Starling: Containerisation Architecture for Scalable Local Development, Deployment and Testing of Multi-UAV Systems

Mickey Li
Bristol Robotics Laboratory
University of Bristol
Bristol, UK
Email: mickey.li@bristol.ac.uk

Robert Clarke
Bristol Robotics Laboratory
University of Bristol
Bristol, UK
Email: robert.clarke@bristol.ac.uk

Arthur Richards
Bristol Robotics Laboratory
University of Bristol
Bristol, UK
Email: arthur.richards@bristol.ac.uk

***Abstract*—A major challenge of UAV research is facilitating the local development, deployment and testing of multi-UAV systems. Inspired by cloud computing, this work proposes Starling, a full-stack, compositional, containerised UAV infrastructure utilising ROS2, Gazebo, PX4, Docker and Kubernetes. By modelling individual UAVs as nodes in a compute cluster, our architecture is natively scalable, fault tolerant and allows for the flexible deployment of custom applications in both simulation and in reality. These technologies allow us to facilitate reproducible research while provide a lower barrier of entry for potential users, as well as the reuse of flight hardware for multiple projects. A multi-UAV path planning case study is presented to demonstrate the streamlined workflow of developing a controller from simulation to reality.**

I. Introduction

A key challenge in unmanned aerial vehicle (UAV) research is the ability to develop, deploy and test algorithms. This requires the development and maintenance of both an experimental platform, and realistic simulation environment. As applications become more complex with multiple UAVs, building bespoke solutions is clearly not sustainable - often becoming inflexible, incomprehensible and, importantly, difficult to reuse, with any level of confidence. This becomes especially hard when coupled with UAV hardware, where many operational details are omitted, making reproducibility all but impossible. Combined with poor documentation, it becomes an exceptionally large barrier for researchers who wish to perform real world experiments.

Notably, many of these problems are shared by software engineering in general. One solution is the use of Docker [4]. Docker is an example of a software containerisation system which can be thought of as lightweight virtual machines which only emulate the user's file system [1]. In particular it allows the encapsulation of the entire runtime environment of an application into an 'executable' image which can be downloaded and ran locally on any OS by any users in a modular fashion. While containers can be ran locally, traditionally in cloud-computing, they are often passed onto a container orchestrator, such as Kubernetes [12] in order to automatically deploy containers to run on physical server nodes within a cluster.

Inspired by this paradigm, we present *Starling* - a full-stack, compositional, containerised UAV infrastructure which models individual UAVs as server nodes within a cluster[1]. A development workflow is introduced where a user's ROS2 compatible UAV application is first developed against simulation containers, and then against a test harness which fully emulates the real system down to the networking and deployment dynamics. Finally, the exact same container image can be deployed to real UAVs. This allows us to exploit the benefits of cloud technology which include networking, scalability, failure recovery, hotswapping as well as make use of the numerous developed tools. This will allow us to facilitate reproducible research while reducing the barrier of entry for users, as well as allow the reuse of flight hardware for multiple projects. A multi-UAV path planning case study is presented to demonstrate the streamlined workflow of developing a controller from simulation to reality.

II. Background and Related Work

The UAVs are assumed to possess an autopilot and a companion computer. In this work we primarily use the PX4 firmware stack[15] for the autopilot, along with its software in the loop (SITL) simulation. The autopilot contains the high frequency control loops required to keep a UAV flying. For communication, the Robot Operating System (ROS) middleware layer is used, specifically ROS2 Foxy. Compared to ROS1, ROS2 has no master node, allowing for a minimal configuration decentralised approach. It also changes the underlying communications to use DDS standards allowing for the setting of best-effort quality-of-service policies. DDS also supports automated discovery which allows us to make use of Kubernetes' failure recovery and hotswapping [14]. The Mavros [6] node provides a ROS topic interface to the autopilot using the MAVLINK protocol standard. Gazebo [11] is a popular ROS compatible physics engine frequently used for simulation.

