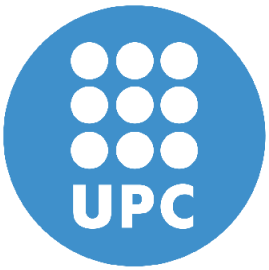


Ubiquitous supercomputing

Design and development of enabling technologies for multi-robot systems

Rethinking Supercomputing

LEONARDO CAMARGO FORERO

Systems Engineer

Master of science in ubiquitous computing and networking

Advisors

DR. XAVIER PRATS I MENÉNDEZ

DR. PABLO ROYO CHIC

Doctorate program in Aerospace Science and Technology

Technical School of Castelldefels

Technical University of Catalonia

Programa de doctorado en Ciencia y Tecnologia Aeroespacial

Escola Politécnica Superior de Castelldefels (EPSC)

Universitat Politécnica de Catalunya (UPC)

A dissertation submitted for the degree of

Doctor of Philosophy

September 2019

Ubiquitous supercomputing
Design and development of enabling technologies for multi-robot systems
Rethinking Supercomputing

Author
Leonardo Camargo Forero

Advisors
Dr. Xavier Prats i Menéndez
Dr. Pablo Royo Chic

Thesis committee
Dr. Eduard Ayguade Parra
Dr. Lino Marques
Dr. Joan Vila

Doctorate program in Aerospace Science and Technology
Technical University of Catalonia
September 2019

PhD. Thesis made in:
Technical School of Castelldefels
Esteve Terradas, 5.
08860 Castelldefels
Catalonia (Spain)

A mi familia,
con quien está mi corazón.

Contents

List of Figures

List of Tables

List of publications

The list of publications resulting from this PhD. work is given in inverse chronological order as follows:

Journal papers

- CAMARGO-FORERO, LEONARDO, ROYO, PABLO & PRATS, XAVIER. The ARCHADE: Ubiquitous Supercomputing for robotics. Part II: Experiments. *Robotics and Autonomous Systems*. Under revision after a first peer-review.
- CAMARGO-FORERO, LEONARDO, ROYO, PABLO & PRATS, XAVIER. 2019 (Apr). The ARCHADE: Ubiquitous Supercomputing for robotics. Part I: Philosophy. *Robotics and Autonomous Systems*. Vol 114. pp. 187-198. DOI: https://doi.org/10.1016/j.robot.2019.01.006.
- CAMARGO-FORERO, LEONARDO, ROYO, PABLO & PRATS, XAVIER. 2018 (Sep). Towards High Performance Robotic Computing. *Robotics and Autonomous Systems*,Vol 107. pp. 167-181. DOI: https://doi.org/10.1016/j.robot.2018.05.011.
- ZAMANI, MARYAM, CAMARGO-FORERO, LEONARDO, & VICSEK, TAMAS. 2018 (Feb). Stability of glassy hierarchical networks. *New Journal of Physics*, Vol. 20. DOI: https://doi.org/10.1088/1367-2630/aaa8ca.

Conference proceedings

- CAMARGO-FORERO, LEONARDO, ROYO, PABLO & PRATS, XAVIER. 2018 (Sep). High Performance Robotic Computing as an enabler for cooperative flights. *In: Proceedings of the 37th IEEE/AIAA Digital Avionics Systems (DASC) Conference*. London (UK). **Best paper in track award.**

Book chapters

- CAMARGO-FORERO, LEONARDO, ROYO, PABLO & PRATS, XAVIER. 2017. On-board high-performance computing for multi-robot aerial systems. *Aerial Robots - Aerodynamics, Control and Applications*, Chap 7, P. 1-19. IntechOpen. Lopez-Mejia, Omar Dario & Escobar-Gomez, Jaime Alberto (eds). ISBN: 978-953-51-5357-3.

Agradecimientos

Cuando era un niño, me la pasaba usando *armatodos*, así los llamamos en Colombia, mundialmente conocidos como Legos, aunque de una diferente marca, para armar naves espaciales y robots. Mi madre Adriana escuchaba todas mis historias, mis fantasías de crear nuevas tecnologías, juguetes y todo tipo de inventos, la mayoría sin un propósito específico más que el simple placer de crear cosas nuevas.

Mientras crecía me interesé en múltiples campos de la ciencia, de la metafísica, la espiritualidad, fascinado por cuanta similitud puede observarse entre la arquitectura de un sistema multi-planetario y un simple átomo, como todo pareciera conectado de alguna manera, aún imperceptible para la humanidad y sobre todo como entidades independientes cooperan entre sí y parecen actuar como si fueran una sola. Quizás por lo que estoy más agradecido, es con el hecho de que mi tesis doctoral mezcla de alguna manera muchos de mis sueños desde que era un niño.

Sin embargo, por años no tuve la oportunidad de trabajar con robots o vehículos inteligentes, de hecho me enfoqué en supercomputación, por lo cual estoy muy agradecido también, porque me permitió trabajar en diferentes campos, óptica, matemáticas, bioinformática, etc., pero gracias a Colciencias, el Departamento Administrativo de Ciencia, Tecnología e Innovación del gobierno Colombiano y la beca que me otorgaron para mis estudios de doctorado, pude mezclar mis pasiones en una sola cosa, The ARCHADE, una tecnología para crear sistemas en los que supercomputación y robótica se integran para ejecutar todo tipo de misiones. De hecho, la beca que se me otorgó para mi doctorado se basó en una entrevista en la que expuse mi idea de unir la supercomputación con sistemas multi-vehículo, multi-robot.

No fue fácil estar lejos de mi familia por mucho tiempo, de mi familia por la cual late mi corazón, pero las palabras de mi padre Gerardo, de mi madre y la increíble capacidad de mi hermano Nicolás de sonreír, no importando en qué situación se encuentre, me dieron la fuerza para continuar siguiendo mis sueños y mucho más importante, mi deseo de contribuir al desarrollo de quizás uno de los más hermosos lugares del mundo, Colombia.

Esta tesis es para mi familia, para mi papá con su mente poderosa y su auto disciplina, su bondad, su amor por todos nosotros, para mi mamá, la persona más noble que he conocido en toda mi vida, la más hermosa, para mi hermano Nicolás, mi mejor amigo en todo el planeta, la persona más fuerte y mi aliado más inteligente con un gran corazón, para mi hermano Juan Diego, con quien espero compartir toda la vida, para mi país, para el amarillo, al azul y el rojo en todos lados y con la esperanza de que The ARCHADE facilite la creación de todo tipo de nuevas

aplicaciones que contribuyan a un mejor mundo. También agradezco a mi cotutor Xavier Prats por su apoyo, su conocimiento y la elegancia que demuestra al vivir lo que debe ser un doctor de la filosofía y por supuesto a mi cotutor Pablo Royo, quien, con su conocimiento técnico y práctico, la paz que inspira y su gran corazón me han enseñado tanto y para todas las personas, amigos, amores, conocidos, compañeros, colegas, que he tenido el privilegio de conocer en toda mi vida y por aquellos que aún no conozco. También agradezco al Dr. Eduard Ayguade Parra, al Dr. Lino Marques y al Dr. Joan Vila por aceptar ser parte de mi presentación final y compartir su vasta experiencia y conocimiento. Finalmente, me queda sola una persona a quien agradecer, un solo ser, con muchos nombres, muchos dogmas, muchas perspectivas, Dios, muchas gracias por todo.

Resumen

La supercomputación, también conocida como Computación de Alto Rendimiento (HPC por sus siglas en inglés) puede encontrarse en casi cualquier lugar (ubicua), desde el pequeño widget en tu teléfono diciéndote que hoy será un día lluvioso o uno soleado, hasta la mayoría de medicinas anunciadas en los medios de comunicación o vendidas en la farmacia local, pasando por el diseño de los vehículos que te llevan rápidamente de un país a otro y que eventualmente te llevarán a otros planetas e incluyendo la siguiente gran contribución al entendimiento de los orígenes del universo, de nuestra genética y aquella de todas las especies, de la naturaleza de la realidad, incluso la de nuestra propia conciencia. La lista TOP 500 de supercomputadores en el mundo y los incontables ejemplos fuera de la lista, evidencian el largo alcance de la supercomputación y su utilización en prácticamente todo aspecto de nuestra vida.

Sin embargo, hay un campo en el que la supercomputación ha sido apenas explorada - la robótica. Además de algunos intentos de optimizar tareas robóticas complejas, las dos fuerzas carecen de una alineación efectiva y de un contrato a largo plazo. Dado los avances en miniaturización, comunicaciones y la aparición de potentes computadores embebidos, optimizados en peso y energía, la siguiente transición lógica corresponde a la creación de un *cluster de robots*, un conjunto de entidades robóticas que juntas se comporten de manera similar a como lo hace un supercomputador. No obstante, hay un aspecto clave, con respecto a nuestra comprensión actual de lo que significa o para qué es útil la supercomputación, que este trabajo pretende redefinir. Durante décadas, la supercomputación ha sido entendida únicamente como un mecanismo de eficiencia computacional, es decir para reducir el tiempo de computación de ciertos problemas, que sin la supercomputación no se podrían resolver en un tiempo razonable. Si bien esta línea de pensamiento ha conducido a innumerables hallazgos, la supercomputación es más que eso, porque para proporcionar una infraestructura con la capacidad de resolver todo tipo de problemas rápidamente, se debe proporcionar otro conjunto completo de características (*características de la supercomputación*) que también pueden ser explotadas en contextos como la robótica. Propósito general, escalabilidad, heterogeneidad, transparencia de usuario, cooperación, etc., conjuntamente conduciendo a la *cohesión*, es decir, un conjunto de entidades independientes que actúan como si fueran una sola.

Esta tesis doctoral pretende repensar lo que realmente significa la supercomputación y diseñar estrategias para establecer de manera efectiva su inclusión dentro del mundo de la robótica, contribuyendo así a la ubicuidad de la supercomputación, el principal ideal de este trabajo. Con esto en mente, se presentará un estado del arte relacionado con intentos anteriores de

mezclar robótica y HPC, seguido de la propuesta de Computación Robótica de Alto Rendimiento (HPRC, por sus siglas en inglés), un nuevo concepto, potencialmente campo de las ciencias computacionales, que mapea la supercomputación a los matices específicos de sistemas multi-robot. HPRC puede pensarse como una *supercomputación en el borde* y si bien este enfoque proporcionará todo tipo de ventajas, para ciertas aplicaciones podría no ser suficiente, en ciertas aplicaciones se requerirá o deseará una interacción con infraestructuras externas. Para facilitar dicha interacción, esta tesis propone el concepto de *supercomputación ubicua* como la unión de HPC, HPRC y dos tipos más de entidades, dispositivos sin computación embebida (por ejemplo, redes de sensores, etc.) y seres humanos.

Para pasar de la filosofía a la realidad, los resultados de esta tesis incluyen una ontología, es decir un conjunto de conceptos interrelacionados que describen el alcance completo de la supercomputación ubicua y una tecnología llamada The ARCHADE (TAC). La tecnología consta de cinco componentes principales y sirve como middleware entre una misión y una infraestructura de supercomputación (HPC, HPRC, etc.) y como un framework para facilitar la ejecución de cualquier tipo de misión, por ejemplo, agricultura de precisión, entretenimiento, inspección y monitoreo, topografía y cartografía, ingeniería civil, etc. Además, las ideas detrás de la supercomputación ubicua y The ARCHADE se usaron para diferentes experimentos y aplicaciones tales como simulaciones de enjambres de pájaros, cluster HPRC de aeronaves, un demo llamado *Tigres versus Cazadores* y en una serie de misiones reales llevadas a cabo con dos rovers.

Al integrar la supercomputación y la robótica, un segundo ideal de esta tesis se presenta, *robótica ubicua*, es decir el uso de robots en todo tipo de aplicaciones. Correspondientemente, una revisión de frameworks existentes relacionados con robótica ubicua será discutida. El diseño y desarrollo de The ARCHADE ha seguido las pautas y sugerencias encontradas en dicha revisión. Además, The ARCHADE se basa en una supercomputación repensada donde la eficiencia computacional no es la única característica proporcionada a sistemas basados en la tecnología. Sin embargo, se analizarán indicadores de eficiencia computacional, junto con otros indicadores relacionados con las demás características de la supercomputación.

La supercomputación ha sido un excelente aliado para la exploración científica y no hace mucho tiempo, también para múltiples actividades comerciales, conduciendo a todo tipo de mejoras en nuestras vidas, en nuestra sociedad y en nuestro futuro. Con los resultados de esta tesis, la unión de dos campos, dos fuerzas previamente desconectadas debido a sus enfoques filosóficos y sus antecedentes divergentes, tiene un enorme potencial para abrir nuestra imaginación para todo tipo de aplicaciones nuevas y para un mundo donde la robótica y la supercomputación estén en todos lados.

Abstract

Supercomputing, also known as High Performance Computing (HPC), is almost everywhere (ubiquitous), from the small widget in your phone telling you that today will be a rainy or a sunny day, up to most medicines advertised in the media or being sold in the local pharmacy, passing by the design of the vehicles that take you fast from one country to another and that will eventually take you to other planets and the next great contribution to the understanding of the origins of the universe, of our genetics and that of every species, of the nature of reality, even that of our consciousness. The TOP 500 supercomputers in the world and the countless examples out of the prominent list, evidence supercomputing's large scope and its utilization in practically every aspect of our modern lives.

However, there is a field where supercomputing has been only slightly explored - robotics. Other than attempts to optimize complex robotics tasks, the two forces lack an effective alignment and a purposeful long-term contract. With advancements in miniaturization, communications and the appearance of powerful, energy and weight optimized embedded computing boards, a next logical transition corresponds to the creation of *clusters of robots*, a set of robotic entities that can behave similarly as a supercomputer does. Yet, there is key aspect regarding our current understanding of what supercomputing means, or is useful for, that this work aims to redefine. For decades, supercomputing has been solely intended as a computing efficiency/performance mechanism i.e. decreasing the computing time for a task, that without supercomputing, could not be succeeded in reasonable time. While such train of thought have led to countless findings, supercomputing is more than just that, because in order to provide an infrastructure with the capacity of solving most problems quickly, another complete set of features must be provided, a set of features (*supercomputing features*) that also can be exploited in contexts such as robotics. General-purpose, scalability, heterogeneity, user-transparency, cooperation, etc., all together leading to cohesion, i.e. a set of independent entities acting as if they were one.

This Ph.D thesis aims at rethinking what supercomputing actually means and to devise strategies to effectively set its inclusion within the robotics realm, contributing therefore to the ubiquity of supercomputing, the first main ideal of this work. With this in mind, a state of the art concerning previous attempts to mix robotics and HPC will be outlined, followed by the proposal of High Performance Robotic Computing (HPRC), a new concept, potentially computer science field, mapping supercomputing to the nuances of multi-robot systems. HPRC can be thought as supercomputing in the edge and while this approach will provide all kind of advantages, in

certain applications it might not be enough, in certain applications interaction with external infrastructures will be required or desired. To facilitate such interaction, this thesis proposes the concept of *ubiquitous supercomputing* as the union of HPC, HPRC and two more type of entities, computing-less devices (e.g. sensor networks, etc.) and humans.

To go from philosophy into reality, the results of this thesis include an ontology i.e. a set of interacting concepts describing the complete scope of ubiquitous supercomputing and an enabling technology depicted as The ARCHADE (TAC). The technology consists of five main components and it serves as a middleware between a mission and a supercomputing infrastructure (HPC, HPRC, etc.) and as a framework to facilitate the execution of any type of mission, i.e. precision agriculture, entertainment, inspection and monitoring, surveying and mapping, civil engineering, etc. Furthermore, the ideas behind ubiquitous supercomputing and The ARCHADE have been used for a set of experiments and applications, such as the simulation of birds' swarming, HPRC cluster of aircraft, a demo called *Tigers vs. Hunters* and a set of real missions carried out with two rovers.

By integrating supercomputing and robotics, a second ideal is targeted, ubiquitous robotics, i.e. the use of robots in all kind of applications. Correspondingly, a review of existing ubiquitous robotics frameworks is presented and based upon its conclusions, The ARCHADE's design and development have followed the guidelines for current and future solutions. Furthermore, The ARCHADE is based on a rethought supercomputing where performance is not the only feature to be provided by TAC-enabled ubiquitous supercomputing systems. However, performance indicators will be discussed, along with those related to the remaining supercomputing features.

Supercomputing has been an excellent ally for scientific exploration and not so long ago for commercial activities, leading to all kind of improvements in our lives, in our society and in our future. With the results of this thesis, the joining of two fields, two forces previously disconnected because of their philosophical approaches and their divergent backgrounds, holds enormous potential to open up our imagination for all kind of new applications and for a world where robotics and supercomputing are everywhere.

List of acronyms

AC	ArduCopter
AES	Advanced Encryption Standard
AI	Artificial Intelligence
AIXM	Aeronautical Information Exchange
ALU	Arithmetic Logic Unit
ANH	Aircraft Network Hierarchy
ANSP	Air Navigation Service Provider
API	Application Programming Interface
APV	ArduPilot Vehicle
A&R	Agents & Roles
AR2	APMrover2
ATC	Air Traffic Controller
ATFCM	Air Traffic Flow and Capacity Management
ATM	Air Traffic Management
BADA	Base of Aircraft Data
BDT	Business Development Trajectory
BVLOS	Beyond Visual Line of Sight
CAGR	Compound Annual Growth Rate
CDR	Conflict Detection and Resolution
CLD	Computing-less devices
CONOPS	Concept of Operations
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DDR2	Demand Data Repository 2
DKD	dkdrone class
DKR	dkrover class
DSP	Digital Signal Processors
DTN	Delay Tolerant Network
E	Entity
EB	Exabytes
ECAC	European Civil Aviation Conference
FABEC	Functional Airspace Block Europe Central

FL	Flight Levels
FLOPS	Floating Point Operations Per Second
FPGA	Field-Programmable Gate Array
FPS	Frames Per second
FR	Free Route
GB	Gigabytes
GFLOPS	Giga FLOPS
GPCM	General-Purpose Computing Mission
GPGPU	General Purpose Computing on GPUs
GPS	Global Positioning System
GPU	Graphics Processing Unit
GRC	Global Reaching Centrality
H	Human Entity
HARC	High Availability Robotic Cluster
HDFS	Hadoop Distributed File System
HITL	Hardware In The Loop
HN	Hierarchy Network
HPC	High Performance Computing
HPCC	High Performance Computing Cluster
HPL	High-Performance Linpack
HPRC	High Performance Robotic Computing
IoT	Internet of Things
INT	Interface
ITU	International Telecommunications Union
KPA	Key Performance Area
KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
LRC	Local Reaching Centrality
LTE	Long-Term Evolution
MIC	Message Integrity Code
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MN	Master node
MPI	Message Passing Interface
NextGen	The Next Generation Air Transportation System
NFS	Network File System
NM	Nautical Mile
NOP	Network Operations Plan
OD	Orders delay
OOP	Object-Oriented Programming
OpenCL	Open Computing Language
OpenCV	Open Computer Vision
OpenRAVE	The Open Robotics and Animation Virtual Environment
Orocos	Open robot control software
P2P	Peer-to-Peer
PM	Physical Machine
PR	Proximity Radius
PRCN	Parallel Robotic Computing Node
RBT	Reference Business Trajectory
REAL	Real mode
RM	Real Mode

ROS	Robot Operating System
RPI3B	Raspberry Pi 3 model B
RTOS	Real-Time Operating System
SAR	Search And Rescue
SBT	Shared Business Trajectory
SimPlat	The ARCHADE simulation platform
SIM	Simulated mode
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SITL	Software In The Loop
SLA	Service Level Agreement
SLAM	Simultaneous Localization and Mapping
SC	System Controller
SC-E	System Controller entity
SESAR	Single European Sky ATM Research
SH	System Hierarchy
SO	System Operator
SoC	System on a Chip
SN	Slave Node
SWIM	System Wide Information Management
TAC	The ARCHADE
TB	Terabytes
TBO	Trajectory-Based Operations
TOD	Top Of Descent
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
UbiSL	Ubiquitous Supercomputing Language
UGV	Unmanned Ground Vehicle
UN	United Nations
UV	Unmanned Vehicle
VM	Virtual Machine
WEB	Wired Equivalent Privacy
WAN	Wide Area Network
WPA	Wi-Fi Protected Access
YOLO	You Only Look Once
ZB	Zettabytes

Ahora, si por algún motivo pudiera escuchar mi voz luego de pasar una vida entera sin conocerla, diría para mi mismo, es una carga innecesaria? , o un regalo, o caería irremediablemente en una revelación?, al notar que mi voz es tan diferente de mis pensamientos así como las palabras son tan diferentes de la verdad

— Leonardo CF

I

Introduction

A robot should be more than a specific-purpose machine, used only for a particular task, designed and developed with specific hardware and software and consequently difficult to integrate with other robots or reuse for a different purpose.

If a robot were to become a *general-purpose computing unit*, no significantly different from a computer or a cellphone, it could be reused for different missions and multi-robot integration would become transparent. The advantages of rethinking what a robot is, are endless. In fact, the robotics community, at research and industrial level, is transitioning from an era, where robots were made with single-purpose, exclusive hardware and software, towards a new one where scalability and integration are of the upmost importance. Perhaps, the greatest evidence of such transition is the appearance of embedded computing cards (*companion computers*), optimized in terms of cost, size, weight, energy, etc., which facilitates easy integration empowered by the use of traditional operating systems.

Since a robot's companion computer is installed with an operating system, all kind of software can be adapted in order to enhance the capabilities of the robot and integration between multiple robots becomes a straightforward endeavor via mainstream protocols such as TCP/IP. Consequently, a set of connected robots has the potential of becoming a *supercomputing infrastructure*.

Supercomputing, formally known as High Performance Computing (HPC), is a powerful tool devised to optimize, in terms of efficiency i.e. computing time, the execution of complex software. Software applications used to predict the weather, understand the origins of the universe, create incredibly realistic Sci-Fi movies, manufacture new pharmaceutical drugs and vaccines, extract information from our genetics and that of any species, model aircraft wings, simulate new

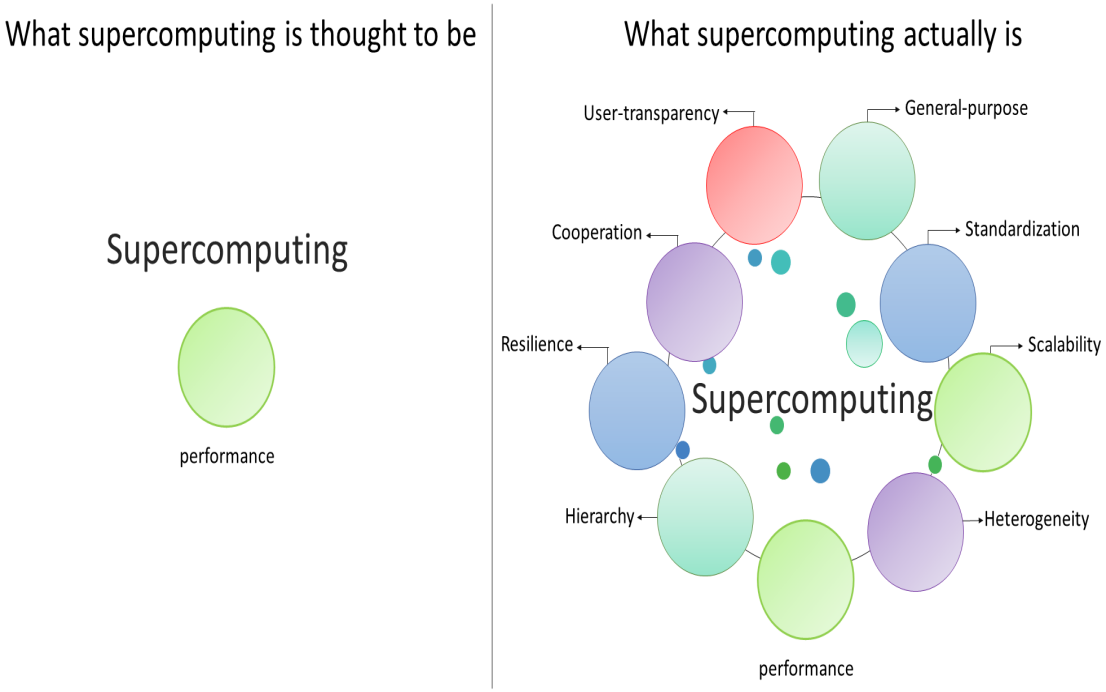


Figure I-1: *What supercomputing actually is. The small circles represent potential supercomputing features, not discussed in this thesis*

propulsion techniques, oil and gas exploration, space research, send personalized advertisement to millions of users worldwide and much much more are all examples of the supercomputing's long reach. Applications that without supercomputing, cannot find applicable results in reasonable time.

While supercomputing was conceived and it has been used with the sole intent of decreasing computing time for complex tasks, in this thesis, the main core idea requires a change of mind, one in which supercomputing means much more than only computing-efficiency as portrayed in Figure I-1. In fact, the term *super*, from its Latin roots, means *above* or *beyond*, which does not necessarily imply only strength or performance, or more precisely, it could mean much more.

With this in mind, this thesis aims at redefining what supercomputing is, general-purpose, standardization, scalability, heterogeneity, user-transparency, cooperation, resilience, hierarchy, performance, etc., all together leading to *cohesion* i.e. *a tool to create systems made up of independent entities that behave as a single unit*.

I.1 Motivation

The scope upon which supercomputing is utilized is large, almost everywhere. In fact, nowadays it is difficult to find an economic or social activity in which supercomputing does not have a relevant impact. A main driver can be identified for such ubiquity, *Massive amount of data*. According to CISCO (2018):

- Annual global data center IP traffic will reach 20.6 Zettabytes (ZB) (1.7 ZB per month) by the end of 2021, up from 6.8 ZB per year (568 exabytes [EB] per month) in 2016.
- Global data center IP traffic will grow 3-fold over the next 5 years. Overall, data center IP traffic will grow at a Compound Annual Growth Rate (CAGR) of 25 percent from 2016 to 2021.

- By 2021, data center storage installed capacity will grow to 2.6 ZB, up from 663 Exabytes (EB) in 2016, nearly a 4-fold growth.
- Globally, the data stored in data centers will nearly quintuple by 2021 to reach 1.3 ZB by 2021, up 4.6-fold (a CAGR of 36%) from 286 EB in 2016.
- Big data will reach 403 EB by 2021, up almost 8-fold from 25 EB in 2016. Big data alone will represent 30 percent of data stored in data centers by 2021, up from 18 percent in 2016.
- The amount of data stored on devices will be 4.5 times higher than data stored in data centers, at 5.9 ZB by 2021.
- Driven by the Internet of Things (IoT), the total amount of data created (and not necessarily stored) by any device will reach 847 ZB per year by 2021, up from 218 ZB per year in 2016. Data created is two orders of magnitude higher than data stored. However, approximately 10 percent of such data will be actually useful.
- Useful data exceeds data center traffic (21 ZB per year) by a factor of four. Edge or Fog computing might help bridge this gap

Based on Cisco indicators, it is important to conclude that local storing and computing (i.e. at the edge) is the clearest alternative to solve the gap between the amount of generated data and the storage capacity. Even, in cases where data still needs to be transferred externally, preprocessing and filtering, i.e. selecting only useful data, could play an important role to be done at the edge.

A particular type of IoT device, of importance for this thesis are robots, specifically Unmanned Vehicles (UV), whose range of possible applications is vast, from simple aerial photography and video to highly complex missions in fields such as precision agriculture, inspection and monitoring, surveying and mapping, civil engineering, military missions, etc. Specifically, Unmanned Aerial Vehicles (UAVs) missions generate enormous quantities of data e.g. 200 Gigabytes (GB) of digital imagery per day with fixed-wing UAVs (Krest, 2017) or around 70 Terabytes (TB) of data in a 14-hour mission carried out by a Gorgon Stare drone (Abhishek, 2014) or 150 TB of data per day with a small UAV fleet (Logist, 2017).

Furthermore, the UAV applications industry is expected to generate 100 billion USD between 2016 and 2020 (Castellano, n.d.) and to grow at a CAGR of 53.9% during 2018-2023 (Reportlinker, 2018). These previous indicators lead to an important conclusion, data processing is a necessity. Two approaches can be taken into consideration when processing data, specifically IoT produced data as within the context of this thesis, *local* and *remote* approach. The former advocates to exploit locally-available computing resources and storage in order to optimize response-time, reliability, decrease bandwidth consumption, avoid high latencies (e.g. caused by Wide Area communications - WAN, see Satyanarayanan (2010)), as proposed by Edge computing (Shi *et al.* , 2016). Contrary, the latter advocates the use of external computing and storage resources e.g. cloud computing or other remote infrastructures, given that local resources do not have sufficient power for specific applications or missions requiring it. Several works in robotics have contemplated the use of cloud computing resources. For example in Benavidez *et al.* (2015), the authors proposed a cloud mechanism for aiding a robot in visual SLAM (Simultaneous Localization and Mapping). Even, a Robot as a Service platform has been proposed (Chen *et al.* , 2010).

It is worth clarifying that the Edge computing paradigm does not exclude the utilization of external computing resources. In fact, both approaches are valid, depending on the application's particular requirements. This leads to the conclusion that it is important to define mechanisms and strategies for computing-load distribution, either at the edge or in external computing resources, or both, if required. For example, Hu *et al.* (2012) proposed machine-to-cloud and machine-to-machine strategies to facilitate computing loading over the cloud and the edge correspondingly.

Therefore, the foundation bricks for hybrid approaches have been laid out, which gives confidence into going further by including supercomputing in the equation.

I.1.1 What supercomputing is thought to be

HPC on-demand services over the cloud are a reality nowadays, e.g. Amazon (n.d.) and works such as Khan *et al.* (2018) and Trancoso & Efstathiou (2017) have studied the use of supercomputing over the edge, in terms of energy and cost, i.e. using low-cost low-energy computing devices, such as SoCs (System on a Chip), for supercomputing.

With constant improvements in computing power, energy consumption, etc., SoCs, such as cellphone processors, Raspberry Pi (Raspberry-Pi-Foundation, n.d.), NVIDIA Jetson Series (NVIDIA, n.d.a), etc., are a very attractive platform for supercomputing. Real or pseudo real-time, latency-sensitive, information-sensitive or remote-locations-with-no-Internet-access applications are suitable candidates for the use of HPC with SoCs. Moreover, complete HPC clusters based on SoCs have been deployed (Cox *et al.* , 2014) and results suggests that SoCs are good candidates for traditional HPC (Rajovic *et al.* , 2013), even using OS-level HPC-supported virtualization (Beserra *et al.* , 2017).

On the robotics side, many works, to be referenced in chapter II section II.1, have implemented traditional supercomputing in robotics settings, mostly with single robots. Furthermore, standard HPC technologies, such as Message Passing Interface (MPI) (Walker & Dongarra, 1996) or Compute Unified Device Architecture (CUDA) (Garland *et al.* , 2008) have been used to optimized the execution of complex robotic tasks.

Traditional supercomputing, i.e. aiming at computing-efficiency, with SoCs, has been explored in the past. However, other supercomputing features, that could potentially lead to all kind of new applications, have been left behind. Moreover, regarding robotics, previous works have exploited strategies belonging mostly to the applications layer, while a supercomputing infrastructure (e.g. a *HPC cluster of computers*) is composed of more layers, *the HPC software layers*, as it can be observed in Figure I-2, devised with the objective of providing features such as scalability, user-transparency, etc.

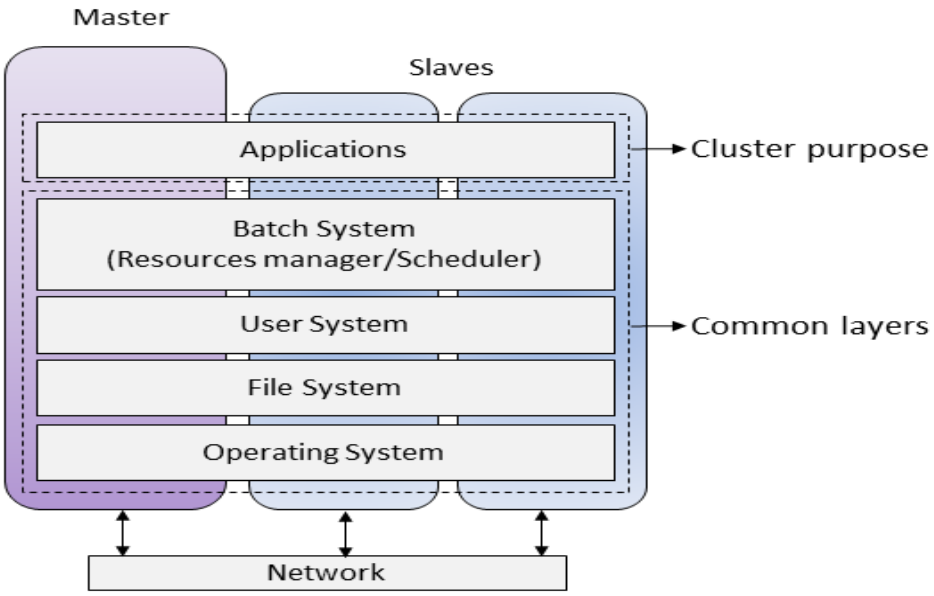


Figure I-2: *HPC software layers. A HPC cluster of computers is the most common supercomputing infrastructure. Nodes in a HPC cluster are classified as master and slaves*

I.1.2 What supercomputing actually is

This thesis aims at rethinking what supercomputing actually is (Figure I-1), beyond only performance or computing-efficiency, and to set strategies to apply its features in the context of multi-robot systems, bringing truly everywhere i.e. *Ubiquitous supercomputing*. In this sense, while performance measurements will be discussed, it is important to emphasize that the approach in here, differs from all previous attempts of using supercomputing in robotics or IoT settings.

Consider for example scalability. A supercomputing infrastructure is scalable in two fronts. First, adding or removal of computing nodes (slaves) is fairly transparent. Second, software executing over a supercomputing infrastructure, provided that it is written with standard technologies (e.g. MPI), can run in parallel, over any quantity of computing cores, distributed across multiple nodes. Correspondingly, scalability in multi-robot systems can be exploited in two fronts. First, transparent adding and removal of robotic entities, e.g. for optimizing area recognition, energy consumption, low cost, etc. Second, parallel software running over any quantity of computing cores, distributed across multiple companion computers, *robotic nodes*. However, such software is not necessarily expensive in terms of computing power but it still can benefit from the scalability feature.

Generally speaking, supercomputing was designed with the objective of targeting certain problems, with the idea of increasing the assigned computing power and therefore reduce its computational time. And it basically worked, we can now solve, in reasonable time, problems that could take millennia to be solved without supercomputing. In fact, countless activities in our daily lives make use of supercomputing on a regular basis Chabowski (2016), NVIDIA (n.d.h) .

However, in order to provide a tool that could effectively speed up any problem (*general-purpose*), another set of features had to be implemented. Such features, the *supercomputing features* are:

- *General-purpose*: Supercomputing can be used for anything, the TOP500 supercomputers in the world (Dongarra *et al.* , n.d.) are evidence of it. The applications layer in a HPC cluster of computers, sets the infrastructure's purpose, i.e. the software installed on such layer can be used for all kind of user cases. Such software must make use of a HPC software technology, e.g. MPI, in order to be able to run over multiple computing units that are distributed across different nodes. Software written without such technologies will only run on single nodes. Furthermore, the other HPC software layers are common in any HPC cluster. If a single robot or a multi-robot system is thought as a general-purpose computing unit, its applications are endless.

- *Standardization*: Supercomputing makes uses of standard technologies in all the HPC software layers. Several operating systems, mostly Linux-based, can be set upon nodes belonging to a HPC cluster, e.g. CentOS, RedHat, SUSE, Debian, Ubuntu, Scientific Linux, etc. Examples of standard technologies used in supercomputing are: Network File System (NFS) (Shepler *et al.* , 2003) for the *File System* layer, Open Lightweight Directory Access Protocol (Open LDAP), an implementation of the LDAP standard (Zeilenga, 2006), for the *User System* layer, Torque (Adaptive-Computing, n.d.b) for the *Batch System* layer, etc. Furthermore, MPI is the most used HPC software technology but other technologies exist as well, for example OpenMP (Dagum & Menon, 1998), OmpSs (Duran *et al.* , 2011) or CUDA for general-purpose computing on Graphics Processing Units (GPUs), in the applications layer. If a single robot or a multi-robot system is embedded with standard software technologies, integration among robots, even those carrying different purposes, becomes straightforward. For example, a set of robots, each one embedded with its own Linux-based computing board can be used to execute MPI parallel software, exploiting the total distributed computing units.

- *Scalability*: Supercomputing allows transparent integration of nodes. In a HPC cluster, nodes are classified as *Master* and *Slaves*. The master node implements the server-side of each software solution in the HPC software layers. Correspondingly, the slaves implement the client-side. Integration of new slaves can be done by installing the corresponding HPC software layers (client-side) and connect them to the master node. A scalable multi-robot system can find many uses, i.e. by simply splitting a mission area or the mission's tasks and even scaling inwards, by including more nodes, configured as a HPC cluster inside single robots in order to increase their local computing power.

- *Heterogeneity*: Supercomputing allows integration of different vendors, operating systems, computing-resources (e.g. quantity and type of CPUS, GPUs, other coprocessors, etc.), into the same infrastructure. However, for simplicity and error-avoidance, it is advisable to set the same operating system in all the nodes belonging to a supercomputing infrastructure. In the robotics world, heterogeneous systems can be used depending of their individual or group peculiarities. For example, ground and air vehicles can cooperate with each other by sharing information only accessible in their own medium. Moreover, different kind of equipment and devices can be embedded in robotics entities.

- *User-transparency*: The master node, is also the front-end machine, to which the users interact with. Among several responsibilities, the master node masquerades the distributed nodes as if they were a single computing unit i.e. the user does not need to know the underlying complexity in the supercomputing infrastructure. Moreover, a supercomputing infrastructure provides a multi-user/multi-purpose environment. Commonly, data is stored locally only in the master node, whilst the slaves access it via the file system. Similar approaches can be applied with software in the applications layer, i.e. existing locally only in the master node. This facilitates managing procedures, licenses control, etc. Furthermore, user replication is managed with the user system. A single pilot controlling several unmanned vehicles or multi-user multi-robot systems, where each robot is piloted by a different user but yet under the control of a single person, opens the spectrum of robotics applications.

- *Cooperation*: Cooperation is achieved in the applications layer, where software, running in distributed computing units, cooperates into solving specific problems. Such cooperation must be explicitly program in the applications layer's software. Multi-robot systems cooperating with human entities and other type of elements, can result into all kind of applications. For example, a set of robots can split an area, individually assess health in a particular crops plantation by running locally parallel software, while together estimating the total health of the area, information that yet can be improved by human interpretation, extra sensors, etc.

- *Resilience*: A distributed infrastructure is prone to errors, possibly occurring in different entities. Different approaches are taken into consideration to guarantee resilience in supercomputing infrastructures. For example, a second master node (*shadow master*) is set upon the infrastructure, commonly in active-passive or active-active, with load-balancing configurations. Furthermore, software checkpointing can be implemented as well e.g. Ansel *et al.* (2009). Upon individual robotics failures, a supercomputing multi-robot system can still accomplish its mission, by redistributing its specific tasks or to guarantee the mission's consistency, up to such point of failure, by constantly synchronizing data with a master entity.

- *Hierarchy*: A supercomputing infrastructure holds different hierarchy schemes. From a platform point of view, hierarchy is as follows: user, master, shadow master and slaves. Furthermore, the batch system can be configured to prioritize certain users or groups when requesting computing resources. With the rise of artificial intelligence, it is very important to maintain hierarchy, one where humans are still the apex entities.

- *Performance/Computing-efficiency*: Finally, it is valid to consider performance as an emergent property of the previous supercomputing features.

The previous features are a direct result of the implementation of the HPC software layers and they represent what supercomputing actually is. With this in mind, this thesis aims at revising the idea that supercomputing is all about performance, as massively observed in the literature, and to adapt concepts, strategies and features from a *rethought supercomputing*, into the world of multi-robot systems, in order to transform a set of distributed entities into a *single cohesive unit*, exposing all the previous mentioned features.

I.2 Objectives

With the objective of adapting traditional supercomputing in the context of multi-robot systems, this thesis proposes a new concept called *High Performance Robotic Computing - HPRC*. The term *performance* in HPRC is kept in order to simplify its adoption. However, HPRC inherits the same frame of mind, where its scope is beyond solely computing-efficiency. However, in order to support all kind of missions, even beyond current computing limitations with companion computers, HPRC does not deny the use of external computing resources, i.e. traditional HPC infrastructures or cloud services. Therefore, *Ubiquitous supercomputing*, under the scope of this thesis is defined as the union of traditional HPC (performance-oriented), HPRC, computing-less devices (CLD), i.e. those not embedding computing boards and Humans (H).

Furthermore, with the objective of bringing ubiquitous supercomputing everywhere (*ubiquitous*), a framework and middleware called *The ARCHADE (TAC)* will be designed, developed and tested during the execution of this thesis.

To sum up, the objectives of this PhD thesis can be outlined as follows:

- Give a thorough review of the state of the art in the use of traditional High Performance Computing in robotics settings.
- Formally define the new concept of High Performance Robotic Computing.
- Formally define ubiquitous supercomputing through the creation of an ontology and a set corresponding concepts.
- Design and develop a ubiquitous supercomputing framework and middleware
- Test the ubiquitous supercomputing framework and middleware with a set of simulated and real missions.

I.3 Scope and limitations

Supercomputing, as defined by this thesis, can indeed provide computing-efficiency, despite that it is not its main objective. Thus, a deep computing-efficiency analysis is out of the scope of this work. Nevertheless, some basic metrics will be calculated and discussed.

Deployed TAC systems will be composed of seven entities in simulation mode and four entities in real mode. Moreover, a ubiquitous supercomputing system composed of 256 entities, in simulated mode, will be presented. However, systems with higher quantities of entities in real mode will not be evaluated given hardware availability limitations.

Furthermore, only UGVs (Unmanned Ground Vehicles) will be used. Unfortunately, there was no UAVs (Unmanned Aerial Vehicles) availability during the period of the real tests. However, the tested autopilot firmware is fairly transparent disregarding the type of unmanned vehicle, as confirmed by simulation experiments.

High Performance Robotic Computing clusters, just as traditional HPC clusters, rely on network communications. Current advancements in communications technologies e.g. 5G (ITU, n.d.) have a strong potential to provide the necessary features for applications requiring performance and constant message exchange. Therefore, issues related to connectivity and network communications are out of the scope of this thesis. However, Wi-Fi latency and signal strength, with HPRC clusters while in motion, will be discussed.

I.4 Outline

The material in this document is organized in six Chapters, which are summarized as follows:

- **Chapter II** presents the state of the art of traditional HPC in robotics, coupled with relevant background (section II.1). In addition, this chapter includes a literature review related to ubiquitous robotics frameworks and middleware (section II.2). Furthermore, a discussion is portrayed in section II.3.
- **Chapter III** starts with the definition of HPRC and all the necessary adaptations for traditional HPC towards multi-robot systems (section III.1). Afterwards, Ubiquitous supercomputing is described in its whole extent, including its ontology and main concepts (section III.2). Following, the ideas behind general-purpose computing missions (section III.3), the ubiquitous supercomputing language (section III.4), hierarchy (section III.5), stability (section III.6) and automation (section III.7) are discussed.
- **Chapter IV** introduces and discusses The ARCHADE and all its capacities and components. Specifically, The ARCHADE's API (section IV.1), its framework component (section IV.2), its middleware component (section IV.3), The ARCHADE's simulation platform (section IV.4) and the PLUS component (section IV.5).
- **Chapter V** presents and discusses the results of a set of experiments and missions carried out under the scope of ubiquitous supercomputing. Specifically, *performance* (section V.1), *swarming* (section V.2), *HPRC cluster of aircraft* (section V.3), *Tigers vs. Hunters* (section V.4) and *complete missions* (section V.5), where the latter includes a benchmark composed of analysis for CPU load, RAM usage, latency, etc.
- **Chapter VI** summarizes the contributions of this thesis, discuss concluding remarks (section VI.1) and points out future work (section VI.2).

Cuando aquellos vidrios rotos cayeron al suelo, la pequeña ave sintió el dolor de saberse sola en medio de la nada, en la oscuridad, así que cerró los ojos, abrió las alas y esta vez voló con el viento a su favor sin detenerse, hasta que al pasar la ultima nube, comprendió que debía tener los ojos cerrados para no verse a sí misma y poder ver la luz del sol

— Leonardo CF

II

State of the art

Robotics are following a path towards ubiquity, one filled with many promises and even more potential, inspired by advancements in embedded computing cards, artificial intelligence (AI), the IoT era and all kind of new applications, fields, business models, etc. Specially, unmanned vehicles are a promising research and industrial field as it was introduced in section I.

In this chapter, a state of the art, regarding the use of High Performance Computing (computing-efficiency oriented) - HPC, in the context of robotics is presented. Traditionally, HPC and robotics have not been common companions. The reason of such separation can be understood in two fronts. First, classic robots did not have on-board computers as the ones available nowadays, i.e. such computers were not installed with a mainstream operating system and they were very limited in terms of computing power (e.g. CPU speed, RAM, etc.) and more importantly they were single-purpose, i.e. devised for a specific unchangeable solution. Second, HPC infrastructures are composed of either commodity computers or state-of-the-art servers, often with multi-core CPUs or even many-core computing cards such as GPUs. Moreover, HPC infrastructures are by default multi-purpose/multi-user and ultimately research backgrounds in robotics and HPC significantly differ.

However, attempts to *join the two forces* have been made and section II.1 will explore it with a corresponding HPC background. Furthermore, this chapter discusses the ideas of ubiquitous robotics and existing software solutions. It is important to clarify the differences between *ubiquitous computing*, *ubiquitous robotics* and *ubiquitous supercomputing for robotics*, as described in this thesis. However, while ubiquitous supercomputing differs from the concepts of ubiquitous robotics and ubiquitous computing, as it will be discussed in section II.2, its objectives can be effectively aligned. Furthermore, a comparison between existing ubiquitous robotics frameworks and The ARCHADE is given in the same section.

II.1 HPC in robotics

High Performance Computing (HPC) can be defined as a tool for speeding up the computation of complex tasks or to be able to analyze large data sets in reasonable time, a feat that would not be possible in common computing infrastructures such as personal computers, neither with sequential, i.e. non-parallel, software.

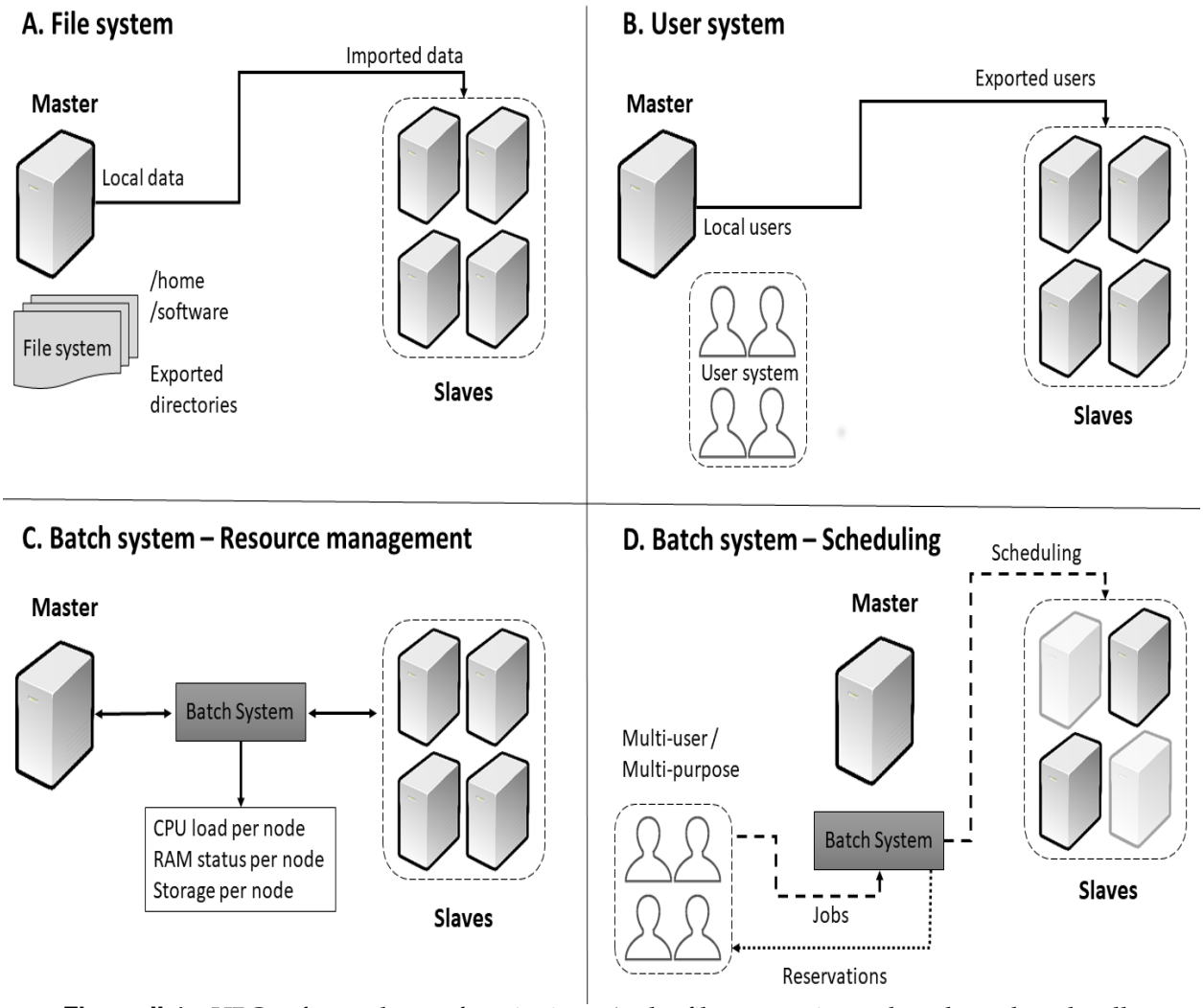


Figure II-1: *HPC software layers functioning. A. the file system is used to share data, locally stored in the master node. B. The user system is used to export the master's local users. C. The batch system monitors resources such as CPU load, RAM, etc in all the nodes, including itself. D. The batch system implements efficient scheduling policies for computing-power sharing in a multi-user environment*

The most common supercomputing infrastructure is known as a HPC cluster of computers. A HPC cluster of computers consists of a set of computers (nodes), classified as master and slaves, connected via a Local Area Network (LAN), in which a set of software layers (recall the *HPC software layers* shown in Figure I-2) are installed and configured in each of its nodes. Such software layers provide or have the potential of providing the features described in section I.1.2. The HPC software layers, from bottom to top, are: the *operating system (OS)*, the *file system*, the *user system*, the *batch system* and the *applications* layer. HPC clusters differentiate from each other by the applications layer i.e. the rest of the layers are similar in all clusters. Furthermore, HPC nodes are typically connected via high-speed low-latency technologies such as Gigabit Ethernet, Infiniband, etc. Figure II-1 provides a general overview of the HPC software layers and a detailed explanation

is provided as follows:

- *Operating system*: Looking to ease administration tasks, it is advisable to implement the same operating system, in all the machines within a HPC cluster of computers. However, heterogeneous installations are possible nonetheless. Historically, Linux-based operating systems, have been the most used in HPC infrastructures. However, Microsoft provides HPC solutions as well (MICROSOFT, n.d.).
- *File system*: The file system is used for data replication among the computing nodes and for the creation of *single-image systems*, i.e. data is accessible from every point in the infrastructure. In almost all cases, data will be ultimately in the master node or in a Network Attached System (NAS) connected to it. However, technologies such as GlusterFS (GLUSTERFS, n.d.), Hadoop Distributed File System (HDFS), (Borthakur, 2008) etc., provide distributed file systems.
- *User system*: HPC infrastructures are commonly used by many users, which compete for the available resources, for different type of purposes (*multi-purpose/multi-user*). Moreover, parallel software requires user accounts to be replicated in the entire infrastructure. User replication is therefore handled by the user system. The most common HPC file system is Open LDAP (Zeilenga, 2006). It provides a complete grouping and other features scheme, very useful in a multi-user environment.
- *Batch system*: (Resource Manager/Job scheduler): In a multi-user environment, it is important to control access to resources. If multiple users execute software over the same computing resources, at the same time, performance will be negatively impacted. To manage exclusivity, the batch systems assigns reservations. A reservation is the response to a user-requested job i.e. *a request for a set of computing resources (CPU cores, RAM, etc) during a time window (walltime)*. This process is specifically handled by the *job scheduler*, one of the two components of the batch system. Moreover, a complete hierarchy and prioritization scheme can be configured in order to maximize the computing resources utilization and their corresponding accessibility, while satisfying user demands. Furthermore, the other component of the batch system, the *resource manager* is in charge of monitoring indicators such as CPU load, RAM usage, user-access, network communications, etc. Multiple batch systems solutions are available, both open source such as Open Grid Scheduler (OGS, n.d.), PBS Torque (Adaptive-Computing, n.d.b), Simple Linux Utility for Resource Management (SLURM) (SchedMD, n.d.) or proprietary such as MOAB (Adaptive-Computing, n.d.a), etc.). The batch system is the HPC cluster middleware.
- Applications: Depending on the *HPC cluster's purpose* e.g. Oil & Gas exploration (TOTAL, n.d.), genetics research (ORNL, n.d.), etc., software in the applications layer may vary.

Table II-1: *Flynn's taxonomy. Parallel software: SIMD, MISD, MIMD*

Acronym	Classification	Description
$SISD$	Single Instruction Single Data	*Sequential software* i.e. it can only be executed by one single computing unit
$SIMD$	Single Instruction Multiple Data	*Data parallelism* e.g. GPGPU, CUDA
$MISD$	Multiple Instruction Single Data	*Redundant software* i.e. to guarantee results correctness
$MIMD$	Multiple Instruction Multiple Data	*Distributed software* e.g. MPI. Most general case

The uppermost layer, consists mostly on software to be executed upon the HPC cluster and extra libraries required to do so. Moreover, software is classified, according to Flynn's taxonomy (Flynn, 1972) as portrayed in Table II-1 and detailed as follows:

- Single Instruction Single Data (SISD): *Sequential software* is such that is executed only upon one computing unit (i.e. core). However, nowadays most applications implement some level of parallelism given that mainstream CPUs already embedded multiple cores.
- Single Instruction Multiple Data (SIMD) or *Data parallelism* consists of applying the same instruction or set of instructions to different pieces of data. A basic example will be the sum of two vectors, with the sum itself as the single instruction and the different vectors positions, the multiple data. The common feature of this classification is that *data-independence* exists, i.e. individual instructions, over different data, do not depend on each other. Moreover, SIMD does not consider message exchange between parallel processes, which can lead to better performance indicators than software requiring inter-process communication, almost approaching Amdahl's law (Amdahl, 1967) i.e. having N computing cores and t the time required to solve the problem in one single computing unit (*sequential time*), the *parallel time* will be t/N. Furthermore, *General Purpose Computing on GPUs (GPGPU)* is a powerful example of data parallelism. While a GPU was originally conceived for graphics processing exclusively, the large amount of embedded ALUs (Arithmetic Logic Unit), in comparison with standard CPUs, and technologies such as CUDA (Garland *et al.* , 2008) have transformed the GPU in a general-purpose HPC device, used in all kind of applications (NVIDIA, n.d.h), where fine-grain parallelism (massive parallelism) results advantageous.
- Multiple Instruction Single Data (MISD): Certain applications require to guarantee that their results are correct, for example in critical or sensitive missions. To do so, MISD propose to use multiple computing units to process the same data. If results remained the same in all cases, its correctness can be assumed. Examples of MISD can be found in aeronautics, space missions and even in projects such as SETI@HOME (Anderson *et al.* , 2002), the Search for Extraterrestrial Intelligence.
- Multiple Instruction Multiple Data (MIMD): Also known as *Distributed Computing*, corresponds to the most general case of parallelism and therefore the remaining classifications can be understood as MIMD specializations. It consists of applying multiple instructions to multiple data and it is generally based on message exchange between parallel processes.

For more information and technical details regarding High Performance Computing, please refer to the books Hager & Wellein (2010) and Sterling *et al.* (2017).

