2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS) | 978-1-6654-9512-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/DCOSS54816.2022.00061

SynchroSim: An Integrated Co-simulation Middleware for Heterogeneous Multi-robot System

Emon Dey[1], Jumman Hossain[1], Nirmalya Roy[1], Carl Busart[2]
[1]Center for Real-time Distributed Sensing and Autonomy
[1]Department of Information Systems, University of Maryland, Baltimore County, USA
[2]DEVCOM Army Research Lab, USA
[1]{edey1, jumman.hossain, nroy}@umbc.edu
[2]carl.e.busart.civ@army.mil

***Abstract*—With the advancement of modern robotics, autonomous agents are now capable of hosting sophisticated algorithms, which enables them to make intelligent decisions. But developing and testing such algorithms directly in real-world systems is tedious and may result in the wastage of valuable resources. Especially for heterogeneous multi-agent systems in battlefield environments where communication is critical in determining the system's behavior and usability. Due to the necessity of simulators of separate paradigms (co-simulation) to simulate such scenarios before deploying, synchronization between those simulators is vital. Existing works aimed at resolving this issue fall short of addressing diversity among deployed agents. In this work, we propose *SynchroSim*, an integrated co-simulation middleware to simulate a heterogeneous multi-robot system. Here we propose a velocity difference-driven adjustable window size approach with a view to reducing packet loss probability. It takes into account the respective velocities of deployed agents to calculate a suitable window size before transmitting data between them. We consider our algorithm specific simulator agnostic but for the sake of implementation results, we have used Gazebo as a Physics simulator and NS-3 as a network simulator. Also, we design our algorithm considering the Perception-Action loop inside a closed communication channel, which is one of the essential factors in a contested scenario with the requirement of high fidelity in terms of data transmission. We validate our approach empirically at both the simulation and system level for both line-of-sight (LOS) and non-line-of-sight (NLOS) scenarios. Our approach achieves a noticeable improvement in terms of reducing packet loss probability (≈11%), and average packet delay (≈10%) compared to the fixed window size-based synchronization approach.**

***Index Terms*—Heterogeneous multi-robot systems, NS-3, Gazebo, Co-simulation, Synchronization algorithm.**

I. Introduction

In the field of contemporary robotics, multi-agent systems are set to play a vital part. Due to their ability of forming large interconnected networks with coordination among agents make them an integral part in a variety of robotic applications [1], [2]. For example, Unmanned Aerial Vehicle (UAV) systems are being increasingly used in a broad range of applications requiring extensive communications, either to collaborate with other UAVs [3] with each other or with Unmanned Ground Vehicles (UGV) [4], [5]. Specifically in battlefield scenarios where the presence of heterogeneous aerial and ground vehicles coordinating with each other is going to be an essential feature in the future. The mutual information transfer between Unmanned Aircraft Systems (UAS), and ground robots can help to make intelligent decisions in a critical time.

Synchronized communication between UAVs and UGVs can aid in developing situation awareness in the battlefield among each deployed device, and also help execute any specific command through them. Executing such systems directly in the actual environment may bring in harmful consequences as they necessitates the extensive fine-tuning of algorithm parameters [6]. As a result, it is required to simulate the system beforehand using proper technologies in order to establish a baseline of confidence. Being motivated by this scope, research has been started on simulating such an environment to estimate the probable nature of robots before going to actual deployment. The main bottleneck in this endeavor lies in the necessity of synchronizing two different simulators having disparate operating principles [7], [8]. One of these is physics simulators that account for replicating the interaction between physical robots and their operating environment. On the other hand, network simulators try to estimate the deployed agents' communication performance over the network.

So, the primary goal should be to connect the two domains by using existing open-source tools to record closed loop simulation. This is because multi-agent systems are strenuous to build; easy and coordinated communication across diverse technologies can make this process somewhat less difficult. Some of the existing works have already addressed those challenges. For example, FlyNetSim [7], ROS-NetSim [6], CORNET [4], CPS-Sim [9], RoboNetSim [10] are some of such kinds of works implemented on AirSim [11], ARGoS [12], Gazebo [13] as physics simulator and OMNeT++ [14], NS-2 [15], NS-3 [16], Mininet [17] as network simulator. As a whole, some of the major drawbacks existent is those works are: i. compatible with either UAVs or UGVs, not heterogeneous systems, ii. causes low co-simulation speed, iii. gives rise to float-point arithmetic error, iv. difficult to set up proper window size for diverse multi-agent setup, and v. loses synchronization when the simulators relative speed varies.

