SUN: A Simulated UAV Network Testbed with Hardware-in-the-Loop SDR Support

Lars Baumgärtner, Maximilian Bauer, Bastian Bloessl

Technical University of Darmstadt, D-64289 Darmstadt, Germany
Email: {lars.baumgaertner, maximilian.bauer, bastian.bloessl}@tu-darmstadt.de

***Abstract*—Unmanned Aerial Vehicles (UAVs) are already part of everyday life as they are commonly used for disaster response, industrial applications, and smart farming. Resilient and flexible communication is key for the operation of UAVs themselves as well as many applications realized using them, e.g., gathering of data from Internet of Things (IoT) sensors without a direct uplink. Realizing new ideas, optimizations, and what-if analyses are usually lengthy processes to get from theoretical models to the actual real-world implementation. We propose an integrated testbed using emulation, where actual implementations can be evaluated in virtual environments that also simulate the network connectivity. Through the use of a software stack similar to actual UAVs that is built around Linux and the PX4 flight controller, we allow realistic prototyping, automated experiments and interactive missions. Additionally, we enable Hardware-in-the-Loop integration through a wideband Software Defined Radio (SDR)-based channel emulator to experiment with new radio links for more robust UAV control in challenging environments.**

***Index Terms*—Simulation, Network Emulation, Automated Evaluation, SDR, UAV.**

I. Introduction

Remote-operated and autonomous UAVs play an increasingly important role in fields such as smart agriculture, industrial applications, or disaster responses. Having robust and reliable means of communication for such applications is crucial. Thus, many researchers work on improved algorithms and new use-cases for UAVs in various settings. Bringing this research to actual prototypes or real-world products is hard, as conducting experiments and large-scale studies with actual hardware is very time-consuming, plus flying drones is heavily regulated in most countries. Therefore, people resort to evaluations of theoretical models or simulations, even though these approaches cannot always be carried over to actual implementations. Also, there are many simulators to model the flight behavior in a physical world with a proper flight controller as Software-in-the-Loop (SITL). Here, the weak point that is often overlooked is the communication link between the ground station and the UAV or from the UAV to other entities in the mission area. In the field of network simulations, many approaches exist that range from theoretical models to testbeds with emulated nodes and real network code driving the simulation. The motivation for using simulated testbeds can be manifold. For some the interaction and communication between UAVs and other nodes, e.g., IoT devices, people on the ground or Unmanned Ground Vehicles (UGVs) is the focus. To improve the actual UAV control, the simulation of the radio link to the flight controller is most important. Furthermore, some applications require a proper simulation of the physical world, including various obstacles or effects such as weather. Common simulators focus only on a subset of these features, limiting their use to specific areas.

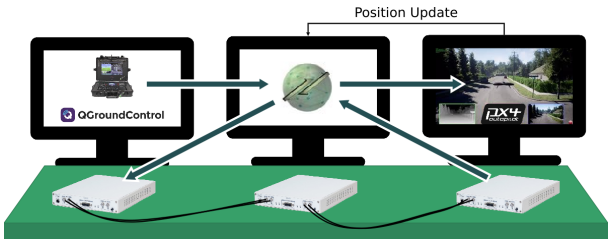

Fig. 1. Overview of the *SUN* testbed.

In this paper, we present a *Simulated UAV Network (SUN)* testbed that brings together physical world simulation, with emulated ground stations, actual flight controllers, and various ways to simulate the effects of network links in different scenarios as shown in Figure 1. Our solution is built around the *core network emulator (coreemu)* [1] to connect virtual nodes as well as physical ones. Furthermore, we integrated the PX4 flight controller with all its flight systems and the *Gazebo* world simulation. Besides the basic Wireless LAN (WLAN) model from *coreemu* and the optional support for advanced radio links using the Extendable Mobile Ad-hoc Network Emulator (EMANE) [2], we also integrated a wireless channel emulator, which uses SDRs as a Hardware-in-the-Loop solution to work on next-generation wireless links for UAV operations. Everything is put together in a portable and reproducible way using Docker containers.

