Etat de l'art sur la co-simulation robotique et réseau des systèmes multi-robots

Théotime Balaguer [a,b,c]
theotime.balaguer@insa-lyon.com

Olivier Simonin[a] olivier.simonin@insa-lyon.fr
Isabelle Guérin Lassous[b] isabelle.guerin-lassous@ens-lyon.fr
Isabelle Fantoni[c] isabelle.fantoni@ls2n.fr

[a]INSA Lyon, Inria, CITI Lab., France

[b]Université Lyon 1, ENS Lyon, CNRS, LIP, UMR 5668, France

[c]Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, France

Résumé

La simulation de systèmes multi-robots nécessite l'intégration des composantes robotique et réseau au sein d'un même environnement de simulation. La réutilisation des outils existants dans les communautés robotique et réseau semble être une évidence, mais la fusion de deux simulateurs présente des défis structurels qu'il faut surmonter afin d'obtenir une simulation "réaliste" d'un système multi-robots. Dans cet état de l'art, nous présentons les co-simulateurs qui abordent cette problématique et détaillons les défis auxquels il faut répondre pour la conception d'un co-simulateur performant.

Mots-clés : *Systèmes multi-robots, Co-simulation, Réseaux de communication*

Abstract

Simulation of multi-robots systems require the integration of both the robotic and network components. Leveraging the existing tools from each community seems evident, but merging two completely different simulators proves challenging. This paper presents the state of the art in co-simulators that tackle this problematic, and provide details on the challenges arising when one wants to create an efficient co-simulator.

Keywords: *Multi-robots systems, Co-simulation, Communication networks*

1 Introduction

Les systèmes multi-robots ont montré, ces dernières décennies, qu'ils pouvaient surpasser un robot seul pour certaines missions, et répondre à des problématiques impossibles à gérer avec une seule entité [4]. Grâce aux coûts toujours décroissants des composants électroniques, il est de plus en plus facile de déployer une flotte de robots collaboratifs qui achèvent les missions plus vite, à plus grande échelle et avec plus de robustesse. Les développements de l'intelligence artificielle et de l'algorithmique distribuée participent par ailleurs grandement à la démocratisation des systèmes multi-robots.

Les systèmes de robots connectés cristallisent des problématiques à la frontière entre le domaine de la robotique et celui des réseaux [14]. Pour les étudier, il est essentiel de disposer d'outils de simulation réalistes pouvant simuler les deux aspects de front et permettant de mener des expériences rapides, reproductibles et sûres. Par exemple, la thématique des flottes de drones reflète parfaitement ce besoin de simulation conjointe entre la robotique et les réseaux. D'un côté, les drones ont des contraintes de déplacement qu'il faut respecter (rayon de braquage, accélération, etc.) et de l'autre, les échanges d'information entre les drones peuvent être vitaux pour la flotte (évitement de collisions, vol en formation, etc.) ou pour la mission (partage d'informations, relais réseau, etc.).

Nous verrons que les communautés de chacun des deux domaines – robotique et réseau – disposent déjà d'outils établis et performants [7][16]. Assembler ces outils, c'est-à-dire faire de la *co-simulation*, semble être une solution adaptée. Il est aussi important d'exploiter pleinement ROS (*Robot Operating System*) [1], une suite de logiciels facilitant le développement et l'intégration des systèmes robotiques et qui s'est imposée comme un outil essentiel dans la conception et l'opération de robots.

Dans ce papier, nous présentons et comparons les co-simulateurs existants et discutons des principales problématiques survenant lors de la conception de co-simulateurs. Cet état de l'art peut permettre de choisir le bon outil de co-simulation pour la recherche ou le développe-

1. ROS : https ://www.ros.org/

ment de systèmes multi-robots communiquants.

Organisation de l'article En Section 2, nous présentons les outils de simulation existants indépendamment pour la robotique et les réseaux. La Section 3 expose les co-simulateurs robotique et réseau existants. En Section 4, nous discutons l'architecture type d'un co-simulateur ainsi que les fonctionnalités recherchées. Les Sections 5 et 6 s'intéressent à deux problématiques récurrentes de la co-simulation, à savoir la synchronisation et l'échange d'information entre les simulateurs. Nous présentons nos premiers résultats et concluons en Section 7.

2 Simulateurs spécifiques

Si l'évaluation rigoureuse et exhaustive de simulateurs de robotique et de réseau dépasse largement les limites de cet article, il est nécessaire pour la compréhension de notre discours de présenter ici les simulateurs les plus notables pour chacun de ces domaines. Pour plus de détails, le lecteur peut se référer à [7] pour les simulateurs de robotique (aussi appelés simulateurs de physique) et [16] pour les simulateurs de réseau.

2.1 Simulateurs multi-robots

Le rôle du simulateur de robotique est de simuler les interactions du robot avec son environnement. Dans [7], les auteurs présentent un état de l'art des simulateurs de robotique complet mais sans se focaliser sur les systèmes multi-robots. Selon les auteurs, un simulateur de robotique doit contenir *a minima* :

- Un moteur simulant les phénomènes physiques ;
- Des modèles de friction et de collision ;
- Une interface graphique ;
- Une fonctionnalité d'import de scènes et de *meshes* ;
- Une interface de programmation applicative (API) permettant l'accès à ses propriétés :
- Une bibliothèque d'effecteurs et de capteurs prêts à l'emploi.