Motivated by the desired of speeding-up complex robotic tasks, such as Simultaneous Localization and Mapping (SLAM), detection and tracking, visual-based landing, etc., and advancements in embedded computing cards, robotics researchers have lightly flirted with HPC. For example, Salamí San Juan *et al.* (2015) showed results of the implementation of two parallel software, hot spot and jellyfish detection on board of a Unmanned Aerial Vehicle (UAV), aiming at fast responsiveness.

By processing raw data on board, instead of downloading it for post-processing, it is possible to send GPS coordinates and level of emergency to firefighters in order to prevent the occurrence of new fires after an initial fire. Also, by quickly detecting jellyfish location, it is possible to asses a fast response, instead of downloading the data, which after processing might be irrelevant, since the animals are in constant motion. The authors compared the execution of the software on different computing processors (EPIA N700-15 VIA C7-1.5, Pandaboard with OMAP 4430 integrated CPU, Intel T7500 on GENE-9655 motherboard and TilenCore-Gx board with TILE-Gx36), evaluating parallelization techniques and mainstream OS support.

Multiple companies provide multi-core computing boards for UAV integration. To mention a few, Raspberry Pi Foundation, NVIDIA, Intel, ASUS, etc. A set of examples is given in Table II-2

Table II-2: *Computing infrastructures for Unmanned Vehicles. Updated list: January 2019*

Computing board	CPU/GPU	Memory [GB]	Storage [GB]	Reference
Raspberry Pi 3 B+	1.4 GHz Quad-Core ARM CPU	1	N/A[1]	RPiFound (n.d.a)
NVIDIA Tegra K1	2.3 GHz Quad-Core ARM CPU 192-core Kepler GPU	2	16	NVIDIA (n.d.i)
NVIDIA Jetson TK1[2]	2.3 GHz Quad-Core ARM CPU 192-core Kepler GPU	2	16	NVIDIA (n.d.e)
NVIDIA Jetson TX1	1.73 GHz Quad-Core ARM CPU 256-core Maxwell GPU	4	16	NVIDIA (n.d.d)
NVIDIA Jetson TX2	2 GHz Quad-Core ARM CPU 2 GHz Dual-core Denver2 CPU 256-core Pascal GPU	8	32	NVIDIA (n.d.d)
NVIDIA Jetson AGX Xavier	2.26 GHz 28-core ARM CPU 512-core Volta GPU	16	32	NVIDIA (n.d.c)
Intel Aero compute	2.56 GHz Quad-Core ATOM	4	32	INTEL (n.d.a)
Odroid H2	2.5 GHz Quad-Core Intel Intel UHD Graphics 600 700Mhz	32	N/A	ODROID (n.d.)
ASUS Tinker	1.8 GHz Quad-Core Rockchip Mali T760 GPU, 2GB RAM	2	N/A	ASUS (n.d.)
Rock64 Media	1.5GHz Quad-Core Rockchip A53 Mali-450 MP2 GPU	1-4	N/A	Pine64 (n.d.)
PocketBeagle	1GHz Octavo OSD3358 ARM	0.5	N/A	PocketBeagle (n.d.)
Huawei HiKey 960	2.3/1.8 GHz 4+4-core ARM Mali G71 MP8 GPU	3	32	Huawei (n.d.)

Even though the previous Table II-2 is not a comprehensive list, including all existing UV embedded computing boards, it shows the market's direction towards smart autonomous vehicles. Moreover, the existence of multi-core and many-core (e.g. NVIDIA Jetson series) computing boards motivates the use of HPC. In fact, all the portrayed solutions support a Linux flavor, mostly Debian-based operating systems, including Ubuntu. Furthermore, throughout this thesis, it will

[1]It depends on micro SD card capacity
[2]The TK1 includes a Tegra K1, ISP (Image Signal Processor) and periferics such as USB ports, etc

be demonstrated that all the required software for each of the HPC software layers can be installed in Raspbian, Ubuntu and Kali Linux ARM versions.

The Raspberry Pi is an inexpensive (around 30 USD) computing board used in all kind of applications (RPiFound, n.d.b), including unmanned vehicles missions. For example, Kizar & Satyanarayana (2016), used MATLAB in a Raspberry Pi in order to perform on-board object detection and location estimation. Choi *et al.* (2016) developed a 99% accuracy Raspberry Pi on-board OpenCV (Open Computer Vision) (Bradski & Kaehler, 2000) algorithm, that allowed real-time flight plan changes, when objects of interest were detected. Moreover, the authors integrated the Raspberry Pi with an autopilot, specifically the Pixhawk 1 (ARDUPILOT, n.d.c), in order to automatically fly towards detected targets. More interestingly, Daryanavard & Harifi (2018) implemented a face detection algorithm running in a Raspberry Pi which achieved 98 %, 93 %, 86 % and 80 % success rate with a camera-to-target distance of 1.5, 3, 4 and 5 meters, respectively. Furthermore, Royo *et al.* (2018) embedded an UAV with a Raspberry Pi for radioactive sources detection.

While the Raspberry Pi is a very useful solution, its computing performance might not be sufficient for certain applications as demonstrated by Vega *et al.* (2015). The authors analyzed data-transfer rates between an on-board camera, controlled by a Raspberry Pi and a laptop acting as a ground station via Wi-Fi connectivity, aiming at real-time, defined as 30 frames per second (FPS) in the experiment setting. The results showed that the combination of the on-board and the ground computers outperformed significantly two baselines, computation only in the ground station and computation only in the UAV computer. Moreover, their results suggest that integration between different computing infrastructures, with different computing capacities, is a very powerful approach that must include a load balancing mechanism. However, the Raspberry Pi/laptop approach did not achieve the desired 30 FPS for real-time video transfer. Consequently, the authors replaced the Raspberry Pi for an NVIDIA Jetson TK1, obtaining therefore a peak data-rate of 42.6 FPS, using only the TK1 embedded CPU.

The NVIDIA Jetson series boards, e.g. TK1, TX1, TX2, AGX Xavier (see Table II-2), etc., embed a CPU and a many-core CUDA-enabled GPU, a very interesting solution for smart UVs. CUDA, which stands for Compute Unified Device Architecture, is a technology that allows to use a GPU as a general-purpose computing (GPGPU) co-processor, where the CPU processor (*host*) uses the GPU (*device*) for highly demanding computing tasks. Cocchioni *et al.* (2016) showed the implementation of visual-Based landing for an Unmanned Quadrotor using the TK1. Moreover, Jeon *et al.* (2016) presented a CUDA real-time UAV vision-guided control algorithm (detection and tracking), showing the potential of the TK1 192-cores embedded GPU, that under certain conditions, outperformed an X86 Intel i5 processor, a common desktop computer's CPU and obtained around five times speedup in comparison with the ARM quad-core Cortex-A15 CPU embedded in the TK1.

While the Jetson TK1 has been discontinued since April 2018 (NVIDIA, n.d.e), the CPU/GPU Tegra K1, embedded in the TK1 toolkit, will still be available until January 2024 (NVIDIA, n.d.i). Nonetheless, the TX1, TX2 and the new Jetson series AGX xavier provide much more computing power than the available in the TK1 series. For example, the TX1 has been used for UAV path planning in military applications, achieving a 33X speedup versus ARM architectures and with a 10 Watt energy requirement (Roberge & Tarbouchi, 2017). Zhang *et al.* (2016) compared the use of the embedded TX1 versus the NVIDIA Titan X (NVIDIA, n.d.g), a desktop GPU, for real-time object detection in surveillance video. Their results showed a speed rate of 46 FPS at 45.8 Watts in the TX1 versus 40 FPS at 90 watts in the desktop GPU, a positive feat for embedded boards in terms of energy consumption, i.e. same performance at half the energy consumption. In fact, the TX1 and TX2 have a typical power consumption of 10 and 7.5 Watts respectively (NVIDIA, n.d.f) and can be used as companion computers interacting with ArduPilot (ARDUPILOT, n.d.). For more information about connecting the TX1 and TX2 with ArduPilot see ARDUPILOT (n.d.a) and

ARDUPILOT (n.d.b).

Furthermore, applications requiring Artificial Intelligence (AI), e.g deep learning, for unmanned vehicles applications, can find necessary computing resources in GPUs such as the TX2. Tijtgat *et al.* (2017) presented the results of a very interesting alert system for SAR applications, which integrates a UAV, performing GPU (TX2) object detection based on the YOLO (You Only Look Once) algorithm (Redmon *et al.* , 2016) and a decision support component running in a ground station. The decision support component alerts ground human troops of imminent distance to dangerous objects, i.e. high-pressure gas pipes, etc., based on the results of the embedded GPU object detection algorithm. This system is a good example of the importance of heterogeneous systems composed of mobile entities, ground stations and humans for real-world applications.

Although embedding a multi-core or many-core board is an approach towards HPC, it only considers strategies belonging to the applications layer, leaving behind the potential offered by the remaining HPC software layers. In fact, it is possible to embed a complete HPC cluster in a single robot, as demonstrated by Ribeiro *et al.* (2015). The authors implemented a robotic flying crane, with an embedded cluster of computers composed of five ODROID boards, Ubuntu and Sun Grid Engine (SGE) (Gentzsch, 2001) as batch system. However, a single robot can only embed a limited amount of computing boards. Such limitation has been addressed by allowing robots to interact with cloud computing services. Benavidez *et al.* (2015) proposed a ROS (Robot Operating System) (Quigley *et al.* , 2009) cloud-based HPC infrastructure to aid a robot in visual SLAM. By using provisioning mechanisms, the HPC cloud can scale to include any quantity of computing nodes if required. Notwithstanding, a second approach can be considered to allow scalability while reducing latencies that might occur when uploading and downloading data from the cloud i.e. a *HPC cluster of robots*.

Holland *et al.* (2005), proposed the UltraSwarm system, an interesting combination of bioinspired UAV flocking and wireless HPC. While many studies focused on extracting algorithms for motion coordination and collision avoidance based on the behavior of biological entities, e.g starlings flocks, etc., the authors devised the UltraSwarm system, with the objective of providing a UAV swarm with a single controlling intelligence based on HPC. However, the authors did not implement any of the HPC software layers, except a Linux operating system. The authors did however, showed the possibility of establishing cluster communications using Bluetooth, with the limitations in 2005, when their work was published. Nowadays, communications technologies have improved remarkably in comparison with 2005, a strong advantage for the setting of a full wireless HPC cluster of computers nowadays.

For example, Marjovi *et al.* (2012) proposed the concept of *robotic cluster* representing a group of robots able to share their computing resources among the group with the objective of quickly solve computationally hard problems. However, the authors did not deploy the user system nor the file system layer. In addition, the authors described MPI as the middleware in a HPC cluster. While MPI provides middleware services, such as scalability (any quantity of parallel processes), the batch system is the actual middleware in a HPC cluster of computers, because it abstracts a distributed set of nodes as a single image system.

An important aspect, when dealing with HPC in the context of mobile robots such as unmanned vehicles, is the underlying communications technology. Sudden disconnection or poor network quality can affect negatively the performance of a HPC cluster of robots. In fact, attempts to devise mobile HPC and solve communications issues have been done in the past. For example, Cheng *et al.* (2000), under DARPA funding, proposed a Java-based framework for hybrid (i.e. stationary and mobile computing elements) cluster computing, called HFC (Hybrid Flexible Cluster). By using mobile IP and wireless awareness (Cheng & Marsic, 2000), a mobile node can access HPC resources distributed across the Internet. However, advancements in communication technologies are currently on the verge of providing full mobile HPC. Qualcomm Technologies, at

the beginning of 2017 released a communications performance study (QUALCOMM, n.d.) of Unmanned Aircraft Systems (UAS) using 4G LTE (Long Term Evolution) networks. The experiments were carried out with UAS operating beyond visual line of sight up to 400 ft above ground level. The results were very positive and suggested that full Internet connectivity can be granted to UAS with current communications technologies. Moreover, according to the International Telecommunications Union (ITU) (ITU, n.d.), some of the requirements for a technology to be classified as 5G are:

- Downlink peak data rate: 20 Gbit/s
- Uplink peak data rate: 10 Gbit/s
- User latency: 4 ms for eMBB (Enhanced Mobile BroadBand) and 1 ms for URLLC (Ultra-Reliable Low Latency Cellular Networks)

Such features outperform 10 Gigabit Ethernet, a wired technology commonly used in HPC settings. While having full Internet connectivity in a UAV does not guarantee high performance computing, when comparing with traditional wired technologies (specially when dealing with MIMD software), current and future technology improvements are very promising.

Steps towards HPC in the context of single and multi-robot systems have been proposed in the past, even at industry level. For example, the UAV Chinese manufacturer DJI has developed the Manifold (DJI, n.d.b) embedded computer, based on the NVIDIA Kepler architecture. However, HPC is more than performance, as it has been solely targeted in all previous works and there is still a lack of a formal definition of HPC in the context of robotics. With this in mind, chapter III section III.1 will introduce such definition and discuss relevant concepts related to the union of HPC and robotics.

Next section II.2 discusses a literature review regarding *ubiquitous computing* and *ubiquitous robotics*, aiming at establishing the differences between both concepts and that of *ubiquitous supercomputing*, as proposed in this Ph.D thesis and presented in chapter III.

II.2 Ubiquitous robotics

While it is not entirely accurate to say that computing is everywhere, it is fair stating that the bases and mechanisms for *Ubiquitous Computing*, i.e. computing everywhere, have been proposed and implemented, e.g. Weiser (1993a), Weiser (1993b), Hightower & Borriello (2001), Krumm (2016), Varshavsky & Patel (2016), Bardram & Friday (2016). Starting from the Internet and up to all kind of existing protocols for the integration of different-purpose applications, computing is part of our lives, all the time. Within the computing realm, a particular type of systems, which are of interest for this Ph.D thesis, are those that exhibit supercomputing features.

Figure II-2: *Ubiquitous supercomputing for robotics is different from ubiquitous robotics*

Despite that supercomputing can be found in all kind of applications, from scientific endeavors towards commercial activities, devices in the edge are still mostly absent of its utilization, beyond the simple integration of HPC-enabled computing boards or the offloading of computing tasks towards cloud infrastructures. This Ph.D thesis aims at proposing strategies to bring supercomputing towards the edge, specifically to robotics, therefore contributing to is ubiquity, i.e. *ubiquitous supercomputing* (Figure II-2).

Attempts to establish ubiquitous supercomputing have been done in the past. For example Foster & Tuecke (1996) defined ubiquitous supercomputing as the capacity of a software application, running locally (whether on a low-end PC or high-end workstation) of exploiting remote supercomputing resources and introduced a Java technology to do so. While the authors published their results in 1996, nowadays such approach can be effectively achieved via cloud services e.g. Amazon (n.d.). In a more recent work, Chu & Hsiao (2010), the authors introduced OpenCL (Open Computing Language), a framework for writing software to be executed across heterogeneous platforms such as CPUs, GPUs, DSPs (Digital Signal Processors), FPGAs (Field-Programmable Gate Arrays), etc., aiming at reducing the inherit complexity of software writing, specially for accelerators (e.g. GPUs) and contributing to the ubiquity of supercomputing and robotics as well (Palossi *et al.* , 2016).

In this sense, robotics is moving towards ubiquity, e.g. service robots (Carbone *et al.* , 2018), healthcare and lifecare robots (Pee *et al.* , 2018), industrial robots (Çürüklü *et al.* , 2010), medical/rehabilitation robotics (Colombo & Sanguineti, 2018), human-robot interaction (Varrasi *et al.* , 2019), space robotics (Cheng *et al.* , 2006), (Yoshida, 2009), (Flores-Abad *et al.* , 2014), (Domínguez *et al.* , 2018), (NASA-JPL, n.d.) teleoperation (Yang *et al.* , 2017), humanoid robots (Spenko *et al.* , 2018), field robots (Martins *et al.* , 2015), biorobotics (Ijspeert, 2014), nanorobotics (Ummat *et al.* , 2016), unmanned vehicles (Valavanis & Vachtsevanos, 2015), (Usach *et al.* , 2018), artificial intelligence (Russell & Norvig, 2016), etc, all contributing at *ubiquitous robotics*. In fact, the IEEE Xplore library lists more than $150,000$ papers related to autonomous robots and more than $30,000$ papers devoted to autonomous vehicles, all part of the fourth industrial revolution (IEEE, n.d.). Moreover, all kind of robots are quickly becoming mainstream or expected to arrive in the future, for example toy robots, household robots, cloud robots, flying robots (air balloons, helicopters, quadcopters, hexacopters, etc.), autonomous driving vehicles, modular self-reconfiguring robots and ubiquitous robots, those capable of networking, transparent user interfaces and accessible at anytime and anywhere (Kopacek, 2016).

In Kim *et al.* (2004), the authors referred to the concept of *Ubiquitous robot - Ubibot*, incorporating three forms of robots, *software robot - Sobot*, *embedded robot - Embot* and *mobile robot - Mobot* (Kim, 2003) and discussed about three generations in robotics, *industrial robotics*, *personal robotics* and *ubiquitous robotics*, multiple networked robots used in all kind of applications as previously mentioned, somehow a specialization of the IoT era. This was a 2004 work, discussing in a very abstract way, the concept of ubiquitous robotics. In a following work Kim *et al.* (2007), the authors presented a proof of concept of their ideas using tree experiments, Sobot interaction and transfer, Sobot-user interaction through Embots and Sobot-user interaction through Mobot. Moreover, Broxvall *et al.* (2007) discussed the importance of integrating robots with smart environments and developed an ubiquitous robotics middleware designed for low-computing-power devices.

While previous works such as Ando *et al.* (2005) or Do *et al.* (2007) contributed to the birth of ubiquitous robotics, Quigley *et al.* (2009) introduced Robot Operating System (ROS), a technology used in all kind of applications. While ROS stands for an operating system, it is in fact a robotics framework, installed over a traditional operating system, e.g. Ubuntu, running on a companion computer (see Table II-2) or common computers.

Under the scope of ubiquitous supercomputing, ROS is a technology belonging to the applications layer of a ubiquitous supercomputing infrastructure. A discussion of this will be given in section II.3. Moreover, ROS does share similar features as those of ubiquitous supercomputing, e.g. scalability and easiness for the deployment of large robotics systems, general-purpose, standardization, etc., same as HPC technologies such as MPI, CUDA, etc, all belonging to the applications layer. However, ROS is not a supercomputing technology. For example, while ROS does provide a form of a file system, called the parameter server, it is not designed for high performance (ROS.org, n.d.f). Moreover, there is no integration between ROS and HPC, other that abandoned attempts such as the ROS MPI package ROS.org (n.d.e) and the *HPC-ROS package*. The HPC-ROS

package (as result of this Ph.D thesis), which will be discussed in section III.1, provides automatic installation and configuration of a HPC cluster.

ROS is an open-source framework for peer-to-peer robotics applications, where distributed loosely-coupled processes (*nodes*) interact with each other via ROS communications services. While ROS is not a traditional operating system, OS-like services, such as hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes and package management (ROS.org, n.d.b) are provided. ROS' main goal is to facilitate code reuse and collaboration. It this sense, the following goals were devised:

- *Thin*: ROS is designed as thin as possible in order to facilitate integration with other robotics frameworks. Current integrated frameworks are OpenRAVE (the Open Robotics and Animation Virtual Environment) (Diankov & Kuffner, 2008), Orocos (Open robot control software) (Bruyninckx, 2001), and Player (Gerkey *et al.* , 2003)
- *Agnostic libraries*: Nodes written in different languages, e.g. Python, C++, etc., can communicate transparently with each other
- *Language independence*: Current support for Python, C++, LISP and experimental libraries in Java and Lua.
- *Easy testing*: Builtin unit/integration test framework (*rostest*)
- *Scalability*: Support for multi-robot systems

Several research works, powered by ROS, can be found in the literature, $15,938$ research works in the IEEE Xplore library and $70,540$ in ScienceDirect in February of 2019. For example, in Chang *et al.* (2018), ROS was applied in an object identification mission, running on a embedded Raspberry Pi, demonstrating the support of ROS in inexpensive companion computers. Moreover, the authors in Zhongyuan *et al.* (2018) developed Alliance-ROS, a ROS-based framework for cooperative mobile robots. Even for the real time operating system Nuttx (Nutt, 2014) , ROS has been proposed as a feasible solution e.g. (Wei *et al.* , 2014) or (Wei *et al.* , 2016). Furthermore, regarding unmanned vehicles, works such as Grabe *et al.* (2013), Weaver *et al.* (2013) Lee *et al.* (2017), Yu *et al.* (2017), Hu *et al.* (2017), Sagitov & Gerasimov (2017), Jiang *et al.* (2018), Hayakawa *et al.* (2018) evidence the use of ROS. As it can be observed, ROS is strongly contributing to ubiquitous robotics. Furthermore, there are many other robotics frameworks, other than ROS. For more information, please visit PLAYER (n.d.), YARP (n.d.), OROCOS (n.d.), CARMEN (n.d.), ORCA (n.d.), MOOS (n.d.b), MOOS (n.d.a).

Chibani *et al.* (2013) reviewed challenges and future trends in ubiquitous robotics, describing an ubiquitous robot as a networked entity, limited not only to physical mobile robots bur rather to any software agent running on daily living objects such as smartphones, TVs, beds, etc. The authors described two important features for current and future trends in ubiquitous robotics, increased level of autonomy, even in multi-robot/multi-agent systems and human-cooperation, e.g. assisting dependent people in carrying out daily tasks, safety guards, rubbish collectors, industrial smart coworkers, clinician assistants/coworkers, etc, all connected to the Internet and capable of networking and accessing cloud services. Additionally, the authors in Jiménez-González *et al.* (2013), drew on the following conclusions regarding existing ubiquitous robotics frameworks:

Table II-3: *Common features in current ubiquitous robotics frameworks*

Conclusion	Remarks
Significant number of non-integrated frameworks and few highly-integrated frameworks	There exist several frameworks for multi-robot systems but integration between systems, developed with different frameworks, is scarce
Multi-robot frameworks tendency to be ad-hoc	The majority of frameworks use ad-hoc approaches, i.e. designed for specific purpose, compromising integration and scalability
Lack of remote access frameworks	Most frameworks do not provide default remote access for robotics entities
Multi-agent coordination	The most common approach with multi-robot frameworks
Heterogeneity	Tendency to create higher integration and heterogeneity
Increasing adoption of reusable software	e.g. ROS, Roboearth (Waibel *et al.* , 2011), etc.

In the previous work, the authors also identified a set of guidelines for current and future ubiquitous robotics frameworks and testbeds. Such are: *General purpose*, *modular and flexible architectures*, *openness*, *APIs, reusable code and standardized interfaces*, *availability of suitable usability tools*, *remote execution* and *experiments to real applications*. Such features will be discussed in chapter IV. Regarding unmanned vehicles, the authors in Sanchez-Lopez *et al.* (2016) introduced Aerostack, an open-source multi-purpose framework for autonomous multi-UAS operation. While Aerostack is very active Sanchez-Lopez *et al.* (2017), Molina *et al.* (2017) and it is a very interesting framework for UAS, it differs from the ideas of this Ph.D thesis because it does not consider supercomputing in its core foundation.

An important feature to evaluate is that ubiquitous robotics frameworks must provide easy mechanisms to guarantee a transparent transition from simulation to real-world missions. Regarding unmanned vehicles, simulation is essential considering deployment costs in terms of time and resources (e.g. personnel, money, etc.), potential failures, maintenance, mission criticality, etc. Koenig & Howard (2004) introduced Gazebo, a robotics simulator providing rapid algorithms testing, AI system training, physics engine, etc. Gazebo and ROS are maintained by Open Robotics (OpenRobotics, n.d.). Furthermore, several robotics simulation platforms can be found commercially, e.g. V-REP (Rohmer *et al.* , 2013), ARGoS (Pinciroli *et al.* , 2011), Webots (Michel, 2004), Virtual Robotics Toolkit (VirtualRoboticsToolkit, n.d.), X-Plane (Laminar Research, n.d.). A good review of commercial robotics simulation platforms can be found in SmashingRobotics (n.d.). Correspondingly, at research level, extensive robotics simulation platforms have been developed e.g. Shah *et al.* (2018), Gonzalez *et al.* (2015), Degrave *et al.* (2019), Ma'sum *et al.* (2013), Araiza-Illan & Eder (2019), even with HPC cluster support (DeMarco *et al.* , 2019) via OGS (OGS, n.d.) or simulation platforms for verification and validation of commercial space vehicles (Rubin, 2019), etc.

Moreover, Pitonakova *et al.* (2018) compared Gazebo, V-REP and ARGoS concluding that Gazebo and V-REP are close in their provided features but Gazebo's interface and default robot models are much simpler and resemble those found in ARGoS. However, Gazebo still have several aspects to work on, usability, edition for 3D meshes, etc. Furthermore, for large systems, i.e. composed of several robots, Gazebo and other mainstream simulation platforms require strong computing capabilities and technical knowledge, according to Schmittle *et al.* (2018). The authors developed OpenUAV, a cloud-enabled testbed for UAVs based on ROS, including MAVROS (ROS.org, n.d.a), a ROS package for communication with the PX4 autopilot (Dronecode, n.d.c), Gazebo, Gazebo Web version Gzweb (Gazebo, n.d.), Docker (Docker, n.d.) and Ansible (RedHat, n.d.). OpenUAV is a multirotor UAV simulator aiming at speeding up research tasks by lowering

the technical complexity found in other robotics simulation platforms. This work is very interesting specifically because of the use of containers, via Docker, over cloud services, facilitating scalability and simplifying deployment complexities.

In addition, Garcia & Barnes (2009) developed a multi-UAV simulator relying on a small set of computers (7 nodes), which the authors called cluster, but without clarifying if the HPC software layers were deployed. In their work, each node represents an UAV. This is, each machine (vehicle) simulate its own dynamics and kinematics, a very interesting approach since it can provide transparent transition from simulation to reality. This work and the more recent OpenUAV (Schmittle *et al.* , 2018) considered a different approach towards simulation platforms, in which independent machines or containers (Docker) are used. Such approach could result advantageous to facilitate Software In The Loop (SITL) or Hardware In The Loop (HITL) simulations, making use of cloud services or traditional HPC infrastructures.

II.3 Discussion

High Performance Computing is a powerful ally in all kind of scientific endeavors and commercial activities. With constant improvements in embedded computing boards, see Table II-2, lightweight, energy-efficient CPUs, GPUs with hundreds of computing cores and several research works, etc., as it was presented in this chapter section II.1, *HPC in robotics is a reality* , when dealing when single robots. However, multi-robot systems are mostly absent of supercomputing in its full extent. In fact, only light flirting has been done in the past e.g. Holland *et al.* (2005) or Marjovi *et al.* (2012).

Consequently, the next step towards bringing supercomputing in the edge, specifically to robots, is to adapt the utilization of all the HPC software layers (not only the applications layer) to the peculiarities of multi-robot systems, therefore truly creating a HPC cluster of robots and contributing strongly to the ubiquity of supercomputing. In addition, while previous works have suggested the use of HPC within robotics scenarios, there is still a lack of a formal definition and successful strategies to adapt traditional HPC into robotic systems. Moreover, keeping in mind that supercomputing is not only about performance, next chapter, section III.1 will introduce the novel concept of *High Performance Robotic Computing - HPRC*.

If the same software architecture, deployed upon common HPC clusters including the TOP500 supercomputers in the world (Dongarra *et al.* , n.d.), is adapted in the context of multi-robot systems, integration amongst HPRC and traditional HPC infrastructures becomes straightforward. In this sense, ubiquitous supercomputing, as it will be discussed in the following chapter III, is defined as the union of HPC, HPRC and two more type of entities, Computing-Less Devices (CLD), those that do not embed a computing board and ultimately humans.

This definition differs from those found in Foster & Tuecke (1996) and Chu & Hsiao (2010), where the general idea is the capacity of using remote supercomputing infrastructures or transparently exploit accelerators, GPUs, FPGAs, etc., respectively, again only within the applications layer realm and without actually having a local or Edge supercomputing infrastructure. Therefore, simply put, under the scope of this Ph.D thesis:

> *If an Edge device can embed a companion computer, it has the potential of becoming part of a ubiquitous supercomputing system, if it does not, it can use or produce data from and for an ubiquitous supercomputing system. Finally human presence is always suggested.*

In order to deepen in the ideas of ubiquitous supercomputing, as portrayed by this Ph.D

thesis, it is necessary to formally define concepts, strategies, etc., all together described as the *Ubiquitous supercomputing ontology*, which will be presented in section III.2, followed by the concept of *general-purpose computing missions* in section III.3 and the *ubiquitous supercomputing language (UbiSL)* in section III.4.

Despite the fact that autonomy is one of the most desired features for robotics, as presented in this state of the art, this Ph.D thesis considers that the role of a human in increasingly intelligent robots, is of the uppermost importance. Autonomy is not left behind within ubiquitous supercomputing, but it must be controlled in a hierarchical way. Section III.5 will discuss the ideas behind hierarchy, its impact towards stability in section III.6 and autonomy in section III.7.

In the same train of thought, solutions such as ROS aim at fully distributed systems, while traditional HPC, can be understood as centralized, given the existence of a master node. However, this Ph.D thesis considers that the best approach is an elegant combination between centralization and distribution as it will be discussed throughout next chapter.

By combining HPC (set up on static or Cloud-based infrastructures), HPRC, CLD and humans into single systems, supercomputing becomes omnipresent and pervasive. However, in order to facilitate the creation and operation of ubiquitous supercomputing systems, this Ph.D thesis includes the development of a ubiquitous supercomputing framework and middleware technology called *The ARCHADE - TAC*, which will be detailed in chapter IV. In simple terms, to be further discussed in such chapter, a framework is a software providing generic functionalities that can be selectively modified or enhanced with the objective of creating specific applications. Correspondingly, a middleware is a software layer separating two commonly disconnected software layers, in this case an underlying supercomputing infrastructure and the software required to carry on a General-Purpose Computing Mission. The ARCHADE also includes an Application Programming Interface (API) and a set of extra libraries collectively called *PLUS*, which aim at contributing to the development of general-purpose computing missions.

While all the ubiquitous robotics frameworks previously introduced, provide all kind of features and have a strong potential for general-purpose robotic missions, under the scope of this Ph.D thesis, they all are suitable candidates for the applications layer of a ubiquitous supercomputing system. For example, ROS is an excellent prospect for integration with TAC given its both flexible architectures and general-purpose goal. However, TAC targets all the layers in a ubiquitous supercomputing infrastructure, not only the applications layer, as is the case of ROS and other technologies, e.g. OpenCL, etc. Furthermore, in order to control entirely all design and programming aspects, The ARCHADE was designed and developed from scratch, rather than using technologies such as ROS for its creation. Nevertheless, The ARCHADE coupled with all kind of applications layer's technologies can effectively contribute to ubiquitous robotics.

The ARCHADE is also a simulation platform in itself, because each of the entities in a ubiquitous supercomputing system, is in fact an independent machine, physical or virtual and in the near future a container as well, therefore SITL is at the core of The ARCHADE, facilitating a transparent transition from simulated to real mode. Finally, chapter V will introduce a set of applications, i.e. missions, based on The ARCHADE.

The greatest deception in life is the belief that everything is individual and disconnected. But it looks like that, does it not?, same with Newton and Quantum physics. They both apply to us, though they are significantly different. Where is the error? Are mathematics wrong? Is human logic wrong? Incomplete? Does not one plus one equals two?. Maybe one plus one is not two, maybe one plus one is one. All is one and one is All

— Leonardo CF

III

Ubiquitous supercomputing

In the previous chapter, attempts to bring High Performance Computing (HPC) towards the edge were discussed. However, as it was seen, a general conclusion can be drawn from the current state of the art. Supercomputing is seen only under the computing-efficiency light, leaving therefore behind all the other supercomputing features (see section I.1.2). With the objective of bringing supercomputing in its whole extent everywhere, especially to multi-robot systems, where each individual robot is embedded with a companion computer, this chapter presents the ideas behind *ubiquitous supercomputing*, its ontology, its approach towards hierarchy, stability, automation, *general-purpose computing missions*, the *Ubiquitous supercomputing Language - UbiSL*, etc. Furthermore, it introduces the novel concept of *High Performance Robotic Computing - HPRC*, an adaptation of traditional HPC in the context of multi-robot systems. Under the scope of this Ph.D thesis, ubiquitous supercomputing is defined as:

Ubiquitous supercomputing

Definition 1 (Ubiquitous supercomputing)**.** *Ubiquitous supercomputing, is the union of traditional High Performance Computing (HPC), High Performance Robotic Computing (HPRC), Computing-Less devices (CLD) and humans with the objective of creating systems composed of distributed entities but acting as a single cohesive unit capable of executing any type of mission. (See Figure III-1)*

In a ubiquitous supercomputing system, a human plays an important role, as the entity with the highest hierarchy. In addition, computing-less devices are those that do not exercise computing but rather produce or consume data. While a robot could embed this type of devices,

e.g. sensors or actuators, in that case, such devices are considered part of the HPRC entity. CLD refers therefore to independent computing-less devices, e.g. sensor networks, vehicle networks, etc. However, for a broader scope, CLD devices could be coupled with computing devices, an approach valid within ubiquitous supercomputing.

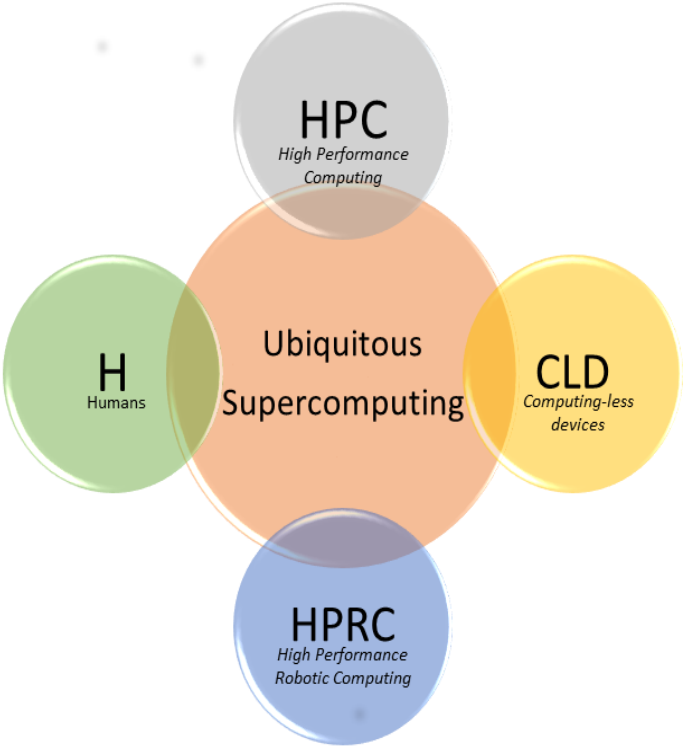


Figure III-1: *Ubiquitous Supercomputing is the joining of traditional High Performance Computing, High Performance Robotic Computing, computing-less devices such as sensor networks and humans such as end-users, operators, managers, etc.*

Simply put, ubiquitous supercomputing aims at serve as a mechanism to do anything imaginable, i.e. precision agriculture, remote sensing, entertainment, *whatever*, by using a ubiquitous supercomputing infrastructure.

Ubiquitous supercomputing infrastructure

Definition 2 (Ubiquitous supercomputing infrastructure)**.** *An Ubiquitous supercomputing infrastructure consists of a set of entities altogether set as a single unit capable of supercomputing, e.g. a HPC cluster, a HPRC cluster, etc.*

Remark. *A HPC cluster or a HPRC cluster by themselves are ubiquitous supercomputing infrastructures. However, a set of CLD alone is not a ubiquitous supercomputing infrastructure, unless it is capable of local computing.*

Though nowadays, given current communications technologies, it is potentially straightforward to implement a HPC cluster of robots, among their companion computers, by simply installing and configuring the HPC software layers, exactly as it is done in any HPC cluster, adaptations have to be made given the mobile nature of robotic entities and other considerations. Furthermore, to rethink what supercomputing means, beyond performance, requires new definitions and concepts to be encompassed in a single concept, HPRC, introduced in the next section.

III.1 High Performance Robotic Computing

In this section, the ideas behind *High Performance Robotic Computing (HPRC)* are introduced. While traditional HPC is commonly seen as a speeding-up tool for complex computing tasks, HPRC goes beyond simply speeding up complex robotic tasks and actually serve as the bedrock for setting up powerful scalable multi-robot systems able to perform all kind of missions. Therefore, High Performance Robotic Computing is defined as:

High Performance Robotic Computing

Definition 3. *High Performance Robotic Computing (HPRC) consists of a set of strategies that allow the deployment of robotic systems that behave and act as a single cohesive unit, able to exploit supercomputing features such as standardization, scalability, heterogeneity, user-transparency, cooperation, resilience, hierarchy, performance, etc., with the objective of creating general-purpose robotic infrastructures.*

Remark. *HPRC includes and extends the traditional HPC's approach towards solely computing-efficiency/performance i.e. works such as those discussed in section II.1 HPC in robotics, fall within HPRC's classification.*

Figure III-2 introduces the HPRC core infrastructure defined as *HPRC cluster* or *HPC cluster of robots* and presents a comparison with a traditional HPC cluster.

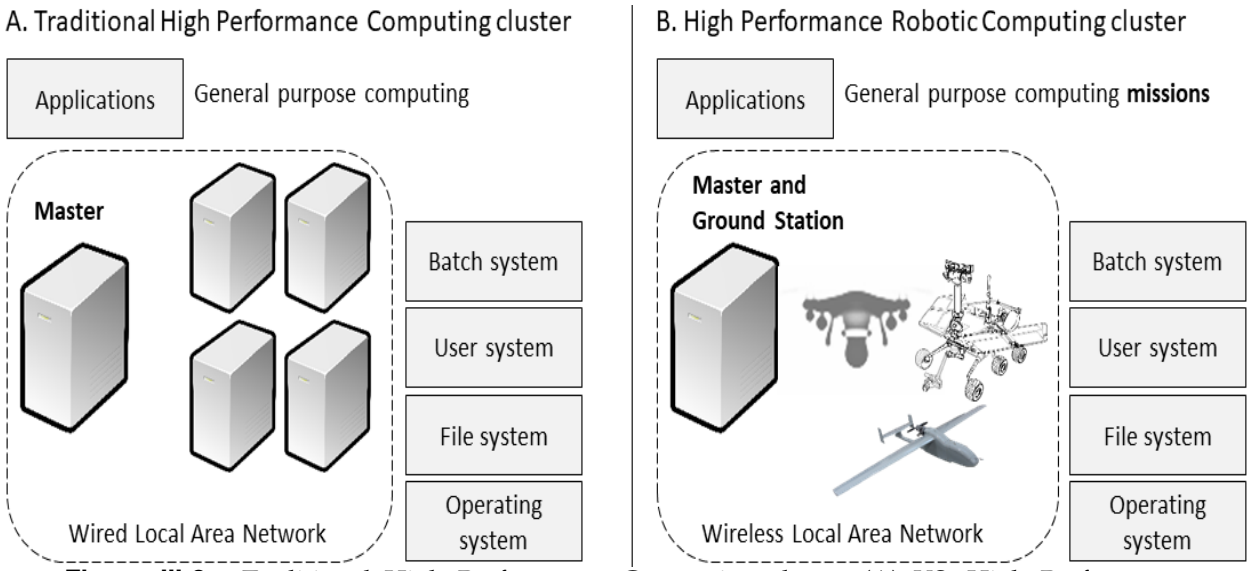


Figure III-2: *Traditional High Performance Computing cluster (A) VS. High Performance Robotic computing cluster (B)*

A traditional HPC cluster (Figure III-2-A) provides general-purpose computing and nodes communicate via wired technologies. Conversely, a HPRC cluster (Figure III-2-B) allows the execution of general-purpose computing missions and nodes communicate via wireless technologies. While HPRC clusters might require the ground station acting as the master node with the objective of protecting the server side of the HPC software layers, other schemes are possible e.g. a robot acting as master or as master and slave simultaneously. More details are given in Table III-1

Remark. *A node in a HPRC cluster is in fact a robot's companion computer. Moreover, a robot can embed any quantity of nodes.*

Table III-1: *HPC VS HPRC Cluster*

Feature	HPC Cluster	HPRC Cluster
Nodes	Commodity computers or state of the art servers with multiple CPUs e.g Intel Xeon (INTEL, n.d.b)), GPUs (NVIDIA, n.d.b), etc., commonly installed with a Linux distribution such as Red Hat, CentOS, Debian, Scientific Linux, etc.	Companion computers in robots such as UAVs, UGVs, etc. Generally any kind of companion computer (see Table II-2), installed with a Linux distribution such as Ubuntu, Raspbian, Kali Linux, etc., and software for robotic interaction and automation, for example DroneKit (3DR, n.d.b) for autopilot control in the case of UVs.
Communications	Wired communications supporting TCP/IP (e.g. Gigabit Ethernet, Infiniband, etc.).	Wireless communications supporting TCP/IP (e.g. Wi-Fi, 4G, 5G, etc.). 5G is expected to outperform Gigabit Ethernet (ITU, n.d.).
Applications layer	General-purpose computing: Any kind of software supporting parallel computing, e.g. MPI, CUDA, OpenMP, OpenCL, etc.	General-purpose computing mission: A graph of tasks, where each task is equivalent to a HPC job. More information about this will be given in section III.3.
Master node	HPC cluster frontend implementing the server side of each of the HPC software layers. A HPC cluster is a centralized architecture, where all data and users reside locally in the master node. However shadow masters can be configured in active-passive or active-active with load balancing configurations.	HPRC cluster frontend implementing the server side of each of the HPC software layers adapted to the context of multi-robot systems. A HPRC cluster is a centralized and distributed architecture (details throughout this section). Several approaches can be applied. The most basic considers the master node as the ground station controlling the multi-robot system. However, a mobile robot can act as master node or master and slave simultaneously. Furthermore, all robots and the ground station can be configured in a P2P mode, where every entity is a shadow master and slave. The advantages of a P2P approach will be that of resilience, however it will increase required computing power. Mixed modes are also permitted, i.e. master and shadow master in active-passive or active-active with load balancing configurations.
Software layers	Operating system, user system, file system and batch system.	Same software layers but adapted to HPRC context. To be discussed throughout the section.
Users	Users in a HPC cluster are all of the same type i.e. computing-resources demanding.	In a HPRC cluster, on top of computing-resources demanding users, there is also pilots, etc. More details in the entire chapter.

Since a node (companion computer) can be installed with any kind of software, including parallel computing libraries such as MPI, CUDA, etc., a node in a HPRC cluster is defined as a *Parallel Robotic Computing Node (PRCN)*.

Parallel Robotic Computing Node

Definition 4. *A Parallel Robotic Computing Node (PRCN) is a robot's companion computer capable of performing parallel computing, either locally or within a HPRC cluster, e.g. via MPI, CUDA, etc. The node is also to be installed with software for robotic interaction, e.g. low-level drivers, autopilot control, sensors and actuators controllers, etc.*

Moreover, given HPRC approach towards general-purpose computing (similar as in traditional HPC), a robot is in fact a *General-purpose computing robot*.

General-purpose computing robot

Definition 5. *A robot embedded with any quantity of PRCNs capable of general-purpose computing. A robot can be used for different missions by simply replacing the software associated to it i.e. a robot is a general-purpose computing unit that temporarily becomes specific-purpose.*

General-purpose computing nature in both HPC and HPRC is given by a simple fact: *Nodes are installed with a mainstream operating system*, i.e. a robot embedded with a companion computer can be used for anything. However, the robot's purpose, i.e. its *mission* is of a more specific nature. This differs from robotic settings where no standard operating system is installed and the robot is used for single-purposes, e.g. industrial robots, complicating interaction and integration with other robots or reutilization. All these issues are solved by setting a robot as a general-purpose computing unit (definition 5). It is the desire and recommendation of this Ph.D thesis that every future robot embeds at least one companion computer running a mainstream operating system.

Other supercomputing features, different from general-purpose computing, such as scalability, heterogeneity, etc., are provided by the HPC software layers as discussed in chapter I section I.1.2 and chapter II section II.1. While traditional HPC infrastructures use Linux server operating systems, robotics community using companion computers is gravitating towards Debian-based Linux distributions, e.g. Ubuntu. For example, ROS officially supports Ubuntu, including ARM architectures, e.g. Raspberry Pi, etc. However, experimental releases can be installed on Windows, Raspbian, OpenSUSE, Android, etc (ROS.org, n.d.c). Moreover, software packages required for each of the HPC software layers are available for most Linux Distributions and can be installed in ARM architectures, such as Raspberry Pi, etc, for Ubuntu, Raspbian and Kali Linux, as demonstrated by this thesis. More information about this will be given in section III.1.1. The reasons for choosing Debian-based operating systems, specially Ubuntu, as the common solution in the HPRC operating system layer are:

- Debian-based distributions (including Ubuntu) are lightweight and very easy to interact with (user-friendly)
- The clear support of the robotics community
- The existence of all kind of robotics frameworks, libraries, e.g. DroneKit (3DR, n.d.b), ArduPilot SITL (ArduPilot, n.d.), etc., all supported by Debian-based distributions, etc
- The AI and data-science community gravitation towards Ubuntu, with supported technologies such as TensorFlow (TensorFlow, n.d.), Keras (Keras, n.d.), etc.
- The close interaction between Ubuntu and Python, the programming language quickly becoming the robotics programming language, even overthrowing C and C++, given its versatility, fast prototyping, easy syntax, thousands of available libraries and performance constant improvements. A good discussion comparing Python and C/C++ for embedded systems can be found in Radcliffe (n.d.).

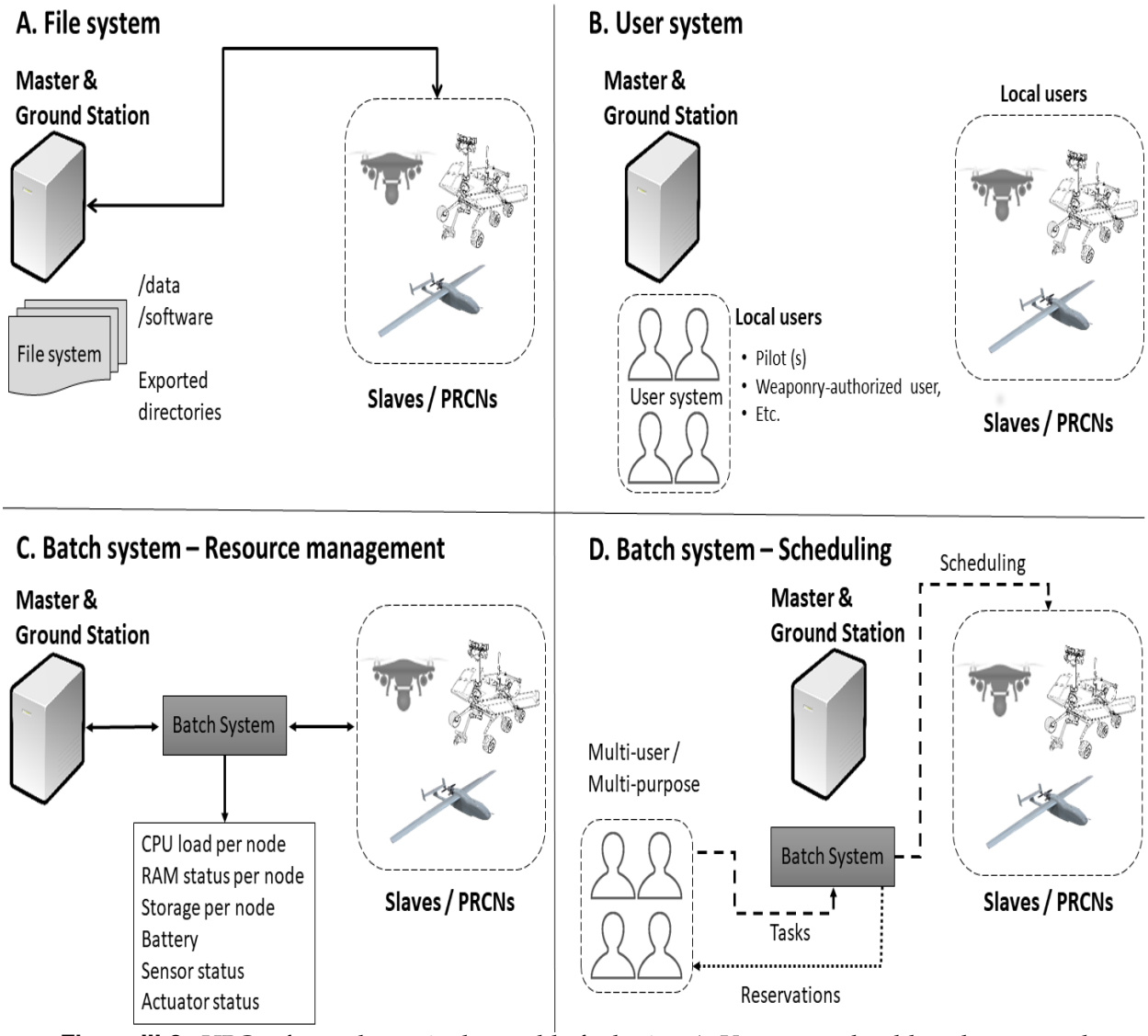


Figure III-3: *HPC software layers in the world of robotics. A. User space should not be exported. B. The same users should exist locally in the master and the slaves rather than remotely in the master node. C. HPRC resources include batteries, sensors, etc. D. A HPRC task is equivalent to a HPC job*

While the same HPC software layers are installed in a HPRC cluster, its configuration and general functioning differs from traditional HPC as it can be observed in Figure III-3 versus Figure II-1 in chapter II. The differences among such configuration are caused by the mobile nature of robotic entities, which could potentially compromised the HPRC infrastructure, leading to errors in software execution or access to data, users, etc. Given these reasons is important to have the following considerations:

- Robots and generally mobile entities must be able to work autonomously without depending on constant connectivity with the master node, the ground station and other robots.
- The utilization of MIMD software, i.e. requiring constant message exchange between companion computers distributed in different robots, is discouraged when a task is critical in nature or when sudden constant disconnections are foreseen. For example, BVLOS (Beyond Visual Line of Sight) missions should not use MIMD software for motion coordination amongst robots, if constant connectivity can not be guaranteed. However, 5G and other technologies can eventually provide constant connectivity, even in remote areas.

- While the robots must be able to operate autonomously, it is important to devise strategies that allow the consistency of a single-image system, i.e. data shall be available in the entire infrastructure, a process that can be done when connectivity is granted. Chapter IV will introduce synchronization mechanisms. However, approaches such as DTNs (Delay Tolerant Networks) are out of the scope of this work and are considered part of the future work (see section VI.2).
- Task redistribution is important when dealing with multi-robot systems subject to possible failures and disconnections, however the mechanism for such purpose in The ARCHADE is currently in development, therefore results will not be introduced in this Ph.D thesis.

Following, the differences between HPC and HPRC clusters functioning (i.e HPC software layers), will be discussed:

- *Master node*: In a HPRC cluster, the most common approach is to set the master node and the ground station in the same machine. This default behavior is suggested in order to protect the server side of each of the HPC software layers from potential failures occurring in mobile entities. However, this is not the only available approach. A robot could act as master node, as shadow master node, as master and slave at the same time or as ground station for other robots. Even each robot could embed its own ground station. All these approaches are valid within the scope of HPRC. In fact, when distance amongst robots is important, given mission's requirements, setting up a mobile robot as a master or multiple robots as shadow masters, is advisable. Not necessarily the ground station needs to be set as the master node, however it should be set at least as a shadow master node. Moreover, by default, a master or shadow master node is a slave node as well, contrary to traditional HPC. This is done to exploit all available computing resources in a HPRC cluster.
- *Data and user sharing*: In a HPC cluster, the master node commonly stores data locally or in a storage-attached unit. Correspondingly, data is exported via the file system to the slave nodes. In addition, the results of a particular simulation or software execution (output data), computed in the slave nodes, are stored in such exported folders, locally at the master node. Traditionally, the most common exported folder is the one containing *users' space*, e.g. /home in a Linux distribution. This implies that all the users exist in all the nodes, usually locally in the master and exported to the slaves via the user system. Consequently, the slaves usually do not store any data or user locally. Contrary, in a HPRC cluster, the users should rather exist locally in the PRCNs (Figure III-3 - B) because in case of disconnection from the master node, during the execution of a mission, the robot should be autonomous enough to continue operating. Accordingly, the user's space (/home) should not be exported from the master node (Figure III-3 -A), but rather exist locally in each PRCN. With this approach, the PRCNs actually store data locally. However, aiming at maintaining a single-image system, such data (e.g. imagery, etc.) must be copied in a shared location (folder locally in the master node) and accessible to all entities and users accordingly to a necessity and privileges scheme. Therefore, in case of failure, the master holds a centralized image of all data acquired by the multi-robot system/HPRC cluster. Data copying can be done constantly or in proximity with Wi-Fi repeaters or antennas, etc. Intelligent approaches can be taken into consideration by monitoring communications status. Other approaches for data sharing can be applied. For example, distributed file systems, in which the local storage available in each node is combined as a single storage. However, this relies on network connectivity and it should be avoided until communications technologies improve sufficiently to guarantee uninterrupted connectivity. Finally, regarding user sharing, Open LDAP can be set to operate in a HPRC cluster. However, it is still a centralized approach that must be handled carefully. Since a HPRC cluster is not necessarily an infrastructure managing hundreds of users like a traditional HPC cluster, local user replication represents the least risky approach.

- *Availability*: A HPRC cluster is not by default an all-the-time available infrastructure, as an HPC cluster should be. Energy limitations, specially with current existing batteries, prevent a HPRC cluster to operate beyond common individual autonomy times, e.g. 60 minutes. However, depending of the power-supply technology available in the robotic entities, an HPRC cluster could be available all the time, for example for remote sensing missions.

High Availability Robotic Cluster

Definition 6. *A High Availability Robotic Cluster (HARC) is a specific type of HPRC cluster that is not used for any particular mission, but rather allows users to connect and execute all kind of tasks online (Figure III-3 - D).*

Remark. *A HARC is similar to a common HPC cluster but it is mostly used for tasks that are dependent on location. For example, a UAV or balloons HARC could be used to take pictures of a designated area, during long periods of time. In addition, users connect to the cluster to perform computing tasks over the collected images, competing for the available computing resources via the batch system. In the future, this could apply to satellites or spacecraft.*

- *Dynamicity*: A HPRC cluster is a dynamic infrastructure. While it can be used for anything (general-purpose), during the execution of a mission, a HPRC cluster is a specific-purpose infrastructure. This means that data, users, etc., are potentially not the same in all missions to which a HPRC cluster is targeted. Consequently, data and user sharing shall be adapted to each mission's requirements. This differs from an HPC cluster, in which all data and users are available even if different simulations, software, etc., are being executed.
- *Batch system*: Regarding the batch system, HPRC extends HPC with two features. First, resources in a HPRC cluster are not only CPUs, GPUs, RAM, etc., but sensors, actuators, cameras, battery, LIDAR, etc (Figure III-3 - C). By default, no batch system monitors such type of resources but it is rather easy to adapt open source solutions such as PBS Torque (Adaptive-Computing, n.d.b) or SLURM (SchedMD, n.d.) to do so. Such adaptation is out of the scope of this thesis but certain features have been developed for the monitoring service of The ARCHADE (see chapter IV section IV.3). Second, the concept of HPC job is replaced for *HPRC task* (Figure III-3 - D). Though both concepts are ultimately equivalent, job is part of HPC terminology while task is consistently used in robotics literature.

Same as with a HPC job, A HPRC task consists of a software requiring a set of resources (e.g. a camera, a thermal camera, X computing cores, etc.) during a walltime. Furthermore, under HPRC's scope, *every robotic task is by default associated with a software*. HPRC tasks are classified as:

- *None*: While a HPRC task is by default linked with a software, None tasks are those that are not linked with any software.
- *Exclusive*: An exclusive HPRC task is such that is carried out by a single entity. For example, consider a scenario where only one robot has certain type of camera and such camera is a requirement for a particular task. This task is only performed by such robot.
- *Parallel*: A parallel HPRC task is such that is carried out by multiple PRCNs at the same time. For example, given a geographic area, each of the robots is given the task of monitoring a sub area. This process is done in parallel. Furthermore, tasks linked with SIMD and MISD software are classified as parallel.
- *Distributed*: A distributed HPRC task is such that is carried out by multiple PRCNs but not necessarily at the same time. Furthermore, tasks linked with MIMD software (e.g. MPI) are classified as distributed. For example cooperative SLAM optimized by MPI.

