VADER: A Hardware and Simulation Platform for Visually Aware Drone Autonomy Research

Andrew Buck[a], Matthew Deardorff[a], Derek T Anderson[a], Timothy Wilkin[b], James M Keller[a], Grant Scott[a], Robert H Luke III[c], and Raub Camaioni[c]

[a]Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, MO, USA
[b]School of Information Technology, Deakin University, Geelong, Victoria, AU
[c]US Army DEVCOM C5ISR Center, Fort Belvoir, VA, USA

ABSTRACT

Low altitude Unmanned Aerial Systems (UASs) provide a highly flexible and capable platform for remote sensing and autonomous control. There are many applications that would benefit from an additional bird's eye view, including mapping, environmental reconnaissance, and search and rescue to name a few. An autonomous (or partially autonomous) drone could assist in several of these scenarios, freeing the operator to focus on higher-level strategic planning. While numerous commercial drones exist on the market, none truly provide a flexible foundation for vision guided autonomy research. Herein, we propose the design of a physical UAS platform, called VADER (Visually Aware Drone for Environmental Reconnaissance), and an accompanying simulation environment that addresses many of these tasks. In particular, we show how Commercial Off The Shelf (COTS) hardware and open source software can now be combined to realize powerful end-to-end UAS research solutions. The beauty of unifying these factors is accelerated prototyping and minimal time to migrate and test in the real world. This article outlines VADER and case studies are presented to demonstrate capabilities.

Keywords: Unmanned aerial systems, hardware platform, drone autonomy, simulation

1. INTRODUCTION

The following article outlines a hardware and simulation framework to facilitate end-to-end academic low altitude unmanned aerial system (UAS) research. Hereafter, we refer to this framework simply as VADER (Visually Aware Drone for Environmental Reconnaissance). VADER is a challenging endeavor because end-to-end UAS research requires us to engage in a wide range of fields, from flight physics to mechanical and electrical design, computer engineering and beyond, e.g., human factors. It is important to explore this topic because UASs are on the rise and they are hard to compartmentalize. That is, UASs are a complex and dynamic system of interconnected pieces; e.g., algorithms (AI/ML, mapping, localization, control, etc.), processing hardware, software environments, sensors, batteries, flight controllers, communication, data visualization, etc.

In response to the sheer complexity of a UAS, academics have been forced to divide-and-conquer. One example is researchers focused on smaller and high-level abstractions outside of the physical universe, e.g., mathematical solutions to the traveling salesman problem, robotic behavior through reinforcement learning, etc. Another set of researchers remain at least somewhat connected to the UAS by exploring the offline processing of UAS acquired data sets, e.g., development and evaluation of structure from motion (SfM) algorithms on real-world drone imagery with optional GPS and IMU metadata for wide area motion image (WAMI) analysis. Another lower-level UAS research line is simulation, e.g., robotic agents in an environment like the Unreal Engine and AirSim or Gazebo. At the lowest level of research is the physical aircraft and its underlying mechanical and

Send correspondence to Andrew Buck
Andrew Buck: E-mail: buckar@missouri.edu

electrical challenges. By far, the vast majority of users rely on pre-scripted flights (e.g., a lawnmower pattern over a region of interest) or manual operation (a human and flight controller). While a few number of programmable UASs are beginning to appear commercially, they are generally expensive (and therefore out of reach for many), have limited functionality, and/or coding is overly specific to the platform. It is desirable, from an academic standpoint, to have a framework that abstracts and unifies these heterogeneous and interconnected parts of the "UAS spectrum". Achieving such a feat would allow us, in theory, to better explore the depths of this complex space and reduce the time it takes to migrate research to real-world UAS solutions like search and rescue, emergency and disaster response, security and defense, human-robot teaming, etc. We are not alone in desiring such a solution. Others in areas like autonomous vehicles, medicine, and materials are actively seeking unified design spaces to explore future technology versus our historical approach of try, fail, and learn. The goal is to reduce the time and cost required to bring solutions to market.

It is important to note that some of the "tools" outlined below are new while others are not. What has changed is the degree of realism, availability of tools across the UAS board, and processing power. These tools allow us to abstract our model of the world, UAS hardware and UAS software. An added benefit of VADER is connecting the simulated and real worlds. That is, real data can now be migrated into simulation with high fidelity and simulated models (e.g., AI/ML) can be migrated to the real world, directly or as an initial model that can be iterated on using additional real world data. In summary, we are living in an exciting research era and the intent of this article is to demonstrate methods that we have found useful in the hope that others find it equally compelling and can use it to power their research.

In addition, we note that we are not the creators of this alignment of hardware and software and we are not the only researchers to explore its value. In subsequent sections we highlight related works. What largely motivates us to write this article is the following. We were unable to find any resource, or staging of resources that outlines how to design and build a research UAS platform, taking into consideration the software stack and demand for real-time operation in a generic research setting. There are a number of useful academic, commercial, and drone hobbyist resources, videos, links, and papers on the web. However, none exist in a state that we could follow and realize VADER. This motivated us to build on those resources and write a new article documenting our process for others. Last, to the best of our knowledge, the following article is the only documentation we are aware of that discusses a range of application potential. Meaning, we demonstrate and discuss aligning hardware and software to build a physical platform, followed by a higher-level software stack for purposes like mapping, detection, tracking, autonomy, robotic exploration, scientific visualization, human-robot teaming, and more.

2. HARDWARE PLATFORM

Here we discuss the physical hardware components used for VADER. Our primary objective in choosing a hardware design for the VADER platform was to develop a flexible system that would be capable of performing many different tasks. We favored a larger drone system that could handle heavy payloads with extended flight times. This would also provide ample space to mount various sensors and electronic equipment, such as the flight controller, companion computer, and radio antennas. We chose to use a hexacopter design based on the Sky Hero Spyder 6,[1] which provided the frame for our build. The frame is made of carbon fiber, and is both light and sturdy.

For the flight controller, we use the Cube autopilot[2] with the ArduPilot[3] firmware. This is mounted on a Kore carrier board[4] to simplify the power distribution and wiring requirements. The companion computer is an NVIDIA® Jetson™ TX2[5] mounted on a Spacely carrier board.[6] This combination provides a robust flight control system with a high performance computing platform that can process sensor data and provide high-level autonomy in real-time. Communication is handled via the MAVLink protocol,[7] which transmits telemetry from the drone to a ground control station running on a laptop.

VADER is designed to be configurable with a variety of different sensors. The flight controller has a built-in IMU and can use an external GPS unit directly. These are used by the autopilot to maintain stability and provide waypoint navigation, but can also be made available to higher level autonomy applications. Each application may have unique requirements as to the most appropriate sensor suite. Typically, the platform will be configured with at least one color (RGB) camera to provide image data for awareness and processing. For mapping applications,

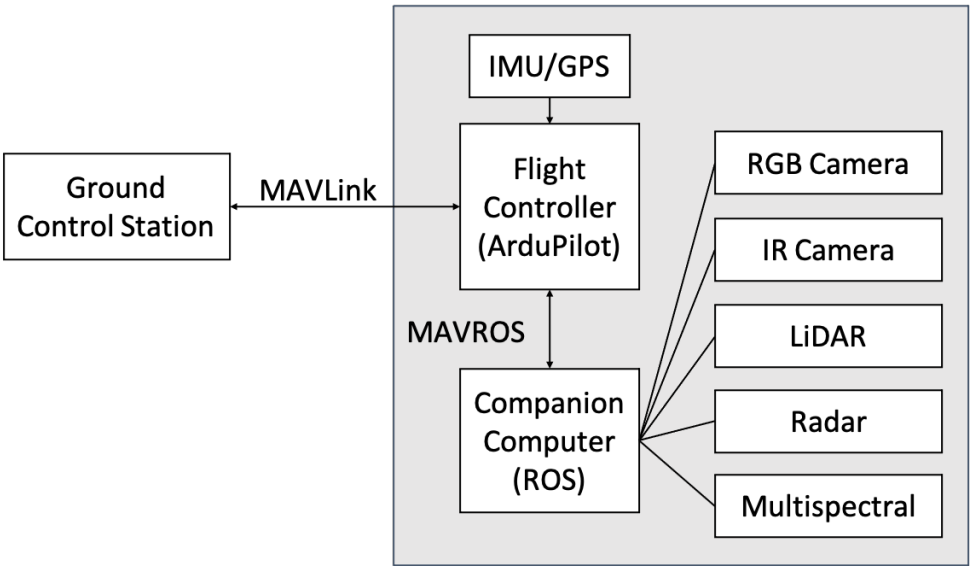

Figure 1: Illustration of the hardware stack for VADER. Missions may be initiated and monitored through the GCS, which maintains communication through the MAVLink protocol. The flight controller and companion computer communicate and execute missions through the MAVROS protocol. The companion computer is responsible for managing and evaluating data from on-board sensors.

it may be desirable to include a LiDAR sensor to ascertain more accurate distance measurements. For target detection problems, specialized sensors such as an infrared (IR) camera, radar, or multispectral camera may be useful. The sensors communicate with the on-board companion computer, which publishes the data to other systems using ROS[8] and the MAVROS[9] protocol. ROS and the software components are discussed in more detail in the next section. Figure 1 shows an overview of the hardware architecture.

