisms for maintenance through daily build images, regular update releases and data recovery mechanisms.

IV. SYSTEM ARCHITECTURE

This section presents the software components of the OpenUAV and the interaction between user and the system. The updated system design was inspired from the work of Will Kessler and the Udacity team. [14] We can classify the software components into three categories: simulation

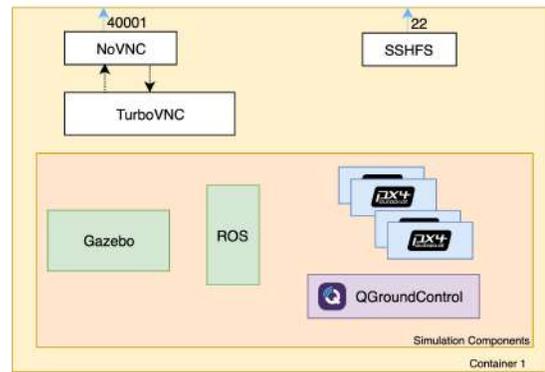


Fig. 2: OpenUAV container has simulation software such as Gazebo, ROS and RViz. In addition to that, it has support for flight stack like PX4 and mission planning software like QGroundControl. Softwares like TurboVNC, NoVNC and SSHFS are added to create interactive containers.

components, virtualization components and interactive components. An overview of the OpenUAV simulation container is shown in Figure 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 OpenUAV framework are the following.

1) *Gazebo*: [9] Gazebo is an open-source robotics simulator that is used to design robots, 3D worlds and perform realistic rigid body dynamics. Simulated objects have mass, velocity, friction, and other attributes that enable them to be realistic when forces are applied. Gazebo can simulate more general robots through its links-and-joints architecture like the sophisticated manipulator arms or legged robots. Gazebo provides a web visualization tool called GzWeb that provides visualization of the environment using browsers. An earlier version of OpenUAV used GzWeb to interact with the simulation running in cloud.[4] GzWeb being a thin web client has a limited set of graphical options when compared to actual Gazebo client. The improved OpenUAV version utilizes remote desktop sharing technologies to enable seamless sharing of the Gazebo client to the users. Moreover, the new OpenUAV provides users with access to other PX4 community developed tools like QGroundControl within the same desktop.

2) *ROS*: The Robot Operating System is a robotics message passing framework that is designed to simplify programming for various robots [15]. ROS is widely used in the robotics community and has various community developed packages for sensors and actuators. A robotics system publishes itself as a ROS node. From that node, it can subscribe and publish a message to named topics. OpenUAV supports ROS over a separate simulation API like in AirSim [12], so that users can use the robotics community developed packages and to reduce the code changes during the transition from simulation to real robots.

3) *PX4*: is an open-source flight stack for UAVs and other unmanned vehicles [16]. It is used to provide basic navigational functionalities like way point navigation, landing, takeoff, hover. Latest versions of PX4 provides the higher level capabilities like collision avoidance in UAVs. MAVROS is a ROS package that simplifies the communication and control between the user code and the UAV by converting ROS messages to MAVLink protocol [17]. This allows a user to send control commands through ros topics, thus allowing users with a higher level abstraction.

4) *QGroundControl*: is a software package used for monitoring and mission planning for any MAVLink enabled drone [18]. QGroundControl can communicate with both PX4 and ArduPilot powered vehicles [19]. Since QGroundControl is developed using Qt, which is a cross-platform software for application development, OpenUAV container base image has support for building and developing Qt applications. QGroundControl can be used to evaluate the simulations, as they provide a visual display on vehicle position, flight track, waypoint and vehicle instruments.

B. Virtualization

Virtualization is the process of creating a software-based representation of something, such as virtual applications, servers, storage and networks [20]. In OpenUAV, we have operating-system-level virtualization to deliver software as packages called containers [21]. Figure 3 shows how each of these components interact with each other.

1) *Docker*: container technology is arising as the preferred means of packaging and deploying applications. Docker is able to take software and its dependencies, package them up into a lightweight container. Out of the box, Docker provides isolation from host, isolation from each other, improve the security of the application by restricting the host system calls possible, and running applications in least privileged mode [13, 22].