De nombreux simulateurs de physique ont été développés ces dernières années. Certains sont spécialisés dans un sous-domaine de la robotique (robotique aérienne [24], sous-marine [18] [6], corps mous [10], etc.) alors que d'autres permettent de simuler une grande variété de robots. Pour la simulation de systèmes multi-robots, nous nous intéressons particulièrement à la flexibilité et l'efficacité du simulateur. La flexibilité d'un simulateur est son aptitude à s'adapter facilement à de nouvelles situations (différents modèles de robots ou moteurs de physique, par exemple). L'efficacité d'un simulateur est sa capacité à utiliser l'ensemble des capacités de calcul mises à sa disposition.

Devant cette grande diversité de simulateurs, nous nous restreignons ici aux simulateurs multi-robots, actuellement maintenus et qui sont les plus utilisés par la recherche. Par exemple, Airsim [24] est un simulateur réaliste très utilisé mais ne supportant pas les systèmes multi-robots.

ARGoS. ArGoS [21] est un simulateur spécifiquement créé pour la simulation multi-robots. Ce simulateur est basé sur une architecture modulaire offrant à l'utilisateur une grande flexibilité et une bonne efficacité. En effet, différents modules caractérisés par leur précision et leur coût en calcul peuvent être choisis, pour allouer la puissance de calcul là où l'utilisateur considère que c'est nécessaire. ARGoS est capable de simuler de manière efficace des essaims de plusieurs milliers de robots grâce à une caractéristique le démarquant des autres simulateurs : le monde 3D simulé peut être découpé en régions. L'utilisateur peut assigner à chaque région un moteur de physique différent (plus ou moins précis, par exemple), et les calculs associés à chaque région peuvent être parallélisés.

Webots. Le développement de Webots a démarré en 1996, faisant de ce simulateur l'un des plus anciens simulateurs multi-robots. Longtemps distribué sous licence propriétaire par Cyberbotics Ltd., Webots est distribué gratuitement depuis 2018, ce qui a accéléré son adoption par la communauté scientifique. En plus de fonctionnalités avancées et d'un haut niveau de réalisme, Webots propose des services annexes tels que l'accès à une version du simulateur sur le web [2] ou un support utilisateur. Concernant notre problématique robotique/réseau, il est intéressant de remarquer que Webot procure des noeuds *émetteur* et *récepteur* pouvant être utilisés pour modéliser des liaisons radio simples, en paramétrant une portée maximale de communication et un angle d'ouverture du cône d'émission pour les émetteurs infrarouges.

CoppeliaSim. CoppeliaSim [23], anciennement appelé V-REP, est un simulateur de robotique dont l'architecture permet une gestion efficace des systèmes multi-robots. Une étude récente comparant les performances et précision

2. RobotBenchmark : https ://robotbenchmark.net/

de quatre simulateurs populaires dans la communauté de la robotique [11] place CoppeliaSim parmi les meilleurs simulateurs multi-robots (Gazebo, Webots, CoppeliaSim et MORSE sont étudiés). Bien que gratuit, CoppeliaSim n'est pas open source, un facteur qui peut être limitant pour l'interfaçage avec d'autres simulateurs.

Gazebo. Gazebo (illustré figure 1) est le simulateur de robotique le plus utilisé dans la recherche [7], et ses fonctionnalités le rendent parfaitement adapté à la simulation multi-robots.
Gazebo s'est construit autour du projet Player/Stage [13], puis son développement a été confié à l'*"Open Source Robotics Foundation"* (OSRF) en 2012. Pour améliorer sa flexibilité, le développement de Gazebo a pris un tournant majeur en 2019, avec la publication d'une version entièrement réécrite du simulateur. L'architecture monolithique historique ayant été remplacée par une architecture modulaire, Gazebo est capable de simuler tous types de robots, dans n'importe quel environnement, et en acceptant des possibilités d'extension infinies. La plus grande force de Gazebo réside cependant dans sa communauté nombreuse et active, qui partage de nouveaux modules, corrige des bugs et procure de l'aide en ligne. Le support des industriels à la fondation *"Open Source Robotics"* [3] fournit un avantage non négligeable pour le maintien et le développement de ce simulateur. L'OSRF étant aussi responsable du développement de ROS, on peut facilement affirmer que ROS et Gazebo sont et resteront entièrement compatibles.

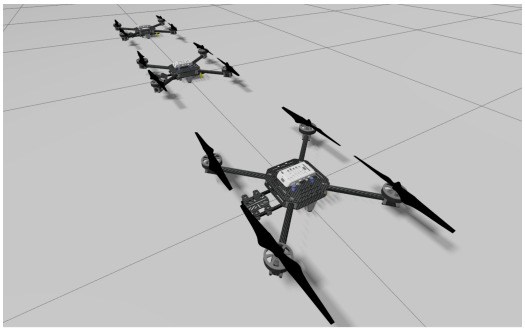

FIGURE 1 – Visualisation de trois robots quadricoptères dans Gazebo

2.2 Simulateurs réseaux

Un simulateur de réseau crée un modèle virtuel d'un réseau d'ordinateurs. Il permet d'expérimenter rapidement des algorithmes, d'évaluer et de comparer les performances d'architectures et de technologies de communication diverses. L'unité principale de modélisation est souvent le "paquet", ce qui rend les simulateurs de réseau très réalistes pour simuler les protocoles mais moins pour les couches physiques (propagation notamment) et applicatives, qui sont représentées par des modèles plus abstraits.
Dans l'écrasante majorité des cas, les réseaux de robots sont des réseaux sans fil, nous nous intéresserons donc spécifiquement aux simulateurs performants dans la simulation des réseaux sans fil. Nous restreindrons de plus notre étude aux simulateurs open-source, actuellement maintenus et qui sont les plus utilisés pour la recherche. Certains simulateurs ont dû être écartés, comme Netsim [4] qui est une alternative payante et OPNET [5] qui est de moins en moins utilisé.