The physical components of the VADER platform were sourced from a variety of online retailers. Upon arrival, we inventoried and tested the individual parts. Assembly was guided by online documentation from the manufacturers and some in-house experience. Upon completing the initial wiring of the motors, which involved some soldering and trimming, we began the firmware calibration procedure using the Mission Planner ground control station.[10] Figure 2 shows the assembled drone in a minimal configuration. Note that the companion computer and sensors have not yet been mounted in these images.

Our initial test flights verified that the drone could operate under manual control and perform simple waypoint missions. First, we performed a basic hover test to ensure that the motors and propellers were properly calibrated. We then confirmed that the drone could be flown manually using a radio transmitter. Finally, we connected the drone to the ground control station operating on a laptop and executed a pre-programmed series of commands to takeoff, fly to a set of waypoints, return home, and land. Figure 3 shows some images of the first test flight.

Continued development of the physical VADER hardware was complicated by the ongoing COVID-19 pandemic, which limited in-person activity. Mounting the final components will allow us to demonstrate the capabilities of the platform on real-world problems. In the meantime, we have focused on software and simulation, which has seen increased interest due to the need for remote work. These efforts are discussed in the next section.

3. SIMULATION ENVIRONMENT

The software stack for VADER can be broadly divided into high-level and low-level systems (see Figure 4). The low-level system consists of the flight controller provided by ArduPilot, and is responsible for ensuring level flight and moving the drone to specified waypoints. We assume that this component is in a mature state and can be used on our platform without needing to modify the existing code. Our primary interest is in developing high-level autonomy using the on-board sensors and computing capabilities to allow the autonomous agent controlling

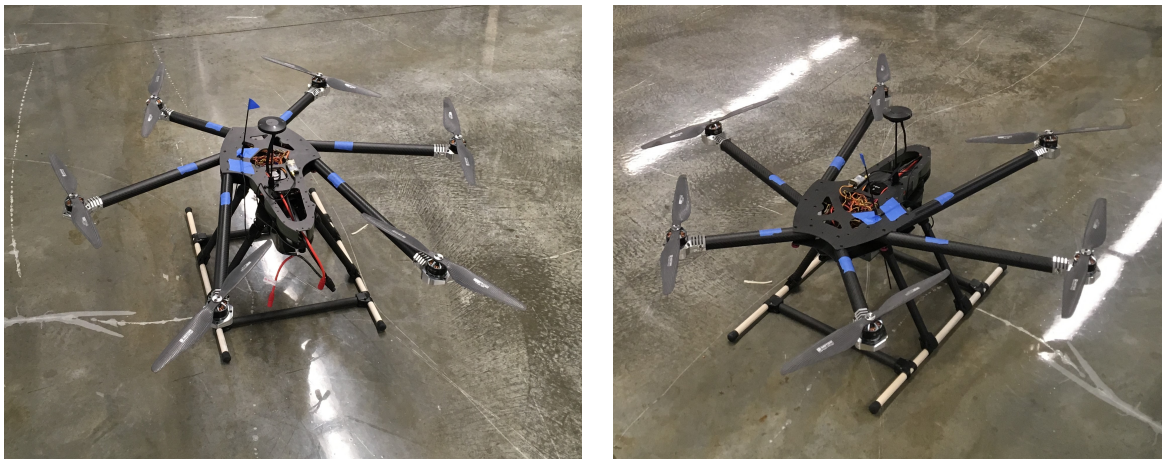

Figure 2: The physical VADER drone platform in a minimal state after initial assembly. No sensor payload is attached and no enclosure (which protects the electronics) is shown.

Figure 3: First test flight of the VADER drone.

the UAS to reason about its environment and decide where to go next. For this, we use ROS, which handles the message passing protocol between sensors, compute nodes, and the flight controller. ROS has a large community of developers who have contributed packages to interface with many different sensors and algorithms. This makes it easy to build up a complete system by combining existing tools and methods. ROS can interface with the flight controller using the MAVROS package, which bridges the ROS and MAVLink protocols.

The software environment we have chosen can be run directly on the physical drone hardware, or simulated (to some degree) in a virtual setting. Using a simulator makes it easy to test out new algorithms and ideas without risking failure on real hardware. There are several ways to approach simulation, depending on the realism required and the intended purpose. For testing the autopilot and flight controller, ArduPilot provides a Software-In-The-Loop (SITL) simulator that can model the system dynamics and control vectors. When paired with a ground control station, such as MAVProxy,[11] this gives the ability to run a simulated waypoint mission and send commands to the virtual drone.

The flight controller can be broadly described as a system that takes inputs from sensors and mission parameters and provides outputs in the form of control signals to the propeller motors. Both inputs and outputs can be simulated in a virtual environment to produce synthetic sensor readings and model the world position of the drone based on the output from the flight controller. Two simulation environments we have explored are Gazebo[12] and AirSim.[13] Gazebo provides integration with ArduPilot and ROS, and can simulate drone flight in a 3D world with physics and access to virtual sensors. AirSim is a simulator built on the Unreal Engine[14] that provides similar capability, with access to the powerful UE4 engine and marketplace. Using AirSim, we

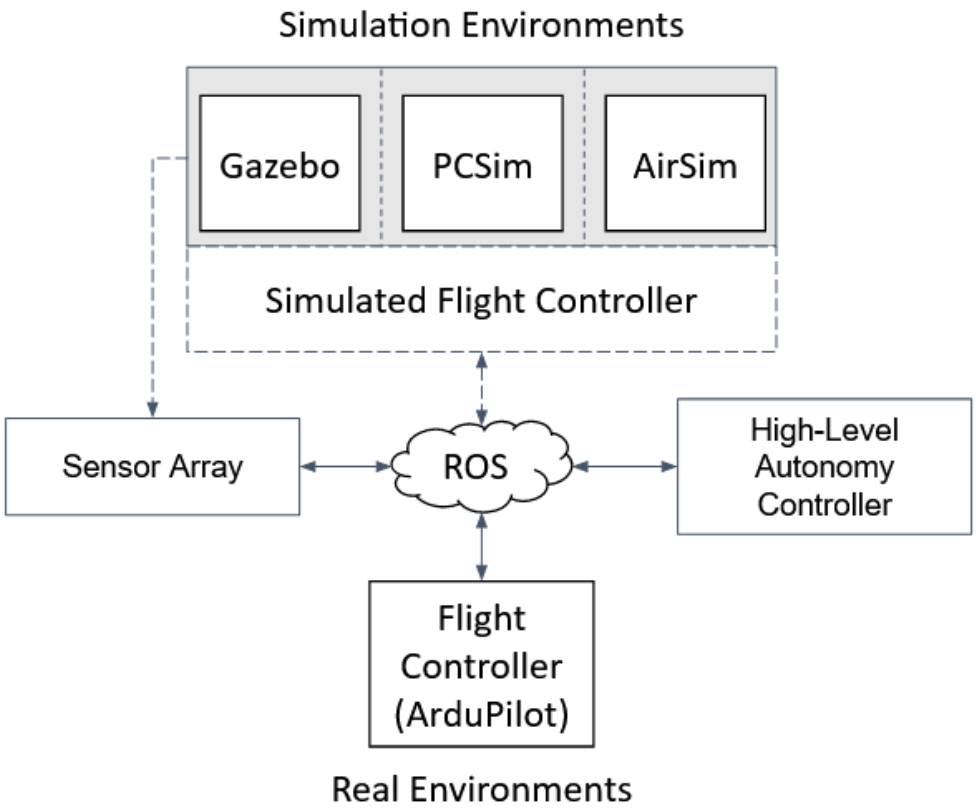

Figure 4: The software stack for the VADER. The flexibility of ROS as a message handler allows each component to communicate with the rest. Dashed lines denote simulated inputs which do not exist in real environments. Crucially, the autonomy controller should be agnostic to whether or not the current environment is simulated. This allows a single code base to plan missions both in the real world and simulation.

can quickly prototype scenarios using pre-modeled environments and assets. Figure 5 shows several example environments in the AirSim simulator and the color, depth, and segmentation images produced by the virtual drone camera.