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) *Kubernetes*: is an open-source software for automating deployment, scaling, and management of containerized applications [23, 24]. OpenUAV architecture was tested in both Docker Swarm [25] 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 [26]. We found that sharing of GPU resources in pods is a work in progress among kubernetes and docker swarm community, because the container isolation guarantees on GPU resources needs to be provided by the GPU vendors [27].

3) *Nginx*: [28] is a web server which can also be used as a reverse proxy, load balancer and HTTP cache. OpenUAV utilises Nginx as reverse proxy and load balancer for accessing the remote desktop sessions. Each remote desktop session running inside a container is given an simulation ID (whole number) and the access to that session is using the following URL pattern `term-<ID>.openuas.us`. Nginx is also 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.

C. 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 [29, 30].

1) *TurboVNC*: when used with VirtualGL, provides a highly performing and robust solution for displaying 3D applications over all types of networks [31, 32]. 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*: is a JavaScript VNC application that provides VNC sessions over the web browser [33]. 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 40xx port where xx is the simulation ID and Nginx proxies it to the outside world over unique terminals with same simulation ID `term-<ID>.openuas.us`. NoVNC provides built-in encryption between client and server. Thus it does not require additional layers of protection such as virtual private network (VPN). A user with 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, it enables users to easily switch between machines while still being presented with the same desktop i.e. each application is exactly as they left it.

3) *Secure Shell FileSystem (SSHFS)*: [34] allows you to mount a remote file system using secure shell file transfer protocol (SFTP). Through SSHFS, OpenUAV provides an easy remote file access mechanism to simulation directories, by having it mounted as 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 [35]. SSH access to container is made available as per requests from users, since it requires exposing a streaming port in Nginx.

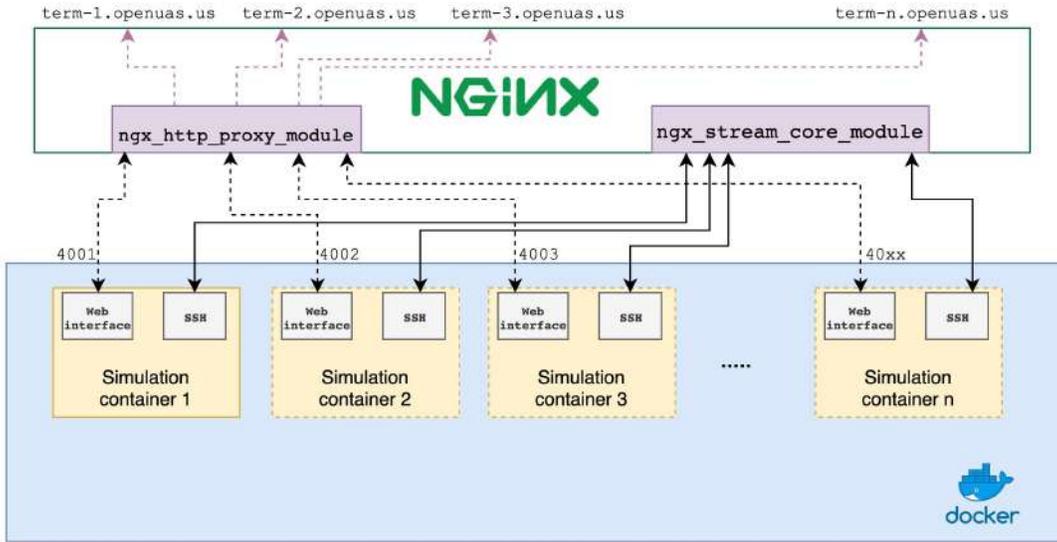


Fig. 3: shows the virtualization components of OpenUAV. Docker port forwards each containers web interface as 40xx ports where xx denote the simulation ID. HTTP proxy module and stream module are adopted to proxy multiple HTTP sessions and stream ssh session to outside users.

V. CASE STUDIES

A. Visual servoing in sUAS using Apriltags and YOLO

Our first case study showcases a single UAS doing visual servoing in the OpenUAV framework. The quadrotor (DJI F450) in simulation is equipped with a fixed on-board camera (without gimbal) facing downward, which is used for the controlled descent towards a target in the environment [4]. This simulation was developed as part of NSF CPS Challenge 2018 [36].