With a view to developing a suitable ROS-compatible synchronizing middleware for the aforementioned scenario, we propose *SynchroSim*. It takes into account the number of agents deployed within a certain cluster and can select

2325-2944/22/$31.00 ©2022 IEEE
DOI 10.1109/DCOSS54816.2022.00061

the most suitable sliding window while sending data among agents. For simulation experiments, we use two distinct open-source simulation engines, Gazebo and NS-3. Gazebo uses the computer's system clock as the simulation time, while NS-3 presents the process as a distinct sequence of time events. Additionally, we observe the performance of our algorithm upon deploying on a cluster consisting of the real world UAVs and UGVs. To the best of our knowledge, it is the first endeavor pointing out the impact of synchronization middleware on a heterogeneous multi-agent system considering battlefield as application scenario. The major contributions we claim here are:

- *Co-simulation of heterogeneous multi-agent systems i.e., UGV (ground robots), UAV (drones), etc. and preserving synchronization among them:* We propose a simulator independent co-simulation setting for multi-agent heterogeneous environment. Furthermore, we extend our simulation work into real world robots to validate the actual deployment performance.
- *Improvising sliding window-based synchronization scheme for diverse multi-agent system :* We offer an application specific modification of sliding window based synchronizing middleware. We name our approach as *SynchroSim* which can vary the window size considering the velocity difference of the agents for the sake of synchronization among them with better communication performance.
- *Empirical evaluation considering different application scenarios:* We present our experiment, taking into account both line-of-sight (LOS) and non-line-of-sight (NLOS) communication scenarios on the basis of probability of packet loss, and average packet delay. We have employed Gazebo as Physics simulator and NS-3 as network simulator to report our experimental results. Experimental results show that our approach works better in comparison with the traditional fixed window based method in ensuring fewer packet losses (on average 10% improvement) even in challenging NLOS environment with heterogeneous agents.

II. Related work

In this section, we will briefly outline the work that has been done in relation to the various aspects of our system.

A. Simulation Tools

While a comprehensive assessment of currently utilized simulators is beyond the scope of this paper, we highlight a few recent noteworthy physics and network simulators that are most relevant to our setting and have had a significant influence on this study.

Aiming to bridge the gap between simulation and reality, AirSim [11] is an open-source platform that is being developed to assist in the development of autonomous vehicles. AirSim provides high-fidelity physical and visual simulation that enables the rapid generation of massive amounts of training data for the development of machine learning models. It's API design enables algorithms to be developed against a simulator and then deployed unchanged on real vehicles. ANVEL [18], [19] provides such a toolkit by integrating popular graphical representation approaches, such as those used in video games, with physically based sensor and UGV platform models. While both AirSim and ANVEL have major simulation capabilities, difficulty arises when creating large-scale complex visually rich environments that are more realistic in their representation of the real world, and they have fallen behind various advancements in rendering techniques made by platforms such as Unreal engine or Unity [20]. We prefer to employ Gazebo in our communication-realistic scenario because of its superior realism. Gazebo [5] includes a modular design that enables the usage of various physics engines, sensor models, and the creation of 3D worlds to be implemented.

In the case of network simulators, OMNeT++ [14] is a discrete event network simulation framework that is object-oriented and modular in design. Additionally, parallel distributed simulation is supported by OMNeT++ and inter-participant communications can be accomplished through a variety of methods. A network emulation program, mininet [17], allows users to create a realistic virtual network on a single computer by running genuine kernel, switch and application code on the network emulator. However, we choose a state-of-the-art event based simulator NS-3 [17], in which the scheduler typically performs the events in a sequential manner without syncing with an external clock. The NS–3 simulator includes representations of all of the network models that make up a computer network including network nodes, network devices, communication channels, communication protocols, protocol headers, and network packets.

B. Synchronization Methods

At this point, we provide a summary of existing synchronization methods.

CORNET [4] is a variable-stepped multi-robot system simulation framework that blends physics and network simulators. CORNET ensures that only one event process can be running at a time, and a global event scheduler maintains a list of all the events from both simulators and schedules. CORNET has the most significant downside is that it has the potential to cause float-point arithmetic error. CORNET 2.0 [21] extended the implementation of CORNET to make it applicable to any robotic framework and the scalability to manage an increasing number of robots has been demonstrated. FlyNetSim [7] maintains a time-stepped-schedule mechanism, however; simulated network events must be buffered until the next sample time if a network simulator runs faster than the physical simulation. Using real-world UAVs and sensors in a simulated complicated environment is possible with the support of emulation mode in FlyNetSim where the network simulator allows a real UAV to communicate with external or simulated resources.

On the other hand, CPS-Sim [9] operates on the basis of a global event-driven system where the server is forced to reproduce the exact same time-steps. Global scheduler-processed events resolve the issue inherent in the time-stepped

method, albeit at the expense of overall co-simulation speed. ROS-NetSim [6] is a sliding window based technique that keep track of and record network events during the course of the window period, and then enable the network simulator to step up to and through the end of the window, but it is difficult to configure the appropriate window size for multi-agent systems. In this consideration, we devised the sliding window method for our heterogeneous multi-agent system. Additionally, we have added a comparison table. (Tab. I) for different synchronization mechanisms to get a better idea about their capabilities and limitations.