Figure 6 shows an example of the VADER drone operating in the Gazebo simulator with a world model generated from real data. To build the world model used in this example, we collected nadir image data with a commercially available drone (a DJI Phantom) flying in a grid or lawnmower pattern. The images were stitched together to form a 3D model using the OpenDroneMap[15] software, which was then loaded into the simulator. The simulated drone is configured to publish a color image from a virtual camera as a ROS topic. We have set up an additional ROS node to process the incoming imagery and output an estimate of the depth using the monocular depth estimation algorithm MonoDepth2.[16] The drone can be controlled via keyboard commands with MAVProxy.

Both Gazebo and AirSim provide the ability to observe the current state of the drone and its sensors and send commands via an external programming interface. This may connect to the simulator via direct API calls, ROS messages, or through a ground control station such as MAVProxy. In any case, we can develop an independent code base using Python to manage the high-level autonomy of the drone and visualize the state of the world based on received observations. We use the Open3D[17] library to build a world model using point clouds, voxel spaces, and mesh data. We call our tool PCSim (Point Cloud Simulator), and it provides an interactive window into the observed world model. This model represents the knowledge available to the agent and can be processed in a variety of ways to decide what actions to take. Figure 7 shows some example images of the point cloud

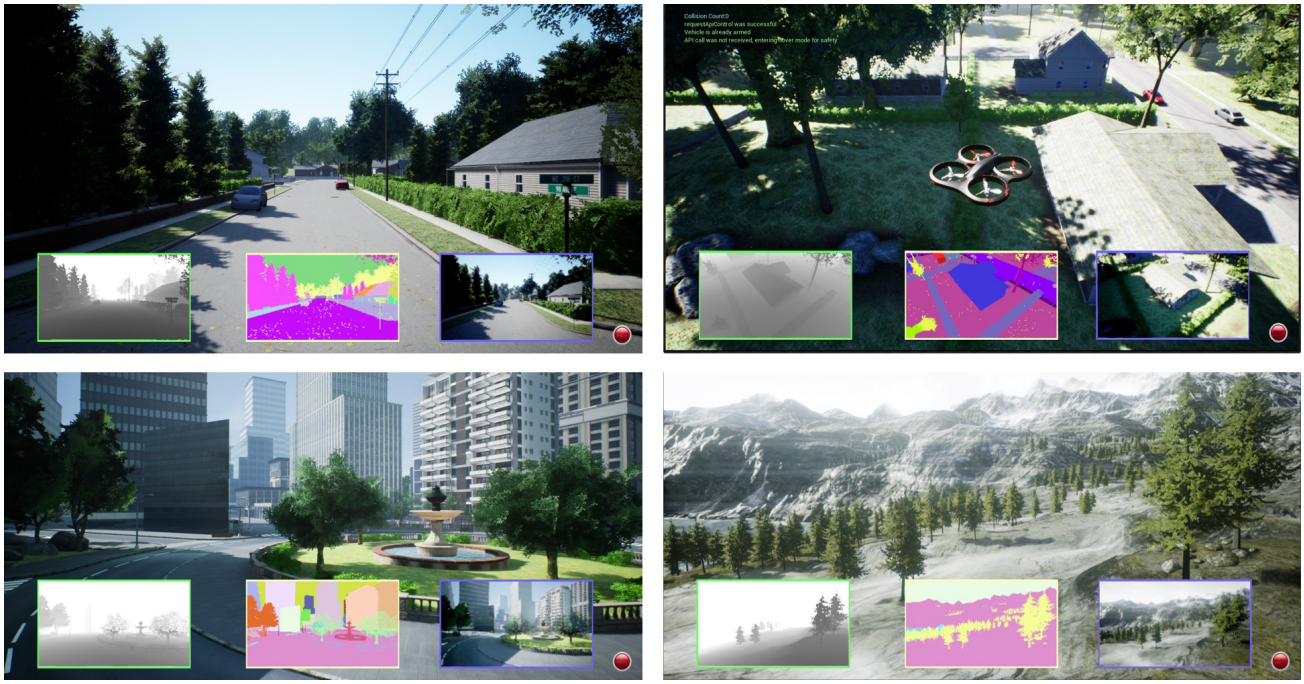

Figure 5: Virtual environments rendered in the Unreal Engine and used with the AirSim simulator. The three subimages for each scene show the depth, segmentation, and color images from the simulated drone camera.

reconstruction and voxelization process from an AirSim environment purchased in the Unreal Marketplace. As the drone moves through the environment, it sends color, depth, and pose information to the Python script, where the points are projected into 3D space with Open3D, and voxelized to reduce the amount of data stored.

To satisfy the autonomy needs of our current applications, we have developed a flexible and extendable framework to implement the navigational planning model of the drone. We refer to this as the agent's mental map, which represents all of the information available to the agent that can be used for planning and completing tasks. As images and odometry are received from the drone, either in simulation or from real hardware, the agent can construct a model of the environment using the best representation available. For instance, when using a simulator, sensor measurements can be made to contain no error, effectively allowing for a perfect reconstruction of the world. The color and depth images along with the drone pose can be used to create a 3D point cloud by projecting each pixel out by the specified depth from the known camera origin. On real hardware, a LiDAR sensor could be used to produce a point cloud, or visual SLAM techniques such as ORB-SLAM2[18] could generate a map of landmarks. In certain scenarios, a pre-computed map of the environment could be used to initialize the mental map.

The map representation is an important part of the autonomy framework. Point clouds, while precise, can grow to be too large to process directly in real-time. A common technique to reduce the computational overhead is to instead store a voxel occupancy grid, such as with the hierarchical approach used in OctoMap.[19] This has the advantage of combining multiple points in a nearby space into a single voxel. Further simplifications, such as a semantic spatial graph using bounding boxes around detected objects, is explored in Ref. 20.

We assume that the intended behavior of the drone can be described as movement toward some (perhaps non-stationary) goal location. The goal may change over time as the situation changes, but in general, the drone can only move within and observe the environment. There are no actuators on the drone that can (significantly) change the state of the environment. The drone is therefore primarily a reconnaissance platform that can be used to gather information about the current state of the world and relay this to a (presumably) human operator. Much of the work of the autonomous controller is then to specify appropriate goal locations and develop plans to reach them.