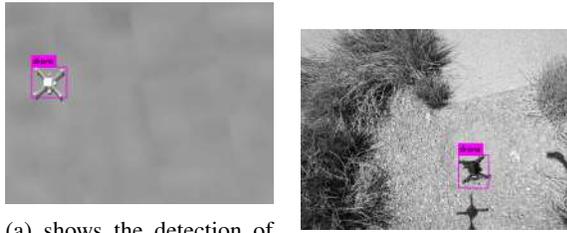
Visual servoing is the fusion of results from many elemental areas, like image processing, dynamics, control systems, and real-time computing [37]. Through visual-servoing, we try to control the UAS system using vision and attempt to descent towards the object of interest. Visual servoing has two main approaches: position-based visual servoing (PBVS) and image-based visual servoing (IBVS). In position-based visual servoing, features of the target are extracted from the image and using intrinsic and extrinsic parameters of the camera, the pose of the target is estimated in global coordinates. Our approach focuses on image-based visual servoing (IBVS), where the control values are calculated based on image features directly. This approach reduces the computational delay due to transformations and eliminates the errors due to sensor modelling and camera calibration [37]. Our approach is implemented in 3 major modes - SURVEY, SCAN, DESCENT. Initially the UAV does a survey (SURVEY mode) of the area in a lawnmower pattern. Once we have a detection of the target on the ground, the scanning phase (SCAN mode) begins where we hover the area to receive continuous detection of the target for 3 seconds. The scanning phase is implemented to improve the robustness of the system to avoid false detection. The third phase is the descent phase (DESCENT mode) where the UAV does a descent towards the target using IBVS strategy.

In survey mode, we use the PX4 position controller to specify the way points for lawnmower based search navigation. The target used in the simulation is a UAV frame and an AprilTag, which is visual fiducial system developed by April Robotics Laboratory. [38] For the detection of the UAV frame, we collected 60 UAV images from simulation and experiments. The images were augmented and trained on You Only Look Once (YOLO) object detection algorithm. [39] The results are shown in Figure 4. Our approach used YOLO for object detection from surveying heights of 7-10m and AprilTag module for precise maneuvering of UAV descent when UAV is less than 5m from ground. [40]

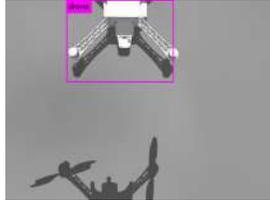
We have adopted ROS software package developed for YOLO object detection that enable us to obtain bounding boxes in image coordinates. The bounding box is further transformed to camera coordinates for the controller. Our PID controller attempts to keep the target at the center of the image plane captured by the camera, to minimize the probability of losing the target from the field of view. The PID parameters for YOLO bounding box based descent is tuned using numerous trails in OpenUAV framework. We use the AprilTag ROS wrapper package for precise descent on the target UAV. The PID parameters for AprilTag based descent are calculated after similar trails in the OpenUAV framework. In addition to the ease of use, the docker environment in OpenUAV framework provides the capability of saving current simulation setup into an image file and replicating the same setup across multiple simulation containers.

B. Terrain Relative Navigation

UAS systems are widely used for mapping in geological sciences, urban planning and precision agriculture. Lawnmower patterns are commonly used for an aircraft to survey a large area. As a result, the ground sampling resolution may vary. The inconsistent ground sampling resolution in



(a) shows the detection of UAS in OpenUAV simulation using GeForce RTX 2080 Ti (b) shows the detection of Intel Aero in outside environment

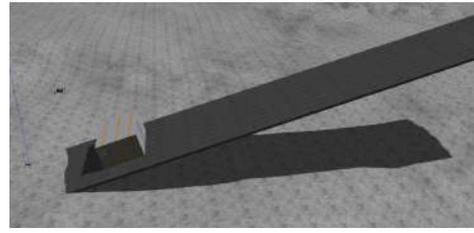


(c) shows the detection of UAS frame during a descent towards the object. It was able to classify UAS frame from noise, such as shadows.