OMNeT++. OMNeT++ [26] est un simulateur à événements discrets modulaire dont la première publication date de 1997. OMNeT++ a la particularité de se présenter comme une *plateforme de simulation*, sur laquelle les équipes de recherche et les industriels peuvent construire leur propre *simulateur de réseau*. OMNeT++ fournit une interface graphique permettant de prototyper plus rapidement des réseaux grâce aux outils de visualisation.

Mininet. Mininet [6] est un émulateur de réseau léger et facile d'utilisation créé pour la simulation des approches dites *Software Defined Network* (SDN). Son extension Mininet-Wifi [12] permet d'émuler des réseaux sans fil SDN. Mininet-WiFi permet des expérimentations riches et facilement déployables grâce à une distribution par conteneurs Docker. Il se révèle très utile pour l'étude des SDNs et de leur protocole de routage phare, OpenFlow [20].

NS-3. NS-3 [22] remplace son prédécesseur NS-2, le simulateur de réseau qui était déjà le plus utilisé dans les années 2010. Bâti sur l'expérience de ce simulateur largement adopté par la communauté, NS-3 capitalise sur des années de contributions d'une communauté nombreuse et active et reste aujourd'hui le simulateur de réseau le plus utilisé dans la recherche. La grande force de NS-3 réside dans le nombre et la diversité de modules disponibles, qui permettent de simuler de nombreuses technologies sous des topologies variées sans avoir à produire beaucoup de code. Notons cependant que NS-3 manque

3. Intrinsic Acquires OSRC and OSRC-SG : https ://tinyurl.com/5eb7bxfb

4. Netsim™ : https ://netsim.boson.com/
5. OPNET : https ://opnetprojects.com/opnet-network-simulator/
6. Mininet : http ://mininet.org/

d'une interface graphique intuitive et permettant de simuler rapidement des cas simples.

Enfin, les standards de développement sévères imposés par NS-3 assurent que la documentation est à jour et que chaque module est scrupuleusement testé avant son intégration dans le simulateur. Ces règles permettent aux chercheurs d'accorder une grande confiance aux résultats des études menées dans NS-3.

3 Co-simulateurs existants

Comme nous l'avons vu ci-dessus, les systèmes muti-robots peuvent être simulés dans leur composante robotique d'un côté et réseau de l'autre, mais aucun des simulateurs mentionnés n'intègrent ces deux parties du spectre en un seul logiciel. Dans cette partie nous proposons un tour d'horizon des initiatives qui tentent de résoudre cette problématique.
Les chercheurs se sont tournés vers la co-simulation (l'assemblage d'outils de simulation) pour éviter de reprendre à zéro le développements des simulateurs robotique et réseau.
Le tableau 1 synthétise notre étude en donnant pour chaque co-simulateur : les simulateurs de robotique et de réseau sur lesquels il se base, la technique de synchronisation utilisée ainsi que la technologie de partage d'information entre les simulateurs. Bien qu'intéressants, les simulateurs AVENS [19] et CUSCUS [27] ne seront pas discutés en détail car ils ne sont plus maintenus et n'implémentent pas de synchronisation entre simulateurs de robotique et de réseau.

FlyNetSim. FlyNetSim [3] est un environnement de co-simulation dédié aux essaims de drones. Il utilise la pile logicielle Ardupilot [7] à laquelle il ajoute une simulation du réseau gérée dans NS-3. Un mécanisme de synchronisation bidirectionnelle y est implémenté : un timestamp est ajouté à chaque message et utilisé pour déterminer quel simulateur est "en avance" sur l'autre. On gèle alors l'exécution du simulateur le plus rapide pour attendre le simulateur le plus lent.
Les calculs de physique sont effectués dans un mini-simulateur de physique intégré à la version *Software-In-The-Loop* (SITL) d'Ardupilot, mais celui-ci n'est pas aussi avancé que les moteurs de physique utilisés dans des simulateurs comme Gazebo ou ARGoS.
FlyNetSim possède par ailleurs un mode "émulation" ou *Hardware-In-The-Loop* (HITL) dans lequel le programme de l'autopilote n'est plus exécuté par l'ordinateur mais directement sur le drone. Le réseau et les entrées des capteurs sont quant à eux toujours simulés. Cette capacité permet de passer rapidement de la phase de simulation à la phase d'expérimentation en milieu réel.

CORNET et CORNET 2.0. Introduit en 2020, CORNET [1] est un middleware de co-simulation capable d'interconnecter Gazebo et NS-3 pour le cas des flottes de drones. La deuxième version de ce logiciel, CORNET 2.0 [2], publiée en 2022, capitalise sur les leçons de cette première expérience. NS-3 y est remplacé par Mininet-WiFi et des améliorations majeures sont apportées : CORNET 2.0 n'est plus limité aux robots aériens, est indépendant des simulateurs de physique et de réseau utilisés, ne dépend pas de ROS et profite du déploiement conteneurisé de Mininet, permettant de distribuer efficacement les calculs sur plusieurs machines.
La synchronisation temporelle est gérée par un plugin Gazebo. Cette approche ne peut fournir qu'une synchronisation unidirectionnelle du simulateur de réseau vers le simulateur de physique, ce qui engendre une grande contrainte : il faut que le simulateur de réseau fonctionne à un rythme plus élevé que le simulateur de physique. Malheureusement, cette hypothèse ne se vérifie pas dans le cas de réseaux fortement dynamiques, très denses ou avec des calculs de propagation réalistes [17].

