CORNET: A Co-Simulation Middleware for Robot Networks

Srikrishna Acharya
*RBCCPS**
Indian Institute of Science
Bengaluru, India
bacharya@iisc.ac.in

Amrutur Bharadwaj
*RBCCPS**
Indian Institute of Science
Bengaluru, India
amrutur@iisc.ac.in

Yogesh Simmhan
Dept. of Computational Data Sciences
Indian Institute of Science
Bengaluru, India
simmhan@iisc.ac.in

Aditya Gopalan
Dept. of Electrical Communication Engineering(ECE)
Indian Institute of Science
Bengaluru, India
aditya@iisc.ac.in

Parimal Parag
Dept. of ECE
Indian Institute of Science
Bengaluru, India
parimal@iisc.ac.in

Himanshu Tyagi
Dept. of ECE
Indian Institute of Science
Bengaluru, India
htyagi@iisc.ac.in

***Abstract*—This paper describes CORNET, a co-simulation middleware for applications involving multi-robot systems like a network of Unmanned Aerial Vehicle (UAV) systems. Design of such systems requires knowledge of the flight dynamics of UAVs and the communication links connecting UAVs with each other or with the ground control station. Besides, UAV networks are dynamic and distinctive from other ad-hoc networks and require protocols that can adapt to high-mobility, dynamic topology and changing link quality in power constrained resource platforms. Therefore, it is necessary to co-design the UAV path planning algorithms and the communication protocols. The proposed co-simulation framework integrates existing tools to simulate flight dynamics and network related aspects. Gazebo with robot operating system (ROS) is used as a physical system UAV simulator and NS-3 used as a network simulator, to jointly capture the cyber-physical system (CPS) aspects of the multi-UAV systems. A particular aspect we address is on synchronizing time and position across the two simulation environments, and we provide APIs to allow easy migration of the algorithms to real platforms.**

***Index Terms*—COSIM, UAV, NS-3, Gazebo, 5G, WiFi, Cyber Physical Systems.**

I. Introduction

Autonomous robots such as self-driving cars and Unmanned Aerial Vehicle (UAVs)/drones are set to be game-changers in many fields such as infrastructure maintenance, transportation, public safety, disaster response, agriculture, mining, health care and exploration. With advances in cyber physical systems, cloud-fog computing, and artificial intelligence, autopilot systems for autonomous vehicle are expected to play a key role in the future of urban transportation systems. Especially with advent of 5G and vehicle-to-infrastructure (V2X) frameworks, use-cases like infrastructure-assisted remote-controlled autopilot systems [30] [21] and situation awareness [26] based advance driver assistance systems are enabled. Implementing such systems requires new algorithms and proper tuning of the parameters without which systems are prone to hazards, making them potentially unsafe. Therefore, it is necessary simulate the system with appropriate tools to ascertain a certain level of confidence.

There is a dearth of open-source tools that can enable co-simulation of both control systems dynamics of UAVs and Unmanned Ground Vehicle (UGVs) and a detailed network-side simulation. In this work we will present an integrated framework for the realistic simulation of connected autonomous vehicles, with a specific example of a multi-UAV system. The solution proposed should satisfy following criteria,

- should be lightweight and be able to run on multiple interconnected computers,
- there should be a common notion of simulated time and position across the two simulation environments and
- must offer APIs allowing developers to write their strategy and control algorithms in a way that enables porting to a real platform with few or no modifications.

Our proposed framework for networked control systems mainly focuses on bridging the two domains with existing open-source tools to capture the closed loop simulation in near real time. Design of the cyber-physical systems aspects of multi-UAVs is a complex task and, therefore, there is no single tool that can capture all the requirements for the design of such systems. Hence, the need for cooperation between different tools in a simplistic and synchronized manner.

The main contribution of this work is an integration strategy and implementation of two different open-source simulation engines, Gazebo-ROS and NS-3 with common mobility model and time synchronization. ROS uses the PCś System clock as wall time, while NS-3 models the operation as a discrete sequence of events in time. Combining periodic sampling and event based sampling is a technique we leverage.

II. Related Work

In this section, we provide a summary of existing UAV and network simulators. There have also been recent efforts in the

Robert Bosch Center for Cyber Physical Systems

direction of joint UAV-network simulation frameworks.

