The CUSCUS simulator for distributed networked control systems: Architecture and use-cases

The CUSCUS simulator for Distributed Networked Control Systems: Architecture and Use-cases

Nicola Roberto Zema[a,*], Angelo Trotta[b], Enrico Natalizio[a], Marco Di Felice[b], Luciano Bononi[b]

[a] *Sorbonne Universités, Université de Technologie de Compiègne, Heudiasyc, UMR CNRS 7253, France.*
{*enrico.natalizio,nicola.zema*}*@hds.utc.fr.*
[b] *Department of Computer Science and Engineering, University of Bologna, Italy.*
{*trotta, difelice, bononi*}*@cs.unibo.it*

Abstract

The current merging of networking and control research fields within the scope of robotic applications is creating fascinating research and development opportunities. However, the tools for a proper and easy management of experiments still lag behind. Although different solutions have been proposed to simulate and emulate control systems and, more specifically, fleets of Unmanned Aerial Vehicles (UAVs), still they do not include an efficient and detailed network-side simulation, which is usually available only on dedicated software. On the other hand, current advancements in network simulations suites often do not include the possibility to include an accurate description of controlled systems. In the middle 2010s, integrated solutions of networking and control for fleets of UAVs are still lacking. In this paper, we fill such gap by presenting a simulation architecture for networked control systems which is based on two well-known solutions in both the fields of networking simulation (the NS-3 tool) and UAV control simulation (the FL-AIR tool). Three main research contributions are provided: (*i*) first, we show how the existing tools can be integrated on a closed-loop architecture, so that the network propagation model (NS-3 side) is influenced by the drone mobility and by the 3D scenario map (FL-AIR

*Corresponding author
Email address: nicola.zema@hds.utc.fr (Nicola Roberto Zema)

side); (*ii*) second, we implement a novel module, which allows modeling realistic 3D environments by importing city-wide characteristics by the popular OpenStreetMap service; (*iii*) third, we demonstrate the modeling capabilities of the CUSCUS framework on two realistic use-cases, corresponding to well-known application scenarios of UAVs, i.e. dynamic formation control and static coverage of a target area.

Keywords: Networked Control Systems, UAVs, Simulator

1. Introduction

Aerial networks composed by Unmanned Aerial Vehicles (UAVs) constitute emerging cooperative systems characterized by unique features such as distributed coordination, autonomous 3D mobility, and context-awareness through the sensing capabilities [1]. In the next few years, the pervasive diffusion of UAVs is expected to pave the way to novel scenarios integrating IoT devices, aerial communications and mobile/multimedia applications. At the same time, the state of art of UAVs already includes a wide range of real-case deployments, from disaster recovery to surveillance and precision agriculture [2][3][4].

A key issue in most of the mentioned scenarios is the management of flying nodes' autonomous mobility in order to meet the Quality of Service (QoS) requirements of the applications [1]. In absence of a centralized controller, the fleet mobility is determined by decisions performed at each UAV, hence *consensus-based* or *distributed coordination* protocols are needed to avoid collisions, keep the network connected and achieve the mission-specific goals [5]. At the same time, the communication among nodes is strongly affected by the *propagation conditions* of the environment, so far that the effect of packet loss must be taken into account in networked robotic architecture design [6][7]. Finally, since the micro-mobility of each UAV involves complex electromechanical dynamics, *robust controllers* are required for tuning the parameters governing the position and orientation of the flying node (e.g. the Proportional, Integrative and Derivative terms of the controller) [8]. The merging of networking and control fields is

a natural consequence of the above mentioned issues: several communication-aware mobility schemes have been proposed for fleet creation and management [9][10]. Similarly, there exists plenty of communication protocols at the MAC, network and transport layers, which are specifically tailored to the UAVs scenarios, in order to cope with the dynamic topology and, at the same time, take maximum benefit from the self-placement capabilities of the nodes [11][12]. The growth of research in this area poses a fundamental question: which methodology to adopt in order to evaluate the performance of distributed networked control systems, producing reliable and accurate results? Several studies rely on small-case test-beds, e.g. [13]. However, experimental studies, in order to be meaningful, should consider many UAVs at the same time, and this might easily introduce excessive costs or present safety problems. Similarly, analytical models might likely become infeasible due to the large number of parameters to take into account, and the unknown correlations among them. Vice versa, simulation tools can provide a cost-effective solution in order to model the UAV applications before their effective deployment on a real scenario. However, although there are several tools enabling to model flight control [14][15] or network protocols [16][17], no software addresses the issues of both the fields at the same time.

In this paper, we fill such gap by proposing a novel simulation framework for networked control system, called CommUnicationS-Control distribUted Simulator (CUSCUS). Differently from the state of the art, CUSCUS allows simulating both the UAV networking and formation phases, via the integration of two existing tools: the Framework Libre AIR (FL-AIR) simulator [18] and the mainstream network simulator NS-3 [19]. Using FL-AIR, a real-time and fine-grained simulation of the micro-mobility of each UAV can be achieved, including the modeling of virtual sensors/actuators, the PID regulations and the drone stability. Moreover, it is possible to create UAV applications and test them on a simulated control environment before the actual deployment, since the same code can also be plugged in real drones. More specifically, we provide three main research contributions in this paper:

- First, we describe how to integrate the FL-AIR and NS-3 simulation with a closed-loop control, so that the fleet mobility is influenced by the propagation conditions and networking protocols. As a result, we are able to perform real-time accurate simulations of the wireless communication among the UAVs, and analyze the impact of the propagation phenomena on the algorithms used for fleet control;

- Second, we add a Scenario Module in both FL-AIR and NS-3, in order to make a step towards the usage of fleets of UAVs in Smart city scenarios. The Scenario Module allows modeling realistic 3D environments, by importing the scenario description directly from OpenStreetMaps and by taking into account the location of buildings and the street topology;

- Third, we demonstrate the capabilities of the CUSCUS framework on two use-cases, corresponding to well-known application scenarios of UAVs, i.e. dynamic formation control [20], and *Static Coverage* of the target area [21][22]. More specifically, we implement a reference algorithm for each use-case in CUSCUS, and we show the impact of micro-mobility control parameters, beaconing frequency, propagation conditions and scenario characteristics, on the application performance. Furthermore, as in CUSCUS it is possible to define the underlying control model, we take into consideration different physical parameters of the drone, such as the length of its arms and its total weight, which are information of utmost importance when it comes to define accurate movement dynamics.

Finally, we show by experimental results the scalability of the CUSCUS framework in terms of resource utilization (e.g. CPU and memory), and its fine-grained ability to model complex UAVs dynamics, characterized by the interplay between network-side configuration, control-side configuration and 3-D scenario characteristics.

The rest of the paper is structured as follows. Section 2 reviews the state-of-art of simulation tools for UAVs. Section 3 illustrates the CUSCUS framework, describing the logical architectures, the three main components (i.e. FL-AIR,

NS-3 and the Scenario Module), and their interworking. Section 4 introduces the use-cases and the mobility algorithms implemented in CUSCUS. Section 5 shows the performance of the CUSCUS framework, and demonstrates the modeling capabilities on the use-case previously mentioned. Finally, conclusions follow in Section 6.

2. Related Works

While pertaining to robotic research, our main purpose in this work is to give the possibility to simulate networked control algorithms on UAVs. From a broader point of view, the literature includes examples of simulation suites that attempt to integrate objectives that belong to the robotics research field along with objectives of other research domains. However, at the best of our knowledge, all the existing solutions lack the ability to simulate UAV flight models.

Historically, the field of integrated networking and robotics simulation have its roots into traffic simulation and analysis for V-2-X networking. The most complete example of this technology is the Simulation of Urban MObility (*SUMO*) simulator. This software couples with a network simulator and is capable to accurately simulate the movements of road vehicles and their communication. Also, it can be integrated with OMNeT++ through the *Veins framework* [23] and with NS-3 through the *iTETRIS* Control System *(iCS)* [24]. At the same time, these solutions are mainly tailored to 2D road-vehicle communications, while we would like to leverage a new architecture that takes into account the increased complexity of the simulation of a three-dimensional model.

The only example of this latter category is composed by the software that leverages the ARGoS simulator suite [25]. ARGoS is a robot simulator aimed at the rendition of large-scale multi-robot system. Its main features revolve around its scalability. Being conceived with parallelism in mind, the simulation suite is built to support multiple simulation engines at the same time. Its advanced modularity allows for the implementation of a plethora of add-on modules for the

most diverse applications. However, one of the main drawbacks of the simulator is that it does not provide suitable models for UAV flight.

The set of features displayed by ARGoS has been exploited RoboNetSim [26], which constitues the only solution that tries to merge the issues of networking and robotics fields. The authors of RoboNetSim integrate ARGoS with two different networking simulators: NS-2 and NS-3. In RoboNetSim the schedulers of ARGoS and of the network simulators are tightly coupled and the suite is optimized for the simulation of networked swarm robotics. This means that the level of detail for the robotic part is not suited for UAV simulation.

One of the most important features we would like to have in our tool is an enhanced level of detail for the robotic part, which should implement a flight and control model for each simulated UAV. For these reasons, we have decided to survey existing solutions on the subject of *control* and *networking* simulators for UAVs, and create our proposal out of those solutions that show an enhanced level of detail on UAV control and flight models and an easy integration and interoperability.

2.1. UAV Simulators

In literature, the existing works concerning the simulation of UAV fleets can be grouped into two sets: pilot training software and robotic behavior simulators.

For what concerns the first group, the number of existing works is increasing quickly because of their purpose as pilot-training software suites like those in [14, 15, 27]. These works fall into the category of classical *flight simulators*, which include realistic flight models and scenery rendering. As their main use is to training pilots, the software is intrinsically not capable to accurately and correctly simulate autonomous flight, as its main purpose is human interaction. Unfortunately, for the cases where the autonomous flight is implemented (i.e. the *opposing forces* in gaming flight simulators), the software distribution is commercial and closed-source. In this case, it is almost impossible to adapt them to simulate mission-oriented UAV fleets.

For what concerns robotic-oriented simulators, the availability of software, often released as open-source and well-documented, is higher. Some notable examples are the Gazebo [28], Morse [29] and FL-AIR [18] simulators. Both of them rely on the middleware provided by the Robot Operating System (ROS) [30]. However, in this last case, when the possibility of integration with real-time simulation or the emulation/simulation of UAVs control laws is considered, the mileage may vary. All the mentioned tools are indeed capable of simulating fleets of multiple robots, but FL-AIR provide the best rendition of control laws applied to the single engines of a quad/octa-rotor and Morse is not capable to interface with popular flight controllers like Pixhawk [31] or APM [32].