RoboNetSim. RoboNetSim [17] est un environnement de simulation intégrant le simulateur de physique ARGoS et les simulateurs de réseau NS-2 ou NS-3. Créé en 2013, ce co-simulateur dédié à l'étude des grands essaims de robots (plusieurs dizaines) propose une synchronisation bidirectionelle et un échange de message par socket. Des études de performances avec RoboNetSim ont montré les points suivants :

- NS-3 est plus performant (rapide) que son alter-ego NS-2 ;
- Le simulateur de physique (ici ARGoS) est le facteur limitant quand moins d'un certain nombre de robots communicants sont simulés (75 pour NS-2, 100 pour NS-3) ;
- Le sur-coût en temps de calcul de la co-simulation sont existants mais pas insurmontables. Pour 100 robots, le coût total de calcul est environ deux fois plus grand quand NS-3 est utilisé en parallèle d'ARGoS (ce qui ne veut pas dire que le temps total de la simulation est deux fois plus grand, puisque chaque simulateur s'exécute en parallèle) ;
- La co-simulation ne rajoute pas d'erreur : les résultats obtenus durant une co-simulation

7. Ardupilot : https ://ardupilot.org/

TABLE 1 – Tableau comparatif des co-simulateurs robotique et réseau existants

Nom	Simulateur Multi-Robot	Simulateur Réseau	Synchronisation	Echanges d'information	Année	Open-Source	Réf.
RoboNetSim	ARGoS	NS-2 / NS-3	Time-stepped (Bidirectionnelle)	Socket (TCP et UDP)	2013	Oui	[17]
FlyNetSim	Ardupilot	NS-3	Time-stepped (Bidirectionnelle)	Message queue (ZMQ)	2018	Non	[3]
CPS-Sim	Matlab Simulink	QualNet OMNeT++	Variable time-stepped (Bidirectionnelle)	Custom (SNSP)	2018	Non	[25]
GzUav	Gazebo	NS-3	Exécution séquentielle	Unix Domain Socket + socket TCP	2019	Oui	[8]
CORNET	Gazebo	NS-3	Variable time-stepped (Unidirectionnelle)	Message queue (ZMQ)	2020	Oui	[1]
ROS-NetSim	Tous	Tous	Sliding Window (Bidirectionnelle)	Unix Domain Socket	2021	Oui	[5]
CORNET 2.0	Gazebo	MiniNet	Variable time-stepped (Unidirectionnelle)	Message queue (ZMQ)	2022	Oui	[2]
SynchroSim	Gazebo	NS-3	Variable time-stepped (Bidirectionnelle)	Non précisé	2022	Non	[9]

sont les mêmes que les résultats fournis par chaque simulateur, indépendemment.

SynchroSim. SynchroSim [9] est un co-simulateur récent qui se focalise sur la simulation de systèmes multi-robot hétérogènes, avec l'objectif principal de simuler l'opération coordonnée et collaborative de robots terrestres et aériens sur le champ de bataille. Il utilise Gazebo et NS-3, ainsi que ROS (1).
La principale originalité de cette proposition réside dans sa méthode de synchronisation. SynchroSim introduit une synchronisation basée sur une fenêtre glissante à taille variable, ce qui règle en partie la problématique récurrente du choix de la taille de la fenêtre. La taille de la fenêtre est ajustée en fonction de la différence de vélocité entre deux agents, mais les auteurs ne justifient pas ce choix et ils n'apportent pas de preuve de justesse de leur simulation. Enfin, ce simulateur n'est pas *open-source*, rendant difficile une évaluation critique.

CPS-Sim. CPS-Sim [25] accorde Matlab Simulink (simulation physique) et au choix QualNet ou OMNeT++ (simulation réseau). Il se démarque des autres co-simulateurs par sa méthode de synchronisation par pas variable, qui permet des calculs plus précis au prix d'un temps de simulation élevé, en particulier quand le nombre d'événements réseau augmente (voir section 5). En revanche, le choix de Matlab/Simulink pour modéliser la physique prive CPS-Sim des avantages d'utilisation de simulateurs robotiques existants, comme une interface visuelle ou un contrôle direct des actionneurs.
Pour certains problèmes, la méthode de synchronisation de CPS-Sim apporte une justesse de résultats nécessaire. Par exemple, dans le cas de la simulation de protocoles de synchronisation d'horloges dans un réseau de capteurs sans fil, la précision temporelle est à la fois très importante et sensible à la moindre erreur. Cette méthode de synchronisation est donc adaptée.

GzUav. GzUav [8] est un environnement de co-simulation qui modélise une flotte de drones communiquants en assemblant trois modules complémentaires : Gazebo gère la simulation physique ; Ardupilot gère la simulation de l'autopilote (SITL) ; NS-3 simule les échanges réseau. Il permet une synchronisation bidirectionnelle et sa charge de travail peut être distribuée facilement sur plusieurs ordinateurs en réseau. L'intérêt de ce simulateur est l'intégration *by design* du code de l'autopilote, ainsi que son mécanisme de synchronisation par exécution séquentielle qui assure sa "justesse" au prix d'un temps de simulation élevé (la simulation physique avance d'un pas, attend que l'autopilote et la simulation réseau aient terminé leurs calculs, puis passe au pas suivant). Pour un réseau inférieur à 100 robots, GzUav montre des performances acceptables. S'il est très spécifique aux flottes de robots volants, ce simulateur se place parmi les co-simulateurs les plus avancés.

