This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JMASS.2020.3027520, IEEE Journal on Miniaturization for Air and Space Systems

”CloudStation”: A Cloud-based Ground Control Station for Drones

Lyuyang Hu, Omkar Pathak, Zeyu He, Hunkyu Lee, Mina Bedwany, Jace Mica, Peter J. Burke, *Sr. Member, IEEE*

***Abstract*—This paper demonstrates an open source cloud based ground control station that allows remote piloting of drones (aerial, land, or sea based vehicle) from anywhere in the world with just a web browser. A standard client-server architecture is used for pilot-server and drone-server communication and it is designed to be scalable for the control of millions of drones simultaneously. CloudStation, a proof of concept web app, is built with Django and hosted on a cloud based service provider with global availability and scalability. A demonstration from two antipode points (separated around the world) shows cloud based command and control of a remote rover.**

***Keywords*—Drone, 4G, autonomous vehicle.**

I. Introduction

UNMANNED vehicles such as aerial vehicles (quad rotors, fixed wing), rovers and self driving cars, and sea vessels can be piloted remotely via a radio or even internet link, or can perform autonomous missions. A Ground Control Station (GCS) is a critical component of the unmanned vehicle system because it provides the operator with full control of the vehicle's parameters, feedback on its location, and ability to program waypoint missions or even pilot manually. In a typical system, the GCS software is installed on a desktop/laptop accessed locally by the pilot. This requires that the pilot always has a desktop/laptop and a locally compiled copy of the GCS software. In some cases the GCS code is compiled on a smartphone or tablet. However in all cases, without the compiled code installed on the specific hardware, the pilot cannot access or control the drone remotely.

It would be an advantage in simplicity, mobility and scalablity if the pilot could access the ground control station through a simple web browser, instead of requiring installation of software locally. Since in most long range control cases, the traffic from GCS to drone occurs over the internet, having a cloud based server manage the communications and commands is a low cost technique to insert between the pilot and the drone.

In this paper, we implement a cloud based GCS software that uses TCP and HTTP protocol over Wifi, 4G/LTE, 5G to facilitate communication between pilots and drones. The cloud based GCS provides a graphical user interface (UI) for pilots to monitor and control drones from anywhere there is a browser and internet connection. The architecture, as shown schematically in Fig. 1 (and Table 1), can be used to control any vehicle on land, air, and sea. We demonstrate the system with a land based rover using a remote pilot controlling the vehicle with a website from around the world. Because our server is deployed in the cloud, it is scalable and can in principle be used to manage and control fleets of millions of drones by a team of pilots.

Peter J. Burke is with the Department of Electrical and Engineering and Computer Science, University of California, Irvine, CA, 92697 USA e-mail: pburke@uci.edu

Manuscript received August 31, 2020.

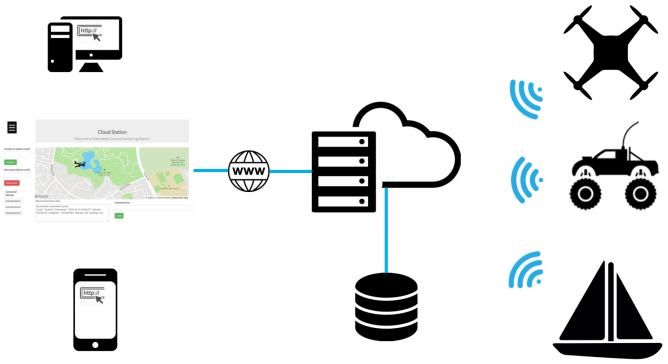


Fig. 1. System overview. The cloud based server maintains all command and control function of the vehicles. The local client, a browser or a mobile app, continuously shows the vehicle status and position.

II. Architecture

CloudStation is divided into three parts: the vehicle, the web server and the client (Fig. 1, 2). Our architecture uses a browser at the pilot end, a Linux server in the cloud, and a Linux companion computer on the drone interfaced with a flight controller running Ardupilot. The communication between the drone and the cloud server is over UDP or TCP/IP and uses the MAVLink protocol, a well-established and well-tested protocol [1].