The fundamental scope of *UAV simulators* is to model physical dynamics and functional aspects of UAV. Early efforts described in flight trainers [5] and Microsoft flight simulator [20] mainly focus on humans operating the flight simulator. Most of the existing simulation tools focus on first person view (FPV) and aerial or line of sight (LOS) view, essentially targeting realistic graphics and technically sound aerodynamic designs. Our focus is on autopilot features and autonomous flight simulators. Ardupilot [2] and PX4 [7] are the most widely used open-source frameworks. Several UAV simulators are built on top of these two frameworks to extend its capabilities. Another interesting feature is support for software in the loop (SITL) simulations, where the target platform is a software simulation. Both PX4 and Ardupilot supports this feature. Gazebo robotic simulator [22] has been used extensively in several UAV-related works but it also extends these to land-rovers, planes and small robots using ROS [8]. AirSim [1] is another open-source tool used to capture the dynamics of robots, which is built on the Unreal engine [28]. Its prime focus is to enable experiments with deep learning, computer vision and reinforcement leaning algorithms for autonomous vehicles.

Similarly, a variety of network simulators offer a simulation/emulation framework for wired/wireless networks along with traditional hardware such as NICs, switches, routers, Access Points (APs) etc. Our primary interest is to support WiFi and 4G/5G cellular communications for enabling UAV networks, along with a wide range of protocols and options at different layers of the network.

Qualnet [14] and OPNET [13] are commercial simulators enabling detailed simulation of wireless networks. Among open-source tools OMNET++ [13] and NS-3 [18] are the most used by research community. On another note, mininet [15] is network emulation tool, which provide realistic virtual network, running real kernel, switch and application code, on a single machine.

Recently, joint UAV-network simulators have been proposed like CUSCUS [31], AVENS [25] and FlyNetSim [11]. CUSCUS has been developed based on FL-AIR and NS-3. In NS-3 communication links are established using tap bridges via containers, which acts as network devices for UAVs in FL-AIR. The use of tap bridges severely limits the extensibility of the framework for technologies such as D2D, LTE and 5G. Another framework AVENS is developed based on OMNET++ and X-Plane. The main limitation of AVENS is that the UAV and network simulator communicate through an XML file, which only updates the number and position of the UAV. This framework does not provide a path for control, telemetry to and from UAVs. FlyNetSim is developed using NS-3 and Ardupilot software-in-the-loop (SITL). To achieve time synchronization they have used real-time schedulers in NS-3, which aligns the processing of scheduled events with actual time. In case of dense simulations the scheduler may actually be late with respect to clock, resulting in either lost synchronization (best-effort) or late events being discarded (hard-limit). They derive the UAV position from deep packet inspection of the telemetry data received.

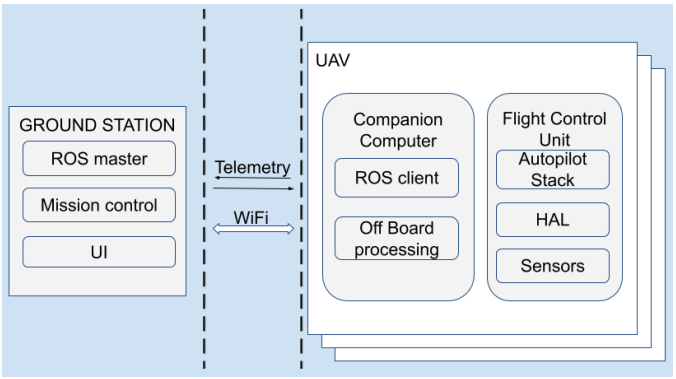

Fig. 1. Generic multi-UAV system framework

Recent surveys on current drone simulation frameworks [10] [17] highlight the necessity to co-design the UAV path planning algorithms and the communication protocols. Along with application-specific requirements, some of the pressing aspects highlighted are,

- The UAV simulator should be augmented with communication aspects of drones, where each drone is treated as a node, enabling flying ad-hoc networks (FANETS) [12],
- Additional modules to enable research on cyber-security aspects of UAVNETs [19],
- Support for various scenarios with different mobility models and
- Ability to support new UAV designs like bio-inspired drones, MAGMA(using elevators, rudders and ailerons) [3], Blimp drones [4] etc.