[1]Open Source on github: https://github.com/StarlingUAS/ProjectStarling

The development of a reusable full-stack testbed for UAV research is a challenging task. Stanford Starmac [7], MIT Raven [21] and UPenn Grasp[16] were the first to develop and successfully utilise Multi-UAV testbeds for research, but it is difficult to replicate these systems elsewhere, a problem our system attempts to solve. More recently, both Luis Sanchez-Lopez et al. [13] and Baca et al. [3] developed a full stack UAV system, defining a workflow from simulation to outdoor flight. Their system utilises ROS1 to facilitate a modular control architecture. Whilst the former uses a custom control stack, the latter asks users then develop controllers which can be manually run in either simulation on one or more UAVs running the PX4 [15] firmware, or through the Gazebo physics simulator [11] running the PX4 software in the loop (SITL) simulated autopilot. The latter system has been extensively tested and extended to numerous applications [18, 2]. They both aim to solve similar challenges to us. However, our system additionally benefits from the inherent scalability, failure recovery, UAV hotswapping and hardware agnostic ability of cloud distributed systems.

There also exist a number of works applying containerisation to UAV systems. For example, OpenUAV [19] encapsulates the entirety of a drone simulation (ROS1/ PX4/ Gazebo) into a single container to provide a web-based simulation testbed. We take a more compositional approach in comparison, in order to support real flight, but are inspired by their focus on accessibility. A few works then consider the system architecture in implementing UAVs as a service [8, 17]. Both papers propose software architectures for safely planning, organising and controlling UAVs from the cloud. Containerisation is used for the cloud applications, but are not used for deployment on the drones themselves. Both papers are also purely theoretical and have no physical implementation. This work has also been inspired by recent work in the field of IoT Edge Computation [10], as well as recent work applying a similar concept to ground robots for an industrial swarm testbed [9] and autonomous cars [20].

III. The Starling System

A. System Requirements

The system was designed to the following requirements:

- Support experimentation with Single and Multi-UAV applications, with a flexible and reusable set of vehicles
- Support both Offboard (centralised) and Onboard (decentralised) control of vehicles.
- Support the MAVLink Protocol (including via a ground-station connection) and ROS interfaces for vehicle control.
- Require minimal extraneous user configurations.
- Provide a defined workflow from simulation to reality.
- Provide one-click deployment for simulation and reality.
- Usable by researchers with basic programming experience.

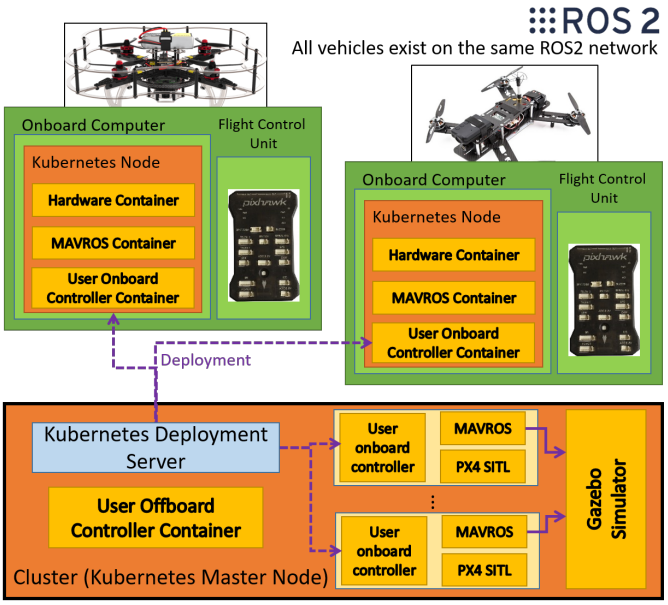

Fig. 1: Starling system overview with physical and simulated UAVs. The small boxes all represent Starling containers deployed by the cluster. The user only implements an onboard or offboard ROS2 controller which can then be deployed. The preconfigured Mavros container acts as the interface between user code and either a simulated or real PX4 drone autopilot. The UAVs used within our system are the Coex Clover (left) and QAV250 (right).

B. System Architecture

The core Starling module facilitates the communication between the user application and the UAV autopilot in simulation or reality. Internally, a Mavros node is used to translate between ROS, and MAVLINK for the autopilot. On one side, this module can be configured to support simulated or hardware autopilot firmwares. On the other, Mavros provides an interface through a consistent set of ROS topics, through which a user application can interact with the vehicle. This core module is packaged up in the `starling-mavros` Docker image[2]. The image is essentially a wrapper around the Mavros package along with a number of useful features: (1) *Automated Initalisation* of network connections to simulation or reality (2) *Automated ROS Namespacing* based on MAVLink system ID for ROS2 discovery (3) *ROS1 to ROS2 Bridge* to run ROS1 Mavros and its parameter bridge to specify a subset of topics to be forwarded to ROS2[3], and (4) *Safety Elements* such as an electronic stop (e-stop) node and topic.