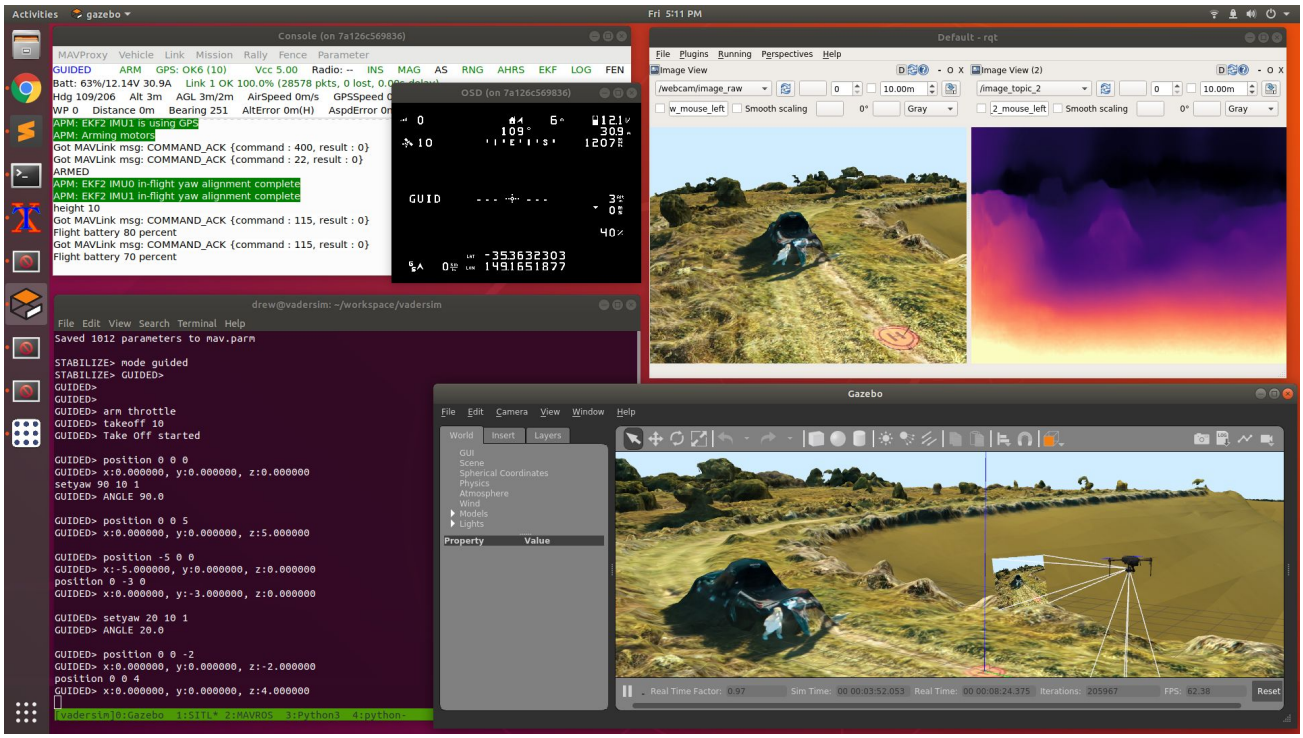

Figure 6: Using Gazebo with the ArduPilot SITL simulator and ROS. A virtual environment is loaded from a precomputed mesh, generated using OpenDroneMap and an earlier data collection flight. ROS shows the front camera view and the depth image produced from monocular depth estimation.

Many different application scenarios can be cast in this way. For example, a mapping task requires the drone to move to unobserved regions of the environment and construct an updated geo-registered model. Here, the goal location may be the nearest unobserved region within some boundary. In a search and rescue problem, the goal is the unknown location of the victim, and so might be cast as a distribution over the search space, from which a specific "next-look" goal location could be sampled. In a more directed human/drone teaming effort, the operator may directly specify a goal location, or a target object of interest.

We can define a set of objectives based on the current mission profile and use these to pick the best goal location. For instance, one objective might be to pick a location that has not been adequately observed, and another might be to pick a location that is not too far from the drone's current position. Other objectives might include trying to maximize the likelihood of observing a particular type of target object at a given location, or minimizing the time since a location was last observed. In general, these objectives form a multi-objective optimization problem, for which each location in the environment is a candidate solution.

Figure 8 shows one way that this multi-criteria planning problem can be realized. Here, we show an the agent's mental map of the neighborhood environment in AirSim after the first observation and again in a planning view after a series of additional observations. The first image shows the reconstructed 3D point cloud in a reduced voxel space. The second image uses a multi-criteria objective function to identify a good target location. In this view, each voxel is evaluated on two criteria and colored accordingly. The distance from the drone is expressed as the blue channel (closer voxels are more blue), and the vertical density of voxels is mapped to the red channel (vertical walls have no red, whereas horizontal planes and scattered voxels are more red). The blending of these two attributes results in various shades of magenta for each voxel. The objective function tries to simultaneously minimize the distance from the drone and the local density, looking for a location that is both nearby and that could be filled in with more information. Each voxel in the map receives a score computed as a weighted average of these two criteria and the highest scoring voxel is shown in green. An appropriate viewing angle for this voxel is computed that avoids obstacles and the resulting location is then set as the next goal location. By following

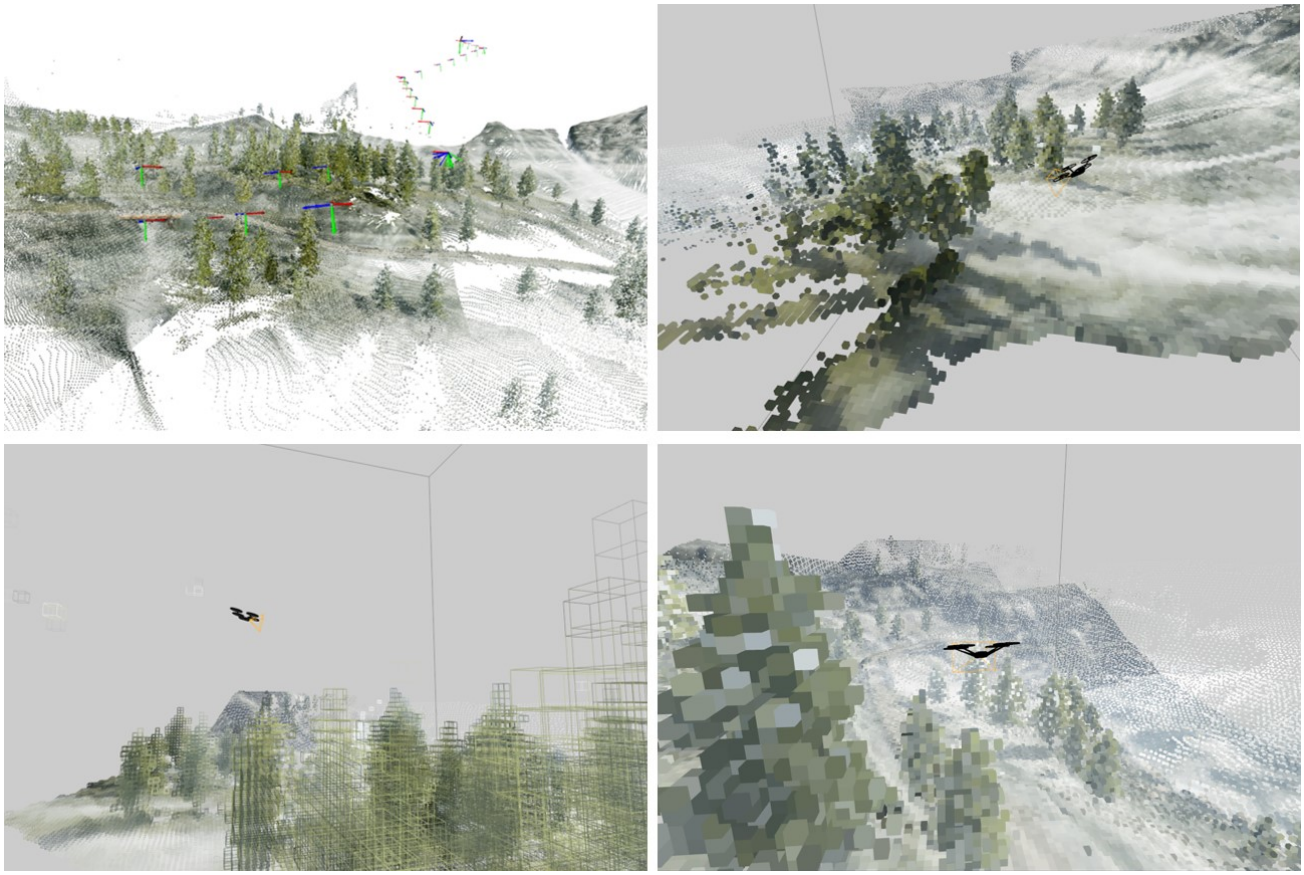

Figure 7: Example of using AirSim to reconstruct point cloud and voxel observations with our PCSim tool using Open3D.

this procedure, the drone can explore the environment and build up knowledge in its mental map.