In choosing the most suited tool to simulate the behavior of a fleet of UAVs, the choice has been restricted to two candidates: Gazebo and FL-AIR, as they are capable to accurately simulate the control laws of the flight model of the UAVs at an extreme level of detail, for example the *flight envelope* can be calculated from the torque applied to each rotor of an UAV. For our final choice, we took into consideration that, using FL-AIR, the transition from simulation to real experiments and prototypes can be done seamlessly, as better detailed in section 3.1.1.

2.2. Network Simulators

In the era of Cloud Computing and of extensive virtualization of networks, the possibilities for what concerns network simulation are endless. Nowadays, any major network simulation software can be classified either in the category of commercial or open-source software. Among the former category, the most important commercial solutions that have been thoroughly exploited in scientific literature are the OPNET [33] and the *Qual-Net* [34] suites.

The conception of our proposal poses, however, a series of constraints on the network simulation part, the most important of which is the capability of arbitrarily extending and modifying the software base. For this reason, we have decided to look for a viable candidate within this last category. Some simulators, such as GloMoSim [35], NS-2 [17] and JiST/SWANS [36], are no longer under

active development. The project that are, as of 2017, still maintained and with an adequately large community are: WSNET [37], OMNeT++ [16] (the open-source version of OPNET) and NS-3 [38]. Focusing again on our specific needs, we realized that, from the network part of our system we would need:

- the ability to simulate network operation at a packet-level detail;
- the ability to interface directly with the operating system and, consequently, emulate a complex network in real-time
- the capability to seamlessly configure all the level of the ISO/OSI stack;
- a strong literature background;

While a combination of the above-mentioned features is to be found in many simulation software suites, only the NS-3 tool supports all of them, and for this reason it has been included in the CUSCUS architecture, as better detailed in the Section below.

3. The CUSCUS Platform

This Section is devoted to the presentation of the CUSCUS platform: we will first introduce the logical architecture and present separately the two main simulators, FL-AIR and NS-3, on which we have built our tool, and then we will show how we put these two blocks to interwork towards the first integrated control-network simulator specific for fleets of UAVs.

3.1. The Logical Architecture

In this Section, we describe the logical architecture and the main components of the CUSCUS framework. Compared to the simulation tools previously mentioned in Section 2, our software provides the following key novelties:

- thanks to the co-simulation environment, it allows modeling the impact of networking dynamics as well as of network protocol operations on the fleet mobility and the UAV formation control, and vice-versa;

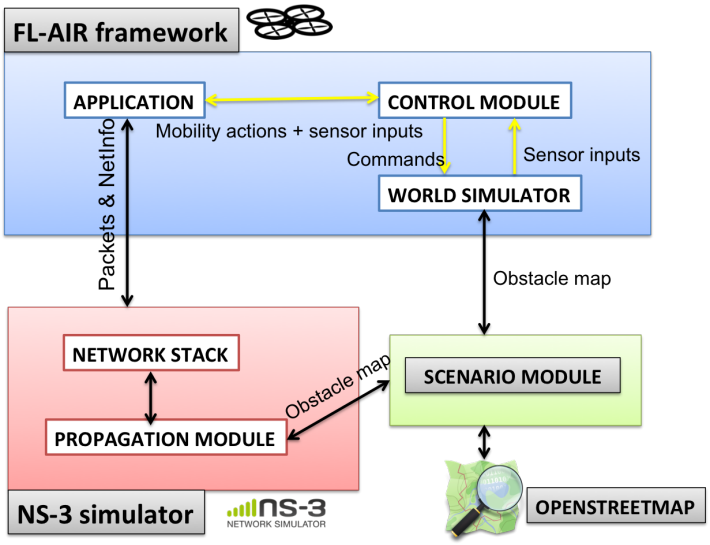

Figure 1: Overview of the CUSCUS logical architecture.

- thanks to the scenario Module, it allows modeling realistic 3D scenarios, with street topologies and building positions imported by the popular OpenStreetMap website;
- finally, thanks to the emulation capabilities of FL-AIR, and to the integrated platform, it allows the testing and pre-deployment analysis of real UAV applications on virtual 3D environments, reproducing the same characteristics of the real scenarios, and the same run-time dynamics. This feature might produce significant advantages in terms of safety and cost/time reduction compared to the experimental test-beds, beside guaranteeing a much wider range of configurations for the testing and the analysis.

Figure 1 shows the logical architecture of the CUSCUS framework, composed by the two simulators mentioned above: the FL-AIR framework, the NS-3 simulator as well as by the Scenario Module. This latter is in charge of loading the real scenario map, and works inbetween the other two components. In FL-AIR, the user can write the C++ code of the UAV Application (e.g. video-surveillance); mobility actions are sent to the Control component, which

is in charge of implementing and translating them into proper commands to send to each motor of the UAV. Based on them, the position of the UAV within the virtual scenario is updated by the World Simulator, which is a component of FL-AIR. Moreover, both FL-AIR applications and the Control component can have access to sensors (cameras, ultrasonic sensors, etc), whose input is provided by the World Simulator, based on the characteristics of the scenario, indicated by the Scenario Module. All the network packets produced by FL-AIR application are routed through the network stack in NS-3, and then transmitted on the simulated wireless channel. To this aim, we developed a novel propagation module which takes into account the signal attenuation caused by obstacles, again based on the information provided by the Scenario Module.
In the following, we detail the functionalities offered by each component, and the connections among them.

3.1.1. FL-AIR

The FL-AIR suite [18] has been developed by the engineer team at the Heudiasyc Laboratory, to facilitate the implementation and testing of algorithms and applications for UAVs.

The components of FL-AIR are a set of independent applications running on the host operating system (Host OS) that communicate using sockets. The components of a running instance of FL-AIR are the *Ground Control Station* (GCS), the *World Simulator* (WS) and *n UAV programs*. Figure 2 summarizes FL-AIR's internal functioning in the case of two simulated UAVs and a *remote controller* interface. In the Figure, the blocks represent each composing applications, that are connected through UDT sockets [39] and local interfaces (Local IF):

- The GCS is the main monitoring and multiplexing block, whose purpose is to act as a routing point for the other components. It displays the information and implements the commands coming from the other blocks;
- The WS is the main rendering engine. It manages the 3D environment,

multiplexes commands to the drones, can accept input from a localization system (such as the Optitrack system [40]) and displays the values measured by the sensors;

- The UAV programs not only represent each drone but are also responsible to make the calculations for the actual flight model, to implement the control laws and the high-level functions responsible for navigation and mission control.

In FL-AIR there is also the possibility to use some interfaces that could support a set of remote controls (joypads and joysticks). These are attached to the GCS and use the network to remotely pilot the drones. The choice of shifting the control and navigation computation on specific applications for each drone lies in the consideration that, using this setup, a real drone could take the place of a simulated one: FL-AIR gives the possibility to cross-compile the same controller for the simulated drone with a real UAV system as a target. In this way a real UAV can communicate with the rest of the components and interact with them. The GCS and the WS are not designed to have any *intelligence*: all the processing is done by the UAVs. To achieve this flexibility, in FL-AIR, UDT sockets handle all the communications between components. This means that all the traffic, coming either from simulated or real drones, is routed through the machine that hosts the *Ground Control Station* software.

The classes interfaces to a specific set of sensor, actuators, state machines and security checks are all included in FL-AIR. Thus, the only difference between a simulated and a real UAV is the use of a specific implementation of them. A *shared memory* (provided by the operating system libraries) superstructure provides the links. In this way, the *World Simulator* can: (i) compute the UAV states through its discretized dynamical model [41] and (ii) update the UAV status in the 3D world. The model output handles collisions and keeps track of the positions, which are successively sent back to the simulated UAVs control models.

To provide a suitable engine for a set of virtual sensors that need to interact

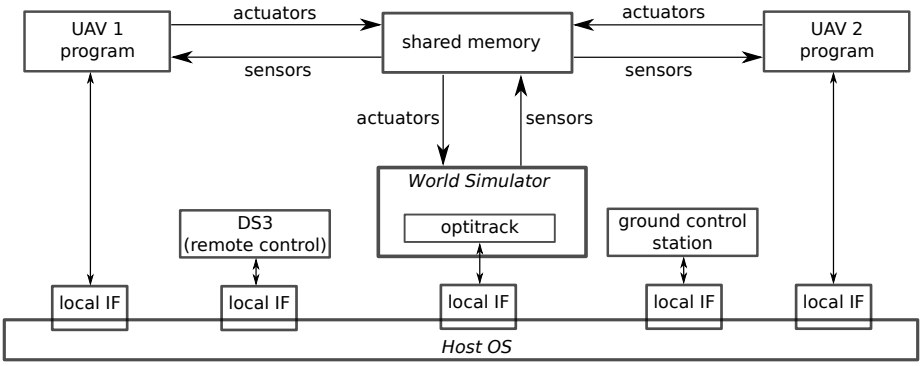

Figure 2: Overview of FL-AIR Simulator's architecture.

with the environment, FL-AIR uses the Irrlicht engine [42] (see Figure 3), which is capable to provide interfaces for: cameras, lidars, ultrasonic sensors and more.

It is possible to execute FL-AIR in both non-real-time and real-time mode. For the last option, it is possible to implement strict bounds and synchronization constraints on the dynamic model by using Xenomai [43], a real-time extension of the Linux kernel whose interfaces are present in FL-AIR.

3.1.2. Networking with NS-3

One of the most outstanding features of NS-3 is the capability to work in *real-time* mode, also called *emulation* mode [44]. In this mode, the simulator can interact with the network interfaces of the machine where is hosted. It can create virtual network interfaces on it and processes the packets sent through them. For instance, virtual nodes from inside the simulator could access the Internet via the host machine and vice versa. In our system, NS-3 creates a simulated environment and a set of virtual network interfaces. The communication between FL-AIR entities are then routed through it. The approach followed to achieve the inter-operation between the two software suites is based on a set of network tools provided by the Linux kernel: *TAP* devices, *Linux Containers* (LXCs) and network *bridges*. In our system, a set of LXCs, each containing a FL-AIR drone, make their networking interface available to the host and all the

Figure 3: Overview of FL-AIR 3D environment. Two virtual cameras are displayed on the right.

traffic coming from or going to the container is routed through a virtual NS-3 via the TAP devices.

TAP device. A TAP device is a virtual network device. All the packets that travel through it are handled by a user-space software: from a operating system point of view, everything that is behind them is virtual.

LXCs. The *Linux Container* is a virtualisation method that grants the possibility to encapsulate and run a separated Linux system in a controlling host [45]. Without the need of a separate kernel, it creates a virtual machine with an environment closely resembling the host and provides the following advantages:

- sealed environment where to run the guest system;
- network interface sharing with the host system;
- ease of installation and minimal resource footprint.

It is important to note that, in fact, the processes executed on *LXC* are not subject to emulation and are executed directly on the host machine. The

container can be seen just as a replication of the host system. Thus, the performance degradation in using them is not at all comparable with the one of a virtual machine in the classical sense.