Our choice of Gazebo with ROS and NS-3 in the Linux environment is primarily driven by the large community support. Another key aspect is that Gazebo's software architecture is modular and can be extended using plug-ins. On the other hand, AirSim which uses the Unreal engine, is not well supported on Linux and requires demanding computational capabilities. Additionally, NS-3 can meticulously interface with external systems, applications and libraries.

III. Multi-UAV system framework

In this section, we provide an overview of a generic multi-UAV system. As shown in the Figure 1 the basic components of multi-UAV systems are the UAV, the Communication interface and the Base station.

The UAV consists of an Flight Control Unit(FCU) with an autopilot stack. A hardware abstraction layer provides drivers for motors, inertial and geographical sensors, and optionally, an on-board companion computer. For example, the PixHawk control board equipped with PX4 or Ardupilot flight stack and single board computer (SBC) platforms like Raspberry Pi, Odroid, TX2, etc., are commonly used platforms for UAVs. The on-board computer augments the FCU with the high-level logic of the mission and also provides it with a

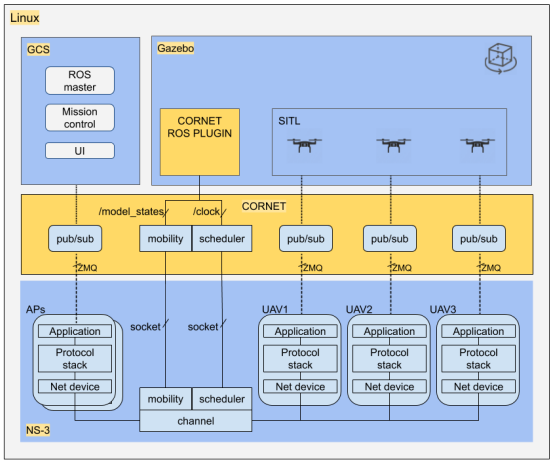

Fig. 2. High level framework for closed loop control simulation

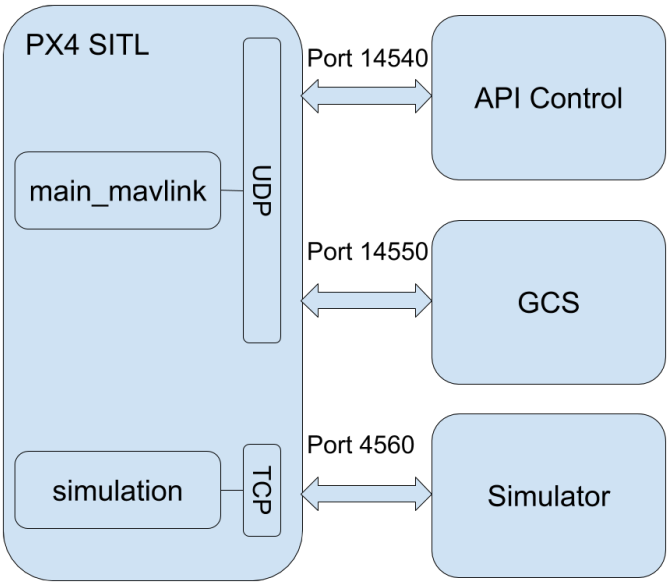

Fig. 3. PX4 SITL environment

wireless communication interface to perform data exchanges with other UAVs or with the base station. The FCUs are also connected using VHF modems to interact with ground control station(GCS) for telemetry and flight control.

The Base Station is a Ground Control Station (GCS), consisting of one or more computers, and serves as virtual cockpit for the UAV systems. It provides a user interface to manage and control the whole mission.

IV. Co-Simulation Architecture

In this section we discuss the details of the proposed co-simulation framework, whose architecture is depicted in Figure 2. The key components of the framework are: *Gazebo* to simulate UAV components with 3D visualization; *NS-3* to provide an end-to-end network infrastructure; and our *CORNET* (Co-Simulation of Robotic Networks) middleware for creating an inter-simulator data-path, with time and position synchronization at both ends. In the following subsections, we describe each component in detail.

