IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 December 2023, accepted 13 December 2023, date of publication 25 December 2023,
date of current version 4 January 2024.

Digital Object Identifier 10.1109/ACCESS.2023.3347494

RESEARCH ARTICLE

Design and Validation of Cyber-Physical Systems Through Co-Simulation: The Voronoi Tessellation Use Case

CINZIA BERNARDESCHI[1], (Member, IEEE), ANDREA DOMENICI[1], ADRIANO FAGIOLINI[2], (Member, IEEE), AND MAURIZIO PALMIERI[1]
[1]Department of Information Engineering, University of Pisa, 56122 Pisa, Italy
[2]Department of Engineering, MIRPALab.it, University of Palermo, 90128 Palermo, Italy

Corresponding author: Cinzia Bernardeschi (cinzia.bernardeschi@unipi.it)

This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence), by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence) and by the European Union through the Next Generation EU project ECS00000017 'Ecosistema dell'Innovazione' Tuscany Health Ecosystem (THE), Piano Nazionale di Ripresa e Resilienza (PNRR), Spoke 3: Advanced technologies, methods and materials for human health and well-being.

ABSTRACT This paper reports on the use of co-simulation techniques to build prototypes of co-operative autonomous robotic cyber-physical systems. Designing such systems involves a mission-specific planner algorithm, a control algorithm to drive an agent performing its task; and the plant model to simulate the agent dynamics. An application aimed at positioning a swarm of unmanned aerial vehicles (drones) in a bounded area, exploiting a Voronoi tessellation algorithm developed in this work, is taken as a case study. The paper shows how co-simulation allows testing the complex system at the design phase using models created with different languages and tools. The paper then reports on how the adopted co-simulation platform enables control parameters calibration, by exploiting design space exploration technology. The INTO-CPS co-simulation platform, compliant with the Functional Mock-up Interface standard to exchange dynamic simulation models using various languages, was used in this work. The different software modules were written in Modelica, C, and Python. In particular, the latter was used to implement an original variant of the Voronoi algorithm to tesselate a convex polygonal region, by means of dummy points added at appropriate positions outside the bounding polygon. A key contribution of this case study is that it demonstrates how an accurate simulation of a cooperative drone swarm requires modeling the physical plant together with the high-level coordination algorithm. The coupling of co-simulation and design space exploration has been demonstrated to support control parameter calibration to optimize energy consumption and convergence time to the target positions of the drone swarm. From a practical point of view, this makes it possible to test the ability of the swarm to self-deploy in space in order to achieve optimal detection coverage and allow unmanned aerial vehicles in a swarm to coordinate with each other.

INDEX TERMS Cyber-physical systems, co-simulation, unmanned aerial vehicles, space coverage, Voronoi tessellation, control parameter calibration.

I. INTRODUCTION

Cyber-Physical Systems (CPS) are a large class of systems characterized by a complex interplay of hardware and software components, often operating in a largely unpredictable environment. CPSs are usually designed with model-based development (MBD) techniques [1]: design begins with a high-level model of the system, which is validated, corrected, and refined in several cycles, until a model is obtained that developers judge fit to be taken as a production blueprint.

System validation is used to check the compliance of each system element and of the overall system with its purpose and functions. Validation activities are planned and carried out throughout the development process.

Validating CPS models is complex, as they must reflect the interactions of several subsystems and different aspects

The associate editor coordinating the review of this manuscript and approving it for publication was Nasim Ullah.

of each subsystem: For example, the physical aspects of an actuator include mechanics and electromagnetism, mechanics in turn require modeling such concepts as torque, inertia, viscosity, and so on. Such complexity makes it desirable, and often necessary, to rely on several modeling languages and tools. Actually, most tools offer a large set of specialized libraries that span a wide range of application fields, nevertheless it is often the case that more specific tools are preferred. Also, CPSs may be developed in parallel by various teams, possibly belonging to different organizations, which may have expertise with different modelling tools.

In cyber-physical systems, a co-simulation approach can be adopted, which allows different system elements to be simulated each by dedicated simulators, with a co-simulation master orchestrating their execution. The Functional Mock-up Interface [2] is a public-domain standard that defines a container and an interface to exchange dynamic simulation models using various languages. FMI is supported by more than one hundred modelling tools [3].

In this work, co-simulation is used to study the behaviour of a swarm of unmanned aerial vehicles (UAV) (or *drones*) that must cover uniformly a given area, which is often useful in many applications, such as search and rescue or environmental monitoring. Due to their maneuvrability, UAVs are convenient tools for monitoring areas difficult to reach for land vehicles [4], [5]. In particular, among different types of UAV's, quadcopters have become the most popular for their flexibility, due to very low moments of inertia, greater stability, hovering capability, as well as lower take-off requirements [6], [7].

Designing a swarm of drones involves three levels of control: (i) a high-level, mission specific *planner* algorithm defining the final position of each drone; (ii) an intermediate-level *control* algorithm to drive each drone along its prescribed trajectory; and (iii) a low-level *plant* model to simulate the drone dynamics.

In this schema, the crucial part is the design of the intermediate-level algorithm. In fact, in a space-coverage task the final position of the drones is entirely determined by the geometry of the area to be covered and by the number of drones, so that designing the high-level control reduces to choosing an appropriate geometric algorithm. In turn, simulating the individual drones relies on standard dynamics models whose parameters depend on the physical characteristics of the drone.

The intermediate-level control, instead, has a more complex task, involving the optimization of various contrasting figures of merit (e.g,, speed and fuel consumption), whose relative weight depends on the particular application with its operational constraints.

This paper discusses how co-simulation and design space exploration support the development of this kind of systems. In particular, the approach based on centroidal Voronoi tessellation has been used for the high-level control.

Its Python implementation was validated with a prototype simulation based on a simplified physical model. The validated implementation was then integrated in a full-scale co-simulation, along with the two other control algorithms. Co-simulation was performed on the INTO-CPS framework [8].

The Voronoi algorithm was first verified using typical values for controller parameters. Then the INTO-CPS design space exploration tool was used to calibrate the nonlinear controller parameters to optimize power consumption and responsiveness.

The main contribution of the paper consists in (i) demonstrating the application of co-simulation to the problem of quadcopter control for space coverage; (ii) showing how design-space exploration coupled to co-simulation supports control parameter calibration to optimize energy consumption and convergence time. A technique to extend the tessellation algorithm to a bounded area is also introduced.

This paper is organized as follows: Section II reports related work on cyber-physical systems design, including modeling/simulation tools and co-simulation. Section III introduces background notions (quadcopters, Voronoi tessellation, and co-simulation), for the selected case study. Section IV presents the technique developed for centroidal tessellation of a bounded area. Section V reports the co-simulation architecture, the simulation scenario, and the preliminary results from co-simulation. Section VI shows and discusses the calibration of controller parameters for energy consumption and convergence time. Finally, Section VII concludes the paper and reports on future work.

II. RELATED WORK

System models are typically built with *block-based* languages, which describe a system as an assembly of functional blocks, each representing a possibly complex mathematical operation, interconnected by data flows. For example, MATLAB/Simulink [9], [10] is a commercial environment for the model-based development of cyber-physical systems. It provides a graphical model editor and functions to generate demonstrative prototypes and to analyse relevant model properties.

Some works define ad-hoc solutions for the integration of different simulators, dedicated to different system components, e.g., the hardware and the software parts. For example, in [11] Simulink and a simulator of logic specifications (the *pvsio* interpreter of the Prototype Verification System [12]) have been integrated to simulate and formally verify properties of a pacemaker.

Farhat et al. [13], instead, use Matlab and Simpack to compare the performance of mechatronically-driven railway vehicles' guidance and steering to that of a conventional vehicle. A non-linear Simpack vehicle model with the specific mechatronic actuation and sensing, and a simplified linear control model in Matlab/Simulink are co-simulated.

Reinhart and Weissenberger [14] address multibody simulation in the context of numerical control machines.

The multibody model considers flexible machine structural components, guideways, feed drive dynamics and axis controllers together with the motion trajectory generation of the control.

In [7], Simulink/Gazebo tools are used for the validation of an approach to quadcopter control where wind disturbance is modeled by unknown exogenous inputs, exploiting the Robot Operating System (ROS) middleware in the simulations. In [15], the architecture ROS/Gazebo has been extended with the possibility of simulation of co-operative UAVs.