In summary, UAV and UGV components are being simulated with 3D visualization using Gazebo, while the network infrastructure is being provided by NS-3 and middleware is being developed for the creation of an inter-simulation data-path with time and position synchronization at both ends using our co-Simulation of robotic networks.

III. Overview

In modern battlefield scenarios, where intelligent and different sensor-equipped robots are envisioned to co-exist with soldiers. Those robots can collect data with the integrated sensors and can take essential decisions on their further step through analyzing the collected data. This can aid in increasing situational awareness if the derived information can be exchanged with other deployed agents and with the base stations also. Such a scenario is illustrated in fig 1. Here, the

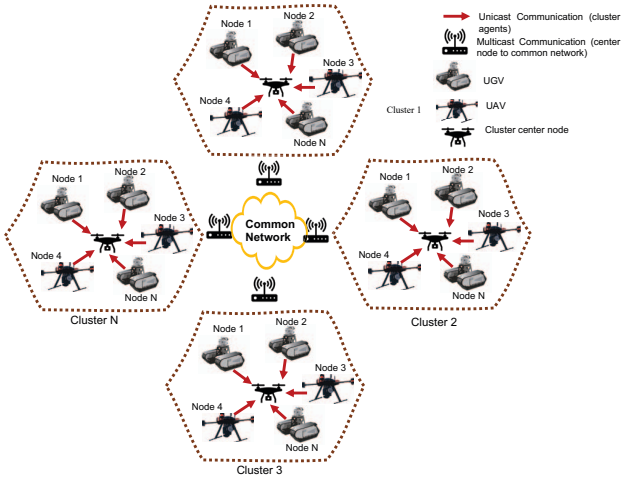


Fig. 1: A sample multi-master communication framework.

total deployed agents are divided into several clusters. Each cluster has a cluster head. The cluster heads can communicate with the agents within the cluster and also among themselves. Furthermore, all of them can report to a central base station. Now, before going to the deployment with actual robots of such systems, it is imperative to simulate such scenarios to get an idea about the probable behavior after deployment. For this reason, a combination of a Physics simulator and a Network simulator is needed to emulate such a communication scenario in an acceptable manner. But synchronization problem arises when these two kinds of simulators are asked to collaborate. As a solution to this, a synchronization middleware can be used to bridge that gap. A working flowchart of such systems is shown in fig 2. This system takes input

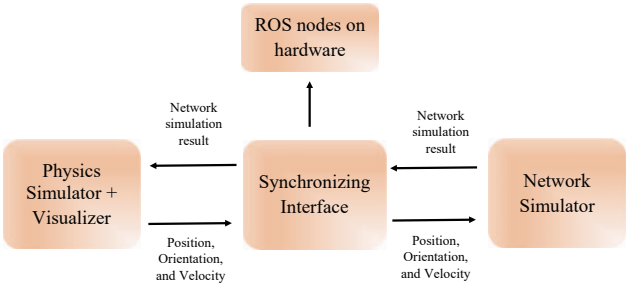


Fig. 2: A probable solution to the synchronization issue between heterogeneous simulators.

from the Physics simulator, the middleware then aligns those data with a networking event. The Network simulator then measures the communication performance and those results can be displayed on the physics simulator interface if needed. Moreover, at the time of actual deployment, it is possible to run the ROS nodes of synchronization middleware on the integrated computing devices of the robots.

IV. Methodology

In this section, we will walk you through the details of our proposed algorithm and its working procedure on the selected application.

A. SynchroSim working Procedure

As described in the earlier sections, our work focuses mainly on the synchronization aspect of heterogeneous multi-agent setup. Our proposed algorithm is based upon the sliding window-based synchronization approach proposed in [6]. The pivotal design parameter while working with a multi-agent system is the choice of window size. Fixed window size at all points will not applicable in such a scenario as different agents may work better with different radio frequencies and also there can be disparities in terms of hardware specifications. *SynchroSim* takes into account the total number of agents present for a specific scenario and starts with a manually initiated value for window size. In the event of the data transfer, it selects the respective publisher and subscriber for that event. After that, it rigorously checks the velocity difference of the participating agents to calculate a suitable window size for a specific communication event. This approach becomes more evident when it comes to the transmission scenario between a ground and an aerial vehicle. To maintain the transmitted information level at a satisfactory point, we chose to utilize packet loss probability value, one of the most important parameters to monitor while sending valuable information, in the operating moment of *SynchroSim*. The following equation is employed to inherently calculate packet loss probability and

TABLE I: Overview of existing synchronization methods.