In closing, this section highlights different software tools and their connectivity. VADER is not a single piece of software, it is more than the sum of its parts. The few provided examples show that the proper abstraction of system and data and staging of these tools can result in a powerful set of features to enable UAS research with low effort and high payout.

4. EXAMPLE APPLICATIONS

In this section we highlight a few VADER enabled applications. Our aim is not a deep dive per application. Instead, the objective is to show the reader a wealth of research possibilities across the UAS spectrum. Figure 9 illustrates the current status of our inter-connectivity. The following subsections highlight applications that engage in subsets of the proposed diagram. While the entire figure is compelling, the possibility of coupling the artificial and real universes, or using the artificial universe predominately, is exciting and a future game changer for UAS research. The less dependant we can become on the real universe, the more we can couple the real and artificial universes, and the more we can control our ideas and introduce them in responsible fashions, the better. Our ideal is to migrate basic science to real world scenarios responsibly.

Mapping: The first application we highlight is mapping, and optionally localization. We begin this discussion by outlining what information is accessible. Figure 5 shows that VADER has access to per-pixel depth and semantic label maps. One emergent field, driven largely by *smart cars*, is depth estimation from single imagery. Very few of these algorithms are supervised due in part to the challenge of obtaining a ground truth per pixel. Stereo vision cameras are only accurate out to a given distance (driven largely by their baseline) and LiDAR,

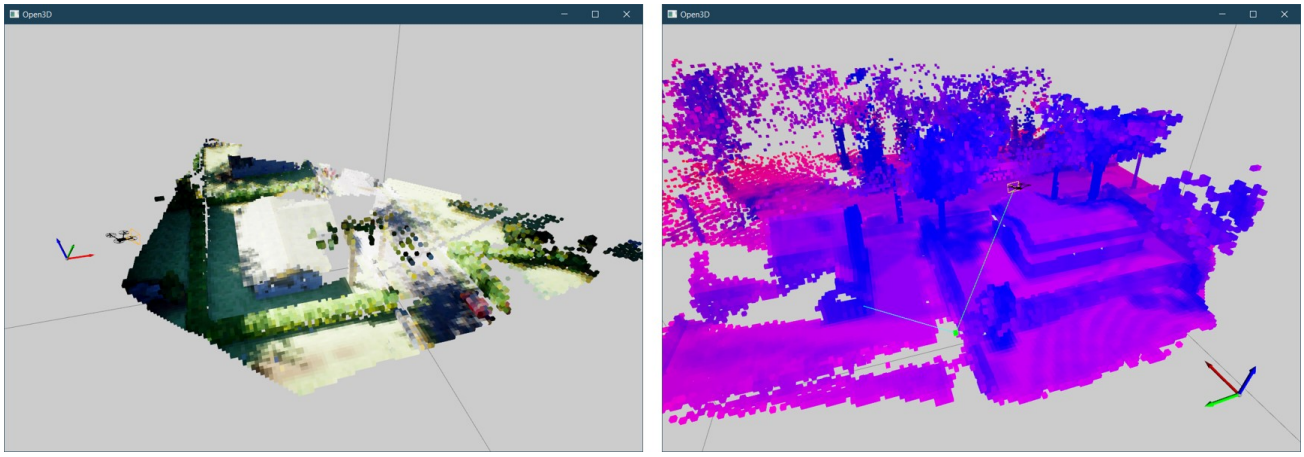

Figure 8: The mental map representation of an environment as a voxel space. The left image shows the initial observation of a neighborhood scene, and the right image shows a planning view after several observations to determine the next best goal location.

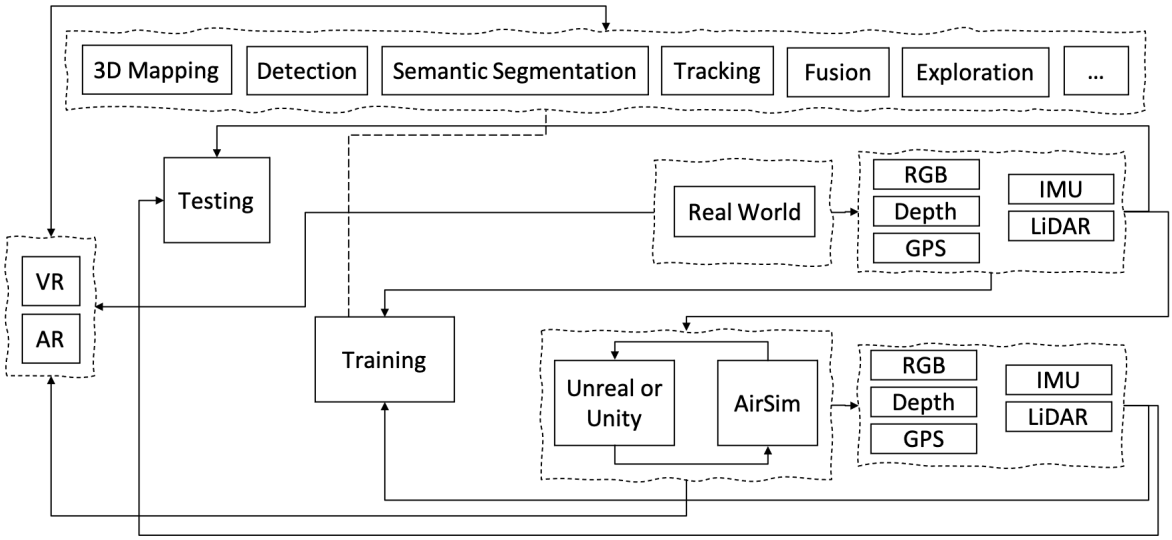

Figure 9: Example of high-level VADER inter-connectivity.

when/if available, is currently sparse. Factors like these have led the field to migrate to unsupervised or self-supervised techniques. An example unsupervised method is the Monodepth algorithm,[21] which operates on a video stream (image sequence) with no metadata (e.g., GPS or IMU data). The Monodepth neural network works, during training, to estimate the camera pose and a mapping from image space to depth, relative to minimizing a photometric loss. At evaluation time, the algorithm takes a single image, camera pose is ignored, and depth (normalized disparities) are estimated per pixel. The point is, accurate training data and furthermore, accurate test data (across different environments and environmental conditions) for evaluation is a big factor and major challenge in passive ranging. We have used VADER to automatically output labeled training imagery to train a supervised UNet and self-supervised Monodepth2. Our initial Monodepth2 experiments (see Figure 10) have resulted in similar performance on real data for simulated versus models derived from real data. For these experiments, we used open source data sets, e.g., the KITTI data set, and purchased content on the Unreal Marketplace, e.g., the Neighborhood Pack. Beyond training, another advantage of simulation is an ability to score our model estimates versus a ground truth to understand performance relative to factors like range, man made structures, time of day, look angles, natural objects, etc. The point is, simulation provides us with more

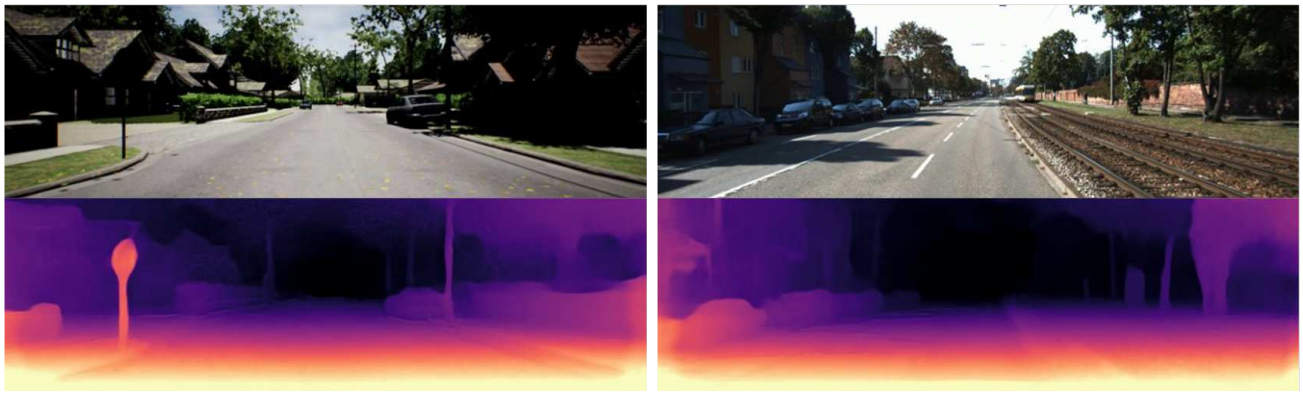

Figure 10: Example of (left) a video sequence from the Unreal Neighborhood Pack and (right) the real world KITTI dataset. The bottom images are estimated normalized disparities (scaled for display) for a Monodepth2 model trained from simulation. Note, the trained model performs well in both simulation and real data (KITTI).