The classification resembles the ideas given by Flynn's Taxonomy (Flynn, 1972). Moreover, a *General-Purpose Computing Mission is a directed graph of tasks*. This concept will be formally introduced in section III.3. Deployment of HPC and HPRC infrastructures requires intimate knowledge of Linux operating systems. In order to simplify installation and configuration of the HPC software layers, a ROS package depicted as the *HPC-ROS package* is presented in next section.

III.1.1 The HPC-ROS package

The HPC-ROS package, a result of this Ph.D thesis, is an ubiquitous supercomputing enabler, that can be used to automatically install and configure all the HPC software layers in Debian-based companion computers or common computers.

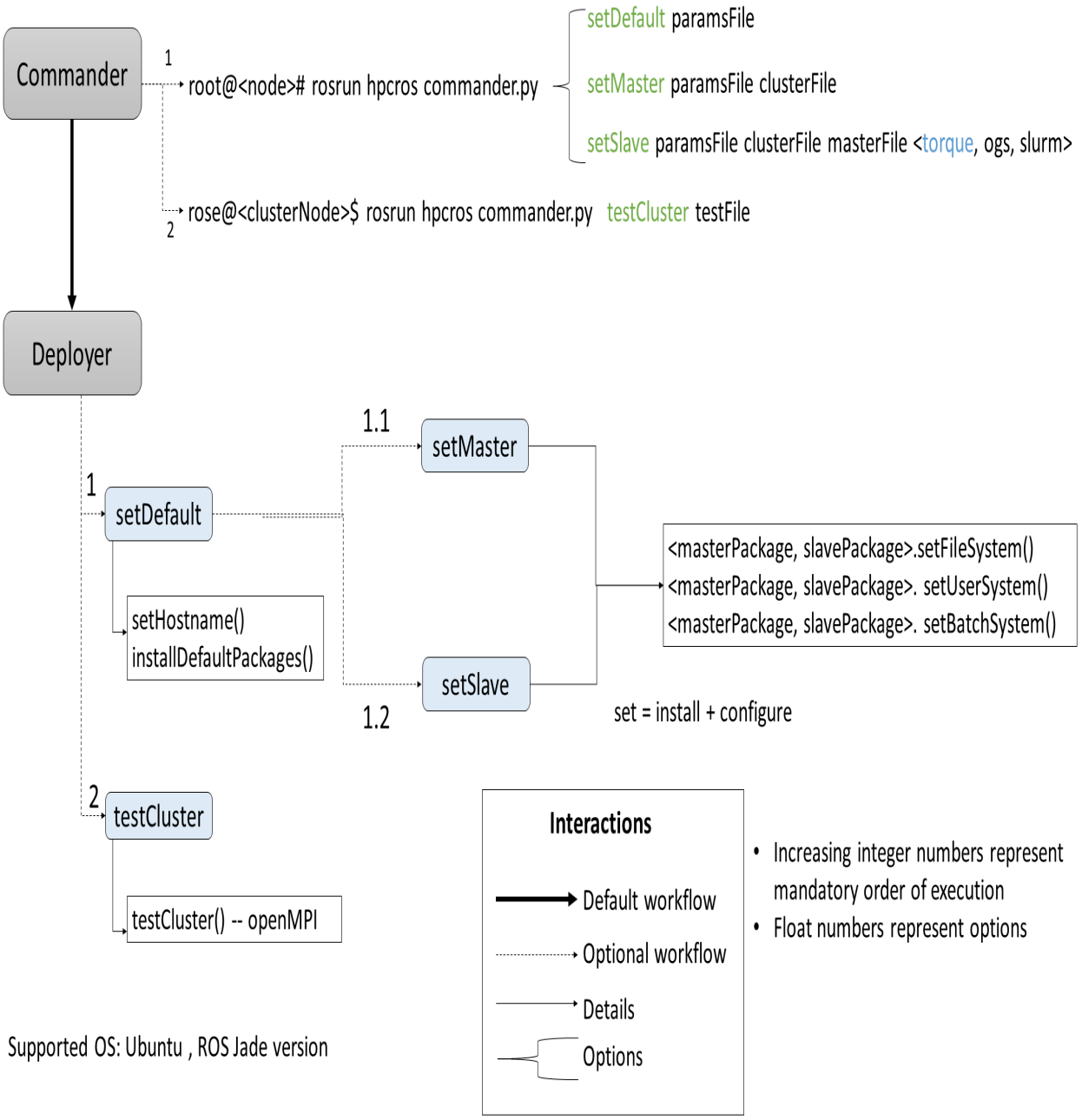


Figure III-4: *HPC-ROS package. Automatic installation and configuration of HPC software layers*

The HPC-ROS package implements two ROS nodes (ROS, n.d.), i.e. commander and deployer to allow:

- *Automatic deployment (installation and configuration) of the HPC software layers*: Setting up a HPC/HPRC cluster, specifically the HPC software layers, is a complex process requiring deep knowledge of Linux operating systems and HPC in general. Facilitating its deployment allows the robotics community to focus only on the mission software (applications layer). The HPC-ROS package can be selectively used for setting up a master or a slave node.
- *The use of standard HPC software applications for each layer*: By using standard HPC software applications, an HPRC cluster can interact with standard HPC clusters, for example, for missions requiring computing power not available in the robots. Furthermore, a mixed HPRC cluster composed of companion computers and normal computers can be set with the HPC-ROS package.

Specifically, the HPC-ROS package deploys and configures the following HPC technologies for each layer over an Ubuntu operating system:

- File system: Network File System - NFS (Shepler *et al.* , 2003).
- User system: Local user replication or Open LDAP (Zeilenga, 2006).
- Batch system: PBS Torque (Adaptive-Computing, n.d.b).
- Applications layer: Open MPI (Gabriel *et al.* , 2004).

Among several software options for each layer (see section II.1), the selected solutions were used as a proof of concept and because of their simplicity in comparison with other solutions. However, the package's roadmap includes the support for other software solutions, even the deployment of similar supercomputing architectures such as those related to Big Data, etc.

The *setMaster* and *setSlave* options (see Figure III-4) install and configure each of the previous software in either a master or slave node correspondingly. In addition, the *testCluster* option automatically executes MPI software to test deployment success. Furthermore a default user, default shared folder and default batch system queue are set during the configuration process.

While the package is based on ROS, HPC deployment libraries can be decoupled from ROS easily if ROS is not available in the targeted infrastructure. The functionalities of the HPC-ROS package have been implemented in The ARCHADE (see chapter IV section IV.2) and the package is currently being adapted to the current ROS version (ROS.org, n.d.d).

The new version will be renamed as the *HPRC-ROS* package and it will take fully into consideration all the guidelines proposed by High Performance Robotic Computing as currently done by The ARCHADE. Moreover, the HPRC-ROS package will be part of The ARCHADE and will include strategies for multi-master and P2P approaches. Ultimately, this is part of a strategy to support ROS via The ARCHADE.

III.1.2 Features and advantages

The features/advantages of adopting HPRC in multi-robot systems are summarized in Table III-2

Table III-2: *Supercomputing features in multi-robot systems*

Feature	High Performance Robotic Computing
Centralization and distribution	In a multi-robot setting, each robot using its own resources can perform its assigned tasks (*distribution*). Moreover, the master and ground station, acting as user frontend, can control the entire robots (*centralization*)
Cooperation	SIMD, MISD, MIMD software, area distribution, etc.
Cost	Using multiple robots could be less expensive that setting a monolithic robot with the same amount of computing power. For example, this ZDNet blog (Nichols, n.d.) shows how to build an UAV with a Raspberry Pi for 200 USD. This low cost per robot allows building inexpensive multi-robot HPRC settings
General-purpose computing	A HPRC cluster can be used for different purposes, different missions, etc., transforming a robot into a general purpose computing infrastructure and facilitating reutilization and integration with other robots, other multi-robot systems, etc.
Hierarchy	Hierarchy, as defined by this Ph.D thesis describes the capacity of an entity of commanding orders over other entities. As the complexity of a system increases, hierarchy is more relevant. More details about this in section III.5
Heterogeneity	All types of robots (UAVs, UGVs, etc.) with different types of sensors, actuators, Linux Operating Systems, etc., can be integrated into an HPRC cluster.
Multi-purpose	A HPRC cluster could be used for several missions at the same time, e.g. HARC (definition 6). For example, a subset of the total computing cores in the HPRC cluster could be dedicated to process imagery, while the remaining computing cores could be used to provide Internet connectivity, content download, etc.
Multi-user	Multiple users could use the resources of a HPRC cluster. For example, a pilot user, a camera user, etc. Moreover, multiple users could be logged in at the same time performing different tasks within a mission or different missions, etc.
OS based robots and standardization	With operating system based robots, a robot becomes a complete Internet of Things' device, as a mobile phone. With traditional and stable operating systems, all kind of software packages, standard libraries, etc., can be installed on a robot's companion computer, even during the execution of a mission (i.e. *on-fly installation*). The possibilities are endless.
Performance	While the main objective of traditional HPC is to provide high computing power, a HPRC cluster does not necessarily needs this. However, it is a good plus
Resilience and failure tolerance	Task redistribution in case of robot failures, creating resilient multi-robot systems. This is not provided by any of the HPC software layers, therefore it would require extra software and it is out of the scope of this Ph.D thesis.
Scalability	Simple integration of new robots at the beginning or during real-time missions. Mission area redistribution, failure tolerance improvements, etc. Furthermore, the same software and strategies can be applied to any number and type of robots providing flexibility as well.
Security	Communications with robots are usually done via insecure and not scalable radio signals. If TCP/IP communications are implemented, the complete secure Internet scheme can be set up among robots communications.
User-transparency	Single image / Centralization and Distribution. The user (e.g. the pilot) could control the entire multi-robot system from a single interface (e.g. the master node). However, each robot could be given sufficient autonomy creating a fully centralized and distributed system.

III.2 Ontology

The ubiquitous supercomputing ontology represents the high-level philosophy of this Ph.D thesis and defines systems that fall under the ubiquitous supercomputing classification (Figure III-5).

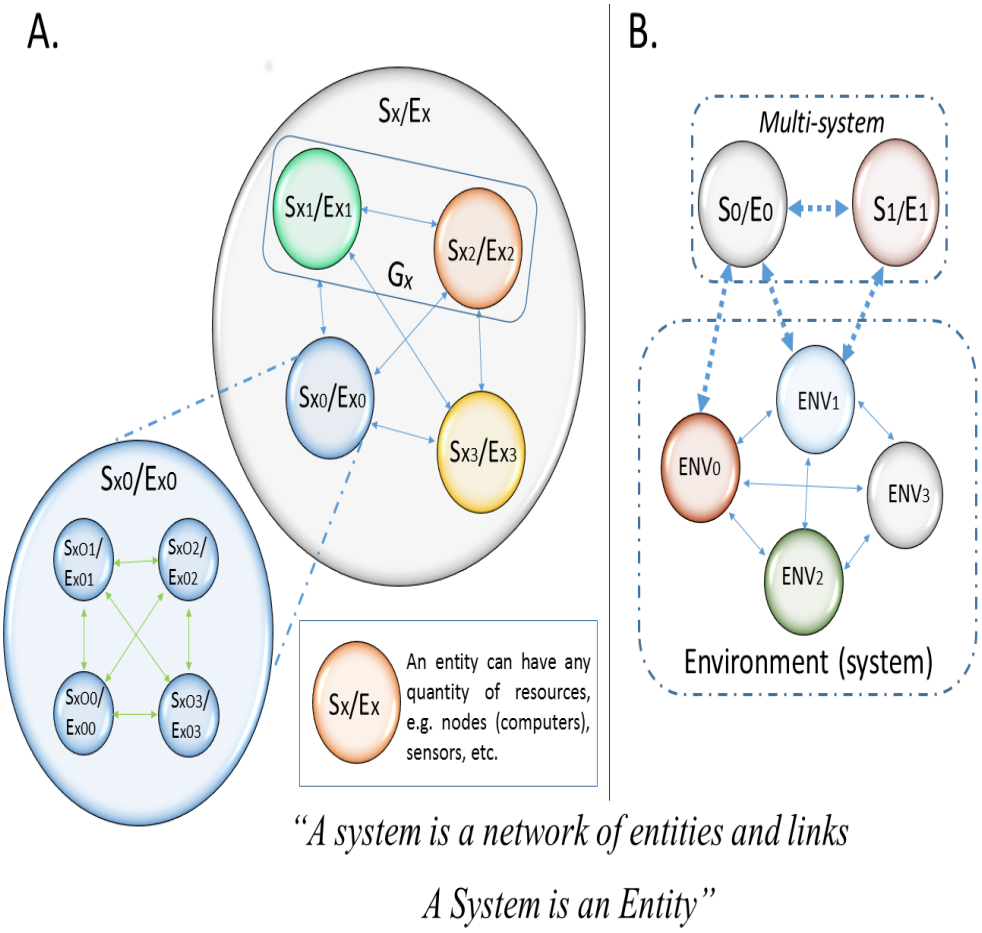


Figure III-5: *Ubiquitous supercomputing ontology.* ***(A)*** *A system is a network of entities and links. A System is an Entity. A system is composed of entities where each entity can be a system as well. Consequently, a system is an entity.* ***(B)*** *A system of type multi, interacting with an environment via interfaces (blue thick dotted lines). Systems 0 and 1 communicate using interfaces as well. In addition, each system can have a different environment to which it interacts with.* Sx *= System X,* Ex *= Entity X,* Gx *= Group X*

Consequently, *ubiquitous supercomputing systems*, or *systems* for the remaining of this Ph.D thesis, are described accordingly to the following lemma:

Ubiquitous supercomputing systems lemma

"A System is a network of entities and links
A System is an Entity"

Remark. *Ubiquitous supercomputing systems are designed targeting at concepts such as scale-free networks Barabási & Bonabeau (2003) and self-similarity, exhibiting fractal-like behavior. Scale-free networks have been found to described computer networks, including the Internet and self-similarity can be found in all kind of natural entities, e.g. trees, geography lines, the human brain, galaxies, etc.*

Entities are the bricks of a system and correspond to elements interacting with each other. Examples of entities are robots, servers, people, sensors, etc. An entity is by default a computing-element, i.e. it embeds one up to any quantity of *nodes*, however entities that do not exercise computing, e.g. a sensor, are defined as *dummies* in the ontology.

Furthermore, entities can be grouped as desired. The *group* concept represents that idea. For example, consider a mission carried out by a set of UAVs and a set of UGVs. The UAVs can be grouped as an air-group, while the UGVs as a ground-group. In addition, a HPC cluster is a group of servers. Entities hold specific *roles* within the groups they belong to and such roles can be of any type or represent any classification.

An user, designing a system, can choose to assign any desired role to the entities, but the ontology specifies a set of default group roles:

- Operators: Human entities operating individual entities, (e.g. pilots) or the entire system i.e. the *system operator*. Operators can exercise control over entities as well.
- Controllers: Software agent controlling an entity i.e. *entity controller* or the entire system i.e. *system controller*

A particular important role is the system controller. This role represents the system frontend and the highest hierarchy controller, only below the system operator. For example, in a multi-robot system, the system controller role can be assigned to the ground station controlling the robots. Correspondingly, all entities are assigned with an entity controller, therefore the system controller entity can be assigned with mission tasks as well. Roles can change dynamically, e.g. during the execution of a mission and furthermore an entity can hold different roles within the same group or between different groups.

A *link* represents the connection between entities i.e. the ability of an entity to communicate with another entity. More specifically, a link represents the communications technology between a pair of entities. The nature of the systems proposed in here is by default dynamic e.g. the links in the network could appear or disappear during the execution of a mission. While a traditional HPC cluster is a wired-based infrastructure, which usually is not subjected to sudden disconnection, an HPRC setting could experience entities' disconnections given its moving essence and wireless connectivity inherit issues, which leads to a dynamic network topology. However, a system, in its most optimal state, is represented by a *full graph*.

The first statement, *"A system is a network of entities and links"*, uses terminology from the field of network science (e.g. link). However, the term node from network science is replaced for entity in the ontology. This way, HPC terminology is preserved. The term *node* refers to a computing unit, e.g. a single computer, an embedded computing board, etc. Entities can have as many nodes as desired. Even an entity could embed a full HPC cluster. Same as with the entities, nodes hold roles. For example, a master node and the slave nodes. The advantages of understanding a system as a network are many, for example:

- If a system is represented as a network, mathematical and conceptual approaches from graph theory and network science can be applied to the system's modeling, understanding and improvement.
- A network is scalable because it allows the adding of new entities and new links, maintaining the autonomy of each of the entities.
- A network can be robust and resilient depending of its specific topology.

The second statement, *"A System is an Entity"*, allows the creation of *systems of systems*, as portrayed by Figure III-5-A. Furthermore, the ontology describes systems equipped with the following features:

- *Cohesion*: From a computer science point of view, user-transparency relates to the capacity of a system of being perceived, from the user's perspective, as a single element, rather than a set of interconnected elements. User-transparency is a default supercomputing feature. However, going further, *cohesion* emerges from all supercomputing features. Entities remain independent but behave as a single unit (*system*).
- *Centralization/distribution*: The system concept represents a set of entities, but all entities/systems are independent from each other i.e. ubiquitous supercomputing systems have a hybrid control mechanism. For example, a multi-robot system is distributed because each of its robots can be given sufficient autonomy to execute a particular mission and it is centralized because a single ground station can control all the robots.
- *Scalability*: Understanding a system as an entity and an entity as a system, where each element is independent, allows easy inclusion and removal of systems, entities, nodes, etc., leading to scalable systems.

Systems are classified as *single* or *multi*. In a multi-system setting (system of type multi), systems communicate with each other via an *interface*. An interface is a communication strategy, which can involve a subset of the entities within two systems. It is defined by a communications technology (link idem), a specific IP / Port tuple or any other approach. Moreover, a system interacts with an environment while performing a mission. The environment is a system as well, according to the ontology here presented, as shown in Figure III-5-B.

Systems can contain as many systems, entities, nodes and groups as desired but a single system should be cohesive, i.e. all entities should focus in a particular mission or a related group of tasks within a mission, during a specific time. Afterwards, the system can be reused for a different mission (idem as a HPRC cluster).

III.3 General-purpose computing mission

A *General-Purpose Computing Mission (GPCM)*, or mission for the remaining of this Ph.D thesis is defined as follows.

General-purpose computing mission

Definition 7. *A General-Purpose Computing Mission, being carried out by a system, consists of a set of interconnected tasks or a directed network as shown by Figure III-6. The direction of the links indicates the task execution order.*

Remark. *Tasks concept is exactly as in within HPRC.*

Remark. *Tasks loops are supported*

A task is defined as a software that requires a set of resources for its execution (idem HPRC tasks). Nodes, computing cores, robotics sensors and actuators, software licenses, etc., are all type of resources within the scope of ubiquitous supercomputing.

Tasks are assigned to entities, holding the task's required resources. Moreover, tasks can be assigned to a subset of the system's entities. For example, a task could consist of a MPI software

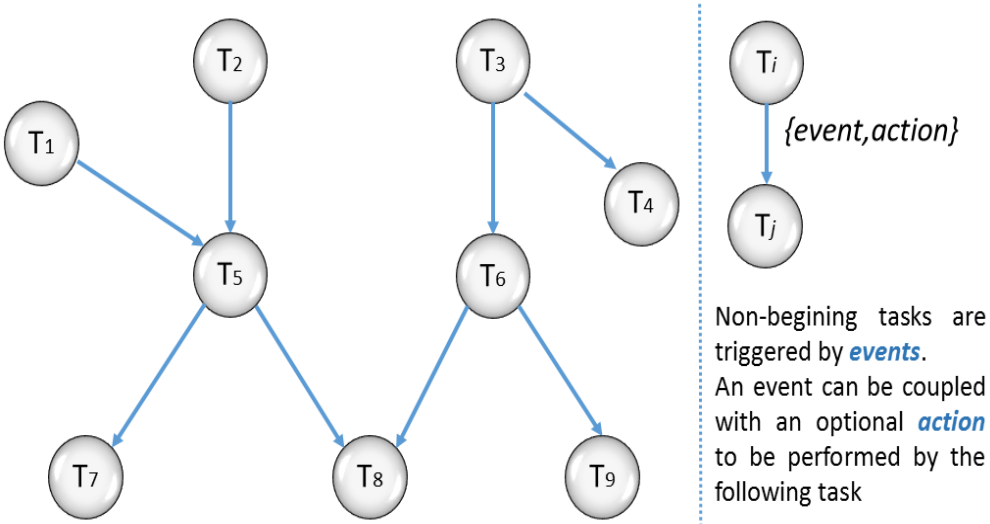


Figure III-6: *General-purpose computing mission. A Mission is a directed network of tasks where each task (T1, T2 ... TN) is a software that requires a set of resources (e.g. computing cores, cameras, sensors, actuators, etc.) in order to be executed*

requiring a set of CPUs not available in a single entity's nodes, therefore the software will be executed in different CPUs that are distributed among different entities. Tasks are classified according to the types:

- *Blocking*: Tasks' blocking types are *go* and *lock*. A task is of type go if the entity executing the task, can proceed with its tasks' children. A task if of type lock, if the entity executing the task, cannot proceed until the task's parents are completed.
- *Execution*: Tasks' execution types are *none*, *exclusive*, *parallel* and *distributed*, exactly as in within HPRC. Parallel (SIMD, MISD) and distributed (MIMD) tasks can be used for computationally demanding problems. This way, ubiquitous supercomputing systems offer computing efficiency.
- *Necessity*: Tasks' necessity types are *mandatory* and *optional*.
- *Topology*: Tasks' topology types are *begin*, *path* and *end*
- *Priority*: Tasks' priority types are *low*, *medium* and *high*.

Missions should be divided in independent tasks, where each task is linked to a different software. Non-beginning tasks (i.e. path and end) are triggered by *events* which are potentially coupled by *actions*.

Events and actions

Definition 8. *An Event, is the default output of a task software, used to trigger children tasks.*

Definition 9. *An Action, is an optional command, coupled with an event, to be performed by a following children task.*

Remark. *A task software can provide any quantity and type of outputs but events are mandatory if the task has children i.e. a children task will not be triggered if parents' events are not triggered, except in type go tasks. Furthermore, actions are considered orders, therefore executed only if the entity triggering the event has a higher hierarchy that the entities performing children tasks. See hierarchy section III.5.*

III.4 Ubiquitous supercomputing language

The *ubiquitous supercomputing language - UbiSL* aims at facilitating the practical implementation of the ubiquitous supercomputing ontology and all concepts exposed in this chapter.

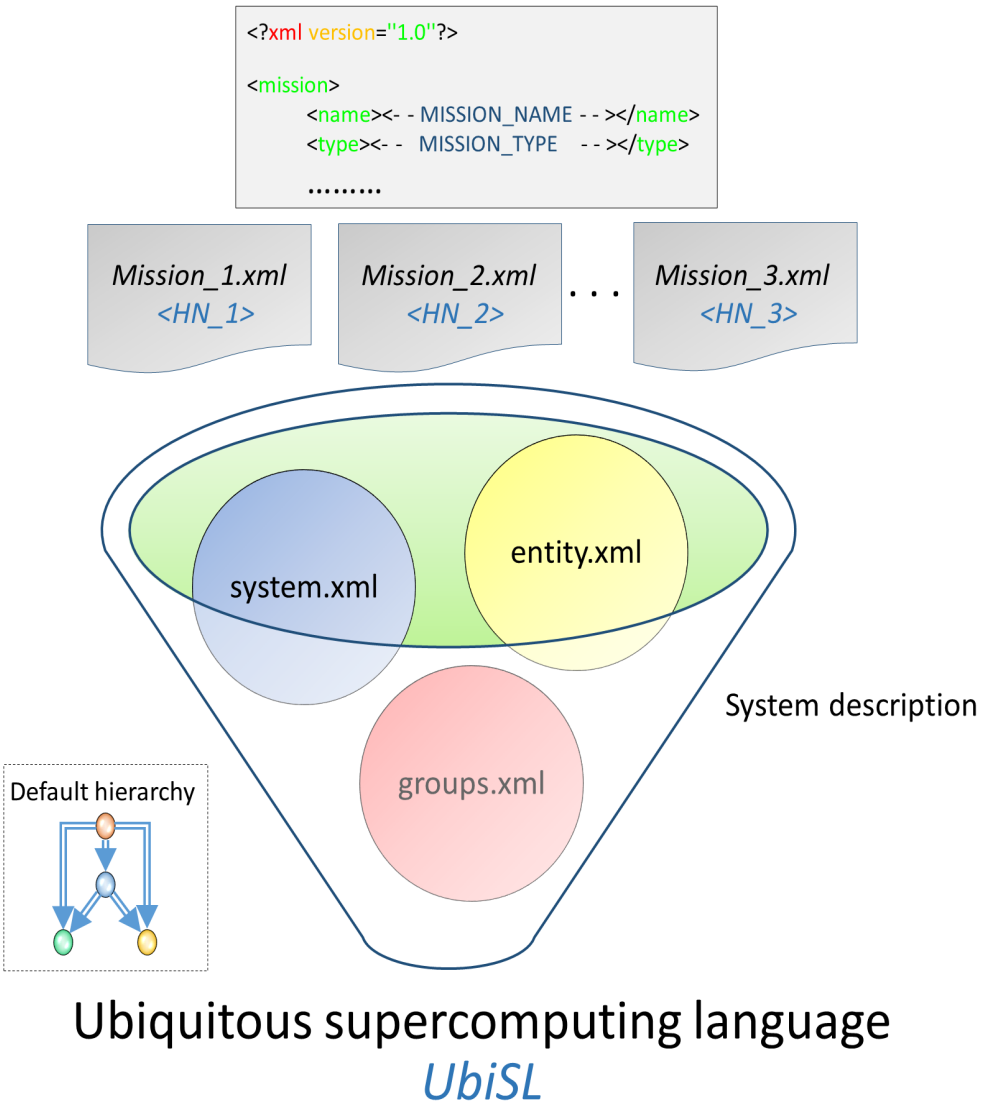


Figure III-7: *Ubiquitous supercomputing language (UbiSL). HN = Hierarchy Network*

Ubiquitous supercomputing language

Definition 10. *The ubiquitous supercomputing language - UbiSL is a high-level language that allows describing all concepts related to ubiquitous supercomputing.*

Remark. *The UbiSL consists of a set of XML templates for the description of systems, entities, nodes, links, groups, tasks, missions, etc. More information in chapter IV section IV.2, Table IV-4. Furthermore, the hierarchy network (HN) (Definition 14) can be changed for a specific mission, otherwise the default hierarchy (Figure III-8) will be maintained.*

In order to facilitate reutilization and separation between the system and the mission(s) carried by it, the concept of *system description* is defined as follows.

System description

Definition 11. *All the templates combined, except the mission-related templates, are called the System description.*

Remark. *The system description is dynamic, it can be modified to remove or add entities, i.e. to change the system size, groups, etc. Furthermore, in order to reuse a system for a different purpose, the mission template can be simply modified.*

This mechanism (separation between the system and the mission) provides flexibility, scalability and ultimately general-purpose. To manage such separation, a middleware is required (chapter IV section IV.3). Furthermore, entities, systems, etc., are classified according to a set of *framework classes* (chapter IV section IV.2) that can be adapted, reused or modified, providing therefore a strong mechanism for heterogeneity, scalability, etc.

Hierarchy networks provide all kind of flexible configurations, where any entity could be allowed to command orders over a subset of the entities, provided that the system operator and the system controller's hierarchy is maintained. For example, a pilot entity (human) can command orders over its piloted robot. Therefore. UbiSL allows to specify if a mobile entity, e.g. a robot, is to be automatic or piloted. Mixed configurations can be set as well, where a subset of robots behaves automatically and another subset is to be piloted. More information about this will be given in section III.7. Next section deeps upon hierarchy within ubiquitous supercomputing.

III.5 Hierarchy

In traditional supercomputing, hierarchy is mostly related with specific configurations in the job scheduling mechanism (batch system). For example, a certain user or group could be given a higher priority for job execution, even preempting running jobs if necessary. Moreover, resources (nodes, computing cores, etc.) access can be dependent on specific users. In this sense, hierarchy in the context of traditional supercomputing is *user-dependent*.

Based on such approach, under the ubiquitous supercomputing philosophy exposed in this Ph.D thesis, hierarchy is *entity-dependent*, i.e. all type of entities, including robots, etc., can have higher hierarchies than other entities. However, in contrast with traditional supercomputing, hierarchy in here describes the capacity an entity holds to command orders over other entities, going beyond solely job/task execution.

Orders

Definition 12. *An order is an instruction to be executed by a subset of the system's entities.*

Remark. *Default orders, e.g. execute mission, return home, etc., will be discussed in chapter IV middleware section IV.3.*

Nevertheless, the default order *execute mission* relates with task (job) execution, therefore ubiquitous supercomputing hierarchy includes the approach in traditional supercomputing and extend it by proposing orders such as return home, etc., aiming at increased control and security.

Deepening upon the definition of ubiquitous supercomputing systems, introduced in the lemma III.2, a system is a hierarchical network, whose default graph is displayed in Figure III-8.

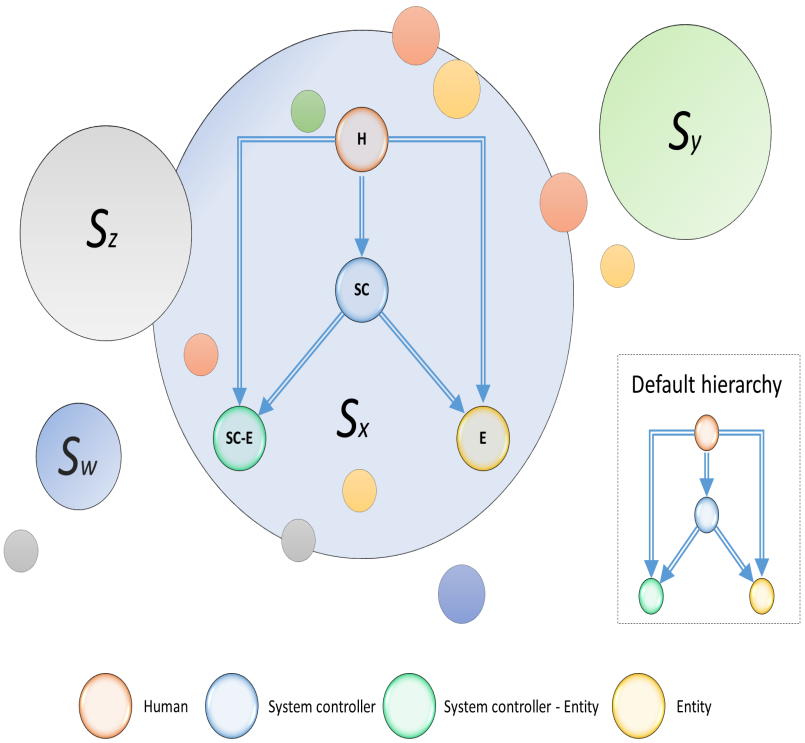


Figure III-8: *Ubiquitous supercomputing systems default hierarchy.* $S_{x,y,z,w} = System_{x,y,z,w}$, $H = Human\ system\ operator$, $SC = System\ Controller$, $SC - E = System\ Controller\ Entity$, $E = Entity$. *The circles represent systems*

While all kind of possible configurations are supported by the ontology, hierarchy's default approach is presented in definition 13. By default, only the human system operator and the system controller (SC), in automatic modes (see section III.7), can command orders over all entities as portrayed by figure III-8 (default hierarchy).

Ubiquitous supercomputing systems default hierarchy and system operator

Definition 13. *A ubiquitous supercomputing system is a hierarchical network, where a human entity, depicted as the system operator (SO), holds the highest hierarchy, followed by the system controller SC (non-human entity). Other entities, including the SC-E have no capacity for order commanding, in default mode.*

Remark. *Individual entities hierarchies can be set using the ubiquitous supercomputing language (section III.4). Moreover, the system controller (SC) is an entity as well (SC-E), as described by the ubiquitous supercomputing systems lemma III.2. More details about this will be given in chapter IV.*

An entity's hierarchy represents the level of authority that such entity exercises over other entities i.e. its capacity to command orders. This way, the hierarchy of a system can be represented as a directed network, the *hierarchy network*.

Hierarchy network

Definition 14. *A system's directed hierarchy network is an abstract representation of an ordering scheme i.e. the direction of an edge specifies the order sender and the order receiver.*

Remark. *The system's hierarchy network is not static i.e. it can be modified depending on the mission carried out by the system, by using the UbiSL.*

Hierarchy and ordering schemes are designed for two main objectives, *control* and *security*. In an increasing intelligent robotics world, it is important to maintain human authority and control. In addition, to prevent network attacks, i.e. hacking, all entities are aware of the hierarchy network and will not accept orders coming from entities that are not allowed to, even from inside the system itself. The individual hierarchy of an entity is calculated accordingly to the *Local Reaching Centrality (LRC) concept* (Mones *et al.* , 2012), which represents the proportion of entities that are reachable, i.e. entities accepting orders ($e \in E$), from the specific entity (i), where E is a set containing all entities, as defined in equation III.1.

$$LRC_i = \frac{e}{|E| - 1} \tag{III.1}$$

Where $|E|$ is the total quantity of entities in the system. Furthermore, the hierarchy of the system itself is calculated with the concept of Global Reaching Centrality (GRC) (Mones *et al.* , 2012), as portrayed in equation III.2:

$$GRC = \frac{\sum_{i \in E}(LRC_{max} - LRC_i)}{|E| - 1} \tag{III.2}$$

Where LRC_{max} corresponds to that of the entity with the highest hierarchy (human system operator) and $GRC \in [0, 1]$. Individual hierarchies can be modified via the UbiSL. However, the following considerations are strict within ubiquitous supercomputing systems:

- In the hierarchy network, the system controller is represented with two nodes, the system controller (SC) and the system controller entity (SC-E), i.e. the entity controller. More information about the two nodes will be given in chapter IV.
- The system controller (SC) is the non-human entity with the highest hierarchy.
- If the system includes human entities (desired), such entities are classified as the system operator (H), i.e. the person with the highest hierarchy in the system and *operators*, human entities controlling a set of entities, e.g. a pilot.

Therefore, for the default hierarchy (see Figure III-8), the following equations apply. Having H = system operator, SC = system controller, $SC - E$ = system controller entity and E = other entities.

Theorem 1 (System operator LRC)**.** *All entities are reachable from the system operator.*

$$LRC_H = 1 \tag{III.3}$$

Theorem 2 (System controller LRC)**.** *All entities, except the system operator, are reachable from the system controller.*

$$LRC_{SC} = \frac{|E| - 2}{|E| - 1} \tag{III.4}$$

Theorem 3 (System controller entity LRC). *No entity is reachable from the system controller entity.*

$$LRC_{SC-E} = 0 \tag{III.5}$$

Theorem 4 (Other entities LRC). *Entities, different from previous entities H and SC, cannot reach other entities.*

$$LRC_{E} = 0 \tag{III.6}$$

Therefore, the GRC for a ubiquitous supercomputing system with default hierarchy network, is calculated as portrayed by the following theorem.

Theorem 5 (Other entities LRC). *Having $|E| \in [3, \infty)$ and $|E| - 2$ entities with $LRC = 0$, the GRC of a ubiquitous supercomputing system, with the default hierarchy network is:*

$$GRC = \frac{|E|^2 - 3|E| + 3}{(|E| - 1)^2} \tag{III.7}$$

Given the mobile nature of a ubiquitous supercomputing system, individual $LRCs$ and consequently GRC are subject to variation during the execution of a mission. Therefore the *system hierarchy (SH)* is defined as the average GRC during the execution of a mission.

$$SH = \overline{GRC} \tag{III.8}$$

That is, GRC is to be computed at all time steps during the execution of a mission and averaged over the times it has been computed. Moreover, two aspects need to be considered when discussing hierarchy and ordering schemes.

- *Order acknowledgment time*, i.e. the delay between the commanding of an order and its acknowledgment, *order delay* ($D[O]$), represents an emergent result of the communications status (latency, bandwidth, etc) at the time the order was sent.
- Order commanding depends of the nature of an entity, i.e. non mobile entities will not be given motion-related orders but all entities will receive an order related to mission execution.

With this in mind, consider a set e_o, of entities entitled for the order o, the average delay for such order (D), i.e. *the average order delay* is defined as:

$$D = \frac{\sum_{i \in e_o} D(i)}{|e_o|}, \quad \text{with } e_o \subseteq E \tag{III.9}$$

Consequently, having a set O, composed of all orders, the *average orders delay* OD, i.e. for all orders is defined as :

$$OD = \frac{\sum_{j \in O} D(j)}{|O|} \tag{III.10}$$

Since a mission (section III.3), consists of a set of interconnected tasks, carried out by independent entities, SH and OD can give an estimation of the capacity of the system to perform a specific mission, in conditions where communications among entities can potentially be affected by disconnections or performance lack.

III.6 Stability

Analyzing how stable is a system, represented as a network, can be done via the evaluation of resistance and resilience. A system is resistant if it can endure external perturbations for a long time or if a higher perturbation needs to be performed to deviate the system from an optimal state. Moreover, a system is resilient if it can recover quickly from a perturbation and return to an optimal state.

The basic optimal state of a ubiquitous supercomputing system is such where at least the hierarchy network is maintained during times when orders are commanded. In other moments, each entity has sufficient autonomy to continue with its assigned tasks. However, other possible optimal states are desired. As mentioned in section III.3, non-beginning tasks are triggered by events. If connectivity among entities requiring such events is nonexistent, the mission can be compromised. Since the creation of an event depends on the duration of the software linked to a task, which can occur at any time, the most optimal state is such of a full graph.

Multiple optimal states can occur, when dealing with ubiquitous supercomputing systems. In Zamani & Vicsek (2017), it was found that complex hierarchical networks, such as a ubiquitous supercomputing system when performing a mission, can maintain several or many metastable states, depending on their initial configuration (at the beginning of a mission, where a full connectivity graph is in place) and the perturbations they are subject, which is known as *glassy behavior*, where the efficiency function associated to the performance of the network resembles the phenomena of the spin glass model (Mézard *et al.* , 1987), (Mezard & Montanari, 2009) and (Newman & Stein, 2013). While efficiency and what it actually means regarding ubiquitous supercomputing systems is out of the scope of this Ph.D thesis, its relationship with stability is discussed as follows. In fact, maximizing efficiency leads to hierarchical organizations, as found by Zamani & Vicsek (2017) and efficiency is typically considered as a stability measurement criteria (Jackson & Watts, 2002), (König *et al.* , 2012). Moreover, connectivity in a networked system is crucial to measure stability as found in Albert *et al.* (2000).

Consider, a ubiquitous supercomputing system with $|E|$ entities and links representing connectivity among entities. The nature of such links is intermittent during the execution of a real world mission and moreover technical features such as latency, etc., vary in time. Furthermore, entities, specially mobile ones, might result into battery draining or any sort of failure, affecting the network topology, i.e. entities can disappear from the network.

In order to observe the effect on stability when entities are lost or removed, the following numerical experiments (*attacks*) were performed. An attack consist on the removal of Q entities, $Q = [1, 2, 3, 4, ..|E|]$, i.e. attack Q consists of removing Q entities at once. After an attack, GRC as a connectivity indicator and efficiency (Zamani & Vicsek, 2017) are measured. Two approaches were applied:

- Entities with high LRC and their corresponding links are removed one by one, and this process is continued with low LRC entities (downward approach).
- Entity removal starts from those with low LRC, and continues up to those with high LRC (upward approach)

GRC and efficiency, of the resulting network, are measured after each attack. With the objective of analyzing the stability of networks with low GRC (lower than 0.5) and high GRC (higher than 0.5), under perturbations by entity removal, Q attacks, with Q = [1, 2, ..., 128] were performed in 32 networks (18 with high GRC and 14 with low GRC), each one with 16 local optimal states. Each attack Q was carried out according to the two approaches. Figure III-9-A shows that when entities, in networks with high GRC, are removed with downward approach, efficiency decreases faster than when entities are removed with upward approach. This is

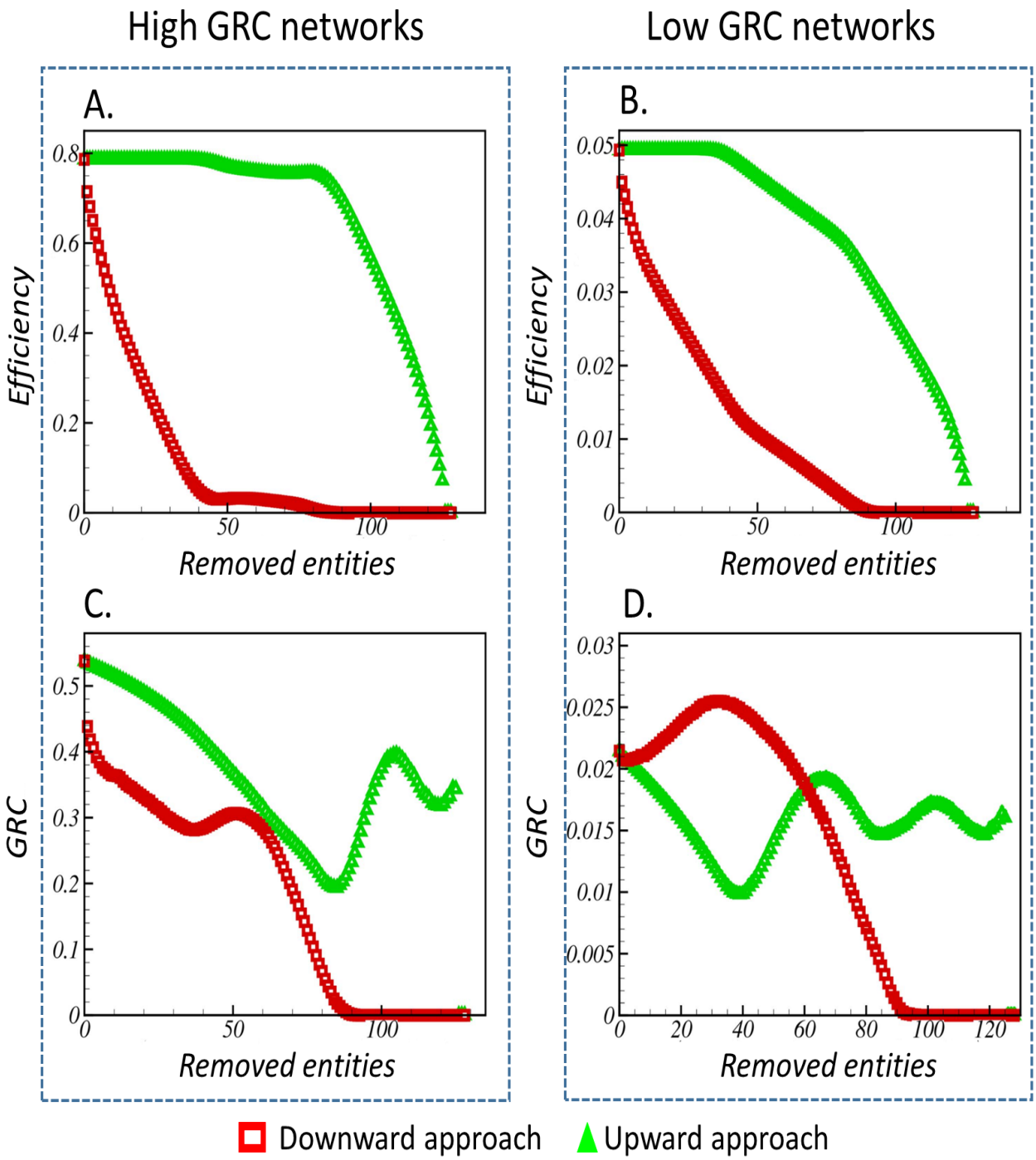


Figure III-9: *Efficiency and GRC after entity removal attacks with high (A and C) and low (B and D) GRC. High GRC: $GRC \in [0.5, 1.0]$ and low GRC: $GRC \in [0.0, 0.5)$. For Q = E entities removed (x-axis), the y-axis represents efficiency and GRC. The fluctuation-like behavior of the plots (especially in C and D) is likely due to the finite (relatively small) size of the networks with only a discrete set of LRCs.*

expected, according to the model presented in Zamani & Vicsek (2017), where entities with high LRC contribute to the efficiency of the network to a greater extent that those with low LRC. A similar behavior appears for networks with low GRC, as depicted in Figure III-9-B. However, with upward approach, only after removing more than 80, out of 128 entities, efficiency drops significantly for high GRC networks, whilst the removal of more than 40 entities causes the efficiency to drop significantly for low GRC networks. These results suggest that *networks with*

high GRC (more hierarchical) are stable and efficient, even when losing large quantities of entities, in comparison with low GRC networks. Therefore, the following finding is a result of this Ph.D thesis.

Finding: Stable hierarchical directed networks

Finding 1. *Hierarchical directed networks with a GRC higher than* 0.5 *are more stable that networks with a lower GRC.*

Moreover, these conclusions suggest that entities prone to perturbations, such as those with a mobile nature, should not be given a high LRC, as devised in the default hierarchy network (see Figure III-8 and definition 13).

Regarding GRC, two conclusions can be drawn from Figure III-9-C. First, in both approaches, GRC decays as expected until around 80 entities are removed, faster again with the downward approach, even considering the slight GRC raise in the same approach. Second, a significant GRC raise occurs in upward approach. Potentially certain optimal states could actually result in GRC increases after being perturbed, an indication of resilience.

Concerning low GRC networks, Figure III-9-D shows fluctuating behavior with attacks according to upward approach. These fluctuations demonstrate that networks with low GRC are not stable against external perturbations, such as entity removal. Again, there is no relevant GRC increase in downward approach, followed by a rapid decay towards GRC equals zero.

Therefore, both efficiency and connectivity, as represented by GRC, lead to the conclusion that a network with a GRC higher than 0.5 is more stable than those with a lower GRC. While the numerical experiments performed in here, targeted networks whose efficiency function corresponds to that in the model presented in Zamani & Vicsek (2017), GRC is calculated according to the general model presented in Mones *et al.* (2012), which suggest that these results are applicable to general hierarchical directed networks, such as ubiquitous supercomputing systems. Nevertheless, this mathematical approach to stability needs further research to reach deeper conclusions and ultimately real experimentation with increasing quantity of entities, but this is out of the scope of this Ph.D thesis.

In regard to resilience, another interpretation for an optimal state, is such in which orders, events and data resulting from the mission tasks (e.g. pictures, etc.) are correctly distributed among the entities, i.e. a *single-image* system. Given that potential disconnections cannot be fully avoided in real-world missions, it is important to provide synchronization mechanisms when connectivity is granted and when restored after a perturbation, as it will be discussed in chapter IV section IV.3. Once connectivity is restored, a synchronization strategy facilitates an optimal state, i.e. resilience.

III.7 Automation

As discussed in chapter II section II.2, an increasing automation level, is a desired feature for ubiquitous robotics systems. However, in an open letter to the United Nations (UN) (Musk, 2016), 116 experts in AI and robotics, requested a ban over the use of the so-called killer robots. The authors requested UN to find ways to protect humanity from a third revolution in warfare powered by autonomous weapons. In this sense, while fully automatic systems can find all kind of uses, human presence, at least in supervision quality, is very important under the scope of ubiquitous supercomputing.

Three automation modes are provided by default, within ubiquitous supercomputing:

1. *piloted-complete mode*
2. *automatic with human system operator* and
3. *automatic full*

The three modes are increasingly automatic (1-3), with mode 2 representing the default hierarchy network. Having $|E|$, the quantity of entities in the system and applying equations III.1 (LRC) and III.2 (GRC), from Mones *et al.* (2012), to the three hierarchy networks in the automation modes, the following details are introduced.

III.7.1 Piloted-complete mode

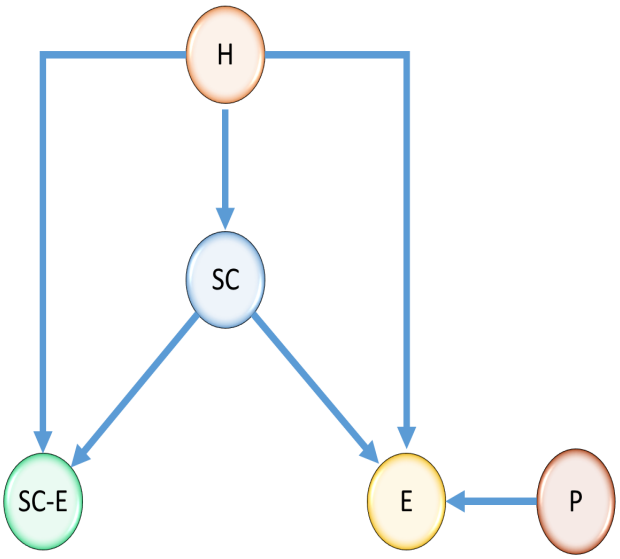


Figure III-10: *Piloted-complete mode. H = Human system operator, SC = System controller, $SC-E$: System controller Entity, E = Entity, P = Pilot Entity*

A. Piloted-complete mode

Definition 15 (Piloted-complete mode). *Each mobile entity has a pilot which controls the specific entity. However, software agents, i.e. system controller, entity controllers and the human operator are part of the system as portrayed by Figure III-10, thus $min|E| = 5$.*

$$LRC_H = \frac{|E|+1}{2(|E|-1)} \tag{III.11}$$

$$LRC_{SC} = \frac{1}{2} \tag{III.12}$$

$$LRC_{SC-E} = 0 \tag{III.13}$$

$$LRC_E = 0 \tag{III.14}$$

$$LRC_P = \frac{1}{|E|-1} \tag{III.15}$$

correspondingly, the GRC for the piloted-complete mode is:

$$GRC = \frac{|E|^2 - 2|E| + 3}{2(|E| - 1)^2} \tag{III.16}$$

III.7.2 Automatic with human system operator mode

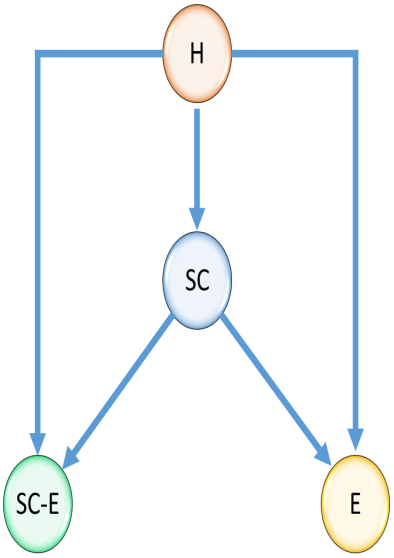


Figure III-11: *Automatic with human system operator mode. H = Human system operator, SC = System controller, $SC - E$: System controller Entity, E = Entity*

B. Automatic with human system operator mode

Definition 16 (Automatic with human system operator mode). *A single human entity controls the entire system (system operator) and each mobile entity is controlled by an agent as portrayed by Figure III-11. No pilots included. This is the default hierarchy mode. Moreover, $min|E| = 3$.*

Remark. *Multiple system operators can be put in place. This is manageable via the user system and other mechanisms to be described in chapter IV section IV.3.*

Remark. *Following equations III.17 to III.21 are the same that those introduced in section III.5, default hierarchy mode. They are presented in here for textual consistency.*

$$LRC_H = 1 \tag{III.17}$$

$$LRC_{SC} = \frac{|E| - 2}{|E| - 1} \tag{III.18}$$

$$LRC_{SC-E} = 0 \tag{III.19}$$

$$LRC_E = 0 \tag{III.20}$$

correspondingly, the GRC for the automatic with human system operator mode is:

$$GRC = \frac{|E|^2 - 3|E| + 3}{(|E| - 1)^2} \tag{III.21}$$

III.7.3 Automatic full mode

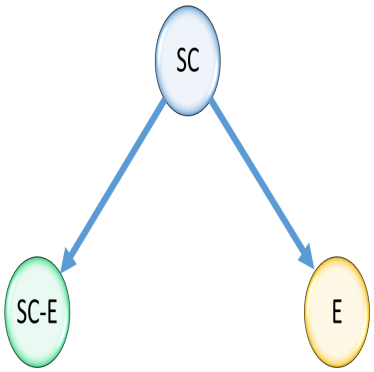


Figure III-12: *Automatic full mode. H = Human system operator, SC = System controller, $SC-E$: System controller Entity, E = Entity*

C. Automatic full mode

Definition 17 (Automatic full mode)**.** *The system is fully automatic, i.e. no human entities, as portrayed by Figure III-12. This mode is discouraged. However, if the system operator does not intervene, mode 2 (subsection III.7.2) will result into the current mode with supervision. Thus, $min|E| = 2$.*

Remark. *The ARCHADE supports this mode but the user can regain control by launching the middleware hydra service (chapter IV section IV.3).*

$$LRC_{SC} = 1 \tag{III.22}$$

$$LRC_{SC-E} = 0 \tag{III.23}$$

$$LRC_{E} = 0 \tag{III.24}$$

correspondingly, the GRC for the automatic full mode is:

$$GRC = 1 \tag{III.25}$$

III.7.4 Stability and automation modes

As it can be observed in Figure III-13, all default automation modes lead to stable systems. In the case of the *piloted-complete mode* (Figure III-13-A) and with increasing quantity of entities, GRC decreases asymptotically to 0.5. Regarding case B, *automatic with human system operator*, GRC increases with the quantity of nodes, approaching asymptotically to one as portrayed by Figure III-13-B. Finally, in case C, *automatic full*, (Figure III-13-C) GRC is equal to one for any quantity of entities. Therefore, the automation modes provide *stable scalability*.

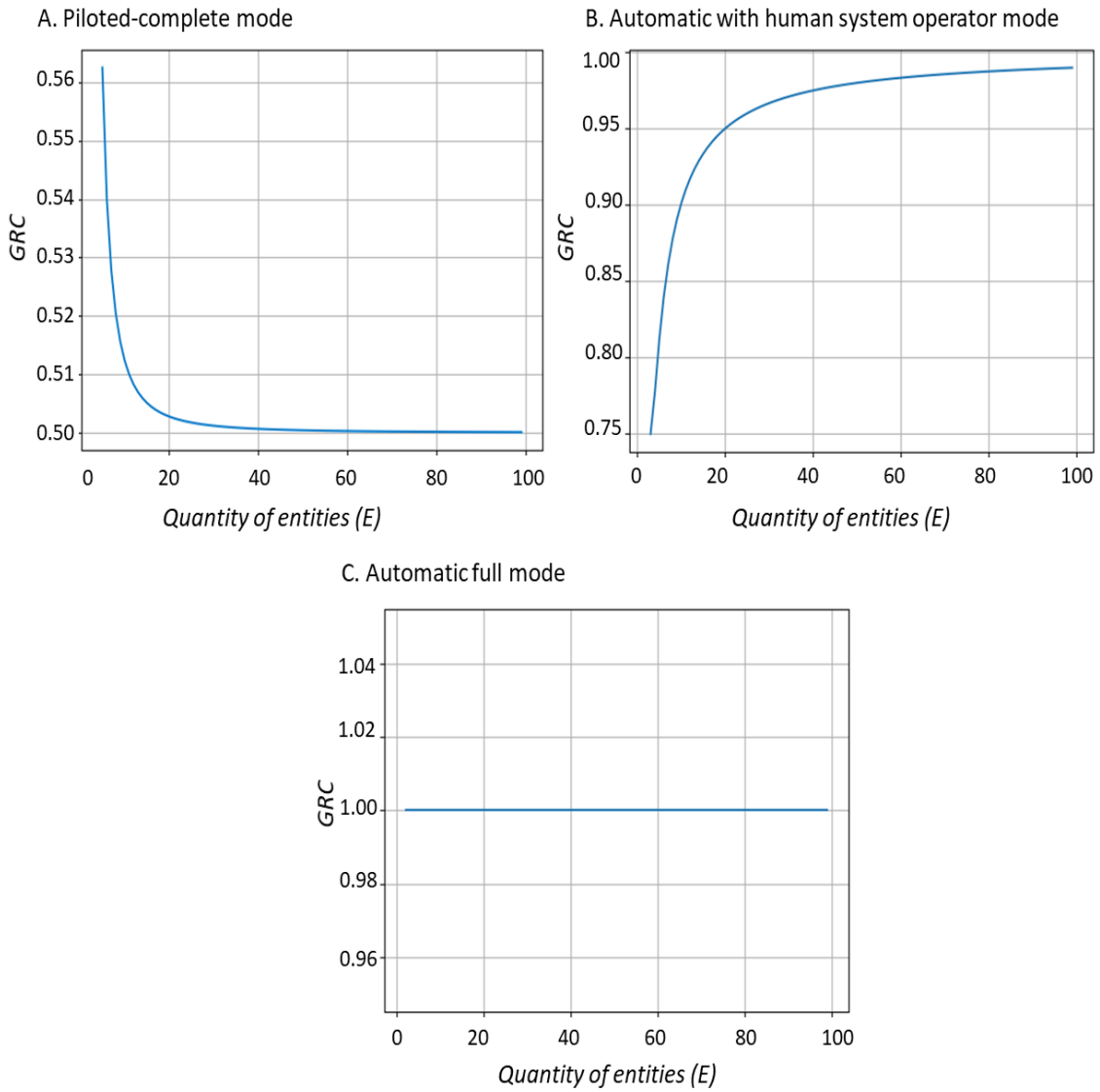


Figure III-13: *Automation modes and stability. All default automation modes lead to stable systems, i.e. GRC is never less than 0.5*

The three cases are devised on purpose to facilitate the creation of stable and scalable systems using the conclusions in section III.6. However, while the three automation modes are provided by default, other approaches can be used by setting hierarchy networks using the UbiSL. Examples of such hierarchy networks include those with lowest hierarchy entities capable of commanding orders (e.g. mobile entities with $LRC > 0$), one pilot controlling several mobile entities, mixed modes i.e. mobile entities with pilot and mobile entities without pilot, etc. Nevertheless, the strict considerations introduced in section III.5 must be maintained and it must be beard in mind that different strategies from the three default cases might result in unstable systems.