The cloud based GCS resides on a Linux server. The software is implemented with Django, a Model-View-Controller web framework. The controllers validate and pre-process the inputs from drones and users. The telemetry information and commands are then sent to the models where they are processed. The views provide a visual representation of the drone status to the users. For drones, the input to controllers are in the form of MAVLink traffic and pyMAVLink is used to parse the commands. For users, the inputs are in the form of application programming interface (API) requests. The current API implementation resembles the MAVLink structure rather than following the more popular REST or SOAP structure. This design choice has made it easier to convert requests to MAVLink commands but as discussed in section VII, advanced features and future developments can benefit from a REST structure.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JMASS.2020.3027520, IEEE Journal on Miniaturization for Air and Space Systems

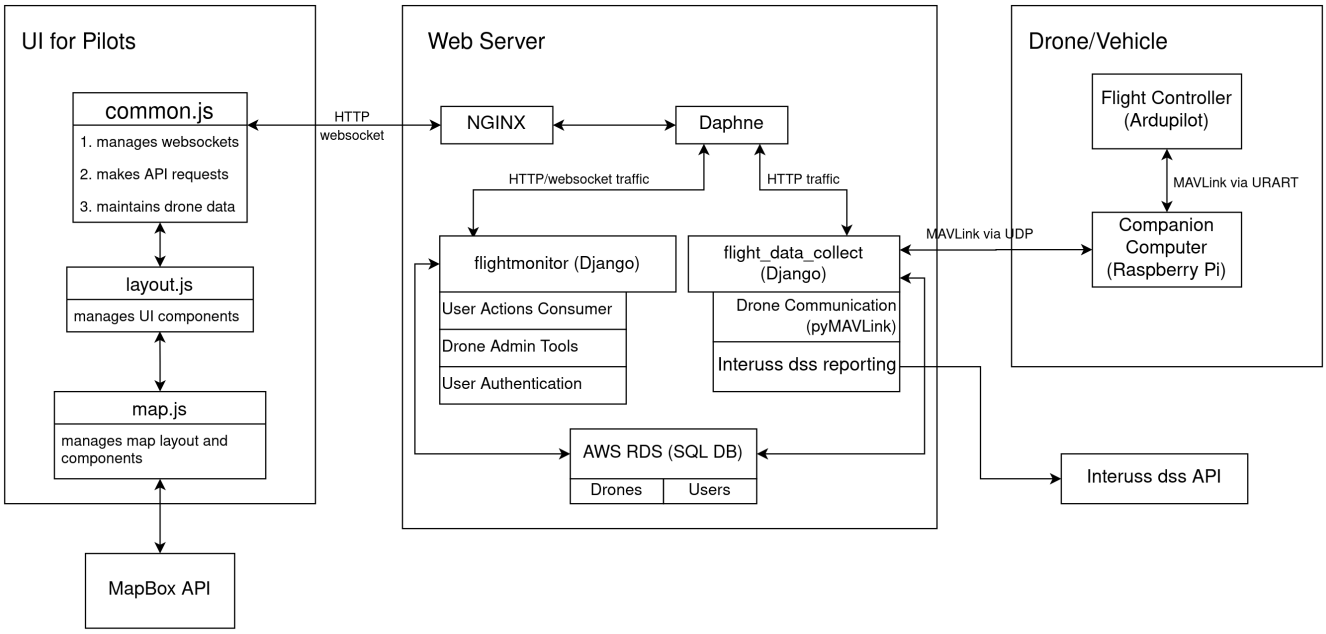


Fig. 2. System internal diagram.

Abbreviations	Meaning
GCS	Ground Control Station
AWS	Amazon Web Services
RDS	Relational Database Service
MAVLink	Micro Air Vehicle Link
UI	User Interface
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
STM	STMicroelectronics
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language

The prototype uses a Structured Query Language (SQL) database to store the current status of the drones. It also provides the option to store all location and telemetry data. In advanced uses, a Relational Database Service (RDS) can be used to manage the data so it can be scaled to more drones and users.