In this paper, we present the following contributions:

- An accessible emulation environment [1] for complex UAV applications with integrated state-of-the-art UAV flight controller, physics simulation, and network emulation.
- A wideband, bidirectional, SDR-based channel emulator for hardware-in-the-loop evaluation of wireless links.
- A flexible experiment setup, suited for interactive as well as automated evaluation of complex scenarios with UAVs and other virtual/physical nodes.

[1] https://github.com/LOEWE-emergenCITY/sun

II. Related Work

Prior research has focused on various very specific aspects of UAV simulations, with also communication networks in mind. Often, discrete event simulators are combined with a flight controller or general flight-specific mobility patterns.

AirMobiSim [3] is a simulator for UAV-specific mobility models that can be combined with discrete event simulators like OMNeT++ for wireless communications. Using a high level of abstraction, it is possible to simulate and evaluate large scenarios. Yet, there is a disconnect between the simulated world and real-world experiments, as both the UAV mobility model and the simulation models for wireless communication are designed specifically for the corresponding simulators. Similarly, FlyNetSim [4] combines the *ArduPilot* flight controller with the ns-3 network simulator. Leading to a solution similar to ours but without support for a human-operated ground station, no separation of virtual nodes through containers, and no integration of Hardware-in-the-Loop (HITL) SDRs. The built-in SITL simulator of Ardupilot, which is based on the *FlightGear* simulator is not built to simulate complex scenarios involving ground robots, walking humans or taking sensor measurements like camera pictures or heat detection. Park et al. [5] also use ns-3 but focus more on time-synchronization issues between flight and network simulation. Also, their main use-cases are automated multi-UAV missions and not highly interactive scenarios or ones involving other simulated/physical nodes in the virtual world. VENUE [6] is another project focused on multi-UAV simulation and built using ns-3 plus Linux containers to separate nodes. Here, no flight controller or physical world simulation are integrated as the main target of the simulator are pure Flying Ad Hoc Networks (FANETs) and not the UAV control channels.

Besides UAV-centric network simulations, we also consider wireless testbeds relevant for our use-case, as we integrate an SDR-based channel emulator to evaluate the impact of the wireless link on the performance of the Unmanned Aerial System (UAS). Colosseum [7] is the world's largest wireless network emulator. The enormous engineering effort invested in the setup, highlights the importance to evaluate wireless technologies in an emulated tested. Especially, the ability to reproduce channel conditions and study different technologies over the same channels makes Colosseum an important tool for wireless research. We adopt the strategy in our testbed, albeit with a much smaller setup, focusing on a single UAV link. Yet, using the same hardware platform as Colosseum (the Ettus Research X310), we are able to expand the wireless emulation in future extensions of the testbed. AERPAW [8] is a closely related project that studies the use of UAVs in 5G networks. Like our testbed, it uses emulation to test, evaluate, and integrate the components of UASs. Being a communication-centric project, AERPAW considers various methods to model wireless communications, including capable commercial emulators or Colosseum. However, in contrast to AERPAW, our testbed architecture is more accessible, lightweight and platform-independent, requiring only affordable and widely available lab equipment. Additionally, we focus on a realistic simulation of complex UAV-tasks like in disaster response. We are capable of incorporating a plethora of different sensors and human interaction into our simulated scenarios.

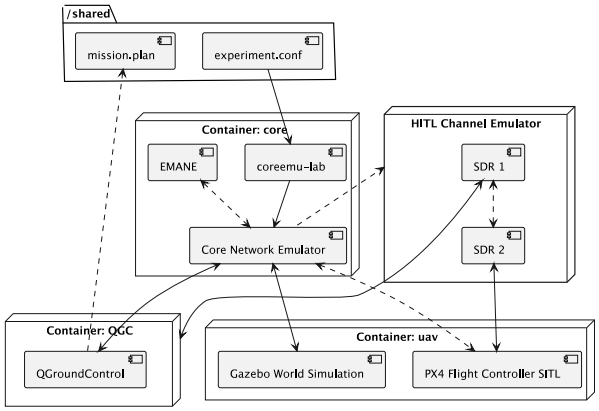

Fig. 2. Overview of SUN components.