Middleware	Synchronization Method	Principle	Drawback	Compatibility
Ros-NetSim [6]	Sliding Window	Capture and track network events over the window period and allow the network simulator to step up to the end of the window.	Difficult to set up proper window size for multi-agent systems.	ROS1, UAV & UGV
CORNET [4]	Variable-stepped	A global event scheduler maintains the list of all the events from both the simulators and schedules according to their timestamps allowing only one event process to run at a time.	Float-point arithmetic error	ROS1 & UAV
FlyNetSim [7]	Time-stepped with scheduler	Common sampling period to be used by both simulators.	If the network simulator runs faster than the physics simulator, the network events must be buffered in a cache and wait to be processed until the next sampling time.	ROS1 & UAV
CPS-Sim [9]	Global event driven	The server is forced to reproduce the exact same time-steps.	Global scheduler processed events overcomes the problem that occurs in time-stepped method but limits the overall co-simulation speed.	Not ROS based

report within a synchronization window:
Packet loss probability,

$$ {}^{i}_{k}Lp = 1 - \frac{{}^{i}N_{sb}}{{}^{i}_{k}N_{pb}} \tag{1} $$

Here, ${}^{i}_{k}N_{sb}$ is the number of data packets delivered to the subscriber and ${}^{i}_{k}N_{pb}$ is the number of data packets transmitted from the publisher. More technical details on the process can be found in Algorithm 1.

Algorithm 1 Multi-agent synchronization algorithm with adjustable window

Require: Total number of agents D, window size w, number of data packets N, velocity of agents V

Ensure: Synchronization between simulators and calculate packet loss probability Lp

1: **Initialize:** Publisher P and subscriber S agents where $P, S \in D$, begin time = 0, window size w
2: **Start simulation:** Transmit data packet of selected topic from publisher
3: *Update* with begin time t = 0
4: **if** running event found **then**
5: **Window adjustment:** Get the velocity of Publisher V_p and Subscriber V_s in meters/sec
6: Calculate the adjusted window,

$$ w_a = w + (V_p - V_s)/1000 \tag{2} $$

7: **Synchronism Period:** Wait until begin time = t
8: **Report Lp:** Calculate packet loss probability for the synchronized event
9: **Report finished window:** Send *Update* with end time = t
10: **Timestamp update:** *Update* $t = t + w_a$ and request for next window
11: **end if**

B. Updating Physics and Network Simulator

At the beginning of the simulation, the Physics simulator determines two key pieces of information about the agents used for a specific communication round: the distance between the agents through positional coordinates and velocity. The network simulator stores information on which communication scheme to use (we have chosen a TCP/IP-based approach over UDP for reliability issues), the number of packets, packet length, and the IPs of both publisher and subscriber agents. Then the Physics simulator and network simulator are advanced with one initially fixed window size *w*. Upon the advancement, the Physics simulator passes the distance and velocity information to the network simulator, and the network simulator calculates the packet loss probability. If the loss value is within a certain threshold, the result is sent to display. Otherwise, this information is reported back to the *Synchrosim* module and the initial window size is adjusted according to the algorithm 1. This process continues until a satisfactory result is achieved for this specific round, and then the initialization process starts over with a new set of agents.

C. Publisher-Subscriber Architecture

SynchroSim utilizes a publisher subscriber-based architecture to accomplice the data transmission task. We have considered image as our data type. A cluster was set up with a combination of a drone and two ground bots. Heterogeneity was maintained while choosing the ground bots and master-based ROS [22] communication was implemented. In the case of communication through the master, all the agents were set to run within a certain area. The flowchart for this master-based setup is illustrated in fig 3. The drone was chosen as the master node and it contained the IPs of all the UGVs. Image data were streaming from all the UGVs. We have utilized the Rosbag concept to record the published messages and select any specific frame to transmit. Rosbag stores the information it is programmed to save with every timestamp. The images information can be extracted from that inventory. When an

event was initiated, the selected image frame was sent to the master node. The master node then published the relevant ROS topic and any agent subscribed to that specific topic can be eligible to receive that image data. After the completion of this transmission process, the packet loss probability for that event was monitored.

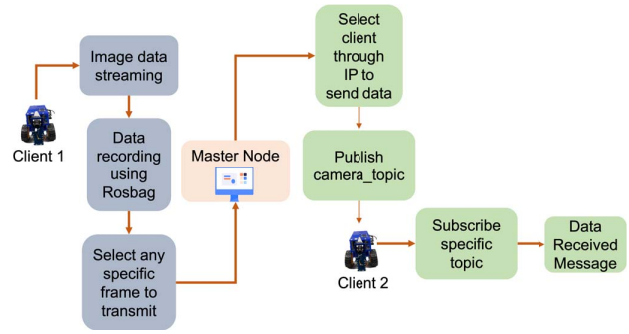


Fig. 3: Data transmission flowchart through the master node.

D. Setting up clusters and communication schemes

In the wireless communication domain, a cluster of robots is a popular setting to measure the inter-agents data transmission performance. In application scenarios involving contested environments such as modern battlefields where the flow of information among heterogeneous multi-robot systems is crucial to increase situational awareness. Among different types of cluster setup, the master-based system is being investigated intensively in recent ROS-based research. A sample master-based multi-agent cluster is illustrated in fig 4. In this work, we will also utilize this scheme.