The backend of the web app uses Django and Django Channels. When a request reaches the server, NGINX directly serves static files and redirects the rest of the traffic to Daphne. Daphne redirects the traffic again to different Django apps according to the connection type and the URLs. To display real-time vehicle location and telemetry on the webpage, WebSocket (Django Channels) is used to maintain a continuous communication channel between the user and the server. A websocket channel is opened for each connected user. The received MAVLink packages are parsed by the controller and the vehicle status stored in the database is updated by the models. The vehicle status data is well-structured so a relational database is used which can be configured to store drone information temporarily or permanently. As soon as the controller finishes parsing the messages, the relevant messages are appended to a Redis message queue before they get sent to the users.

III. Features and Implementation

A. User authentication and registration

Our application includes a user registration and authentication system so that only registered users can access the features of our application like connecting to a vehicle and sending commands. We use Django's built-in methods for user account management and authentication. Users can login, logout, register, or view as guest with limited privileges, in a user friendly UI. All users have full control of the drones with the current implementation and we plan to extend the authentication system so each user can be assigned with different roles. The goal is for CloudStation to give users a way to work in team such that each member can control a different subsystem of the drones.

CloudStation provides an Admin page where admin users can manage the records (drone data) stored in the database. We plan to add more features to the page so system administrators can monitor system status and assign different roles (pilot, co-pilot, gimbal operator, etc.) to users.

B. Connecting to Drones

After the users are logged into the system, they will have the option to connect to and disconnect from drones. A graphical UI is provided for users to monitor drone status and send commands. Users use unique drone addresses for connection and disconnection. The connected drone IDs will be displayed at the left bottom under the "Connected Vehicle" label. Their information, like location, telemetry, and flight mode, will be shown on the right side of the screen. A monitor at the bottom

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JMASS.2020.3027520, IEEE Journal on Miniaturization for Air and Space Systems

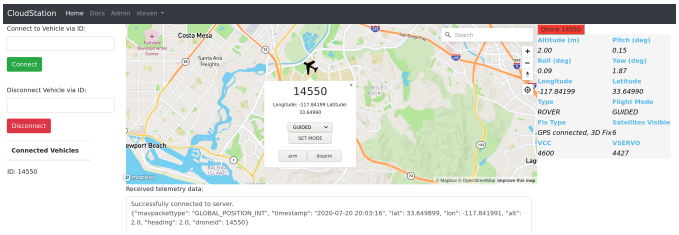


Fig. 3. Real-time vehicle location tracking on a map

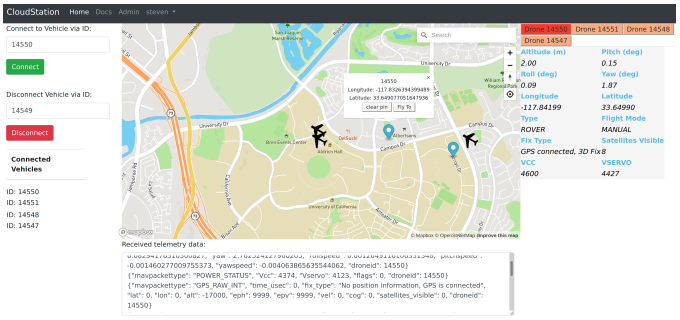


Fig. 4. Initiating “Flyto” command on the map.

of the UI displays MAVLink messages as they are received by the user (Fig. 3). This is a useful debug feature that can be disabled in production.

C. Real-time vehicle status

As the server receives real-time location packages from our hardware setup (Raspberry Pi, Flight controller, GPS module), we parse these packages for telemetry and location data including yaw, roll, speed, latitude, longitude, etc (Fig. 2). This information is stored in a relational database.

The same data (location and telemetry) is sent to the client if the drone is registered (”connected”) by an active user. These JSON messages are then parsed to form an object that stores drone status in the browser memory. A location marker is created on the map once the client receives longitude and latitude data. We use Mapbox API to display the map and the location of the drone. The plane icon on the map moves in real-time and as more location information is received, the user can see the icon moving continuously on the map (Fig. 3).

D. Multiple drones (drone swarms)

Once vehicles are connected using unique drone IDs, users can view flight information and change between multiple drones using the tables on the right side of the screen.