III. Approach

A general overview of the architecture of *SUN* can be seen in Figure 2. To achieve maintainability and modularity, we separated the different components in their own Docker containers. In the following, we are going to give details on the various components used in *SUN*.

A. The Network

Our requirements for the network layer were real-time network emulation with the ability to adapt parameters such as jitter, delay, bandwidth, and communication range. Furthermore, the ability to have complex RF models for increased realism is needed for some scenarios. Besides the UAV and a ground station, we also want the ability to simulate other nodes, e.g., IoT sensors on the ground or people roaming the area. Thus, we need virtual nodes that can run standard Linux software and communicate with each other. As we not only need the network topology for complex scenarios but also want to automate node behavior, start various services, and collect metrics for evaluation, we used *coreemu-lab*[2] [9] as our foundation. It provides an easy and portable environment for reproducible network experiments, export of performance metrics, and automated plotting. *Coreemu* itself can have nodes as Linux namespaces or Docker containers in the simulation but can also bring in other machines through host network interfaces, e.g., *vlan* or *tap*, into the simulated environment. All nodes can be moved and rearranged interactively using *core-gui* and shells for interaction can be spawned. The whole simulation can also be controlled via a gRPC interface. Thus, the virtual nodes can be moved by an external process on another machine. This mechanism is used to update the UAV position based on data from the simulated drone environment.

[2]https://github.com/gh0st42/coreemu-lab

B. The UAV and the Surrounding World

The foundation of our simulated UAV is the *PX4*[3] flight controller, which can be compiled to be used as SITL. This open-source autopilot is also commonly found on many drones used in the real world. We run it in a physical world simulation provided by *Gazebo*[4]. *Gazebo* not only has flight dynamics but includes obstacles, sensors, weather and is also used to simulate disaster response[5]. It is, therefore, the perfect candidate to simulate UAVs in realistic and complex missions. To speed up the startup times of *SUN*, we pre-compile anything necessary for the provided simulation scenario, including the world map and any assets needed, into the Docker images. Furthermore, we developed tools to link the actual positions from *Gazebo* to the network emulator. While a position could also be extracted from the flight controller, the one from the physical world simulation is the ground truth, independent of (simulated) GPS signals or other sources of interference.

C. The Ground Station

As most UAVs use the *MAVLink*[6] protocol, we settled on the open-source flight control and mission planning software, *QGroundControl*[7], in our ground station docker container. We pre-configured it to connect to our virtual UAV either through the emulated direct link or routed via EMANE or SDR interfaces. Additional software needed on the ground station can easily be added to the container and is automatically routed through the virtual network to the UAV.

D. The Integration of Physical Communication

Core of the wireless-in-the-loop emulation is a Field-Programmable Gate Array (FPGA)-based, bidirectional, wideband channel emulator. It is implemented with an Ettus Research X310, a commercially available off-the-shelf SDR frontend that is also used in Colosseum, the world's largest channel emulator [7]. The X310 allows implementing SDR applications on a PC but also comes with ample FPGA resources that can be utilized through the RFNoC framework [10].

The channel emulation is implemented through a 41-tap complex FIR filter on the X310 that models the channel's impulse response. The whole data path is on the FPGA, i.e., the samples are not sent to the host PC, which would limit the usable bandwidth and introduce delay and jitter. While the RFNoC framework comes with an FIR filter, it only supports real-valued taps. To model the complex baseband channel impulse response, we, therefore, implemented a complex filter using two real filters as depicted in Figure 3. We duplicate the sample stream, passes it through two real-valued filters, and merge the streams, realizing the complex multiplication.