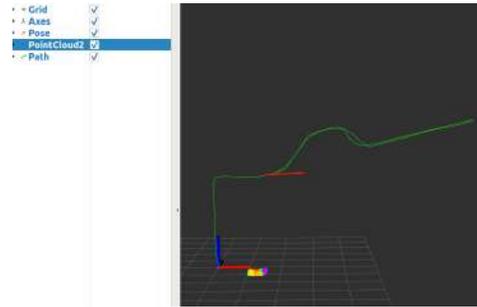
Fig. 4: Predictions after YOLO training

aerial imagery can cause discrepancies in estimates of rock dimensions. To alleviate this issue, we explore lawnmower navigation with realtime awareness of terrain-height. Various approaches have been explored in terrain relative navigation for autonomous landing of planetary rovers. [41] [42]. Our approach, which does not require prior knowledge of the environment, is vision-based and adjusts the z component of the pose based on the terrain height. Our algorithm was tested in simulation with the 3DR Iris quadrotor with Intel RealSense R200 depth camera. We utilized the point cloud library (PCL) [43] to transform point cloud data generated by the depth camera to global coordinates. The transformation from camera coordinates to global coordinates is achieved through a combination of UAV state estimation by PX4 Extended Kalman Filter (EKF) [44], and a static transform of the pose of the camera to the UAV. Since the navigation software is independent of the sensor, we can also use a stereo camera that is better suited for outdoor use, to generate the point cloud.

Throughout our simulation, we try to maintain a fixed height h_f of 3m from the terrain. The UAV simulation aims to reach the goal waypoint in conjunction with terrain relative navigation. The UAV estimates the average height of the current terrain h_t by looking up the centroid of a 0.5m 3D voxel grid containing point cloud. There are 2 PI controllers developed: the first is a simple controller to adjust for errors in the estimated x and y-direction; the second is terrain aware controller that corrects for the errors in the z-direction. The controller adjusts the velocity of the UAV based on the error estimates. The first controller calculates the errors in x and y direction with respect to the goal waypoint. The second



(a) shows the simulation setup that involved a ramp and a rectangular cuboid. The UAV has to maintain a relatively fixed height with changing terrain.



(b) shows the path, pose and point cloud data in Rviz. The path taken by the UAS is very similar to the terrain structure.

Fig. 5: Terrain Relative Navigation

controller measures the error using the following equation:

$$error = (h_t + h_f) - h_{current}$$

where $h_{current}$ represents the current height of the UAV. The parameters of PI controller was selected after numerous iterations of UAV simulations. A great advantage of simulating in OpenUAV framework provides is the ability to replicate the similar setups across multiple containers, to fine tuning controller parameters.

Our simulation involves a world with a ramp and rectangular cuboid on it. The simulation intends to tests the UAV on uphill and downhill scenarios. The controller is provided with the x and y position of the goal waypoint. As figure 5 demonstrates, the UAV can increase and decrease the altitude when it found the rectangular cuboid on the ramp. Moreover, the UAV achieves fixed height navigation throughout the uphill and downhill motion.

VI. CONCLUSIONS

In this paper, we presented a simulation testbed that allows collaborative mission development. We leveraged open-source software such as ROS, Gazebo, PX4 flight stack, Docker, Rviz and TurboVNC to provide a rich development environment that scales with available compute. Two case studies are shown, visual-servoing, and terrain relative navigation. In future, we will enhance the camera rendering to be photorealistic by leveraging the Unity game engine.