To bolster the claim of a working synchronizing middleware

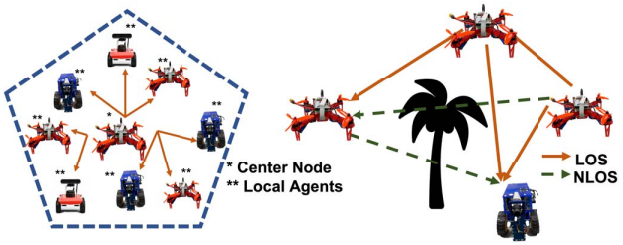
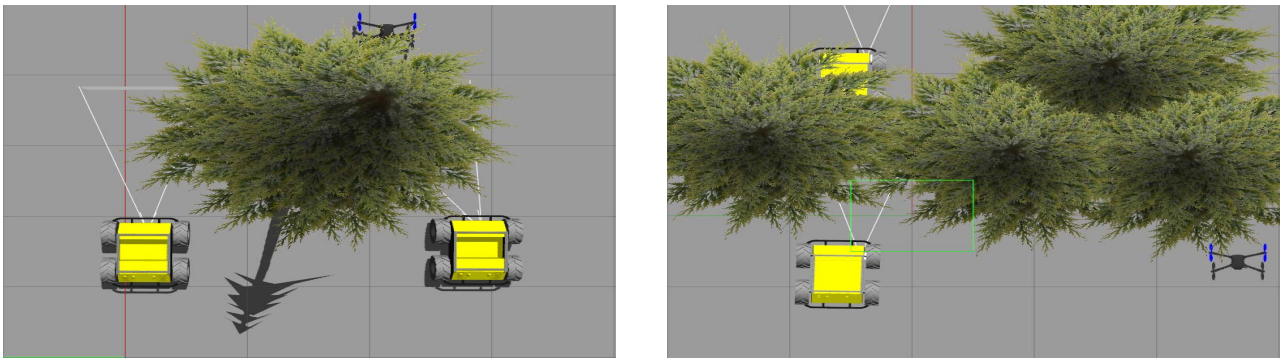


Fig. 4: A cluster consisting multi-agent heterogeneous systems (left), and Visual demonstration of LOS and NLOS communication (right).

it is imperative the set up detailed experimentation with two of the most used communication scheme: line-of-sight (LOS), non-line-of-sight (NLOS). To define those two schemes, when there is no obstruction in between the receiver and sender agents, the data transmission rate becomes higher and this scenario is known as LOS communication. It is the most desirable situation for wireless signal transmission. The NLOS scheme is the opposite of LOS communication and it is the most common one in the real-world environment. The signal transmission can be hindered by both natural and man-made objects. So, a better design of the data transmission scheme is necessary to avoid the loss of valuable information. A simple visual representation of LOS an NLOS system can be found in fig 4. For our experimentation, we have simulated both UGV and UAV with artificial environment settings containing both types of communication scenarios.

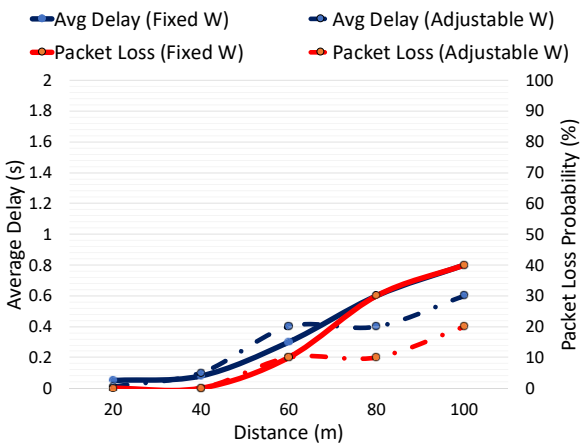
V. Simulation Setup

To validate the working capability of *SynchroSim*, we have arranged simulation scenarios for both LOS/NLOS channels. The environments are launched on Gazebo; the Physics simulator in our case. The dimension of the chosen environment is 100×100 meters as shown in fig 5. Where each of the grids signifies 20×20 meters. The environment is designed to host both LOS/NLOS scenarios and for NLOS abstraction, the environment is populated with trees mostly. The choice of robotic agents is done in a heterogeneous fashion; contains both UGV and UAV. Iris drone with integrated camera as UAV, and two Husky robots are selected as UGV. One of the Husky robots is considered the master node. We have used the *MAVROS* package to establish the connection between the Gazebo and the Iris drone, which runs on the *MAVLink* autopilot setup and an integrated PX4 flight stack to maneuver the drone. For the Husky simulation, we modified *Clearpathrobotics'* official GitHub implementation process according to our use case. For wireless communication medium, we have used IEEE 802.11 (Wi-Fi) interface. The agents are set to stream datapoints through TCP/IP link and the master node has the IP address of each of the deployed agents. The integrated camera of those agents is used as the primary sensor and the streaming image frames are resized into 32×32 before treating them as camera topic of the ROS system. To measure the communication performance, we have chosen one of the state-of-the-art network simulators, NS-3 which is compatible with the selected wireless stack. To integrate both Gazebo and NS-3, *SynchroSim* is deployed as synchronizing middleware. The choice of window size in *SynchroSim* is dependent upon the respective velocity of the agents. To execute this scenario, the velocity of the agents are varied accordingly. The initial window size is set to 1mS and is adjusted with Algorithm 1. For the LOS scenario, the agents are deployed in an environment without obstruction as shown in fig 5. The UGVs and the UAV are simulated with varying speeds and the communication performance is measured with the LOS abstraction is satisfied. The performance matrices are the average delay and percentage of packet loss as stated in section IV. The results are reported with the change in distance between the agents. Also, two different communication scenarios are considered which we define as UGV to UGV and UAV to UGV communication. The distance values reported while reporting the experimental results are derived from the positional coordinates of the agents. The whole simulation area is segmented into symmetrical blocks and the synchronization operation is operated when the agents move to some certain points of the environment.