Co-simulation is a technique in which simulators of heterogeneous models are executed simultaneously while exchanging data during run-time. The FMI (functional mockup interface) [2] is a standardized interface for model exchange and co-simulation of dynamic models across different modeling and simulation tools.

Co-simulation has been applied extensively in different application fields. In [16], a co-simulation approach is used for a brushless motor, with the electrical and mechanichal parts modeled in Simulink and the feedback linearization control modeled with a function written in C.

In [17], a co-simulation open source software for operation and planning studies of distributed energy resources has been presented. The software allows users to perform large-scale high-fidelity simulations for bulk power system (BPS) planning and operation.

Papers [18] and [19] show how human performance models can be incorporated into models of CPSs. In [18], a train driver model is coupled to models of the rolling stock and of the movement authority; in [19] human-machine interfaces of an integrated clinical environment are considered.

In [20], co-simulation was applied to analyse Model Predictive Control systems for autonomous driving. This requires an accurate analysis of the interplay among three main components: the plant, the model predictive control algorithm, and the processor where the algorithm is executed. Co-simulation of the three components was used to determine if the controller running on the chosen hardware meets the time requirements determined by the response time of the plant. Satisfactory tradeoffs between algorithm complexity and processor performance could be studied.

In [21], the co-simulation-based framework *Vico* for marine crane onboard operations is presented. Simulation and analysis of sub-systems is performed with different software tools and the framework enables the digitalization of marine operations.

In [22], co-simulation is applied to co-operative mobile robots. Simulations for multi-robot systems are executed by independent tools, such as Gazebo for physics and mobility, ROS2 for software development, and ns-3 for communications and the networking infrastructure [23]. The framework integrates such simulators and allows running experiments that combine all the involved robotic systems keeping the synchronization time between the simulators consistent.

In [24], an approach to the formal verification of a variant of a well-known consensus protocol of UAVs is presented, using a co-simulation framework for system validation. The present paper builds on [24], discussing control parameter calibration by design space exploration, using a more complex case study.

In [25], Cho et al. present an advanced co-simulation platform that concurrently performs UAV simulation and wireless network simulation. This platform is taylored to the simulation of centrally controlled UAV swarms, whereas the approach of the present paper can be applied to distributed control applications, although only a central control case study is discussed. Also, the implementation reported by Cho et al. does not rely on the FMI standard.

An extensive survey on co-simulation has been published by Gomes et al. [26], providing definitions of the fundamental concepts and a taxonomy of the literature based on the *discrete events* and *continuous time* computational models.

A paper by Fitzgerald et al. [27] introduces foundational and process management issues for the design of CPSs and cites several methods, languages, and tools, using a two-wheel self-balanced personal transporter as an example.

An approach to CPS design extending the software-defined networking (SDN) concept *is the software-defined cyber-physical systems* (SD-CPS) framework [28], wherein a workflow of microservices (web services), coordinated by an SDN controller, realises the support for CPS operation. An SD-CPS federated controller deployment is proposed, which comprises several controllers from different domains or organizations at the edge, in a case study with embedded mobile systems in connected cars, edge cloud nodes and servers. Another approach inspired by SDN and service-oriented architecture is *Cyber Physical System as a Software-defined Service* (CPSS) [29], wherein the CPS control platform is divided into the Infrastructure-as-a-Service, Network-as-a-Service, and Aplication-as-a-Service layers. CPS design methods as exemplified by the above references can be smoothly integrated with co-simulation approaches such as the one discussed in the present paper, due to their reliance on a distributed software architecture.

III. BACKGROUND

This section introduces notions used in the rest of the paper: quadcopter unmanned vehicles, Voronoi tessellation and the INTO-CPS framework for co-simulation.

A. QUADCOPTERS

Quadcopters have become very popular UAVs for their flexibility. However, combined control of rotational and translational motions is required to obtain accurate path tracking and autonomous flight capacity, which results in highly nonlinear modeling [30].

The dynamics of a quadcopter can be described with reference to Fig. 1, where the craft is modeled as a cross-shaped rigid frame supporting four rotors, each turning at speed $\omega_i (i = 1, \ldots, 4)$, whose thrust and torque determine

the velocity and attitude of the craft. A moving reference frame F′ has its origin at the center of mass, with the arms lying on axes x' and y', whereas F, with axes x, y, and z, is a fixed reference frame.

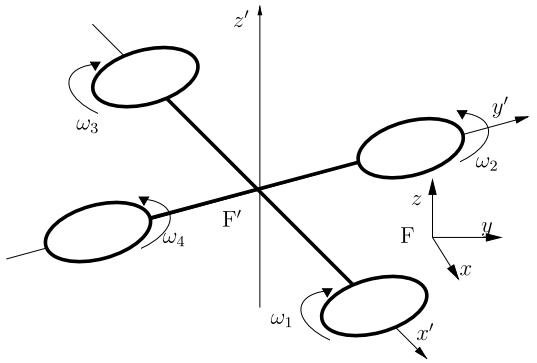

FIGURE 1. Schematic representation of a quadcopter.

The attitude of the craft is defined by its Euler angles ψ, ϕ, and θ (*yaw*, *roll*, and *pitch*), which describe the rotation that brings the axes of F′ to coincide with those of F after an appropriate translation.

Equations (1) and (2) below express the dynamics for position and attitude, respectively, as functions of the rotational speeds of each rotor:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} g(\phi \sin\psi_\mathrm{d} + \theta \cos\psi_\mathrm{d}) \\ -g(\phi \cos\psi_\mathrm{d} + \theta \sin\psi_\mathrm{d}) \\ C_z(\omega_1 + \omega_2 + \omega_3 + \omega_4) \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{pmatrix} = \begin{pmatrix} C_\phi(\omega_2 - \omega_4) \\ C_\theta(\omega_3 - \omega_1) \\ C_\psi(\omega_1 - \omega_2 + \omega_3 - \omega_4) \end{pmatrix}, \tag{2}$$

where g is the gravity acceleration, ψ_d is the desired yaw angle, C_ϕ, C_θ, C_ψ, and C_z are physical parameters.

The quadcopter controller must compute the rotor speeds that will drive it to the desired position (x_d, y_d, z_d) with the desired yaw ψ_d. This can be achieved with the design shown in Fig. 2, where the position controller commands the attitude controller, which computes the rotor speeds. The latter are fed to each rotor's low-level controller, included in the plant block, that translates the desired rotor speeds into current levels for the motors.

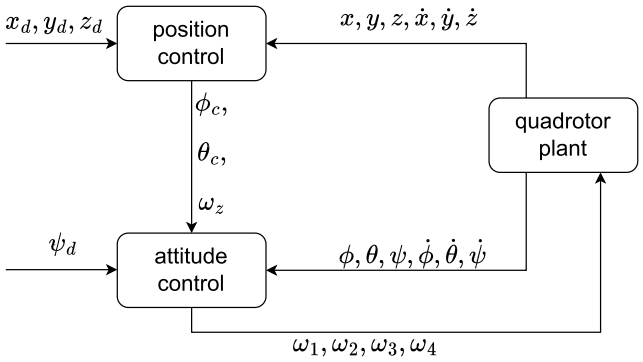

FIGURE 2. Cascaded position-attitude control model.