The automation modes offer *user-transparency* in addition to stability and scalability. In the case of the automatic with human system operator mode, a single user (*system operator*) can control the entire system, i.e. the user perceives the distributed system as a single cohesive unit. Moreover, the piloted complete mode offers user-transparency given that there is a human system operator, but it also provides extra control by having a pilot per mobile entity. Finally, the automatic full mode does not integrate users in the system, beyond the initial launch of the mission, therefore user-transparency does not pertain in here. However, human presence can always be granted using the hydra service to be discussed in chapter IV section IV.3.

III.8 Summary and discussion

While supercomputing can be found in practically everything nowadays, robotics has been mostly absent of its utilization when dealing with multiple entities. However, current technological advancements in embedded computing, networking and other relevant aspects such as increasingly complex missions and a whole new spectrum of applications, facilitate and inspire the union between multi-robot systems and supercomputing. Exploiting such ideas, this thesis has proposed High Performance Robotic Computing (see section III.1), as an adaptation of traditional supercomputing to the nuances of robotic entities, contributing therefore to the ubiquity of supercomputing.

Yet, supercomputing within a new frame of mind, in which all of its capacities, not only computing efficiency, are used to create systems of systems, such that can scale inwards and outwards, act as a single cohesive unit despite being composed of independent distributed units and be capable of execute any type of mission. The goal of bringing supercomputing truly everywhere requires a translation from philosophy to reality, accomplished via the ubiquitous supercomputing ontology introduced in section III.2. With the ontology, multi-robot HPRC settings, traditional supercomputing, computing-less devices and humans are integrated into single systems used for general-purpose computing missions (section III.3). To go further in the translation from philosophy to reality, the ubiquitous supercomputing language has been presented in section III.4. Moreover, throughout the chapter, several aspects related to a rethought supercomputing have been introduced, hierarchy (section III.5), stability (section III.6) and automation (section III.7).

However, to ultimately bring philosophy into reality, next chapter introduces The ARCHADE, a ubiquitous supercomputing framework and middleware.

Un café, a las orillas de algún mítico ró, o cerca al mar, o en una plaza tan antigua como algunas de las civilizaciones que maravillaron la historia de nuestra humanidad. Un café para así poder preguntarte todas las cosas que nunca he entendido, que con locura deseo comprender. Un café para darme cuenta que con solo verte, al final, ninguna respuesta importa para nada

— Leonardo CF

IV

The ARCHADE

In the previous chapter, the ideas behind the philosophy of ubiquitous supercomputing were introduced, including a discussion about High Performance Robotic Computing (HPRC) and its hierarchical union with traditional supercomputing and humans. To bring ubiquitous supercomputing into reality, this chapter introduces The ARCHADE.

The ARCHADE (TAC) is a loosely-coupled component-based, fully general-purpose ubiquitous supercomputing framework and middleware, written in Python[1], designed for the deployment, implementation and operation of ubiquitous supercomputing systems. It consists of:

- A set of classes, e.g. *thing*, *robot*, *drone*, etc.
- A set of templates to describe systems, entities, hierarchy networks, missions and to write general-purpose computing missions software, i.e. the *Ubiquitous supercomputing language (UbiSL)* (see chapter III section III.4).
- A set of services to be used for the building and operation of systems described according to the *ubiquitous supercomputing ontology* (see chapter III section III.2).

Furthermore, The ARCHADE is a hierarchical multi-agent technology whose main objective is to facilitate the execution of any type of mission over any ubiquitous supercomputing infrastructure. To do so, it provides five main functionalities/components, as detailed by Figure IV-1: *Application Programming Interface (API)*, *Framework*, *Middleware*, *SimPlat* and *PLUS*.

[1]TAC is originally written in Python 2.7 but it is currently upgraded to Python 3

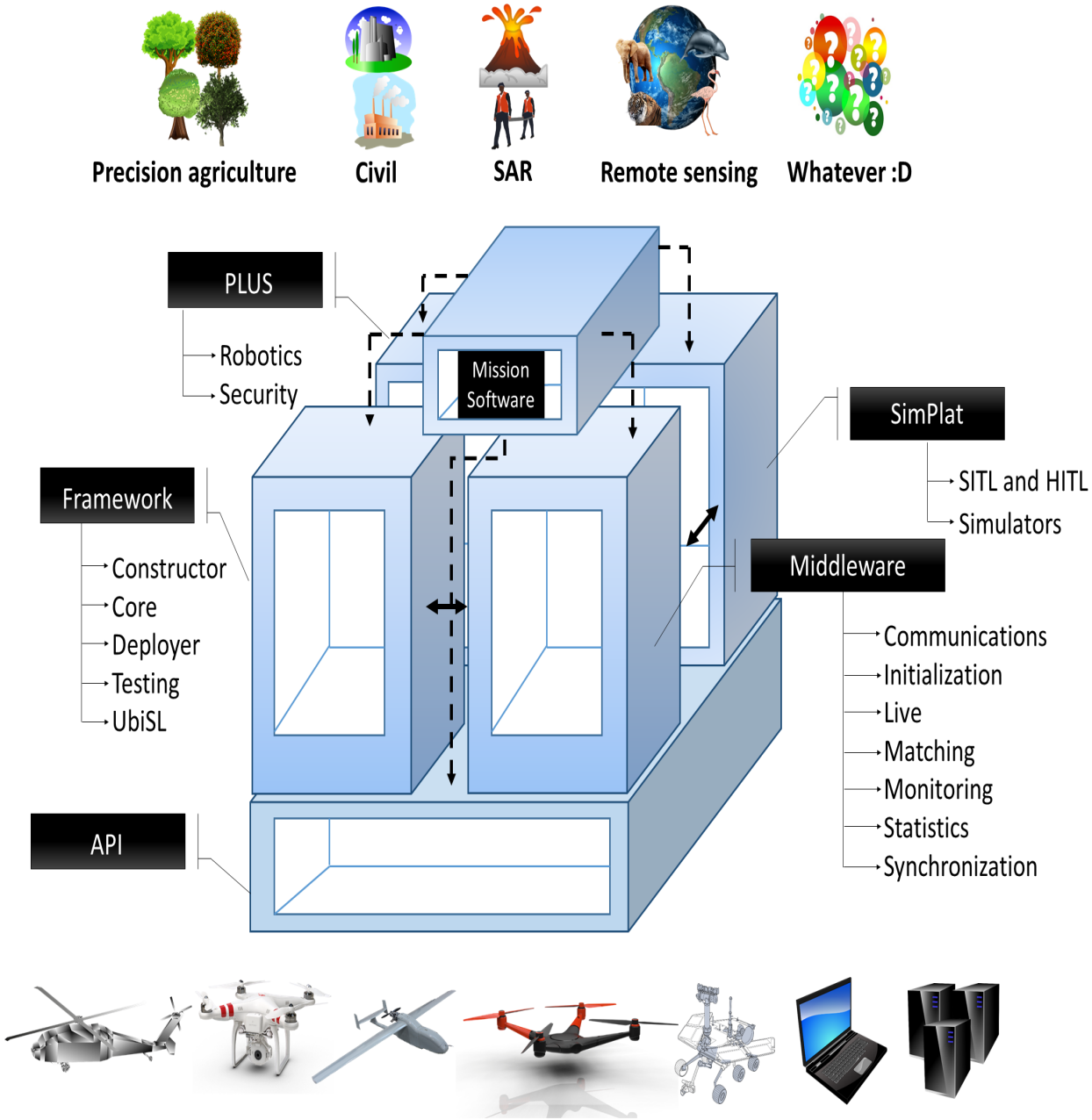


Figure IV-1: *The ARCHADE, general-purpose ubiquitous supercomputing framework and middleware. HPC = High Performance Computing, HPRC = High Performance Robotic computing, CLD = Computing-less devices, H = Humans. SITL = Software In The Loop, HITL= Hardware In The Loop, UbiSL = Ubiquitous Supercomputing Language.*

The ARCHADE targets the guidelines for ubiquitous robotics frameworks, suggested by Jiménez-González *et al.* (2013), discussed in chapter II section II.2, as portrayed by Table IV-1.

Table IV-1: *Guidelines for current and future ubiquitous robotics frameworks from Jiménez-González et al. (2013)*

Guideline	TAC's approach
General purpose	Standard technologies used in all the HPC software layers. Multipurpose and multi-user HPC and HPRC infrastructures. Framework (section IV.2), middleware (section IV.3), API (section IV.1), general purpose computing missions (see chapter III section III.3), etc.
Modular and flexible architectures	The ARCHADE framework inheritable classes. Independent and modular middleware and framework services, TCP / IP communications
Openness	The ARCHADE is not open-source but cooperation is encouraged.
APIs, reusable code and standardized interfaces	The ARCHADE's API. In addition, agents (live service section IV.3) can be modified to add extra functionalities. Moreover, communications mechanisms can be integrated with other technologies such as ROS, Data Distribution Services (DDS) (Pardo-Castellote, 2005) i.e. for real time systems, etc. Furthermore, FLEX templates (section IV.2) can be used to build mission software, new classes, etc.
Availability of suitable usability tools	Future work
Remote execution	All communications are based on TCP / IP. Moreover companion computers can be accessed via SSH or other protocols as supported by any mainstream operating system
Experiments to real applications	The ARCHADE facilitates transition from experimentation to real-world applications. It transparently allows connection with simulated or real autopilots. More information in section IV.5

While The ARCHADE is not a ubiquitous robotics framework, ubiquitous supercomputing does contribute to the ubiquity of robotics by facilitating the implementation of smarter multi-robot systems based on supercomputing strategies. Moreover, currently The ARCHADE is restricted to unmanned vehicles but its flexibility can be used to include all kind of robots into a TAC-based system.

In addition, The ARCHADE has followed a set of designing and development principles to be presented in Table IV-2.

Table IV-2: *The ARCHADE design and development principles*

Design and development principles	TAC's approach
Object-Oriented Programming (OOP)	The ARCHADE followed the OOP paradigm in all its core functionalities to guarantee reusability and flexibility.
Component-based	As mentioned before, The ARCHADE is currently made of five main components. Moreover, each component is made of multiple subcomponents representing specific services.
Loose coupling	Each component is self-contained and interaction between components at execution level is managed via the middleware services.
From scratch	The ARCHADE was designed and developed from scratch. The utilization of existing base technologies, e.g. ROS, was discouraged in order to attain maximum control. However, flexibility strategies can be used to facilitate future interaction with such base technologies.
Simplicity	Related functionalities have been packed into specific libraries without co-dependency. With this approach, each feature is kept as simple as possible.
Security	Communications and interactions between agents and services are managed via secured approaches to be described in section IV.3. This is specially important with the usage of embedded companion computers running standard operating systems subject to network and hacking attacks.

Implementation details are kept hidden on purpose. Following sections describe the five components of The ARCHADE.

IV.1 Application Programming Interface

The ARCHADE's Application Programming Interface (API) can be used to build mission tasks software and it has been used to create the remaining four components. It currently includes five groups of libraries[2], as it can be observed in Figure IV-2 and its main objectives are:

- To be used to write new TAC features.
- To be used in the development of mission tasks software.
- To be used to interact with existing services.

Moreover, this section describes the base API. However, the framework, middleware, SimPlat and PLUS components include their own API that can be used as well for the three previous objectives.

[2]The API currently includes more than 30 libraries. More are currently being developed

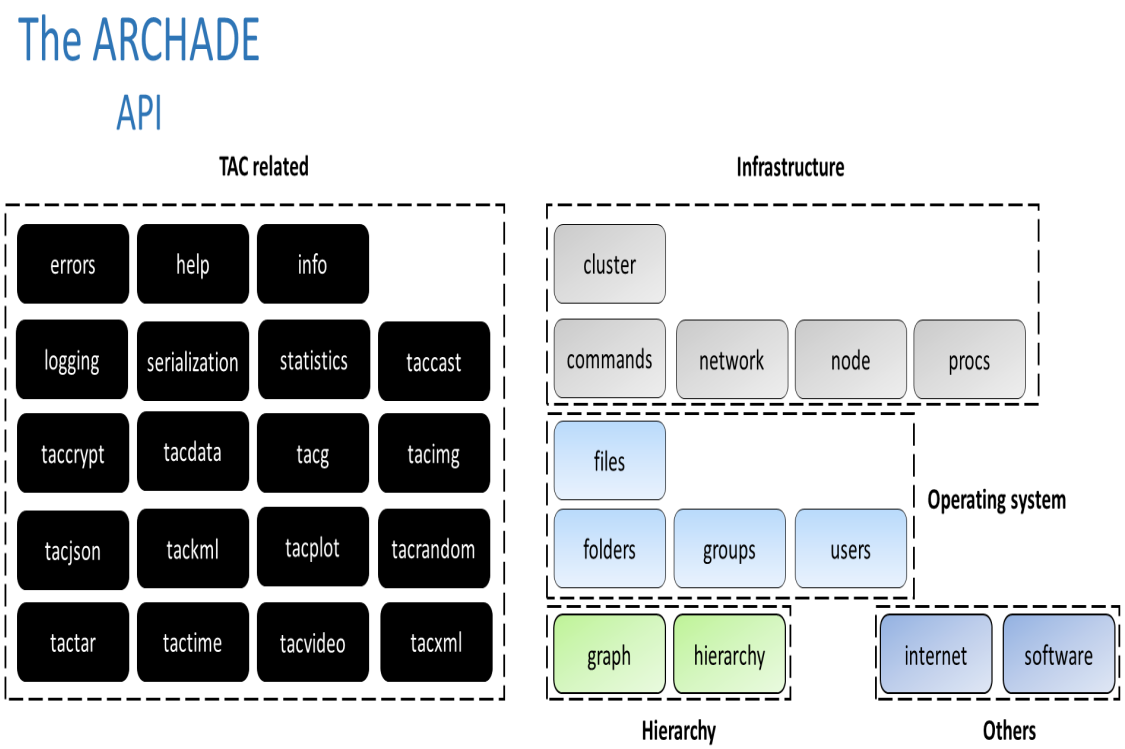


Figure IV-2: *The ARCHADE Application Programming Interface (API). Five groups, for a total of more than 30 libraries: TAC, infrastructure, operating system, hierarchy and others*

The mentioned five groups are described in the following:

- *TAC related*: Libraries used for TAC specific purposes such as error handling (*error*), object serialization (*serialization*), time management (*time*), etc.
- *Infrastructure*: Libraries for interaction with the supercomputing infrastructure e.g. individual nodes (*node*), complete HPC or HPRC clusters (*cluster*), network (*network*), etc.
- *Operating system*: Libraries for file, folders, users, etc., manipulation e.g. telemetry management, user creation, etc.
- *Hierarchy*: Libraries for system analysis, from network science approach (*graph*) and hierarchy management.
- *Others*: Libraries for internet accessing and automatic software installation, even during the execution of a mission.

The *tacimg* and *tacvideo* libraries are particularly important as they are being used for the development of the vision-based navigation service, part of the middleware component. The *tacimg* library currently provides more that 20 image transformations using OpenCV, that can be used by the *tacvideo* library to process live feeds. Moreover, such processed images will be used by the AI component, currently in development as well.

The framework, middleware, SimPlat and PLUS components were written using the API libraries. For example, operating system interaction is done via the *commands* library and cluster-level interaction is managed via the *cluster* library, for instance SSH connectivity towards a subset of the nodes, etc. Moreover, users developing mission software can import the libraries into their own software. In fact, the other components, e.g. framework, middleware, etc., include their own API libraries (not portrayed in Figure IV-2). For example, the middleware API can be used to allow mission software to manage vehicles motion, create events, actions, etc.

IV.2 Framework

The ARCHADE framework's objective is to serve as the bedrock for the development of applications in fields such as precision agriculture, search & rescue, construction, insurance claims, oil & gas, real state, e-sports and every field where ubiquitous supercomputing systems result advantageous.

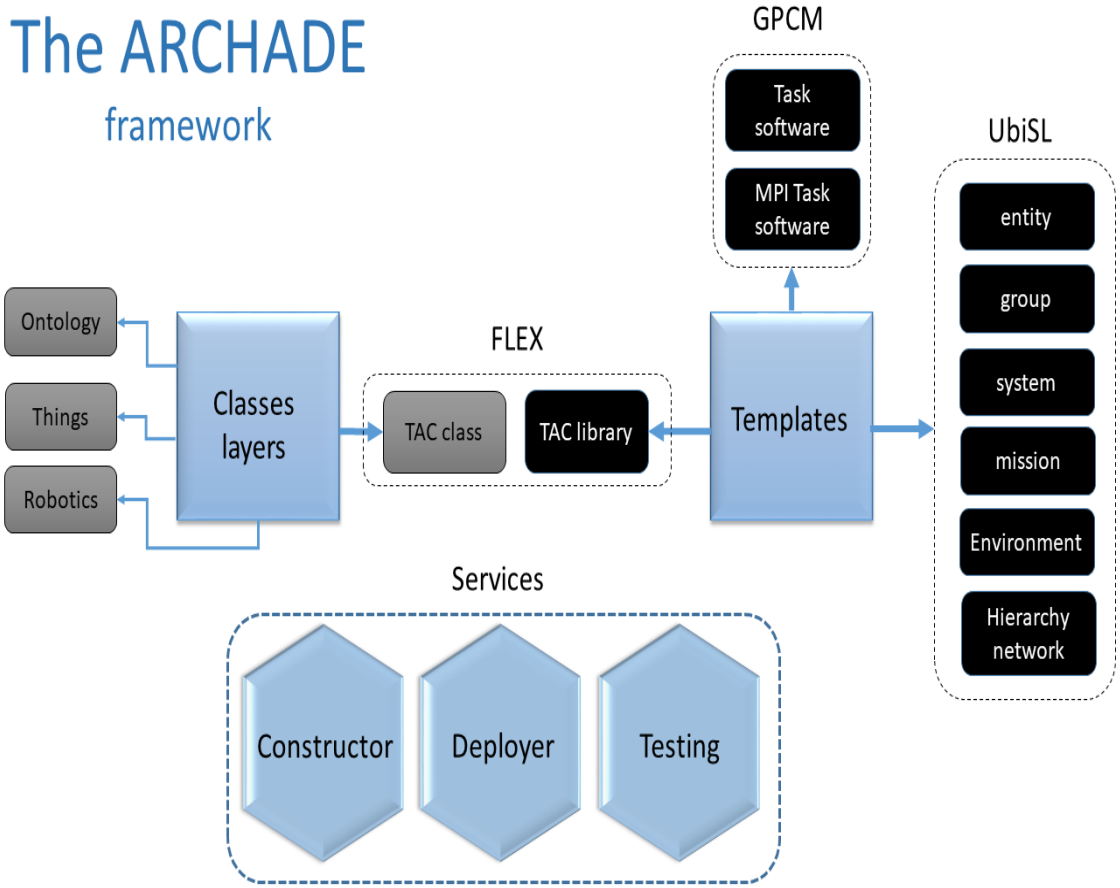


Figure IV-3: *The ARCHADE framework. The framework consists of a set of classes, templates and services for the creation of ubiquitous supercomputing systems. GPCM = General-Purpose Computing Mission, UbiSL = Ubiquitous supercomputing language*

Figure IV-3 introduces The ARCHADE framework, which consists of a set of classes layers (gray boxes), templates (black boxes) and services (hexagonal blue boxes), allowing the creation of systems that can be described according to the ontology.

The *FLEX* mechanism is intended to extend TAC's functionalities e.g. to write new TAC libraries (for the API, the framework, etc.) or new classes to augment TAC's heterogeneity. For example, the *dkrobot* class, using DroneKit, was created by inheriting from the class *robot*, part of the robotics classes layer, therefore new classes can be created by inheriting from such class, e.g. to support MAVROS, etc. In addition, the GPCM (chapter III section III.3) mechanism provides templates for mission software development. Both the FLEX and the GPCM mechanisms are described in Table IV-3. Furthermore, Table IV-4 describes the ubiquitous supercomputing language (chapter III section III.4) templates.

The UbiSL templates account for the UbiSL implementation and are used to set the system description (definition 11), which can be modified to allow the inclusion of new entities, new groups, new nodes per entity, etc. (*inward and outward scalability*). Moreover, by replacing the mission template, the system can be reutilized (*multi-purpose*).

Table IV-3: *TAC software templates*

Software template	Description
TAC library and TAC class	Aiming at flexibility (FLEX), The ARCHADE includes templates to create new TAC libraries and classes to be included in any of the components or to facilitate the creation of new services, components, etc.
Task software	In order to facilitate mission software development, The ARCHADE provides the task software template. This template includes coded functions for events and actions creation, etc. Moreover, existing software can be wrapped within this template.
MPI task software	Similar to the task software template, but using MPI (i.e. distributed software) to be executed by multiple entities in parallel.

Table IV-4: *UbiSL: TAC templates*

UbiSL template	Description
Entity	The entity template is used to set its name, class (e.g. drone, rover, etc.), role(s), nodes (hostnames, IPs, network interfaces, allowed users, roles, etc), embedded CLD e.g. sensors, cameras, devices, GPUs, etc. In addition, UbiSL supports two general modes, *simulation* and *real* i.e. changing the value of the field *mode* in this template, transparently enable the entity to proceed from experimentation to real-world missions. Even a subset of the entities can be set in simulation mode while another in real mode, allowing increasing testing levels – mandatory template.
Group	The group template is used to define a group of entities and specify the roles of each entity within the group. Multiple groups can be set.
System	The system template is used for high-level description, i.e. entities names, groups names, etc. Moreover, this template can be used to set multi-system interfaces, i.e. the system type is to be set as *multi* (see chapter III section III.2 Figure III-5) – mandatory template.
Mission	The mission template is used to describe the General-Purpose Computing Mission (GPCM) as a tasks' tree (parents-children), where each task is specified by the software used to carry it, software parameters, its type (blocking, execution, necessity, etc.), requirements (e.g. cameras, sensors), triggering events, etc. Furthermore, the template is used to set the mission area, i.e. longitude, latitude, length, shape, etc. The template can be used to describe task-less missions as well. – mandatory template.
Environment	Similar to the system template. However, non-system entities can be described, i.e. mission targets, etc. This template is currently being modified and enhanced. This is an optional template.
Hierarchy network	The HN template is used to set the system's hierarchy network. This is an optional template. If not set, the default hierarchy network is maintained (definition 13).

With simplicity in mind, it is recommended to link each task with a single-purpose software instead of multi-purpose software, which could result difficult to distribute or coupled with output events. Nevertheless, existing software can be linked to a task by wrapping it within the task or MPI task software templates, specifying its output events and consequently avoiding software rewriting.

Framework classes are designed to ease new classes creation in each layer or even complete new layers by inheriting from the main layer, the TAC class. While ubiquitous supercomputing differs from the general ideas behind the Internet of Things, given that systems are fundamentally supercomputing-based, the layer class *thing* opens up the spectrum of possibilities, i.e. supercomputing at the edge. For example, sensors or actuators (CLD) classes can be created by inheriting from such class. In addition, as previously stated the robotics layer has been used to create the *dk* classes, a set of classes, i.e *dkrobot*, *dkdrone*, *dkplane* and *dkrover*, using TAC's services and DroneKit (3DR, n.d.b), a Python library used for autopilot interaction. Finally, the framework includes the following services:

- *Constructor*: The constructor service is used to build workspaces for new missions, carried out by a ubiquitous supercomputing system or for mission test cases. A *workspace* consists of a set of folders organized for the execution of a *mission test case*, i.e. a single occurrence of a mission carried out by the ubiquitous supercomputing system. The constructor service uses a set of *workspace templates*. Furthermore, the constructor service adapts the HPC software layers, e.g. users, etc., to the specific mission test case.
- *Deployer*: The deployer service provides automatic deployment of ubiquitous supercomputing infrastructures, i.e. the automatic installation and configuration of the HPC software layers, currently in computing units using Debian-based operating systems e.g. Ubuntu, Raspbian or Kali Linux. This service embeds the HPC-ROS package (chapter III section III.1.1). The package has been tested on Raspberry Pi computing boards and common desktop computers, including virtual machines. Furthermore, deployment of new types of HPC clusters, Cloud provisioning mechanisms, Docker, etc., are being designed to be included in this service.
- *Testing*: The testing service is used for *applications layer testing*, i.e. MPI, in order to guarantee the correct deployment of a ubiquitous supercomputing infrastructure and estimate the available computing power. The high-level testing guarantees the correctness of all the HPC software layers implementation.

Each service includes an API related with its specific scope. For example, the deployer service provides an API for HPC software layers installation and configuration. This API is being used to provide The ARCHADE with live installation and configuration capacities to be used by the middleware, during the execution of a mission, i.e. to automatically integrate a new entity into a system executing a mission.

By using the framework, system developers need to focus solely on developing mission software, i.e. the software associated to each task. Moreover, FLEX templates can aid in such endeavor, smoothing integration with TAC, specially with the middleware services introduced in the following section.

The ARCHADE's framework most important contribution and objective is to facilitate the use of supercomputing in existing and new applications, e.g. precision agriculture, entertainment, etc., and therefore contributing to the ubiquity of supercomputing. Furthermore, every class, service and template is self-contained with the objective of providing simplicity and flexibility. In this sense, each system is self-contained as well so in order to interact with other systems, interfaces are to be defined in the UbiSL system template. With this approach, every system, part of a multi-system configuration, can focus on its specific current mission while still interacting with other systems.

In plain terms, the framework is all about creating ubiquitous supercomputing systems and mission software while the middleware, to be described in the next section, is about operating such system during the execution of a mission.

IV.3 Middleware

The ARCHADE middleware's objectives are:

- Transform a set of independent entities into a single cohesive unit. This is the main idea behind a rethought supercomputing, as proposed by this Ph.D thesis.
- Separate and connect the ubiquitous supercomputing infrastructure, i.e. the complete set of entities configured as a supercomputer, from the mission carried by it. By doing so, the same infrastructure can be used for different purposes (*multi-purpose*).

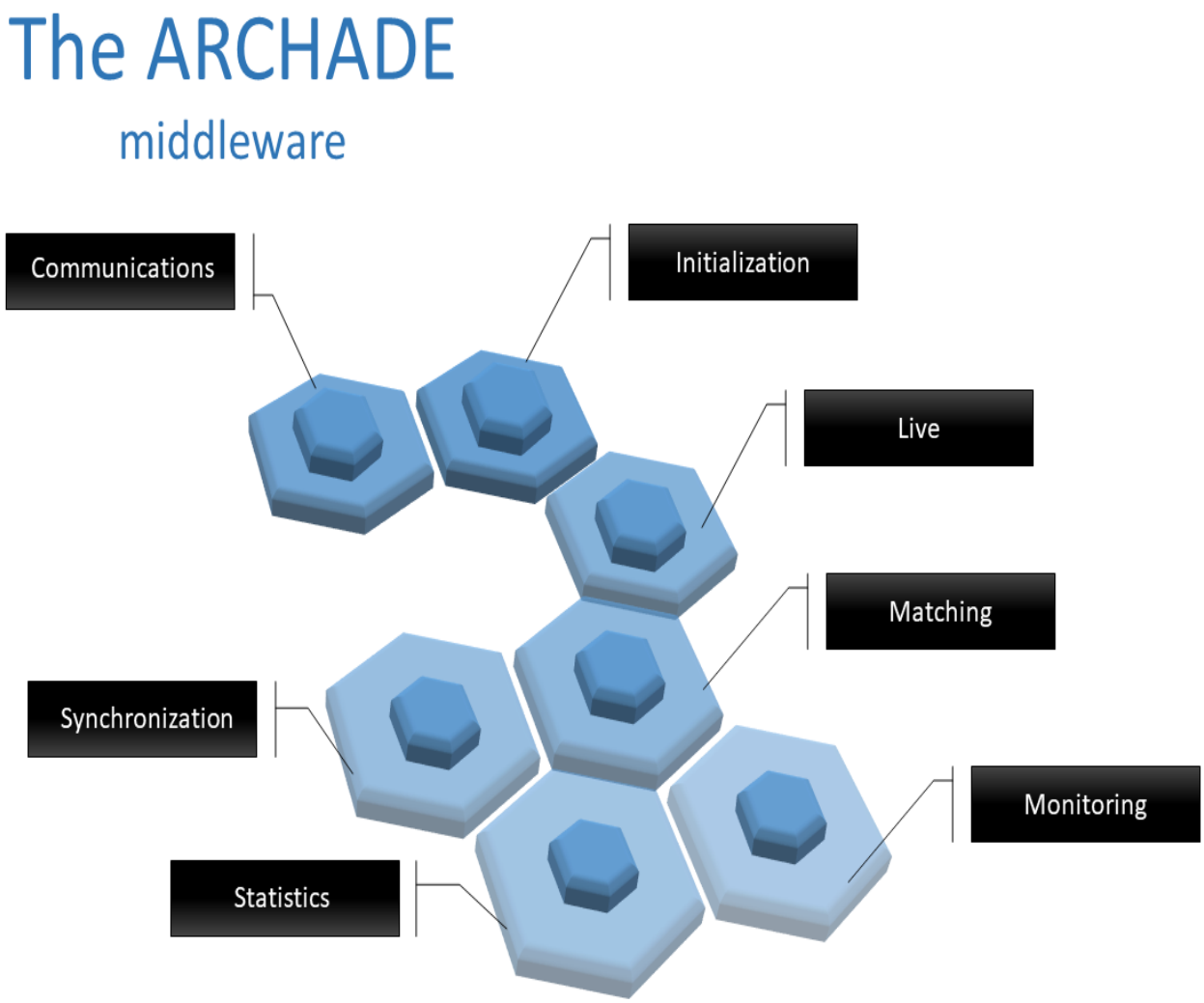


Figure IV-4: *The ARCHADE middleware. The middleware includes the services: communications, initialization, live, matching, monitoring, statistics and synchronization.*

To achieve such objectives, the middleware is currently composed of the services: *communications*, *initialization*, *live*, *matching*, *monitoring*, *statistics* and *synchronization* as portrayed by Figure IV-4 and described in Table IV-5.

In a HPC cluster of computers, the batch system acts as the middleware, providing user-transparency, resources monitoring, job scheduling, etc. In the context of ubiquitous supercomputing infrastructures, The ARCHADE sits on top of the batch system, in the applications layer, and augments the middleware capacities.

Table IV-5: *TAC middleware services*

Service	Description
Communications	The communications service implements the *TAC communications protocol*, purposely unpublished. The service consists of a set of classes, API libraries and sub services for secure message exchange between live agents (see *Live* item). The service includes two sub services, *Server* and *Hydra*. During the execution of a mission, every entity runs a *Server* service, intended to listen for orders and events. Correspondingly, the *Hydra* service can be used by the system operator, operators or live agents to command orders over a subset of the entities (see automation modes in chapter III section III.7) respecting the established hierarchy network. The hydra service establishes bidirectional communication channels with Server services running in distributed entities during the execution of a mission and it can be used to overthrown autonomous behavior. Furthermore, the service's API provides methods for events and orders exchange. Such methods are used in the task and MPI task software templates described in Table IV-3.
Initialization	The initialization service reads the system description, i.e. the framework templates (UbiSL) in order to create entities, groups, systems, etc., objects in execution time. Furthermore, it performs initial mission area distribution between mobile entities.
Live	The live service is used during the execution of a mission and its main objective is to provide *autonomy*. Each entity executes a *live service agent*, which controls the entity. The agents provide autonomous behavior by implementing methods for autopilot control (takeoff, move, return home, etc), mission execution (autonomous tasks software launching, selectively yet not necessarily interacting with the batch system), etc. Autonomous features such as moving the vehicle can be disable by setting an entity as *piloted* in the entity template. With this approach, all other autonomous behavior, e.g. mission execution, can be maintained while allowing pilot presence or writing mission software coordinating vehicle's motion such as the *followMe* software (see section IV.5).
Matching	The matching service compares the tasks' requirements in the mission template, with the available resources in the ubiquitous supercomputing infrastructure, assigning tasks to those entities satisfying the requirements. This process is automatically executed at mission initialization, based on the entities and mission templates. The service takes into consideration tasks classification, e.g. parallel, distributed, etc., in the assigning process.
Monitoring	The monitoring service checks tasks status and updates a global image of the mission. Moreover, the service monitors nodes status (RAM, CPU, etc), network status (connectivity, latency, signal strength, etc), GRC, vehicles telemetry, etc. This service can selectively interact wit the batch system but it can operate on its own as well.
Statistics	The statistics service computes output data from all the services providing a mission test case summary. Currently, it is executed at the end of a mission but it is being upgraded to provide live statistics.
Synchronization	The synchronization service provides *user-transparency* and *centralization/distribution* by synchronizing data (e.g. images, etc.), events, orders, etc., between all entities requiring it i.e. from the system controller to the entities and vice versa creating a *single-image system*. The service is a background process, synchronizing at all times, only extra data, based on a previous image. It also provides *resilience* by attempting synchronization when connectivity is available. Live agents cooperate to construct a centralized image of the entire distributed system, including all data collected during the mission. Finally, all data is synchronized in the system controller and or ground station.

The live service is in charge of order execution, i.e. reacting to orders received via the communications service. The default orders are: *connect to autopilot*, *start vehicle*, *take off*, *execute mission*,

return home, etc. While the list is not comprehensive, the live service can be easily configured to allow new orders and its associated methods can be written using the live service API. Moreover, order exchange is based on a secure method using cryptography approaches, whose details are purposely unpublished. In this way, orders will not be executed if coming from an unauthorized source nor if the source does not have sufficient hierarchy to command such order.

Furthermore, the service provides two types of agents, the *system controller agent* (highest hierarchy underneath the system operator) and the *entity agent i.e. entity controller*. The system controller agent performs tasks distribution (using the matching system), global monitoring, automatic order commanding, etc. Correspondingly, the entity agent controls its entity, e.g. order acknowledgment and execution, task software deployment, autopilot interaction including motion, in case of a mobile entity, etc. The system controller entity runs the two agents at the same time, therefore it can be used for mission tasks execution as well and moreover facilitates multiple configurations, i.e. a mobile entity as the system controller, the HPRC master node as the system controller, etc.

All the middleware services require the underlying presence of an ubiquitous supercomputing infrastructure. This way, supercomputing is pushed towards ubiquity. Given the complexity encountered when dealing with distributed software, especially those running over potentially expensive and subject to damages hardware, next section introduces the TAC's simulation platform component.

IV.4 Simulation platform

The *Simulation platform (SimPlat)* aims at facilitating transparent transition between simulated and real systems and algorithm testing as portrayed by Figure IV-5, A and B correspondingly. Regarding case A, if at least one entity template is set in simulated mode, i.e. setting the tag *real* to *false* in the entity template (Table IV-4), the SimPlat services are triggered. Currently, SimPlat supports interaction with the ArduPilot Software In The Loop (SITL) (ArduPilot, n.d.). Such interaction currently consist of:

- Automatic launching and termination of the SITL software.
- Vehicle type selection (e.g. ArduCopter, APMrover2, etc).
- Geographic area setting based on distributed area per entity from the initialization service (Table IV-5).
- Output and input sockets opening for vehicle control and monitoring.

The ArduPilot SITL, allows to test autopilot functionalities for a complete range of vehicles including multi-rotor aircraft, fixed-wing aircraft, ground vehicles, underwater vehicles, etc, without the actual autopilot hardware. Multiple ubiquitous robotics solutions such as Gazebo, X-Plane 10, etc. can be coupled with the SITL software.

TAC's SimPlat API libraries are called from objects instantiated from the framework classes, e.g. *dkrobot* (section IV.2)) in order to automatically interact with the ArduPilot SITL or via DroneKit-SITL (3DR, n.d.c). In the case of setting the tag *real* to *true* in the entity template, the SimPlat services will not be triggered. As portrayed by Figure IV-5-A, a system can be simulated having combinations of virtual machines (VM) and real machines (RM), each one installed with either Dronekit-SITL or ArduPilot SITL and in addition simulated and real entities (e.g. robots embedded with a companion computer, the autopilot hardware, etc.) providing increasing testing

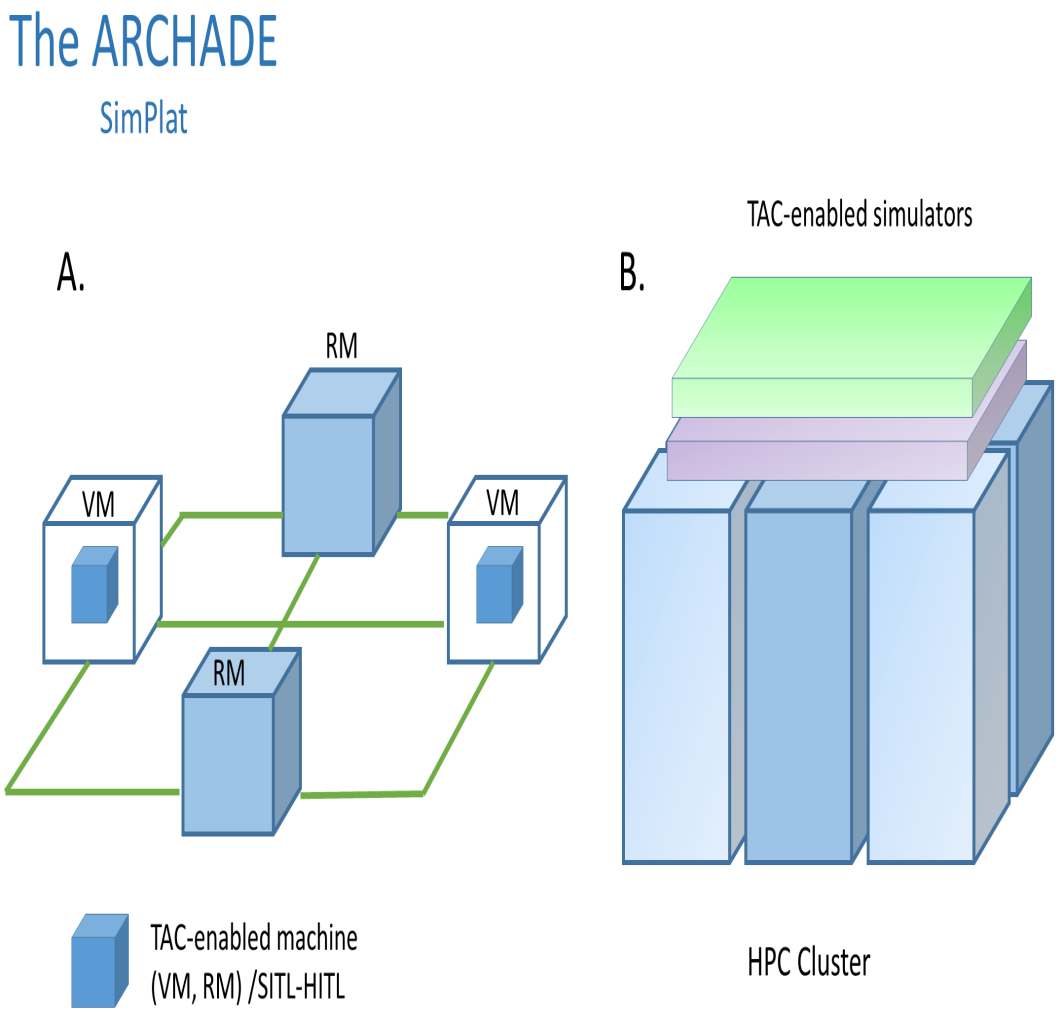


Figure IV-5: *The ARCHADE simulation platform. VM = Virtual machine, RM = Real machine*

levels. It is worth mentioning that DroneKit-SITL does not currently support ARM architectures such as the Raspberry Pi. However, ArduPilot SITL does support such architecture.

The simulation platform has been tested in common computers, virtual machines and Raspberry PIs 3 model B, see chapter V section V.4 for a complete example. Currently, the SimPlat is being coupled with an interface towards Amazon Web Services (AWS) and Docker integration, managed by the deployer service (TAC's framework, section IV.2). This is done in order to facilitate large-scale simulations composed of hundreds or thousands of entities, i.e. VMs or containers, each one installed with a SITL autopilot.

Moreover, another feature is provided by the SimPlat, a *general-purpose simulator template*, based on MPI, to be used as a mechanism to create TAC-enabled simulators (Figure IV-5-B), e.g. for algorithm testing, for example, swarming. Every simulator, based on the template, can be executed upon a HPC cluster of computers by default. In such simulators, each MPI process represents an entity, therefore two approaches are envisioned and supported:

1. Instead of using VMs or RMs, MPI processes can be mapped to the cores in a HPC cluster and be used to simulate cooperative motion algorithms, mission software, etc., without the need of independent VMs or RMs each one with a SITL autopilot.

2. Having a set of VMs or RMs, configured as a HPC cluster, each one installed with a SITL autopilot, each MPI process (one per machine) can interact with its corresponding autopilot us-

ing the SimPlat's API. This differs from the ideas behind Figure IV-5-A or TAC's middleware for real systems, given that TAC does not use MPI for services such as live, synchronization, etc.

Chapter V section V.3 will introduce a simulator for aircraft cooperative flights (*multi-system*), under approach one, which was tested with up to 256 entities at the same time, using a small HPC cluster composed of 4 nodes and 16 CPU cores. The simulator template fundamentelly differs from traditional HPC i.e. it does not use MPI to speed-up a complex computing task, but rather to set any quantity of parallel processes representing individual TAC entities. However, the template can be used for computing-efficiency as well, exploiting all MPI's features. Next section introduces the fifth and last TAC component, the *PLUS* component.

IV.5 PLUS

The ARCHADE *PLUS* consists of a *General-Purpose Computing Mission (GPCM) API* and complete software (using MPI) developed to be used by different type of missions. Currently, it includes two modules, *Motion* and *Security* as portrayed in Figure IV-6.

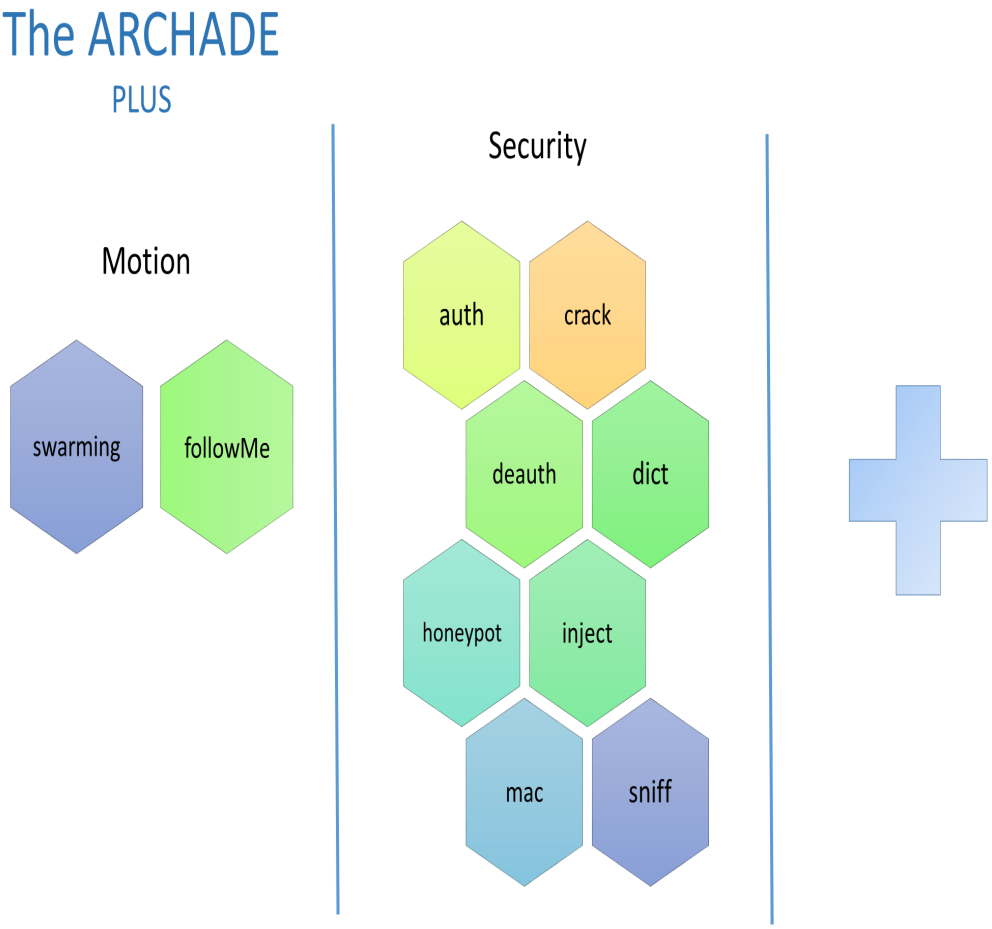


Figure IV-6: *The ARCHADE PLUS.*

- *Motion*: The motion module currently includes *swarming* software for motion coordination based on the Vicsek model (Vicsek *et al.* , 1995) and a leader/follower software, i.e. *followMe*. Details about their usage will be presented in chapter V section V.2 and section V.5 respectively. The software in this module can be used as tasks software for a mission. To do so,

entities must be set in piloted mode, i.e. out of the control of the live service in the middleware component (section IV.3)

- *Security*: The security module is in fact a set of API libraries for ethical hacking and it aims at enabling an ubiquitous robotics system to perform hacking attacks and to protect itself from the same type of threats. Details about its usage will be presented in chapter V section V.5.

TAC's PLUS shares the ideals of projects such as RoboEarth (Waibel *et al.* , 2011), by setting a set of libraries and software that can be used in all kind of applications. This component is fully dedicated to the applications layer but it is integrated with the rest of the components, providing therefore user-transparency and easing GPCM software development.

IV.6 Summary and discussion

In this chapter, The ARCHADE's scope, functionalities and components were described. The technology is composed of multiple libraries organized into five self-contained components, each with a clear purpose and specific objectives. Its development has followed the guidelines for current and future ubiquitous robotics frameworks proposed by Jiménez-González *et al.* (2013) and design principles such as simplicity, loose coupling, etc., as presented in Table IV-2.

In addition, The ARCHADE was conceived as the vehicle for the ubiquitous supercomputing philosophy (chapter III) into reality and ultimately to serve to the highest ideal of this research, to *bring supercomputing everywhere*. Therefore, its software components were designed to provide systems with all the ubiquitous supercomputing features, i.e. see *What supercomputing actually is* in the introduction chapter section I.1.2 and *the features and advantages of HPRC* in the ubiquitous supercomputing chapter section III.1.2.

Following Table IV-6 maps each of the ubiquitous supercomputing features with The ARCHADE's specific component and or functionality. Furthermore, more details will be given in the conclusions chapter VI. Consequently, next chapter will introduce a set of experiments and applications carried out under the scope of ubiquitous supercomputing and or using The ARCHADE.

Table IV-6: *The ARCHADE and the ubiquitous supercomputing features*

Feature	The ARCHADE's approach
General-purpose	The framework classes (Figure IV-3) can be used to describe different kind of entities (e.g. robots, IoT, etc), aiming to facilitate any kind of mission/application. Furthermore, the API (section IV.1), the API per component and the PLUS component (section IV.5) can be used to build mission tasks software, eased by the utilization of the task and MPI task software templates (Table IV-3).
OS based robots and standardization	The framework deployer service (Table IV-5) and the HPC-ROS package (chapter III section III.1.1) create ubiquitous supercomputing infrastructures using standard software in each of the HPC software layers. This facilitates integration with other supercomputing infrastructures. In addition, the MPI task software template and the Motion module in the PLUS component, make use of MPI, the most standard HPC technology for software parallelization. Furthermore, OS-based robots open the spectrum of possibilities for all kind of new applications.
Scalability	In the next chapter, scalability tests (e.g. sections V.3 and V.5.2.5) will be discussed to demonstrate The ARCHADE's capacity to scale with simulated and real systems.
Heterogeneity	Traditional supercomputing entities (e.g. servers, etc.), robots (UAVs, UGVs), CLD, humans, etc., evidence The ARCHADE's approach towards heterogeneity.
User-transparency	The system and entity controllers agents (live service in Table IV-5), the automation modes (chapter III section III.7) and the hierarchy ideas (chapter III section III.5) provide user-transparency.
Cooperation	The ARCHADE provides entity cooperation via task distribution with the middleware matching service (Table IV-5), the philosophy behind General-Purpose Computing Missions (chapter III section III.3), area distribution with the middleware initialization service and MPI software in the PLUS component.
Resilience and failure tolerance	While this feature is out of the scope of this Ph.D thesis, the middleware synchronization service (Table IV-5) provides resilience by constantly attempting the creation of a single-image system.
Hierarchy	See hierarchy section (chapter III section III.5) and the UbiSL hierarchy network template (Table IV-4).
Computing-efficiency	MPI usage for performance analysis will be discussed in next chapter section V.1.
Centralization and distribution	The middleware synchronization service provides centralization and distribution by guaranteeing data to be distributed to entities requiring it but maintaining a complete copy in the system controller. Furthermore, by using the live service agents, each entity has the capacity of operating autonomously (*distribution*) but under hierarchical control (*centralization*)
Multi-purpose and multi-user	By default all supercomputing infrastructures are multi-purpose and multi-user as discussed throughout this thesis. In addition, see the concept of High Availability Robotic Cluster (definition 6)
Security	Order commanding via the middleware communications service (Table IV-5) is managed using a secure approach based on cryptography keys. Moreover, the framework deployer service is being enhanced to perform firewall configuration to defend entities from hacking attempts and network attacks.

How nice will it be to have a conversation with you, a real one. I can't claim all those we had before, were not real, but I am adapted to this world, I mostly see with my eyes, mostly hear with my ears, mostly feel with my hands. It doesn't have to be a long conversation, maybe I can convince you to join me for a coffee, upon that place I used to sit back in Barcelona, or Toulouse, or Budapest, Saint Petersburg, maybe in Pasadena, or a beer in Bucaramanga, same as I do often with my brother, do you remember?

— Leonardo CF

V

Experiments and Applications

The previous chapter IV introduced The ARCHADE, the materialization of the ideas behind the ubiquitous supercomputing philosophy described in chapter III. The ARCHADE's general objective is twofold, first to provide systems with all the ubiquitous supercomputing features (see chapter IV Table IV-6) and to bring supercomputing everywhere, even in multi-robot systems. In this chapter, a set of experiments and ubiquitous supercomputing applications are introduced and discussed. All the experiments and applications were carried out using the following four ubiquitous supercomputing infrastructures:

- *HPC cluster*: HPC cluster composed of 4 nodes with a total of 16 CPU cores. Figure not portrayed.
- *RPI-HPRC cluster*: HPRC cluster composed of 4 Raspberry Pi 3 model B and a total of 16 CPU cores. (Figure V-1).
- *HPRC-HPC cluster*: HPRC cluster composed of 1 laptop computer, 3 virtual machines, 1 Raspberry Pi 3 model B and a 7-node HPC cluster. The infrastructure is composed of 34 physical CPU cores and 6 virtual CPU cores. (Figure V-2).
- *HPRC-Rovers cluster*: HPRC cluster composed of 1 laptop computer and 2 Unmanned Ground Vehicles (rovers), each one embedded with a Raspberry Pi 3 model B, with a total of 12 CPU cores. (Figure V-3).

Figure V-1: *RPI-HPRC cluster*

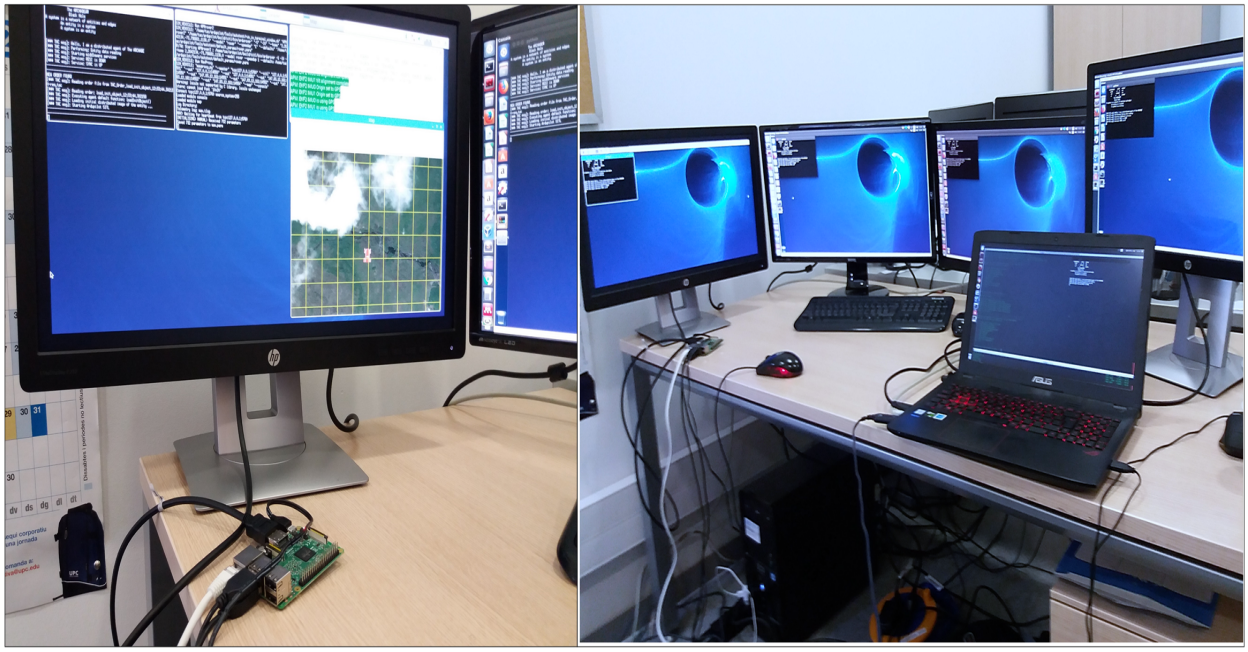


Figure V-2: *HPRC-HPC cluster. The HPC cluster is not displayed in the picture*

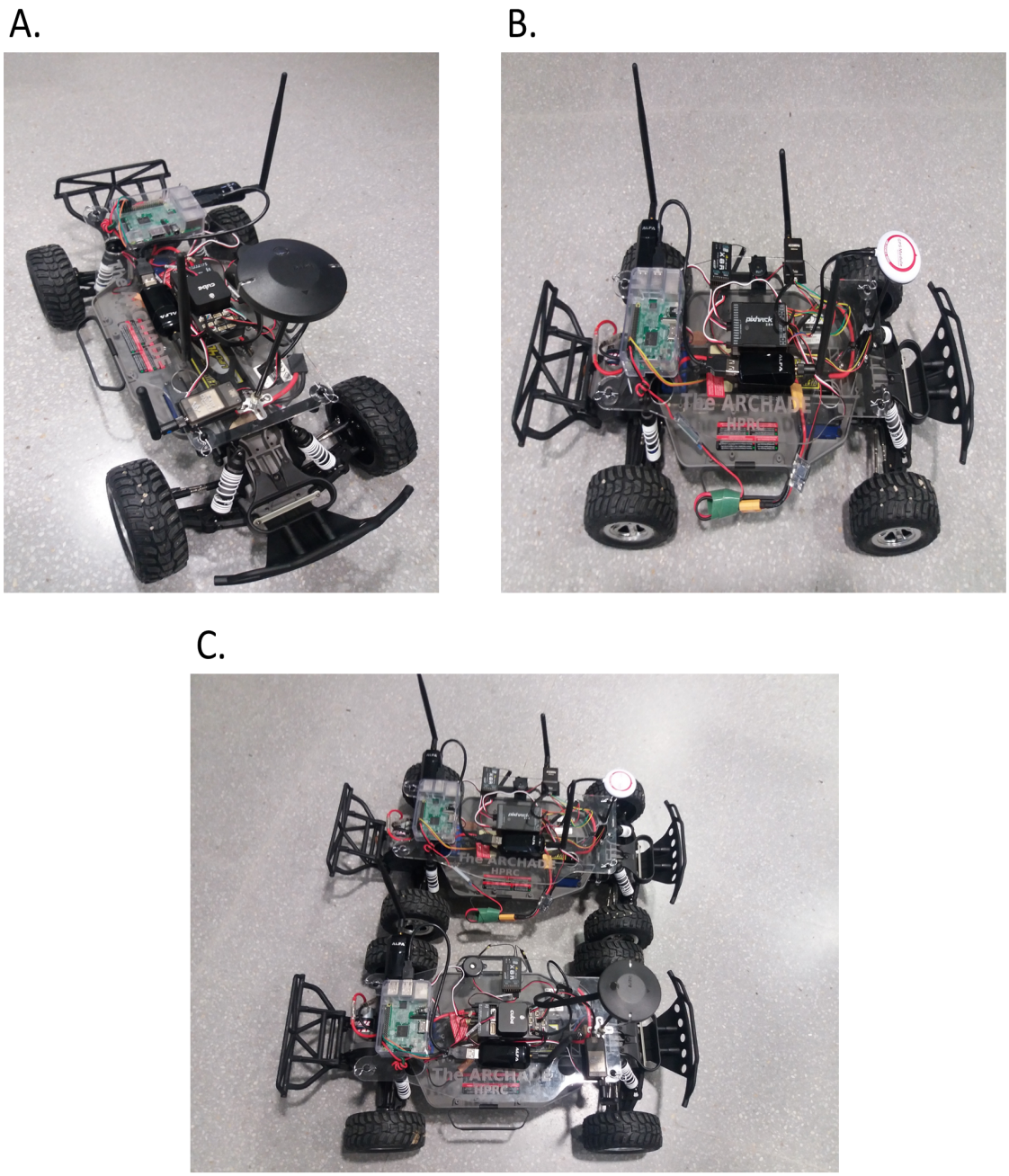


Figure V-3: *HPRC-Rovers cluster. A. Hackrover1, B. Hackrover2. C. Hackrovers. The laptop is not portrayed in here, but it can be observed in Figure V-2*

The two rovers were named as *hackrover1* and *hackrover2* to designate their capacity for ethical hacking. Figure V-4 shows the architecture of hackrover1, which uses the Pixhawk 2.1 a.k.a. *the cube* (Dronecode, n.d.b). Conversely, hackrover2, Figure V-3-B, embeds the Pixhawk 1 (ARDUPILOT, n.d.c). However, both flight controllers run ArduPilot version 3.4.2 for the missions presented in here. Moreover, the cars are Traxxas Slash 1/10 2W (Traxxas, n.d.).