A. Gazebo

Gazebo is an open-source robotic simulator, which provides physics simulation, rendering, user interface, communication, and sensors models. It has two executable programs:

- the *gzserver* runs physics update-loop with sensor data generation,
- the *gzclient* provides a graphical interface to visualize and interact with the simulation environment.

The client and server communicate using the Gazebo communication library, and use protobuf [27] for message serialization. A world-description file contains all the elements in a simulation, including quad-rotor frame, lights, sensors and static objects. The simulation description file (SDF) defines the model of the frame and the inertial properties of the UAV. In our examples, we have used definition file called *iris*, which is a quad-rotor frame available in Gazebo's default model library.

An autopilot stack provides proper frame pose and commands to motors. The PX4 SITL framework is used to provide the software stack enabling interactions with simulated UAV frame.

As shown in the Figure 3, PX4 SITL uses a specific module to listen on TCP port 4560, to communicate with Gazebo to receive sensor data and to send the motor and actuator values to the simulated world. It also uses the main_mavlink module to connect to GCS (on port 14550) and external developer APIs like Dronecode SDK or ROS (on port 14540). This port is also used to connect to the on-board computer.

B. NS-3 Network Simulator

The Network infrastructure is simulated using NS-3 libraries. Borrowing the idea from FlyNetSim [11], we have integrated NS-3 with the zero message queue (ZMQ) publish-subscribe protocol for configuring one-to-one correspondence between UAVs in Gazebo and nodes in NS-3. Moreover it enables the framework to migrate to other interfaces such as LTE and 5G in future. In NS-3, each node has three layers. The application layer provides a socket API to interact with applications, the protocol stack provides a TCP/IP stack, and the net device provides simulated NICs. It also provides helper modules to manage mobility and simulation scheduler. Channel abstraction allows the use of any wireless protocol supported by NS-3. In our framework, mobility and time synchronization in NS-3 and Gazebo is handled using ROS plugins.

C. CORNET Middleware

The primary function of our proposed middleware is to interconnect Gazebo and NS-3 in a synchronized manner. It constitutes of two components:

- ZMQ - to provide end-to-end data path, and

- CORNET-ROS Plugin - for mobility and time synchronization.

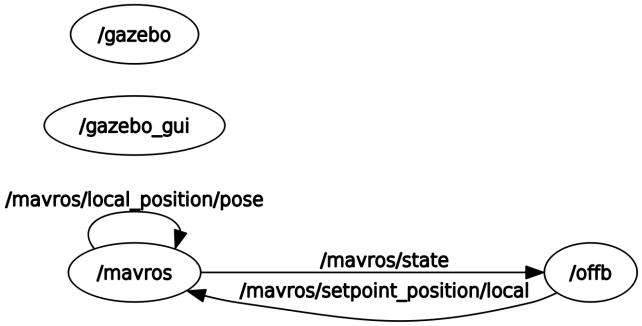

Fig. 4. Rqt_graph of MAVROS control message

The ZMQ component is responsible for creating an end-to-end data path between UAVs, or from UAV to ground station. For each UAV/GCS, we use a pair of ZMQ publisher/subscriber for communication at both ends as illustrated in Figure 2. We have used the API control port in the PX4 SITL environment integrated with the MAVROS [6] library for sending commands and to get the telemetry data back to GCS through ZMQ data-path. MAVROS is a mavlink extendable communication node for ROS with ground station control (GCS). It gets the current state of the model and updates the setpoint_position ROS topic with the new state. The rqt_graph [9] of the MAVROS illustrates sequences of the control commands as shown in Figure 4.

Using this state information obtained from the UAV model, the position of the UAV nodes in NS-3 are updated periodically. The overall synchronization of time and position updates are maintained in our middleware. In the next section, we elaborate our proposed time synchronization technique.

V. Time Synchronization

For a cyber-physical systems simulation framework, it is critical to have a common notion of time across the framework. Both of these simulators are built for different purposes. Gazebo simulates physical systems, that are modeled as continuous-time dynamic systems involving differential equations. So periodic sampling is the appropriate choice of time management. In contrast, NS-3 employs discrete-event model, where time stops only at discrete events such as packet transmission and reception. So an event based sampling is the appropriate choice of time management. As we see, the natural choice of time management are different for these two simulation environments. Thus, to realise a cyber-physical systems co-simulation, it is crucial to design a systematic mechanism to connect the two simulators. Combination of periodic and event based sampling is not a trivial problem.