It may be shown [30] that a position controller can be defined by an equation of the following form, where $e_x = x - x_\mathrm{d}$, $e_y = y - y_\mathrm{d}$ and $e_z = z - z_\mathrm{d}$ are the tracking errors; moreover, $\mathrm{s_d} = (\sin\psi_\mathrm{d})/g$ and $\mathrm{c_d} = (\cos\psi_\mathrm{d})/g$:

$$\begin{pmatrix} \phi_c \\ \theta_c \\ \omega_z \end{pmatrix} = \begin{pmatrix} -\mathrm{s_d}(2\lambda_\mathrm{P}\dot{x} + \lambda_\mathrm{P}^2 e_x) + \mathrm{c_d}(2\lambda_\mathrm{P}\dot{y} + \lambda_\mathrm{P}^2 e_y) \\ -\mathrm{c_d}(2\lambda_\mathrm{P}\dot{x} + \lambda_\mathrm{P}^2 e_x) - \mathrm{s_d}(2\lambda_\mathrm{P}\dot{y} + \lambda_\mathrm{P}^2 e_y) \\ -\frac{2\lambda_\mathrm{P}}{C_z}\dot{z} + \frac{\lambda_\mathrm{P}^2}{C_z} e_z \end{pmatrix} \tag{3}$$

An attitude controller can be defined by equations of the following form:

$$\begin{pmatrix} \omega_\phi \\ \omega_\theta \\ \omega_\psi \end{pmatrix} = \begin{pmatrix} -\frac{2\lambda_\mathrm{A}}{C_\phi}\dot{\phi} - \frac{\lambda_\mathrm{A}^2}{C_\phi}(\phi - \phi_c)) \\ -\frac{2\lambda_\mathrm{A}}{C_\theta}\dot{\theta} - \frac{\lambda_\mathrm{A}^2}{C_\theta}(\theta - \theta_c)) \\ -\frac{2\lambda_\mathrm{A}}{C_\psi}\dot{\psi} - \frac{\lambda_\mathrm{A}^2}{C_\psi}(\psi - \psi_\mathrm{d})) \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} \omega_z - 2\omega_\theta + \omega_\psi \\ \omega_z + 2\omega_\phi - \omega_\psi \\ \omega_z + 2\omega_\theta + \omega_\psi \\ \omega_z - 2\omega_\phi - \omega_\psi \end{pmatrix}. \tag{5}$$

In the above equations, λ_P and λ_A are the controller gain parameters, which affect the responsiveness of the controller to tracking errors. Specifically, λ_A is related to the desired responsiveness of attitude stabilization, while λ_P is related to that of position control. Since the attitude control loop is internal to the position control one, the frequencies of the eigenvalues associated with the attitude dynamics should be higher than those of the position dynamics. As a rule of thumb, the two gain parameters should ideally satisfy the condition $\lambda_\mathrm{A}/\lambda_\mathrm{P} > 10$. In this work, the physical parameters of Erle-copters (Fig. 3) are used in the simulations [7].

FIGURE 3. An Erle-copter.

B. VORONOI DIAGRAMS

Given a set S of k points in a plane Π, a Voronoi diagram [31] is a partition of Π into k *Voronoi regions* (or *cells*) V_i, each associated with a distinct point (*seed* or *generator*) $s_i \in S$, and consisting of all the points in the plane closer to s_i than to any other point s_j, with $j \neq i$, in S, i.e. [32]:

$$\forall i \in [1..n]: \quad V_i = \{p \in \Pi \mid \forall j \neq i : \forall s_j \in S : |s_i - p| \leq |s_j - p|\}. \tag{6}$$

In general, the Voronoi region V_i of a point $s_i \in S = \{s_1, \ldots, s_k\}$ can be constructed as follows: (i) the

perpendicular bisectors l_{ij} of the segments between s_i and the other points of S are drawn; (ii) if Π_{ij} is the half plane bounded by l_{ij} and containing s_i, V_i is the intersection of the Π_{ij}'s. Figure 4 shows the construction of the cell for seed s_1 in a set of six seeds. The Voronoi diagram for set S is then the union of the Voronoi regions, as shown in Figure 5. Each region is bounded by a finite number of segments or half lines, called *ridges*. In the figure, dashed lines represent infinite ridges delimiting unbounded regions. The points where two or more ridges meet are called *vertices*.

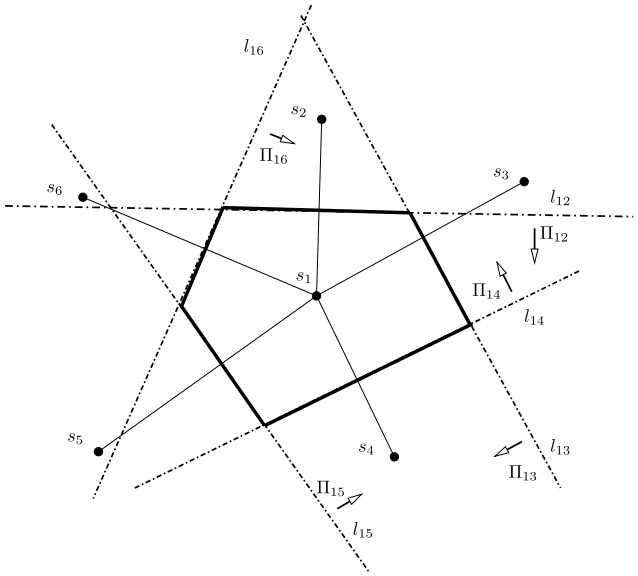

FIGURE 4. Construction of the Voronoi region for seed s_1.

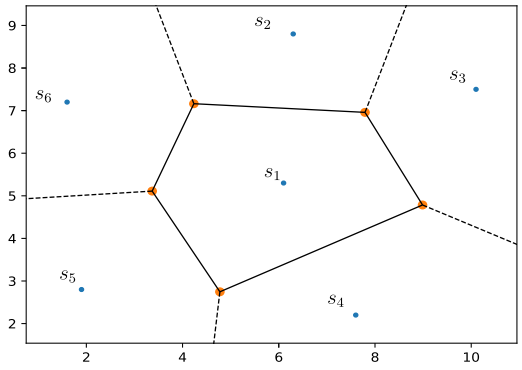

FIGURE 5. Voronoi diagram for the six points of Fig. 4.

1) AN IMPLEMENTATION OF THE VORONOI TESSELLATION ALGORITHM

The implementation of the Voronoi algorithm used in this work relies on the Python class *SciPy.Spatial.Voronoi* [33]. In order to show how a Voronoi diagram is represented in the Python code, we use the simple example of Figures 4 and 5. The `Voronoi` class is initialized with an array containing the tessellation seeds, as shown in Listing 1.

The class constructor computes the tessellation vertices and ridges, stored in class attributes, as shown in

```
>>>seeds = np.array([
...                  [1.6, 7.2],
...                  [1.9, 2.8],
...                  [6.1, 5.3],
...                  [6.3, 8.8],
...                  [7.6, 2.2],
...                  [10.1, 7.5]
...                 ])
>>> vor = Voronoi(seeds)
```

LISTING 1. Set of seeds.

```
>>> vor.vertices
array([[8.98840764, 4.78471338],
       [3.36882475, 5.11037441],
       [4.77593918, 2.74642218],
       [4.23518289, 7.16227526],
       [7.79255162, 6.95899705]])
>>> vor.ridge_vertices
[[-1, 0], [-1, 2], [0, 2], [-1, 1],
[1, 2], [3, 4], [-1, 3], [-1, 4],
[0, 4], [1, 3]]
```

LISTING 2. Data structures for example of Fig. 5.

Listing 2. Each vertex is identified by its index in the array `vor.vertices`, ranging from 0 to 5 in the example. The vertex indices are used in the list `vor.ridge_vertices` to define the ridges. Each ridge is identified by its extremes: if an extreme is a vertex it is referred to by its index in the `vor.vertices` array, while an extreme at infinite distance is represented by -1. For example, the pair $[-1, 0]$ represents a half line originating from vertex 0, i.e., point [8.98840764, 4.78471338], whereas [0, 1] represents the ridge between vertices 0 and 1, i.e., points [8.98840764, 4.78471338] and [3.36882475, 5.11037441].

The above data can be compared with Figure 5, produced by the program. The blue dots are the seeds, the solid lines are the ridges of the only finite region, and dashed lines represent (semi)infinite ridges.

2) CENTROID COMPUTATION

In many applications of drone swarms, it is convenient to place each drone in the geometric center, or *centroid* of the respective cell. A Voronoi tessellation whose generating points are the centroids of each cell is called a *centroidal Voronoi tessellation* [31]. A centroidal Voronoi tessellation can be obtained by *Lloyd's method* [34]:

Given an initial set of k seeds $\{s_{0,i}\}_{i=1}^{k}$;

1) construct the initial Voronoi tessellation $\{V_{0,i}\}_{i=1}^{k}$ for the seeds $\{s_{0,i}\}_{i=1}^{k}$;
2) at each step $n+1$, compute the centroids of the Voronoi regions $\{V_{n,i}\}_{i=1}^{k}$ found at step n; these centroids become the new set of seeds $\{s_{n+1,i}\}_{i=1}^{k}$;
3) If this new set of seeds meets some convergence criterion, terminate; otherwise, return to step 2.