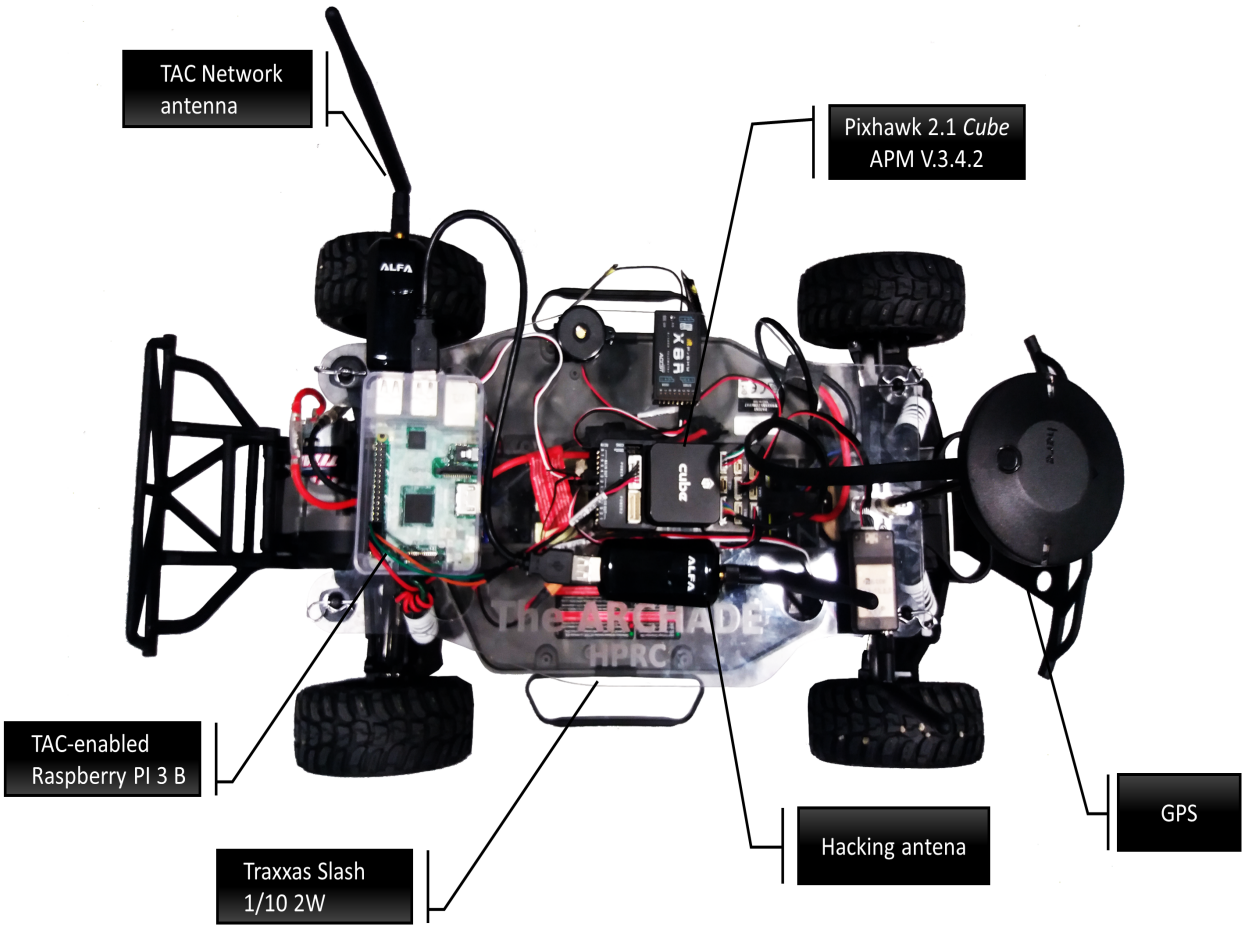


Figure V-4: *Hackrover1 architecture. A. Hackrover1, B. Hackrover2. C. Hackrovers. The laptop is not portrayed in here*

The Raspberry Pi 3 model B boards, used in the ubiquitous supercomputing infrastructures, come with a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU and 1 GB RAM. Following section discusses a set of performance, i.e. traditional HPC, indicators.

V.1 Performance

In this section, the standard official High Performance Computing test *High Performance Linpack*[1] (HPL) (Petitet *et al.* , n.d.) is performed upon two of the ubiquitous supercomputing infrastructures, the RPI-HPRC cluster (Figure V-1) and HPRC-Rovers cluster (Figure V-3).

The HPL test measures the amount of FLOPS (Floating Point Operations Per Second) that can be obtained from a computing infrastructure, when solving a linear system using MPI. From its web site (Petitet *et al.* , n.d.): *HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers*. Depending on the quantity of CPU cores in a HPC or HPRC cluster, the type of technology and status of the underlying network communications, etc., the test will output the maximum amount of FLOPS and the time required to solve the linear system.

The 4-node RPI-HPRC cluster, with a total of 16 CPU cores, consists of one master and four slave nodes, i.e. the master is configured as slave as well, deployed with the HPC-ROS package

[1]HPL is used to estimate the computing power of the TOP500 supercomputers (Dongarra *et al.* , n.d.) – https://www.top500.org/project/linpack/

(chapter III section III.1.1). Commonly, it takes a fair amount of tuning to obtain the maximum possible FLOPS available in a HPC cluster. However, such tuning is outside the scope of this Ph.D thesis, because the objective of the test, in here, is not to estimate the maximum available computing power, but rather to show the feasibility of HPC with embedded computing boards, i.e. in HPRC environments. Nevertheless, three HPL variables are important to set. Those are:

- $N = 3600$ (Linear system size i.e. matrix order)
- $P = 1$
- $Q = 16$ and
- Block sizes, $NB = \{32, 64, 128, 256\}$

P and Q specify the parallel processes mapping and $P \cdot Q$ should be equal to the maximum amount of cores in the HPC setting, in this case sixteen. The choices, for each of the variables, were based on information provided by the official HPL documentation (Dongarra, n.d.a). Figure V-5 shows a performance comparison using HPL over Ethernet versus Wi-Fi.

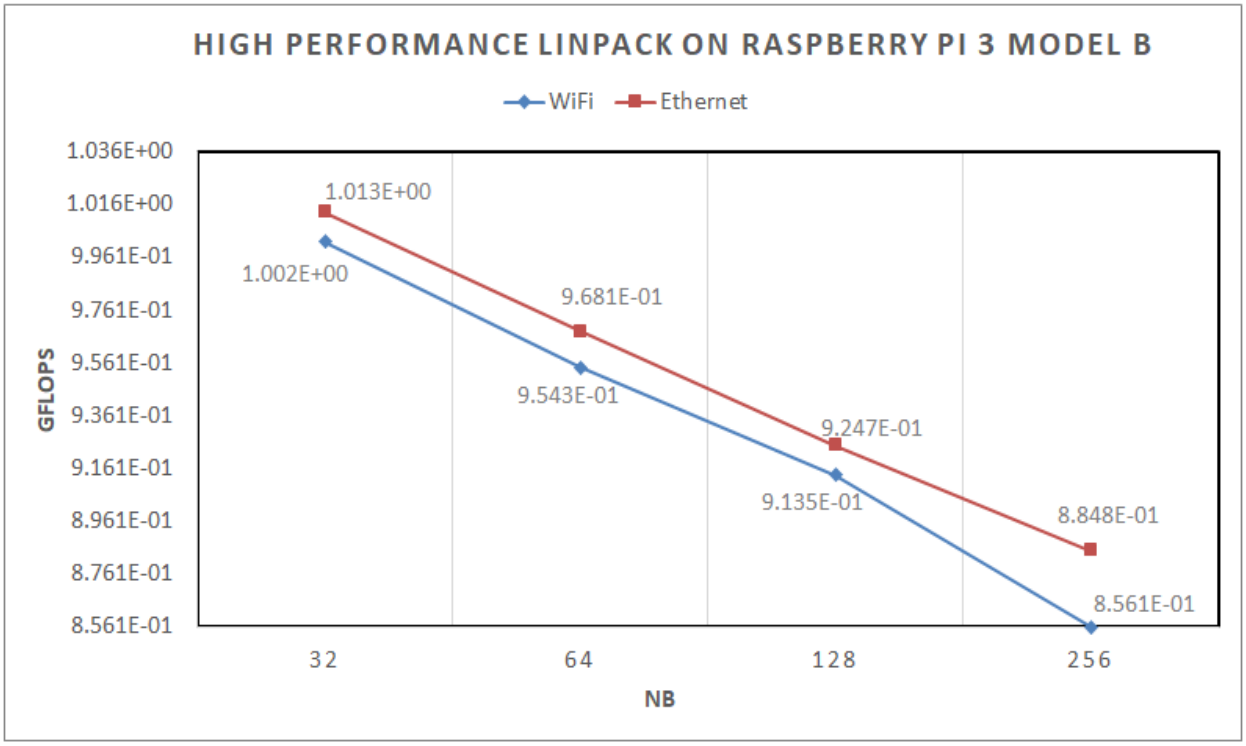


Figure V-5: *HPL using Ethernet and Wi-Fi. The test was performed over a 16-core setting made up of four Raspberry Pi 3 model B. Ethernet and Wi-Fi performance are very similar*

The max amount of 1.013 Giga FLOPS (GFLOPS), is obtained with a NB of 32 over Ethernet. As it can be observed in Figure V-5, performance is quite similar when using Ethernet and Wi-Fi. However, the Wi-Fi router, used for the test, is in proximity to the Raspberry Pi boards, within around 1 to 2 meters and it is not used for heavy internet search or communications-consuming activities.

In average, Wi-Fi has a 1.742% less performance that Ethernet, which suggests that traditional HPC is valid over wireless communication channels. In a closed space, i.e. a Wi-Fi area, a HPRC cluster could operate at seemly the same performance as a wired traditional HPC cluster. On the contrary, in open space, e.g. a Long-Term Evolution (LTE) area, HPC software such as HPL

(Multiple Instruction Multiple Data -MIMD), which relies on network message exchange, might not operate efficiently.

In order to resemble a LTE scenario, where connectivity among nodes might not be constant, a experiment consisting on node disconnection and reconnection, was performed. Purposely, one, two and three nodes were disconnected during the execution of the HPL test with a time window of 30, 60, 90 and 120 seconds. Following, the nodes are reconnected.

Node disconnection is a highly possible scenario, when using wireless communications, especially with robots such as UAVs, in BVLOS (Beyond Visual Line of Sight) missions, which can disconnect from each other when entering areas without network coverage, e.g. when using 4G or 5G. However, in line of sight it is a rare occurrence to have long-term disconnections, like those evaluated in here, e.g. 30 seconds.

Table V-1: *Performance decay with node disconnection over Wi-Fi communications*

Disconnection/Time window	30 s	60 s	90 s	120 s
One-node	63.802 %	81.377 %	81.248 %	87.914 %
Two-node	68.543 %	81.307 %	81.357 %	87.465 %
Three-node	68.29 %	81.447 %	81.707 %	86.756 %

The maximum Wi-Fi performance (1.002 GFLOPS), with a duration of 31.06 seconds, is used as the baseline value for benchmarking without node disconnection. Table V-1 shows the percentage of performance decay for the three cases (one, two and three nodes disconnection) during the selected time windows (60, 90 and 120 seconds). As it can be observed, HPC performance significantly drops with node disconnection. However, HPL does not crash. This is an advantage of the MPI implementation (OpenMPI (Gabriel *et al.* , 2004)), where messages, to be shared among processes, are queued until the communications channel is available. Such queuing however is not endless. It actually depends on the network channel being used, the MPI implementation, etc. In addition, during node disconnection, distributed processes can still perform its assigned computation. This is the reason why the performance decrease is not linear and it is similar between the studied cases. This behavior relates to HPL algorithm (Dongarra, n.d.b).

The results suggest that fully distributed software (MIMD), relying heavily on message exchange, might not achieve high performance with HPRC clusters nowadays. Limitations when using MIMD, especially with heavy (large-data) and constant message exchange, should be kept in mind when designing robotic missions using HPRC settings. Nowadays, focus should be given to SIMD especially, or lite data message exchange. In addition, the programmer should guarantee that distributed tasks are able to complete even in network failing conditions. However, using the scalability feature of HPRC, software has the potential of adapting to underlying network technologies, therefore in a nearby future, when communications have achieved expected features, we could have powerful traditional HPC carried out with HPRC clusters.

Nevertheless, for missions in line of sight, traditional HPC is still possible as demonstrated by the following experiment. The HPRC-Rovers cluster (Figure V-3), acting as a High Availability Robotic Cluster (HARC) (definition 6), was tested with HPL and Wi-Fi, while in motion, i.e. following a set of waypoints in a square area (see more information about such area in section V.5). The companion computers connect to the Wi-Fi router via a USB onboard antenna, as portrayed in Figure V-4.

HPL was executed manually via SSH (HARC), with 6 and 8 MPI processes, distributed over the two embedded Raspberry PIs, each with four CPU cores. Figure V-6 shows the three HPL test cases. The first two (6 and 8 MPI processes) executed over the 2-UGVs (HPRC-Rovers cluster) and the third case (16 MPI processes) running over the four static Raspberry Pi 3 model B (RPI-HPRC cluster)

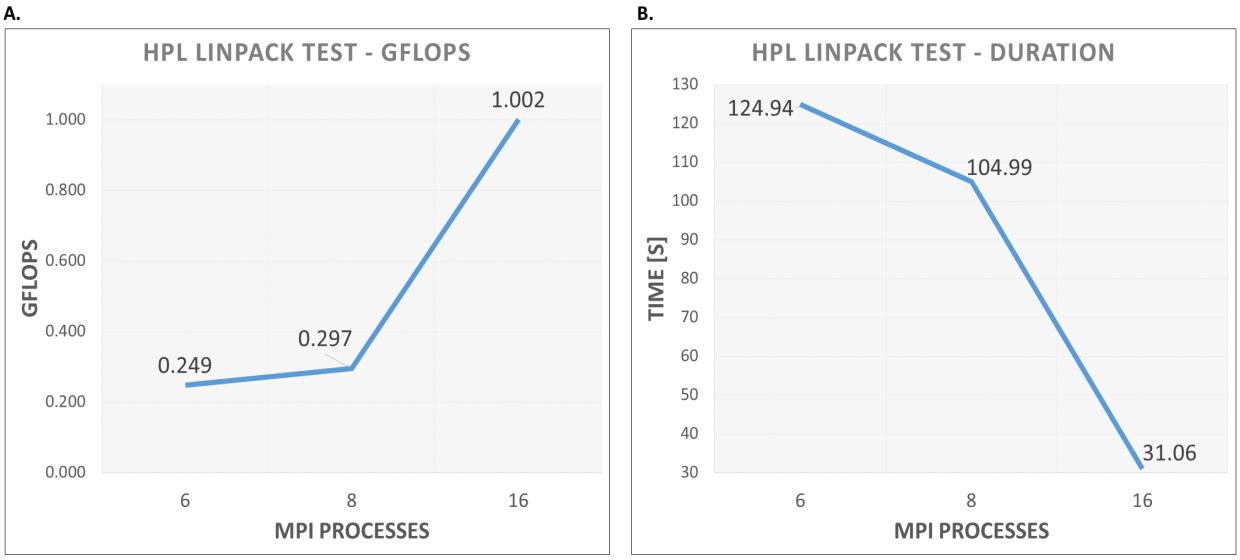


Figure V-6: *HPL over the HPRC-Rovers cluster VS RPI-HPRC cluster using Wi-Fi. The higher the computing power (FLOPS), the faster the test will finish.*

As it can be observed in Figure V-6-A, performance increases with the quantity of MPI processes. Consistently, the duration of the test decreases with larger quantities of MPI processes (Figure V-6-B). However, as displayed in Table V-2, there is a considerable higher performance increase when using 16 over 8 MPI processes ($2X$) than when using 8 over 6 MPI processes ($1.33X$).

Table V-2: *HPL test cases comparison*

Test cases comparison	performance increase (%)	Test duration decrease (%)
6 to 8 MPI Processes	19.03	15.97
8 to 16 MPI processes	237.94	70.42

These results suggest that performance is negatively impacted by mobility, as it might be intuitively assumed. However, further tests will be needed to confirm such hypothesis and quantifying it but this is out of the scope of this Ph.D thesis. Nevertheless, there is an observable performance growth when increasing the quantity of parallel processes and consequently a computing time reduction, i.e. HPC computing efficiency, even when using mobile robots (19.03%). This demonstrates HPRC support of traditional HPC. Furthermore, scalability is demonstrated as well, i.e. the same software (HPL), can be used with increasing quantities of nodes, distributed across ubiquitous supercomputing entities.

While HPRC and ubiquitous supercomputing's main objective is not to offer high computing-efficiency, they do have the potential of providing it. Regarding BVLOS missions, many applications might consist on splitting a given area, where each entity will perform a set of tasks. Since each entity can focus on its corresponding area, message exchange is minimum, contrary to fully MIMD software, e.g HPL. This is an approach for the application of ubiquitous supercomputing,

in conditions where disconnections might occur. Moreover, next sections V.2 and V.3 show how MIMD can be used for non-computationally expensive tasks, whilst still providing scalability.

V.2 Swarming

Previous attempts in robotics research have used HPC as a speeding up tool for complex computing tasks such as vision-based navigation, etc. However, as far as it was found in the state of the art, HPC software has never been used to actually control the motion of a multi-robot system, which is not inherently a task requiring traditional HPC.

Given the mobile nature of robotics entities, in this section, a software called *parallelMotion*, part of the *TAC PLUS Motion* module (chapter IV section IV.5), able to perform the coordinated motion of any number of robots using two approaches / models: *SIMD* and *MIMD* is presented. In the first approach (SIMD), which stands for Single Instruction Multiple Data, there is no communication between the moving robots, in this case UAVs. In the second approach, the UAVs constantly exchange messages in order to fly resembling a birds' swarm according to a simplification of the renowned Vicsek model (Vicsek *et al.* , 1995).

For both approaches, the RPI-HPRC cluster (Figure V-1) is used, coupled by a laptop computer acting as a master node. In each of the Raspberry Pi, the ArduPilot SITL software (ArduPilot, n.d.) and DroneKit (3DR, n.d.b) are installed. This mission is carried out in simulated mode given its complexity and current Spanish legislation where it is necessary to have one pilot per UAV.

DroneKit consists of a set of python functions to interact via MAVLink (Micro Air Vehicle Communication Protocol) (MAVLINK, n.d.) with an automatic pilot. It supports different types of vehicles such as copters, rovers and planes and it can be installed in different companion computers (3DR, n.d.a), including the Raspberry Pi. Furthermore, DroneKit allows interaction also with simulated autopilots such as the ArduPilot SITL.

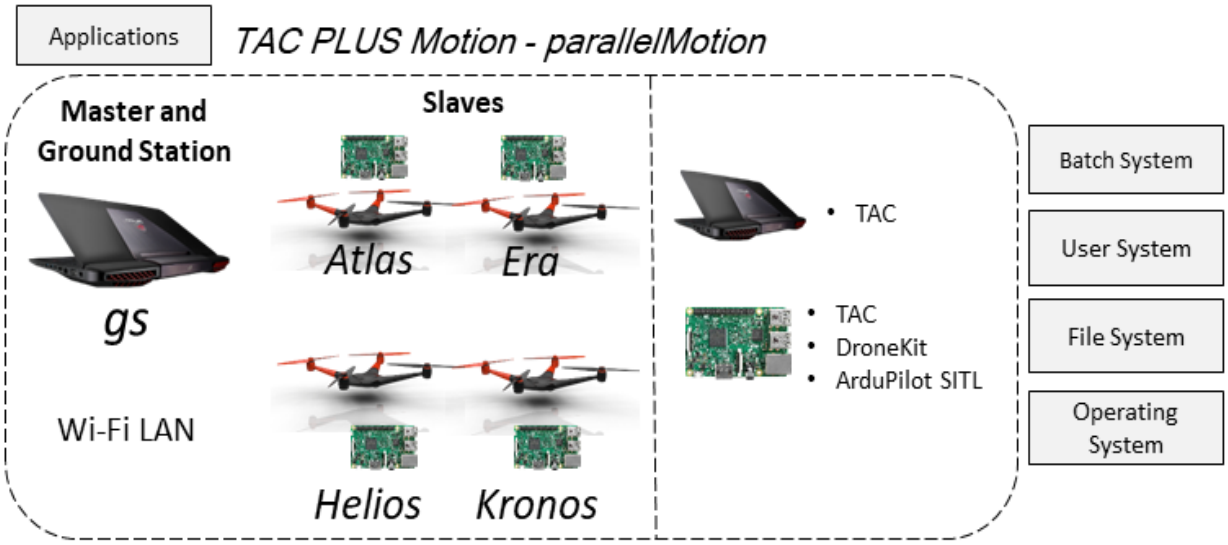


Figure V-7: *Parallel UAV motion software Architecture. TAC PLUS Motion software, such as parallelMotion belongs to the applications layer of the ubiquitous supercomputing infrastructure. The setting also implements the remaining HPC software layers*

Figure V-7 shows the software architecture for the parallel multi-robot motion software (*parallelMotion*), where the laptop computer acts as ground station/master node and it is installed with The ARCHADE (TAC). Correspondingly, the slave nodes (Raspberry PIs) are installed with

TAC, DroneKit and the ArduPilot SITL. The five nodes code names are *gs* (laptop) and *Atlas*, *Era*, *Helios* and *Kronos*.

TAC PLUS Motion uses mpi4py (Dalcin, n.d.), a Python library for the development of parallel MPI software. The library requires an existing installation of an MPI implementation, in this case Open MPI. While traditionally, MPI software is written mostly with C or Fortran, the decision of using Python, lies with the fact that Python is quickly becoming very strong in the world of embedded computing (Radcliffe, n.d.).

MPI software is executed by a set of parallel processes, uniquely identified by a rank ($0, 1...R$ having $R + 1$ parallel processes). Depending of the particular rank that a process has, it will be its purpose. For example, a process with rank 0 might execute different computing functions that another process having rank 1. The processes are distributed to one or multiple computers, and are managed and orchestrated by the MPI implementation (Open MPI), which provides features such as scalability, resilience, etc. MPI provides scalability by allowing the user to request any amount of processes, subjected to a set of rules:

1. Given a set of P processes and a set of C computing cores (distributed in one or several computers), Open MPI splits the processes among the computing cores trying to match one process per core. If $P < C$, some of the C cores will not be assigned a process. Conversely if $P > C$, some cores will be assigned more than one process. In this case, the actual amount of processes that are running in parallel, depends on the internal operating system scheduling mechanism, but normally there will not be P processes running in parallel.
2. If the ratio between P and C is larger than one ($P > C$), there is a maximum amount of processes that the MPI implementation (e.g. Open MPI) can handle.
3. It is generally recommended to have a ratio between P and C of one ($P = C$). This way, there are in fact P parallel processes.

Furthermore, the MPI middleware provides resilience by queuing messages to be exchanged between processes, until the intended recipient is available as demonstrated by the node disconnection experiments showed in section V.1. Finally, MPI handles the communications between parallel processes but it is possible for the user to request a mapping between the processes and the nodes' cores by using a *rank file*. For example, rank 0 can be bound to a specific core in a specific node, etc.

Figure V-8 shows parallelMotion software algorithm. Five processes, each one representing an entity, with one node, in the ubiquitous supercomputing system, are launched using a specific mapping. This mapping is carried out with a rank file (*rankfile*) which requests the execution of one MPI process in each of the nodes. The rank file also guarantees that the process with rank 0 is executed in the laptop computer (master / ground station). The command to launch the software is:

mpirun -np 5 *–rankfile* ***rankfile*** *python* ***parallelMotion***

Where the rank file (*rankfile*) includes the following lines:

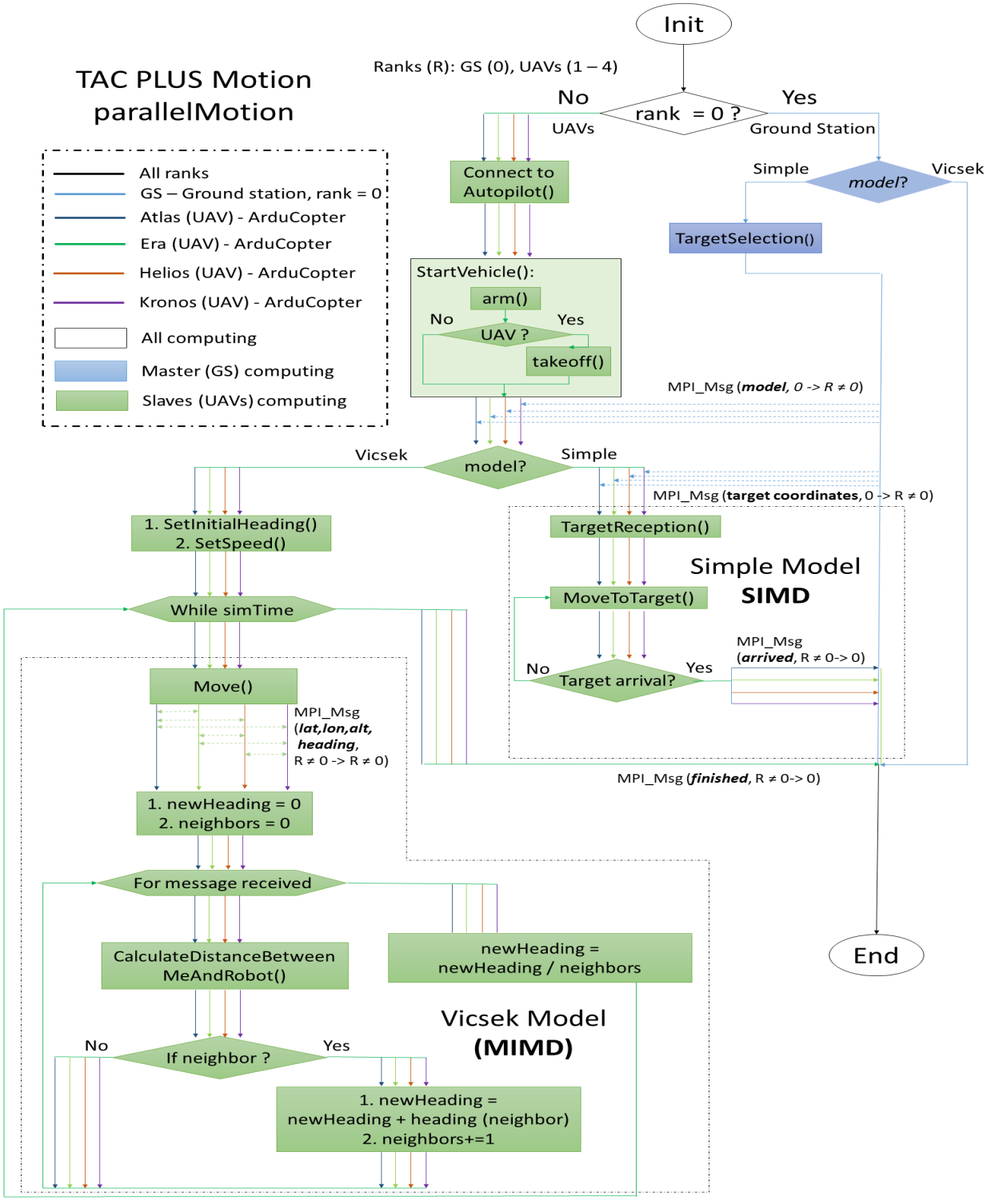
rank 0=gs slot=0

rank 1=atlas slot=0

rank 2=era slot=0

rank 3=helios slot=0

rank 4=kronos slot=0



mpirun –np 5 –rankfile rankfile python parallelMotion

Figure V-8: *parallelMotion algorithm based on MPI. The software provides fully autonomous motion for the multi-robot setting with two modes of operation, SIMD and MIMD*

Since, each of the Raspberry Pi comes with four computing cores, only one is used for the motion of the robot while the remaining three can be used for other tasks, e.g. image processing, etc. In addition, using the batch system, each core can be securely used for the requested purpose, without being shared by other processes. The pilot user can start the autonomous motion of the entire multi-robot system simply by executing the previous *mpirun* command, forgetting about the underlying complexity, i.e. user-transparency. Using MPI software is advantageous from a software development point of view as well, because a single software, executed by different parallel processes, can be written in a single module, rather than have different modules for each of the distributed nodes

In this experiment, the mission consists of one task and it overwrites the live service motion's functionalities (live agents). To do so, the entities must be set as *piloted* in the corresponding entity template in order to disable the middleware's control over motion. Therefore, this flexibility allows the user to set its own motion algorithms, in this case parallelMotion from the TAC PLUS motion module. The piloted mode can be also set to allow a real pilot controlling the vehicle.

The software is launched from the master / ground station entity but each process executes its corresponding functions according to its specific rank as it can be observed in Figure V-8 i.e. the ground station (process with rank 0) functions and the functions of the ranks 1 to 4 (UAVs). This can be observed in the first conditional (triangular shape rank = 0).

The explanation is as follows, all processes start the execution of the software and once they have reached the first conditional, the ground station process will proceed with the right part of the algorithm, while the remaining ranks (UAVs) will proceed with the left part. Following, depending on the selected model, i.e. *simple* (SIMD) and *Vicsek* (MIMD), each of the process will continue with its specific part. The ground station process is the one in charge of sending a message specifying the model to follow (*model MPI_Msg instruction* in Figure V-8).

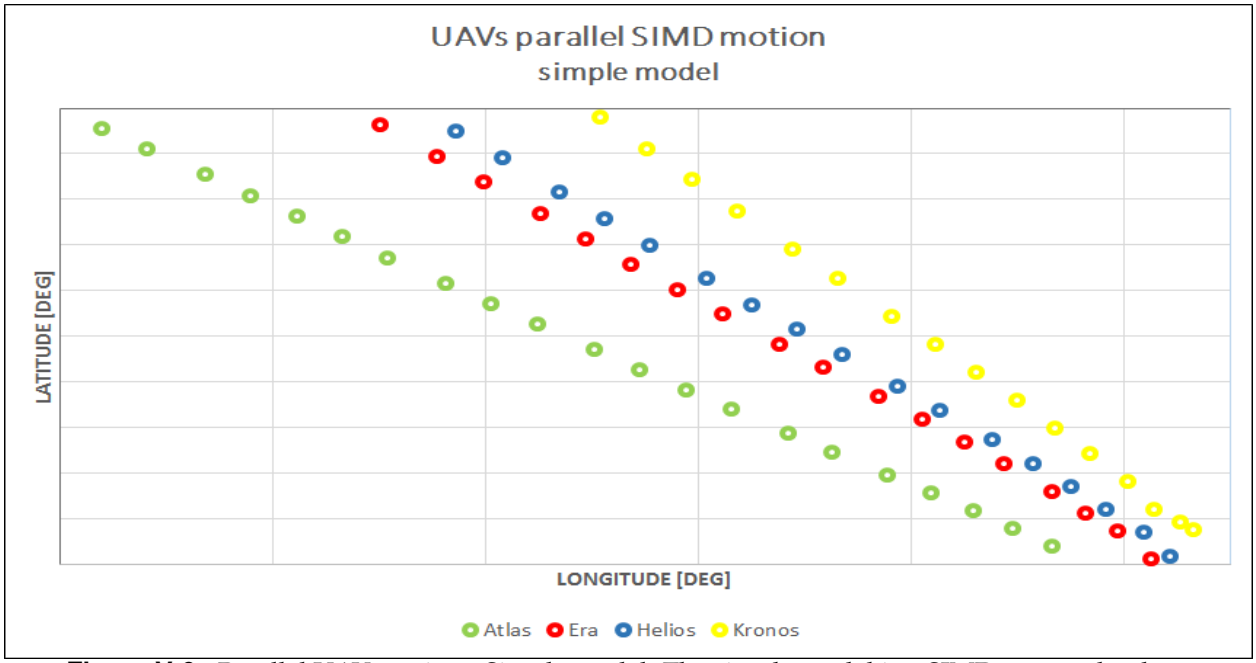


Figure V-9: *Parallel UAV motion - Simple model. The simple model is a SIMD approach where there is no message exchange between the parallel processes representing the UAVs while moving*

In the first approach, depicted as simple model, the ground station requests the UAVs to fly to a specific target. This request is done via the *target coordinates MPI_Msg instruction*. The simple model is a SIMD approach because the UAVs are performing the same functions (flying

to the specific target) but over different locations (data), i.e. each UAV fly from a different home location. Furthermore, the UAVs do not exchange messages. This means that the UAVs are fully independent of each other. Figure V-9 shows the motion of the four UAVs using the simple model.

For certain type of missions, the fact that the UAVs are independent from each other (SIMD simple model), results advantageous. Consider a mission in which a set of UAVs are used to monitor a hostile area (target). If some of the UAVs were to be compromised, the remaining UAVs, by being independent, could continue with the mission. Moreover, communications cost is minimum with this approach, given that the only exchanged messages are those related with an initial coordination, for example target coordinates, sent from the ground station to the UAVs but no inter-robot message is necessary. For the second approach, the Vicsek model (Vicsek *et al.* , 1995) is based on two ideas:

1. Each entity (e.g. bird, UAV) flies with constant speed and direction (heading) equal to the average of the directions of its neighbors. An entity i is neighbor of entity j if at time t, i and j are at a distant less or equal than a selected radius.

2. The noise in the system. Two types of noise are considered: *extrinsic* and *intrinsic*. Extrinsic noise refers to the one caused by the environment. In biological systems, it could relate to the entities (e.g. birds) not able to see properly if another entity is a neighbor. In artificial systems, extrinsic noise might relate to GPS precision issues. Conversely, intrinsic noise refers to the entities' decision to move in certain direction even if they fully understand the direction of its neighbors. In artificial systems, this could relate to computing errors at execution time.

For the purpose of this experiment, parallelMotion does not implement the noise feature of the Vicsek model, because the parallel motion software is used to show the ideas behind HPRC. Therefore, parallelMotion would require further development if it is to be used on real flights. Figure 10 shows the motion of the four UAVs using the Vicsek model (MIMD).

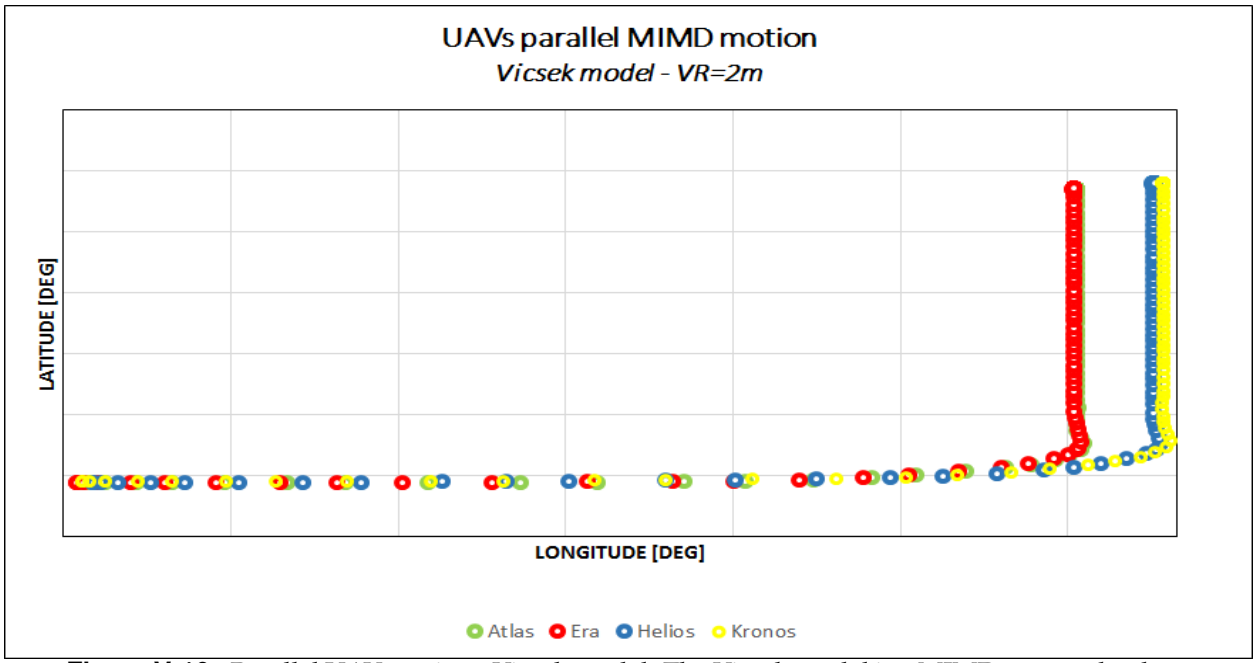


Figure V-10: *Parallel UAV motion - Vicsek model. The Vicsek model is a MIMD approach where there is constant message exchange between the parallel processes representing the UAVs while moving*

As it can be observed in Figure V-10, the UAVs, moving from different home positions, approach each other until they resemble a birds' flock. The Vicsek model, provided by parallelMotion is based on a MIMD approach, where not all the UAVs perform the same actions, given than during execution time, some UAV might have neighbors or not. If an UAV were to have neighbors, it will change its heading according to the Vicsek model feature one. Furthermore, with this approach there is constant messages exchange (*lat, lon, alt, heading MPI_Msg instruction* in Figure V-8). These messages inform the UAVs with the location and heading of other UAVs, in order to calculate the quantity of neighbors each rank has. The overlapping between UAVs observed in Figure V-10 is because parallelMotion does not implement a collision avoidance mechanism. However, this basic algorithm depicts the powerful environment provided by parallel software.

The use of MIMD software is subjected to more possible errors than SIMD, especially when facing wireless communications. Nevertheless, the scalability feature provided by MPI could lead in a no-distant future to have thousands of UAVs flying very similar to a spectacular birds flock.

The maximum amount of parallel processes that can be handled by the MPI implementation depends on the quantity of computing cores, the quality of the subjacent communications technology (e.g. LTE, Wi-Fi, etc.) and the quantity, size and frequency of exchanged messages. If these conditions are met and optimized, it is possible to build very interesting multi-robot systems carrying out all types of missions.

MPI is a worldwide renowned HPC standard technology maintained by a large community of developers. Newer versions are constantly released and its use is well documented in several fields of science. Moreover, MPI allows using the same software with any number of processes, i.e. moving robots, by simply modifying the *mpirun* command and the *rankfile*, providing therefore *scalability*.

Moreover, parallelMotion behaves in a centralized / distributed approach. The master process (ground station, rank 0) acts as a single interface for a pilot to control the multi-robot system (*centralization*), while each UAV (ranks different from zero) have sufficient autonomy to keep moving or performing its mission's tasks (*distribution*). Furthermore, since the software is launched from the master node, the user does not need to be aware of the subjacent complexity of the HPRC system (*user-transparency*). Finally, parallelMotion by means of using TAC's framework classes and DroneKit can be used to control the motion of different type of vehicles, e.g. copters, planes and rovers. Even some of the entities could be UAVs, others UGVs, etc., i.e. *heterogeneity*.

The parallelMotion software is fully autonomous, i.e. the user (pilot) only needs to launch the software using the *mpirun* command and afterwards the vehicles will move autonomously. However, future work will include a hybrid controlled / autonomous scheme allowing the pilot to be involved during autonomous motion if desired.

The motion of a multi-robot system is not a computationally expensive task that would require HPC as a speeding up tool, except in vision-based motion. However, as proposed by HPRC, a multi-robot system motion can be carried out with traditional HPC technologies, such as MPI, providing therefore scalability, centralization / distribution and user-transparency. However, in order to evaluate scalability, with hundreds of entities, the following section deepens upon the Vicsek model by using the SimPlat.

V.3 HPRC cluster of aircraft

Ubiquitous supercomputing can target any kind of entity, that can embed a computing board, i.e. any type of vehicle, cars, aircraft, etc. In this section, the concept of *HPRC cluster of aircraft* is introduced.

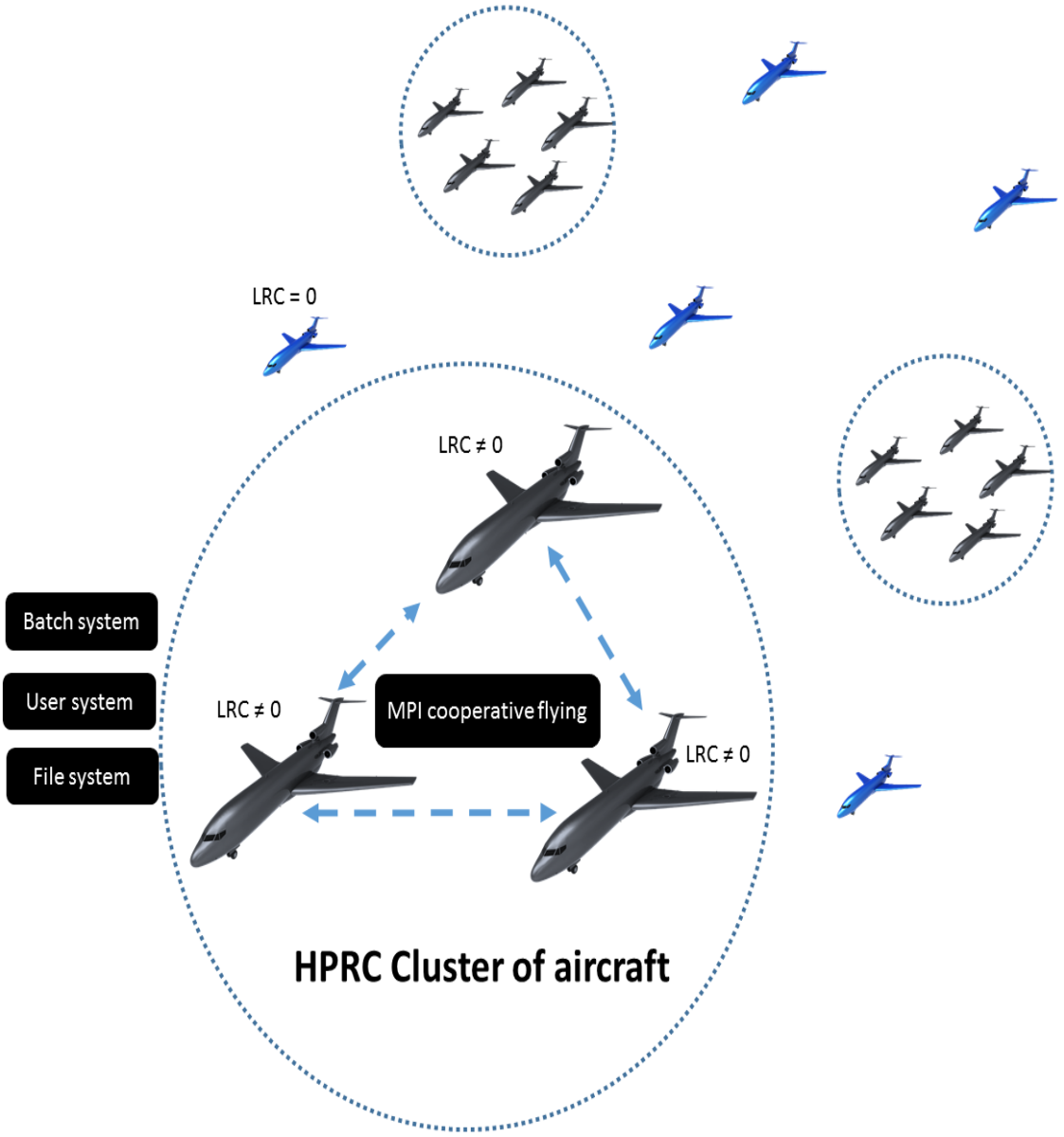


Figure V-11: *HPRC cluster of aircraft. Joined aircraft will dynamically implement the HPRC software layers (file system, user system, etc). Furthermore, in the applications layer, a MPI software for cooperative flying is executed*

HPRC cluster of aircraft

Definition 18. *A HPRC cluster of aircraft consist of a set of aircraft, which use embedded computing boards and wireless communications technologies, in order to integrate into a single cohesive unit, capable of sharing data, local users and computing resources.*

Remark. *A HPRC cluster of aircraft is a dynamic scalable infrastructure that can accommodate any quantity and type of aircraft that are found within a proximity radius.*

The applications of a HPRC cluster of aircraft are many. For example, Air Traffic Flow and Capacity Management (ATFCM), i.e. in order to decrease air traffic controllers workload, a cooperative set of aircraft could keep themselves safely separated. Moreover, a set of aircraft could cooperatively fly to safe fuel. Several examples are evidence of the advantages of cooperating while moving. In nature, flocks of birds, school of fish, countless examples of insects, etc., show that

swarming behavior is an outstanding and efficient phenomenon. Even at human level, cyclists cooperate while on route to save energy and at military level, aircraft perform flying formation with the objective of saving fuel.

In Durango *et al.* (2016), it was found that around 3.5 % savings in fuel can be achieved using flying formation schemes in civilian air traffic. Moreover in Bower *et al.* (2009) the findings were even better at around 16.5%. In addition, cooperative flights have the potential of being more efficient, i.e. saving fuel, correspondingly decreasing CO_2 emissions and furthermore creating a stable hierarchical network of aircraft. HPRC can be used as the vehicle to bring cooperative flights into air traffic schemes, by providing strategies for dynamic aircraft joining into the novel concept of HPRC cluster of aircraft. These ideas apply to Unmanned Aerial Vehicles (UAVs) as well. While formation flights differ from the Vicsek model, in this section, it is introduced a SimPlat-based simulator called *hprccoopflying*, implementing the aforementioned model for large-scale simulations.

At a specific time, an aircraft can be clustered or not. If clustered, aircraft within a HPRC cluster would execute MPI cooperative flying software, which relies on exchanging telemetry messages and modifies aircraft original trajectories. Non-clustered aircraft will continue with its original trajectory.

The procedure for aircraft joining and separation, composed of a set of steps to be performed by each aircraft, at each moment while in cruise phase, is as follows:

1. Aircraft, within a specific proximity radius and in cruise phase, are considered as potential candidates to join.
2. If the next position of its original trajectory corresponds to its Top Of Descent[1] (TOD), the aircraft will not join or it will separate from the HPRC cluster. Each aircraft will inform its potential candidates of its decision using an MPI message. The TOD in the original trajectory is kept for practicality reasons but clustering could occur in descent phase as well. However, this approach is not considered in this Ph.D thesis.
3. Aircraft performing TOD in their next position of their original trajectory will be removed from the list of potential candidates to join.
4. The cluster size and remaining entities will be updated.
5. If the cluster size is one, the aircraft will continue performing its original trajectory
6. If the cluster size is at least two, aircraft will execute the Vicsek software.
7. *Clustering approaching phase*: Once the aircraft are at a selected distance from each other, they will estimate if there will be fuel savings (*fuel test*) while staying in the cluster. Otherwise, they will separate. Each aircraft will inform the rest of the members of the cluster its decision to continue or not via an MPI message.
8. The cluster members and size will be updated.

Clustered aircraft hold an LRC different than zero, given that all the nodes in their cluster are reachable via MPI messages. Correspondingly, non-clustered aircraft hold a LRC equal to zero as portrayed by Figure V-11

An aircraft can be part of only one cluster at the same time but during its trajectory, it can be part of different clusters. In fact, given aircraft separation because of TOD and fuel tests, the size of a cluster may vary. In addition, an aircraft can join an already existent cluster. The cluster dynamic updating is possible given HPRC scalability feature. Furthermore, several HPRC clusters

[1]Planned coordinates of the location representing the transition from the cruise phase of a flight to the descent phase

of aircraft can occur at the same time in a scenario composed of hundreds or thousands of flying aircraft, i.e. this is an example of a multi-system, while multiple HPRC clusters can dynamically accept other aircraft via interfaces.

A fuel test consists of checking which option is more profitable regarding fuel, either to remained clustered or to return to the original trajectory. Figure V-12 displays the ideas of the fuel test.

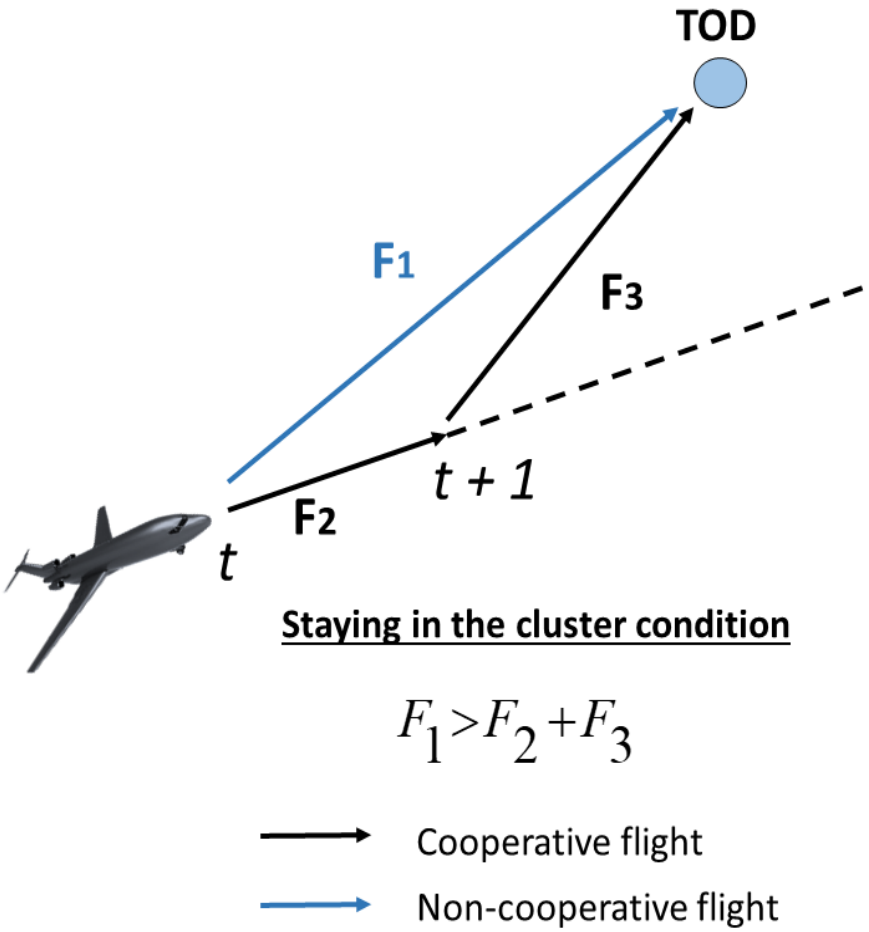


Figure V-12: *HPRC Cluster fuel test. Aircraft perform the fuel test at each step of its trajectory while clustered.*

Fuel (F) types 1, 2, and 3 are defined accordingly to the following equation, D being the traversed distance, in Nautical Miles (NM), during the corresponding segment.

$$F_i = K_c \cdot D \cdot K_f, \; i \in \{1, 2, 3\} \quad [\text{Kg}] \tag{V.1}$$

where,

$$K_c = \frac{F_c}{D_c} \quad [\text{Kg/NM}] \tag{V.2}$$

$$K_f = 1 \; , \quad \text{for non-cooperative flying} \tag{V.3}$$

$$K_f < 1 \; , \quad \text{for cooperative flying} \tag{V.4}$$

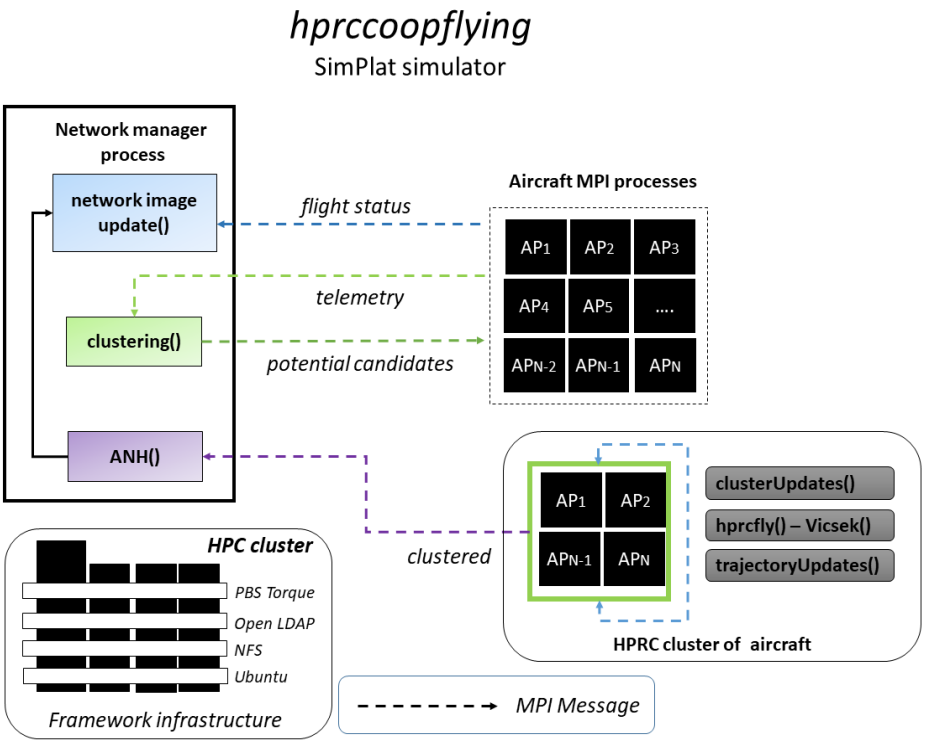


Figure V-13: *SimPlat hprccoopflying general workflow*

In equation V.1, it is assumed that fuel is a function of the traversed distance D in a particular segment of the trajectory. The constant K_c is calculated as the total fuel consumed during cruise phase (F_c) divided by the total distance traverse while on it (D_c), in the original trajectory. This constant is applied to segments both in cooperative and non-cooperative flying. Correspondingly, K_f is a dimensionless quantity used for the assumption that cooperative flights are more efficient in terms of fuel that non-cooperative flights (flying solo) as described in the literature.