A. Approach

Time-stepped method [29] [23] is one of the simplest and most popular approach for time synchronization. In this method, synchronization is achieved by introducing a common sampling period to be used by both simulators, which is an integral multiple of the sampling period of the physical systems simulator i.e., Gazebo. Both simulators achieve time synchronization by exchanging variables and status at the common sampling time. Here the major problem is that, if network simulator runs faster than the physics simulator, the network events have to be buffered in a cache and wait to be processed until the next sampling time. Therefore, it will introduce erroneous exchange of variables that could be accumulated over the time and hamper the system fidelity. One simple solution to keep pace with slow ticking simulator is for the faster ticking simulator to freeze regularly, leading to extra simulation overheads.

The method we adopted is called the global event-driven method [24] and it is flexible in selecting the synchronization times. Our approach for time and position synchronization is outlined below.

First, each simulator is independently initialized and Gazebo's sampling period is denoted as δ

- At start of the cycle, time t_1 Gazebo forces NS-3 to advance its time by updating the position information of the nodes.
- New positions of the nodes create additional events and NS-3 advances the time to $t_1 + \delta$, if no other network events occur.
- If there are any network events between t_1 and $t_1 + \delta$, the time of the NS-3 advances to time t_2 and forwards the contents of events to Gazebo.
- This communication of network events to Gazebo forces, to update its states and advance the time to t_2.

Typically, NS-3 updates the events, which causes Gazebo to update the states and advance in time to achieve synchronization.

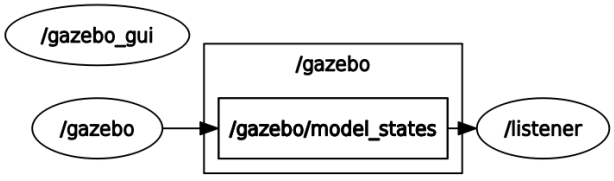

Fig. 5. Rqt_graph of ROS module for time and position

B. Implementation

In our implementation we have used Gazebo simulation time as the reference clock. Gazebo exposes time and state of the models using ROS topics listed below:

- **/clock** - simulation time,
- **/gazebo/link_states** - states of all the links,
- **/gazebo/model_states** - pose of all the models

Setting /use_sim_time parameter to true, notifies ROS modules to use the published /clock topic. So the entire simulation

is synchronized to Gazebo's simulation time. As described in Figure 5 we have used ROS topics to get the pose of the Gazebo models by subscribing to the model_states topic and update the appropriate node's mobility in NS-3 environment. Since the ROS modules are synchronized with Gazebo, the mobility updates by ROS creates new events in NS-3 nodes. This provide time synchronization in NS-3 by advancing in time.

The middleware checks for the packet leaving NS-3 and releases to Gazebo in the next synchronization cycle. If the packets are delivered by NS-3 after the synchronization cycle we discard these packets and flag them. Therefore, for proper functioning of the co-simulation, we have to ensure that the NS-3 event processing is faster than Gazebo, which is the case in normal execution scenarios.

In summary, our method for synchronization is based on the physical interpretation of the Gazebo sampling clock and the NS-3 communication event generation rates. The former represents the rate at which sensor observations are recorded and actuation is completed, while the latter represents the communication delay time-scale. In our method, we use the Gazebo sampling clock to generate mobility events at NS-3. In case such a mobility event occurs before a reception event in NS-3, this can only happen because the simulated communication time delay by NS-3 is larger than the sensor/actuation sampling period of Gazebo. Indeed, this situation arises in practice when the packet delay is long. In such cases, the packet is discarded even when it is received. Our simulation framework follows exactly this policy: reception event occurring after a mobility event (when transmission was started before the previous mobility event) is treated as a packet-lost event. This not only solves our synchronization problem, but also allows us to model packet-losses due to communication-actuation clock mismatch in our simulation framework.