Step 2 above is performed by computing the centroid coordinates (C_x, C_y) with the formulae for a polygon with n

vertices [35], [36]:

$$A = \frac{1}{2}\Sigma_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$
$$C_x = \frac{1}{6A}\Sigma_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$
$$C_y = \frac{1}{6A}\Sigma_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\ .$$

This method produces a set of drone placements that optimize the average distance of each drone from the points of its assigned region (*centroid* property) and the spacing between drones (*Voronoi tessellation* property).

C. CO-SIMULATION

Co-simulation techniques allow a complex and heterogeneous simulated system to be split into simple sub-systems. The simulation of each sub-system can be executed by a dedicated simulator, possibly supporting a different modeling language.

The *Functional Mockup Interface* (FMI) [37] is a standard for co-simulation. Each sub-system is exported as a *Functional Mockup Unit* (FMU) and a co-simulation is the execution of a set of FMUs that exchange data and signals under the co-ordination of a *master* program. Figure 8 in Section V shows an example of co-simulation architecture, where the *Co-simulation Orchestration Engine* (COE) is the master program. Various frameworks for FMI-compliant co-simulation have been developed, including INTO-CPS [8], [38] and CoSim20 [39]. In particular, the INTO-CPS framework is used in this work.

The INTO-CPS toolchain includes the Design Space Exploration (DSE) tool [40], used to find optimal combinations of parameter values within specified ranges or discrete sets searched by a DSE engine that executes the co-simulations for each combination. In particular, the Pareto method is applied to ranking the results of the DSE. The set of combinations with the highest rank (rank 1) represents the best compromises of a pair of objective functions. The set contains all combinations of parameter values where it is not possible to find other combinations which improve both objective functions.

IV. CENTROIDAL TESSELLATION OF A BOUNDED REGION

The Python implementation of the Voronoi algorithm described in Section III-B1 was taken as a starting point for the simulation of a drone swarm. That implementation computes a tessellation over an unbounded plane, while the envisioned application involves covering a bounded region. A workaround was found to tessellate a convex polygonal region: *dummy* points at appropriate positions *outside* the bounding polygon are added to the original set of seeds, then the Python implementation is applied to the extended set to tessellate the infinite plane containing the polygon. More precisely, for each seed one dummy point is placed symmetrically to the seed with respect to each side of the polygon. so that, for each seed, as many symmetric points as the number of sides of the polygon are created.

As an example, Figure 6 shows the generation of the tessellation of a square with four seeds, with sides parallel to the cartesian axes. Figure 6(a) shows the seeds and the dummy points corresponding to the leftmost seed and Figure 6(b) shows the tessellation computed by the Voronoi algorithm. The tessellation of the bounding square is obviously obtained by deleting all cells outside the square, as shown in Figure 6(c).

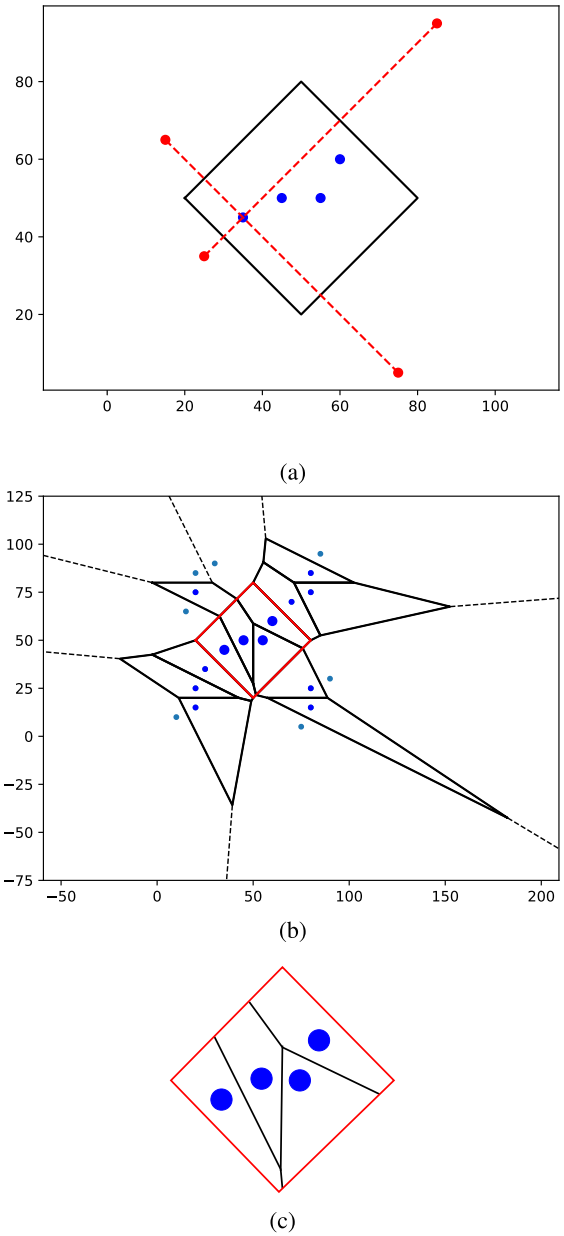

FIGURE 6. How to build a Voronoi diagram inside a polygon.

This modified algorithm was used to perform Step 1 of Lloyd's method (Section III-B2). In this way, the behaviour of a swarm of drones that must optimally cover an area bounded by a polygon can be described as follows:

1) The initial positions of the drones are the seeds of the initial tessellation;
2) the (modified) Voronoi tessellation for the current seeds is computed;
3) for each cell, the centroid is computed;
4) each drone is moved to the centroid of the respective cell;
5) if the new position of all drones is within a given (small) distance from the previous one, stop; otherwise, repeat Step 2 with the new positions as current seeds.

This behavioral model was implemented in Python, obtaining a high-level simulator of a space coverage strategy. Figure 7 shows the initial, an intermediate, and the final configurations of a ten-drone swarm deployed over a hexagonal area. In the pictures, produced by the Python simulation, the blue dots are the initial drone position at each cycle of the algorithm, and the red ones are the corresponding centroids.

The model abstracts from the physical model of the drones, thus ignoring finite speeds, control-loop delays, and communication delays. A more accurate model, implemented by co-simulation, is discussed in the next section.

V. CO-SIMULATION OF A UAV SWARM

Co-simulation was used to study the behavior of a drone swarm with the task of reaching an optimal coverage of a plane surface bounded by a polygon. To this purpose, the planner algorithm, implemented in Python, uses the centroidal Voronoi tessellation described above to assign, at each simulation step, a destination point to each drone, starting from arbitrary initial positions. The related FMU has been generated using UniFMU [41]. The drone's destination is the command input to the intermediate-level drone controller, implemented in C, that computes the rotor speeds according to (3), (4), and (5) in Section III-A. A low-level plant model, implemented in Modelica [42], simulates the drone according to (1) and (2). More details on the control FMU and plant FMUs are discussed in [24].

Figure 8 shows the architecture of the co-simulation of a swarm of drones, where all the FMUs are connected to the master algorithm of the co-simulation, the Co-simulation Orchestration Engine (COE), while Figure 9 shows the logical connections between the planner and one drone. The logical connections can be explicitly provided to the COE through a graphical user interface made available by the INTO-CPS Association.

The co-simulation has two synchronization granularities: the fixed co-simulation step T at which the plant and the intermediate-level controller interact, and a step T' at which the planner FMU is executed. Step T' is an integral multiple of T, i.e., $T' = \tau T$, with τ a positive integer.

A. SIMULATION SCENARIO

In the simulated scenario, ten drones must be placed over a hexagonal area of 1600 m^2. The co-simulation step T (0.05 s) was chosen to be of the same order as the period between successive updates of the outputs computed by actual controllers in drone systems.