If cooperative flights were to be implemented in air traffic operations, a hierarchical aircraft network will emerge, where aircraft within HPRC clusters will hold higher hierarchies (LRCs) than those flying solo. Moreover, LRCs and GRC will vary during the time window of air traffic operations, given the HPRC clusters' dynamic nature, where clusters size may vary and new clusters could appear or disappear. Therefore, a novel performance indicator, depicted as *Aircraft Network Hierarchy (ANH)*, based on the System Hierarchy (SH) indicator (equation III.8) is defined as:

$$ANH = SH \cdot \frac{Q}{|E|} \tag{V.5}$$

where Q = Quantity of clustered aircraft and $|E|$, the total aircraft in an air traffic scenario. With the objective of evaluating the ideas of *HPRC cooperative flying* in realistic air traffic scenarios composed of hundreds or thousands of aircraft, *hprccoopflying* is introduced. The software is a tactical quasi-real time simulation environment, in which each aircraft is represented by an independent parallel MPI process that follows an inputted original trajectory based on an extended version of a So6 file (Eurocontrol, n.d.), which includes fuel consumption per segment and it is sampled with a 10-second time step. Figure V-13 presents the platform general workflow. An extra MPI process represents the network manager, which at each step of the simulation, computes and inform each aircraft, in cruise phase, with its potential clustering candidates (clustering green box in Figure V-13), i.e. those aircraft found within a proximity radius or an empty list in the opposite case. Upon received clustering candidates, each aircraft decides to join or not its potential cluster.

The simulator, based on the SimPlat (see Figure IV-5 - B) implements the procedure for aircraft joining and separation previously described and provides the following features:

- It can be used to simulate any quantity of flights by mapping MPI parallel processes to computing cores distributed upon a traditional HPC cluster using Ubuntu, NFS, Open LDAP and PBS torque for the HPC software layers from bottom to top.
- Simulated HPRC clusters are created by means of MPI message exchange between clustered aircraft (clustered MPI processes).
- Its component-based design allows for the introduction of new algorithms for cooperative flying, e.g. formation flying or clustering schemes. This is powered by the SimPlat.
- It is highly configurable and provides libraries for benchmark automation, including filtering aircraft by departure and destination airports.
- The simulator computes several indicators related to existing clusters, fuel consumption, Global Reaching Centrality (GRC), ANH, computing time and other computing and communications performance indicators.

V.3.1 Benchmark setup

The simulator was used to perform a benchmark described in the following. The benchmark, composed of 20 test cases, consists of aircraft flying to the *Paris Charles de Gaulle Airport (LFPG)* using free route (FR) and flight levels (FL) Concept of Operations (CONOPS). Original trajectories were calculated and optimized by using an in-house software solution (Dalmau & Prats, 2015) over real air traffic from the 28 of July of 2016 on Functional Airspace Block Europe Central (FABEC) airspace. Table V-3 describes the benchmark setup. The decision of using the value of 0.9 for K_c, when flying cooperatively is based on the average of the findings in Durango *et al.* (2016), which suggested 3.5 % fuel savings and the findings in Bower *et al.* (2009) which suggested 16.5 % fuel savings. While the two mentioned references focused on flying formation rather than on the Vicsek model, the objective of this experiment is not to find the optimal configurations necessary to provide fuel-efficient cooperative flights but rather to present results of the simulator. Nevertheless, hprccoopflying is highly configurable and can be used to study all kind of scenarios in order to find such optimal configurations. Moreover, the test cases are described in Table V-4.

Table V-3: *SimPlat hprccoopflying benchmark setup*

Benchmark setup	
Testcases	20
Original trajectories CONOPS	FR + FL
Origin airport	Indifferent
Destination airport	LFPG
Difference in tracks for joining [deg]	15
Cooperative flying model	Vicsek
K_f for non-cooperative flights	1
K_f for cooperative flights	0.9
HPC infrastructure	4 nodes / 16 CPU cores

Table V-4: *SimPlat hprccoopflying test cases. PR = Proximity radius, NM = Nautical Miles*

Test cases					
ID	*PR [NM]*	*Aircraft*	*ID*	*PR [NM]*	*Aircraft*
1	5	16	11	15	16
2		32	12		32
3		64	13		64
4		128	14		128
5		256	15		256
6	10	16	16	20	16
7		32	17		32
8		64	18		64
9		128	19		128
10		256	20		256

Ideally, it will be preferable to assign one MPI parallel process, representing one aircraft to one core. However, for scenarios composed of hundreds or thousands of aircraft, it results difficult to do so unless having access to a large HPC infrastructure. Nevertheless, hprccoopflying by using MPI, can adapt to its underlying computing resources, up to some extent i.e. in the available 16 CPU cores, each core could be assigned with maximum 16 - 17[1] MPI processes (256 aircraft test cases). This is possible given that the simulator is very lite, but for larger scenarios, more computing resources will be necessary.

V.3.2 Benchmark results

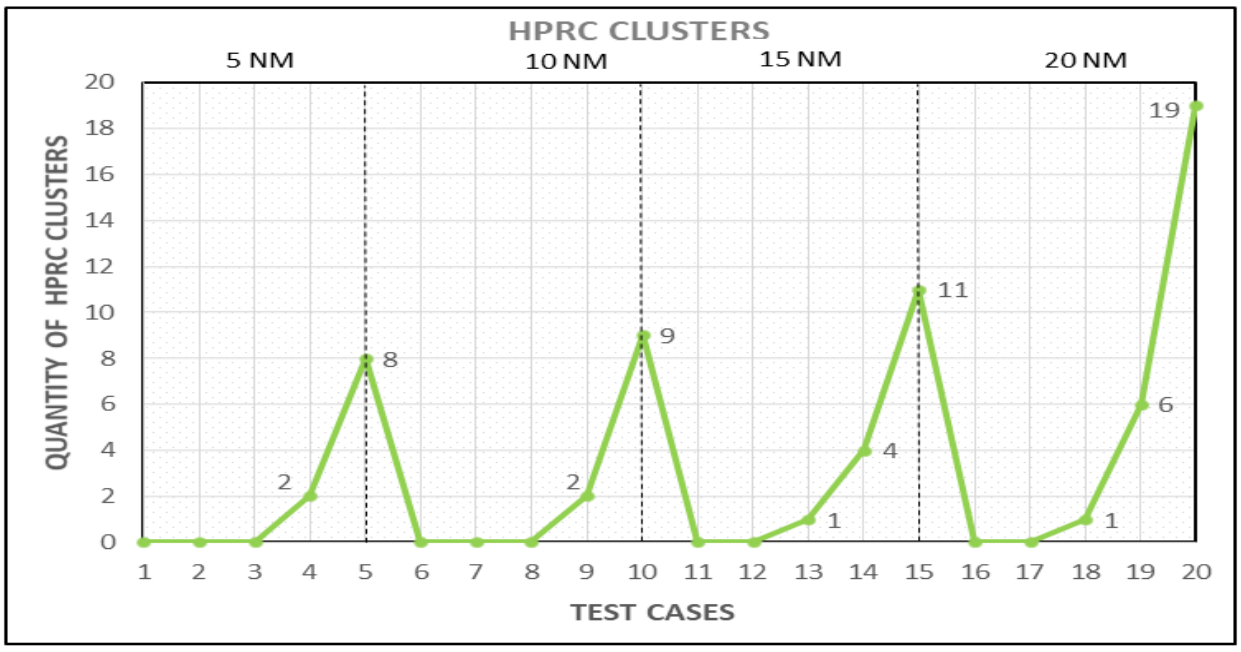


Figure V-14: *Occurring HPRC clusters in all test cases*

As expected, the quantity of HPRC clusters increases as the proximity radius and the number

[1] the network manager extra process

of aircraft increases as portrayed by Figure V-14. Same conclusion is found with clustered flights as displayed in Figure V-15.

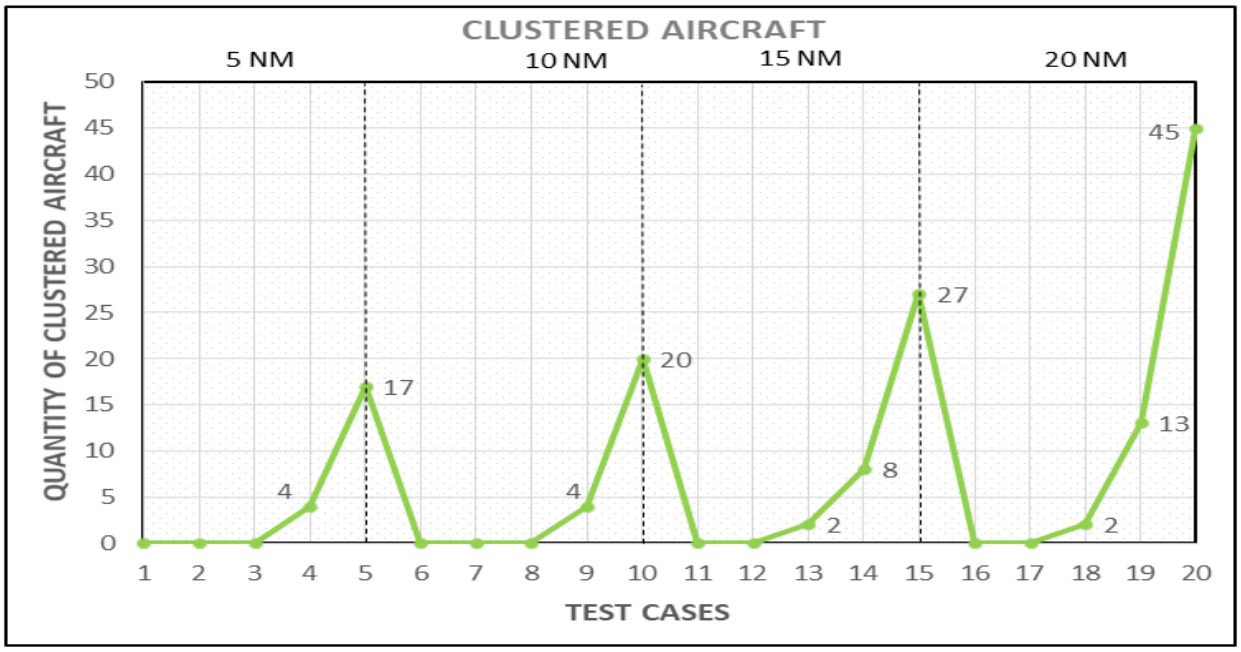


Figure V-15: *Clustered aircraft in all test cases*

Figure V-16 shows the average, maximum and minimum quantity of HPRC clusters occurring at the same time, in each test case. Up to seven clusters at the same time occurred in case 20, PR = 20 NM / aircraft quantity = 256.

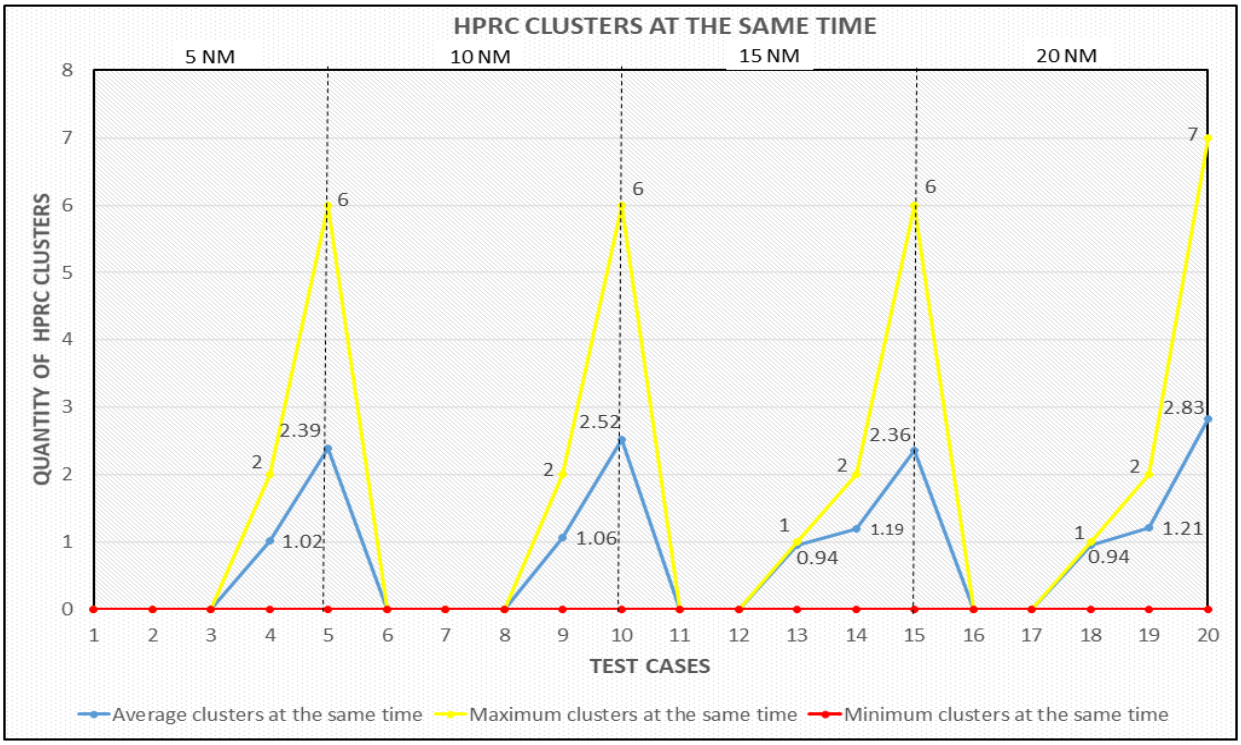


Figure V-16: *HPRC clusters at the same time in all test cases*

Furthermore, cluster size, i.e. the quantity of aircraft within a cluster, varies across the execution of an air traffic scenario, i.e. dynamic adding and removal of entities in a HPRC cluster of aircraft, which demonstrates scalability, as portrayed by Figure V-17.

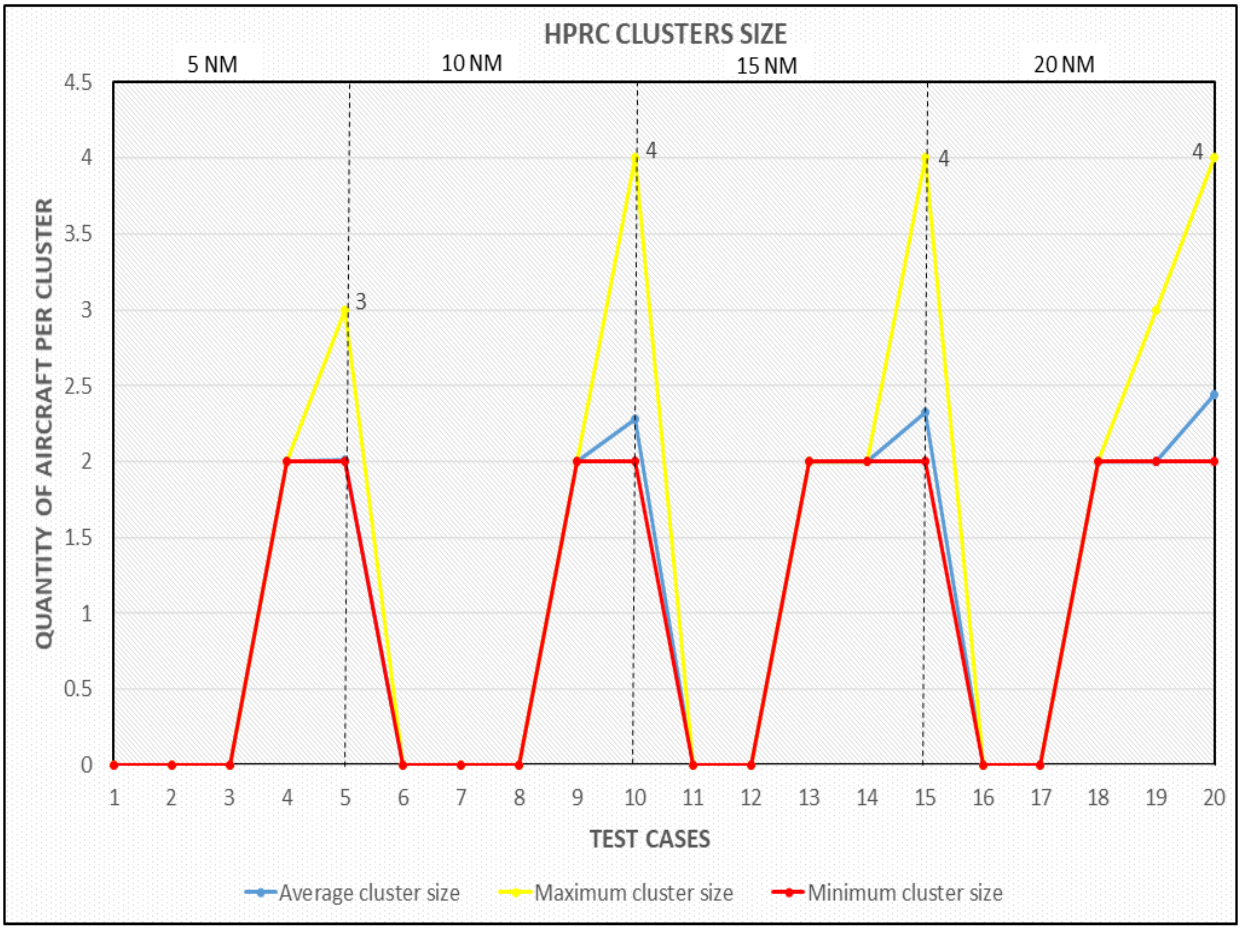


Figure V-17: *HPRC clusters size in all test cases*

In order to fly cooperatively, aircraft within a HPRC cluster exchange telemetry MPI messages, composed of their individual track and ground speed, which are used to compute Vicsek model, i.e. their next position in their local HPRC trajectory. If HPRC clusters were to be implemented in real life, its communications cost would be very low as evidenced by the simulation results, with a maximum of approximately 0.0001 MB (Figure V-18-A) of total exchanged data (location coordinates, heading, etc) with 0.008 seconds (Figure V-18-B) of total exchanging time for around 9000 messages exchanged in total (Figure V-18-C) for all emergent HPRC clusters.

It is worth mentioning that the nodes in the HPC infrastructure, used for the execution of the simulation, are using Gigabit Ethernet as communications technology. Nevertheless, with expected advancements in communication technologies, i.e. 5G (ITU, n.d.), HPRC cluster of aircraft or UAVs have a good potential to become a reality.

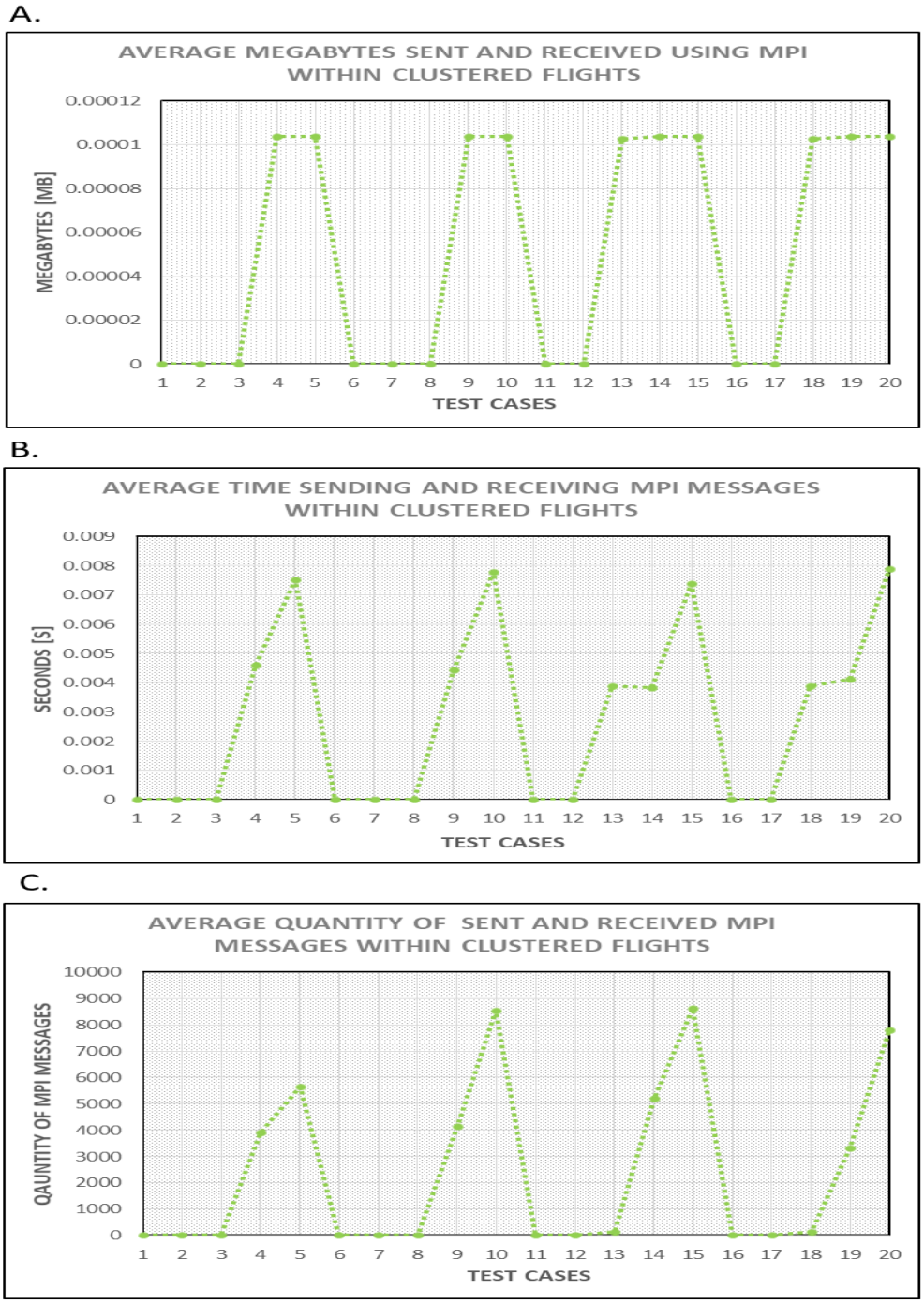


Figure V-18: *HPRC clustering communications cost in all test cases. A. Average MB sent and received within HPRC clusters, B. MPI messages average sending and receiving time within HPRC clusters. C. Number of sent and received MPI messages within HPRC clusters*

Regarding fuel consumption, Figure V-19 shows total fuel savings (all aircraft) with a minimum of $1,996.02$ kg in test case 4 (PR=5 NM / Aircraft = 128), up to a maximum of $14,789.10$ kg in test case 20 (PR=20 NM / Aircraft = 256), saved in comparison with the original trajectories. Furthermore, an average of $6,854.65$ kg is saved amongst all test cases. However, at individual level, not all aircraft trajectories result into fuel savings. In fact, amongst all test cases, an average of 69.48 % of flights achieved fuel savings. The remaining 30.52 % of trajectories incurred into fuel increases, in comparison with their original trajectories as observable in Figure V-20. This occurs because in the approaching phase, i.e. when flights are clustered, there are not fuel tests.

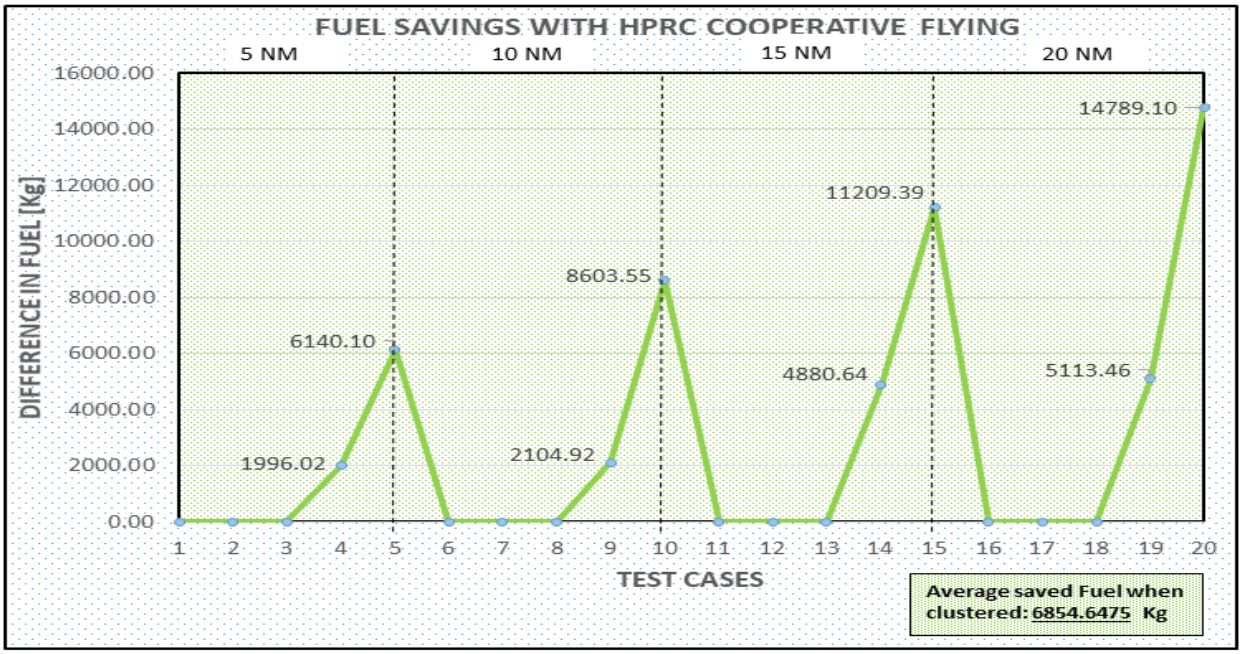


Figure V-19: *HPRC fuel savings in all test cases*

These results suggest that the proximity radius (PR) for clustering should be smaller than the ones used for the simulation. However, with smaller PRs, clusters did not emerge for the contemplated test cases. For future work, simulations considering larger quantities of aircraft, not specifically flying to the same airport, will be evaluated. Moreover, similar fuel tests will be included in clustering approaching phase.

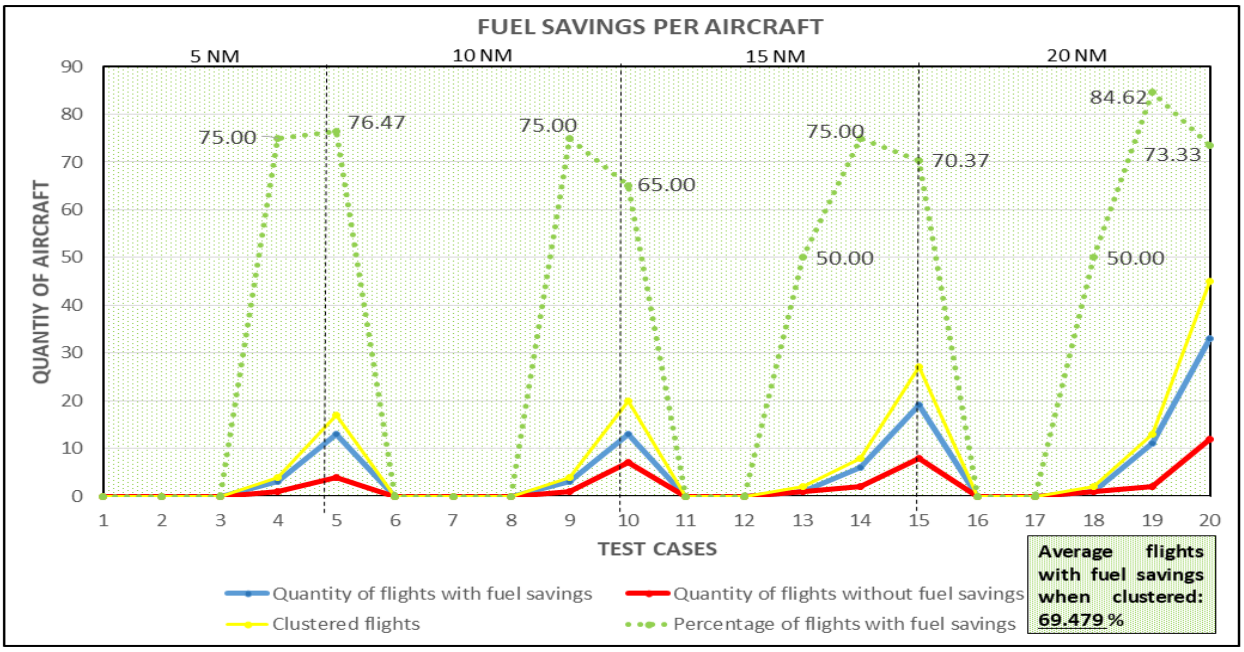


Figure V-20: *HPRC fuel savings per aircraft in all test cases*

Figure V-21 shows a comparison between fuel savings and *ANH*. As expected, the higher the ANH, the higher the total fuel savings. These results are promising and suggest that having strategical, rather than tactical HPRC clustered flights could lead to enormous fuel savings.

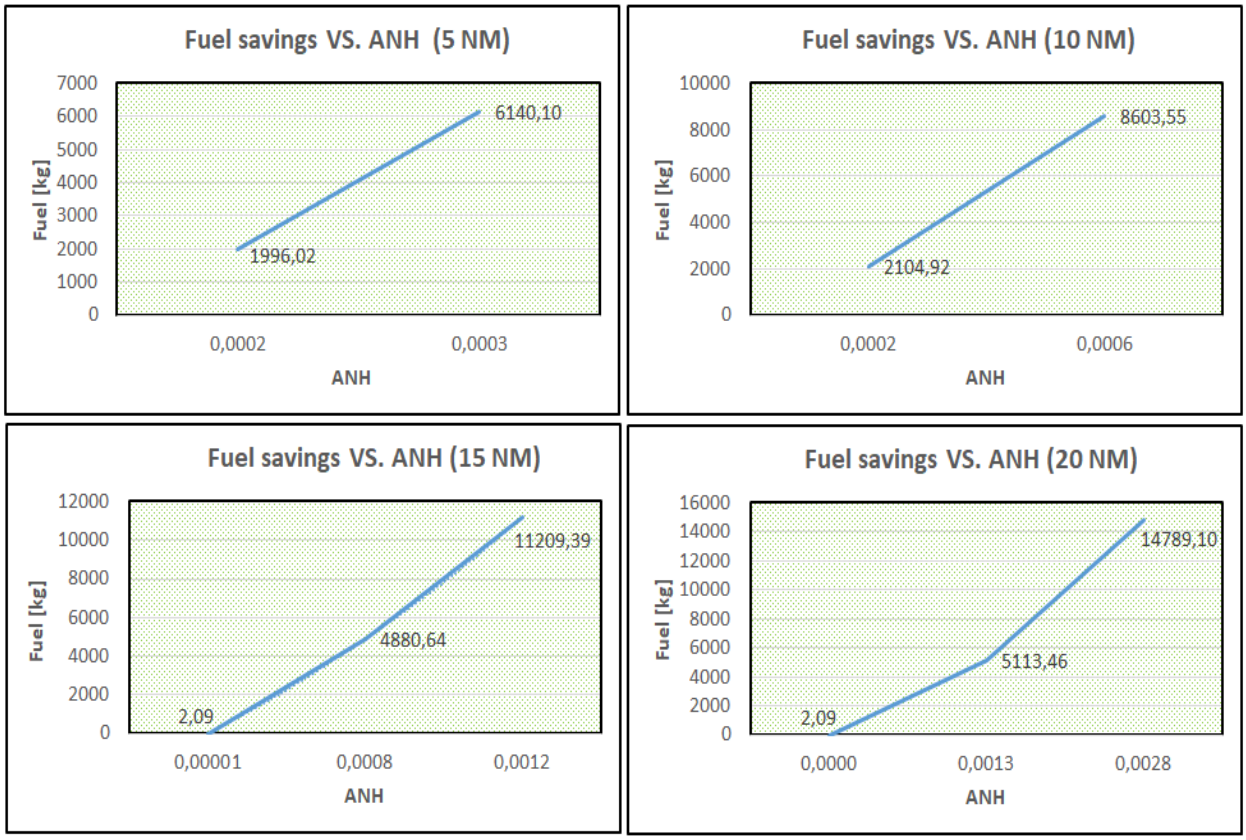


Figure V-21: *Fuel savings VS. Aircraft Network Hierarchy*

Furthermore, if HPRC clusters were to be implemented at strategic level, GRC could increase which will lead to a more stable aircraft network (see chapter III stability section III.6).

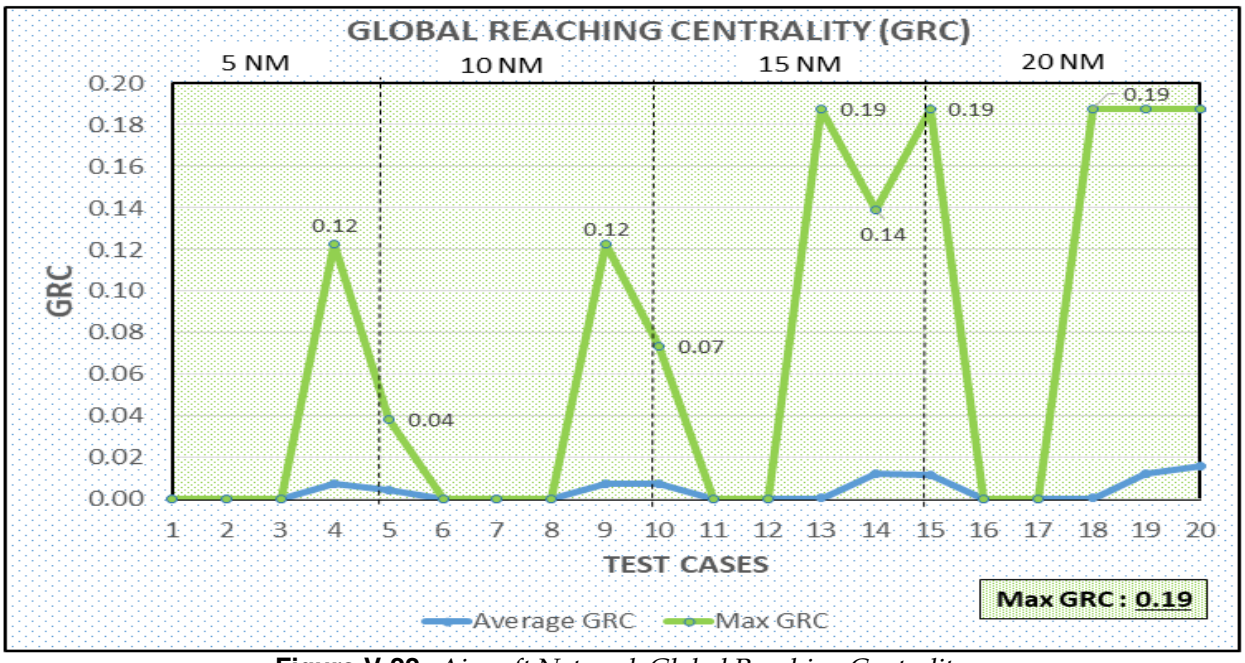


Figure V-22: *Aircraft Network Global Reaching Centrality*

The simulation results suggest that with current CONOPS, a maximum of 0.19 GRC occurs for the selected date over FABEC airspace, an unstable hierarchical network as observed in Fig-

ure V-22. However, only aircraft flying to LFPG were analyzed. Finally, as the quantity of aircraft increases so does the computing time (*clock time*) of the simulation, up to approximately 13 hours with 256 aircraft test cases as displayed in Figure V-23. In fact, the computing cost is caused by the clustering algorithm performed by the network manager process. However, the use of several less computing cores than parallel processes certainly impacts negatively the performance results.

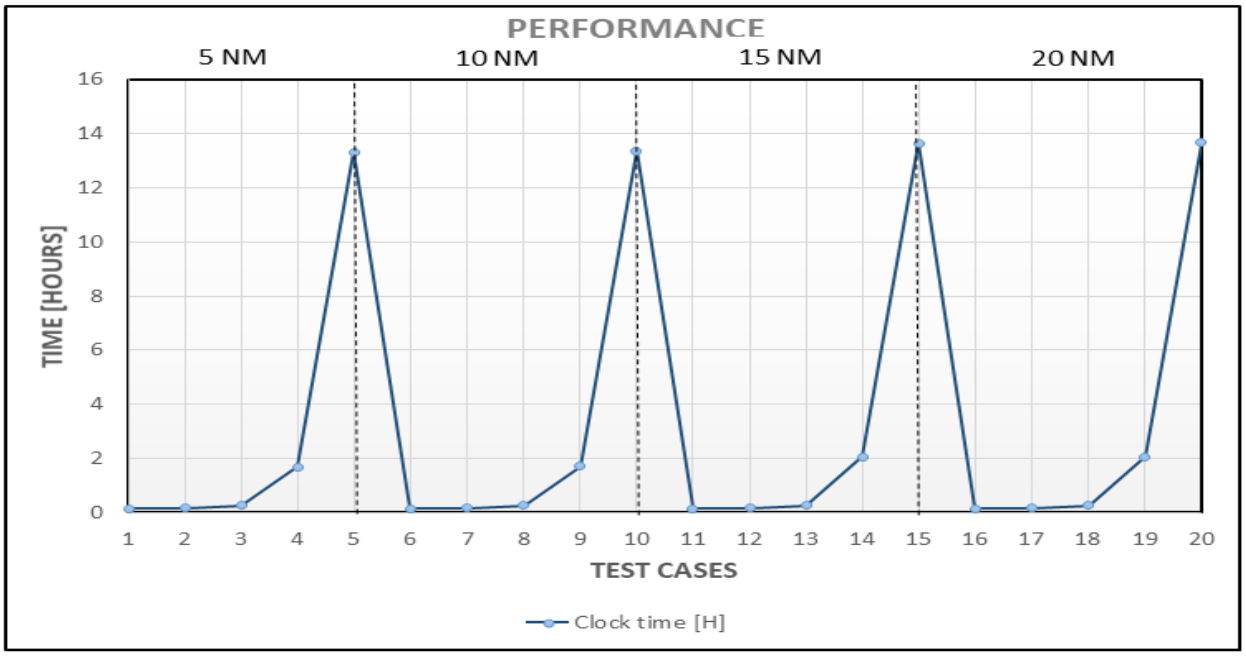


Figure V-23: *HPRC cluster of aircraft simulator performance*

The philosophy behind HPRC and ubiquitous supercomputing can be applied to all kind of vehicles, even those in manned mode, i.e. aircraft. With advancements in embedded computing cards and communications technologies, the implementation of HPRC clusters is a logical next transition that can bring all supercomputing features into the context of multi-vehicle systems.

The ideas behind HPRC cooperative flying can be used to produce significant savings in total fuel consumption, scalability and stability as proposed by the Aircraft Network Hierarchy performance indicator. However, extensive tests, considering larger quantity of aircraft and multiple airports scenarios need to be done to have a better estimation of the impact of using HPRC clusters. In fact, perhaps the most accurate results are those given by the test cases with a proximity radius of five nautical miles, because in the other test cases, the clustering approaching phase can cause an increasing individual fuel consumption. Nevertheless, the experiments in here are not intended to actually find the proper proximity radius, nor the most suitable clustering or cooperatively flying algorithms, etc., that would lead to more fuel efficient air traffic scenarios but rather to introduce novel ideas that have the potential of doing so.

Furthermore, the simulator is a very useful tool that can be exploited in order to find such optimal configurations. In addition, its component-based approach, based on the SimPlat (chapter IV section IV.4) and TAC philosophy, allows easy inclusion of new algorithms e.g. flying formation or different clustering schemes and moreover its use of parallel software libraries, allows its execution to properly scale over its underlying HPC infrastructure.

In future work, substantial simulations to better understand the use of HPRC clusters in real air traffic scenarios and the behavior of the ANH performance indicator will be carried out. Furthermore, clustering algorithms will be optimized to decrease simulation-computing times. If the results of such simulations are as positive as the ones found with the presented benchmark, implementing HPRC clusters at strategic level hold an enormous potential in fuel savings and cor-

respondingly environmental impact, especially with the expected increases in air traffic demand for the upcoming years. However, another air traffic indicator would need to be addressed, i.e. safety.

The use of HPRC could also lead to lower air traffic controllers (ATC) workload, because clustered aircraft would have information that can be used locally to avoid conflicts. Moreover, even non-clustered aircraft could exchange telemetry information to distributively coordinate safer air traffic scenarios. Many different schemes can be designed, even a ubiquitous supercomputing system made up of all aircraft, both flying and on-ground, all airports and all ATC ground stations could be set, creating therefore a very powerful hierarchical network exploiting intelligently supercomputing features.

In the previous sections V.1 and V.2, it was demonstrated that HPRC clusters are implementable nowadays with embedded computing boards such as the Raspberry Pi and interacting with simulated autopilots, in order to perform cooperative motion. These results extend to aircraft, which could easily embed similar computing boards, keeping the main avionics system and the pilot as the main actors but exploiting the extra knowledge and features given by the HPRC cluster of aircraft.

V.4 Tigers vs. Hunters

In order to discuss a full example of The ARCHADE, this section introduces the *Tigers vs. Hunters* system. The system uses specific classes and mission software developed for it and it consists of one human operator, a set of simulated robots (two UAVs and two UGVs), a ground station and a HPC cluster of computers, used for a mission described as:

- Localization of a group of hunters and a group of tigers distributed on a geographic area
- The protection of the tigers from the hunters and
- The identification of the hunters.

The ARCHADE framework allows the creation of new classes by inheriting from the three main framework classes layers: core, things and robotics (chapter IV section IV.2). For the system example, the TAC and DroneKit classes *dkdrone* and *dkrover* were used. Objects, instantiated from the two classes, can interact with the ArduPilot SITL and directly with MAVProxy (ArduPilot, n.d.) for real entities, facilitating easy transition from experimentation to real mode. The system is in fact a SITL simulator, because in each of the robot entities, the ArduPilot SITL has been deployed. The ubiquitous supercomputing infrastructure used for this application is the HPRC-HPC cluster, depicted in Figure V-2. Specific details about the infrastructure and the system's entites are described in Table V-5, where:

- *Pack 1*: Camera, thermal camera, GPU, LIDAR
- *Pack 2*: Camera, thermal camera, weaponry, LIDAR

The devices, i.e. packs, in each entity are not actually real for the simulation but they are used by the matching service for task assignment.

Table V-5: *Tigers vs. Hunters system entities information. Agents & Roles = A&R, Notations: SC = System Controller, MN = Master Node, SN = Slave Node, INT = Interface, PM = Physical Machine, VM = Virtual Machine, L = Laptop, HPCC = HPC cluster, RPI3B = Raspberry Pi 3 Model B, APV = ArduPilot Vehicle AC = ArduCopter, AR2 = APMrover2, DKD = dkdrone, DKR = dkrover*

Entity	nodes	OS	Type	A&R	CPU cores	RAM [GB]	APV	Devices
Demo	1	Ubuntu 14.04	PM - L	SC, MN, SN	4	24	None	None
drone1	1	Ubuntu 16.04	VM	E, SN	2	2	AC DKD	Pack 1
drone2	1	Ubuntu 16.04	VM	E, SN	2	2	AC DKD	Pack 1
rover1	1	Ubuntu 16.04	VM	E, SN	2	2	AR2 DKR	Pack 2
rover2	1	Raspbian Stretch	PM - RPI3B	E, SN	4	1	AR2 DKR	Pack 2
HPC cluster	7	Ubuntu 16.04	HPCC	INT	26	NA	None	None

Moreover, Figure V-24 presents the mission tasks' network. Links between tasks represent event and action dependencies.

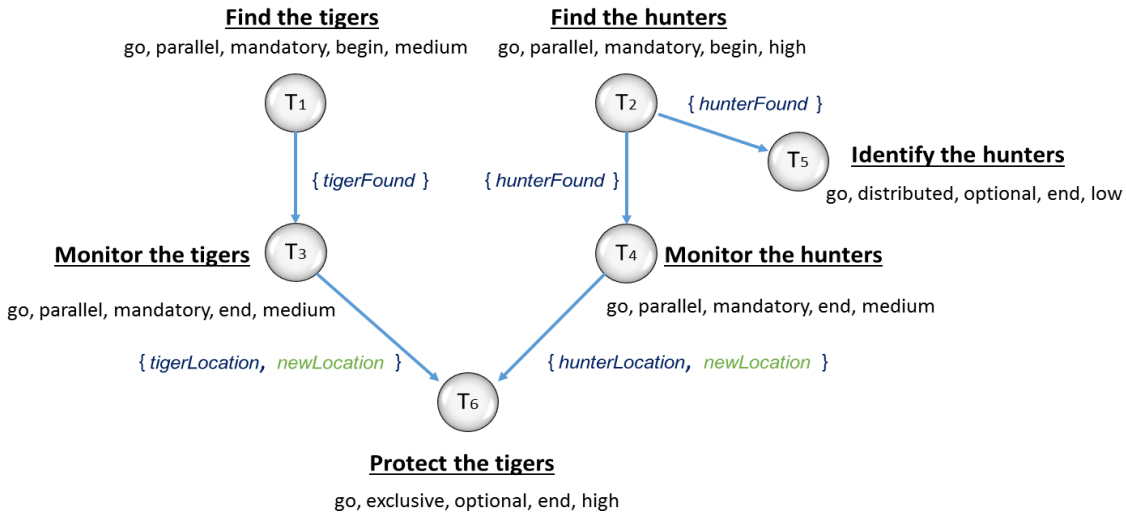
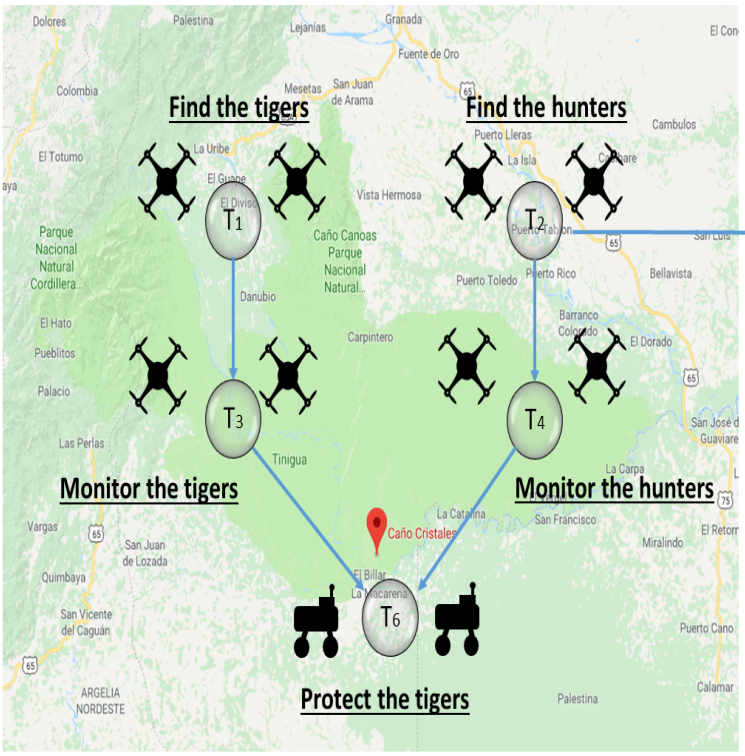


Figure V-24: *Tigers vs. Hunters mission network. Events and actions triggering forward tasks are contained within curved brackets*

After using the matching service, which is automatically called by the system controller agent, tasks are assigned as shown in Figure V-25.



Task	Task software
Find the tigers	*findTigers*
Find the hunters	*findHunters*
Monitor the tigers	*monitorTigers*
Monitor the hunters	*monitorHunters*
Identity the hunters	*identifyHunters*
Protect the tigers	*protectTigers*

Area name: Caño Cristales, Meta, Colombia, **Latitude**: 2.264897, **Longitude**: -73.794225, **Altitude**: 276.90

Figure V-25: *Tigers vs. Hunters matched tasks. Matched tasks after using the matching service. Map image taken from google maps.*

The matching service checks the available resources (specified in the entities' templates. See chapter IV Table IV-4) and assign the tasks, requiring such resources, to the corresponding entities taking into consideration the tasks' classification (e.g. blocking, execution, etc.). Furthermore, the robotic entities are grouped as *air group* (UAVs) and *ground group* (UGVs). The environment area is equally distributed within the groups i.e. each of the UAVs, in the air group and each of the UGVs, in the ground group are assigned half of the area. For the simulation, a Colombian area known as *Caño Cristales* was chosen. It is colloquially known as *El río de siete colores*, the seven-color river. While there are not actual tigers in such location, the location was chosen to inspire the reader to visit the place and experience its beauty.

The process of defining a mission consists simply on filling up the mission template and once the agents are running, the mission execution is automatic. However, the mission software, i.e. the software for each task should make use of live service API to create events and actions. Consequently, the framework and the middleware allow the creation of automatic ubiquitous supercomputing systems for multi-purpose missions that only require from the user creating the system, to write the software for each of its designated tasks.

Since the system is a simulator, the tasks' software does not actually do what would be necessary if the system were to be used for the real mission. However, since the mission software does use the live service API methods for events and actions creation, the entire mission is in fact simulated and it serves well as a proof of concept of The ARCHADE and the ubiquitous supercomputing ontology. Figure V-26 shows the complete mission workflow, including orders commanding and events/actions occurrence.

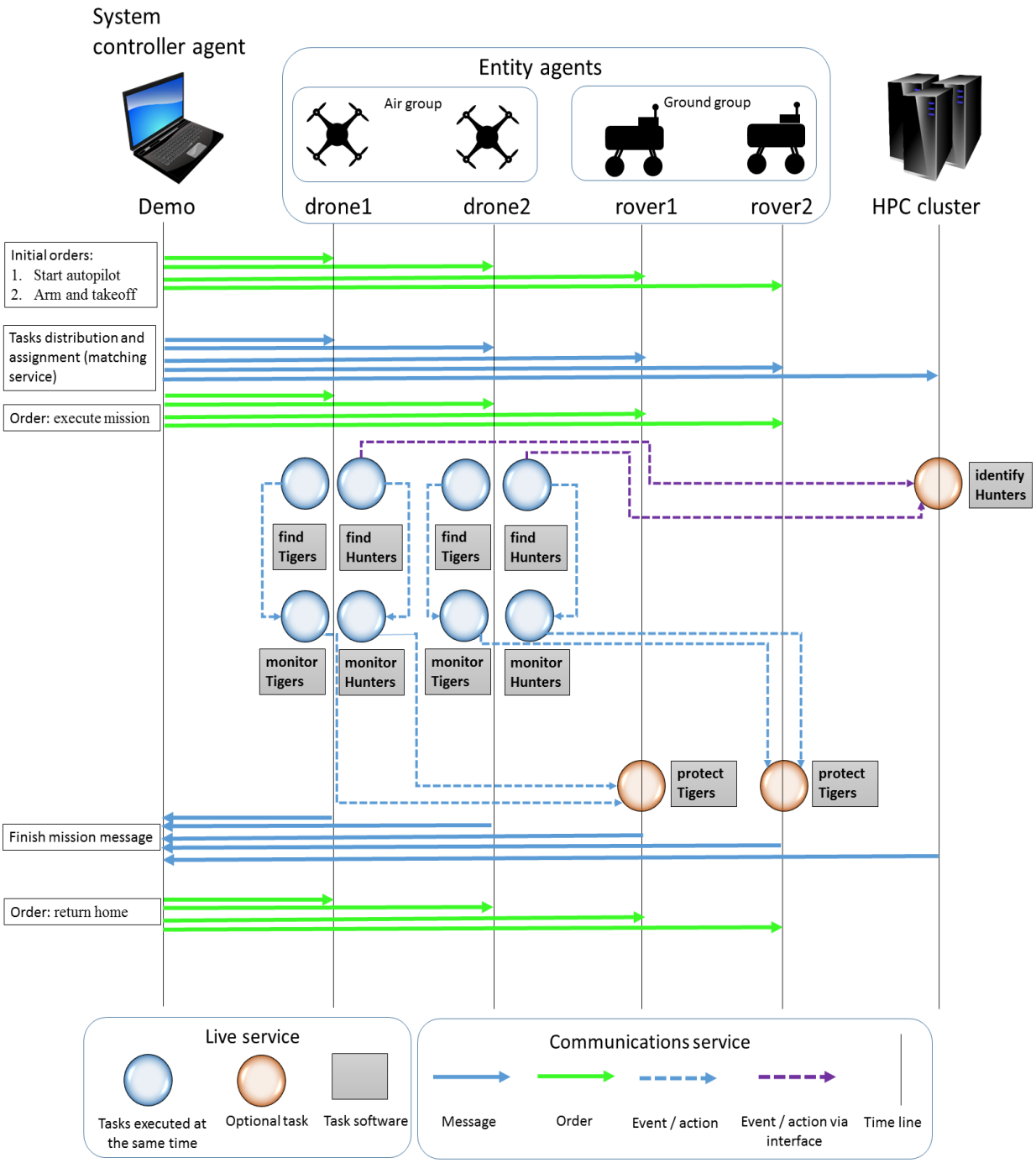


Figure V-26: *Tigers vs. Hunters mission workflow. There is corresponding software for each task. Events should be named in the same way that are inputted in the mission template. Correspondingly, the agents running in each entity constantly polls the shared file system to detect in-events in order to continue with its assigned tasks. The area is distributed equally between entities within a group. Consequently, events and actions are exchanged between entities in the same sub area in the case of protectTigers software. IdentifyHunters is a MPI parallel software.*

As mentioned in chapter III section III.3, each task in a mission network is to be linked with a independent software. Therefore, for the Tigers vs. Hunters system, the following tasks software were developed:

- *findTigers*: The software findTigers simulates image processing for tiger detection over the designated area. It uses the live service API to create a *tigerFound* event in an entities' shared folder using the file system layer. In a real software, i.e. use for the actual finding of tigers, the event can be coupled with the tigers' location data, which can be used by the next task (*monitorTigers*) to set its initial monitoring coordinates. For the purpose of the simulator, such locations are randomly fabricated.
- *findHunters*: The software findHunters simulates image processing for hunter detection over the designated area. It uses the live service API to create a *hunterFound* event in an entities' shared folder using the file system layer.
- *monitorTigers*: The software monitorTigers simulates the use of a LIDAR for the monitoring of the tigers found by the findTigers software. It uses the live service API to create a *tigerLocation* event coupled with a *newLocation* action in an entities' shared folder using the file system layer.
- *monitorHunters*: The software monitorHunters simulates the use of a LIDAR for the monitoring of the hunters found by the findHunters software. It uses the live service API to create a *hunterLocation* event coupled with a *newLocation* action in an entities' shared folder using the file system layer
- *protectTigers*: The software protectTigers reads events and actions from the software monitorTigers and monitorHunters, commands the motion of the rovers towards the tigers location (newLocation action) and calculates the distance to the hunters. If the calculated distance is below a predefined tolerance value, the rovers are authorized to scare the hunters by using gas bombs. This is a simulated process and the protectTigers software is only triggered if the hunters and the tigers are found.
- *identifyHunters*: The software identifyHunters is executed by the ground station, which interfaces (INT) with the HPC cluster of computers, by executing MPI parallel software, which simulates the identification of the hunters.

Finally, Figure V-27 shows a set of pictures of the implemented simulator. This section shows a proof of concept of The ARCHADE which is used to stimulate the reader to imagine the potential applications of ubiquitous supercomputing. Furthermore, next section will introduce a set of real missions carried out with The ARCHADE.