VI. Evaluation

TABLE I
Comparison Table

	FlyNetSim	CUSCUS	AVENS	**CORNET**
Network Simulator	NS-3	NS-3	OMNET++	NS-3
Physics Simulator	Ardupilot SITL	FL-AIR	X-Plane	Gazebo
Mobility	deep packet inspection	shared memory	XML based	ROS topic
data-path	zmq	ns-3 bridge	-	zmq
Time Sync	Yes	No	No	Yes
Open-source	Yes	-	Yes	Yes

A brief comparison between the existing simulators against our work is provided in Table I. Only FlyNetSim provides time synchronization, by timestamping each packet flowing through NS-3. In case of NS-3 lags behind the real-time simulator, they propose to freeze the UAV simulator. In our implementation the clock source is generated by the physics simulator i.e., Gazebo. *So if the network simulator falls behind the physics simulator then that event is discarded and flagged by the middleware, and the physics simulator proceeds to the next sampling time* . This is important as this case cannot be distinguished from the "no packet available" case in the network. This is the true for real networks as well, where one cannot wait indefinitely for network events.

Each simulator in the Table I manages mobility in its own way. In our implementation, we have used a generic approach to get the pose information of the model using ROS topics. The main advantage with our approach is that it can be extended to any robot platform supporting ROS.

A. Case Study I: Offboard control of Single UAV

In this scenario, we control a single UAV from GCS over the WiFi. This is a simple scenario to evaluate the effects of communication on both ends.

In this experiment, the GCS is connected directly (using Ethernet) to the access point (AP) and the UAV is connected over the WiFi. We measure end-to-end network latency for both the real world and the simulation, with the UAV moving away from the AP.

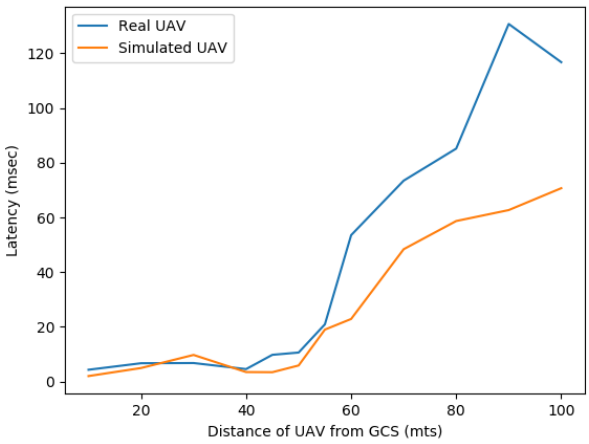

Fig. 6. Latency vs Distance Real and Simulated Drone

The results for network latency as a function of distance between UAV and GCS is illustrated in Figure 6. One can observe that the network latency is increasing with the distance between GCS and UAV. The simulated results also shows an increasing trend of the latency with the distance to the AP. However the errors in simulated versus measured delays for larger distances are large due to a simplistic channel model used in NS-3 for this experiment. This use-case is a simple evaluation of the framework's ability to develop and test new network functions for UAVNETs.

B. Case Study II: Fixed Trajectory Control

In this scenario, the GCS sends commands to the UAV to follow a pre-determined trajectory. This is a way-point based navigation with the controller present in the GCS. Figure 7

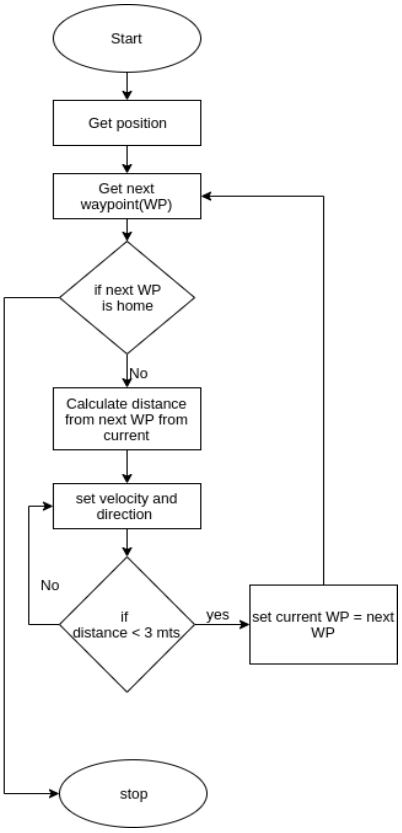

Fig. 7. Simple waypoint navigation

shows the flowchart for the simple controller algorithm that we have used for navigation.