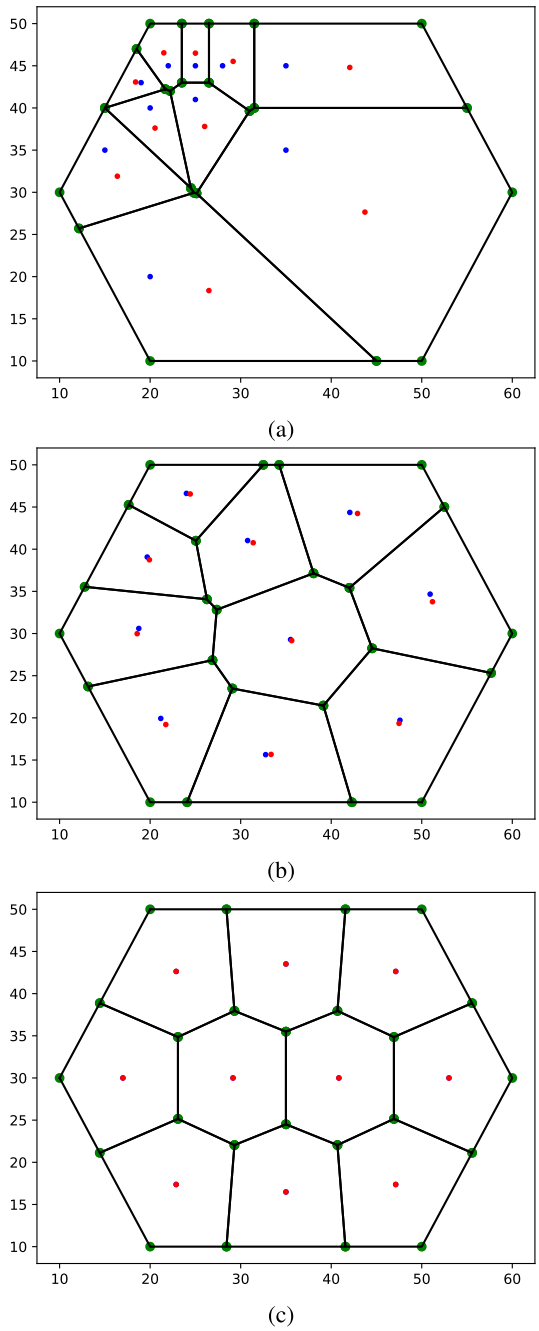

FIGURE 7. How to build a Voronoi diagram inside a polygon.

Figure 10 shows the initial displacement of the drones as shown by the graphical user interface created with the technology discussed in [43] that allows users to track the movement of the drones through the evolution of the co-simulation and Figure 10b shows the final placement achieved by the Voronoi tessellation.

In the co-simulation, the system evolves as follows:

1) the Voronoi tessellation is computed every τ co-simulation steps;

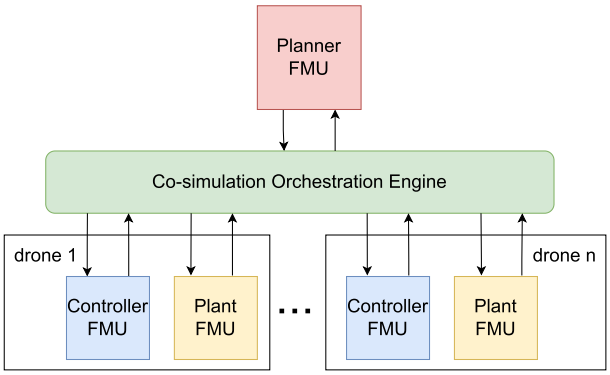

FIGURE 8. Co-simulation architecture of a swarm.

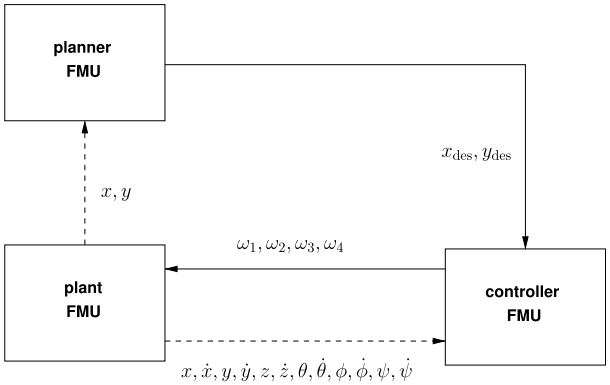

FIGURE 9. Logical communication between the FMUs.

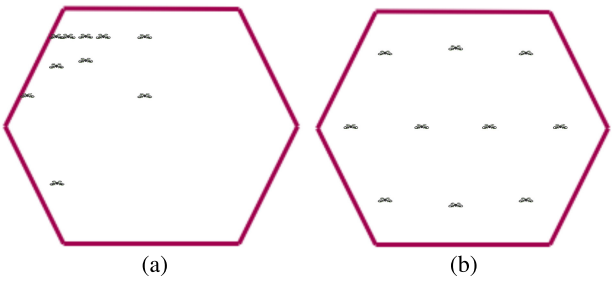

FIGURE 10. Drone positions shown by the GUI.

2) each drone executes the control algorithm every co-simulation step T, moving towards the centroid of its region for τ co-simulation steps, then a new tessellation is computed;
3) the simulation ends at time T_s, This value is chosen large enough to let the drones reach the desired final "honeycomb" formation. Informally, when the drones reach such a configuration, they keep executing the algorithm, which does not further change their positions (a formal definition of convergence is given in subsection VI).

B. PRELIMINARY TESTS FOR λ_A AND λ_P

A preliminary series of co-simulations was performed to find starting points and ranges for the parameters to be optimised by exploring the design space. The following results were obtained using a trial and error approach. For example, with $T = 0.05$ s, the co-simulation reveals divergent behaviors, even for values of λ_A and λ_P that are close to satisfying the ideal condition $\lambda_A/\lambda_P > 10$ (Section III-A). Specifically, Figures 11 and 12 show that the coordinates x and y begin to oscillate around the desired values and finally diverge. In addition, it was found that the acceptable ranges of values for controller parameters depend also on the controller update rate T. Indeed, in the instances of Figures 13 and 14, the x and y coordinates oscillate and diverge, even though the ideal ratio between the two control gain parameters is satisfied. In general, it was found that, for $T = 0.05$ s, the controller parameters must satisfy the two constraints $\lambda_A \leq 9.0$ and $\lambda_P \leq 0.9$. Figure 15 shows the result of a co-simulation where these last constraints are met, and as expected the x and y coordinates move towards the values of the assigned centroid, without oscillations; the final position is reached in approximately two minutes of simulated time.

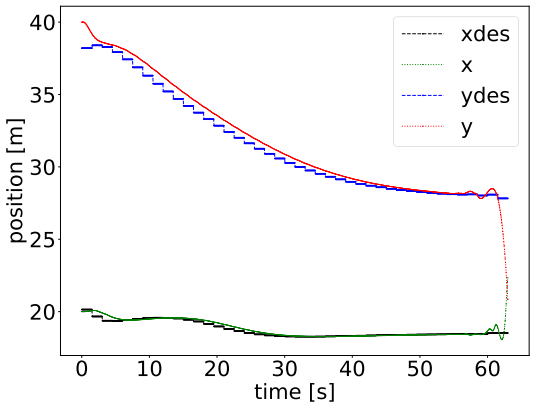

FIGURE 11. Drone 1, T = 0.05 s, $\lambda_A = 9.5$, $\lambda_P = 0.98$.

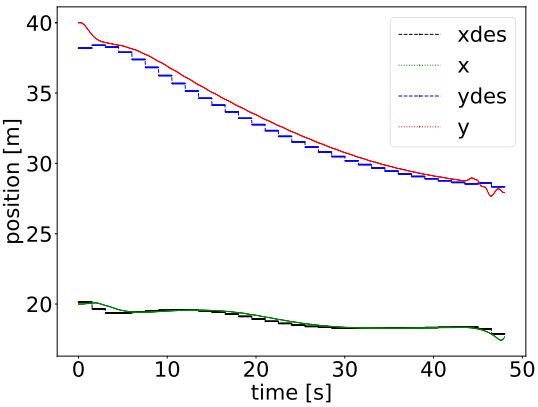

FIGURE 12. Drone 1, T = 0.05 s, $\lambda_A = 9$, $\lambda_P = 1.0$.

VI. DESIGN SPACE EXPLORATION

The DSE tool was then used to find optimal values for controller parameters λ_A and λ_P, according to the criteria of minimizing (i) time t_c to attain convergence, and (ii) energy consumption.