Figure V-27: *Tigers vs. Hunters simulator. (A) HPC/HPRC system platform with agents running in each entity (black boxes xterm in the left corner of the screen). The cluster is not shown in the picture (B) Running simulator. (C) Simulated drone. (D) Simulated rover. (E) Raspberry Pi 3 model B used for simulated rover. (F) Simulated drone carrying out tasks findTigers and findHunters. (G) Simulated drone finishing findTigers and findHunters tasks and starting tasks monitorTigers and monitorHunters tasks. (H) Ground station finalizing task identifyHunters, the displayed pictures show the two coadvisors of this Ph.D thesis. For clarification, they are not tigers' hunters in real life.*

V.5 Complete missions

In this section, a set of general-purpose computing missions (GPCMs) carried out with The ARCHADE are discussed. All missions were executed with the *HPRC-Rovers cluster* (Figure V-28) ubiquitous supercomputing infrastructure. Each of the UGVs in the HPRC-Rovers cluster is composed of:

- Traxxas Slash 1/10 2W (Traxxas, n.d.) chassis (1/10 ratio aspect and back wheels traction). The RC car is commonly used in competitions.
- Raspberry Pi 3 model B installed with Kali Linux 2018.3 RPI3 and The ARCHADE
- Pixhawk 2.1 (Dronecode, n.d.b) for hackrover1 and Pixhawk 1 (ARDUPILOT, n.d.c) for hackrover2
- Two Alfa AWUS036NEH Wi-Fi antennas (ALFA, n.d.), capable of monitoring mode and packet injection. One of the antennas is used to connect to a network used for TAC communications (*TAC network*), while the other is used for mission purposes to be described in the following.
- GPS module

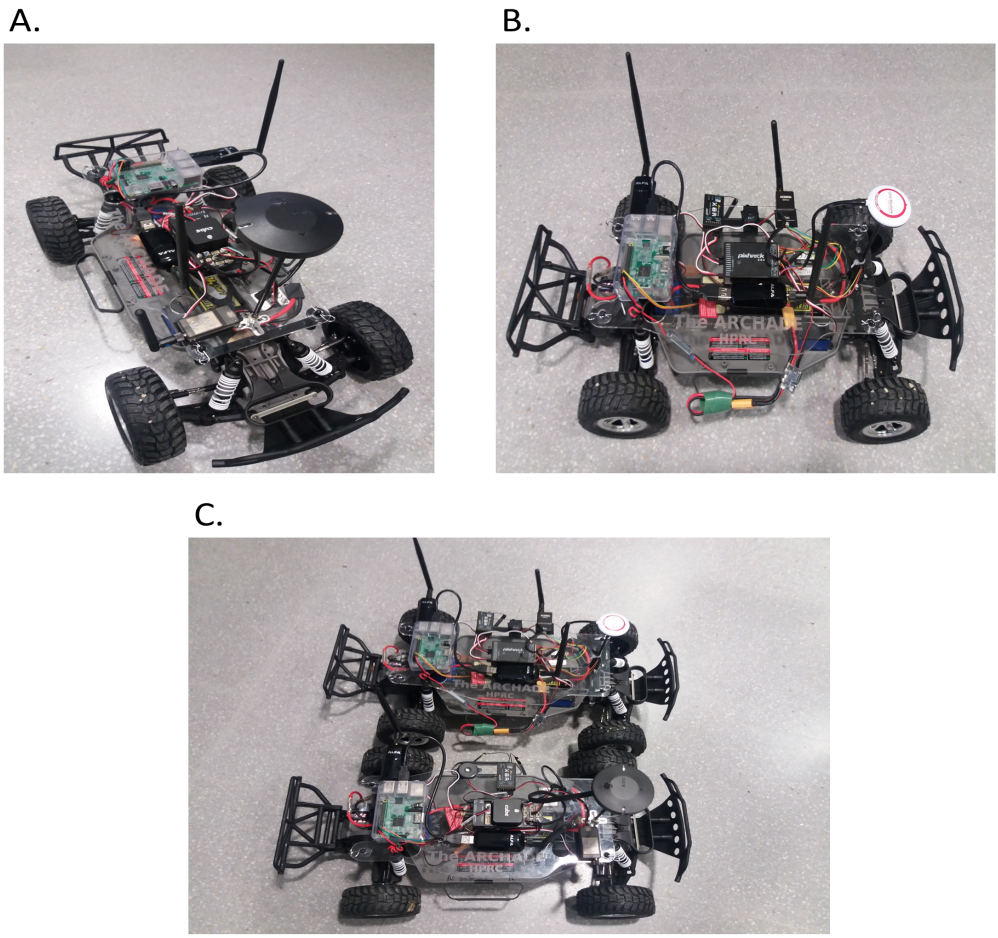


Figure V-28: *HPRC-Rovers cluster. A. Hackrover1, B. Hackrover2. C. Hackrovers. The laptop is not portrayed in here, but it can be observed in Figure V-2*

Furthermore, the ground station (the system controller) was installed with Ubuntu 18.04 and was set as the HPRC master node in a HPRC cluster composed of 3 nodes, the ground station and the two Raspberry Pi, embedded in the hackrovers.

V.5.1 The missions

The three General-Purpose Computing Missions are: *simple motion*, *follow me* and *WPA cracking*. The missions' selection differs from common approaches found in the literature, especially those benefiting from supercomputing in its traditional approach, i.e. as a speeding up tool, such as Marjovi *et al.* (2012). This decision follows two objectives:

- To demonstrate that supercomputing does not necessarily need to focus solely on computing efficiency / performance. However, supercomputing in its traditional approach will be discussed with the WPA cracking mission.
- To show that The ARCHADE can be used for all kind of missions, even those not commonly found in robotics research.

The mission simple motion does not execute any computing task and consists on automatically following a set of waypoints, a process carried out by TAC's live service. For the other missions, each one consisting on a single computing task, software was written using TAC's API or specific components' API. Moreover, mobile entities in the follow me mission are configured as *piloted*, i.e. their motion is controlled at mission software level, rather than using TAC's live service.

Figure V-29: *Cooperative area splitting missions. The two rovers are assigned with half of the mission area. Missions with area splitting are: Simple motion and WPA cracking. Image created with Google Earth Google (n.d.) and TAC's mission data*

The missions demonstrate different cooperative approaches, for example the simple motion and WPA cracking missions are *area-cooperative*, i.e. the given area is splitted between the mobile entities (hackrovers) as shown in Figure V-29. Moreover, the follow me and WPA cracking missions are *software-cooperative* (distributed task - chapter III section III.3) i.e. both use MPI software, executed in both the Raspberry Pi nodes in parallel. In the case of the follow me mission, one rover follows the other as shown in Figure V-30.

Figure V-30: *Follow me mission. Hackrover2 follows hackrover1 using MPI collaborative software. Image created with Google Earth Google (n.d.) and TAC's mission data.*

The *followMe* software, used for the follow me mission, is part of the TAC PLUS (chapter IV section IV.5) Motion module and it allows a master/leader to command any quantity of slaves to follow it by broadcasting its current and next position, via non computationally-expensive MPI software, a good example of the potential of HPRC to go beyond computing efficiency. Contrary, the WPA cracking mission software is computationally expensive because it consists on a dictionary-based brute-force attack. A dictionary is a list of potential passwords, that is used to attempt to crack a Wi-Fi network password.

The WPA cracking software, developed for these tests, is a Multiple Instruction Multiple Data (MIMD) application (according to the Flynn's taxonomy (Flynn, 1972)), written to automatically, i.e. without user supervision, crack WPA security passwords. The software was written with the API libraries in the TAC's PLUS security module, specifically the *crack* library and MPI, which allows it to be executed by any quantity of parallel processes, where each process performs a brute force attack, using a subset of the passwords in the dictionary (*data parallelism*).

However, before attempting individual brute-force attacks, a set of steps (black boxes in Figure V-31) carried out by only one of the parallel processes (the master process), are necessary. The complete software, depicted as *wpaCrackingRPI* is portrayed in Figure V-31.

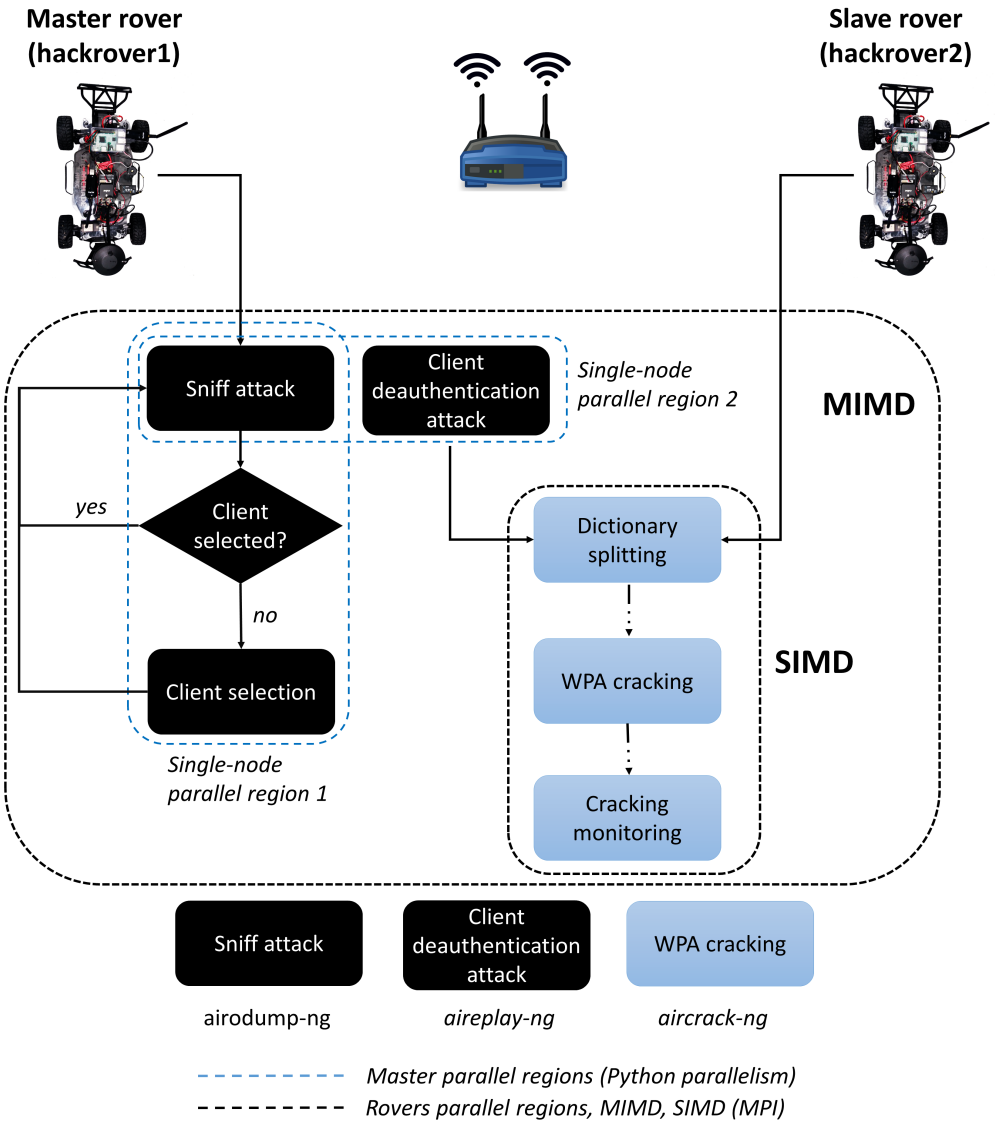


Figure V-31: *wpaCrackingRPI software. While the complete software is classified as MIMD, the brute force attack is classified as SIMD.*

While the software can be executed by any quantity of MPI processes, only one parallel process per rover was used (two MPI processes in total) in order to provide sufficient computing power to The ARCHADE services. This is because of the limited computing power available in the Raspberry Pi 3 model B (4-core CPU and 1 GB of RAM).

The steps carried out by the master rover, attempt to capture a *4-way handshake package*, which includes data that can be used to find the WPA/WPA2 (Wi-Fi Protected Access) password. A 4-way handshake package allows an access point (router) and a client/supplicant to prove independently that both know the PSK/PMK (Pre-Shared Key/Pairwise Master Key) without sharing the actual password (IEEE, 2004). This strategy was part of IEEE approach to solve WEB (Wired Equivalent Privacy) security vulnerabilities.

In parallel, the master MPI process (rank 0) performs a *sniff attack* (sniff attack black box) and identifies connected clients to the target network (client selection black box). The sniffing attack collects packages sent and received at the Wi-Fi router. To do so, the airodump-ng command, from

the aircrack-ng suite (Aircrack-ng, n.d.) is used as follows:

airodump-ng −−bssid *APMAC* −−channel *APChannel* −−write *packetsFile NDMON*

Where *APMAC* is the router's MAC address, *APChannel* the router's broadcasting channel, *packetsFile* the file where collected packets will be outputted and *NDMON* the network interface, in monitoring mode, used to collect packages (second antenna). Randomly, one client, connected to the Wi-Fi router, is selected. These two tasks (sniffing and client selection) are done in parallel via Python multprocessing library (*Single-node parallel region 1*). Once a client has been selected, a second parallel region is initiated. Handshake packages are only exchanged, between a router and a client, upon its first connection, therefore, coupled with the sniff attack, a deauthentication attack is performed (*Single-node parallel region 2*). To do so, the aireplay-ng command, from the aircrack-ng suite is used as follows:

aireplay-ng −−deauth *PCKGS* -a *APMAC* -c *CientMAC NDMON*

Where *PCKGS* is the quantity of deauthentication packets to be sent to both the router and the client and *CientMAC* is the client's MAC address. Via the deauthentication packets, both the router and the client are told that the other has disconnected. Therefore, a reconnection attempt is done, in which a 4-way handshake package is recorded by the sniff attack in the *packetsFile*. At this point, the master and slave processes split an existing dictionary among each other.

The dictionary was created previously to the execution of the mission with Kali's *crunch* tool. A secure password must include small and capital letters, numbers and symbols. Such guidelines plus the quantity of characters in the password can make a brute-force attack very slow, even potentially unachievable. In order to guarantee the success of this mission and given that the target router and both of its networks are known, the dictionary was created using a *pattern*, i.e. a set of known and a set of unknown characters for the creation of the potential passwords, resulting into two dictionaries, one composed of 512 and the other of 4096 potential passwords. This is done to reduce the computing time, given energy limitations in the UGVs, with a maximum autonomy of around 30 minutes.

To further decrease mission computing time, the original dictionary is splitted among the parallel processes during the execution of the mission. The splitting process is done at mission time in order to provide scalability, i.e. wpaCrackingRPI can be used by any quantity of parallel processes, not known until the execution of the mpirun directive. Following, each parallel process will attempt to find the password by combining data in the packetsFile with each of the potential passwords to generate a *Message Integrity Code (MIC)*, also collected in the packetsFile. If one of the passwords computes to the same MIC in the packetsFile, the password has been found, *WPA cracking* blue box. Handshake packets in the packetsFile are encrypted using AES (Advanced Encryption Standard) (Rijmen & Daemen, 2001). The WPA cracking process is done using the aircrack-ng command from the aircrack-ng suite as follows:

aircrack-ng *packetsFile* -w *splitDict*

Where *splitDict* is the split dictionary that each MPI process has. If one of the potential passwords at each of the parallel process results to compute the correct MIC, other parallel processes will stop its cracking attempt (*Cracking monitoring* blue box). WPA cracking is a very expensive computational problem that can be speed up using GPUs or using crunch and aircrack-ng at the same time, i.e. a dictionary file can be very big, which can slow I/O, specially when splitted dictionaries are shared via network protocols. However this optimization techniques are outside the scope of this Ph.D thesis.

The complete wpaCrackingRPI is a MIMD software, because parallel processes do not execute the same tasks over the same data, but it does include a SIMD region, i.e. the actual WPA cracking. Furthermore, the TAC PLUS security libraries, *dict*, *sniff*, *deauth* and *crack* were used within wpaCrackingRPI. Commonly, all of these attacks require user supervision, therefore the libraries were developed to provide hacking automation. The motivation to present such mission is twofold. First, to expose The ARCHADE's potential, by going beyond common missions found in the literature, such as SLAM, vision-based navigation, etc. Second, missions such as this, could have potential in military applications where it might be necessary being in proximity to a specific target, before attempting a hacking attack. However, this type of mission also represents a risk in civilian contexts. Once the password of a Wi-Fi network is cracked, the mobile robot could connect to the target network and perform all kind of attacks over the network clients. This lead to emphasize the necessity of devising strategies to guarantee user's privacy and security with the rise of the mobile robotics era.

In order to evaluate TAC's performance, the following section introduces a complete benchmark with the three missions discussed in here, in both simulated and real mode.

V.5.2 Benchmark

In this section, the results of executing the three proposed missions, *simple motion*, *follow me* and *WPA cracking* are discussed. All the missions were carried out in the modes:

- *Simulated* (*SIM*), i.e. using the SimPlat (chapter IV section IV.4) with ArduPilot SITL for the mobile entities.
- *Real* (*REAL*), i.e. using the two hackrovers shown in Figure V-28.

In both modes, the laptop shown in Figure V-2 was set as the system controller entity, ground station and HPRC master node. The missions were executed in both modes to observe TAC's transparency to *transcend* from experimentation to real-world missions, which from the user's point of view, is done solely by setting the tag *real* to *false* in each of the corresponding entity templates (chapter IV Table IV-4).

SIM mode was performed using the same Raspberry PIs, used as companion nodes in real mode, but running the SITL version of the ArduPilot software. Furthermore, the same Wi-Fi antennas and Wi-Fi router were used in both SIM and REAL mode.

The following results consider one test case per mission, i.e. each mission was executed once and individual mission results are compared and averaged to observe TAC's indicators disregarding the specific mission to be carried out by the same ubiquitous supercomputing system.

Figure V-32 displays the missions' total time duration, including the two test cases for the WPA cracking mission (512 and 4096 words dictionaries). The follow me mission has a longer duration than the other missions because it was set to carry multiple following rounds.

As it can be observed, the missions' duration is fairly similar when in SIM versus REAL mode. However, the slight differences between the two modes are potentially caused by operating system loads, network conditions, different performances between the ArduPilot SITL and its hardware implementation, etc., out of the control of The ARCHADE. Such differences were not investigated in detail given that do not compromise TAC's capacity to transcend from SIM to REAL mode. Moreover, an average between the four test cases of 21.85 seconds and a standard deviation of 13.12 seconds do not suggest specific reasons for the differences in mission time duration between SIM and REAL mode, other than unexpected occurrences as those mentioned before, e.g. different performances between the ArduPilot SITL and its hardware implementation.

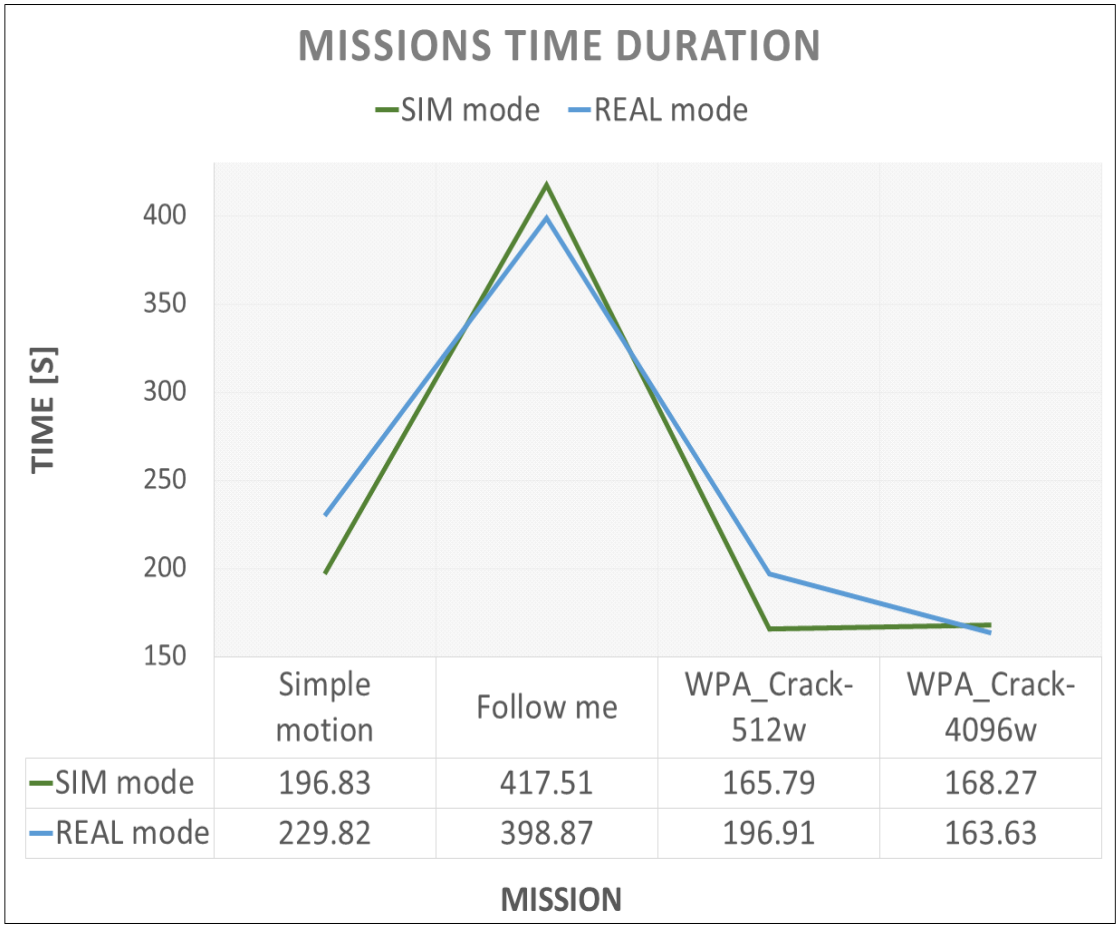


Figure V-32: *Missions' time duration*

The benchmark in this section is composed of the following tests:

- *Orders acknowledgment and hierarchy*: This test was done to discuss order acknowledgment and delays, with the objective of giving an estimation of TAC's control and hierarchy approaches.
- *Data synchronization*: In order to provide a single-image system, where data is collected in independent entities but ultimately synchronized to the system controller (ground station) and entities requiring it, these tests aimed at measuring synchronization data size and time.
- *CPU load and RAM usage*: To evaluate TAC's scalability features, average CPU load and RAM usage, both in the system controller and in the mobile entities, were measured.
- *Wi-Fi signal strength and latency*: These tests were carried out to measure the conditions of a HPRC cluster while in motion, an important aspect when dealing with supercomputing infrastructures, especially when considering MIMD software (Flynn, 1972), i.e. such that is based on message exchange.

The four items aimed at demonstrating TAC's features and ultimately to observe its capacity to transform a set of independent entities in a supercomputing infrastructure. Furthermore, two extra tests will be discussed, *scalability*, carried out in SIM mode and *WPA cracking software performance*. The WPA cracking mission is the only one in which supercomputing is done with the traditional frame of mind, i.e. performance. The results presented in here show that supercomputing, as it has been rethought in this Ph.D thesis, can still be used in its traditional approach.

V.5.2.1 Orders acknowledgment and hierarchy

During the execution of a mission, a set of default orders, as mentioned before, are commanded by the system controller and executed by all the entities depending of its nature, i.e. *static*, such as the ground station or *mobile*, like the UGVs. Such default orders, in its default sequence, are: *connect_to_autopilot*, *start_vehicle*, *execute_mission* and *return_home*. Non-mobile entities will not be commanded orders related to the autopilot, the vehicle, etc. Figure V-33 shows average order delay D (equation III.9) in SIM (A) and REAL (B) mode for the three missions, four test cases. Moreover, average orders delay OD (equation III.10) in SIM and REAL mode are displayed in Figure V-33-C.

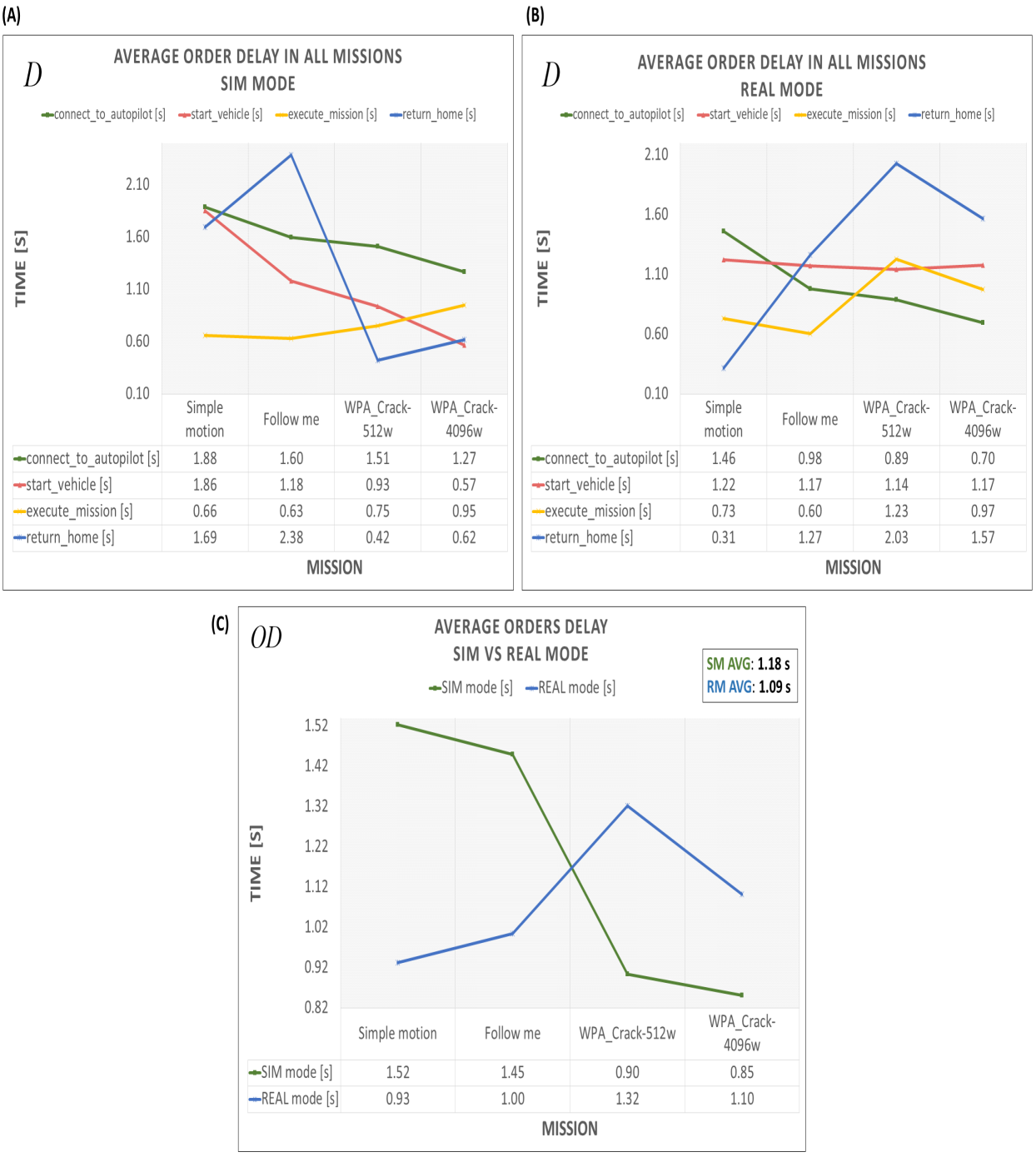


Figure V-33: *Orders delay SIM VS REAL mode*

As it can be observed, the maximum delay in SIM mode, is of 2.38 seconds, for the *return_home* order. Correspondingly, the maximum delay in REAL mode, is of 2.03 seconds, again for the *return_home* order. This order is the slowest to be acknowledged because of a set of processes, related to the *execute_mission* order, that are finalized before proceeding with the acknowledgment. In addition, the total average orders delay, including all the mission cases, is smaller in REAL mode (1.09 s) that in SIM mode (1.18 s), as depicted in figure Figure V-33-C. While the difference is almost neglectable, this is a good indication that mobility has not negatively affected the missions. Moreover, Table V-6 shows the average orders delay difference in seconds between SIM and REAL mode, for all missions.

Table V-6: *Average orders delay difference in seconds between SIM and REAL mode, for all missions*

Simple motion	Follow me	WPA_Crack-512W	WPA_Crack-4096W
0.59	0.45	0.42	0.25

With a total average of 0.43 seconds, these results confirm TAC's transparent transition between simulation and real mode. The system presented in this work, composed of the human operator, the system controller (SC and SC-E) and the 2 UGVs ($|E| = 5$) falls under the *automatic with human system operator* automation mode (chapter III Figure III-11), whose GRC is calculated accordingly to Equation III.21. Therefore, the optimal GRC, i.e. without disconnection, for the studied system is 0.8125. Given that there were no disconnections, during the execution of all the missions, *SH* (Equation III.8) remain as the optimal GRC in all cases. This is because of the small size ($\approx$ 30 m) of the area mission.

However, with a larger area-size, disconnections are quite possible, which will impact *SH* negatively. Nevertheless, for missions requiring large areas, mobile communications technologies, e.g. 4G, could provide sufficient performance to maintain constant connectivity between the entities of a system. Moreover, with the appearance of 5G and its powerful requirements (ITU, n.d.), ubiquitous supercomputing with little or non-disconnections is a true possibility with real-world missions.

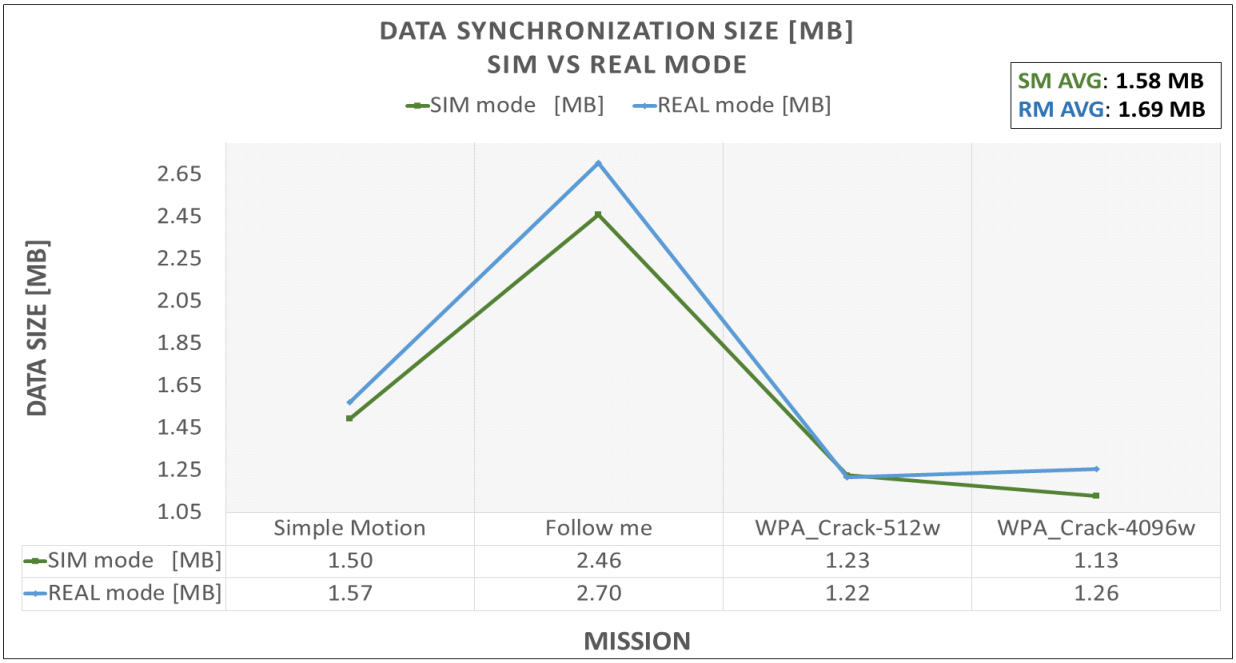
V.5.2.2 Data synchronization

In order to achieve a single image system, at all times, during the execution of a mission, the TAC's synchronization service automatically performs data synchronization between the entities and the system controller and vice versa. The TAC's synchronization service guarantees that:

- The system controller, e.g. the ground station, holds a local copy of all mission-related data. Such data includes the telemetry of each of the mobile entities, e.g. their position, velocity, attitude, GPS status, etc., mission-generated data, e.g. files, images, video, etc., and TAC data e.g. orders, events, logs, tasks, etc.
- Each entity receives orders and events data (from the system controller and other entities), holds telemetry data (of all mobile entities) and shared data, e.g. data required for mission execution or TAC purposes.
- If necessary, the entities are aware of the mission status and corresponding tasks.

Data synchronization is affected by two factors. First, the duration of the mission i.e. the longer the mission, more data to be synchronized. For example, telemetry data is synchronized

(A)



(B)

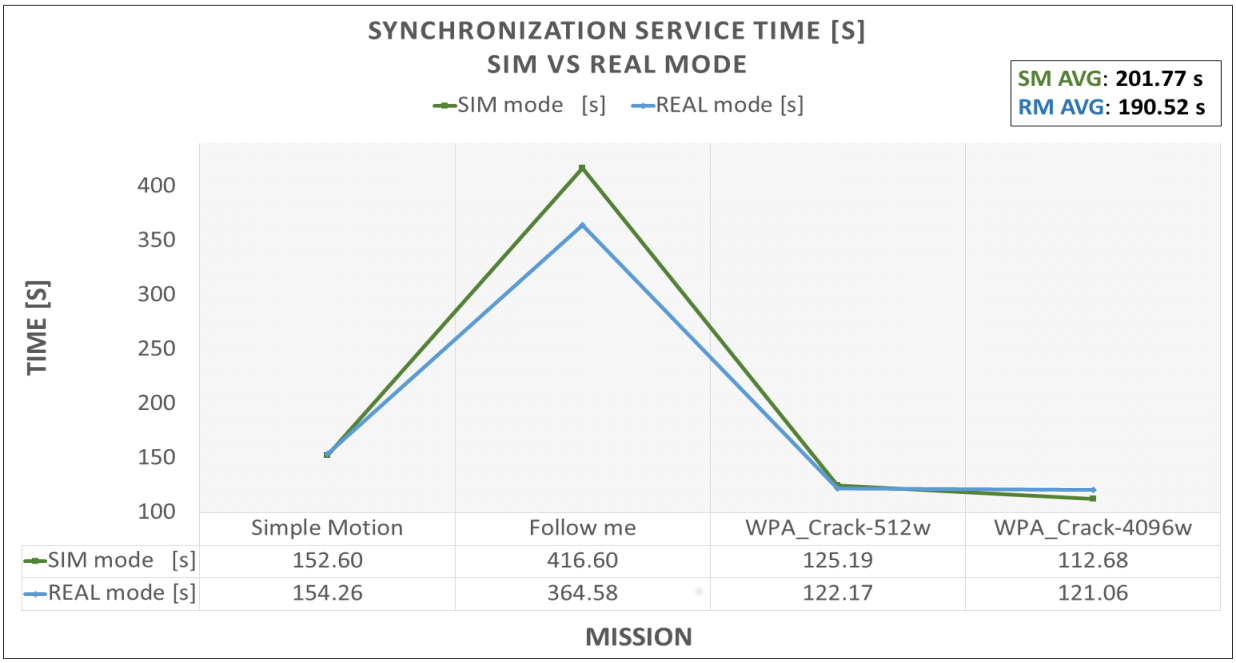


Figure V-34: *Data synchronization per mission, SIM VS REAL mode*

each second during the execution of the mission. Second, the specific network conditions in a synchronization iteration, which can affect data synchronization time. When in SIM mode, the SITL Raspberry PIs are not actually moving, therefore communications are not potentially affected such as in REAL mode. However, it is possible that better network conditions occur during REAL mode vs SIM mode at specific moments during the execution of a mission, for example when a vehicle is in direct line of sight with the ground station or other vehicle.

Figure V-34 shows statistics from the synchronization service. The average total data, considering the three missions, amounts to 1.69 MB in REAL mode and 1.58 MB in SIM mode (Figure V-34-A). This includes all the aforementioned data, e.g. telemetry, etc. The data size, in both SIM and REAL mode, is minimal given the fact that these missions do not include heavy data acquisi-

tion, like imagery, video, etc. In the case of the follow me mission, its execution takes longer time than the other three cases as it can be observed in Figure V-32, therefore more data needed to be synchronized, e.g. logs (TAC and mission-related) and telemetry data. Moreover, the minimal differences in data size between SIM and REAL mode are caused by minor differences in telemetry files (e.g. variations between real and simulated GPS coordinates), logs, shared data, etc.

Regarding Figure V-34-B, the average of 190.52 seconds (REAL mode) and 201.77 seconds represent synchronization service time, i.e. the total average time, in the four test cases, that the synchronization service performed a synchronization task. Moreover, Table V-7 shows a comparison between mission time (MT) and synchronization time (TM).

Table V-7: *Mission time VS. Synchronization time. MT = Mission Time, ST = Synchronization Time*

Mission	SIM			REAL		
	MT	ST	ST/MT [%]	MT	ST	ST/MT [%]
Simple motion	196.83	152.60	77.53	229.82	154.26	67.12
Follow me	417.51	416.60	99.78	398.87	364.58	91.40
WPA_Crack-512w	165.79	125.19	75.51	196.91	122.17	62.04
WPA_Crack-4096w	168.27	112.68	66.97	163.63	121.06	73.98
Average ST/MT	**SIM**		**79.95**	**REAL**		**73.64**

As it can be observed in Table V-7, the average *ST/MT* is 6.31% higher in SIM mode that in REAL mode, i.e. for almost the same data, synchronization in REAL mode is faster. In fact, for all the test cases, except the WPA cracking with 4096 words dictionary, synchronization is faster in REAL mode. This is a positive result that suggests that communications can be better when having mobile entities. In the case of the follow me mission, synchronization over mission time ratio (ST/MT) is considerably higher than in the other missions. This is because, the follow me mission generates data more often than the other missions, in order to coordinate the cooperative motion of the two vehicles.

In a multi-robot system, where entities could result into all sort of failures, data synchronization mechanisms are very important in order to generate a single-image system, for example to facilitate task reassignment. The results in here show that very little data, less than 2 MB average is sufficient to coordinate a small ubiquitous supercomputing system, that does not generate high amounts of data, e.g. imagery, etc., i.e. this is mostly TAC data, including as well the two vehicles complete telemetry information. Ignoring data synchronized from the system controller towards the mobile entities, which is mostly related with orders, it is possible to assume that less than 1 MB is synchronized per mobile entity, therefore for N mobile entities, around N MB will have to be synchronized. This is a very positive result that shows that a ubiquitous supercomputing system does not need large quantities of data to fully operate and can scale properly. Furthermore, while the missions duration is fairly small, data is synchronized as it is created, therefore, a single-image system can be maintained at all times provided that communications are maintained correctly as it can be observed in Wi-Fi covered areas (see section V.5.2.4). Nevertheless, further testing with missions collecting heavy data and using communications technologies different from Wi-Fi need to be addressed to evaluate performance in such conditions, this is part of this Ph.D thesis future work.

V.5.2.3 CPU load and RAM usage

CPU load and RAM usage for the system controller (ground station) and the mobile entities (hackrovers/Raspberry PIs) are analyzed independently. This is because the computing features of the ground station and the Raspberry PIs are considerably different as presented in Table V-8.

Table V-8: *Entities' computing features*

Entity Type	CPU speed [MHz]	CPU cores	RAM [GB]
System controller (ground station)	3200	4	24
Mobile entities (Raspberry Pi)	900	4	1

Following Table V-9 shows total average CPU load and RAM usage, i.e. including operating system tasks and TAC's services, for the system controller and the mobile entities, considering the three missions and the four test cases, in SIM and REAL mode.

Table V-9: *System controller and mobile entities total average CPU load and average RAM usage*

Entity type	Average CPU load (%)		Average RAM usage (GB)	
	SIM mode	REAL mode	SIM mode	REAL mode
System controller	46.48	56.42	1.58	1.95
Mobile entities	68.44	59.91	0.17	0.16

The system controller's average CPU load, in both modes, accounts to around half of the total CPU computing power. This is because the system controller executes multiple TAC services plus it acts as the master HPRC node. Regarding the mobile entities, the average CPU load of 68.44% in SIM mode and 59.91% in REAL mode are very promising when dealing with embedded computing boards, such as the Raspberry Pi, which provides limited computing power as depicted in Table V-8. Moreover, minimal average RAM was used in both the system controller and the mobile entities during the execution of all studied missions.

Regarding the mobile entities, the differences in CPU load between SIM and REAL mode relate with the ArduPilot SITL software. The difference of 0.01 GB in RAM usage is neglectable. In the case of the system controller, both CPU load and RAM usage are higher in REAL mode. This needs further testing and analysis and it is left for future work given that the system controller, set as the ground station, is not simulated in both modes.

As previously mentioned, TAC's monitoring service current version computes total performance, which includes operating system tasks. Such approach is taken to evaluate complete CPU load and RAM usage, rather than only TAC-related indicators and observe the feasibility of a ubiquitous supercomputing infrastructure composed of embedded computing boards. However, future work will include specific TAC's CPU and RAM indicators. Nevertheless, these results show TAC's light weight, especially important when dealing with computing boards such as the Raspberry Pi or similar options. Furthermore, the ground station can be set upon an inexpensive computer with as much as 2 GB of RAM as evidenced in Table V-9. Even a mobile entity, embedded with a slightly more powerful board than the Raspberry Pi, could be set as master/system controller, if only RAM were to be considered.

For systems composed of larger quantities of entities, further tests need to be carried out in order to evaluate the system controller's performance. Nevertheless, in the scalability section

V.5.2.5, results of using three mobile entities, including an UAV, in SIM mode will be discussed.

V.5.2.4 Wi-Fi signal strength and latency

In this section, results regarding Wi-Fi signal strength and latency for the system controller and the mobile entities during the execution of the three missions, in SIM and REAL mode are discussed as portrayed in Table V-10. Wi-Fi signal strength is monitored periodically (monitoring service) via the operating system control over the network interface. Correspondingly, latency is computed periodically from each entity towards the rest of the entities in the system.

Table V-10: *System controller and mobile entities average Wi-Fi signal strength and latency*

Entity type	Average Wi-Fi signal strength (%)		Average latency (ms)	
	SIM mode	REAL mode	SIM mode	REAL mode
System controller	100	100	8.67	6.44
Mobile entities	100	91.17	20.81	14.75

The system's controller Wi-Fi signal strength, remained in optimal conditions (100 %), during the execution of all the missions, given that in both modes, SIM and REAL, the ground station is a non-mobile entity, located at a minimal distance from the Wi-Fi router. Same results occurred for the mobile entities in SIM mode. However, in REAL mode as expected, mobile entities average Wi-Fi signal strength was slightly less than optimal, 91.17 %.

Regarding latency, results are very promising. The system controller latency is of the uppermost importance given that it is the non-human entity with the highest hierarchy in charge of commanding default orders. With this in mind, the 8.67 ms and 6.44 ms average latencies in SIM and REAL mode respectively, are very good considering the mobile nature of the two UGVs. Moreover, mobile entities' average latency is important especially with missions implementing MIMD software, such as the follow me mission, where there is constant MPI messages exchange for motion coordination. Again latency is better in REAL mode than in SIM mode.

These results suggest that HPRC in its whole extent, even including MIMD software, is fully achievable in missions within line of sight, at small distances from a Wi-Fi router. For future work, Beyond Visual Line of Sight (BVLOS) missions will be studied, including 4G communications in order to observe performance indicators, but current communications technology advancements offer great promises for a future filled with general-purpose ubiquitous supercomputing systems.

V.5.2.5 Scalability

In this section, it is discussed the results of executing the WPA cracking mission with the 512-words dictionary, in SIM mode with the following ubiquitous supercomputing infrastructures/cases:

1. System controller and 1 UGV
2. System controller and 2 UGVs
3. System controller, 2 UGVs and 1 UAV (*hackdrone1*)

The scalability tests were performed using the same Raspberry PIs that in real mode, with an extra Raspberry Pi, all running TAC and ArduPilot SITL, specifically the APMrover2 (UGVs) and ArduCopter (UAV) firmware and the two Wi-Fi antennas per Raspberry Pi. Figure V-35 shows average orders delay.

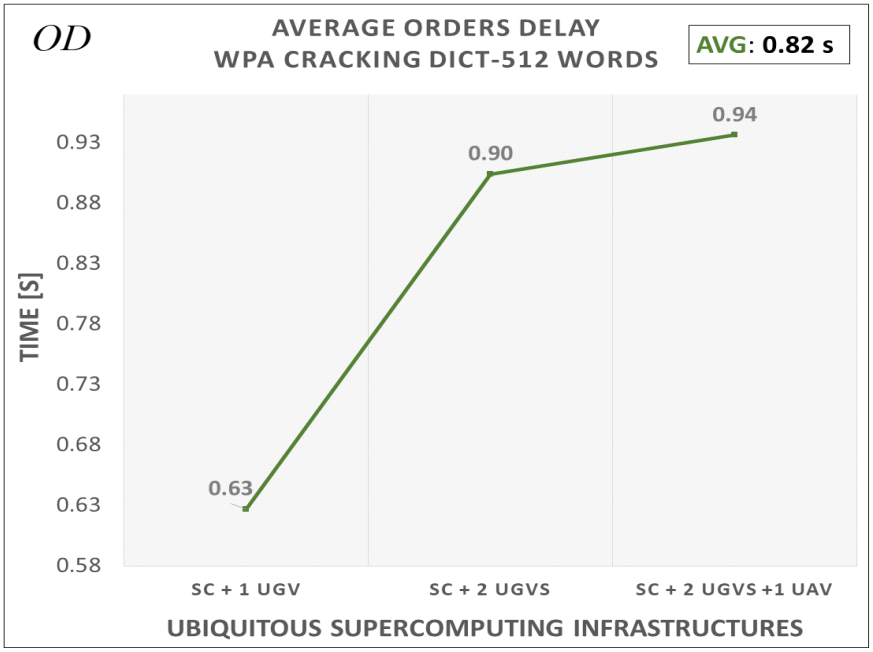


Figure V-35: *Scalability tests - orders delays*

TAC's scalability, regarding orders acknowledgment, is demonstrated as it can be observed with the slight time increasing of 0.04 seconds between case 2 (SC+2 UGVs) and 3 (SC+2 UGVs+1 UAV). Furthermore, the total average of 0.82 seconds falls within the same time frame as the averages of 1.09 seconds in REAL mode and 1.18 seconds in SIM mode for the three missions, as presented in section V.5.2.1.

Table V-11: *Scalability tests: CPU load and RAM usage*

Entity Type	Average CPU load [%]	Average RAM usage [GB]
System controller	42.92	2.03
Mobile entities	66.90	0.17

Moreover, as it can be observed in Table V-11, TAC's scalability is demonstrated in terms of CPU load and RAM usage when increasing the quantity of entities, i.e. no major spikes when increasing the system's size. In fact, the system controller CPU load is lower (42.92%) than the 46.48% and 56.42% in SIM and REAL mode respectively, as discussed in section V.5.2.3. This needs further testing for confirmation but results are very positive. However, the average 2.03 GB of RAM usage is higher than the 1.58 GB and 1.95 GB in SIM and REAL mode respectively. Again, this needs further study and it is left for future work, although the system controller's hardware features are not an issue, provided that it is set in a non-robotic entity, since nowadays it would be quite achievable to use a rather inexpensive computer with high computing power.

Regarding the mobile entities, the average CPU load of 66.90% is close with the average 68.44%, as found in the SIM mode tests (section V.5.2.3). In addition, the average 0.17 GB of RAM usage is exactly the same as the previous results in the same section. These findings are positive given current limited hardware features with companion computers and considering the fact that non system controller entities' CPU load and RAM usage are not affected by the system's size. Furthermore, computing boards such as the NVIDIA Jetson AGX Xavier (NVIDIA, n.d.c) would not have any issue with The ARCHADE, even potentially acting as the system controller entity and the master HPRC node.

Scalability is an important feature when discussing supercomputing systems and while unfortunately, during the development of this Ph.D thesis, was not possible to have access to larger quantities of entities, the results presented in here are promising.

V.5.2.6 WPA cracking software performance

WPA cracking is a complex computational task that benefits from the use of supercomputing in its traditional approach, i.e. as a performance tool. However, for the studied mission, its complexity has been decreased by the use of patterns, accounting for a known password and correspondingly, computing times are very small in total. This is done taken into account energy considerations, i.e. for a long complicated password, WPA cracking could take hours, beyond the robots' energy capacity. The *wpaCrackingRPI* software is composed of the steps/attacks: 1. Sniffing attack, 2. Client deauthentication attack and 3. WPA cracking attack (SIMD) (see Figure V-31), where attacks $1-3$ individual computing times amount to the complete attack time.

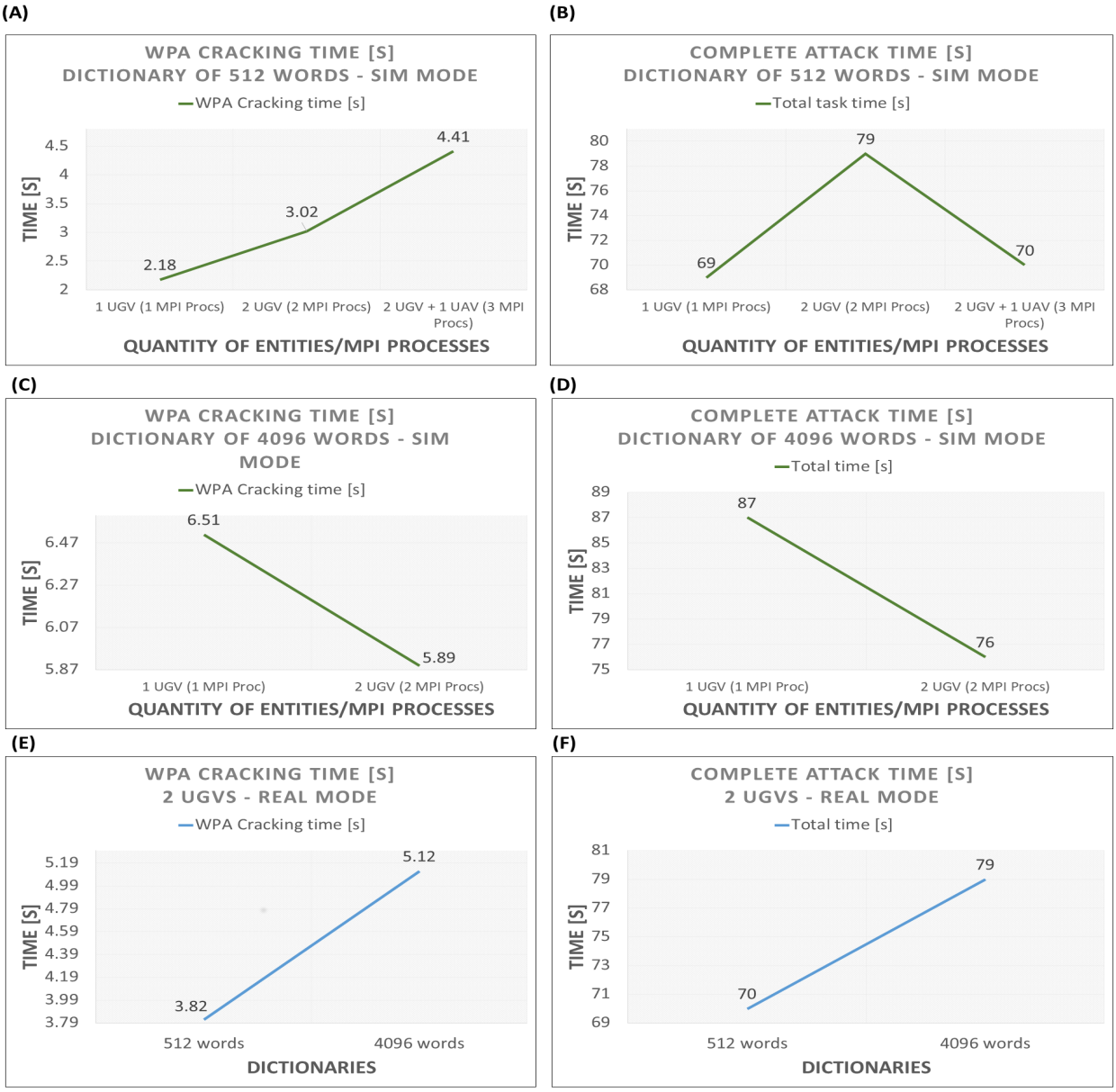


Figure V-36: *WPA cracking software performance*

Figure V-36 shows computing times for the following test cases:

1. WPA cracking with 512-words dictionary in SIM mode with the three scalability cases (section V.5.2.5). As mentioned before, one MPI process was executed per mobile entity, i.e. having three mobile entities, three MPI processes were executed - Figure V-36 A and B.
2. WPA cracking with 4096-words dictionary in SIM mode with 1 and 2 MPI processes - Figure V-36 C and D.
3. WPA cracking with 512 and 4096-words dictionaries in REAL mode with the 2 UGVs - Figure V-36 E and F.

While the difference in WPA cracking times (attack 3), for the tree scalability cases, is very small ($\approx$ 2 s), as shown in Figure V-36-A, the time increase, as the MPI processes quantity increases, might evidence the communications cost impacting negatively the software performance. However, there is a time reduction (9 s) for the complete attack time when using 3 over 2 MPI processes (Figure V-36-B). Nevertheless, attacks 1 and 2 are executed only by a single MPI process, which suggests simply better network conditions at the specific execution time of the 3 MPI processes attack. This is a rather common occurrence in traditional supercomputing, where having small data sets, (512 words) does not benefit from parallelism.

Regarding the second test case (Figure V-36 C and D), the WPA cracking time and the complete attack time decrease as the quantity of MPI processes increase, satisfying traditional supercomputing's main objective, *computing efficiency*. Furthermore, the test case number 3 results (Figure V-36 E and F), show that for a data ratio of 8 (4096/512), WPA cracking time increases with a ratio of 1.34 (5.12/3.82) and the complete attack time with a ratio of 1.13 (79/70), a good indication of computing efficiency and scalability.

These results demonstrate that ubiquitous supercomputing, as rethought by this Ph.D thesis, can provide in fact its traditional approach. Next chapter discusses conclusions and future work.

Life, as we carelessly conceive it, is nothing but an emergent property, something that does not belong to our heart nor to our brain, or to any individual classification. Aren't we alive because all of us are alive? Has the universe found itself trapped into a fractal? Or did I wish it to be like that? Supercomputing is more than performance, performance is just another emergent property

— Leonardo CF

VI

Concluding remarks

Supercomputing has been found very useful in the last decades. We can do all kind of things nowadays, things that were impossible just fifty years ago. Imagine Einstein connecting to the Barcelona Supercomputing Center and requesting some nodes to prove its gravitational waves theory or Tesla fighting Edison yet again to run their "current" simulations. Would Newton or da Vinci be even greater? Imagine the genius with millions of computing cores at his/her fingertips.

Supercomputing is everywhere, from the little widget telling you that today will be a sunny day, up to that new medicine annihilating that old disease and passing by those discoveries reaching the depths of the universe, our history, our still being told story and that of the universe itself. Only one can compete its scope, if it manages to find stability. Quantum computing, but yet again human curiosity will eventually require Quantum Supercomputing.

This Ph.D thesis aimed at taming the monster, to find a way to bring its armor to the small places, those not yet capable of 4.1 GHz or $2,397,824$ computing cores, those not requiring $9,783.00$ kW (TOP500, n.d.), those at the Edge, those left behind, those previously thought for a simple task, a single purpose. Yet, not only about bringing power to the little ones, not only its armor, but its sword, the one elegantly hidden behind its impressive cavalry.

VI.1 Summary of contributions

Supercomputing has been lightly flirted with by roboticists, as section II.1 has shown. Previous works have suggested the use of parallel software libraries, such as CUDA or MPI for computationally demanding robotic tasks. In such works, supercomputing has always been understood as a set of tools and infrastructures for speeding up tasks that are too slow when running on single computing units.

However, supercomputing is more than that, more that simply placing a larger set of computing units together and consequently solving complex problems faster. As a first attempt to rethink what supercomputing actually means, the concept, potentially new computer science field of *High Performance Robotic Computing (HPRC)* was introduced by this Ph.D thesis, in which a set of robots, still capable of higher performance that the one found in a single one, can scale, be used for different purposes and even execute tasks based on MPI, the most standard High Performance Computing (HPC) technology, but for purposes not necessarily requiring high computing power, e.g. applications such as V.2, V.3 in chapter V.