As shown in the Figure 8, A,B,C,D (D is hidden under the drone symbol) are the way-points defined in the GCS controller. A black line represents the pre-determined trajectory that should be followed. The location of the UAV is represented by an arrow labeled with H (Home) and red line indicates the actual trajectory followed by the UAV in the simulation framework. A clear drift is observed between the predetermined and the actual trajectories, and *this is due to latency over the network*. This highlights the need for co-simulation of the UAV and the network. Using such a joint simulation the perturbations caused by the network functions will help to accurately model and test mission-level algorithms for the UAV that then can be reliably applied to navigate real UAVs. This case-study demonstrates the potential algorithms and mission-level applications that can be developed over this framework.

C. Case Study III: Multi-UAV

In this scenario we test the scalability of the framework by instantiating multiple UAVs on single computer. We run the simulator on a laptop with 8 GB of RAM and an Intel Core-i7 7th generation processor. We instantiate 30 UAVs and establish communication to GCS using a single AP with basic functions such as telemetry and control paths. Our framework does not impose any restrictions on number of UAVs, but is only limited

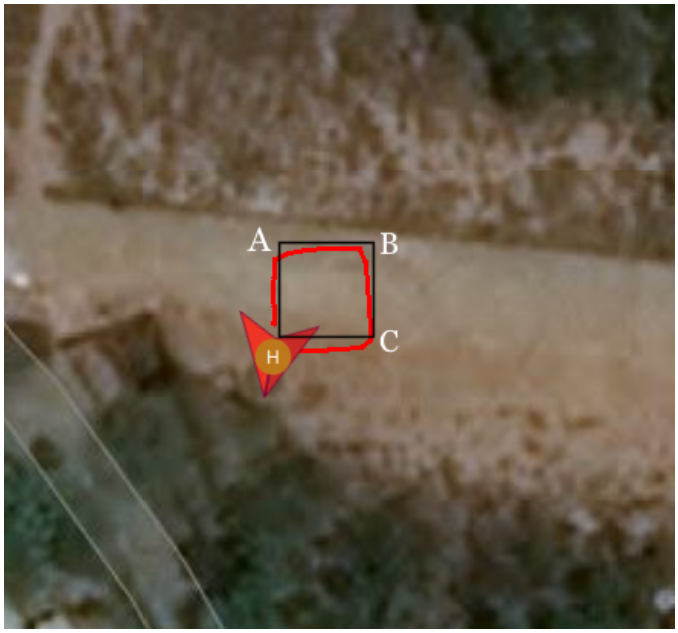

Fig. 8. Fixed Trajectory Control

by the availability of underlying compute resources. We can extend the framework to multiple machines using ROTORS [16] plugin in Gazebo.

VII. Conclusion

In this work, we have presented CORNET, a middleware for simulating networked robots with example applications from a network of UAVs. CORNET connects NS-3, an event driven network simulator to Gazebo which is a physics engine based UAV simulator. CORNET can be extended to any other physics engine simulator that supports ROS objects. A key feature of CORNET is its support for time synchronization between the two simulation environments. In particular, delayed events from NS-3 are dropped (and flagged as errors) by CORNET. This allows them to be treated as communication errors by the application code. Since we have used MAVROS for controlling the UAVs, the same application code can be used to run on real UAV. This approach of co-simulation framework with tightly coupled integration with time synchronization enables research on real-world UAVs in context of urban IoT. We have demonstrated the capabilities of the framework and plan to apply it to LTE and 5G to enable multi-technology networking. This will allow us to evaluate new Flying Ad-hoc Networks (FANETs) algorithms for routing and handover problems in UAVNETs.

Acknowledgment

We would like to thank IISc's 5G-V2X group, Department of Telecom, Govt of India and Indian National Academy of Engineers for their constant support. We are grateful to Varun, who provided expertise in Gazebo simulation that greatly assisted the research.

REFERENCES

[1] Airsim. https://microsoft.github.io/AirSim/. Accessed: 2019-09-01.