Figure 1 shows an example multi-drone setup including simulation within the Starling system. Observe that the Mavros container is deployed to every vehicle (real and simulated). Kubernetes includes a number of different deployment types. In particular, the 'DaemonSet' type[4] automatically deploys containers to all nodes on the cluster matching a pre-specified label. In a similar manner, a user can then develop their own

[2]https://github.com/StarlingUAS/ProjectStarling/tree/master/system/mavros
[3]https://github.com/ros2/ros1_bridge needed until ROS2 mavros is released
[4]https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

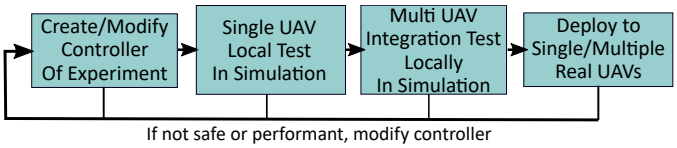

Fig. 2: The proposed workflow for Starling. It all revolves around developing not only the code, but the complete container. That same container can then be used to test in environments which get successively more realistic.

container using a Mavros compatible ROS controller, to also be deployed with its own DaemonSet against real vehicles or simulation. Through this feature, this work proposes a development workflow described by Figure 2. All containers are open source and stored in the Docker Hub online repository[5] to allow immediate running and access.

C. Simulation

Drawing from the *'Test Like You Fly'* paradigm developed by NASA[5], Starling attempts to minimise the changes from development to deployment. Therefore all user controllers must be locally tested in as realistic an environment as possible, not only in physics, but in system architecture, networking, user interaction, communications, etc. The use of cloud technologies allows us to meet these requirements.

The simulation is formed of two core elements. The first is the autopilot firmware's *software in the loop* (SITL) which emulates the autopilot. Importantly, to the Mavros container, nothing has changed whether running against SITL or a real autopilot. This general approach allows us to support both PX4 and Ardupilot SITL, with a container for running each. The second element is a container which contains the physics simulation and visualisation. This work utilises Gazebo [11], but any simulator supported by the SITLs could be used. Gazebo was chosen as it provides the GZWeb web-based frontend[6], providing an easily accessible GUI from both Linux- and Windows-based systems. The one or more SITL containers automatically connect to Gazebo for physics, and to spawn their vehicles for visualisation.

To avoid having the user manually run all of these containers, we use the *Docker-Compose* tool [7] to provide pre-made simulator configurations. Running a configuration will locally start up Mavros, SITL, Gazebo and other containers, which the user can then test their own controller against. This provides a quick development cycle and a reproducible environment, alleviating oft-quoted frustrations with robotics development.

If the user intends to fly in reality, the user is then encouraged to test within the Starling full test harness. This test harness not only provides the simulation, but also provides a virtual version of the real system architecture, complete with virtual drone nodes, identical networking models and deployment dynamics. This is achieved with Kubernetes in Docker (*KinD*)[8] which runs a simulated Kubernetes cluster within a set of Docker containers. *KinD* runs a container for each simulated drone which then internally runs the Starling container stack. This allows for the testing of deployment scripts and networking compatibility before real flight. To simplify user experience, the Mumuration command line interface (CLI)[9] has been developed for using Starling.

D. Real Flight

Once performance is satisfactory in simulation, the controller can be flown in reality. For our purposes, this will be indoors, but the same setup can be deployed anywhere, including outdoors. We operate multiple Coex Clover quadcopter UAVs [10] which run PX4 on a custom PixRacer-like autopilot with a 2GB Raspberry Pi 4B as a companion computer. A central server runs the lighter K3S Kubernetes variant[11] which is designed for low powered electronics. Each Pi runs a preconfigured Ubuntu OS and automatically connects to the server as a K3S node over WiFi upon startup.

Each vehicle contains a configuration file specifying its vehicle ID and other identifying variables. On deployment, this file is mounted into each of the containers, along with hardware inputs and serial links if needed. The Mavros container uses this file to ensure its configurations align to flying on a real vehicle. A Vicon and a hardware container are also automatically deployed via DaemonSet. The former communicates the real vehicle position to the drone using the arena wide Vicon motion capture cameras. The latter is used to interface with the sensors and LEDs on the vehicle. Through the use of Kubernetes, these containers are automatically deployed onto the vehicle upon startup, negating the need for any user setup. In an identical manner to testing within *KinD*, the user's deployment file is deployed to the cluster, with the individual nodes downloading user containers either from a local repository or from the internet.