Bridge. A *network bridge* is a device that is used to unite two or more network segments. Under Linux, a bridge behaves transparently: it works like a network switch (L2) so that virtual and real devices can easily connect to it.

These elements can be exploited by the use of the *NS-3 Tap Bridge* module. The module is responsible of bridging different networks from inside NS-3. Using it, an external application, nominally using a *TAP device* on the host system, can connect to one of the network devices of a *ghost* node running on the simulator. All the traffic generated from and to the *TAP device* is intercepted and routed through the network device of the NS-3 node.

Network stack and Propagation Module. The packets routed by FL-AIR to NS-3 are then managed by the simulated network protocols, based on the stack configured by the user. To this aim, NS-3 offers a wide library of network protocols, from the PHY/MAC to the transmission layers. However, the currently available propagation models do not allow taking into account the attenuation caused by the presence of obstacles at a specific location, unless tuning the standard deviation of the shadowing distribution. However, the same shadowing random variable is used at all the locations of the scenario. For this reason, we implemented a novel Path Loss (PL) model, which takes into account the map of obstacles provided by the Scenario Module. More specifically, we compute the Path Loss between 3D locations i and j as follows:

$$\begin{aligned} PL(i,j) &= \alpha \cdot log(\frac{d_{ij}}{1000}) + \alpha \cdot log(f) + 92.45 + \\ &+ n_W \cdot \gamma_W + n_w \cdot \gamma_w + \Psi \end{aligned} \quad (1)$$

Here, d_{ij} is the 3D distance between the locations j and i, f is the transmitting frequency, n_W and n_w are respectively the number of outdoor/indoor walls traversed by the Line-of-Sight (LOS), with γ_W and γ_w the corresponding attenuation factor, and Ψ is a Gaussian variable with zero mean and variance

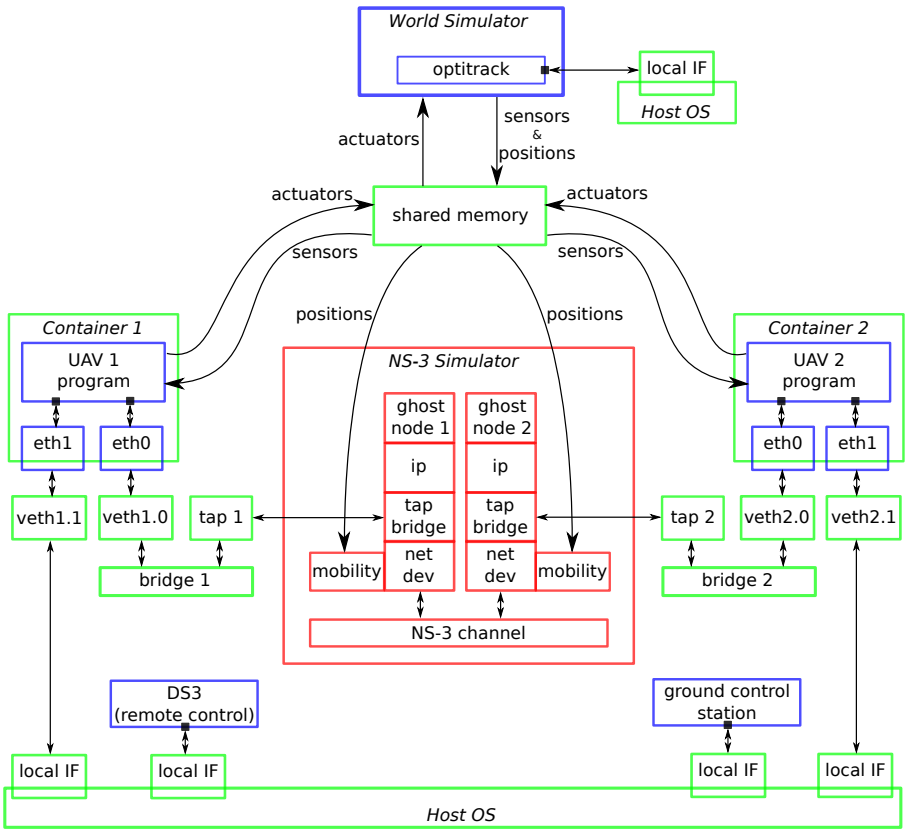

Figure 4: Overview of CUSCUS architecture

equal to χ. The tuning of γ_W, γ_w and χ is decided by the user, based on the characteristics of the scenario.

3.2. Implementation/Interworking

After the description of the composing elements, here we present the core component of CUSCUS, i.e. how it enables the connection and inter-operation of FL-AIR and NS-3.

In our proposal, FL-AIR and NS-3 will be run in parallel within an unified environment. After defining the simulation scenario to implement, the two components are setup according to it: the UAVs and their control logics are set

up in FL-AIR and their arrangement is replicated as a set of nodes and their connections in NS-3. For the drone simulation part, in CUSCUS, a set of FL-AIR UAV applications, as the ones described in section 3.1.1, will be run inside *LXC* containers: they will envelope the UAV application and intercept all of its network traffic. For the networking simulator part, a set of *TAP devices*, equal, in numbers, with the UAVs to simulate, will be created on the host machine. In this way each container, with one drone inside, can connect to the virtual NS-3 network. Again on the host machine, the set of *network bridges* will allow the connection between the *LXCs* and the *NS-3 TAP devices*. Among the various operating modes of the *Tap bridge NS-3* module, we have chosen to use the one that connects directly to a preconfigured *TAP device*: the *UseLocal* mode. In this way the configuration of the interlocks between FL-AIR and NS-3 is completely external to the two.

In Figure 4, the main architecture of the CUSCUS framework is depicted.

From the figure it is possible to identify the former structure of Figure 2. The UAV programs are enclosed in containers with multiple interfaces. It is still present the former UDT socket structure where the *GCS* is directly connected to the UAVs and the remote controls through the local interfaces. It is present altogether a new set of communication links: using another interface (eth0 in the figure), an UAV can offload its packets to NS-3 using the *TAP-bridge-LXC* chain. To further increase the level of abstraction, it is possible to route all the traffic through NS-3, including the one with the GCS and the remote controls, allowing to have a completely customizable environment. In our proposal, as the containers will run on the same host machine, they use the *shared memory* to exchange data with the world simulator and with NS-3. In the system there are always two running abstractions of each simulated UAV: the FL-AIR UAV in its container is mirrored in NS-3 ghost node. The former is responsible for the simulation of the movement and the interaction with the physical environment; the latter is responsible of network operation. Each position update in FL-AIR is written in NS-3 using the shared memory and a closed loop is maintained. While a UAV moves in FL-AIR, its position is mirrored in the NS-3 world and

the communication is modified accordingly.

4. Use Cases

To demonstrate the validity and the effectiveness of CUSCUS, we have chosen to showcase its features by analyzing its behavior in two relevant use cases. The two use cases implemented in CUSCUS are: (*i*) a UAV dynamic formation control that leverages *Corrective Consensus*, and (*ii*) the implementation of a *Static Coverage* algorithm. Both of them represent important scientific challenges for the worlds of networking and control. In the first use case a networked fleet of UAVs is required to follow a leader, while maintaining an arbitrary formation, by running a control model that requires the exchange of messages. In the *Static Coverage* case, a fleet of sensor-equipped UAVs is required to cover a target area, while maintaining the network connectivity in order to transmit the collected data to a central server. We remark here that the goal of the discussion is not to introduce novel solutions for UAV fleet management. Instead, we aim at demonstrating the capabilities of CUSCUS on (*i*) modeling realistic UAV scenarios, (*ii*) supporting communication-aware algorithms and (*iii*) capturing the impact of UAV characteristics, propagation conditions and micro-mobility on the performance of the algorithms described below.

4.1. Dynamic Formation Control

The first use case refers to the implementation of dynamic formation control in fleets of flying robots. For this last class of systems, a cooperative broadcast-based packet-loss-tolerant algorithm is introduced in [46] and [20]. The scheme is designed to effectively reduce link losses by introducing a negligible level of redundancy in a leader-follower scenario coupled with formation control. In the paper, the authors consider a WNR composed of $n+1$ robots with n followers and one leader. A graph $G = (V, E)$ is used to denote the network topology among $n+1$ vehicles with $E \subset V \times V$ as the edge set and $V = \{1, ..., n+1\}$ as the node set. $F = \{1, ..., n\}$ denotes the set of *followers*. The authors

consider a discrete-time second-order model [47] to describe the dynamic of leader/followers, namely:

$$v_i(t+1) = v_i(t) + T \cdot \begin{cases} u_i(t, x_i, v_i, x_j, v_j), \begin{cases} i \in V \\ j \in V \end{cases} \\ f(t), \qquad\qquad i = n+1 \end{cases} \tag{2}$$

$$x_i(t+1) = x_i(t) + T \cdot v_i(t) \tag{3}$$

where $x_i(t)$, $v_i(t) \in \mathbb{R}^2$, $u_i(t, x_i, v_i, x_j, v_j) : [0, +\infty[\times \mathbb{R}^{2(n+1)} \to \mathbb{R}^2$ are, respectively, the position, velocity and control input[1] associated with the $i-$th vehicle. $f(t) : [0, +\infty[\to \mathbb{R}^2$ is a signal describing the leader acceleration and T is the step-size. Their objective is to design $u_i(t)$ such that the robots follows the leader and, at the same time, maintain a desired formation. Assuming the related matrix $\overline{D} = [\overline{D}_{ij}]$ being a skew-symmetric matrix, let $\overline{D}_{ij} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$ the desired Euclidean distance between vehicle i and vehicle j. Following this definition, at each *follower* is applied the following *consensus-based* control law [48] to the robot motion:

$$\begin{aligned} & u_i(t) = u_{iF}(t) + u_{iL}(t) = \\ & = \sum_{j=1}^{n} a_{ij}(t)\big[\big(x_j(t) - x_i(t)\big) + \overline{D}_{ij} + \gamma\big(v_j(t) - v_i(t)\big)\big] + \\ & + a_{iL}(t)\big[\big(x_L(t) - x_i(t)\big) + \overline{D}_{iL} + \gamma\big(v_L(t) - v_i(t)\big)\big] \end{aligned} \tag{4}$$

Where the coupling strength is represented by $\gamma > 0$.

For the control law to operate, it is necessary that each robotic element knows the state of the others. Assuming full-duplex, instantaneous and perfect communications, an undirected edge $(j, i) \in E$ exists if vehicle i and vehicle j can access information from each other. It is also assumed that $(i, i) \notin E$. The adjacency matrix $A = [a_{ij}] \in \mathbb{R}^{(n+1)\times(n+1)}$ is defined as $a_{ij} = 1$ if $(j, i) \in E$

[1] in the following we use $u_i(t)$ as the acceleration, for the sake of simplicity of notation

and $a_{ij} = 0$ otherwise. The desired formation is asymptotically reached if $\| x_j(t) - x_i(t) \| \to \overline{D}_{ij}$ and $\| v_j(t) - v_i(t) \| \to v_L(t)\ \forall i, j$.