Convergence was defined to be attained at the first instant since when each drone remains within a circle of radius

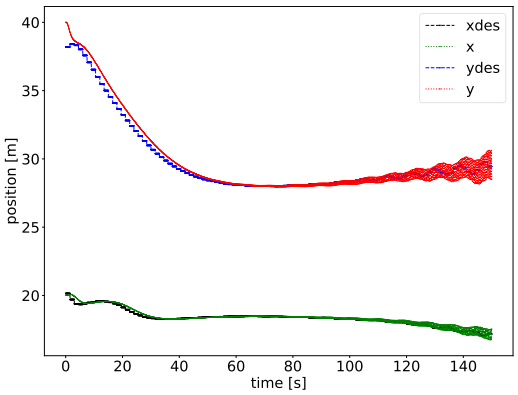

FIGURE 13. Drone 1, T = 0.05 s, $\lambda_A = 9.5$, $\lambda_P = 0.9$.

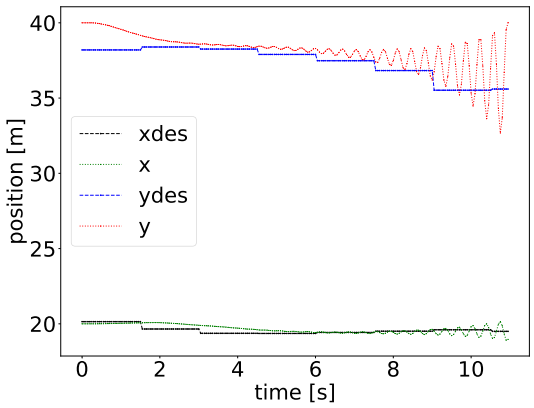

FIGURE 14. Drone 1, T = 0.05 s, $\lambda_A = 10.1$, $\lambda_P = 1.0$.

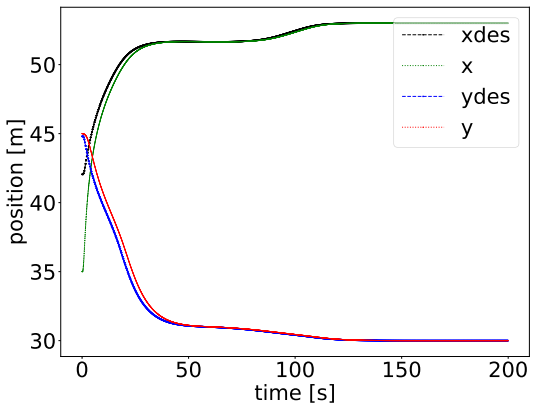

FIGURE 15. Drone 3, $\lambda_A = 8.0$, $\lambda_P = 0.9$.

R from its target position for a minimum interval of δ seconds. The chosen values for R and δ were 0.01 m and 5 s. Convergence times were computed off-line from simulation traces.

Energy consumption for a motor is computed with the following formula:

$$E = \sum_{i=0}^{t_c/T} P_i T \tag{7}$$

where E is the total energy until t_c, T is the simulation step, and P_i is the motor power at step i:

$$P_i = K_f \omega_d^2 \tag{8}$$

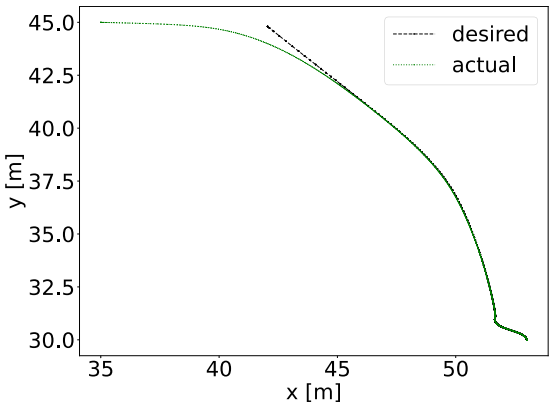

FIGURE 16. Drone 3, $\lambda_A = 8.0$, $\lambda_P = 0.9$.

where K_f is a constant of the motor and ω_d is the difference between motor speed and the hovering speed ω_h, a characteristic of the drone. We observe that energy consumption is affected by the update frequency of the desired co-ordinates (τT).

Table 1 shows the parameter values defining the design space considered in the co-simulations, with the constraint $\lambda_A \geq 5\lambda_P$, resulting in twenty-six combinations. The simulation time T_s is 300 s for all simulation runs.

TABLE 1. Design space.

parameter	values	unit
τ	6, 20	
λ_A	3.5, 4.5, 7, 8, 9	s
λ_P	0.7, 0.8, 0.9	s

Results are shown ordered by convergence time in Tables 2 and 3, by energy consumption in Tables 4 and 5, and by Pareto rank in Tables 6 and 7.

TABLE 2. Ordered by convergence time ($\tau = 6$).

t_c [s]	rank	E [J]	λ_A	λ_P	τ
150.1	1	2.242	9	0.9	6
152.25	1	1.266	8	0.9	6
155.25	1	1.095	7	0.9	6
160.65	1	0.953	9	0.8	6
162.3	1	0.280	8	0.8	6
164.6	1	0.171	7	0.8	6
175.35	2	0.579	9	0.7	6
175.65	3	1.001	4.5	0.9	6
176.65	1	0.141	8	0.7	6
178.45	1	0.076	7	0.7	6
178.65	2	0.112	4.5	0.8	6
188.65	1	0.041	4.5	0.7	6
200.8	1	0.038	3.5	0.7	6

Tables 2 and 3 confirm that larger values of τ cause longer convergence times. As an example, consider the case of $\lambda_A = 7$ and $\lambda_P = 0.9$. For $\tau = 6$ (Table 2) the update frequency of desired co-ordinates is 0, 3 s and the convergence time is 155.25 s. For $\tau = 20$, (Table 3) the update frequency of desired co-ordinates is 1 s and the convergence time is 244, 85 s. In particular, for $\tau = 20$ (Table 3), the values of t_c show that the maximun simulated time of 300 s was reached

TABLE 3. Ordered by convergence time (τ=20).

t_c [s]	rank	E [J]	λ_A	λ_P	τ
224.7	4	1.003	4.5	0.9	20
244.85	5	1.111	7	0.9	20
253.75	3	0.601	9	0.7	20
255.35	6	1.300	8	0.9	20
256.6	3	0.113	4.5	0.8	20
260.25	4	0.151	8	0.7	20
267.65	7	2.346	9	0.9	20
270.45	2	0.080	7	0.7	20
278.45	5	1.003	9	0.8	20
290.4	2	0.038	3.5	0.7	20
300	3	0.042	4.5	0.7	20
300	5	0.180	7	0.8	20
300	6	0.299	8	0.8	20

before reaching convergence. Further, for a given value of τ, parameter λ_P determines the speed of convergence.

TABLE 4. Ordered by energy consumption (1).

E [J]	rank	t_c [s]	λ_A	λ_P	τ
0.038	1	200.8	3.5	0.7	6
0.038	2	290.4	3.5	0.7	20
0.041	1	188.65	4.5	0.7	6
0.042	3	300	4.5	0.7	20
0.076	1	178.45	7	0.7	6
0.080	2	270.45	7	0.7	20
0.112	2	178.65	4.5	0.8	6
0.113	3	256.6	4.5	0.8	20
0.141	1	176.65	8	0.7	6
0.151	4	260.25	8	0.7	20
0.171	1	164.6	7	0.8	6
0.180	5	300	7	0.8	20
0.280	1	162.3	8	0.8	6
0.299	6	300	8	0.8	20

TABLE 5. Ordered by energy consumption (2).

E [J]	rank	t_c [s]	λ_A	λ_P	τ
0.579	2	175.35	9	0.7	6
0.601	3	253.75	9	0.7	20
0.953	1	160.65	9	0.8	6
1.001	3	175.65	4.5	0.9	6
1.003	4	224.7	4.5	0.9	20
1.003	5	278.45	9	0.8	20
1.095	1	155.25	7	0.9	6
1.111	5	244.85	7	0.9	20
1.266	1	152.25	8	0.9	6
1.300	6	255.35	8	0.9	20
2.242	1	150.1	9	0.9	6
2.346	7	267.65	9	0.9	20

Tables 4 and 5 show that energy consumption increases with higher values of λ_A and λ_P (with few exceptions). It may be observed that the impact of λ_P on energy consumption is greater than the impact of λ_A and that τ has a relatively low impact on energy consumption. The ordered data series is split into two tables for readability.

Finally, Tables 6 and 7 compare the optimality of the various combinations of values for τ, λ_A and λ_P by respective Pareto rank. Here, too, the ordered data series is split into two tables.