A similar filter is used in both directions to allow bidirectional communication (cf. Figure 1). While the data path is completely on the FPGA, the filter taps can be configured during runtime through existing RFNoC APIs. Therefore, we can adapt the channel during runtime depending on the position of the UAV in the Gazebo scenario. Adjusting the taps, we can realize relevant channel models, ranging from empirically determined, frequency-flat models that only define an attenuation to geometric models that define multi-path components, resulting in frequency-selective fast-fading channels.

In the current testbed, we use the channel emulator mainly for research on adaptable wireless communication, using SDR prototypes based on FutureSDR[8], a modern SDR runtime for heterogeneous architectures. While the relevance of SDRs is widely recognized by the community [11], it is also possible to connect off-the-shelf hardware. A current limitation of our HITL testbed is that we only consider a single link and do not mix signals from multiple communication partners, which could be a future extension. Overall, we believe that the emulator provides a practical solution that is easy to reproduce, given the cost and availability of the X310.

E. Bringing it all together

All three main Docker containers are brought together using *docker compose*. We separate them from each other using Docker networks for *ground*, *air*, and the outside communication to the host. Furthermore, any direct communication between the ground station and the UAV is blocked by firewall rules. Thus, all traffic has to go through Virtual LANs (VLANs), which are then bridged or routed, depending on the scenario, into *coreemu*. Since the UAV position in *coreemu* needs to be synchronized with the position in the simulated physical world in *Gazebo*, we developed a bridging middleware that subscribes to *Gazebo* position changes and updates the location of the UAV in the network emulation via gRPC. The channel emulator must interface with physical hardware, the SDRs, and, thus, runs directly on the host machine from which it is patched through to the Docker instances. The network and ground station containers expose VNC servers to interact with GUI software such as *core-gui* and *QGroundControl*. Through the network GUI scenario topologies can also be created and loaded as well as running simulations with other virtual nodes can be interacted with, e.g., moving nodes or running various commands on them.

F. Additional Helpers

The network emulation container also has the *babel* [12] mesh routing software preinstalled, so virtual nodes can be used in experiments to bridge traffic for the UAV or to connect each other. For experiments involving "data-mules", opportunistic networking and Disruption-Tolerant Networking (DTN), we also included *dtn7-rs*[9][13], which is an implementation of RFC 9171 [14]. As nodes in *coreemu* usually do not know their own positions, we have developed a tool to periodically update their positions on disk, so they can easily be queried from within the virtual nodes. Furthermore,

[3]https://px4.io
[4]https://gazebosim.org
[5]https://rescuesim.robocup.org
[6]https://mavlink.io/en/
[7]http://qgroundcontrol.com
[8]https://www.futuresdr.org/
[9]https://github.com/dtn7/dtn7-rs

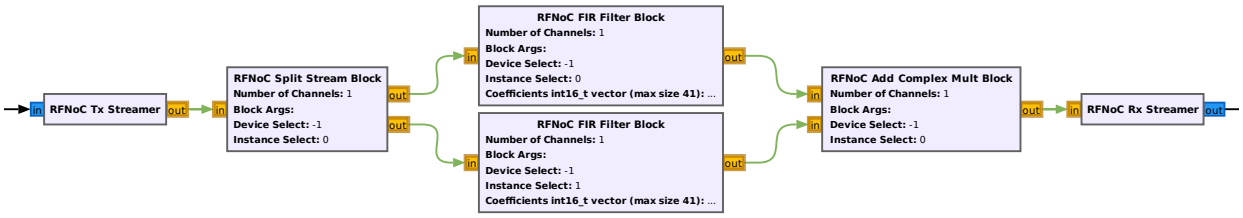

Fig. 3. RFNoC implementation of the complex FIR filter that models the channel's impulse response.

we provide terminal-based live dashboards using *wtfutil*[10] to show statistics about node positions, peer numbers, etc. These dashboards can easily be extended with shell scripts to fit different use cases and scenarios.

IV. Evaluation

To evaluate and demonstrate *SUN*, we conducted several tests regarding our wireless channel emulator and implemented a test mission as a case-study.