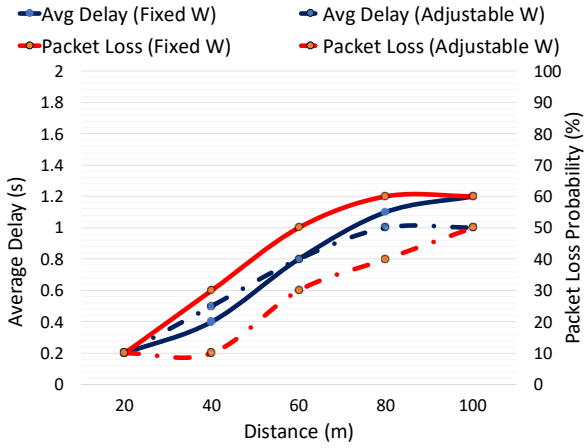


(a) (b)

Fig. 5: Simulation environment design in Gazebo for (a) LOS, and (b) NLOS communication. The whole environment is 100m×100m, and each grid stands for 20m×20m.



(a)



(b)

Fig. 6: Result comparison between fixed window and adjustable window based approaches on both (a) LOS, and (b) NLOS communication scheme using average delay (s) and packet loss probability (%) matrices considering a UGV to UGV communication scenario. The fixed window size chosen here is 1ms and the average delay is reported on a batch of 10 packets. Blue lines stand for the average delay values and orange lines are for percentage of packet loss. In both cases the dotted lines signify the results of our proposed method.

In the case of NLOS abstraction, the agents are set to run within an object-populated environment. The signal strength is hypothesized to be hampered under this circumstance. The same reporting paradigm and communication scenarios are used for this case also.

During UGV to UGV communication where the relative velocity difference is lower, the agents are noticed to sustain the data quantity during transmission for about 40m as shown in fig 6. The average delay (average value of delays during 10 image frame transmission) is also below 0.1s in this case. The velocity information of two selected agents for a communication incident is extracted through subscribing to $/gazebo\_states/twist$ topic. The baseline fixed window-based approach and the proposed adjustable window method are working at par for LOS communication up to this point. But fixed window-based approach begins to lose valuable information drastically after this point subsequently giving rise to delay. A slight adjustment in the sliding window value (0.1ms in this case) is seen to contribute to almost 20% improvement in retrieving information for LOS and at least 10% for NLOS condition when the agents are the furthest distance apart in this simulation (100m). Also, the transmission is becoming faster by a similar percentage compared to the baseline.

While experimenting with UGV to UAV, severe performance degradation is seen for NLOS communication, illustrated in fig 7. To be specific, almost all the data are lost when the distance is 100m between the agents for the fixed width