In Figure 17, it is possible to observe the combinations with the best rank (rank 1), shown with green points joined by a green line. The combinations that provide the best tradeoff

TABLE 6. Ordered by ranking (1).

rank	E [J]	t_c [s]	λ_A	λ_P	τ
1	0.038	200.8	3.5	0.7	6
1	0.041	188.65	4.5	0.7	6
1	0.076	178.45	7	0.7	6
1	0.141	176.65	8	0.7	6
1	0.171	164.6	7	0.8	6
1	0.280	162.3	8	0.8	6
1	0.953	160.65	9	0.8	6
1	1.095	155.25	7	0.9	6
1	1.266	152.25	8	0.9	6
1	2.242	150.1	9	0.9	6
2	0.038	290.4	3.5	0.7	20
2	0.080	270.45	7	0.7	20
2	0.112	178.65	4.5	0.8	6

TABLE 7. Ordered by ranking (2).

rank	E [J]	t_c [s]	λ_A	λ_P	τ
2	0.579	175.35	9	0.7	6
3	0.042	300	4.5	0.7	20
3	0.113	256.6	4.5	0.8	20
3	0.601	253.75	9	0.7	20
3	1.001	175.65	4.5	0.9	6
4	0.151	260.25	8	0.7	20
4	1.003	224.7	4.5	0.9	20
5	0.180	300	7	0.8	20
5	1.003	278.45	9	0.8	20
5	1.111	244.85	7	0.9	20
6	0.299	300	8	0.8	20
6	1.300	255.35	8	0.9	20
7	2.346	267.65	9	0.9	20

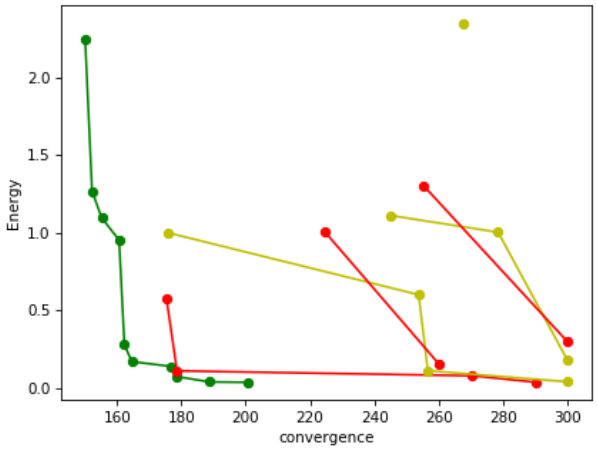

FIGURE 17. Pareto ranking graph.

are the ones closest to the bottom left corner of the graph, i.e, $\tau = 6$, $\lambda_P = 0.8$, and $\lambda_A = 7$ or $\lambda_A = 8$.

The controller defined by (3) is nonlinear, which makes the characterization of optimal combinations of parameter values more difficult than in the case of linear systems. For this reason, simulation-based design space exploration is an effective tool to deal with this problem.

VII. CONCLUSION

This paper proposes an approach based on co-simulation to design and validate co-operative robotic systems since the early phases of the system design. In particular, this work shows how design space exploration can be successfully

applied to the expensive task of parameter calibration for the control algorithms of a co-operative UAV-based cyber-physical system.

One of the advantages of this approach is the easy interaction between multidisciplinary teams (e.g., information engineers, automation engineers, etc), due to the support for heterogeneous languages and simulation tools afforded by co-simulation. For example, in the use case discussed in this work, Python was used for the co-ordination algorithm, C for the drone controller and Modelica for the drone plant. Another advantage is found in model reuse, as models already available can be imported as black boxes in the co-simulation framework, provided that the original modeling tools support the FMI standard. Moreover, this modularity eases system design refinement. The model presented in this paper, for example, could be refined by including network communications and delays or extended by considering alternative system architectures.

In general, this approach makes it possible to develop a consistent set of models for a given system at different levels of abstraction, enabling a wide spectrum of analyses to be performed while minimizing development effort. It may be added that the same approach can be used in more complex applications. For example, instead of a swarm of drones of the same type, the deployment of vehicles of different kinds, possibly both air- and land-faring, could be considered.

As further work, it is planned to consider (i) different system architectures for the co-ordination of vehicles; and (ii) study UAVs equipped with further sensors, for example cameras, for activities such as disaster management and recovery.

REFERENCES

[1] B. Selic, "The pragmatics of model-driven development," *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, Sep. 2003.

[2] T. Blockwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *Proc. Linköping Electron. Conf.*, Nov. 2012, pp. 173–184, doi: 10.3384/ecp12076173.

[3] *FMI Standard*. Accessed: Nov. 21, 2023. [Online]. Available: https://fmi-standard.org/

[4] S. Saeedi, C. Thibault, M. Trentini, and H. Li, "3D mapping for autonomous quadrotor aircraft," *Unmanned Syst.*, vol. 5, no. 3, pp. 181–196, 2017.

[5] N. Michael, S. Shen, K. Mohta, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro, "Collaborative mapping of an earthquake damaged building via ground and aerial robots," in *Field Service Robotics*. Berlin, Germany: Springer, 2014, pp. 33–47.

[6] K. Nonami, F. Kendoul, S. Suzuki, W. Wang, and D. Nakazawa, *Autonomous Flying Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles*. Berlin, Germany: Springer, 2010.

[7] K. Kumar, S. I. Azid, A. Fagiolini, and M. Cirrincione, "Erle-copter simulation using ROS and gazebo," in *Proc. IEEE 20th Medit. Electrotechnical Conf. (MELECON)*, Jun. 2020, pp. 259–263.

[8] P. G. Larsen, J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis, and A. Sadovykh, "Integrated tool chain for model-based design of cyber-physical systems: The INTO-CPS project," in *Proc. 2nd Int. Workshop Model., Anal., Control Complex CPS (CPS Data)*, Apr. 2016, pp. 1–6.

[9] *Scicoslab Web Site*. Accessed: Nov. 21, 2023. [Online]. Available: http://www.scicoslab.org

[10] *Simulink Web Site*. Accessed: Nov. 21, 2023. [Online]. Available: http://www.mathworks.com/products/simulink

[11] C. Bernardeschi, A. Domenici, and P. Masci, "A PVS-simulink integrated environment for model-based analysis of cyber-physical systems," *IEEE Trans. Softw. Eng.*, vol. 44, no. 6, pp. 512–533, Jun. 2018.

[12] S. Owre, J. Rushby, and N. Shankar, "PVS: A prototype verification system," in *Automated Deduction—CADE* (Lecture Notes in Computer Science), vol. 607, D. Kapur, Ed. Berlin, Germany: Springer, 1992, pp. 748–752.

[13] N. Farhat, C. P. Ward, R. M. Goodall, and R. Dixon, "The benefits of mechatronically-guided railway vehicles: A multi-body physics simulation study," *Mechatronics*, vol. 51, pp. 115–126, May 2018.

[14] G. Reinhart and M. Weissenberger, "Multibody simulation of machine tools as mechatronic systems for optimization of motion dynamics in the design process," in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, Sep. 1999, pp. 605–610.

[15] C. Bernardeschi, A. Fagiolini, M. Palmieri, G. Scrima, and F. Sofia, "ROS/gazebo based simulation of co-operative UAVs," in *Modelling and Simulation for Autonomous Systems*, J. Mazal, Ed. Cham, Switzerland: Springer, 2019, pp. 321–334.

[16] C. Bernardeschi, P. Dini, A. Domenici, M. Palmieri, and S. Saponara, "Formal verification and co-simulation in the design of a synchronous motor control algorithm," *Energies*, vol. 13, no. 16, p. 4057, Aug. 2020. [Online]. Available: https://www.mdpi.com/1996-1073/13/16/4057

[17] K. Balasubramaniam, S. Plathottam, S.-I. Yim, N. Kang, K. Jhala, R. Bhattarai, and S. Zhao, "Co-simulation of transmission and distribution systems—From modeling to software development," *IEEE Access*, vol. 10, pp. 127061–127072, 2022.

[18] T. Hotzel Escardo, K. Pierce, D. Golightly, and R. Palacin, "Modelling train driver behaviour in railway co-simulations," in *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops*, L. Cleophas and M. Massink, Eds. Cham, Switzerland: Springer, 2021, pp. 249–262.

[19] M. Palmieri, C. Bernardeschi, and P. Masci, "A framework for FMI-based co-simulation of human–machine interfaces," *Softw. Syst. Model.*, vol. 19, no. 3, pp. 601–623, May 2020.