A. SDR Integration for UAV Control

In the following, we evaluate various properties and components of our channel emulator that can be used for the UAV control link.

1) Verification: To validate the filter design and verify its implementations, we integrate our custom RFNoC blocks into GNU Radio [15], an open-source SDR framework with a graphical editor to configure and run signal processing flowgraphs. This integration allows us to directly compare the GNU Radio CPU implementation of a complex FIR filter with our FPGA implementation. To this end, we use a random source to generate complex noise (normally distributed with an average of zero and a standard deviation of 0.1), filter them with similar FPGA and CPU implementations of complex FIR filters, and compare the results.

For 5M filtered, complex numbers the average difference between the implementations is 0.6 %, which stems from the fixed-point integer conversion (32-bit floating point number to 16-bit signed integer), required for the FPGA. Note that this conversion is only necessary when using the filter with GNU Radio. When using the emulator, the whole data path on the FPGA will use 16-bit signed integers.

2) Channel Resolution vs. Delay Spread: The radio frontends of the X310 operate at a fixed sample rate of 200 Msps, providing a theoretical maximum bandwidth of 200 MHz. The baseband rate can, however, be reduced using Digital Down Conversion (DDC)/Digital Up Conversion (DUC) blocks to down/up sample the baseband signal. With this, we can adjust a trade-off: Using a higher sample rate results in a larger usable bandwidth and higher time resolution of the channel impulse response (one tap corresponds to 5 ns @ 200 MHz) but limits the delay spread of the 41 available taps (205 ns @ 200 MHz), corresponding to a maximum path difference of ≈61 m. Using a lower sample rate limits the usable bandwidth and reduces the time resolution of the channel impulse response (20 ns @ 50 MHz) but increases the maximum delay spread that can be modeled by the filter (820 ns @ 50 MHz), corresponding to a maximum path difference of ≈244 m. These delay spreads are well within the range of UAV channel models [16], where the channel models with the largest time dispersion have an Root Mean Square (RMS) delay spread of ≈60 ns.

[10]https://www.wtfutil.com

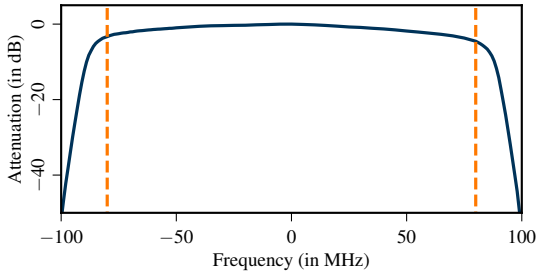

Fig. 4. Usable bandwidth of the channel emulator. The vertical lines indicate the 80 MHz offsets, corresponding to the nominal bandwidth of the analog frontend.

3) Usable Frequencies and Bandwidth: The X310 can be used with different RF frontends (so-called *daughterboards*) that do analog up and down conversion. In our testbed, we utilize the UBX 160, which offers send and receive bandwidths of 160 MHz for RF frequencies between 10 MHz and 6 GHz. To be precise, the receive bandwidth is 84 MHz for frequencies below 500 MHz, which are, however, not of primary interest for this work. The following experiments, therefore, use higher frequencies, which provide the full 160 MHz.

The filter acts as a 160 MHz low-pass filter for the complex baseband signal that is applied twice, i.e., during reception and transmission. This cascaded application results in steeper roll-off. To quantify the impact of the analog filter and to measure the usable bandwidth of the emulator, we set the emulator to a fixed frequency (2.45 GHz) and a sample rate of 200 MHz. We, then, tune sender and receiver to different offsets and send a sine tone. Since the sine is in the center of the baseband and has zero bandwidth, it is not affected by the filters of sender and receiver. Measuring the power of the sine at different offsets from the fixed frequency of the emulator, we measure the combined effect of send and receive filters of the channel emulator.