ROS-NetSim. L'ambition de ROS-NetSim [5] est de connecter un simulateur de physique avec un simulateur de réseau, peu importe les simulateurs choisis, pour n'importe quel scénario multi-robots, en se chargeant de la synchronisation bidirectionnelle et de l'échange de messages. Ce co-simulateur cherche à rester le plus général possible et utilise donc ROS pour s'adapter à tous types de robots. Sa méthode

de synchronisation est à fenêtre fixe et il utilise la technologie Protobuf [8] pour l'échange d'information entre les simulateurs, ce qui permet à l'utilisateur de facilement changer le contenu des échanges.
ROS-NetSim propose aussi une méthode d'abstraction des informations du canal radio : le simulateur de physique fournit au simulateur de réseau les informations nécessaires (chemins d'ondes multiples et pertes associées) pour calculer l'affaiblissement dû à la propagation.
Le grand intérêt de ROS-NetSim est sa transparence du point de vue du robot. Un robot ROS ne peut distinguer s'il évolue dans un environnement simulé ou réel. Quand il est dans le monde réel, son trafic réseau sera géré par sa carte réseau, alors que s'il est simulé, ce trafic sera intercepté, passera dans le simulateur de réseau puis sera réinjecté au nœud destinataire. Cette architecture permet des simulations HITL ainsi qu'une transition fluide vers des expérimentations en milieu réel.

4 Architecture et propriétés de co-simulateur

Les co-simulateurs mentionnés ci-dessus ont des architectures proches : un programme surmesure initialise les simulateurs robotique et réseau puis met en place des canaux de communication entre ceux-ci. Il s'assure ensuite que les simulateurs fonctionnent de manière synchronisée tout au long de la simulation. Une représentation de l'architecture-type est donnée dans la figure 2.
Les co-simulateurs mentionnés fournissent cependant des fonctionnalités différentes. Nous listons ici les fonctionnalités qui semblent utiles pour la simulation de systèmes multi-robots.

4.1 Hardware In The Loop

Étudier un nouveau protocole ou une nouvelle technologie sur une version simulée d'un robot (SITL) peut mener à des erreurs ou des biais. Effectivement, les limitations du robot réel (puissance de calcul, énergie limitée, chauffe des composants, etc.) doivent être prises en compte dans le développement. Un simulateur permettant le *Hardware In The Loop* (HITL) permet de surpasser ces limitations en exécutant le programme du robot directement sur le robot réel. Seuls les entrées (capteurs) et sorties (actionneurs) sont alors simulés. Ainsi, le plan de développement d'une nouvelle technique multi-robots est le suivant : d'abord, une expérimentation rapide en simulation, puis une validation du logiciel exécuté sur le robot mais avec un environnement simulé, puis une expérimentation en milieu réel qui valide définitivement notre nouvelle technique.

8. Protocol Buffer : https ://protobuf.dev/

4.2 Contrôle du temps

Il est intéressant d'avoir la possibilité d'enregistrer les expériences dans un format qui permet de rejouer la simulation, de la mettre sur pause ou de la jouer à l'envers. Ce format standard rend l'échange et la reproduction des expériences plus accessibles. La possibilité de rejouer les simulations avec un interface de visualisation est un plus non négligeable pour identifier rapidement les problèmes. Par exemple, pour les simulateurs fonctionnant autour de ROS, les *rosbags* sont des fichiers enregistrant l'ensemble des messages échangés pendant la simulation, permettant de rejouer le scénario exactement à l'identique.

4.3 Intégration ROS

Le *Robot Operating System* (ROS) est omniprésent dans la recherche en robotique aujourd'hui. Il sera donc essentiel que le co-simulateur accueille facilement des robots exécutant ROS. ROS présente une grande bibliothèque de modules (capteurs, algorithmes, ...) pouvant être facilement intégrés et interfacés avec un robot.

4.4 Performances

Enfin, les co-simulateurs seront *in fine* surtout évalués sur leur efficacité. Un co-simulateur rajoutant une surcharge de temps de calcul trop grande ne sera pas utilisé pour l'étude de systèmes multi-robots. Bien sûr, la performance d'un co-simulateur dépend directement des performances des simulateurs qu'il connecte. L'exécution d'une co-simulation ne pourra pas être plus rapide que le temps de calcul du plus lent des simulateurs (robotique ou réseau).

5 Synchronisation temporelle

La synchronisation des horloges entre plusieurs simulateurs est une problématique récurrente de la co-simulation. Les simulateurs de robotique décrivent la dynamique et cinématique des modèles grâce à des équations différentielles, qui doivent être discrétisées pour être implémentées de manière logicielle. Cette discrétisation