E. Flight modes

Users have the ability to change the flight mode as well. As shown in Fig. 3, they can click on the drone icon on the map, and it will show a pop-up box where they can choose from a list of supported modes.

F. Fly to a chosen location

The user can drop a pin on the map to set as a waypoint for the vehicle to fly to. As shown in Fig. 4, once a pin is dropped on the map, clicking on the pin opens a pop-up box on the map that allows the user to clear the pin or initiate a ”fly to specified location” command by clicking on the “Flyto” button. The vehicle will receive the waypoint information and start moving to the specified destination. The real-time vehicle location is displayed on the map as it moves.

On the backend, the server sends a command to the vehicle to first change mode to “Guided” and then uploads the waypoint to the flight controller memory. Once the user sends an “arm” command, the vehicle will arm its motors and start to go to the specified location. The onboard autopilot system performs several safety checks to ensure the vehicle has a strong GPS signal, enough battery life and the onboard sensors work properly. A future goal of our project is to display the safety check information so the users can easily debug if the autopilot system determines it is not safe to arm the vehicle (Fig. 4).

G. Continuous Integration/ Continuous Delivery

We have open sourced our project and published the code on Github. Since we have a number of collaborators, we have built an end to end pipeline to deploy the open source code on the cloud based servers. For this purpose, we use two Amazon Web Services (AWS) resources called CodePipeline and CodeDeploy. Our Github organization consists of multiple repositories, one of which contains the backend code that is deployed on the server. AWS CodePipeline is authorized to access this repository using web hooks. Every time changes are made to the master branch on this repository, the CodePipeline is triggered and it pushes the changes on to the server. This process also makes sure that the appropriate server commands are executed prior and post the sync. As we continue to make this pipeline more robust, we plan to set up a separate dev and prod environment for developer testing.

H. Future work

The UI only displays the basics for now. A customizable UI with all the drone parameters would be a good next step. Manual control with integrated HD video would also be valuable. Our architecture has laid the foundation for this and is designed in a way that it can scale.

IV. System Demonstration

In order to demonstrate the system operation, we used a micro-rover with Ardupilot and a 4G/LTE modem, described in detail in [2]. Briefly, an STM32 based micro controller (Omnibus F4 Pro) was used to run the Ardupilot firmware, and a Raspberry Pi Zero W was used as the on board companion computer to pass the MAVLink traffic from Ardupilot to the CloudStation. The CloudStation was deployed on a Linux instance provided by a cloud based service provider.

To demonstrate the reach, we contacted ”Les Petits Débrouillards de La Réunion”

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JMASS.2020.3027520, IEEE Journal on Miniaturization for Air and Space Systems

Fig. 5. Antipode control, adapted from [3].

(https://lespetitsdebrouillards.re/), an organization aiming to develop interest in sciences and technologies among youths, supported by the ENERGY-lab of the University of La Réunion. A group of young people from the city of Saint-Denis managed to log onto our Cloudstation website and connected to the micro rover. The location chosen was 18,500 km away from California (Reunion Island, France), and is almost as far as two points can be on Earth (antipode). Also this was an interesting educational activity for the student group in France. They were able to send mode change commands and send a "fly to" command to have the rover drive to a specific location determined randomly by the students. Thus, the demonstration shows that our software can be used by anyone with a web browser from as far away as around the world (antipode) (Fig. 5). Note we recently performed this with a Mission Planner station running on a Windows machine in the cloud [3], but that was sub-optimal as the interface was remote desktop which is slow and clunky compared to the web browser interface used in this paper.

Although this was done for a micro rover, the same technology could be applied for drones, flying wings [4], sea vessels, etc.

V. Comparison to previous work

Web or cloud-based ground control stations (GCSs) usually aim to solve three major problems of radio controllers and traditional offline GCSs like Mission Planner: limited control range, limited ability to support multiple operators and limited ability to control multiple drones simultaneously. To have a fully functional GCS, the software needs to have the ability to receive, interpret and send messages to the vehicle via a communication protocol for unmanned vehicles such as MAVLink[1]. Prior work has focused on either extending control range by streaming MAVLink messages to a local GCS via the internet, or incorporating some features of the Mission Planner GCS software on web app. We review both of these categories next.