The result of the experiment is shown in Figure 4. The vertical lines indicate the 80 MHz offsets corresponding to the nominal bandwidth of the daughterboard. With 160 MHz,

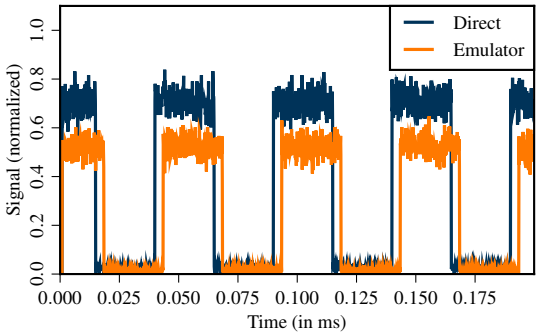

Fig. 5. Delay induced by the channel emulator.

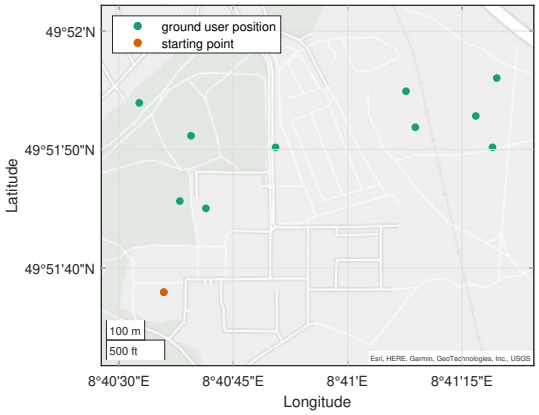

Fig. 6. Overview of the evaluation area containing ground users (green) and the starting location (red).

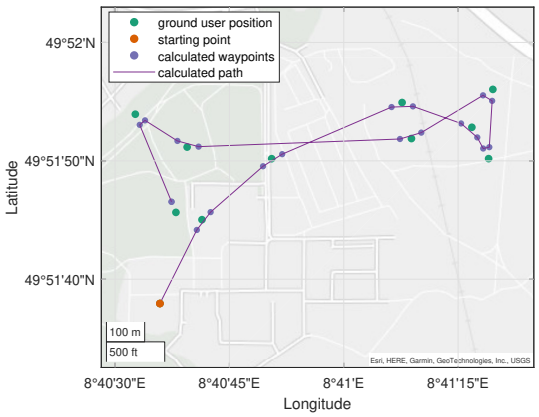

Fig. 7. Calculated waypoints and path using the algorithm from [17].

the emulator provides a practical solution for state-of-the-art technologies. It can, for example, filter cellular bands with multiple 4G/5G cells or the whole 2.4 GHz band with a mix of WLAN, ZigBee, and Bluetooth.

4) Time-Domain Impact: While implementing the complete data path on the FPGA minimizes delay and jitter, there is an unavoidable filter and processing delay compared to over-the-air signal propagation. In this experiment, we measure the impact of the channel emulator on the timing of the signal. To this end, we use the Multiple Input Multiple Output (MIMO) capabilities of the X310, allowing us to send time synchronized signals. We connect one output of a sending SDR directly to a receiving SDR and another through the channel emulator. Since the emulator is not synchronized with the sender, it will introduce phase and frequency offsets. We, therefore, send a square wave, allowing the receiver to calculate the magnitude and log the position of the flanks at two inputs. This measurement is robust against the impairments of the emulator.

Exemplary results on the time-domain signal are shown in Figure 5. The receiving SDR operated at a sample rate of 10 MHz and logged the two signals to disk. Over 70k flanks were determined and matched in post-processing to calculate the delay between the signals. The delay was stable at 3.5 µs with no measurable jitter. To put this into context, this corresponds to over-the-air signal propagation delay to a target with a distance of ≈1 km.

While this is low, it can have an impact on technologies with tight MAC layer timings. For WLAN, this delay is in the order of the short inter-frame space of 16 µs for IEEE 802.11a and 10 µs for IEEE 802.11g. Depending on the timing margins of the hardware, standard-compliant unicast communication might, therefore, not be possible over the emulator. For broadcast communication, this is not an issue.