IV. Multi-Drone Path Following Use Case

In order to demonstrate Starling in action, the development of a synchronous multi-drone path follower is discussed with results shown. Many of our multi-drone path planning projects require validation on real vehicles. However open loop path following is often not sufficient as real vehicles lag behind planned times, negating offline planned vehicle avoidance. A simple method is developed for synchronisation by communicating vehicle delays to a central monitor. This project is open source[12] for readers to try themselves.

A. Synchronous Path Following

For n UAVs, a path plan $= f\psi^1, .., \psi^n g$, consists of UAV trajectories $\psi^i = [(t^i_k, p^i_k)]_k$ specifying a set of coordinates p to visit at time t. During execution, vehicle i may be delayed by d^i_k upon arrival of its k^{th} task, t^i_k offset by the vehicle's

[5]https://hub.docker.com/orgs/uobflightlabstarling
[6]https://github.com/osrf/gzweb
[7]https://docs.docker.com/compose/
[8]https://kind.sigs.k8s.io/
[9]https://github.com/StarlingUAS/Murmuration
[10]https://coex.tech/clover
[11]https://k3s.io/
[12]https://github.com/StarlingUAS/synchronous_position_trajectory_controller

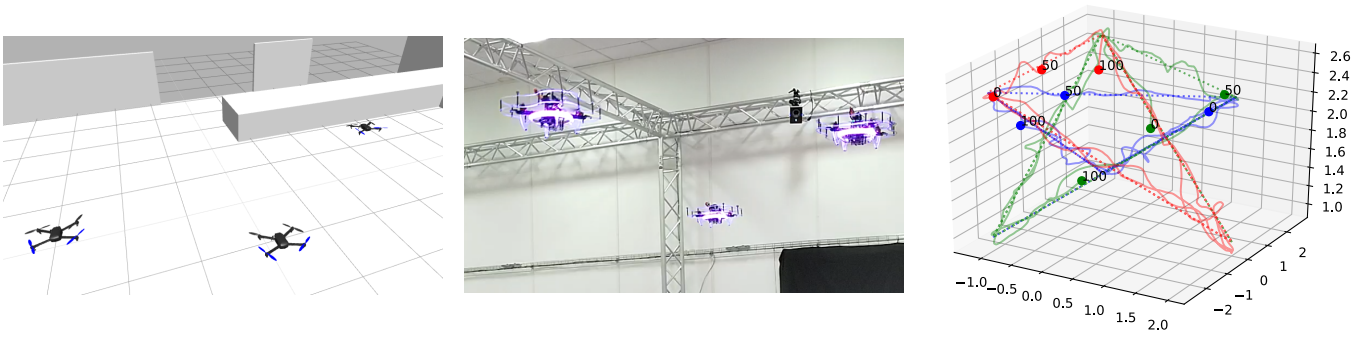

Fig. 3: Workflow for developing the multi-drone path planner. Left: Development in simulated *KinD* test harness. Middle: Testing on real Clover drones. Right: The final traversed paths in solid, with plan in dashed. A UAV per colour. Marked points are vehicle location at a given time. Synchronisation in effect as UAVs are not colliding.

previous cumulative delay. This delay is broadcast to all other vehicles.

Approximate synchronisation, defined as the transit times between non-delayed tasks, is approximately equal to the original planned transit times between tasks. When a vehicle reaches a task it will wait stationary for $w_k^i = \max_{i'}(d_k^{i'}) \quad d_k^i$ for the delayed drones to catch up. If the drone itself is late, it can be seen that it will incur no delay (i.e. has maximum delay over all drones) unless another UAV is even more delayed. It is assumed that no vehicle will be assigned more than one task during the transit of another. An analytical proof is out of scope of this report, but the method is validated in practice.

B. Development

The first step was to construct a valid Starling project. This involves constructing a Docker container using a Dockerfile which inherits from a `starling-controller-base` container. This base container includes the core dependencies (ROS2 and Mavros) required by a user controller.