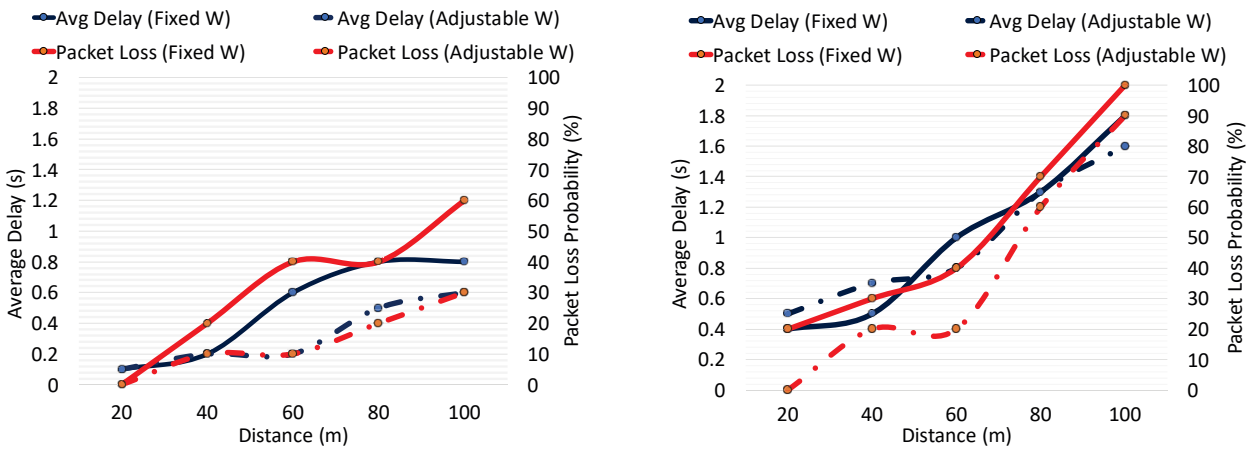


Fig. 7: Result comparison between fixed window and adjustable window based approaches on both (a) LOS, and (b) NLOS communication scheme using average delay (s) and packet loss probability (%) matrices considering a UGV to UAV communication scenario. Same measurement paradigm and legend conventions as fig 6 are used for this figure also.

method. This scenario also affects the proposed adjustable window approach (0.3ms increase) which leaves a point of improvement. Apart from this specific point, the adjusted window showcases significant improvement for both LOS and NLOS communication. Noticeably, the packet loss probability is reduced by almost 30% when we consider a LOS scenario and the agents are furthest apart.

VI. System Implementation

In this section, we describe the devices used for our hardware-level implementation, procedure details, and performance evaluation.

With a view to validating the application compatibility of our proposed algorithm on actual systems, we have conducted a system-level implementation and evaluation. We have chosen ROS-compatible Duckiebots to act as our deployed agents. A brief description of the configuration of such robots are given below:

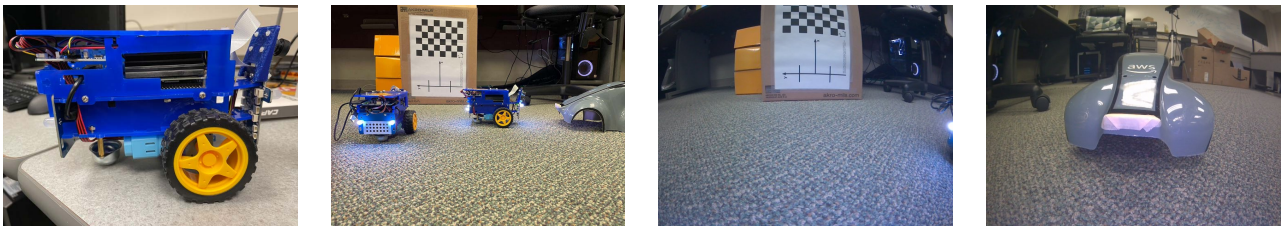
Duckiebot:

The Duckiebot is an autonomous platform developed for research purposes and convenient for studying complex real-world problems. The bot used in our work has sensors like a front-facing camera, programmable LED, IMU; camera inputs are used for our implementation. The camera used in this specific Duckiebot version is a 5MP, 1080p camera with a wide view range of 160 degrees. We have assembled the robot from scratch to make sure it is equipped with all the necessary components we need to operate the experiment. A fully assembled Duckiebot is illustrated in fig 8. Its onboard processor of it is Nvidia Jetson Nano (2GB). Docker containers are built to run the ROS nodes and all the necessary scripts are written on Python.

As stated in the earlier sections, we have implemented a master-based multi-agent communication scheme. This implementation operation is carried out as a corroboration attempt of the simulation results obtained and described in section V. An Ubuntu desktop was set up as the master node and two Duckiebots were clients. The master node will have the TCP of each agent and can receive or send information to any cluster agents based on the requirement. This implementation is carried out through a Publisher-Subscriber approach based on ROS Melodic. The experimentation is done in a laboratory environment. The experimental setup along with the sample camera outputs (primary sensor) of the two Duckiebots are illustrated in fig 8. The Duckiebots are set to roam around in an approximately 2m x 2m area and some sample objects are placed within it. The robots are continuously capturing image frames and as soon as an object is detected by one bot, it will send the corresponding frame to the master node. The master node will then send this information to the other bot. A visualizer is designed to check the completion of data transmission and also the communication performance packet delay and packet loss probability. The synchronization algorithm was deployed on the docker container to run on behind when it is needed to display the forwarded data from the clients and also the performance metric values. The data used for transmission is a gray scale image which is resized in 32×32 before sending. Under the aforementioned laboratory setup, *SynchroSim* works seamlessly in a confined small scale setting, ensuring negligible delay and no packet loss for both LOS and NLOS abstraction. This successful demonstration makes us hopeful to achieve satisfactory performance with our ongoing deployment endeavor in wild contested environment.

VII. Conclusion and Future Work

In this work, we have presented *SynchroSim*, a middleware for co-simulation of heterogeneous multi-robot systems.