[2] Ardupilot. http://ardupilot.org/. Accessed: 2019-09-01.

[3] Bae's magma. https://tinyurl.com/yylhjrxg. Accessed: 2019-09-01.

[4] Blimp drones. https://fortune.com/2016/03/17/drone-blimp-advertising/. Accessed: 2019-09-01.

[5] Early history of flight simulation. https://www.simulationinformation.com/education/early-history-flight-simulation. Accessed: 2019-09-01.

[6] Mavros. http://wiki.ros.org/mavros. Accessed: 2019-09-01.

[7] Px4. https://px4.io/. Accessed: 2019-09-01.

[8] Robot operating system. https://www.ros.org/. Accessed: 2019-09-01.

[9] rqt graph. http://wiki.ros.org/rqt_graph. Accessed: 2019-09-01.

[10] A. I. B. Aakif Mairaj and A. Y. Javaid. Application specific drone simulators: Recent advances and challenges. pages 100–117, 07 2019.

[11] S. Baidya, Z. Shaikh, and M. Levorato. Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 37–45. ACM, 2018.

[12] I. Bekmezci, O. K. Sahingoz, and Ş. Temel. Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270, 2013.

[13] X. Chang. Network simulations with opnet. In *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 1*, WSC '99, pages 307–314, New York, NY, USA, 1999. ACM.

[14] T. Doerffel. Simulation of wireless ad-hoc sensor networks with qualnet. *Documentation*, 2009.

[15] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389. IEEE, 2015.

[16] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. Rotorsa modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[17] M. Hassanalian and A. Abdelkefi. Classifications, applications, and design challenges of drones: A review. *Progress in Aerospace Sciences*, 05 2017.

[18] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[19] A. Y. Javaid, W. Sun, and M. Alam. Uavsim: A simulation testbed for unmanned aerial vehicle network cyber security analysis. In *2013 IEEE Globecom Workshops (GC Wkshps)*, pages 1432–1436. IEEE, 2013.

[20] K. L.-C. Jeff Van West. *Microsoft Flight Simulator X For Pilots:*. 2012.

[21] L. Kang, W. Zhao, B. Qi, and S. Banerjee. Augmenting self-driving with remote control: Challenges and directions. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, HotMobile '18, pages 19–24, New York, NY, USA, 2018. ACM.

[22] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[23] W. Li, M. Ferdowsi, M. Stevic, A. Monti, and F. Ponci. Cosimulation for smart grid communications. *IEEE Transactions on Industrial Informatics*, 10(4):2374–2384, 2014.

[24] H. Lin, S. S. Veda, S. S. Shukla, L. Mili, and J. Thorp. Geco: Global event-driven co-simulation framework for interconnected power system and communication network. *IEEE Transactions on Smart Grid*, 3(3):1444–1456, 2012.

[25] E. A. Marconato, M. V. Rodrigues, R. de Melo Pires, D. F. Pigatto, L. C. Q. Filho, A. R. Pinto, and K. R. L. J. C. Branco. Avens - a novel flying ad hoc network simulator with automatic code generation for unmanned aircraft system. In *HICSS*, 2017.

[26] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017.

[27] S. Popić, D. Pezer, B. Mrazovac, and N. Teslić. Performance evaluation of using protocol buffers in the internet of things communication. In *2016 International Conference on Smart Systems and Technologies (SST)*, pages 261–265. IEEE, 2016.

[28] A. Sanders. *An Introduction to Unreal Engine 4*. AK Peters/CRC Press, 2016.

[29] A. Suzuki, K. Masutomi, I. Ono, H. Ishii, and T. Onoda. Cps-sim: Co-simulation for cyber-physical systems with accurate time synchronization. *IFAC-PapersOnLine*, 51(23):70–75, 2018.

[30] M. Tsukada. Roadside-Assisted V2V Messaging for Connected Autonomous Vehicle. In *The Thirteenth International Conference on Wireless and Mobile Communications*, Nice, France, July 2017. IARIA.

[31] N. Zema, A. Trotta, G. Sanahuja, E. Natalizio, M. Felice, and L. Bononi. Cuscus: An integrated simulation architecture for distributed networked control systems. pages 287–292, 01 2017.