A. Prior Architecture I: Streaming MAVLink messages to local GCS via internet

Streaming MAVLink from drone to GCS at the pilot's physical location work is not the goal of this paper, but very related, so we review it here. See also the review in our recent paper[4].

1) XB-UAV: XB-UAV[5] is a subscription based (VPN) connection software solution between a 4G modem and a Windows or Linux GCS running Mission Planner or QGroundcontrol. The service provides connectivity only between drone and PC, and does not have any cloud basis for the control station software. The company recently started offering some 4G modem hardware to connect a 4G modem to Pi with Mini-PCI interface, instead of USB.

2) Envirover: Envirover SPL[6] is a global satellite telemetry system for autonomous vehicles controlled by ArduPilot or PX4 autopilots. It provides Iridium based connectivity and cloud based networking, however the control station software (Mission Planner and/or QGroundcontrol) still reside on the pilot's local laptop/desktop, not in the cloud. There is a cloud based storage of the MAVLink traffic sent between the drone and local ground control station. The software is freeware/open-source and offered a pre-configured "Stack" on AWS.

3) Burke: One of us[4] developed an open source networking solution to stream Mavlink from a drone to a GCS over 4G/LTE and 5G, using double reverse ssh that allows penetration through firewalls. We developed it and showed it to work on the smallest 4G connected areal platform[3] (a flying wing at 300 g all up weight) as well as the smallest 4G/LTE connected microrover [2] (1/16 scale model at 900 g.). In contrast to prior proprietary solutions, this work is open source.

4) UAVcast: UAVcast[7] is software that runs on a Raspberry Pi companion computer on the drone to establish, together with ZeroTier, a VPN to allow a local GCS to connect to the drone over 4G/LTE and 5G.

B. Prior Architecture II: Cloud based GCS

There have been some false starts and one proprietary solution with similar architecture to ours discussed below.

1) Hawkeye: Project HawkEye was a short lived web based GCS built with Node.js and Socket.io [8]. Only the announcement of the project starting, but not the code, was published. Another proposal to use this for controlling marine based search and rescue was floated[9] but it never was performed.

2) Inoue: Innoue in Japan[10] installed Windows on a cloud computer (Amazon EC2 instance), and installed Mission Planner on that Windows instance as the GCS. Mission Planner was controlled by remote login to the EC2 instance over screen share. We did a similar thing in our recent antipode paper [3]. This is not ideal since the local pilot has to access a Windows Machine running in the cloud through pixel based screen share instead of a web browser as we do in this paper.

3) FlytBase: FlytBase is a seed stage startup founded in 2017 in India, and is the closest comparable project to this paper[11]. The on-board FlytOS software is based on ROS and it provides APIs for users to monitor and control the drone. The FlytCloud software extends control to a cloud based server, similar to this paper, but the software is proprietary, subscription based (SAAS) on servers run by the company, and no details are known publicly about the technology deployed on the servers (i.e. it is a closed source, proprietary solution). There is an API which extends MAVLink, similar to DroneKit. FlyteBase also provides various high level drone applications like computer vision, drone swarms and navigation.

4) Koubâa: In[12], Koubâa et. al. propose a high level description of Dronemap Planner, a service-oriented cloud-based management system for drones. Using the MAVLink and ROSLink protocols, it virtualizes access to drones through web services (SOAP and REST).

C. What is new about this work

Our work is open source and the code is publicly available. Our work is most similar to Flytbase, but we do not invent or add an extra API. Our entire network, hosting strategy is laid out publicly and clearly in this paper, and in the source code. The user command and control is from an open source web-page that enables a pilot to control the drone from anywhere in the world with a mobile device and a web browser. Thus, this paper is the first publication demonstrating the detailed architecture, implementation, and demonstration of a cloud based ground control station for drones.