B. Case Study: Data-ferrying for People on the Ground

a) Mission Outline: Here, we build upon an analytic paper for optimized route planing with uncertain user locations [17]. UAVs are commonly suggested for data-ferrying tasks in store-carry-forward networks. For this mission, we assume people are spread in a disaster area and can only communicate opportunistically in their near vicinity, e.g., through local WLAN mesh. The UAV needs a flight plan to collect and redistribute all messages from the users using DTN. To test the flexibility in usage of different vehicles, we will use a Multirotor (MR) and a Vertical Takeoff and Landing (VTOL) hybrid UAV to fulfill the mission.

b) Implementation: We configured *SUN* with parts of our university campus as a starting geo-location, as we also have flight permission for real world tests there. Our simulation covers an area of 1000x1000 m^2. In Figure 6, one can see the overall scenario with the node distribution. The ground station is placed virtually near the building from which the UAV is deployed. Furthermore, there are ten virtual nodes, only existing in the network emulation container that run *dtn7-rs*, an open-source implementation of the *Bundle Protocol* [14]. Additionally, a virtual network node is attached to the UAV node, also running *dtn7-rs* and representing physical ad-hoc communication infrastructure and access points, which we usually attach to our drones. The *coreemu-lab* scenario config is set to pre-generate messages on all ground nodes. User locations and relevant information serve as input to the algorithm from Yilmaz et al. [17]. The waypoints are then calculated and connected by straight lines to generate the flight path. The result can be seen in Figure 7. Finally, the flight

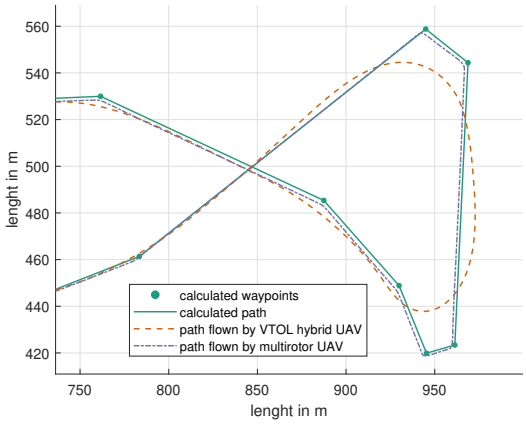

Fig. 8. Comparison of a segment of the calculated and flown flight paths.

path is converted into a mission plan and uploaded through QGroundControl. First, a Multirotor UAV will fly this mission and then a VTOL hybrid UAV will fly it.

c) Experimental Results: Part of the flown paths can be seen in Figure 8. The actual flown paths differ from the intended path. This happens due to the path tracking algorithm of the flight controller and the uncertainty of the position measurement of the UAV. The MR is able to hover and fly at low speeds, while the VTOL hybrid UAV needs a minimum speed to stay airborne while in cruise. This also limits its turning radius. The maximum deviation of the MR is therefore less ($< 2.5m$) than that of the VTOL hybrid UAV ($< 20m$). The integration of real flight dynamics and control into the simulator is not only important for testing flight guidance and planning algorithms (temporal and spatial accuracy is needed if for e.g. collision avoidance or inspection tasks) but also for realistic evaluation of communication performance. As expected, due to the flight path deviation still being in WLAN range and the opportunistic nature of the communication system deployed in this scenario, we achieved a 100% delivery rate for the pre-generated messages.

V. Conclusion

In this paper, we presented *SUN*, a novel environment for simulating network effects in different UAV scenarios. We provide a convenient way to design complex scenarios with a mixture of virtual and physical nodes in a virtual world where we have various options to model network connectivity ranging from basic WLAN emulation to more complex models and even the integration of HITL SDRs for realistic UAV control channel emulation. Through our channel emulator, we can explore the effects of various low level wireless links, new communication mechanisms and tweaks on existing ones on the UAV remote-operation link. By integrating an industry standard flight controller (PX4) and a common physical world simulator (*Gazebo*), we achieve a high degree of realism and are able to test flight guidance and communication performance concurrently. The simulator speeds up the process of going from research prototype to real world deployment.