However, the above algorithm is not resilient to network disruption as packet losses would significantly reduce the accuracy of the control action. For these reasons, the authors proposed a Corrective Consensus approach where, starting from the already defined distributed rendezvous algorithm of equation (4), they defined a new set of variables $\phi_{ij}(t)$ with $(i, j) \in V^2$.

$$\begin{aligned} \phi_{ij}(t+1) =& \phi_{ij}(t) + a_{ij}(t)[(x_j(t) - x_i(t)) + \\ &+ \overline{D}_{ij} + \gamma(v_j(t) - v_i(t))] \end{aligned} \tag{5}$$

$$\phi_{ij}(0) = 0. \tag{6}$$

For each node i, the auxiliary variables $\phi_{ij}(t)$ represent the amount of change that node i has made to its state variables $x_i(t)$ and $v_i(t)$ due to neighbor j. To update the $\phi_{ij}(t)$ according to (5) and (6) do not need any additional message exchange. If robot i and j always take the same action, then the changes they make should be symmetric, i.e., $\phi_{ij}(t) =$ -$\phi_{ji}(t)$. From the last assumption it is possible defining a new set of variables:

$$\Delta_{ij}(t) = \phi_{ij}(t) + \phi_{ji}(t) \tag{7}$$

that represent the amount of *bias* (as the sum of the states) that a robot has accumulated on both directions of the (i, j) *link*. By minimizing the bias, the robots reduce the error in a distributed and iterative manner: each robot i corrects its own state value. Specifically, node i collects $\phi_{ij}(t)$ $(j \in N_i)$ from its neighbors to calculate the $\Delta_{ij}(t)$. Then node i adjusts its control input u_i using the $\Delta_{ij}(t)$'s. After this iteration, the robots resume the *Standard Consensus* shown in (4) while periodically performing the *corrective* step described above.

For the Corrective Consensus after every k consecutive Standard Consensus iterations, the corrective one takes place using the following control input:

$$u_i(k) = -\frac{1}{2} \sum_{j=1}^{n+1} \Delta_{ij}(k), \quad i = 1, \ldots, n \tag{8}$$

The auxiliary variables, instead, after k consecutive standard iterations update their value according to:

$$\phi_{ij}(k+1) = \phi_{ij}(k) - \frac{1}{2}\Delta_{ij}(k), \quad i,j = 1,\ldots,n+1 \tag{9}$$

In their paper, the authors proposed a practical implementation of the message exchange mentioned earlier.

Their simulative approach involved simulation on the NS-3 platform with a custom-built two-dimensional mobility model and a network implementation based on IEEE 802.11g. In this paper we implement the Corrective Consensus of [46] in CUSCUS. In this way, we will demonstrate the capability of our proposal to simulate a three-dimensional control model tailored for a Wireless Networked Robot system based on UAVs, and the impact of micro-mobility parameters on the overall fleet mobility.

4.2. Static Coverage

The second use case refers to a well-known networking problem, i.e. the *Static Coverage* (SC) of a target area from a group of UAVs. From video-surveillance to emergency communications [49], several different applications, especially in the context of Smart cities, might benefit from SC functionalities [7]. In its general formulation, the SC problem can be defined as follows. Let S be the set of available UAVs, with $|S| = n$, moving on a 2D plane. Each UAV is equipped with a short-range radio interface, through which it can communicate with all the neighbors in a transmitting range equal to R_t; moreover, it is provided with sensors through which it can gather location-aware information from the environment. Let A_s be the sensing area, assumed circular and equal for all the UAVs. The goal is to determine the Euclidean graph $G(V, E)$ such that: (i) each vertex represents the location of a UAV on the target scenario, i.e. $V = S$; (ii) there exists an edge $e(i, j) \in E$ iff the UAV i and j are in their reciprocal wireless communication ranges; (iii) G is a connected graph, which means that the aerial network is not partitioned and (iv) the total area A^* covered by the aerial network is maximized, i.e. $A^* = max(\bigcup_{i=0}^{n} A_s)$. Under

ideal assumptions about homogeneity of propagation conditions, transmitting ranges and sensing areas among the UAVs, the optimal SC can be determined by placing the UAV in regular patterns [50]. However, in a real world scenario, the assumption of homogeneity might not hold since the wireless links can likely experience different propagation conditions caused by multipath fading and by the attenuation due to obstacles. Moreover, in some applications like disaster recovery, the UAVs might not know the characteristics of the environment in advance, hence they should be able to self-place in order to maximize the coverage of the aerial network, while still keeping the connectivity with at least one other node. To address such goals, several distributed mobility schemes have been proposed, including also communication-aware approaches [9][10]. In this paper, we focus on the STEM-NET algorithm [22][21], which has been implemented within the CUSCUS framework; the evaluation results are presented in Section 5. The STEM-NET algorithm extends the virtual spring approach in [51], by considering emergency scenarios where the aerial network is used as a backup communication infrastructure connecting isolated users' devices on the ground. Each wireless link between two UAVs (e.g. UAV i and j) is modeled as a virtual spring force, acting according to the Hooke's law:

$$\vec{F_{i,j}} = -K \cdot (\vec{x} - l_0) \tag{10}$$

Here, $\vec{x}$ denotes the spring displacement, l_0 its natural length and K the stiffness constant. On each UAV i, multiple forces $\vec{F_{i,1}}$, $\vec{F_{i,2}}$ ... $\vec{F_{i,d}}$ might apply at each instant, based on the current degree of the node, denoted as d. At fixed intervals, UAV i computes the resultant force $\vec{F_i} = \sum_{j=1}^{d} F_{i,j} + \sum_{j=k}^{d} F^R_{i,O_j}$, and moves accordingly toward the direction of $\vec{F_i}$, with constant speed. In the Equation above, $F^R(i, O_j)$ are purely repulsive forces modeling the path clearance, and acting between each UAV i and obstacle O_j. Without loss of generality, we assume that the intensity of $F^R(i, O_j)$ is made proportional to the current distance between the UAV and the obstacle, i.e. $F^R(i, O_j) = \epsilon \cdot dist(i, O_j)$, and applies only for the obstacles in the visibility range (i.e. only if $d(i, O_j) > \kappa$, where κ is the visibility threshold).

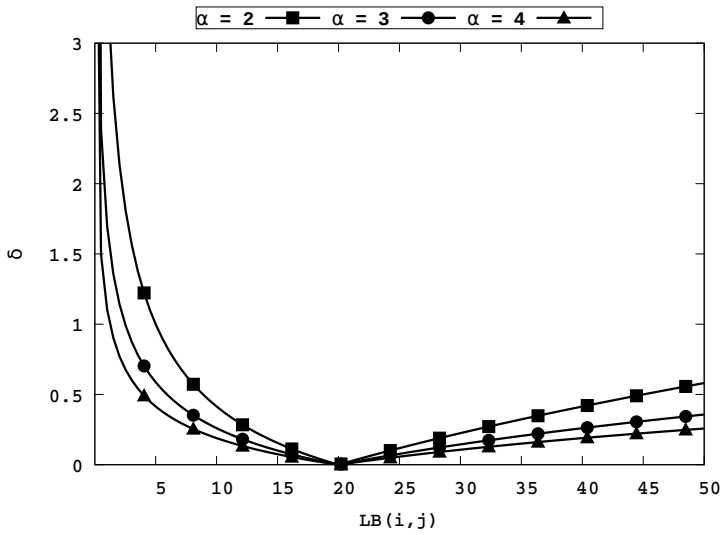

Figure 5: The virtual spring displacement as a function of $LB(i,j)$, with LB_{req}=20 dbm.

Differently from other virtual spring approaches, STEM-NET achieves communication-awareness by relating the formulation of the spring parameters (i.e. $\vec{x}$, l_0 and K) to the current propagation conditions of each link. More specifically, we assume that every T_B seconds, each UAV transmits a $BEACON$ message containing its identifier. Based on the Received Signal Strength (RSS), each UAV computes the Link Budget of the link $i \leftrightarrow j$ (i.e. $LB(i,j)$) as follows:

$$LB(i,j) = Pr_j^i - RS_{thr}^i \tag{11}$$

where RS_{thr}^i is the reception threshold. The LB metric, depending on the fading margin set in the design phase, can be considered a proxy of the communication reliability (i.e. it tells when the link is going to break), as well as of the the maximum achievable rate. The requested link budget (LB_{req}) models the Quality of Service (QoS) which must be guaranteed on each link of the aerial network. Based on it, the spring displacement $\delta = (\vec{x} - l_0)$ is formulated as the current error between the requested and current LB on the $i-j$ link, i.e.:

$$\delta = \sqrt[\alpha]{\frac{max(LB(i,j), LB_{req})}{min(LB(i,j), LB_{req})}} - 1 \tag{12}$$

Here, α is the propagation decay exponent; more details about the Equation above can be found in [22]. Figure 5 shows the δ values, as a function of $LB(i,j)$ and for different values of α; LB_{req} is set to 20dBm. It is easy to notice that: (i) δ is 0 when $LB(i,j)$ is equal to LB_{req}, (ii) δ increases when $LB(i,j) > LB_{req}$, reflecting that the two spring end-points are too close, (iii) δ increases more quickly when $LB(i,j) << LB_{req}$, since the two nodes are moving out of the communication range. The second component of Equation 10, i.e. the stiffness constant K, defines how quickly the UAVs should react to an increase/decrease of the current displacement. In [22][21], the value of K is made parametric based on the link type (e.g. air-to-ground link vs air-to-air). Here, we consider a simplified formulation where the value of K is assumed constant; the impact of such parameter is explored in Section 5.

5. Performance Evaluation

We performed a four-folded simulation campaign in order to display the features of CUSCUS and evaluate the feasibility of its deployment. The first campaign aims at showing the impact of CUSCUS on its host system. The second campaign aims at evaluating the simulator's ability in integrating accurate control models. The capability to incorporate real-world UAV parameters into network-oriented simulations is the object of the third simulation campaign. The last campaign summarizes the simulator capabilities by providing the maximum level of simulative integration between networking, control and physical scenario rendering.

For the experiments, we have used a Dell XPS 8500 workstation with an Intel® Core™i7-3770 CPU @ 3.40 GHz and 16 GB of RAM memory. The entire CUSCUS, comprised of NS-3 and FL-AIR runs on this host machine. From the NS-3 perspective, we imagine the UAVs being equipped with a single IEEE 802.11 interface. We make it render, for each node, IEEE 802.11g Physical and MAC layers. We use the stock NS-3 simulated devices with the radio parameters taken from the market [52]. As we suppose the nodes being equipped with a

single WiFi interface, we have chosen to employ a Ricean fading model, as for UAVs the signal on the LOS path is much stronger than the one on indirect paths. We assume that there is no packet fragmentation and that the nodes stay always in each others' connection range.
To ease the prototyping of the simulated scenarios we decided to use the non-real-time version of FL-AIR.