controlled experiments that what we can achieve via costly data collections, which do not have the same degree of truth completion nor accuracy. Furthermore, simulation in an environment like the Unreal Engine allows for rapid development, testing, of LiDAR (see AirSim) and SfM[22] algorithms. It is significantly easier to change factors like Mie or Rayleigh scattering, solar location, UAS viewing (nadir or slant angle), altitude, image resolution, field of view, focal distance, etc., and study its effects on a proposed algorithm before running it in the real world. Last, for applications not concerned with mapping, perhaps focused specifically on tasks like robotic navigation, since depth is automatically generated per pixel, it is trivial to automatically reverse the depth values per pixel relative to the simulator's position and roll, pitch, and yaw to obtain a dense point cloud, see Figure 7. The point is, mapping can be automatically generated, algorithms can be tested across conditions, and/or models can be built and migrated to the real world. As VADER is a generic software and hardware stack, we only have to build these tools once and refine them later relative to a specific systems attributes, e.g., particular error, streaming, and format of a GPS and IMU sensor. However, it is worth noting that even these factors can be simulated if a goal is to understand their effects.

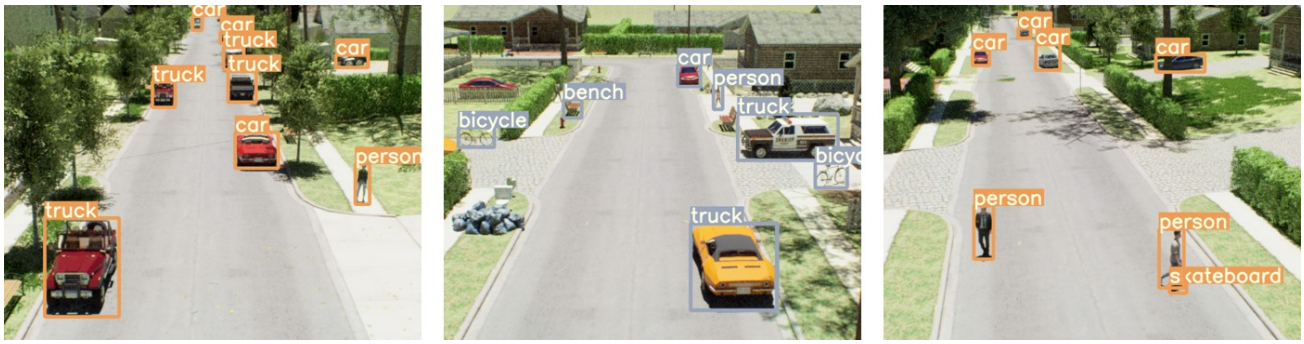

Figure 11: Examples of using a detection and localization algorithm, e.g., YOLOv5, to detect objects on simulated ray traced imagery from the Unreal Engine.

Localization, Detection, Semantic Segmentation, and Tracking: Another VADER application is testing and training of real-time computer vision algorithms. Herein, we highlight the use of the YOLOv5 detection and localization neural network; but other real-time similar algorithms such as CenterNet or the Faster RCNN could be used. Figure 11 shows example imagery from a modified scene in the Unreal Neighborhood Pack. We inserted objects like cars, people, trash, and bicycles. We used community available YOLOv5 model with full (not small) weights. The reader can see that simulated ray traced imagery "looks" good enough to enable reliable YOLOv5 detection's. As in real world imagery, YOLOv5 is not perfect and false alarms are present.

The image on the left is perfect (classification wise), whereas the middle image has a vehicle miss-labeled (truck versus car), an occluded car is missed, and far off objects (people and benches) are missed. Similarly, the right image in Figure 11 shows a skateboard incorrectly detected at the foot of a person, likely due to a shadow, which is a plausible detection given its context. The point is, VADER can be used to stress test algorithms. In our research, we make scripted cinematic flight sequences and also consider random placements of a UAS. We are interested in understanding where algorithms break, e.g., too few of pixels on target, illumination changes or shadows, variation in object color or sub-object categories, etc. To this end, VADER can automatically generate imagery and labels, a bottleneck in real world data and supervised learning techniques. As shown in Figure 5, object IDs, per object type or object instance, are automatically produced. We achieved this via custom Unreal shaders and stencil buffers. However, the recently released Unreal Movie Render Queue support automatic exporting of object IDs per pixel as well. From here, we automatically generate image space bounding boxes via the use of connected components and axis aligned bounding boxes (AABB). VADER automatically exports this data to a JSON file that can be directly used to score a system or train or update a neural network algorithm in PyTorch. Furthermore, this is a natural output that we also use directly to train modern semantic segmentation algorithms, e.g., UNet. The reader can refer to our recent article[23] on simulation for data augmentation to learn more about how to generate simulated data for real world algorithm training. In Ref. 23, we explored making realistic looking simulated scenes and we compared them to random scenes with dense random object placement. In Ref. 23 we showed better results from a model trained on simulated data versus real data for explosive hazard detection. A caveat is that application involved small class imbalanced datasets. Future work is required to understand how the proposed process compares to other tasks, e.g., person detection from a UAS, relative to benchmark community datasets. Last, as stated above, we develop all our codes in Python and ROS. As such, this generic architecture enables simple plug and play between a real UAS and simulation. For example, the system discussed in Ref. 23 runs in real-time on an NVIDIA Jetson at multiple frames per second.

Tracking: While above we outlined object detection and localization, in Ref. 24 Aggarwal et al. discusses a simulator approach to object detection and tracking in AirSim. No real-world port to a drone was discussed. In particular, Aggarwal used Deep SORT,[25,26] a real-time object tracking algorithm that uses YOLO weights. Furthermore, in Ref. 27 Kyungtae Ko used a combination of AirSim and deep reinforcement learning for optimal policy learning of an UAS. Again, Ref. 27 is focused on just simulation, but as discussed throughout this article nothing strictly prohibits it from being extended to a real-time UAS. The reader can refer to the community for a deeper literature search, beyond AirSim, on state-of-the-art in tracking.

Environmental Reconnaissance: While the above applications demonstrate generic functionality, e.g., mapping, detection, tracking, this subsection outlines a specific UAS use case. For example, a simulation for wildlife environment conservation using UASs in infrared via an AirSim extension called AirSim-W was put forth in Ref. 28. The authors considered an African savanna environment and the goal was to enable functions like counting animals, locating animals in parks, and finding poachers. The open source AirSim plug-in models IR imagery, under a simplified set of assumptions. In Ref. 29, Microsoft discusses additional projects, including distance response and detection and recovery of crashed aircraft.

(a) (b) (c) (d)

Figure 12: Example of simulated imagery produced by VADER for rapid prototyping of metadata driven contextual fusion in explosive hazard detection.[30] Example images show (a) higher versus (b) low altitude and (c,d) collection at different times of day.