From requirements, 3 differing packages are implemented in accordance with ROS2 practices. First, the trajectory follower node was implemented in C++ and included a state machine to provide takeoff, landing and safety monitoring in the case of user defined mission abort. The following itself was a linear interpolator between mission points, which then sent position setpoints to Mavros for the drone to follow. If the vehicle arrives late at its location, it publishes a `NotifyDelay` message to the monitor. Secondly, the centralised synchronisation monitor is a Python node which dynamically inspects the current list of topics for live UAVs. On receipt of a delay message, it will forward the delay via publishing a `NotifyPause` message to all other vehicles. On receipt, the follower updates its internal state with the other vehicles' delay.

Once implemented, the first step was to test the container against a local Docker-Compose configuration. This allowed for the identification of syntax and simple logical errors when running with a single drone. The ability to only restart the controller on each code update allows for efficient development. Following that, a Kubernetes deployment file was created to test deployment to the *KinD* test harness with multiple vehicles. This allowed for the debugging of the

Task	Delay d_k^i		Wait w_k^i		Arrival		Transit	
	1	2	1	2	1	2	1	2
1-2	5.9	0.1	0	0	26	20	20	19.9
2-3	0.9	0.3	0	5.6	47	40	20.1	19.6
3-4	1.6	0.1	0	0.8	68	66	19.4	20.3

TABLE I: Shows the relevant times for 2 of the 3 UAVs for the first four tasks. Planned task transitions are 20s. After offsetting ($20.3 = 66 \quad 40 \quad 5.6 \quad 0.1$), it can be seen that corrected transit times are all approximately 20s.

synchronisation mechanism through observation on a discrete time trajectories. Any changes only require a rebuild and redeploy which is of minimal hassle with the CLI. Once confident in its performance in simulation, the exact same deployment file was deployed within the flight arena on 3 Clover drones. Figure 3 shows the workflow in action, with Table I demonstrating approximate synchronisation. For a full demonstration, please refer to the companion video[13].

V. Conclusion

This work proposes *Starling*, a full-stack, compositional, containerised UAV infrastructure which addresses a number of the core challenges in UAV research. Namely, the ability to develop, deploy and test algorithms in a reliable and reproducible manner in both local simulation and on UAVs in reality. In addition it does so in a manner which both reduces the barrier to entry for potential researchers wishing to perform real world experiments, but also allows for the reuse of hardware platforms for multiple projects.

There is a lot of future scope for Starling, including better user interfaces and GUIs to reduce the barrier to entry further and supporting other robotic systems such as ground vehicles and automated cameras. It would also be desirable to port Starling outside to validate the use of containerisation more broadly in general UAV applications.

Acknowledgments

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) iCASE with Toshiba Research Europe Ltd, CASCADE Programme Grant EP/R009953/1 and FARSCOPE Centre of Doctoral Training at the Bristol Robotics Laboratory. The authors would like to thank all contributors to ROS, Mavros and PX4, all of which have enabled this work.

[13]Demonstration Video: https://youtu.be/73dadxUsbxA

References

[1] Docker: lightweight Linux containers for consistent development and deployment: Linux Journal: Vol 2014, No 239. URL https://dl.acm.org/doi/10.5555/2600239.2600241.

[2] Tomas Baca, Robert Penicka, Petr Stepan, Matej Petrlik, Vojtech Spurny, Daniel Hert, and Martin Saska. Autonomous Cooperative Wall Building by a Team of Unmanned Aerial Vehicles in the MBZIRC 2020 Competition. 12 2020. doi: 10.48550/arxiv.2012.05946. URL https://arxiv.org/abs/2012.05946v1.

[3] Tomas Baca, Matej Petrlik, Matous Vrba, Vojtech Spurny, Robert Penicka, Daniel Hert, and Martin Saska. The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles. *Journal of Intelligent & Robotic Systems 2021 102:1*, 102(1):1–28, 4 2021. ISSN 1573-0409. doi: 10.1007/S10846-021-01383-5. URL https://link.springer.com/article/10.1007/s10846-021-01383-5.

[4] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 1 2015. ISSN 01635980. doi: 10.1145/2723872.2723882. URL https://dl.acm.org/doi/abs/10.1145/2723872.2723882.

[5] A.W. Bucher. Test Like You Fly [spacecraft]. pages 7–3327, 11 2002. doi: 10.1109/AERO.2001.931408.

[6] Vladimir Ermakov. mavlink/mavros: MAVLink to ROS gateway with proxy for Ground Control Station, 2022. URL https://github.com/mavlink/mavros.