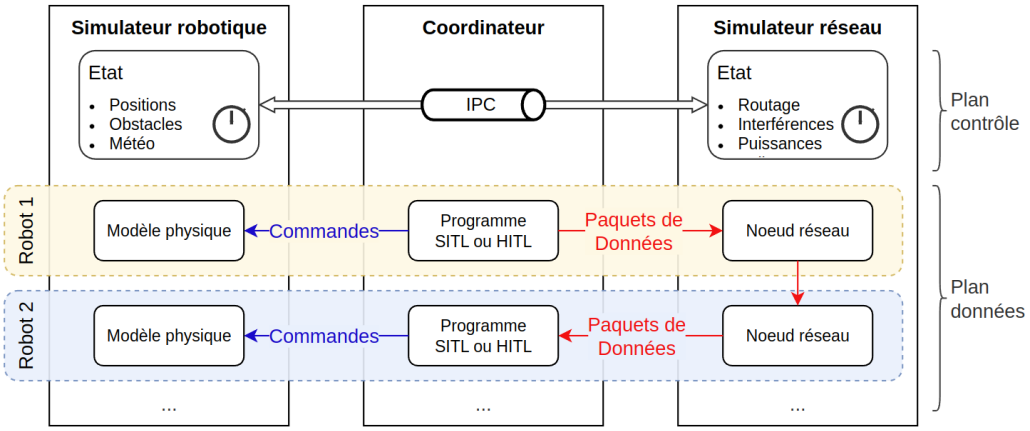

FIGURE 2 – Architecture représentative d'un co-simulateur robotique et réseau

est faite avec un temps d'échantillonnage, qui constitue la granularité de la simulation. Ces simulateurs sont dits *temps discret à pas constant* (Gazebo, ARGoS, Webots, etc.). Les simulateurs de réseau n'ont pas la contrainte d'un temps qui s'écoule régulièrement. Un nœud peut être silencieux pendant un temps long, puis déclencher une suite d'événements en décidant d'émettre un paquet. Dans ces simulateurs, le temps simulé avance d'événement en événement, le temps entre ceux-ci pouvant changer à loisir. Ce sont des simulateurs à *événements discrets* (NS-3, OMNeT++, QualNet, etc.)
Cette différence de nature rend nécessaire l'implémentation de mécanismes de synchronisation qui peuvent être unidirectionnels (un simulateur utilise l'horloge simulée de l'autre simulateur) ou bidirectionnels.
Cette section présente les techniques de synchronisation utilisées dans les co-simulateurs existants. Une représentation graphique de ces techniques est donnée en figure 3.

Time-stepped. C'est l'approche la plus simple : la synchronisation a lieu régulièrement avec une période multiple de la durée du pas du simulateur de physique (sur la figure 3, cette période est de 5 pas). A chaque synchronisation, les simulateurs échangent leurs statuts et variables puis avancent indépendamment jusqu'à la prochaine synchronisation. Cette méthode est simple à implémenter mais présente deux problèmes affectant la justesse des calculs : un délai injustifié est introduit pour le simulateur de robotique et les informations de mobilité fournies au simulateur de réseau sont imprécises.
Le premier effet s'explique facilement : le simulateur de robotique est notifié de la réception d'un message seulement lors de la prochaine synchronisation. Il reçoit toujours les paquets avec un délai qui dépend de la taille de la fenêtre de synchronisation. Si les paquets arrivent de manière uniforme et aléatoire, le délai moyen introduit est donc de la moitié de la fenêtre de synchronisation (vérifié en pratique dans [17]).
Le deuxième problème est dû à la fréquence des mises à jour des positions dans le simulateur de réseau. En effet, les robots sont considérés immobiles durant toute la fenêtre de simulation, ce qui peut entraîner des imprécisions dans les calculs du simulateur de réseau. Cependant, cet effet ne représente en pratique que de faibles imprécisions [17].

Variable time-stepped. Pour améliorer la précision des calculs, il est possible d'adapter la méthode précédente en brisant la linéarité du simulateur de physique. La procédure est alors la suivante : si le simulateur de réseau possède un événement pendant la prochaine fenêtre du simulateur de physique, ce dernier se met sur pause. Le simulateur de réseau calcule l'événement puis envoie au simulateur de physique les informations de synchronisation et le *timestamp* de l'événement. Le simulateur de physique doit alors avancer jusqu'à ce *timestamp*. Avec cette méthode, le simulateur de physique obtient les informations des activités du réseau sans délai, ce qui représente un avantage dans les applications où le réseau et la robotique sont très intriqués. En revanche l'implémentation de pas variables dans le simulateur de physique se révèle souvent techniquement difficile. Les performances de cette méthode dépendent beaucoup de l'application étudiée : s'il y a peu d'événements réseau, les performances sont bonnes car il y a peu de synchronisation et donc peu d'échanges de messages. En revanche avec un grand nombre d'événements réseau, la surcharge de temps de calcul augmente dramatiquement.

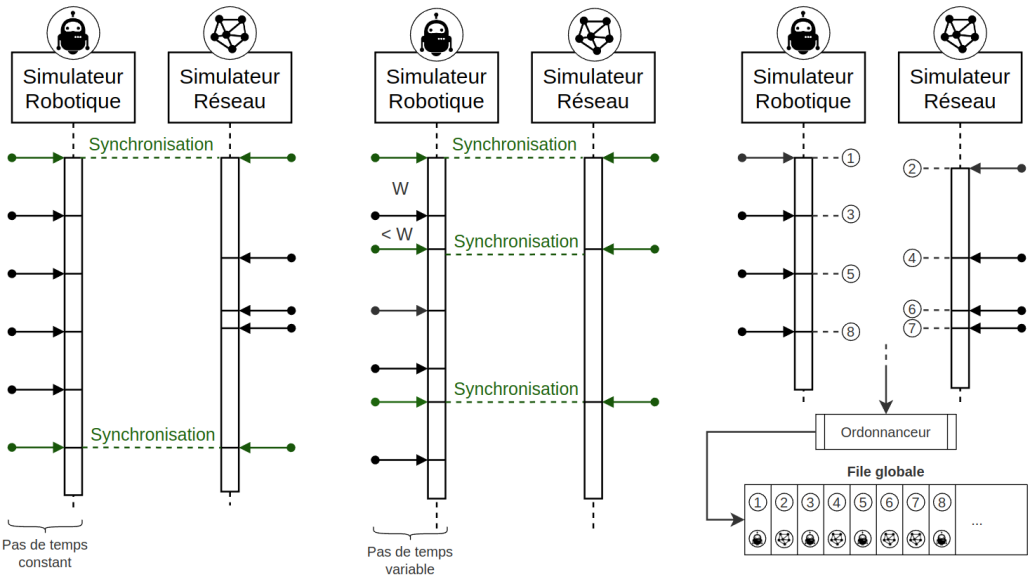

FIGURE 3 – Méthodes de synchronisation de simulateurs de physique et de réseau. de gauche à droite : *time-stepped*, *variable time-stepped*, *global event-driven*