Data and Information Fusion: Another use case is explosive hazard detection. In Ref. 30, we used

VADER to explore metadata driven contextual fusion for a low altitude UAS for explosive hazard detection. This application is an example where we were unable to travel and collect data with our collaborators due largely to COVID-19. In Ref. 30, VADER was used to generate controlled training and testing data. Specifically, we varied altitude (an example of platform metadata) and the time of day (an example of environmental metadata) in which data is collected relative to visual spectrum imagery (See Figure 12). While we focused on a small number of metadata, there is nothing that prohibits us from exploring variations in additional metadata. A set of detectors were trained per context and the Choquet integral (ChI) was used to fuse these models in and across context. Specifically, we presented an adaptive and online way to pick a ChI if the UAS recognizes its context, an algorithm was proposed to build a ChI if the UAS cannot distinguish its context, and a method for anomalous context identification and a default detection strategy was outlined. The point is, VADER was used in Ref. 30 to produce a range of controlled experiments that we could study and use to validate ideas before moving to increased time and cost real world experiments.

Cinematography: Another use case is the film and entertainment industries. In Ref. 31, Pueyo et al. outlined and relased an open source AirSim extension that allows users to control artistic elements in image generation. Namely, they extended the simulation capabilities of AirSim to have greater fidelity over various filming lens and common cinematographic properties. Whereas they demonstrated examples relative to movie production that looks closer to what a director would desire, plugins like this enable a greater range of high fidelity sensor simulations that go behind a full in-focus pinhole camera model.

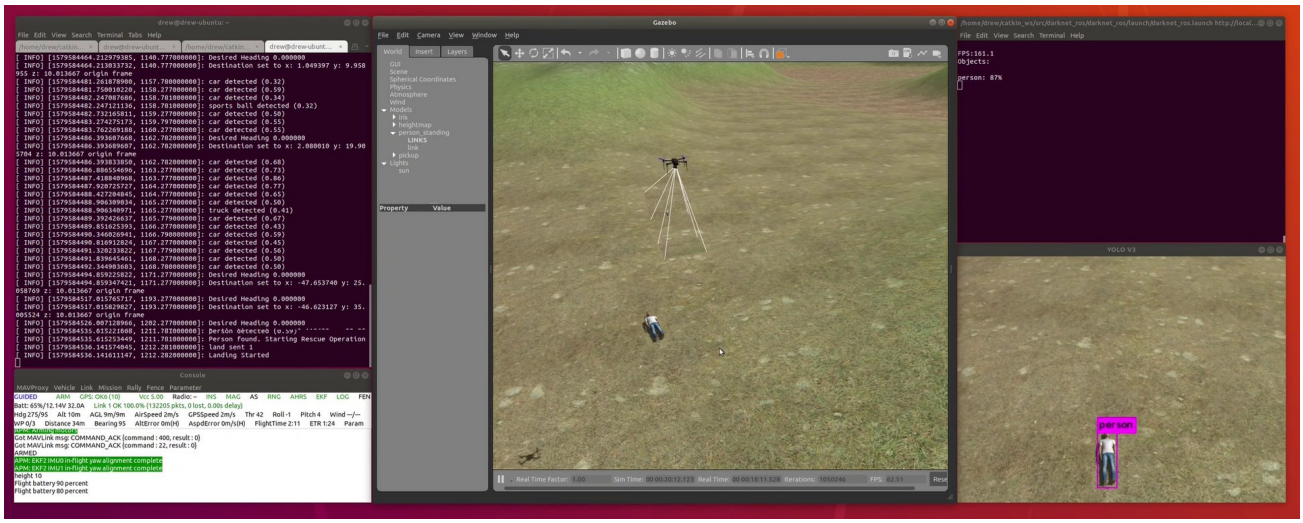

Figure 13: An example simplified search and rescue mission prototyped in Gazebo and ROS. Here the UAS flies in a structured lawnmower pattern and YOLO is run in real-time on the simulated drone imagery. When a person is detected, the drone begins a landing sequence, demonstrating the ability to respond to environmental stimuli. This example scenario extends the Intelligent Quads Tutorials.[32]

Search and Rescue (S&R): In this subsection we outline another use case. In Figure 13, we show a prototype for S&R with a UAS in VADER. In this setting, we use Gazebo and an existing UAS flight controller.[32] Open source libraries like these are useful for many researchers who do not want to concern themselves with the low level details of controlling a UAS but desire similar behavior, e.g., battery lifetime, quadrotor movement, etc. An advantage of using Gazebo over a simulator like the Unreal Engine is its community support for increased robotic functionality; via a combination of Gazebo and ROS libraries. However, the reader can see that there is clearly a trade off, increased robotic platform support versus rendering quality. In this example, we searched an open area by flying a popular lawnmower UAS pattern (parallel lines), versus something like a waffle pattern (parallel and perpendicular lines to maximize scene observably) that is used often for SfM. When the detector finds an object (person) using YOLO, the UAS sends a message and lands. Landing was simply selected to show a tie-in between UAS navigation and object detection. However, in the area of human-robot interaction there

exists research on optimal robotic design and behavior for medical triaging and emotional support during life threatening moments.[33] The purpose of this subsection is to show that VADER can tie together UAS control and detection to enable geospatial search. All of this code runs real-time in simulation and equally on an NVIDIA Jetson. The only piece that needs to be ported for real world operation is migration of the ROS flight controller (which we tackle using MavROS and Mavlink commands). While we highlight S&R, the tasks outlined here are equally applicable to our search and recovery (not rescue) of clandestine graves and surface human remains for forensic anthropology.[34] In search and recovery, the goal is not rescue but detection and documentation of a crime scene spatially and spectrally, as well as ways to interact with this data.

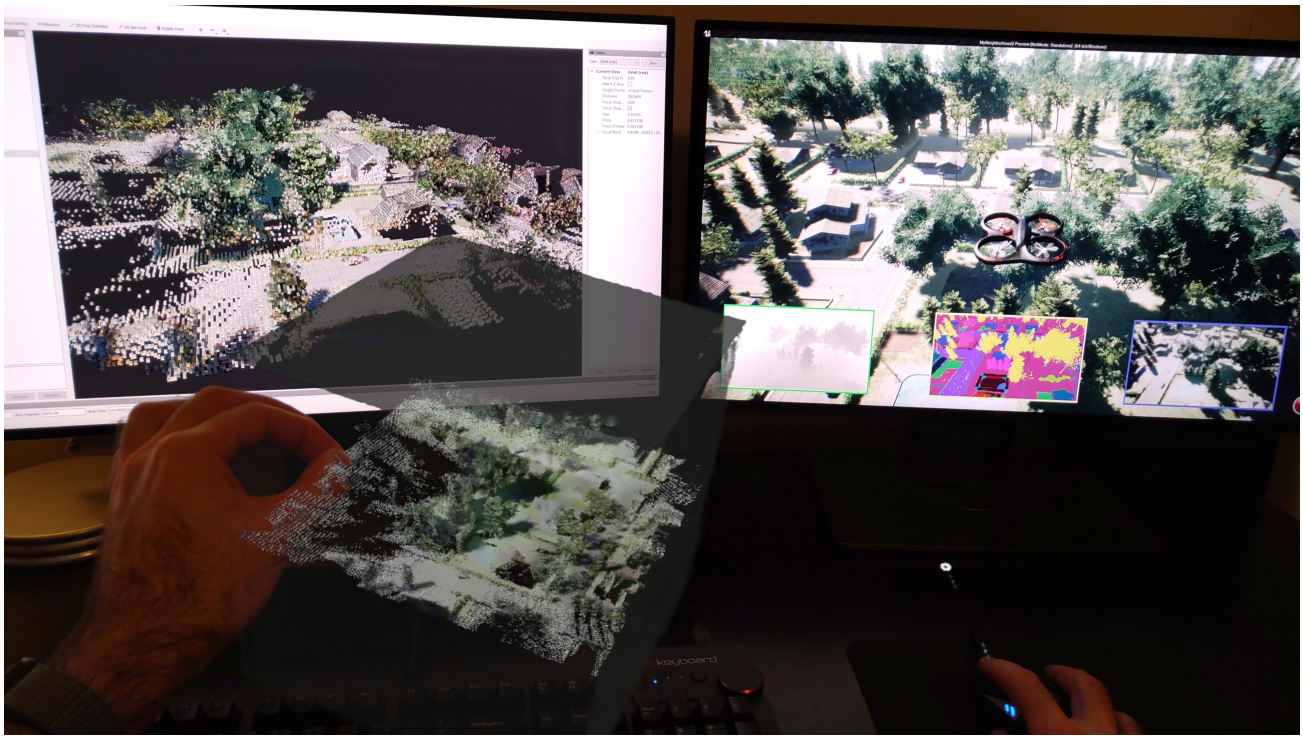

Figure 14: Example of (right) simulated Unreal scene with a UAS, (left) streaming voxelized model, and (bottom) real-time steaming scene that a user can interct with using Microsoft's HoloLens augmented reality headset. The full image is a snapshot from the AR headset, i.e., this is the view that a user sees.