ACKNOWLEDGMENT

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

REFERENCES

- [1] 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.
- [2] *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).
- [3] 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.
- [4] 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.
- [5] 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.
- [6] Nassim Ammour et al. “Deep learning approach for car detection in UAV imagery”. In: *Remote Sensing* 9.4 (2017), p. 312.
- [7] 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.
- [8] Fadri Furrer et al. “RotorS—A modular Gazebo MAV simulator framework”. In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.
- [9] 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.
- [10] ETHZ-ASL. *RotorS is a MAV gazebo simulator*. 2018. URL: https://github.com/ethz-asl/rotors_simulator.
- [11] Winter Guerra et al. “FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality”. In: *arXiv preprint arXiv:1905.11377* (2019).
- [12] Shital Shah et al. “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and service robotics*. Springer. 2018, pp. 621–635.
- [13] Paolo Di Tommaso et al. “The impact of Docker containers on the performance of genomic pipelines”. In: *PeerJ* 3 (2015), e1273.
- [14] Will Kessler. *Building a GPU-enhanced Ubuntu Desktop with nvidia-docker2*. 2018. URL: <https://github.com/willkessler/nvidia-docker-novnc>.
- [15] 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.
- [16] 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.
- [17] Vladimir Ermakov. *MAVLink extendable communication node for ROS with proxy for Ground Control Station*. URL: <http://wiki.ros.org/mavros> (visited on 09/14/2019).
- [18] E Zurich. *Qgroundcontrol: Ground control station for small air land water autonomous unmanned systems*. 2013. URL: <http://qgroundcontrol.com>.
- [19] ArduPilot Dev Team. “Ardupilot”. In: *URL: www.ardupilot.org, accessed 2* (2016), p. 12.
- [20] VMWARE. *Virtualization*. URL: <https://www.vmware.com/solutions/virtualization.html>.
- [21] Carl Boettiger. “An introduction to Docker for reproducible research”. In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 71–79.
- [22] Docker. *What is a Container?* URL: <https://www.docker.com/resources/what-container>.
- [23] David Bernstein. “Containers and cloud: From lxc to docker to kubernetes”. In: *IEEE Cloud Computing* 1.3 (2014), pp. 81–84.
- [24] Kubernetes community. *Kubernetes*. URL: <https://kubernetes.io/> (visited on 09/14/2019).
- [25] Docker community. *Docker Swarm mode overview*. URL: <https://docs.docker.com/engine/swarm/> (visited on 09/14/2019).
- [26] CoreOS community. *CoreOS*. URL: <https://coreos.com/kubernetes/docs/latest/pods.html> (visited on 09/14/2019).
- [27] *Is sharing GPU to multiple containers feasible?* 2017. URL: <https://github.com/kubernetes/kubernetes/issues/52757> (visited on 09/14/2019).
- [28] Will Reese. “Nginx: the high-performance web server and reverse proxy”. In: *Linux Journal* 2008.173 (2008), p. 2.
- [29] Karin Meier-Fleischer, Niklas Röber, and Michael Böttinger. “Visualization in a Climate Computing Centre”. In: *EGU General Assembly Conference Abstracts*. Vol. 16. 2014.
- [30] 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).
- [31] 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.

- [32] TurboVNC. *A Brief Introduction to TurboVNC*. URL: <https://turbovnc.org/About/Introduction> (visited on 09/14/2019).
- [33] Joel Martin et al. *noVNC: HTML5 VNC Client*. 2015. URL: <https://novnc.com/info.html>.
- [34] Matthew E Hoskins. "Sshfs: super easy file access over ssh". In: *Linux Journal* 2006.146 (2006), p. 4.
- [35] JetBrains. *PyCharm The Python IDE for Professional Developers*. URL: <https://www.jetbrains.com/pycharm/>.
- [36] Harish Anand, Zhiang Chen, and Jnaneshwar Das. *Visual Servoing for UAV's using Apriltags and YOLO*. 2018. URL: https://github.com/DREAMS-lab/nsf_cps_challenge.
- [37] Seth Hutchinson, Gregory D Hager, and Peter I Corke. "A tutorial on visual servo control". In: *IEEE transactions on robotics and automation* 12.5 (1996), pp. 651–670.
- [38] John Wang and Edwin Olson. "AprilTag 2: Efficient and robust fiducial detection". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4193–4198.
- [39] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.
- [40] Zhiang Chen. *YOLO ROS Drone detection*. URL: <https://www.youtube.com/watch?v=x6ajf5Kv5to>.
- [41] Andrew E Johnson and James F Montgomery. "Overview of terrain relative navigation approaches for precise lunar landing". In: *2008 IEEE Aerospace Conference*. IEEE. 2008, pp. 1–10.
- [42] Andrew Johnson et al. "A general approach to terrain relative navigation for planetary landing". In: *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*. 2007, p. 2854.
- [43] Radu B Rusu and S Cousins. "Point cloud library (pcl)". In: *2011 IEEE international conference on robotics and automation*. 2011, pp. 1–4.
- [44] Greg Welch, Gary Bishop, et al. "An introduction to the Kalman filter". In: (1995).