[20] C. Bernardeschi, P. Dini, A. Domenici, A. Mouhagir, M. Palmieri, S. Saponara, T. Sassolas, and L. Zaourar, "Co-simulation of a model predictive control system for automotive applications," in *Software Engineering and Formal Methods. SEFM 2021 Collocated Workshops*, A. Cerone, M. Autili, A. Bucaioni, C. Gomes, P. Graziani, M. Palmieri, M. Temperini, and G. Venture, Eds. Cham, Switzerland: Springer, 2022, pp. 204–220.

[21] Z. Liu, Y. Chu, G. Li, and H. Zhang, "A co-simulation-based system using Vico for marine operation," in *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops*, P. Masci, C. Bernardeschi, P. Graziani, M. Koddenbrock, and M. Palmieri, Eds. Cham, Switzerland: Springer, 2023, pp. 228–241.

[22] M. Richart, F. Velázquez, F. Ciuffardi, J. Visca, and J. Baliosian, "CoCoSim: A tool for co-simulation of mobile cooperative robots," in *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops*, P. Masci, C. Bernardeschi, P. Graziani, M. Koddenbrock, and M. Palmieri, Eds. Cham, Switzerland: Springer, 2023, pp. 258–268.

[23] G. F. Riley and T. R. Henderson, *The NS-3 Network Simulator*. Berlin, Germany: Springer, 2010, pp. 15–34.

[24] C. Bernardeschi, A. Domenici, A. Fagiolini, and M. Palmieri, "Co-simulation and formal verification of co-operative drone control with logic-based specifications," *Comput. J.*, vol. 66, no. 2, pp. 295–317, Feb. 2023, doi: 10.1093/comjnl/bxab161.

[25] W. J. Cho, S. Kim, Y. Kim, and Y. H. Moon, "Advanced co-simulation platform for UAV simulations under virtual wireless network environments," *IEEE Access*, vol. 10, pp. 95498–95508, 2022.

[26] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, p. 49, 2018.

[27] J. Fitzgerald, C. Gamble, P. G. Larsen, K. Pierce, and J. Woodcock, "Cyber-physical systems design: Formal foundations, methods and integrated tool chains," in *Proc. IEEE/ACM 3rd FME Workshop Formal Methods Softw. Eng.*, May 2015, pp. 40–46.

[28] P. Kathiravelu and L. Veiga, "SD-CPS: Taming the challenges of cyber-physical systems with a software-defined approach," in *Proc. 4th Int. Conf. Softw. Defined Syst. (SDS)*, May 2017, pp. 6–13.

[29] H. Yu, H. Qi, and K. Li, "CPSS: A study of cyber physical system as a software-defined service," *Proc. Comput. Sci.*, vol. 147, pp. 528–532, Jan. 2019.

[30] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robot. Autom. Mag.*, vol. 19, no. 3, pp. 20–32, Sep. 2012.
[31] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: Applications and algorithms," *SIAM Rev.*, vol. 41, no. 4, pp. 637–676, Jan. 1999.
[32] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Berlin, Germany: Springer-Verlag, 1985.
[33] *Class Scipy Spatial Voronoi*. Accessed: Nov. 21, 2023. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html#scipy-spatial-voronoi
[34] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.
[35] P. Bourke. (1988). *Calculating the Area and Centroid of a Polygon*. [Online]. Available: http://paulbourke.net/geometry/polygonmesh/
[36] R. Nurnberg. (1988). *Calculating the Area and Centroid of a Polygon in 2D*. [Online]. Available: http://nurnberg.maths.unitn.it/centroid.pdf
[37] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, "The functional mockup interface for tool independent exchange of simulation models," in *Proc. Linköping Electron. Conf.*, Dresden, Germany, Jun. 2011, pp. 105–114, doi: 10.3384/ecp11063105.
[38] (2019). *INTO-CPS Online Documents*. [Online]. Available: https://into-cps-association.readthedocs.io/en/latest/
[39] G. Liboni and J. Deantoni, "CoSim20: An integrated development environment for accurate and efficient distributed co-simulations," in *Proc. Int. Conf. Big Data Manage.*, New York, NY, USA, May 2020, pp. 76–83.
[40] C. Gamble, "DSE in the INTO-CPS platform," INTO-CPS Deliverable, Aarhus Univ., Denmark, Tech. Rep. D5.3e, 2017.
[41] C. Legaard, D. Tola, T. Schranz, H. Macedo, and P. Larsen, "A universal mechanism for implementing functional mock-up units," in *Proc. 11th Int. Conf. Simul. Model. Methodologies, Technol. Appl.*, 2021, pp. 121–129.
[42] P. Fritzson, *Principles of Object Oriented Modeling and Simulation With Modelica 2.1*. Hoboken, NJ, USA: Wiley, 2004.
[43] M. Palmieri, C. Bernardeschi, and P. Masci, "A flexible framework for FMI-based co-simulation of human-centred cyber-physical systems," in *Software Technologies: Applications and Foundations*, M. Mazzara, I. Ober, and G. Salaun, Eds. Berlin, Germany: Springer, 2018, pp. 21–33.

CINZIA BERNARDESCHI (Member, IEEE) received the Laurea degree in computer science and the Ph.D. degree in electronic, computer, and telecommunications engineering from the University of Pisa, Italy, in 1987 and 1996, respectively. Currently, she is an Associate Professor with the Department of Information Engineering, University of Pisa. Her research interests include software engineering, dependable systems, the application of formal methods for specification and verification of safety-critical systems, and reliability and security issues of cyber-physical systems. Recently, she collaborated on the following projects: "EPI: European Processor Initiative," funded by the EU under call H2020; "HiEfficient: Highly EFFICIENT and reliable electric drivetrains based on modular, intelligent, and highly integrated wide band gap power electronics modules," funded by ECSEL Research and Innovation Actions (RIA). She is a member of the ERCIM Working Group on Formal Methods for Industrial Critical Systems (FMICS) and of the IEEE Systems, Man, and Cybernetics Society (SMC) Technical Committee on Homeland Security.

ANDREA DOMENICI received the Ph.D. degree in information engineering from the University of Pisa, Italy, in 1992, with a thesis on the implementation of the Gödel logic programming language. He has been an Assistant Professor with the Sant'Anna School of University Studies and Doctoral Research, Pisa, and the Department of Information Engineering, University of Pisa, where he taught software engineering and did research in the fields of dependable systems and application of formal methods in the development of safety and mission-critical systems. He has previously worked on grid architectures, in cooperation with CERN and the Italian National Institute of Nuclear Physics. Since his retirement, in 2022, he has been under a Research Contract with the Department of Information Engineering.

ADRIANO FAGIOLINI (Member, IEEE) received the M.S. degree in computer science engineering and the Ph.D. degree in robotics and automation from the University of Pisa, in 2004 and 2009, respectively. He was a Visiting Researcher with the Department of Energy, IUT Longwy, Université de Lorraine, France, in 2019, and the Department of Mechanical Engineering, University of California at Riverside, in 2015 and 2017. He enrolled at the Summer Student Program, European Center for Nuclear Research (CERN), Geneva, in 2002. He is currently an Associate Professor with the University of Palermo, Italy. His research interests include soft robotics, self-driving racecars, autonomous aircraft, and security and distributed intrusion detection in cyber-physical systems. He is a member of the IEEE Robotics and Automation Society, the IEEE Control System Society, the IEEE Industry Application Society, and the I-RIM Italian Institute for Intelligent Machines. In 2008, he led the team of the University of Pisa during the first European Space Agency's Lunar Robotics Challenge, which resulted in a second-place prize for the team. He was one of the recipients of the IEEE ICRA Best Manipulation Paper Award, in 2005.

MAURIZIO PALMIERI received the master's degree in computer engineering from the University of Pisa, in 2016, and the joint Ph.D. degree in smart computing from the University of Florence, the University of Pisa, Italy, and the University of Siena, in 2020. Currently, he is a Research Fellow of computer engineering with the Department of Information Engineering, University of Pisa. He is also working in collaboration with the INTO-CPS Association, an association that maintains a toolchain for cyber-physical systems modeling, simulation, and verification. His research interests include the application of machine learning approaches to the healthcare system and the application of the prototype verification system theorem prover in co-simulation scenarios for safety analysis.