Virtual and Augmented Reality: The last application we highlight here is VR and AR. As VADER can communicate easily with the Unreal Engine (or Unity) we can easily stream our simulated and real environments to the Oculus rift VR. The result is that a user can take the viewpoint of the UAS (either control it or just observe) or a user can walk around in the virtual world using a keyboard or the Oculus hand controllers. The shortcoming of VR is that a user is naturally anchored to one physical location, e.g., chair or where they are standing. VADER is equally able to naturally stream to AR. In Figure 14 we show the streaming of one of our drone flights to the Microsoft HoloLens AR headset. As Figure 14 shows, we can use AR to work with a manipulable 3D map in small scale or it can be expanded to the users world space scale. When at a user's scale, we find tie points to map the simulator local space into the AR/humans world space. This allows users to view scenes in a shared space with the UAS for purposes like mission planning (see our paper on UAS navigation in spatially attributed graphs[20]) or training. The human can now move around in a fully simulated space (relative to a physical open space), or if a UAS and user share the same physical space then UAS algorithm outputs can be rendered in the human's view or linguistically communicated.[20] In summary, we highlight that VADER is able to naturally interface in an abstract and minimalist fashion, again due to availability of current open source software tools, with VR and AR. In return, this enables a wealth of possibilities from collaborative spaces to training, playback, human-robot teaming, and more. In an academic setting, this is an exciting time period.

That is, ideas that were previously out of our reach due to technology and/or price are now a reality. These evolutions are likely to enable exciting powerful future research in shorter time increments.

5. CONCLUSION

The VADER platform represents our vision for the possibilities of drone autonomy research. The state of UAS hardware and simulation software has matured to a point that many different types of applications can be studied without requiring extensive knowledge of the low-level systems. We described in this article a hardware design and software architecture that can be set up with commercial off-the-shelf hardware and open-source software. This framework can be used in a generic research setting to facilitate the study of advanced automation techniques, visual perception and localization methods, human-robot teaming, synthetic data acquisition, and a variety of other applications. Our intent with this article was to document our approach to working with UAS hardware and software in the hope that other researchers facing similar research problems might gain insight into the capabilities and become inspired by the possibilities. Unmanned aerial systems offer an exciting research platform that can transform many existing research areas if the tools can be utilized. Likewise, simulated and virtual environments have the ability to accelerate research when physical hardware systems are unavailable. The future of UAS research as outlined by the VADER platform certainly appears to hold great potential.

REFERENCES

[1] "Sky Hero Spyder 6." Ultimate Hobby https://ultimate-hobby.com/products/sky-hero-spyder-6. (Accessed: 25 February 2021).

[2] "The Cube User Manual 1.0." https://docs.cubepilot.org/user-guides/autopilot/the-cube-user-manual. (Accessed: 25 February 2021).

[3] "ArduPilot." https://ardupilot.org. (Accessed: 25 February 2021).

[4] "Kore Carrier Board." https://docs.cubepilot.org/user-guides/carrier-boards/kore-carrier-board. (Accessed: 25 February 2021).

[5] "Jetson TX2." https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/. (Accessed: 25 February 2021).

[6] "Spaceley Carrier for NVIDIA® Jetson™ TX2/TX2i/TX1." https://connecttech.com/product/spacely-carrier-nvidia-jetson-tx2-jetson-tx1/. (Accessed: 25 February 2021).

[7] "Mavlink." https://mavlink.io/en/. (Accessed: 1 March 2021).

[8] "Robot Operating System (ROS)." https://ros.org. (Accessed: 25 February 2021).

[9] "MAVROS." http://wiki.ros.org/mavros. (Accessed: 1 March 2021).

[10] "Mission Planner." https://ardupilot.org/planner/. (Accessed: 7 March 2021).

[11] "MAVProxy." https://ardupilot.org/mavproxy/. (Accessed: 1 March 2021).

[12] "Gazebo." http://gazebosim.org/. (Accessed: 1 March 2021).

[13] "AirSim." https://github.com/microsoft/AirSim. (Accessed: 1 March 2021).

[14] "Unreal Engine." https://www.unrealengine.com/. (Accessed: 1 March 2021).

[15] "OpenDroneMap." https://www.opendronemap.org/. (Accessed: 2 March 2021).

[16] Godard, C., Mac Aodha, O., Firman, M., and Brostow, G. J., "Digging into self-supervised monocular depth prediction," (October 2019).

[17] "Open3D." http://www.open3d.org/. (Accessed: 1 March 2021).

[18] Mur-Artal, R. and Tardós, J. D., "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics* **33**(5), 1255–1262 (2017).

[19] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots* **34**, 189–206 (Apr. 2013).

[20] Buck, A. R., Anderson, D. T., Keller, J. M., III, R. H. L., and Scott, G., "A fuzzy spatial relationship graph for point clouds using bounding boxes," in [*under review by FUZZ-IEEE*], (2021).

[21] Godard, C., Mac Aodha, O., Firman, M., and Brostow, G. J., "Digging into self-supervised monocular depth estimation," in [*Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*], (October 2019).

[22] Schonberger, J. L. and Frahm, J.-M., "Structure-from-motion revisited," in [*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*], (June 2016).

[23] Alvey, B., Anderson, D. T., Keller, J. M., Buck, A., Scott, G., Ho, D., Yang, C., and Libbey, B., "Improving explosive hazard detection with simulated and augmented data for an unmanned aerial system," in [*SPIE*], (2021).

[24] "SmartDrone." http://kastner.ucsd.edu/ryan/wp-content/uploads/sites/5/2014/03/admin/SmartDrone-Final-Report.pdf. (Accessed: 1 March 2021).

[25] Wojke, N. and Bewley, A., "Deep cosine metric learning for person re-identification," in [*2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*], 748–756, IEEE (2018).

[26] Wojke, N., Bewley, A., and Paulus, D., "Simple online and realtime tracking with a deep association metric," in [*2017 IEEE International Conference on Image Processing (ICIP)*], 3645–3649, IEEE (2017).

[27] Ko, K., "Visual object tracking for uavs using deep reinforcement learning," in [*Ph.D. dissertation, Iowa State*], (2020).

[28] Bondi, E., Dey, D., Kapoor, A., Piavis, J., Shah, S., Fang, F., Dilkina, B., Hannaford, R., Iyer, A., Joppa, L., et al., "Airsim-w: A simulation environment for wildlife conservation with uavs," in [*Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*], 40, ACM (2018).

[29] "AI UAS AirSim." https://www.microsoft.com/en-us/ai/ai-lab-airsim-drones. (Accessed: 2 March 2021).

[30] Deardorff, M., Alvey, B., Anderson, D. T., Keller, J. M., Scott, G., Ho, D., Buck, A., and Yang, C., "Metadata enabled contextual sensor fusion for unmannedaerial system-based explosive hazard detection," in [*SPIE*], (2021).

[31] Pueyo, P., Cristofalo, E., Montijano, E., and Schwager, M., "Cinemairsim: A camera-realistic robotics simulator for cinematographic purposes," (2020).

[32] "Intelligent Quads Tutorials." https://github.com/Intelligent-Quads/iq_tutorials. (Accessed: 3 March 2021).

[33] Bethel, C. L. and Murphy, R. R., "Non-facial and non-verbal affective expression for appearance-constrained robots used in victim management," in [*Journal of Behavioral Robotics*], **1** (2011).

[34] Murray, B., Anderson, D. T., Wescott, D. J., Moorhead, R., and Anderson, M. F., "Survey and Insights into Unmanned Aerial-Vehicle-Based Detection and Documentation of Clandestine Graves and Human Remains," *Human Biology* **90**(1), 45 – 61 (2018).