[7] Gabe Hoffmann, Dev Gorur Rajnarayan, Steven L. Waslander, David Dostal, Jung Soon Jang, and Claire J. Tomlin. The Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC). *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, 2, 2004. doi: 10.1109/DASC.2004.1390847.

[8] Chen Hong and Dianxi Shi. A Cloud-based Control System Architecture for Multi-UAV. *Proceedings of the 3rd International Conference on Robotics, Control and Automation - ICRCA '18*, 2018. doi: 10.1145/3265639. URL https://doi.org/10.1145/3265639.3265652.

[9] Simon Jones, Emma Milner, Mahesh Sooriyabandara, and Sabine Hauert. DOTS: An Open Testbed for Industrial Swarm Robotic Solutions. 3 2022. doi: 10.48550/arxiv.2203.13809. URL https://arxiv.org/abs/2203.13809v1.

[10] Paridhika Kayal. Kubernetes in Fog Computing: Feasibility Demonstration, Limitations and Improvement Scope : Invited Paper. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE, 6 2020. ISBN 978-1-7281-5503-6. doi: 10.1109/WF-IoT48130.2020.9221340. URL https://ieeexplore.ieee.org/document/9221340/.

[11] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2149–2154, 2004. doi: 10.1109/IROS.2004.1389727.

[12] Kubernetes. Kubernetes Documentation — Kubernetes, 2022. URL https://kubernetes.io/docs/home/.

[13] Jose Luis Sanchez-Lopez, Martin Molina, Hriday Bavle, Carlos Sampedro, Ramón A Suárez Fernández, Pascual Campoy, J L Sanchez-Lopez, H Bavle, · C Sampedro, R A Suárez Fernández, · P Campoy, P Campoy, M Molina, · M Molina, and · H Bavle. A Multi-Layered Component-Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework. *Journal of Intelligent & Robotic Systems 2017 88:2*, 88(2):683–709, 5 2017. ISSN 1573-0409. doi: 10.1007/S10846-017-0551-4. URL https://link.springer.com/article/10.1007/s10846-017-0551-4.

[14] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ROS2. *Proceedings of the 13th International Conference on Embedded Software, EMSOFT 2016*, 10 2016. doi: 10.1145/2968478.2968502. URL http://dx.doi.org/10.1145/2968478.2968502.

[15] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June (June):6235–6240, 6 2015. ISSN 10504729. doi: 10.1109/ICRA.2015.7140074.

[16] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The GRASP multiple micro-UAV testbed. *IEEE Robotics and Automation Magazine*, 17(3):56–65, 9 2010. ISSN 10709932. doi: 10.1109/MRA.2010.937855.

[17] Jerico Moeyersons, Martijn Gevaert, Karl-Erik Réculé, Bruno Volckaert, and Filip De Turck. *UAVs-as-a-Service: Cloud-based Remote Application Management for Drones; UAVs-as-a-Service: Cloud-based Remote Application Management for Drones*. 2021. ISBN 9783903176324.

[18] Martin Saska, Daniel Hert, Tomas Baca, Vit Kratky, and Tiago Nascimento. Formation control of unmanned micro aerial vehicles for straitened environments. *Autonomous Robots*, 44(6):991–1008, 7 2020. ISSN 15737527. doi: 10.1007/S10514-020-09913-0/FIGURES/20. URL https://link.springer.com/article/10.1007/s10514-020-09913-0.

[19] Matt Schmittle, Anna Lukina, Lukas Vacek, Jnaneshwar Das, Christopher P. Buskirk, Stephen Rees, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. OpenUAV: A UAV Testbed for the CPS and Robotics Community. *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, pages 130–139, 8 2018. doi: 10.1109/ICCPS.2018.00021.

[20] Jacopo Tani, Liam Paull, Maria T. Zuber, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. Duckietown: An Innovative Way to Teach Autonomy. *Advances in Intelligent Systems and Computing*, 560:104–121, 2017. ISSN 21945357. doi: 10.1007/978-3-319-55553-9{_}8. URL https://link.springer.com/chapter/10.1007/978-3-319-55553-9_8.

[21] Mario Valenti, Brett Bethke, Gaston Fiore, Jonathan P. How, and Eric Feron. Indoor multi-vehicle flight test bed for fault detection, isolation, and recovery. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006*, 2:1270–1287, 2006. doi: 10.2514/6.2006-6200. URL https://arc.aiaa.org/doi/abs/10.2514/6.2006-6200.