5.1. Scalability analysis

First, we performed a set of preliminary experiments in order to evaluate the feasibility of CUSCUS deployment and to have a glimpse on its scalability.

In order to verify its scalability as well as the impact of the delays introduced by the simulator architecture on the control systems, we have designed a simulation scenario that can highlight the potentialities of both FL-AIR and NS-3. Concerning the FL-AIR perspective, the scenario consists in a set of UAVs that start from fixed positions and circle around a fixed point located at the center of the scenario. The distributed control system employed at each UAV is fed with the UAV position and orientation from a simulated Optitrack system, and tries to maintain the same distance between all the UAVs while they circle. As the number of UAVs increases, the UAVs will arrange themselves at the vertexes of a polygon with an increasing number of sides. The UAVs keep a fixed altitude of 10 m for the whole simulation.

In our experiments the robots sample their position and broadcast it periodically, each T_b seconds. When the broadcasted information is received at the other nodes, it is relayed to FL-AIR and stored. In our results we distinguish between two definitions of delay: the *architectural* delay and the *network* delay. The former is the delay that the packets experience when they are routed through the CUSCUS architectural components. The latter is due instead to the simulated network behavior.

A first index that can be used to evaluate the performances of CUSCUS is represented by the physical resource used by the host during the simulation.

Figure 6 shows the percentages of CPU and RAM usage, when the number of UAVs increases. We can notice that the CPU utilization increases linearly with the number of simulated entities, and sub-linearly when considering the memory allocation; this constitutes an interesting property for a simulation tool. Hence, this result shows the *scalability* of CUSCUS, which is able to efficiently exploit the available resources. Although no bottleneck could be observed in the Figure, it is worth remarking that the system performance is bound by the hardware characteristics of the host machine. A straightforward approach in order to achieve scalability regardless of the number of simulated nodes and of the complexity of the scenario is to increase the number of available hosts, balancing the simulation load within the cluster (vertical scalability). The NS-3 simulator already provides the capability to execute a simulation on multiple processors, by splitting the simulation entities on multiple Logical Processes (LPs). Each LP can then be executed on a different CPU. Hence, the support for distributed simulation can be implemented in a straightforward way in CUSCUS, since each LP could correspond to a simulated UAV. We plan to further investigate such issue as future work. The second test that we have done in order to characterize our architecture is the analysis of the *architectural* delay introduced by CUSCUS in the *TAP-bridge-LXC* chain (see Section 3.2). This measure is important as it can estimate the fixed time delays introduced by the CUSCUS framework. The extension and stability of these overheads assume a high importance when CUSCUS is used to simulate control systems for UAVs. In Figure 7, we see the delay [μs] the packets experience in the path from FL-AIR to NS-3. The broadcast time T_b is set to $50ms$ for this experiment. We can notice from the Figure 7 that the *architectural* delay introduced is constant and it stands around $70\mu s$. This result shows that the delay introduced solely by CUSCUS is negligible and stable with respect to the UAVs number, i.e. the traffic generated by the simulation scenario does not influence the variation of the *architectural* delay. With the last experiment, we want to show the proposed architecture *suitability* for the study and analysis of distributed networked control system. We executed a series of tests, modifying the number of UAVs and the T_b value, in order to

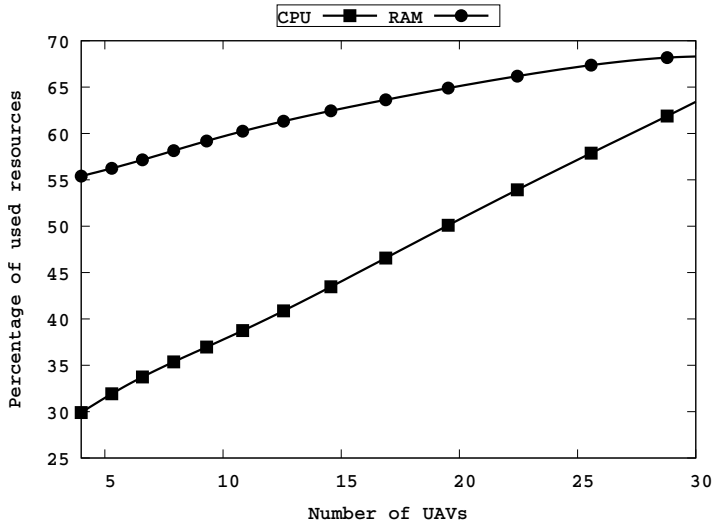

Figure 6: Resource Usage at the *host* varying the number of UAVs.

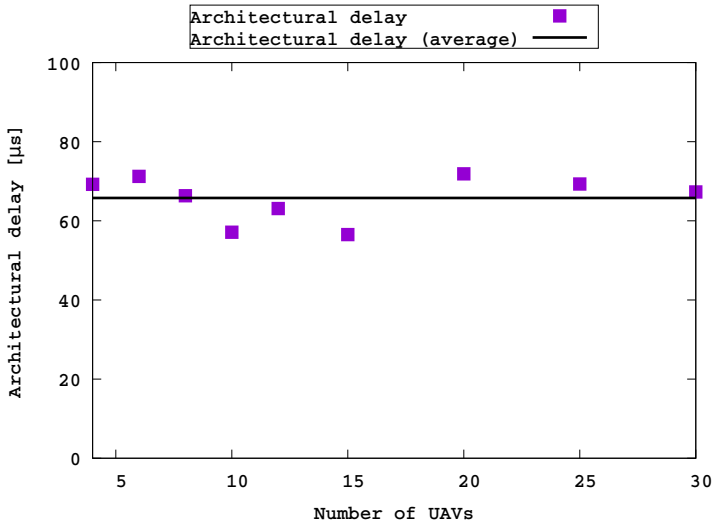

Figure 7: Architectural delay of CUSCUS varying the number of UAVs.

study the impact of using a simulated communication network for the exchange of control system messages. By using a modified version of the scenario used in the previous experiments, we measure the error introduced in the formation control inside FL-AIR due to *network* delays. In this case, the nodes will not retrieve the information about the position of the other UAVs from the Optitrack system; instead, they will use the positions their neighbors have broadcasted through the simulated NS-3 channel. This *error* $[m]$ is defined as the distance between the *actual position* for the simulated UAVs and their *reference position* in the formation. The *reference position* is the position that nodes should be at, when their control system does not use the network for exchanging position data, but only the *Optitrack* system. In this last case ($T_b = 0ms$ in the figures), the *error* is reduced to a minimum and it is due only to the control system itself. Instead, when the simulated UAVs receive the information concerning the position from their neighbors, there is a *delay* introduced by the network. Figure 8 shows this *error*, as the result of these tests. From these experiments, we can infer the following conclusions: (i) an increase of the T_b value strongly impacts the formation error, whereas (ii) the number of UAVs does not affect this error. The former conclusion comes from the fact that we are running a distributed control system that has to work with outdated information about the neighboring nodes' positions. The latter comes from the fact that, in this particular case, communications issues such as packets collisions and, hence, packet retransmissions, have a negligible impact on the wireless communication network. It is to remark that the collision probability is very small since the exchanged packets are also very small. They are composed by only the sender node's position and orientation. In conclusion we can state that the CUSCUS framework is able to execute reliable distributed networked control system simulations by keeping time overheads constant when the number of simulated UAVs varies. Hence, it enables the user to study and analyze practical instances of this kind of control systems.

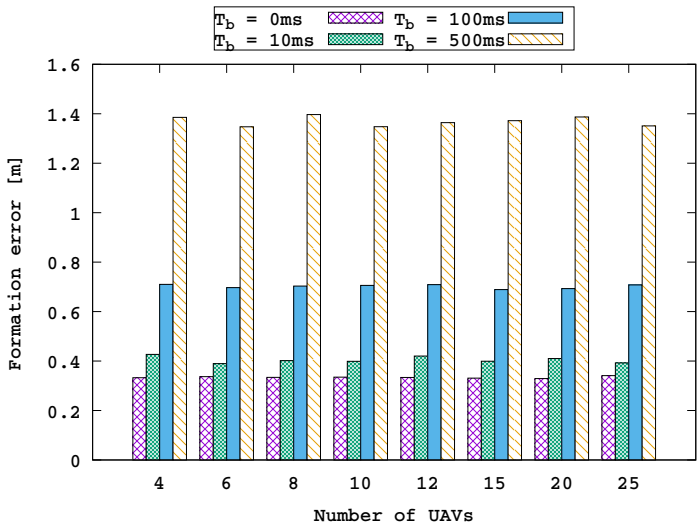

Figure 8: Formation Error varying the number of simulated UAVs and T_b.

5.2. Dynamic Formation Control scenario analysis

To demonstrate the capability of CUSCUS to integrate accurate control models, we replicated the simulation campaign of [46]. We have implemented the *Corrective Consensus* described in the paper as a CUSCUS application for UAVs, and we have set the simulator suite to replicate the network conditions proposed in the cited work. In this experiment, a fleet of 10 drones have to keep a circular formation of 20 meters radius and, at the same time, have to follow a *leader* UAV, whose autopilot is set to visit a set of predefined way-points. The formation is kept by using *Corrective Consensus*: the algorithm of the reference paper is implemented as a working application and the packets produced by the robotic nodes are handled by the simulated NS-3 network. We have used the same LOS propagation model and the same periodic reductions of connectivity to simulate jamming. In the original paper, independently of the propagation model, the connectivity was periodically reduced according to the concept of *Network Disruption*: the percentage of time, over a reference interval, where the communications were completely jammed. In this paper, we have called

this parameter γ and we have tried to replicate the results by using the highest reported values, as, according to the authors, for undisturbed communications there were negligible performance variations.

In Figures 9 and 10 is displayed the evolution of the positioning error between the UAVs simulated position and their reference one, according to the Consensus algorithm and averaged among all the drones. We have varied the control algorithm parameters according to the reference paper. We have set up the Corrective Action periodicity of eq (9) as the parameter ρ: 500, 1000 and 2000 ms.

The Figures reflect the reference behavior.

For low level of γ, there is a negligible difference in performance between the non-corrected and the corrected version of Consensus. In Figure 9, it is shown the same behavior of the reference, with all the cases converging to the same error value. Interesting details can be reported: a faster application of the corrective iteration increases the convergence time, as it can be seen in the figure from the curve with $\rho = 500ms$. This result is in accordance with the literature, as the Corrective Consensus applies always a strong input, even when the robots approach their target position.