Global Event-driven. Une approche radicalement différente consiste à gérer les deux simulateurs depuis un ordonnanceur unique. Chaque pas du simulateur de physique est alors traité comme un événement et mis dans la même file d'attente que les événements du simulateur de réseau. Les événements sont exécutés de manière séquentielle. L'avantage majeur de cette méthode est qu'elle n'engendre pas de délai supplémentaire ni d'approximations de calcul. En retour, les événements des deux simulateurs n'étant pas calculés simultanément, le coût en temps de calcul est décuplé.

6 Flux d'information entre les simulateurs

6.1 Technologie d'échange

Les simulateurs s'échangent des informations tout au long de la simulation, au minimum pour se synchroniser mais aussi pour collaborer. Ces échanges de messages représentent l'essentiel de la surcharge de calcul générée par la co-simulation, il est donc important de choisir une technologie de communication judicieusement. Le tableau 1 précise les techniques de communication utilisées dans les co-simulateurs étudiés.

Si détailler les avantages et inconvénients de chacune de ces techniques sort du cadre de ce papier, nous discutons ci-après les techniques utilisées dans les co-simulateurs que nous avons étudiés. Le choix de la technique de communication entre simulateurs sera avant tout déterminé par l'exécution distribuée du co-simulateur : s'il doit pouvoir s'exécuter sur plusieurs ordinateurs en réseau (pour partager la charge de calcul), l'utilisation de *sockets* Internet (UDP ou TCP) sera presque automatique. Dans le cas contraire, les autres techniques de communication inter-processus (*Inter-Process-Communication*, IPC) peuvent s'avérer plus efficaces.
Les *sockets* sont probablement la technique la plus versatile. Ils peuvent être utilisés localement ou en réseau (permettant des calculs distribués), sont indépendants de la plateforme (permettant l'exécution sur différents systèmes d'exploitation) et proposent des communications bloquantes ou non-bloquantes (ce qui rend possible la synchronisation des horloges).
Les files de messages présentent moins de charge de calcul que les *sockets* mais dépendent du système d'exploitation et peuvent être problématiques quand le nombre de message est grand.
La mémoire partagée ne peut pas être utilisée si les calculs doivent être distribués sur plusieurs machines, mais représente la technique la plus efficace en temps de calcul. Cependant, cette technique requiert une organisation rigoureuse des accès en mémoire, complexifiant le développement du co-simulateur.

6.2 Contenu des échanges

Le contenu des échanges entre les simulateurs relève aussi une importance déterminante. Il est crucial de correctement définir les informations

échangées : partager trop d'information générera une surcharge du temps de calcul, mais en partager trop peu limite l'intrication des simulateurs, rendant le co-simulateur inutile. Par exemple dans notre cas, il est primordial que les positions des robots soient cohérentes entre la simulation de physique et la simulation de réseau. Le concepteur du co-simulateur doit donc répondre à deux questions essentielles : quel est le contenu des messages échangés entre les deux simulateurs, et à quelle fréquence ces échanges ont-ils lieu ?
Le problème de l'abstraction du canal radio est un exemple intéressant dans le cas des simulations de systèmes multi-robots. Pour déterminer la réussite de réception d'un paquet entre deux robots, le simulateur de réseau doit calculer un bilan de liaison et comparer la puissance reçue au seuil de réception du robot destinataire. Pour cela, il doit détenir une représentation du canal radio (distance, obstacles, interférences, etc.), qui soit cohérente avec la représentation du monde dans le simulateur de physique. Pour faire passer ces informations du simulateur de physique au simulateur de réseau, [5] propose une abstraction du canal qui cherche à compresser les informations essentielles nécessaires au calcul du bilan de liaison.
La question du niveau de détail nécessaire à la simulation des réseaux sans fil revêt en réalité une grande complexité. [15] étudie cette question en profondeur et montre que c'est la question scientifique qui doit dicter la finesse de la simulation. Dans certains cas, un haut niveau d'abstraction permettra d'accélérer la simulation et d'éviter que les résultats propres à la question posée ne soient pollués par les artefacts d'une simulation trop précise. Dans d'autres cas, une simulation trop abstraite peut empêcher le chercheur de déceler les problèmes que son nouvel algorithme ou nouveau protocole pourrait rencontrer.

7 Conclusion

Dans le cadre des systèmes multi-robots communicants, nous avons présenté la nécessité de simuler simultanément la physique et le contrôle des robots d'un côté, et les flux d'information traversant le réseau de l'autre. Une étude rapide des simulateurs de robotique et de réseau existants a montré qu'il était essentiel de bénéficier des années de développement dans chacun de ces domaines. Les équipes de recherche se sont donc tournées vers la co-simulation et nous en avons présenté les initiatives les plus pertinentes. Ces projets ont chacun leurs avantages, mais aucun n'a pour l'instant fait consensus dans la communauté multi-robots. En particulier, le maintien de ces plateformes et leur compatibilité avec les outils modernes (ROS, Ubuntu, etc.) est à notre sens une barrière difficile à surmonter.