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

Fig. 8: (a) A fully assembled Duckiebot (UGV), (b) Image data transmission between two Duckiebots using a master based system in a laboratory environment. Sample sensor output of (c) agent 1, and (d) agent 2 within the experimental setup.

SynchroSim can adjust the size of the window based on the speed differences between the agents in order to provide better synchronization and communication. Even in the challenging non-line-of-sight (NLOS) environments with heterogeneous agents, experimental data show that our solution outperforms the standard fixed window based strategy in terms of ensuring fewer packet losses (on average 10% improvement). Furthermore, we have also arranged a small scale cluster in a laboratory environment with real world UGVs to get an idea of the applicability of our proposed synchronizing approach when it comes to real terrain. In this work, we have demonstrated our synchronizing method on master-based communication system. However, one interesting extension can be implementing a masterless communication to ensure more data security; one of the fundamental requirements of battlefield scenarios. We plan to explore the facility offered by ROS2 on top of our current system with the help of ROS bridge to make the robots enable to use the Data Distribution Service (DDS).

ACKNOWLEDGMENT

This work has been supported by U.S. Army Grant #W911NF2120076. The authors would also like to thank Dr. Kasthuri Jayarajah and Dr. Aryya Gangopadhyay for their constructive feedback on this work, Jonathan Harwood for setting up the simulation and Sreenivasan Ramasamy Ramamurthy for helping with Duckiebot assembly.

REFERENCES

[1] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a survey. *ACM Computing Surveys (CSUR)*, 51(3):1–33, 2018.

[2] Taemin Ahn, Jihoon Seok, Inbok Lee, and Junghee Han. Reliable flying iot networks for uav disaster rescue operations. *Mobile Information Systems*, 2018, 2018.

[3] Argho Sarkar and Maryam Rahnemoonfar. Uav-vqg: Visual question generation framework on uav images. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4211–4219. IEEE, 2021.

[4] Srikrishna Acharya, Amrutur Bharadwaj, Yogesh Simmhan, Aditya Gopalan, Parimal Parag, and Himanshu Tyagi. Cornet: A co-simulation middleware for robot networks. In *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 245–251. IEEE, 2020.

[5] Sangwoo Moon, John J Bird, Steve Borenstein, and Eric W Frew. A gazebo/ros-based communication-realistic simulator for networked suas. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1819–1827. IEEE, 2020.

[6] Miguel Calvo-Fullana, Daniel Mox, Alexander Pyattaev, Jonathan Fink, Vijay Kumar, and Alejandro Ribeiro. Ros-netsim: A framework for the integration of robotic and network simulators. *IEEE Robotics and Automation Letters*, 6(2):1120–1127, 2021.

[7] Sabur Baidya, Zoheb Shaikh, and Marco 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, 2018.

[8] Aaron Staranowicz and Gian Luca Mariottini. A survey and comparison of commercial and open-source robotic simulator software. In *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, pages 1–8, 2011.

[9] Atsushi Suzuki, Kazuyuki Masutomi, Isao Ono, Hideaki Ishii, and Takashi Onoda. Cps-sim: Co-simulation for cyber-physical systems with accurate time synchronization. *IFAC-PapersOnLine*, 51(23):70–75, 2018.

[10] Michal Kudelski, Luca M Gambardella, and Gianni A Di Caro. Robonetsim: An integrated framework for multi-robot and network simulation. *Robotics and Autonomous Systems*, 61(5):483–496, 2013.

[11] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[12] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, et al. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4):271–295, 2012.

[13] Gazebo. https://gazebosim.org/. Accessed: 2021-12-17.

[14] Andras Varga. Omnet++. In *Modeling and tools for network simulation*, pages 35–59. Springer, 2010.

[15] Ns-2. https://www.isi.edu/nsnam/ns/. Accessed: 2021-12-17.

[16] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[17] Mininet. http://mininet.org/. Accessed: 2021-12-17.

[18] Phillip J Durst, Christopher Goodin, Chris Cummins, Burhman Gates, Burney Mckinley, Taylor George, Mitchell M Rohde, Matthew A Toschlog, and Justin Crawford. A real-time, interactive simulation environment for unmanned ground vehicles: The autonomous navigation virtual environment laboratory (anvel). In *2012 Fifth international conference on information and computing science*, pages 7–10. IEEE, 2012.

[19] MaryAnne Fields, Ralph Brewer, Harris L Edge, Jason L Pusey, Ed Weller, Dilip G Patel, and Charles A DiBerardino. Simulation tools for robotics research and assessment. In *Unmanned Systems Technology XVIII*, volume 9837, page 98370J. International Society for Optics and Photonics, 2016.

[20] Unity simulator. https://unity.com/products/unity-simulation-pro. Accessed: 2021-12-17.

[21] Srikrishna Acharya, Amrutur Bharadwaj, Bharatheesha Mukunda, and Yogesh Simmhan. Cornet 2.0: A co-simulation middleware for robot networks. ArXiv, 2021.

[22] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.