Even if we focus on large values of γ, the resulting behavior is corresponding to the literature. A fast application of consensus improves the performance, as it can be seen from Figure 10. In the figure, a more frequent application of the corrective action yields an improvement of performance. For a value of $\rho = 500ms$, the overall formation error is reduced and, as the value increases, the difference in positioning decreases, up to the case of $\rho = 2000ms$, where the error is almost the same as the uncorrected version.

These results give already an interesting insight about the Corrective Consensus. The gap between the non-corrected version and the corrected one is smaller using a more complex and accurate flight model for the drones and our results could be used by the original authors to refine their approach.

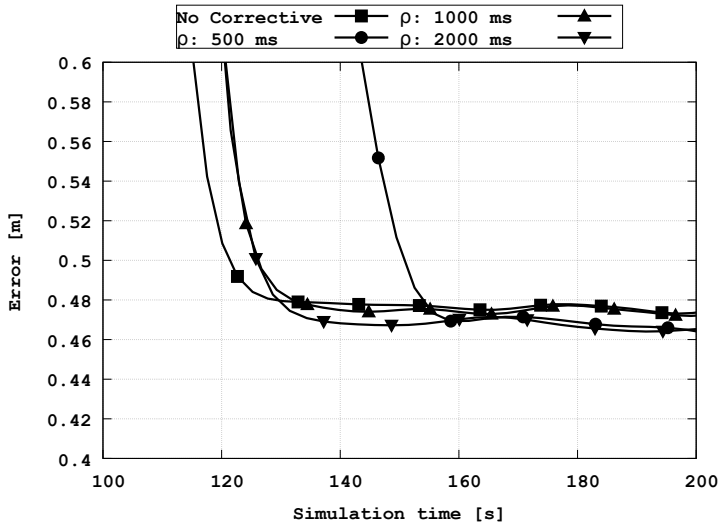

Figure 9: Evolution of the average formation error with a varying ρ and $\gamma = 40$

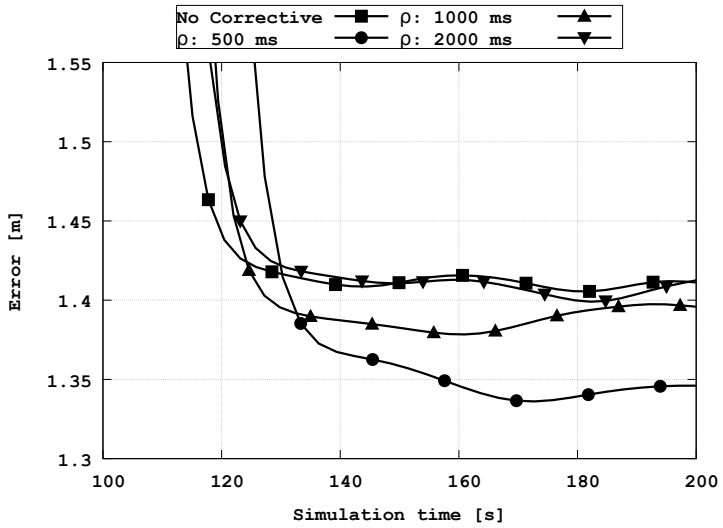

Figure 10: Evolution of the average formation error with a varying ρ and $\gamma = 70$

5.3. The UAV weight as a parameter

As stated beforehand, the peculiarity of CUSCUS is the capability to take into account, at the same time, specific networking and control aspects. To show this feature, we executed a set of experiments that took into consideration the weight of the single UAV in the same scenario of the previous Consensus campaign.

Specifically, we simulated the same Corrective Consensus-driven leader-follower behavior and we analyzed the evolution of the *formation error* according to the physical weight of the single drone.

In this simulation scenario, we have fixed the ρ at 500 ms and we have varied the value of γ between 0 (no jamming), 40 and 80. The drone weight, `DW` in the Figures, has been varied between 1, 1.5 and 2 kg.

Figures 11, 12 and 13 present the results of our campaign, by showing the evolution of the formation error averaged over all the follower UAVs.

In all of them, it is possible to see the same behavior: a fleet of heavier drones is faster to reach formation and display reduced, albeit slightly, positioning errors. This behavior is due to the fact that the positional instability produced by the constant application of a control input by the Corrective Consensus is mitigated by the physical inertia of the drone.

Figure 11 shows the results with a scenario where no jamming is applied. In this case the flight model of a lighter drone, having less inertia, is subject to constant movements that postpone the time where the fleet reaches formation. On the contrary, heavier drones are less influenced by the instability of Corrective Consensus.

By increasing the value of γ we have the behaviors of Figures 12 and 13. We continue to observe the same behavior: a fleet of heavier robots is faster in reaching formation. The same consideration applies if the value of the steady-state (i.e. the formation error obtained where the value does not decrease anymore) value is analyzed. In accord to the literature, other important differences are absent, except for the steady-state value for the error, if the value of γ is increased.

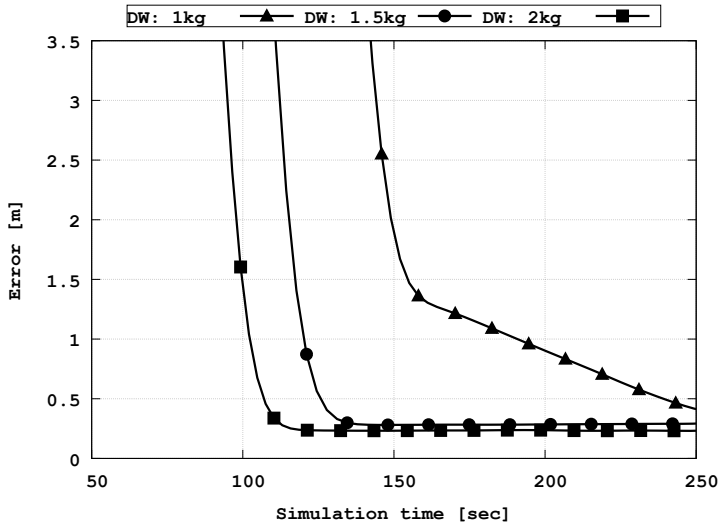

Figure 11: Evolution of the average formation error with a varying drone weight and $\gamma = 0$

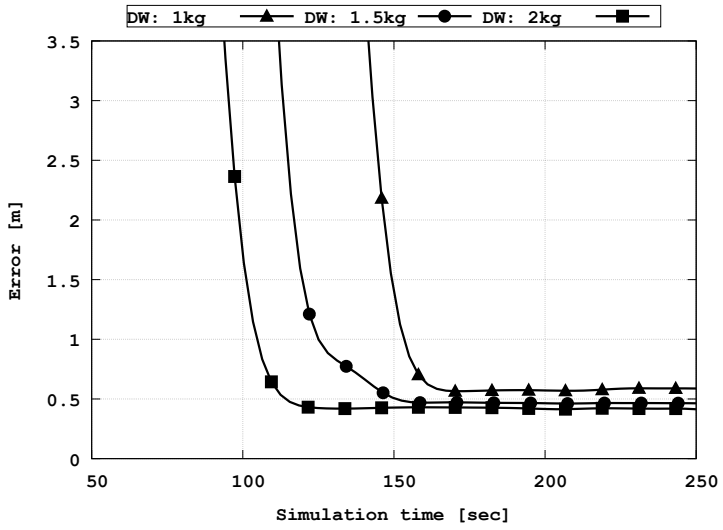

Figure 12: Evolution of the average formation error with a varying drone weight and $\gamma = 40$

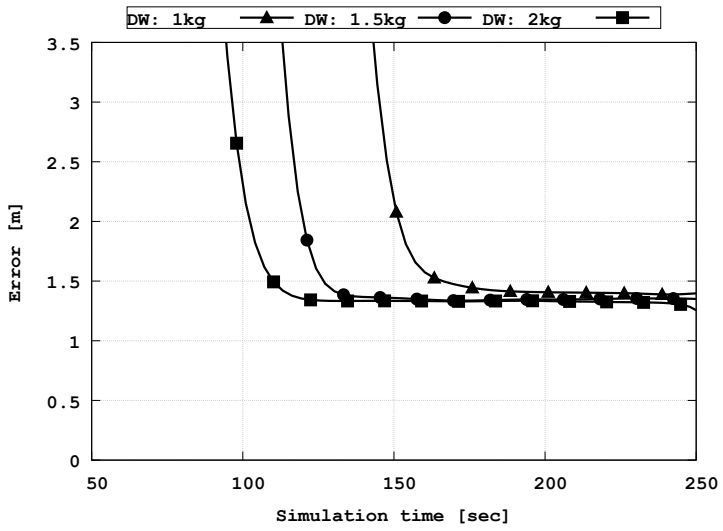

Figure 13: Evolution of the average formation error with a varying drone weight and $\gamma = 80$

5.4. Coverage scenario analysis

We now analyse the problem of *Static Coverage* (SC) by using the distributed method described in Section 4.2. For this case-study, we used two different metrics to evaluate the algorithm, i.e.:

- *Coverage*: defined as the total area covered by at least one UAV.
- *Stability Index (SI)*: defined as the average velocity held by the fleet of UAVs. This index is a proxy for the system stability, since the SC problem -by definition- requires a static placement of the UAVs within the scenario.

We used the *OpenStreetMap* import function that allows CUSCUS to import a real map inside the *Scenario Module*. More specifically, we used two different maps: (*i*) the historical center of Bologna (Italy) that is characterized by narrow streets with small and irregular buildings and (*ii*) a slice of the Manhattan borough (New York) that is characterized by large streets with huge and regular buildings. We simulated a fleet of 10 UAVs having the radius of the sensing area A_s equal to $40m$. The requested link budget (LB_{req} of Equation 12) is set to $40dBm$.

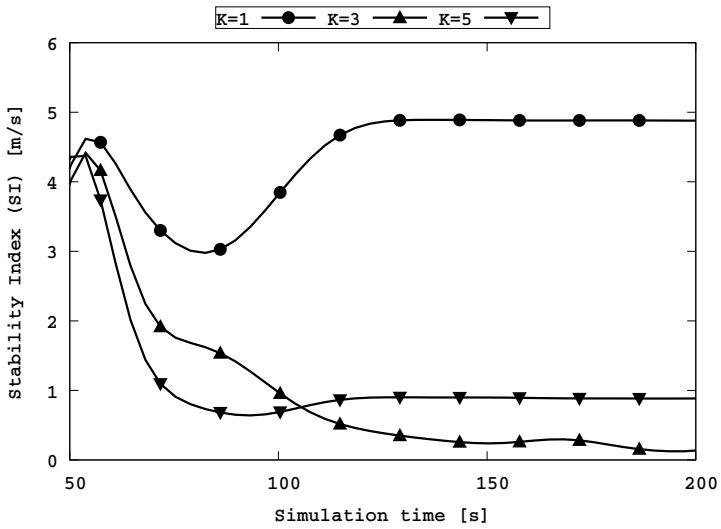

Figure 14: The Stability Index for different values of K.