Acknowledgment

This work has been co-funded by the LOEWE initiative (Hessen, Germany) within the *emergenCITY* center, as well as the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 MAKI. The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the project “Open6GHub” (grant number: 16KISK014).

References

[1] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, “Core: A real-time network emulator,” in *IEEE Military Communications Conference (MILCOM)*, 2008.

[2] J. Ahrenholz, T. Goff, and B. Adamson, “Integration of the core and emane network emulators,” in *IEEE Military Communications Conference (MILCOM)*, 2011.

[3] T. Hardes, D. Logan, T. H. Pritom, and C. Sommer, “Towards an Open-Source Fully Modular Multi Unmanned Aerial Vehicle Simulation Framework,” in *International Workshop on Wireless Sensor, Robot and UAV Networks*. IEEE, 2022.

[4] S. Baidya, Z. Shaikh, and M. Levorato, “FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot,” in *International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*. ACM, 2018.

[5] S. Park, W. G. La, W. Lee, and H. Kim, “Devising a Distributed Co-Simulator for a Multi-UAV Network,” *Sensors*, vol. 20, no. 21, 2020, Multidisciplinary Digital Publishing Institute.

[6] V. Sanchez-Aguero, F. Valera, B. Nogales, L. F. Gonzalez, and I. Vidal, “VENUE: Virtualized Environment for Multi-UAV Network Emulation,” *IEEE Access*, vol. 7, 2019.

[7] L. Bonati, P. Johari, M. Polese, S. D’Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia, A. Gosain, K. R. Chowdhury, S. Basagni, and T. Melodia, “Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation,” in *International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2021.

[8] A. Panicker, O. Ozdemir, M. L. Sichitiu, I. Guvenc, R. Dutta, V. Marojevic, and B. Floyd, “AERPAW Emulation Overview and Preliminary Performance Evaluation,” *Computer Networks*, vol. 194, 2021.

[9] L. Baumgärtner, T. Meuser, and B. Bloessl, “coreemu-lab: An automated network emulation and evaluation environment,” in *IEEE Global Humanitarian Technology Conference (GHTC)*, 2021.

[10] M. Braun, J. Pendlum, and M. Ettus, “RFNoC: RF Network-on-Chip,” in *GNU Radio Conference (GRCon)*. GNU Radio Project, 2016.

[11] K. Powell, A. S. Abdalla, D. Brennan, V. Marojevic, R. M. Barts, A. Panicker, O. Ozdemir, and I. Guvenc, “Software Radios for Unmanned Aerial Systems,” in *International Workshop on Open Software Defined Wireless Networks (OpenWireless)*. ACM, 2020.

[12] J. Chroboczek and D. Schinazi, “The babel routing protocol,” Internet Requests for Comments, RFC Editor, RFC 8966, January 2021.

[13] L. Baumgärtner, J. Höchst, and T. Meuser, “B-dtn7: Browser-based disruption-tolerant networking via bundle protocol 7,” in *International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, 2019.

[14] S. Burleigh, K. Fall, E. Birrane, and III, “Bundle protocol version 7,” Internet Requests for Comments, RFC Editor, RFC 9171, January 2022.

[15] T. W. Rondeau, “On the GNU Radio Ecosystem,” in *Opportunistic Spectrum Sharing and White Space Access: The Practical Reality*, O. Holland, H. Bogucka, and A. Medeisis, Eds. Wiley, 2015.

[16] A. A. Khuwaja, Y. Chen, N. Zhao, M.-S. Alouini, and P. Dobbins, “A Survey of Channel Modeling for UAV Communications,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, 2018.

[17] B. Yilmaz, L. Xiang, and A. Klein, “UAV-Assisted Delay-Sensitive Communications with Uncertain User Locations: A Cost Minimization Approach,” in *International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2022.