VI. Applications

A. FAA InterUSS: Integration with Unmanned Traffic Management (UTM)

As drones become more sophisticated and ingrained in society, regulation and standards must be adopted. A proposal by the Federal Aviation Administration (FAA) published on 12/31/2019 would require all unmanned aircraft systems to adhere to remote identification. Currently, the docket is under public dissection and awaits to be enacted. In order to get ahead and accommodate this, CloudStation aimed to implement InterUSS[13], an API joint-developed by AirMap, Alphabet's Wing, and Kittyhawk.io. InterUSS facilitates communication and drone operations between UAS Service Suppliers in accordance with UAS Traffic Management standards outlined by the aforementioned FAA proposal.

We were able to setup an instance of the DSS on a local machine using the given developer resources, however our attempts to make API calls were unsuccessful. We believe this to be due to the preliminary nature of InterUSS's development. As both the FAA's proposed rule and InterUSS are in their infancy, the API's current iteration is not equipped to be fully functional at this time, but could easily be implemented as part of our framework.

B. Machine learning and artificial intelligence: Integration with the Cloud

Recently, there have been many papers that cover topics such as communications networks from drone-to-drone or drone-to-ground[14], [15], [16], [17], [18], [19], [20], [21], [22]. This indicates that the idea of internet connected drones is an important and emerging concept, although our cloud based approach has not appeared in any IEEE journals. An advantage of this cloud based approach, which has not appeared in any of the cited IEEE papers on drone communications, is the potential ability to integrate real-time environmental data and, more generally, situational awareness, that is already available in the cloud through other sources (such as the National Weather Service) to provide navigational adjustment and course correction in response to changing situation. Such a situational awareness could leverage artificial intelligence and machine learning in the cloud with autonomous control of drones. For example, an recent paper in this journal demonstrated through simulations that a 50 percent reduction in battery usage for UAVs can be achieved by incorporating meteorological data into the flight path planning process[23].

VII. Discussion

A. MAVLink Processing

We heavily relied on pyMAVLink to parse and send MAVLink messages in this project. MAVProxy is a well-known command line style GCS software that uses pyMAVLink. CloudStation and MAVProxy both aim to control and monitor vehicles but we focused on providing a graphical user interface and adding support for multi-user and multi-vehicle. The interface we used to interact with MAVLink messages followed the design of MAVProxy. We also reused some of the source code from MAVProxy to perform message validation and parsing.

We did not use the architecture provided by MAVProxy directly because it was designed to run on an offline, standalone PC. It continues to run until the software is terminated and it saves state information in memory. This design is not ideal for a web application because the software relies on in-memory state information so it does not provide an easy way for multiple users to access the same vehicle. It also consumes server resources even when the vehicle connection is idle. To design a highly available and somewhat stateless web app, CloudStation stores vehicle state information in databases and in the browser memory. Each MAVLink message is treated with a separate process and the process terminates as soon as the specific task finishes.

B. API Design

The current API design resembles the design of MAVLink messages to provide an easy way to parse and create MAVLink messages with pyMAVLink. An obvious alternative design is the RESTful architecture. CloudStation can maintain a separate JSON file for each drone. Future developments of CloudStation can benefit from such a design since it is more versatile and it is easier to plug in new, complex modules to such a system.

C. Extensibility

The current implementation of CloudStation doesn't provide much extensibility. One of our goals is to provide a middle

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JMASS.2020.3027520, IEEE Journal on Miniaturization for Air and Space Systems

layer that makes it possible for developers to easily implement external modules that communicate with CloudStation. These modules can be hosted with Docker containers as what we did with Redis and they can provide advanced features including computer vision and drone formation.

D. Hardware and software agnostic

This system is hardware and software agnostic, as long as the vehicle is using Ardupilot and can connect in any way to the cloud/internet (e.g. 4G/LTE, 5G, Wifi, satellite such as Iridium, or any other method).

E. Latency

We did not quantify the latency, but it did not pose a factor for guided, waypoint missions with a slow ground vehicle which we demonstrated with "Les Petits Débrouillards de La Réunion". Real-time video feed also was functional. Future work will be needed to improve the latency to allow for real-time fully manually piloting of high speed vehicles. However, the most typical use case would not be fully manual "stick and rudder" aggressive maneuvers. This software approach is more suitable to the typical use case scenario of a guided or waypoint mission with some degree of autopilot input at the drone side.