First, we show the capabilities of the virtual spring force algorithm for the *Static Coverage* problem. For this reason we initially set the flight altitude above the rooftop of the buildings. As in the previous Section, we want to show the CUSCUS ability to model in detail the characteristics of a UAV, as for example the weight of the vehicle, and the impact on the application performance.
Figure 14 shows the SI metric varying the virtual spring stiffness K, defined in Equation 10. The stiffness defines how strong the virtual force is, i.e. the amount of force that the spring generates if it is elongated more than its natural length $l_0 = LB_{ref}$ (see Section 4.2). We can notice that, for a low value of stiffness, the system becomes unstable. This is because the low responsiveness of the virtual spring does not allow the UAVs to stop at fixed locations. We can also notice that a too high value of stiffness ($K = 5$ in Figure 14) results in a bigger error with respect to a medium stiffness value ($K = 3$ in Figure 14).

In Figure 15 we show the impact of the weight of the UAVs (DW) on the stability of the virtual spring algorithm. Here we used $K = 5$, and hence an high responsiveness to the position error, since heavier UAVs need more time to update their positions, and hence experiment less oscillations. We can see from

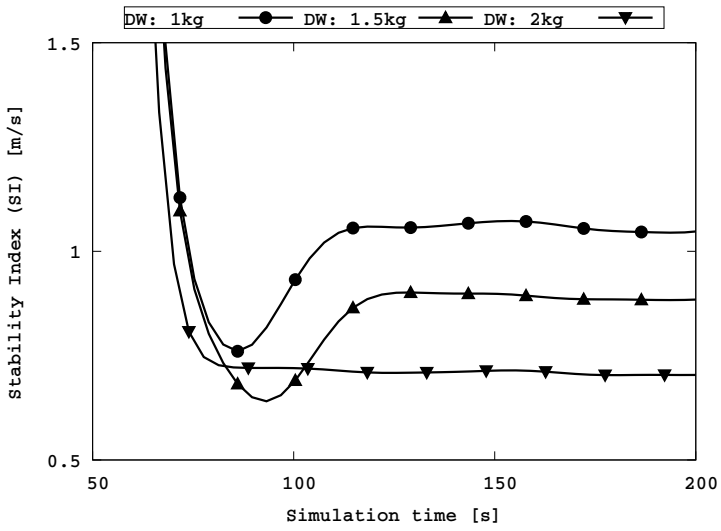

Figure 15: The Stability Index for different values of *weight*.

Figure 15 that increasing the drone weight leads to improvements in terms of fleet stability.

Next, we start analysing the *3D map* modelling capability of CUSCUS. We need to remark that the *OpenStreetMap* maps do not contain the full description of the buildings and, in general, the height of the buildings is missing. For this reason we define the buildings height in this way: for the map of Bologna, we set the height with uniform random values on the range $[10..15]m$, while for the Manhattan map we set the height with uniform random values on the range $[50..150]m$. We modelled in *NS-3* the communication path loss according to the model described in Section 3.1.2. For the experiments, we set stone block walls ($12dB$ of signal attenuation) for the map of Bologna and concrete walls ($15dB$ of signal attenuation) for the map of Manhattan.

The next experiment studies the impact of the flight height on the coverage ability of the virtual springs algorithm. We made the experiments considering three different heights: low ($5m$), medium ($13m$) and high ($20m$), always using the map of Bologna. Figure 16 shows the results about scenario coverage for the three configurations. We can notice a huge difference when deploying the UAV

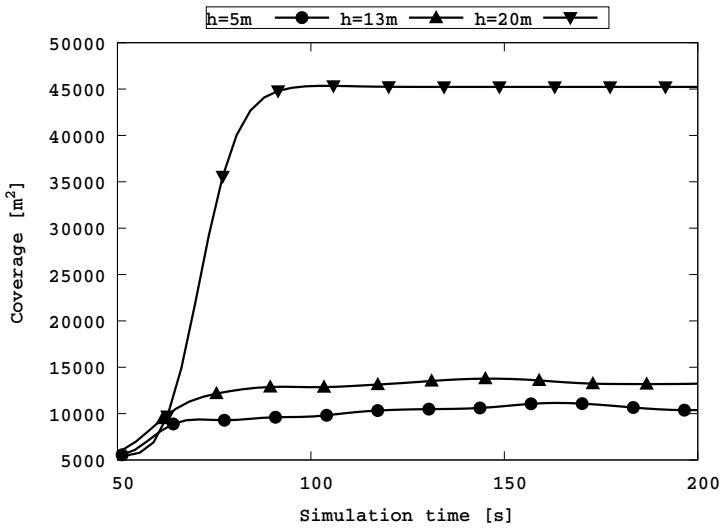

Figure 16: The aria covered by the fleet of UAVs at different heights.

fleet above or below the rooftop. At $h = 20m$, in fact, the UAVs can freely move without obstacles and their communications are always in line-of-sight. When the flight height is below the rooftop, the mobility of the UAVs is constrained by the presence of buildings. Furthermore, the communication links are no more in line-of-sight, hence the received link budged is drastically reduced. To cope with the presence of obstacles, the virtual spring algorithm reduces the relative distance between the UAVs in order to satisfy the requested LB_{ref}; as a result, the total coverage area is also reduced. These intuitive results validate the ability of the CUSCUS framework to model the urban environment and the issues related to it.

The possibility of loading a real map inside the simulator brings many advantages for the realism of the simulation analysis and for the prospect of a real deployment. In the next set of experiments, we show the SI metric for different values of K with the flight height $h = 5m$, i.e. below the rooftops, considering both the *Manhattan* (Figure 17) and the *Bologna* (Figure 18) scenarios. It is straightforward to notice that the behavior of the stiffness parameter K is very different from the previous experiments, when the UAVs were above the

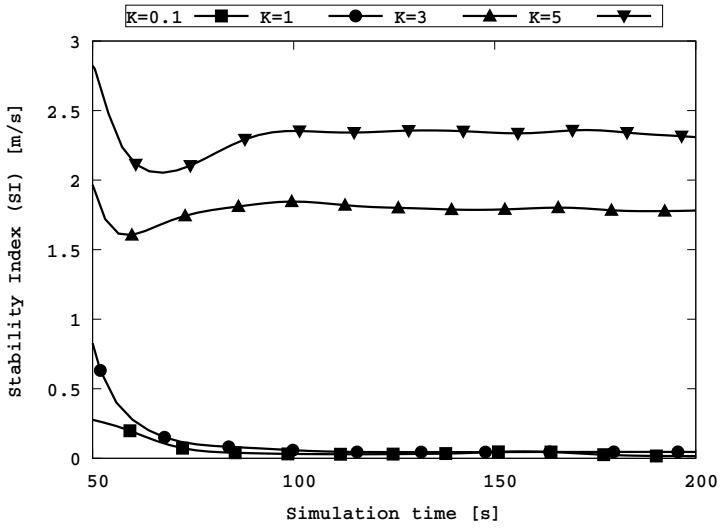

Figure 17: The Stability Index at low altitude for the Manhattan scenario.

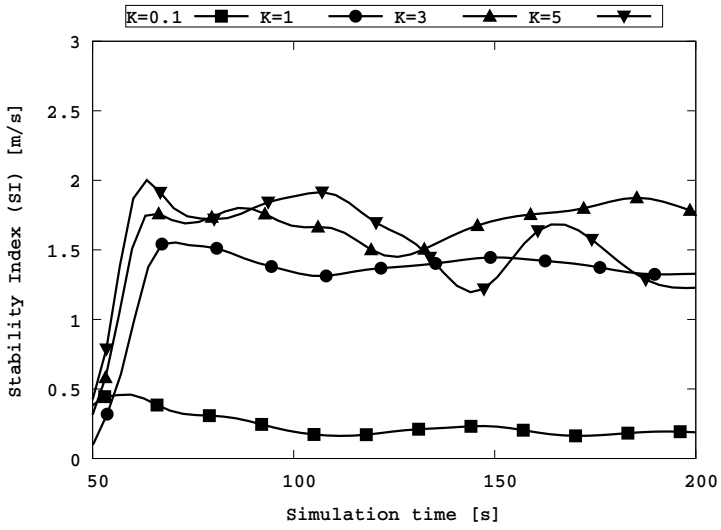

Figure 18: The Stability Index at low altitude for the Bologna scenario.

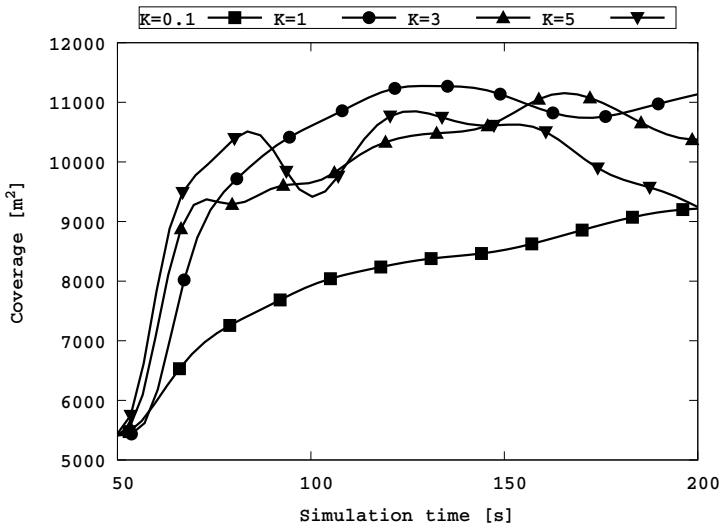

Figure 19: The area covered by the UAVs at low altitude in the center of Bologna.

rooftops. At low altitude, in fact, the system is quite unstable since it is too responsive to the position errors due to the the presence of buildings that impact the communication link budget between the UAVs.

We can observe that the virtual spring algorithm behaves very differently over the two scenarios. When considering the map of Bologna, in fact, the presence of narrow streets and of irregular buildings makes the UAVs movement highly constrained. As a result, stability is achieved only for very small values of the stiffness parameter K (see $K = 0.1$ in Figure 18). When instead considering the Manhattan map, characterized by the presence of large streets and of regular buildings, the system achieves stability even with greater values of K (see $K = 0.1$ and $K = 1$ in Figure 17). However the use of small values of K can impact the time required for system convergence. This is demonstrated in Figure 19, showing the Coverage Index (CI) for the Bologna scenario. We can see that for $K = 0.1$ the system is stable (Figure 18) but the coverage convergence is very slow (Figure 19). On the other side, using greater values of K, the system is largely unstable (Figure 18) but experiences a fast coverage convergence (Figure 19). These results further demonstrate the need of integrated

communication/control simulation tools like CUSCUS for determining the best trade-off between fleet stability and overall scenario coverage.