Travaux en cours Dans le cadre du projet ANR/AID CONCERTO, nous avons débuté le développement d'un environnement de co-simulation robotique et réseau adapté aux flottes de robots aériens. Après une étude approfondie des co-simulateurs existants, nous avons choisi d'utiliser ROS-NetSim, Gazebo et NS-3. Les principes de simulation apportés par ROS-NetSim sont les plus concluants. Sa grande flexibilité et l'intégration à ROS jouent aussi en sa faveur. Cependant, ROS-NetSim n'est qu'une preuve de concept et son intégration réelle avec des simulateurs de robotique et de réseau nécessite le développement de connecteurs appropriés. Nos choix de Gazebo et NS-3 ont été essentiellement motivés par la présence d'une communauté active et nombreuse qui accélérera le développement de notre co-simulateur. Nous avons commencé par la modernisation de ROS-NetSim (passage à ROS 2, changements mineurs) puis nous avons développé un coordinateur permettant de connecter la dernière version de Gazebo au co-simulateur. Les prochaines étapes de nos travaux comprendront donc l'implémentation d'une simulation de réseau sans fil basé sur la technologie Wi-Fi dans NS-3 et l'intégration d'un programme d'auto-pilote (PX4) permettant de réaliser des simulations SITL réalistes. Enfin, la mise en place d'une expérience simple mettant en scène le réseau et le déplacement des drones permettra de valider la pertinence de notre plan de recherche en trois temps : simulation SITL ; simulation HITL ; déploiement en milieu réel.

Remerciements

Ce travail a été financé par l'Agence de l'Innovation de Défense et l'INSA Lyon et dans le cadre du projet CONCERTO (ANR-20-ASTR-0003).

Références

[1] S. Acharya et al. "CORNET : A Co-Simulation Middleware for Robot Networks". In : COMSNETS. 2020.

[2] S. Acharya et al. "CORNET 2.0 : A Co-Simulation Middleware for Robot Networks". In : *COMSNETS*. 2022.

[3] S. Baidya et al. “FlyNetSim : An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot”. In : *Proceedings of the ACM MSWIM*. 2018.

[4] İlker Bekmezci et al. “Flying Ad-Hoc Networks (FANETs) : A survey”. In : *Ad Hoc Networks* (1er mai 2013).

[5] M. Calvo-Fullana et al. “ROS-NetSim : A Framework for the Integration of Robotic and Network Simulators”. In : *IEEE Robotics and Automation Letters* (2021).

[6] P. Cieślak. “Stonefish : An Advanced Open-Source Simulation Tool Designed for Marine Robotics, With a ROS Interface”. In : OCEANS MTS/IEEE. 2019.

[7] J. Collins et al. “A Review of Physics Simulators for Robotic Applications”. In : *IEEE Access* (2021).

[8] F. D’Urso et al. “An integrated framework for the realistic simulation of multi-UAV applications”. In : *Computers & Electrical Engineering* (2019).

[9] E. Dey et al. “SynchroSim : An Integrated Co-simulation Middleware for Heterogeneous Multi-robot System”. In : DCOSS. 2022.

[10] E. J. F. Dickinson et al. “COMSOL Multiphysics® : Finite element software for electrochemical analysis. A mini-review”. In : *Electrochemistry Communications* (2014).

[11] A. Farley et al. “How to pick a mobile robot simulator : A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion”. In : *Simulation Modelling Practice and Theory* (2022).

[12] R. R. Fontes et al. “Mininet-WiFi : Emulating software-defined wireless networks”. In : International CNSM. 2015.

[13] B. P. Gerkey et al. “The Player/Stage Project : Tools for Multi-Robot and Distributed Sensor Systems”. In : *Proceedings of the ICAR*. 2003.

[14] Samira Hayat et al. “Survey on Unmanned Aerial Vehicle Networks for Civil Applications : A Communications Viewpoint”. In : *IEEE Communications Surveys & Tutorials* (2016).

[15] J. Heidemann et al. “Effects of Detail in Wireless Network Simulation”. In : USC/ISI. 2001.

[16] M. H. Kabir et al. “Detail Comparison of Network Simulators”. In : *IJSER* (2014).

[17] M. Kudelski et al. “RoboNetSim : An integrated framework for multi-robot and network simulation”. In : *Robotics and Autonomous Systems* (2013).

[18] M. M. M. Manhães et al. “UUV Simulator : A Gazebo-based package for underwater intervention and multi-robot simulation”. In : OCEANS MTS/IEEE. 2016.

[19] E. A. Marconato et al. “AVENS - A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System”. In : Hawaii International Conference on System Sciences. 2017.

[20] N. McKeown et al. “OpenFlow : enabling innovation in campus networks”. In : *ACM SIGCOMM Computer Communication Review* (2008).

[21] C. Pinciroli et al. “ARGoS : a modular, parallel, multi-engine simulator for multi-robot systems”. In : *Swarm Intelligence* (2012).

[22] G. F. Riley et Thomas R. Henderson. “The ns-3 Network Simulator”. In : *Modeling and Tools for Network Simulation*. Springer, 2010.

[23] E. Rohmer et al. “V-REP : A versatile and scalable robot simulation framework”. In : IEEE/RSJ IROS. 2013.

[24] S. Shah et al. “AirSim : High-Fidelity Visual and Physical Simulation for Autonomous Vehicles”. In : Springer Proceedings in Advanced Robotics. 2018.

[25] A. Suzuki et al. “CPS-Sim : Co-Simulation for Cyber-Physical Systems with Accurate Time Synchronization”. In : IFAC NECSYS. 2018.

[26] A. Varga et Rudolf Hornig. “An overview of the OMNeT++ Simulation Environment”. In : ICST. 2010.

[27] N. R. Zema et al. “CUSCUS : An integrated simulation architecture for distributed networked control systems”. In : IEEE CCNC. 2017.