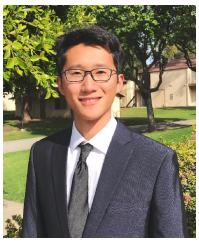
VIII. Conclusion

CloudStation is an open-source, cloud-based web app that helps pilots monitor and control multiple drones over the internet. The application provides a user-friendly UI and incorporates a secure user authentication system. We built the software to support MAVLink so it is suitable for a wide variety of drones in the Ardupilot ecosystem.

IX. Acknowledgement

The CloudStation team would like to sincerely thank our project advisor, Professor Peter Burke, for being always in support of our project and vision for our senior design team, and for providing guidance and advice on any concern or issue we faced throughout all stages of this project. We would also like to thank the EECS 159 class instructor, Professor Stuart Kleinfelder, and his Teaching Assistants, Beverly Quon, Ali Tazarv, Sadjad Sedighi, Ned Beigiparast, and Sahar Aali, for their constant guidance and financial support. We would also like to thank UC Irvine and UROP for providing us with the opportunity to work on such a project and helping us with any needed resources like lab rooms, tools, online resources, and funding for purchases. Finally, we thank the group of young people in Reunion Island supervised by Jean-Nicolas Surjus, director of the the organization "Les Petits Débrouillards de La Réunion" and Patrick Jeanty, research engineer of the ENERGY-lab, for beta testing our software.

PETER J. BURKE (M'02–SM'17) received the Ph.D. degree in physics from Yale University, New Haven, CT, USA, in 1998. From 1998 to 2001, he was a Sherman Fairchild Postdoctoral Scholar in physics with the California Institute of Technology, Pasadena, CA, USA. Since 2001, he has been a Faculty Member with the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA, USA. He has been the recipient of the Office of Naval Research Young Investigator award, and the Army Research Office young investigator program award. His current research interests include nanoelectronics, probes of cellular and sub cellular electrophysiology, radio and antenna propagation and systems (including nano-electromagnetics), and drones. The author has been a Part 107 licensed pilot (certificate 390342, issue date 31 August 2016) since the beginning of the FAA drone license program.



Lyuyang Hu received the B.S. degree in Computer Science and Engineering from University of California at Irvine, Irvine, CA, USA, in 2020. He is interested in software systems and natural language processing. He will join the Language Technologies Institute at Carnegie Mellon University in 2021 to continue his study in Computer Science and Language Technologies.

Omkar Pathak received the B.S. degree in Computer Science and Engineering from University of California at Irvine, Irvine, CA, USA, in 2020. He was a honors student at the school of Information and Computer Science and a research fellow at the Undergraduate Research Opportunity Program.

Mina Bedwany received the B.S. degree in Computer Science and Engineering from University of California at Irvine, CA, USA, in Winter 2020. He is interested in firmware and embedded systems. He will be joining Western Digital as an ASIC Systems Design Engineer in Summer 2020.

Jace Mica received the B.S. degree in Computer Science and Engineering from University of California at Irvine, CA, USA, in Fall 2020. He is interested in algorithmic trading and fintech. Following a Summer 2020 internship, he will be pursuing new grad software engineering roles.

Zeyu He received the B.S. degree in Computer Science and Engineering, and Civil Engineering from University of California at Irvine, CA, USA, in 2020. He is interested in software development and natural language processing. He will join Personable Inc as a software engineer in Summer 2020.

Hunkyu Lee received the B.S. degree in Computer Engineering from University of California at Irvine, CA, USA, in 2020. He is interested in GPU Design and Computer Vision. He is working on self-driving car project with Prof Jeon from UC Merced.

REFERENCES

[1] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey," *IEEE Access*, 2019.

[2] P. Burke, "A 4G connected micro rover with infinite range," *IEEE Journal on Miniaturization for Air and Space Systems*, 2020.

[3] P. Burke, "4G Antipode: Remote Control of a Ground Vehicle From Around the World," *IEEE Journal on Miniaturization for Air and Space Systems*, 2020.