6. Conclusion and Future Works

In this paper, we have presented CUSCUS, a novel framework for modeling and simulation distributed Networked Control Systems, and more specifically fleets of Unmanned Aerial Vehicles (UAVs). Differently from the existing tools, our software is able to take into account both realistic UAV micro-mobility, drone dynamics and wireless communications, via the integration of the FL-AIR suite with the mainstream network simulator NS-3. Furthermore, CUSCUS enables realistic 3D simulations by importing the scenario description from the OpenStreetMaps, and by modeling the impact of obstacles on the wireless propagation as well as on the UAV mobility and path planning. The CUSCUS framework supports two usage modes: (*i*) as a benchmarking tool, it allows analyzing the performance of cross-layer algorithms (i.e.. mobilty-aware network protocols, or network-aware mobility algorithms), which constitute the main approaches in the literature of UAV systems; (*ii*) as a pre-deployment tool, it allows testing the operations of UAV applications in highly realistic simulated scenarios before their utilization in the real world, by using the same code. In this journal version, we have investigated the modeling in CUSCUS of two well-known research issues of distributed fleet management, i.e. UAV dynamic formation control and *Static Coverage*. With plenty of simulation results, we have demonstrated that the characteristics of the flying nodes (e.g. the weight), of the simulation scenario (e.g. position of the obstacles), and of the aerial control parameters (e.g. the PID parameters), might have a significant impact on the application performance, hence justifying the need of integrated control/networking tools for the accurate modeling of distributed Networked Control Systems. The current extension includes the comparison of the CUSCUS framework against test-beds in order to validate the realism of the software integration of the NS-3 and FL-AIR tools. Other future works pertain

to: the design and implementation of a library of cross-layer algorithms for the CUSCUS framework, the support for natural obstacles on rural scenarios (e.g. trees and hills), the implementation of load-balancing techniques for distributed simulation.

Acknowledgements

This work has been carried out in the framework of the DIVINA Challenge Team, which is funded by the Labex MS2T program. Labex MS2T is supported by the French Government, through the program "Investments for the future", managed by the French National Agency for Research (Reference ANR-11-IDEX-0004-02).

Bibliography

[1] L. Gupta, R. Jain, G. Vaszkun, Survey of important issues in uav communication networks, IEEE Communications Surveys Tutorials 18 (2) (2016) 1123–1152.

[2] M. Erdelj, E. Natalizio, Uav-assisted disaster management: Applications and open issues, in: 2016 International Conference on Computing, Networking and Communications (ICNC), 2016, pp. 1–5.

[3] S. Hayat, E. Yanmaz, R. Muzaffar, Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint, IEEE Communications Surveys Tutorials 18 (4) (2016) 2624–2661.

[4] M. Erdelj, E. Natalizio, K. R. Chowdhury, I. F. Akyildiz, Help from the sky: Leveraging uavs for disaster management, IEEE Pervasive Computing 16 (1) (2017) 24–32.

[5] S. Rao, D. Ghose, Sliding mode control-based autopilots for leaderless consensus of unmanned aerial vehicles, IEEE Transactions on Control Systems Technology 22 (5) (2014) 1964–1972.

[6] M. Yajnik, S. Moon, J. Kurose, D. Towsley, Measurement and modelling of the temporal dependence in packet loss, in: INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 1, 1999, pp. 345–352 vol.1.

[7] Y. Zeng, R. Zhang, T. J. Lim, Wireless communications with unmanned aerial vehicles: opportunities and challenges, IEEE Communications Magazine 54 (5) (2016) 36–42.

[8] T. T. Mac, C. Copot, T. T. Duc, R. D. Keyser, AR.Drone uav control parameters tuning based on particle swarm optimization algorithm, in: 2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), 2016, pp. 1–6.

[9] K. Daniel, S. Rohde, N. Goddemeier, C. Wietfeld, Cognitive agent mobility for aerial sensor networks, IEEE Sensors Journal 11 (11) (2011) 2671–2682.

[10] S. Morgenthaler, T. Braun, Z. Zhao, T. Staub, M. Anwander, UAVNet: A mobile wireless mesh network using unmanned aerial vehicles, in: 2012 IEEE Globecom Workshops, 2012, pp. 1603–1608.

[11] S. Rosati, K. Kruelecki, G. Heitz, D. Floreano, B. Rimoldi, Dynamic routing for flying ad hoc networks, IEEE Transactions on Vehicular Technology 65 (3) (2016) 1690–1700.

[12] Y. Cai, F. R. Yu, J. Li, Y. Zhou, L. Lamont, Medium access control for unmanned aerial vehicle (uav) ad-hoc networks with full-duplex radios and multipacket reception capability, IEEE Transactions on Vehicular Technology 62 (1) (2013) 390–394.

[13] E. Yanmaz, R. Kuschnig, C. Bettstetter, Achieving air-ground communications in 802.11 networks with three-dimensional aerial mobility, in: 2013 Proceedings IEEE INFOCOM, 2013, pp. 120–124.

[14] Phoenix rc, http://www.phoenix-sim.com.

[15] Simdrone from h-sim, http://www.h-sim.com.

[16] A. Varga, Omnet++, in: Modeling and Tools for Network Simulation, Springer, 2010, pp. 35–59.

[17] Network Simulator- ns (version 2), available from http://www.isi.edu/nsnam/ns/.

[18] FL-AIR: Framework libre air, available from https://uav.hds.utc.fr/software-flair/.

[19] P. Fuxjaeger, S. Ruehrup, Validation of the ns-3 interference model for ieee802.11 networks, in: 2015 8th IFIP Wireless and Mobile Networking Conference (WMNC), 2015, pp. 216–222.

[20] S. Manfredi, C. Pascariello, N. R. Zema, I. Fantoni, M. Krl, A cooperative packet-loss-tolerant algorithm for wireless networked robots rendezvous, in: 2017 International Conference on Computing, Networking and Communications (ICNC), 2017, pp. 1058–1062.

[21] A. Trotta, M. D. Felice, L. Bedogni, L. Bononi, F. Panzieri, Connectivity recovery in post-disaster scenarios through cognitive radio swarms, Computer Networks (Elsevier) 91 (2015) 68 – 89.

[22] M. D. Felice, A. Trotta, L. Bedogni, K. R. Chowdhury, L. Bononi, Self-organizing aerial mesh networks for emergency communication, in: 2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014, pp. 1631–1636.

[23] C. Sommer, F. Dressler, Progressing toward realistic mobility models in vanet simulations, IEEE Communications Magazine 46 (11) (2008) 132–137.

[24] V. Kumar, L. Lin, D. Krajzewicz, F. Hrizi, O. Martinez, J. Gozalvez, R. Bauza, iTETRIS: Adaptation of ITS technologies for large scale integrated simulation, in: 2010 IEEE 71st Vehicular Technology Conference, 2010, pp. 1–5.

[25] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, M. Dorigo, Argos: a modular, parallel, multi-engine simulator for multi-robot systems, Swarm Intelligence 6 (4) (2012) 271–295.

[26] M. Kudelski, L. M. Gambardella, G. A. D. Caro, RoboNetSim: An integrated framework for multi-robot and network simulation, Robotics and Autonomous Systems 61 (5) (2013) 483 – 496.

[27] Realfight drone, http://www.realflight.com/drone.

[28] Gazebo, http://gazebosim.org.

[29] Morse, http://www.openrobots.org/morse/doc/stable/morse.html.

[30] Ros, http://www.ros.org.

[31] Pixhawk, https://pixhawk.org.

[32] Ardupilot mega, http://www.ardupilot.co.uk.

[33] H. Nawaz, H. M. Ali, G. Nabi, Simulation based analysis of reactive protocols metrics in manet using opnet, Sindh University Research Journal (Science Series) 46 (4) (2014) 531–538.

[34] Q. N. Simulator, Scalable network technologies, Inc.[Online]. Available: www. qualnet. com.

[35] X. Zeng, R. Bagrodia, M. Gerla, Glomosim: a library for parallel simulation of large-scale wireless networks, in: Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on, 1998, pp. 154–161.

[36] F. Kargl, E. Schoch, Simulation of MANETs: A qualitative comparison between JiST/SWANS and Ns-2, in: Proceedings of the 1st International Workshop on System Evaluation for Mobile Platforms, MobiEval '07, ACM, New York, NY, USA, 2007, pp. 41–46.

[37] E. Ben Hamida, G. Chelius, J. M. Gorce, Impact of the physical layer modeling on the accuracy and scalability of wireless network simulation, Simulation 85 (9) (2009) 574–588.

[38] WNS3 '15: Proceedings of the 2015 Workshop on Ns-3, ACM, New York, NY, USA, 2015.

[39] Udt: Breaking the data transfer bottleneck, available from http://udt.sourceforge.net/.

[40] N. Point, Optitrack, Natural Point, Inc.,[Online]. Available: http://www.naturalpoint.com/optitrack/. [Accessed 22 2 2014].

[41] R. L. P. Castillo, A. Dzul, Modelling and Control of Mini-Flying Machines, Advances in Industrial Control, Springer-Verlag, London, 2005.

[42] Irrlicht 3d engine, available from http://irrlicht.sourceforge.net/.

[43] Xenomai, http://xenomai.org.

[44] A. Fouda, A. N. Ragab, A. Esswie, M. Marzban, A. Naser, M. Rehan, A. S. Ibrahim, Real time video streaming over NS3-based emulated LTE networks, Int. J. Electr. Commun. Comput. Technol.(IJECCT) 4 (3).

[45] M. Helsley, LXC: Linux container tools, IBM devloperWorks Technical Library.

[46] S. Manfredi, E. Natalizio, C. Pascariello, N. R. Zema, A packet loss tolerant rendezvous algorithm for wireless networked robot systems, Asian Journal of Control (2017).

[47] A. Eichler, H. Werner, Closed-form solution for optimal convergence speed of multi-agent systems with discrete-time double-integrator dynamics for fixed weight ratios, Systems & Control Letters 71 (2014) 7 – 13.

[48] W. Ren, Consensus strategies for cooperative control of vehicle formations, IET Control & Theory Applications 1 (2) (2007) 505–512.

[49] Federal communication commission (fcc) report, deployable aerial communications architecture in emergency communications, https://www.fcc.gov/general/deployable-aerial-communications-architecture-emergency-communications, 2011.

[50] B. Wang, H. B. Lim, D. Ma, A survey of movement strategies for improving network coverage in wireless sensor networks, Computer Communications 32 (13 - 14) (2009) 1427 – 1436.

[51] K. Derr, M. Manic, Extended virtual spring mesh (evsm): The distributed self-organizing mobile ad hoc network for area exploration, IEEE Transactions on Industrial Electronics 58 (12) (2011) 5424–5437.

[52] Cisco Aironet 802.11a/b/g Wireless CardBus Adapter, Data Sheet available on line at. http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-802-11a-b-g-cardbus-wireless-lan-client-adapter-cb21ag/product_data_sheet09186a00801ebc29.html.