[4] P. J. Burke, "A Safe, Open Source, 4G Connected Self-Flying Plane With 1 Hour Flight Time and All Up Weight (AUW) $<$300 g: Towards a New Class of Internet Enabled UAVs," *IEEE Access*, vol. 7, pp. 67 833–67 855, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8718270/

[5] XBUAV, "XBStation Docs." [Online]. Available: https://docs.xbstation.com/

[6] Envirover, "SPL Global Telemetry," 2020. [Online]. Available: http://envirover.com/docs/spl.html

[7] UAVMatrix, "UAVMatrix." [Online]. Available: https://uavmatrix.com/

[8] Johan, "EMLID Community," 2017. [Online]. Available: https://community.emlid.com/t/project-hawkeye-cloud-control-station/5503

[9] F. Falkan, "SURTSEY.ORG," 2017. [Online]. Available: http://www.surtsey.org/projects/cloud-control-station/

[10] S. Qazi, A. S. Siddiqui, and A. I. Wagan, "UAV based real time video surveillance over 4G LTE," in *ICOSST 2015 - 2015 International Conference on Open Source Systems and Technologies, Proceedings*, 2016.

[11] Flytbase, "FlytDocs." [Online]. Available: https://docs.flytbase.com/

[12] A. KOUBAA, A. Koubâa, B. Qureshi, M.-F. Sriti, A. Allouch, Y. Javed, M. Alajlan, O. Cheikhrouhou, M. Khalgui, and E. Tovar, "Dronemap Planner: A service-oriented cloud-based management system for the Internet-of-Drones," *Ad Hoc Networks*, vol. 86, pp. 46–62, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870518306814

[13] Wing, Uber, Airmap, Federal Office of Civil Aviation Switzerland, "InterUSS Project." [Online]. Available: https://interussplatform.org/

[14] X. Lin, V. Yajnanarayana, S. D. Muruganathan, S. Gao, H. Asplund, H. L. Maattanen, M. Bergstrom, S. Euler, and Y. P. Wang, "The Sky is Not the Limit: LTE for Unmanned Aerial Vehicles," *IEEE Communications Magazine*, 2018.

[15] Q. Zhang, M. Jiang, Z. Feng, W. Li, W. Zhang, and M. Pan, "IoT Enabled UAV: Network Architecture and Routing Algorithm," *IEEE Internet of Things Journal*, 2019.

[16] R. Amorim, H. Nguyen, P. Mogensen, I. Z. Kovács, J. Wigard, and T. B. Sørensen, "Radio Channel Modeling for UAV Communication over Cellular Networks," *IEEE Wireless Communications Letters*, 2017.

[17] L. Gupta, R. Jain, and G. Vaszkun, "Survey of Important Issues in UAV Communication Networks," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1123–1152, 2016.

[18] A. Al-Hourani and K. Gomez, "Modeling Cellular-to-UAV Path-Loss for Suburban Environments," *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 82–85, 2018.

[19] J. Jiang and G. Han, "Routing Protocols for Unmanned Aerial Vehicles," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 58–63, 2018.

[20] B. Van Der Bergh, A. Chiumento, and S. Pollin, "LTE in the sky: Trading off propagation benefits with interference costs for aerial nodes," *IEEE Communications Magazine*, 2016.

[21] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: Opportunities and challenges," *IEEE Communications Magazine*, 2016.

[22] P. J. Burke, "Demonstration and application of diffusive and ballistic wave propagation for drone-to-ground and drone-to-drone wireless communications," *Scientific Reports*, vol. 10, no. 1, pp. 1–12, 2020. [Online]. Available: https://doi.org/10.1038/s41598-020-71733-0

[23] A. R. Hovenburg, F. A. d. A. Andrade, R. Hann, C. D. Rodin, T. A. Johansen, and R. Storvold, "Long range path planning using an aircraft performance model for battery powered sUAS equipped with icing protection system," *IEEE Journal on Miniaturization for Air and Space Systems*, 2020.

APPENDIX A
SOURCE CODE

The source code of the project as well as deployment and user guides can be found at:

https://github.com/CloudStationTeam.