Moreover, for certain applications, the amount of total computing power that can be embedded nowadays in a multi-robot system might not be enough, therefore integration between traditional HPC and HPRC results advantageous. However, in order to control advancements regarding AI and automation, human hierarchy shall be maintained. In addition, hierarchy amongst the entities belonging to a single system, must be distributed accordingly in order to provide scalability, stability, resilience, etc. With this in mind, this Ph.D thesis defined ubiquitous supercomputing as the hierarchical joining of HPC, HPRC, other devices that can be included in single systems, i.e. CLD, and ultimately humans, all with a twofold objective. First, to create systems, made of independent entities and capable to act as a single cohesive unit and second to contribute to the ubiquity of supercomputing and robotics. The following list summarizes the contributions of this Ph.D thesis:

- A *comprehensive review of previous attempts to join supercomputing and robotics* was presented in chapter II section II.1. Most contributions, found in the literature, targeted single-robot systems, by augmenting its computing power via embedded computers running traditional operating systems and standard HPC technologies such as MPI and CUDA, or via interfaces with remote computing facilities (e.g. cloud, etc). In addition, few works proposed the idea of transforming a set of robots into a HPC cluster. However, a lack of a formal definition integrating HPC and robotics and more importantly, detailed strategies to adapt it within multi-robot systems were still missing. Furthermore, previous research endeavors focused solely upon the performance feature of supercomputing infrastructures, something that in fact can be useful in robotic settings but that is not always necessary. Therefore, in order to exploit supercomputing in its full extent, High Performance Robotic Computing was proposed by this Ph.D thesis.

- *High Performance Robotic Computing* (chapter III section III.1) aims at transforming a set of independent robots into a supercomputing infrastructure capable of doing anything and not requiring external computing power. In this sense, HPRC can be thought as *HPC Edge or Fog computing*, as long its objective is understood beyond solely computing performance. This Ph.D thesis provided strategies to adapt each of the HPC software layers utilization to multi-robot systems, taken into consideration the nuances of mobile entities and current limitations. Two types of *multi-user/multi-purpose* clusters: *HPRC cluster* (Figure III-2) and *HARC* (Definition 6) were defined and to facilitate their installation and configuration, the HPC-ROS package (section III.1.1) was developed. In addition, to demonstrate that standard HPC software can be used for non-demanding computing tasks and still provide scalability, user-transparency, etc., swarming motion MPI software based on the Vicsek model

(Vicsek *et al.* , 1995) was developed and tested in simulation mode (chapter V sections V.2 and V.3).

- *Ubiquitous supercomputing* defined as the union of HPC, HPRC, CLD and humans with the objective of creating systems composed of distributed entities but acting as a single cohesive unit capable of executing any type of mission (chapter III definition 1). For practicality, an ontology describing concepts such as systems, entities, roles, nodes, groups, links, interfaces, etc, was proposed in the same chapter. Furthermore, the concept of *General-Purpose Computing Mission GPCM* was described in section III.3, including the ideas behind tasks, events, actions, etc. All of these novel concepts were defined with the objective of facilitating the creation of any type of mission executed with ubiquitous supercomputing systems.

- The *ubiquitous supercomputing language UbiSL* (chapter III section III.4) was proposed to serve as a practical implementation of the ubiquitous supercomputing ontology and general-purpose computing missions, including all its relevant concepts (chapter III). The UbiSL provides flexibility for the creation of systems composed of any quantity and type of entities embedded with any quantity of nodes, forming any quantity and types of groups and being used for any type of mission, where each task is linked with an individual software, therefore effectively separating the ubiquitous supercomputing system from the missions carried out by it. In practical terms, the UbiSL is a set of templates that can be used to build ubiquitous supercomputing systems and general-purpose computing missions.

- *Hierarchy and ubiquitous supercomputing*. Hierarchical systems have often been found in nature and all kind of human activities. A ubiquitous supercomputing system is a hierarchical network, representing the capacity that entities hold to command orders over other entities, as defined by this Ph.D thesis. Based on the ideas of hierarchy, three *automation modes* were proposed. In addition, it was found that networks with a Global Reaching Centrality (hierarchy) higher than 0.5 are more stable than those less hierarchical. The three automation modes were designed to scale towards any quantity of entities, while still maintaining stability.

- *A literature review regarding ubiquitous robotics* was presented in chapter II, section II.2. It was found that Robot Operating System (ROS) is quickly becoming a standard robotics technology, one that can be gracefully integrated in ubiquitous supercomputing infrastructures at their uppermost layer, the applications layer, similar as many other technologies such as OpenCV, OpenCL, MPI, CUDA, etc. Additionally, the guidelines for new ubiquitous robotics frameworks in Jiménez-González *et al.* (2013) were used to design a ubiquitous supercomputing framework and middleware.

- *The ARCHADE - TAC* (chapter IV), a loosely-coupled component-based, fully general-purpose ubiquitous supercomputing framework and middleware, written in Python, designed for the deployment, implementation and operation of ubiquitous supercomputing systems. The ARCHADE has been tested with simulated and real systems as discussed in chapter V sub section V.5.2. TAC has obtained very good performance indicators, i.e. orders delay, CPU load, RAM usage, etc., and it can be concluded that it can effectively transform a set of independent entities into a scalable, computing-efficient, centralized/distributed, etc., supercomputing infrastructure. TAC is composed of 5 components: *Application Programming Interface (API)*, *Framework*, *Middleware*, *SimPlat* and *PLUS*, to be listed as individual contributions.

- *The ARCHADE API* (chapter IV section IV.1) was used to developed the other TAC's four components and it includes libraries for ubiquitous supercomputing infrastructures management and control, operating system directives, hierarchy, etc. Furthermore, the most

important contribution of the TAC API is that it can be used to build new services, components, mission software, etc. Moreover, each of the other components is packed with a corresponding API.

- *The ARCHADE Framework* (chapter IV section IV.2), which consists of a set of classes and templates that allow the description and implementation of a ubiquitous supercomputing system. Moreover, the framework provides services for deployment of HPC or HPRC settings e.g. the HPC-ROS package providing *standardization*, testing, etc. The single most important feature of the framework is to provide a mechanism to create all kind of classes representing vehicles, robots, *things* that can be integrated transparently into ubiquitous supercomputing systems, providing *scalability*, *heterogeneity*, *flexibility*, etc. Furthermore, TAC framework implements the UbiSL and provides extra templates for mission software. Such templates make use of the API and in some cases MPI to create all kind of *general-purpose computing* software easily integrated with The ARCHADE. Moreover, these templates provide *computing efficiency/performance*, i.e. by facilitating MPI utilization.

- *The ARCHADE Middleware* (chapter IV section IV.3) separates the ubiquitous supercomputing infrastructure from the mission carried by it and provides a set of services between the two layers such as *communications*, *initialization*, *live*, *matching*, *monitoring*, *statistics* and *synchronization*. All services are component-based. Each service is portrayed as a separated contribution.

  - *The ARCHADE Middleware communications service* most important function is to provide a mechanism for orders (hierarchy) and events exchange amongst entities to facilitate the automatic execution of a mission. Orders are shared accordingly to the automation modes, therefore this service also includes the *hydra* sub service, which can be used by the system operator to control the entire system. In addition, orders are encrypted and only accepted from higher hierarchy entities and via pre-shared keys. This service ultimate goal is to provide *hierarchy* and *security*.
  - *The ARCHADE Middleware initialization service* most important function is to interpret UbiSL templates to create the ubiquitous supercomputing system in execution time and to effectively mapped it to the ubiquitous supercomputing infrastructure.
  - *The ARCHADE Middleware live service* most important function is to automatically control the ubiquitous supercomputing system and the execution of the mission, including the automatic control of the vehicles. Two main agents are part of this service, the *system controller* and the *entity controller*. The agents hold different hierarchies, but both control its corresponding system and entity. See ubiquitous supercomputing lemma III.2. This service provides *automation*, *cooperation* and *user-transparency*.
  - *The ARCHADE Middleware matching service* most important feature is to assign mission tasks to entities by using the UbiSL templates. In this sense, the service replaces and enhances the scheduling feature of the batch system.
  - *The ARCHADE Middleware monitoring service* holds different important functions, such as mission monitoring, individual nodes indicators, e.g CPU load, RAM usage, network conditions, etc, replacing and enhancing the resources monitoring feature in the batch system.
  - *The ARCHADE Middleware statistics service* main function is to compute mission and system's statistics such as hierarchy, CPU load, etc. The service interacts with the monitoring service results to compute said statistics.
  - *The ARCHADE Middleware synchronization service* holds a very important feature, to maintain a *single-image system*, i.e. to synchronize TAC and mission data among the entire system depending of its necessity. Furthermore, it provides *resilience* by acting

from the background at all times. Even in periods of disconnection, the service is active and will synchronize all data once reconnection occurs. This service provides *centralization/distribution* and *user-transparency*.

- *The ARCHADE SimPlat* (chapter IV section IV.4) provides two main mechanisms. Easy transition from experimentation to real-world missions and a *general-purpose simulator template*.
  - *hprccoopflying* (chapter V section V.3), a TAC-based simulator implementing the concept of *HPRC cluster of aircraft*, which currently includes swarming motion but can be enhanced to include other algorithms.
- *The ARCHADE PLUS* (chapter IV section IV.5) is a GPCM (General-Purpose Computing missions) API, i.e. it can be used to build mission task software. It currently includes two modules, to be listed as individual contributions.
  - *The ARCHADE PLUS Motion module* includes *swarm* and *follow me* motion software as portrayed in chapter V sections V.2, V.3 and V.5. To use such software, entities must be put in piloted mode in order to overthrown mobility control at TAC's middleware live service.
  - *The ARCHADE PLUS Security module* includes several libraries for ethical hacking, as discussed in chapter V section V.5.

The single most important contribution of this Ph.D thesis has been to provide a philosophy and a technology (The ARCHADE) that can transform a set of independent entities, e.g. servers, supercomputers, robots, computing-less devices, people, etc., into an *enhanced supercomputer*, a *collective centralized and distributed intelligence*, capable of doing anything by exploiting *all supercomputing features*.

VI.2 Future research

Several research fronts spawn from the results and contributions of this Ph.D thesis, as it was discussed throughout all the chapters, especially chapter III. However, the following main research lines are considered:

- *Cloud computing*: The ARCHADE currently supports the use of interfaces for interaction with external computing resources, as it was demonstrated in the proof of concept, the *Tigers vs. Hunters system* (chapter V section V.4). However, in order to provide the simulation platform with the capacity of testing simulated systems made up of hundreds of entities, the creation of a *Cloud computing interface* is in TAC's roadmap. Such interface will allow the transparent deployment and provisioning of TAC-enabled virtual machines and or containers via Docker (Docker, n.d.). The interface is currently in design phase.
- *Scalability*: The benchmark carried out in real mode was done with a maximum of three entities (chapter V Figure V-3), the HPRC-Rovers cluster. While the results of the benchmark were very positive, future work will include further testing with larger quantities of entities, in order to evaluate TAC's performance indicators. Such testing will be done when the cloud computing interface is finished and tested. In addition, a deeper analysis of the differences between SIM and REAL mode is foreseen in TAC's roadmap.
- *Adaptability*: In order to provide dynamic system reconfiguration, future work includes the development of a *Service Level Agreement (SLA) service*, part of the next version of TAC's

middleware. The SLA service will allow dynamic integration of entities, i.e. during mission execution and furthermore live mission changing.

- *Resilience*: TAC currently supports resilience via the middleware synchronization service (chapter IV section IV.3). However, to further provide resilience mechanisms, TAC's roadmap includes the development of a live redistribution tasks mechanism.
- *Automation*: The three automation modes (chapter III section III.7) provide mathematical stability that requires further testing including tens and hundreds of entities, first in simulation mode and following with real entities. Furthermore, *mixed automation modes*, e.g. a single pilot controlling a group of entities, multiple master entities, multiple system operators, low-LRC entities capable of commanding orders over groups of entities, (modified hierarchy networks), etc., testing is part of the future work.
- *Real-time systems*: In order to support real-time and sensitive missions, TAC's roadmap includes the integration of the Data Distributed Service (DDS) (Pardo-Castellote, 2005) within the TAC's middleware communications service. Furthermore, testing with RTOS (Real-Time Operating Systems) such as Nutt (2014), e.g. deployment of HPC software layers is part of the future work.
- *Heterogeneity*: Based on the framework core layers, TAC's roadmap includes support for Dronecode (Dronecode, n.d.a), DJI (DJI, n.d.a), ROS, etc.

The main goals of this Ph.D thesis and its future work is to serve as the foundation layer for the creation of a new computer science field *High Performance Robotic Computing*, the rise of ubiquitous supercomputing as defined in here and The ARCHADE as a standard ubiquitous robotics technology that can successfully being used in all kind of applications, related to the Internet of Things, robotics, etc., and therefore truly bringing supercomputing everywhere.

Bibliography

3DR. *DroneKit companion computers*. http://python.dronekit.io/develop/companion-computers.html. [Online; accessed February-2019]. 74

3DR. *DroneKit. Developer tools for drones*. http://dronekit.io/. [Online; accessed February-2019]. 26, 27, 58, 74

3DR. *DroneKit-SITL*. https://github.com/dronekit/dronekit-sitl. [Online; accessed September-2019]. 61

ABHISHEK, SHARMA. 2014. *Drone Data Adds a New Horizon for Big Data Analytics*. http://www.infoq.com/news/2014/09/drone-data-big-data-analytics. [Online; published September-2014, accessed January-2019]. 3

ADAPTIVE-COMPUTING. *Adaptive Computing Moab*. http://www.adaptivecomputing.com/moab-hpc-basic-edition/. [Online; accessed January-2019]. 11

ADAPTIVE-COMPUTING. *TORQUE Resource Manager*. http://www.adaptivecomputing.com/products/torque/. [Online; accessed January-2019]. 5, 11, 30, 32

AIRCRACK-NG. *Aircrack-ng suite*. https://www.aircrack-ng.org/. [Online; accessed February-2019]. 102

ALBERT, RÉKA, JEONG, HAWOONG, & BARABÁSI, ALBERT-LÁSZLÓ. 2000. Error and attack tolerance of complex networks. *nature*, **406**(6794), 378. 43

ALFA. *Alfa AWUS036NEH Wi-Fi antenna*. https://www.alfa.net.my/products/Alfa-AWUS036NEH-802.11n-WIRELESS-N-USB-Wi-Fi---Antenna/11. [Online; accessed February-2019]. 98

AMAZON. *AWS High Performance Computing (HPC)*. https://aws.amazon.com/hpc/. [Online; accessed February-2016]. 4, 17

AMDAHL, GENE M. 1967. Validity of the single processor approach to achieving large scale computing capabilities. *Pages 483–485 of: Proceedings of the april 18-20, 1967, spring joint computer conference*. ACM. 12

ANDERSON, DAVID P, COBB, JEFF, KORPELA, ERIC, LEBOFSKY, MATT, & WERTHIMER, DAN. 2002. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, **45**(11), 56–61. 12

ANDO, NORIAKI, SUEHIRO, TAKASHI, KITAGAKI, KOSEI, KOTOKU, TETSUO, & YOON, WOO-KEUN. 2005. RT-middleware: distributed component middleware for RT (robot technology). *Pages 3933–3938 of: IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 17

ANSEL, JASON, ARYA, KAPIL, & COOPERMAN, GENE. 2009. DMTCP: Transparent checkpointing for cluster computations and the desktop. *Pages 1–12 of: International Symposium on Parallel & Distributed Processing*. IEEE. 6

ARAIZA-ILLAN, DEJANIRA, & EDER, KERSTIN. 2019. Safe and Trustworthy Human-Robot Interaction. *Humanoid Robotics: A Reference*, 2397–2419. 19

ARDUPILOT. *ArduPilot*. http://ardupilot.org/. [Online; accessed January-2019]. 14

ARDUPILOT. *MAVProxy*. http://ardupilot.github.io/MAVProxy/html/index.html. [Online; accessed February-2019]. 92

ARDUPILOT. *NVidia TX1 as a Companion Computer*. http://ardupilot.org/dev/docs/companion-computer-nvidia-tx1.html. [Online; accessed January-2019]. 14

ARDUPILOT. *NVidia TX2 as a Companion Computer*. http://ardupilot.org/dev/docs/companion-computer-nvidia-tx2.html. [Online; accessed January-2019]. 15

ARDUPILOT. *Pixhawk 1*. http://ardupilot.org/copter/docs/common-pixhawk-overview.html. [Online; accessed January-2019]. 14, 69, 98

ARDUPILOT. *SITL Simulator (Software in the Loop)*. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. [Online; accessed February-2019]. 27, 61, 74

ASUS. *ASUS Tinker board*. https://www.asus.com/us/Single-Board-Computer/Tinker-Board/. [Online; accessed January-2019]. 13

BARABÁSI, ALBERT-LÁSZLÓ, & BONABEAU, ERIC. 2003. Scale-free networks. *Scientific american*, **288**(5), 60–69. 34

BARDRAM, JAKOB, & FRIDAY, ADRIAN. 2016. Ubiquitous computing systems. *Pages 51–108 of: Ubiquitous computing fundamentals*. Chapman and Hall/CRC. 16

BENAVIDEZ, PATRICK, MUPPIDI, MOHAN, RAD, PAUL, PREVOST, JOHN J, JAMSHIDI, MO, & BROWN, LUTCHER. 2015. Cloud-based realtime robotic visual SLAM. *Pages 773–777 of: Proceedings of the annual systems conference (SysCon)*. IEEE. 3, 15

BESERRA, DAVID, PINHEIRO, MANUELE KIRSCH, SOUVEYET, CARINE, STEFFENEL, LUIZ ANGELO, & MORENO, EDWARD DAVID. 2017. Performance evaluation of OS-level virtualization solutions for HPC purposes on SOC-based systems. *Pages 363–370 of: 31st international conference on advanced information networking and applications (AINA)*. IEEE. 4

BORTHAKUR, DHRUBA. 2008. HDFS architecture guide. *Hadoop apache project*, **53**, 1–13. 11

BOWER, GEOFFREY, FLANZER, TRISTAN, & KROO, ILAN. 2009. Formation geometries and route optimization for commercial formation flight. *Page 3615 of: 27th AIAA Applied Aerodynamics Conference*. 81, 84

BRADSKI, GARY, & KAEHLER, ADRIAN. 2000. OpenCV. *Dr. dobb's journal of software tools*, **3**. 14

BROXVALL, MATHIAS, SEO, BEOM-SU, & KWON, WOOYOUNG. 2007. The peis kernel: A middleware for ubiquitous robotics. *Pages 212–218 of: IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications*. 17

BRUYNINCKX, HERMAN. 2001. Open robot control software: the OROCOS project. *Pages 2523–2528 of: Proceedings of the international conference on robotics and automation (ICRA)*, vol. 3. IEEE. 18

CARBONE, GIUSEPPE, CECCARELLI, MARCO, & PISLA, DOINA. 2018. *New Trends in Medical and Service Robotics: Advances in Theory and Practice*. Vol. 65. Springer. 17

CARMEN. *Carmen. Robot navigation toolkit*. http://carmen.sourceforge.net/. [Online; accessed February-2019]. 18

CASTELLANO, FRANCESCO. *Commercial Drones Are Revolutionizing Business Operations*. https://www.toptal.com/finance/market-research-analysts/drone-market. [Online; accessed January-2019]. 3

CHABOWSKI, MEIKE. 2016. *How HPC Impacts Our Lives I: Space, Weather, and More*. https://www.suse.com/c/hpc-impacts-lives/. [Online; published May-2016, accessed January-2019]. 5

CHANG, YEONG-HWA, CHUNG, PING-LUN, & LIN, HUNG-WEI. 2018. Deep learning for object identification in ROS-based mobile robots. *Pages 66–69 of: International Conference on Applied System Invention (ICASI)*. IEEE. 18

CHEN, YINONG, DU, ZHIHUI, & GARCÍA-ACOSTA, MARCOS. 2010. Robot as a service in cloud computing. *Pages 151–158 of: Fifth International Symposium on Service Oriented System Engineering (SOSE)*. IEEE. 3

CHENG, LIANG, & MARSIC, IVAN. 2000. Wireless awareness for multimedia applications. *Pages 1376–1382 of: Proceedings of the 2000 International Conference on Communication Technology WCC-ICCT*, vol. 2. IEEE. 15

CHENG, LIANG, WANCHOO, AJAY, & MARSIC, IVAN. 2000. Hybrid cluster computing with mobile objects. *Pages 909–914 of: Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, vol. 2. IEEE. 15

CHENG, YANG, MAIMONE, MARK W, & MATTHIES, LARRY. 2006. Visual odometry on the Mars exploration rovers. *IEEE Robotics and Automation magazine*, **13**(2), 54. 17

CHIBANI, ABDELGHANI, AMIRAT, YACINE, MOHAMMED, SAMER, MATSON, ERIC, HAGITA, NORIHIRO, & BARRETO, MARCOS. 2013. Ubiquitous robotics: Recent challenges and future trends. *Robotics and Autonomous Systems*, **61**(11), 1162–1172. 18

CHOI, HYUNWOONG, GEEVES, MITCHELL, ALSALAM, BILAL, & GONZALEZ, FELIPE. 2016. Open source computer-vision based guidance system for UAVs on-board decision making. *Pages 1–5 of: IEEE aerospace conference*. IEEE. 14

CHU, SLO-LI, & HSIAO, CHIH-CHIEH. 2010. OpenCL: Make ubiquitous supercomputing possible. *Pages 556–561 of: 2010 12th International Conference on High Performance Computing and Communications (HPCC)*. IEEE. 17, 20

CISCO. 2018. *Cisco Global Cloud Index: Forecast and Methodology, 2016 - 2021 White Paper*. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html. [Online; accessed January-2019]. 2

COCCHIONI, FRANCESCO, FRONTONI, EMANUELE, IPPOLITI, GIANLUCA, LONGHI, SAURO, MANCINI, ADRIANO, & ZINGARETTI, PRIMO. 2016. Visual based landing for an unmanned quadrotor. *Journal of intelligent & robotic systems*, **84**(1-4), 511–528. 14

COLOMBO, ROBERTO, & SANGUINETI, VITTORIO. 2018. *Rehabilitation Robotics: Technology and Application*. Academic Press. 17

COX, SIMON J., COX, JAMES T., BOARDMAN, RICHARD P., JOHNSTON, STEVEN J., SCOTT, MARK, & O'BRIEN, NEIL S. 2014. Iridis-pi: a low-cost, compact demonstration cluster. *Cluster computing*, **17**(2), 349–358. 4

ÇÜRÜKLÜ, BARAN, DODIG-CRNKOVIC, GORDANA, & AKAN, BATU. 2010. Towards industrial robots with human-like moral responsibilities. *Pages 85–86 of: 5th International Conference on Human-Robot Interaction (HRI)*. ACM/IEEE. 17

DAGUM, LEONARDO, & MENON, RAMESH. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, **5**(1), 46–55. 5

DALCIN, LISANDRO. *MPI for Python*. https://mpi4py.readthedocs.io/en/stable/. [Online; accessed February-2019]. 75

DALMAU, RAMON, & PRATS, XAVIER. 2015. Fuel and time savings by flying continuous cruise climbs: Estimating the benefit pools for maximum range operations. *Transportation Research Part D: Transport and Environment*, **35**, 62–71. 84

DARYANAVARD, H, & HARIFI, A. 2018. Implementing face detection system on UAV using Raspberry Pi platform. *Pages 1720–1723 of: Iranian Conference on Electrical Engineering (ICEE)*. IEEE. 14

DEGRAVE, JONAS, HERMANS, MICHIEL, DAMBRE, JONI, *et al.* . 2019. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, **13**. 19

DEMARCO, KEVIN, SQUIRES, ERIC, DAY, MICHAEL, & PIPPIN, CHARLES. 2019. Simulating collaborative robots in a massive multi-agent game environment (SCRIMMAGE). *Pages 283–297 of: Distributed Autonomous Robotic Systems*. Springer. 19

DIANKOV, ROSEN, & KUFFNER, JAMES. 2008. OpenRAVE: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, **79**. 18

DJI. *DJI developer*. https://developer.dji.com/. [Online; accessed February-2019]. 120

DJI. *DJI Manifold*. https://www.dji.com/manifold. [Online; accessed January-2019]. 16

DO, HYUN MIN, KIM, BONG KEUN, KIM, YONG-SHIK, LEE, JAE HOON, OHARA, KENICHI, SUGAWARA, TAKAYUKI, TOMIZAWA, TETSUO, LIANG, XUEFENG, TANIKAWA, TAMIO, & OHBA, KOHTARO. 2007. Development of simulation framework for ubiquitous robots using RT-middleware. *Pages 2483–2486 of: International Conference on Control, Automation and Systems ICCAS*. IEEE. 17

DOCKER. *Docker*. https://www.docker.com/. [Online; accessed February-2019]. 19, 119

DOMÍNGUEZ, RAÙL, GOVINDARAJ, SHASHANK, GANCET, JEREMI, POST, MARK, MICHALEC, ROMAIN, OUMER, NASSIR, WEHBE, BILAL, BIANCO, ALESSANDRO, FABISCH, ALEXANDER, LACROIX, SIMON, *et al.* . 2018. A common data fusion framework for space robotics: architecture and data fusion methods. *In: International Symposium on Artificial Intelligence, Robotics and Automation in Space Symposia*. 17

DONGARRA, JACK. *Frequently Asked Questions on the Linpack Benchmark and Top500*. http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html. [Online; accessed February-2019]. 71

DONGARRA, JACK. *HPL Algorithm*. http://www.netlib.org/benchmark/hpl/algorithm.html. [Online; accessed February-2019]. 72

DONGARRA, JACK, STROHMAIER, ERICH, & SIMON, HORST. *Supercomputers Top 500*. http://www.top500.org/. [Online; accessed February-2016]. 5, 20, 70

DRONECODE. *Dronecode SDK Python*. https://github.com/Dronecode/DronecodeSDK-Python. [Online; accessed February-2019]. 120

DRONECODE. *Pixhawk 2.1. The cube*. https://docs.px4.io/en/flight_controller/pixhawk-2.html. [Online; accessed February-2019]. 69, 98

DRONECODE. *PX4 autopilot*. https://px4.io/. [Online; accessed February-2019]. 19

DURAN, ALEJANDRO, AYGUADÉ, EDUARD, BADIA, ROSA M, LABARTA, JESÚS, MARTINELL, LUIS, MARTORELL, XAVIER, & PLANAS, JUDIT. 2011. OmpSs: a proposal for programming heterogeneous multi-core architectures. *Parallel processing letters*, **21**(02), 173–193. 5

DURANGO, GJ, LAWSON, C, & SHAHNEH, ABOLGHASEM ZARE. 2016. Formation flight investigation for highly efficient future civil transport aircraft. *The Aeronautical Journal*, **120**(1229), 1081–1100. 81, 84

EUROCONTROL. *DDR2 quick reference guide*. https://www.eurocontrol.int/sites/default/files/content/documents/nm/airspace/DDR2_Quick%20Reference%20Guide_%2020141205.pdf. [Online; accessed February-2019]. 83

FLORES-ABAD, ANGEL, MA, OU, PHAM, KHANH, & ULRICH, STEVE. 2014. A review of space robotics technologies for on-orbit servicing. *Progress in Aerospace Sciences*, **68**, 1–26. 17

FLYNN, M. 1972. Some computer organizations and their effectiveness. *IEEE transactions on computers*, **C-21**(9), 948–960. 12, 31, 100, 104

FOSTER, IAN, & TUECKE, STEVEN. 1996. Enabling technologies for web-based ubiquitous supercomputing. *Pages 112–119 of: Proceedings of 5th International Symposium on High Performance Distributed Computing*. IEEE. 17, 20

GABRIEL, EDGAR, FAGG, GRAHAM E, BOSILCA, GEORGE, ANGSKUN, THARA, DONGARRA, JACK J, SQUYRES, JEFFREY M, SAHAY, VISHAL, KAMBADUR, PRABHANJAN, BARRETT, BRIAN, LUMSDAINE, ANDREW, *et al.* . 2004. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Pages 97–104 of: European parallel virtual machine/message passing interface users' group meeting*. Springer. 32, 72

GARCIA, RICHARD, & BARNES, LAURA. 2009. Multi-UAV simulator utilizing X-Plane. *Pages 393–406 of: Selected papers from the 2nd International Symposium on UAVs*. Springer. 20

GARLAND, MICHAEL, LE GRAND, SCOTT, NICKOLLS, JOHN, ANDERSON, JOSHUA, HARDWICK, JIM, MORTON, SCOTT, PHILLIPS, EVERETT, ZHANG, YAO, & VOLKOV, VASILY. 2008. Parallel computing experiences with CUDA. *IEEE micro*, 13–27. 4, 12

GAZEBO. *Gzweb Web client for Gazebo*. http://gazebosim.org/gzweb.html. [Online; accessed February-2019]. 19

GENTZSCH, WOLFGANG. 2001. Sun grid engine: Towards creating a compute power grid. *Pages 35–36 of: Proceedings of the First International Symposium on Cluster Computing and the Grid*. IEEE/ACM. 15

GERKEY, BRIAN, VAUGHAN, RICHARD T, & HOWARD, ANDREW. 2003. The player/stage project: Tools for multi-robot and distributed sensor systems. *Pages 317–323 of: Proceedings of the 11th international conference on advanced robotics*, vol. 1. 18

GLUSTERFS. *Glusterfs*. https://docs.gluster.org/en/latest/. [Online; accessed January-2019]. 11

GONZALEZ, RAMON, MAHULEA, CRISTIAN, & KLOETZER, MARIUS. 2015. A Matlab-based interactive simulator for mobile robotics. *Pages 310–315 of: International Conference on Automation Science and Engineering (CASE)*. IEEE. 19

GOOGLE. *Google Earth*. https://www.google.com/earth/. [Online; accessed February-2019]. 99, 100

GRABE, VOKER, RIEDEL, MARTIN, BÜLTHOFF, HEINRICH H, GIORDANO, PAOLO ROBUFFO, & FRANCHI, ANTONIO. 2013. The TeleKyb framework for a modular and extendible ROS-based quadrotor control. *In: European Conference on Mobile Robots (ECMR)*. 18

HAGER, GEORG, & WELLEIN, GERHARD. 2010. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press. 12

HAYAKAWA, HIROKI, AZUMI, TAKUYA, SAKAGUCHI, AKINORI, & USHIO, TOSHIMITSU. 2018. ROS-based support system for supervision of multiple UAVs by a single operator. *Pages 341–342 of: Proceedings of the 9th International Conference on Cyber-Physical Systems*. ACM/IEEE. 18

HIGHTOWER, JEFFREY, & BORRIELLO, GAETANO. 2001. Location systems for ubiquitous computing. *Computer*, **34**(8), 57–66. 16

HOLLAND, OWEN, WOODS, JOHN, DE NARDI, RENZO, & CLARK, ADRIAN. 2005. Beyond swarm intelligence: the UltraSwarm. *Pages 217–224 of: Proceedings of the Swarm Intelligence Symposium SIS*. IEEE. 15, 20

HU, GUOQIANG, TAY, WEE PENG, & WEN, YONGGANG. 2012. Cloud robotics: architecture, challenges and applications. *IEEE network*, **26**(3). 3

HU, JIA, NIU, YIFENG, & WANG, ZHICHAO. 2017. Obstacle avoidance methods for rotor UAVs using RealSense camera. *Pages 7151–7155 of: Chinese Automation Congress (CAC)*. IEEE. 18

HUAWEI. *Huawei HiKey 960*. https://www.96boards.org/product/hikey960/. [Online; accessed January-2019]. 13

IEEE. *IEEE is Fueling the Fourth Industrial Revolution*. https://innovate.ieee.org/innovation-spotlight-ieee-fueling-fourth-industrial-revolution/. [Online; accessed February-2019]. 17

IEEE. 2004. IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements. *IEEE Std 802.11i-2004*, July, 1–190. 101

IJSPEERT, AUKE J. 2014. Biorobotics: Using robots to emulate and investigate agile locomotion. *Science*, **346**(6206), 196–203. 17

INTEL. *INTEL AERO COMPUTE BOARD*. https://software.intel.com/en-us/aero/compute-board. [Online; accessed January-2019]. 13

INTEL. *Intel XEON processors*. https://www.intel.com/content/www/us/en/products/processors/xeon.html. [Online; accessed February-2019]. 26

ITU. *International Telecommunications Union (ITU). Minimum requirements related to technical performance for IMT-2020 radio interface(s)*. https://www.itu.int/md/R15-SG05-C-0040/en. [Online; accessed -2019]. 8, 16, 26, 87, 106

JACKSON, MATTHEW O, & WATTS, ALISON. 2002. The evolution of social and economic networks. *Journal of economic theory*, **106**(2), 265–295. 43

JEON, DONGWOON, KIM, DOO-HYUN, HA, YOUNG-GUK, & TYAN, VLADIMIR. 2016. Image processing acceleration for intelligent unmanned aerial vehicle on mobile GPU. *Soft computing*, **20**(5), 1713–1720. 14

JIANG, JINGQI, ZHANG, XUETAO, YUAN, JING, TANG, KAITAO, & ZHANG, XUEBO. 2018. Extendable Flight System for Commercial UAVs on ROS. *Pages 1–5 of: 37th Chinese Control Conference (CCC)*. IEEE. 18

JIMÉNEZ-GONZÁLEZ, ADRIÁN, MARTINEZ-DE DIOS, JOSE RAMIRO, & OLLERO, ANIBAL. 2013. Testbeds for ubiquitous robotics: A survey. *Robotics and Autonomous Systems*, **61**(12), 1487–1501. 18, 53, 64, 117

KERAS. *Keras: The Python Deep Learning library*. https://keras.io/. [Online; accessed February-2019]. 27

KHAN, AMIN M, UMAR, IBRAHIM, & HA, PHUONG HOAI. 2018. Efficient compute at the edge: Optimizing energy aware data structures for emerging edge hardware. *Pages 314–321 of: International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 4

KIM, JONG-HWAN. 2003. IT-based UbiBot. *The Korea Electronic Times*. 17

KIM, JONG-HWAN, KIM, YONG-DUK, & LEE, KANG-HEE. 2004. The third generation of robotics: Ubiquitous robot. *In: Proceedings of the 2nd International Conference on Autonomous Robots and Agents*. 17

KIM, JONG-HWAN, LEE, KANG-HEE, KIM, YONG-DUK, KUPPUSWAMY, NAVEEN SURESH, & JO, JUN. 2007. Ubiquitous robot: A new paradigm for integrated services. *Pages 2853–2858 of: International Conference on Robotics and Automation*. IEEE. 17

KIZAR, SK NOOR, & SATYANARAYANA, GSR. 2016. Object detection and location estimation using SVS for UAVs. *Pages 920–924 of: International conference on automatic control and dynamic optimization techniques (ICACDOT)*. IEEE. 14

KOENIG, NATHAN P, & HOWARD, ANDREW. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. *Pages 2149–2154 of: IROS*, vol. 4. Citeseer. 19

KÖNIG, MICHAEL D, BATTISTON, STEFANO, NAPOLETANO, MAURO, & SCHWEITZER, FRANK. 2012. The efficiency and stability of r&d networks. *Games and economic behavior*, **75**(2), 694–713. 43

KOPACEK, P. 2016. Development Trends in Robotics. *IFAC-PapersOnLine*, **49**(29), 36 – 41. 17

KREST, SHAWN. 2017. *Drones: Flying Data Machines Digitizing the World from Above*. https://iq.intel.com/drones-flying-data-machines-digitizing-the-world-from-above/. [Online; published September-2017, accessed January-2019]. 3

KRUMM, JOHN. 2016. *Ubiquitous computing fundamentals*. Chapman and Hall/CRC. 16

LAMINAR RESEARCH. *XPlane11*. https://www.x-plane.com/. [Online; accessed February-2019]. 19

LEE, BRAD HYEONG-YUN, MORRISON, JAMES R, & SHARMA, RAJNIKANT. 2017. Multi-UAV control testbed for persistent UAV presence: ROS GPS waypoint tracking package and centralized task allocation capability. *Pages 1742–1750 of: International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 18

LOGIST, KEVIN. 2017. *Drones generate a lot of data*. https://www.dronecommunity.biz/drones-data/. [Online; published May-2017, accessed January-2019]. 3

MARJOVI, ALI, CHOOBDAR, SARVENAZ, & MARQUES, LINO. 2012. Robotic clusters: Multi-robot systems as computer clusters: A topological map merging demonstration. *Robotics and Autonomous Systems*, **60**(9), 1191–1204. 15, 20, 99

MARTINS, HENRIQUE, OAKLEY, IAN, & VENTURA, RODRIGO. 2015. Design and evaluation of a head-mounted display for immersive 3D teleoperation of field robots. *Robotica*, **33**(10), 2166–2185. 17

MA'SUM, M ANWAR, ARROFI, M KHOLID, JATI, GRAFIKA, ARIFIN, FUTUHAL, KURNIAWAN, M NANDA, MURSANTO, PETRUS, & JATMIKO, WISNU. 2013. Simulation of intelligent unmanned aerial vehicle (UAV) for military surveillance. *Pages 161–166 of: International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE. 19

MAVLINK. *Micro Air Vehicle Communication Protocol MAVLink*. https://mavlink.io/en/. [Online; accessed February-2019]. 74

MEZARD, MARC, & MONTANARI, ANDREA. 2009. *Information, physics, and computation*. Oxford University Press. 43

MÉZARD, MARC, PARISI, GIORGIO, & VIRASORO, MIGUEL. 1987. *Spin glass theory and beyond: An Introduction to the Replica Method and Its Applications*. Vol. 9. World Scientific Publishing Company. 43

MICHEL, OLIVIER. 2004. Cyberbotics Ltd. Webots TM: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, **1**(1), 5. 19

MICROSOFT. *Microsoft HPC pack*. https://docs.microsoft.com/en-us/powershell/high-performance-computing/overview?view=hpc16-ps. [Online; accessed January-2019]. 11

MOLINA, MARTIN, SUAREZ-FERNANDEZ, RAMON A, SAMPEDRO, CARLOS, SANCHEZ-LOPEZ, JOSE LUIS, & CAMPOY, PASCUAL. 2017. TML: A language to specify aerial robotic missions for the framework Aerostack. *International Journal of Intelligent Computing and Cybernetics*, **10**(4), 491–512. 19

MONES, ENYS, VICSEK, LILLA, & VICSEK, TAMÁS. 2012. Hierarchy measure for complex networks. *PloS one*, **7**(3), e33799. 41, 45, 46

MOOS. *Microsoft Robotics Developer Studio 4*. https://www.microsoft.com/en-us/download/details.aspx?id=29081. [Online; accessed February-2019]. 18

MOOS. *The MOOS. Cross Platform Software for Robotics Research*. http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php. [Online; accessed February-2019]. 18

MUSK, ELON, SULEYMAN MUSTAFA ET AL. 2016. *An Open Letter to the United Nations Convention on Certain Conventional Weapons*. https://www.dropbox.com/s/g4ijcaqq6ivq19d/2017%20Open%20Letter%20to%20the%20United%20Nations%20Convention%20on%20Certain%20Conventional%20Weapons.pdf?dl=0. [Online; published August-2017, accessed February-2019]. 45

NASA-JPL. *Helicopter Could Be Ścoutfor Mars Rovers*. https://www.jpl.nasa.gov/news/news.php?feature=4457. [Online; accessed February-2019]. 17

NEWMAN, CH, & STEIN, D. 2013. *Spin glasses and complexity*. 43

NICHOLS, GREG. *ZDNet How to build a $200 smart drone with the Pi Zero*. https://www.zdnet.com/article/how-to-build-a-200-smart-drone-with-the-pi-zero/. [Online; published February-2016, accessed February-2019]. 33

NUTT, GREGORY. 2014. *Nutt X operating system user's manual*. http://wwww.nuttx.org/doku.php?id=documentation. 18, 120

NVIDIA. *Embedded Systems for Next-Generation Autonomous Machines. NVIDIA Jetson: The AI platform for autonomous everything*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/. [Online; accessed February-2019]. 4

NVIDIA. *High Performance Computing. A Supercharged Law*. https://www.nvidia.com/en-us/high-performance-computing/. [Online; accessed February-2019]. 26

NVIDIA. *Jetson AGX Xavier Developer Kit* . https://developer.nvidia.com/embedded/buy/jetson-agx-xavier-devkit. [Online; accessed January-2019]. 13, 111

NVIDIA. *NVIDIA Jetson systems TX1/TX2*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/. [Online; accessed January-2019]. 13

NVIDIA. *NVIDIA Jetson TK1*. http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html. [Online; accessed January-2019]. 13, 14

NVIDIA. *NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge*. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/. [Online; accessed January-2019]. 14

NVIDIA. *NVIDIA TITAN X*. https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/. [Online; accessed January-2019]. 14

NVIDIA. *Social impact of the GPU*. http://www.nvidia.com/object/social-impact-gpu.html. [Online; accessed July-2019]. 5, 12

NVIDIA. *Tegra K1 Chips*. https://developer.nvidia.com/embedded/tegra-k1. [Online; accessed January-2019]. 13, 14

ODROID. *ODROID-H2*. https://www.hardkernel.com/shop/odroid-h2/. [Online; accessed January-2019]. 13

OGS. *Open Grid Scheduler*. http://gridscheduler.sourceforge.net/. [Online; accessed January-2019]. 11, 19

OPENROBOTICS. *Open Robotics*. https://www.openrobotics.org/. [Online; accessed February-2019]. 19

ORCA. *Orca: Components for Roboticst*. http://orca-robotics.sourceforge.net/. [Online; accessed February-2019]. 18

ORNL. *Oak Ridge National Laboratory Summit supercomputer*. https://www.olcf.ornl.gov/summit/. [Online; accessed April-2019]. 11

OROCOS. *The Orocos Project. Smarter control in robotics and automation*. http://www.orocos.org/. [Online; accessed February-2019]. 18

PALOSSI, DANIELE, FURCI, MICHELE, NALDI, ROBERTO, MARONGIU, ANDREA, MARCONI, LORENZO, & BENINI, LUCA. 2016. An energy-efficient parallel algorithm for real-time near-optimal UAV path planning. *Pages 392–397 of: Proceedings of the ACM International Conference on Computing Frontiers*. ACM. 17

PARDO-CASTELLOTE, GERARDO. 2005. OMG Data Distribution Service: Real-Time Publish/Subscribe Becomes a Standard. *Rtc magazine*, **14**. 53, 120

PEE, LG, PAN, SHAN L, & CUI, LILI. 2018. Artificial intelligence in healthcare robots: A social informatics study of knowledge embodiment. *Journal of the Association for Information Science and Technology*. 17

PETITET, ANTOINE, WHALEY, CLINT, DONGARRA, JACK, & CLEARY, ANDY. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. http://www.netlib.org/benchmark/hpl/. [Online; accessed February-2019]. 70

PINCIROLI, CARLO, TRIANNI, VITO, O'GRADY, REHAN, PINI, GIOVANNI, BRUTSCHY, ARNE, BRAMBILLA, MANUELE, MATHEWS, NITHIN, FERRANTE, ELISEO, DI CARO, GIANNI, DUCATELLE, FREDERICK, *et al.* . 2011. ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. *Pages 5027–5034 of: International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. 19

PINE64. *Rock64 Media board*. https://www.pine64.org/?product=rock64-media-board-computer. [Online; accessed January-2019]. 13

PITONAKOVA, LENKA, GIULIANI, MANUEL, PIPE, ANTHONY, & WINFIELD, ALAN. 2018. Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators. *Pages 357–368 of:* GIULIANI, MANUEL, ASSAF, TAREQ, & GIANNACCINI, MARIA ELENA (eds), *Towards Autonomous Robotic Systems*. Cham: Springer International Publishing. 19

PLAYER. *The Player Project. Free Software tools for robot and sensor applications*. http://playerstage.sourceforge.net/index.php?src=index. [Online; accessed February-2019]. 18

POCKETBEAGLE. *PocketBeagle board*. https://beagleboard.org/pocket. [Online; accessed January-2019]. 13

QUALCOMM. *Qualcomm Technologies releases LTE drone trial results*. https://www.qualcomm.com/news/onq/2017/05/03/qualcomm-technologies-releases-lte-drone-trial-results. [Online; accessed January-2019]. 16

QUIGLEY, MORGAN, CONLEY, KEN, GERKEY, BRIAN, FAUST, JOSH, FOOTE, TULLY, LEIBS, JEREMY, WHEELER, ROB, & NG, ANDREW Y. 2009. ROS: an open-source Robot Operating System. *Page 5 of: ICRA workshop on open source software*, vol. 3. Kobe, Japan. 15, 17

RADCLIFFE, TOM. *Python vs. C/C++ in Embedded Systems*. https://www.activestate.com/blog/python-vs-cc-embedded-systems/. [Online; published August-2016, accessed February-2019]. 27, 75

RAJOVIC, NIKOLA, CARPENTER, PAUL M, GELADO, ISAAC, PUZOVIC, NIKOLA, RAMIREZ, ALEX, & VALERO, MATEO. 2013. Supercomputing with commodity CPUs: Are mobile SoCs ready for HPC? *Page 40 of: Proceedings of the international conference on high performance computing, networking, storage and analysis*. ACM. 4

RASPBERRY-PI-FOUNDATION. *Raspberry Pi*. https://www.raspberrypi.org/. [Online; accessed February-2019]. 4

REDHAT. *Red Hat Ansible*. https://www.ansible.com/. [Online; accessed February-2019]. 19

REDMON, JOSEPH, DIVVALA, SANTOSH, GIRSHICK, ROSS, & FARHADI, ALI. 2016. You only look once: Unified, real-time object detection. *Pages 779–788 of: Proceedings of the conference on computer vision and pattern recognition*. IEEE. 15

REPORTLINKER. 2018. *Global drone data services and analytics market generated $3,054.9 million in 2017, and the market is estimated to grow at a CAGR of 53.9% during 2018-2023* . http://www.infoq.com/news/2014/09/drone-data-big-data-analytics. [Online; published September-2018, accessed January-2019]. 3

RIBEIRO, CONSTANTINO GONCALVES, DUTRA, MAX SUEL, RABELO, ALINE, OLIVEIRA, FILIPE, BARBOSA, ALLAN, FRINHANI, LUCIANO, PORTO, DOUGLAS, & MILANEZ, RAYANNE. 2015. A Robotic Flying Crane Controlled by an Embedded Computer Cluster. *Pages 91–96 of: 12th Latin American Robotics Symposium and 3rd Brazilian Symposium on Robotics (LARS-SBR)*. IEEE. 15

RIJMEN, VINCENT, & DAEMEN, JOAN. 2001. Advanced encryption standard. *Proceedings of federal information processing standards publications, national institute of standards and technology*, 19–22. 102

ROBERGE, VINCENT, & TARBOUCHI, MOHAMMED. 2017. Fast path planning for unmanned aerial vehicle using embedded gpu system. *Pages 145–150 of: 14th international multi-conference on systems, signals & devices (SSD)*. IEEE. 14

ROHMER, ERIC, SINGH, SURYA PN, & FREESE, MARC. 2013. V-REP: A versatile and scalable robot simulation framework. *Pages 1321–1326 of: International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. 19

ROS. *Robot Operating System nodes*. http://wiki.ros.org/Nodes. [Online; accessed May-2019]. 32

ROS.ORG. *MAVROS*. http://wiki.ros.org/mavros. [Online; accessed February-2019]. 19

ROS.ORG. *ROS documentation. Introduction*. http://wiki.ros.org/ROS/Introduction. [Online; accessed February-2019]. 18

ROS.ORG. *ROS Installation*. http://wiki.ros.org/Installation. [Online; accessed February-2019]. 27

ROS.ORG. *ROS Melodic Morenia*. http://wiki.ros.org/melodic. [Online; accessed February-2019]. 32

ROS.ORG. *ROS mpi package*. http://wiki.ros.org/mpi. [Online; accessed February-2019]. 17

ROS.ORG. *ROS Parameter Server*. http://wiki.ros.org/Parameter%20Server. [Online; accessed February-2019]. 17

ROYO, PABLO, PASTOR, ENRIC, MACIAS, MIQUEL, CUADRADO, RAUL, BARRADO, CRISTINA, & VARGAS, ARTURO. 2018. An unmanned aircraft system to detect a radiological point source using RIMA software architecture. *Remote sensing*, **10**(11), 1712. 14

RPIFOUND. *Raspberry Pi 3 Model B+*. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Online; accessed January-2019]. 13

RPIFOUND. *Raspberry Pi Blog*. https://www.raspberrypi.org/blog/. [Online; accessed January-2019]. 14

RUBIN, MICHAEL. 2019. The role of software simulators in the independent verification and validation of commercial space vehicles. *Page 1708 of: AIAA Scitech Forum*. 19

RUSSELL, STUART J, & NORVIG, PETER. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,. 17

SAGITOV, ARTUR, & GERASIMOV, YURI. 2017. Towards DJI Phantom 4 Realistic Simulation with Gimbal and RC Controller in ROS/Gazebo Environment. *Pages 262–266 of: 10th International Conference on Developments in eSystems Engineering (DeSE)*. IEEE. 18

SALAMÍ SAN JUAN, ESTHER, SOLER, JOSÉ ALBERTO, CUADRADO SANTOLARIA, RAÚL, BARRADO MUXÍ, CRISTINA, & PASTOR LLORENS, ENRIC. 2015. Virtualizing super-computation on-board UAS. *The international archives of the photogrammetry, remote sensing and spatial information sciences*, **40**, 1291–1298. 12

SANCHEZ-LOPEZ, JOSE LUIS, FERNÁNDEZ, RAMÓN A SUÁREZ, BAVLE, HRIDAY, SAMPEDRO, CARLOS, MOLINA, MARTIN, PESTANA, JESUS, & CAMPOY, PASCUAL. 2016. Aerostack: An architecture and open-source software framework for aerial robotics. *Pages 332–341 of: International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 19

SANCHEZ-LOPEZ, JOSE LUIS, MOLINA, MARTIN, BAVLE, HRIDAY, SAMPEDRO, CARLOS, FERNÁNDEZ, RAMÓN A SUÁREZ, & CAMPOY, PASCUAL. 2017. A Multi-Layered Component-Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework. *Journal of Intelligent & Robotic Systems*, **88**(2-4), 683–709. 19

SATYANARAYANAN, MAHADEV. 2010. Mobile computing: the next decade. *Page 5 of: Proceedings of the 1st workshop on mobile cloud computing & services: social networks and beyond*. ACM. 3

SCHEDMD. *Simple Linux Utility for Resource Management*. https://slurm.schedmd.com/. [Online; accessed January-2019]. 11, 30

SCHMITTLE, MATT, LUKINA, ANNA, VACEK, LUKAS, DAS, JNANESHWAR, BUSKIRK, CHRISTOPHER P, REES, STEPHEN, SZTIPANOVITS, JANOS, GROSU, RADU, & KUMAR, VIJAY. 2018. OpenUAV: a UAV testbed for the CPS and robotics community. *Pages 130–139 of: Proceedings of the 9th International Conference on Cyber-Physical Systems*. ACM/IEEE. 19, 20

SHAH, SHITAL, DEY, DEBADEEPTA, LOVETT, CHRIS, & KAPOOR, ASHISH. 2018. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. *Pages 621–635 of: Field and service robotics*. Springer. 19

SHEPLER, SPENCER, CALLAGHAN, BRENT, ROBINSON, DAVID, THURLOW, ROBERT, BEAME, CARL, EISLER, MIKE, & NOVECK, DAVE. 2003. *Network file system (NFS) version 4 protocol*. Tech. rept. 5, 32

SHI, WEISONG, CAO, JIE, ZHANG, QUAN, LI, YOUHUIZI, & XU, LANYU. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal*, **3**(5), 637–646. 3

SMASHINGROBOTICS. *Most Advanced Robotics Simulation Software Overview*. http://www.smashingrobotics.com/most-advanced-and-used-robotics-simulation-software/. [Online; accessed February-2019]. 19

SPENKO, MATTHEW, BUERGER, STEPHEN, & IAGNEMMA, KARL. 2018. *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*. Vol. 121. Springer. 17

STERLING, THOMAS, ANDERSON, MATTHEW, & BRODOWICZ, MACIEJ. 2017. *High Performance Computing: Modern Systems and Practices*. Morgan Kaufmann. 12

TENSORFLOW. *TensorFlow. An open source machine learning framework for everyone*. https://www.tensorflow.org/. [Online; accessed February-2019]. 27

TIJTGAT, NIELS, VAN RANST, WIEBE, VOLCKAERT, BRUNO, GOEDEMÉ, TOON, & DE TURCK, FILIP. 2017. Embedded real-time object detection for a UAV warning system. *Pages 2110–2118 of: The International Conference on Computer Vision (ICCV2017)*. 15

TOP500. *Summit supercomputer*. https://www.top500.org/system/179397. [Online; accessed February-2019]. 115

TOTAL. *TOTAL's Pangea supercomputer: among the global top Ten in terms of computing power*. https://www.total.com/en/media/video/totals-pangea-supercomputer-among-global-top-ten-terms-computing-power. [Online; accessed April-2019]. 11

TRANCOSO, PEDRO, & EFSTATHIOU, MICHALIS. 2017. Low-Cost Sub-5W Processors for Edge HPC. *Pages 529–532 of: Euromicro conference on digital system design (DSD)*. IEEE. 4

TRAXXAS. *Traxxas Slash 1/10 2W*. https://traxxas.com/products/models/electric/58034-2slash. [Online; accessed February-2019]. 69, 98

UMMAT, AJAY, SHARMA, GAURAV, MAVROIDIS, C, & DUBEY, A. 2016. Bio-nanorobotics: State of the art and future challenges. *Pages 309–354 of: Tissue Engineering and Artificial Organs*. CRC Press. 17

USACH, HECTOR, VILA, JUAN A, TORENS, CHRISTOPH, & ADOLF, FLORIAN. 2018. Architectural design of a Safe Mission Manager for Unmanned Aircraft Systems. *Journal of systems architecture*, **90**, 94–108. 17

VALAVANIS, KIMON P, & VACHTSEVANOS, GEORGE J. 2015. Future of unmanned aviation. *Pages 2993–3009 of: Handbook of unmanned aerial vehicles*. Springer. 17

VARRASI, SIMONE, LUCAS, ALEXANDER, SORANZO, ALESSANDRO, MCNAMARA, JOHN, & DI NUOVO, ALESSANDRO. 2019. IBM Cloud Services enhance automatic cognitive assessment via human-robot interaction. *Pages 169–176 of: New Trends in Medical and Service Robotics*. Springer. 17

VARSHAVSKY, ALEXANDER, & PATEL, SHWETAK. 2016. Location in ubiquitous computing. *Pages 299–334 of: Ubiquitous computing fundamentals*. Chapman and Hall/CRC. 16

VEGA, AUGUSTO, LIN, CHUNG-CHING, SWAMINATHAN, KARTHIK, BUYUKTOSUNOGLU, ALPER, PANKANTI, SHARATHCHANDRA, & BOSE, PRADIP. 2015. Resilient UAV-embedded real-time computing. *Pages 736–739 of: 33rd international conference on computer design (ICCD)*. IEEE. 14

VICSEK, TAMÁS, CZIRÓK, ANDRÁS, BEN-JACOB, ESHEL, COHEN, INON, & SHOCHET, OFER. 1995. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, **75**(6), 1226. 63, 74, 78, 117

VIRTUALROBOTICSTOOLKIT. *Virtual Robotics Toolkit*. https://www.virtualroboticstoolkit.com/. [Online; accessed February-2019]. 19

WAIBEL, MARKUS, BEETZ, MICHAEL, CIVERA, JAVIER, D'ANDREA, RAFFAELLO, ELFRING, JOS, GALVEZ-LOPEZ, DORIAN, HÄUSSERMANN, KAI, JANSSEN, ROB, MONTIEL, JMM, PERZYLO, ALEXANDER, *et al.* . 2011. Roboearth. *IEEE Robotics & Automation Magazine*, **18**(2), 69–82. 19, 64

WALKER, DAVID W, & DONGARRA, JACK J. 1996. MPI: A standard message passing interface. *Supercomputer*, **12**, 56–68. 4

WEAVER, JOSHUA N, FRANK, DANIEL Z, SCHWARTZ, ERIC M, & ARROYO, A ANTONIO. 2013. UAV performing autonomous landing on USV utilizing the robot operating system. *In: ASME Early Career Technical Symposium*. Citeseer. 18

WEI, HONGXING, HUANG, ZHEN, YU, QIANG, LIU, MIAO, GUAN, YONG, & TAN, JINDONG. 2014. RGMP-ROS: a Real-time ROS Architecture of Hybrid RTOS and GPOS on Multi-core Processor. *Pages 2482–2487 of: International Conference on Robotics and Automation (ICRA)*. IEEE. 18

WEI, HONGXING, SHAO, ZHENZHOU, HUANG, ZHEN, CHEN, RENHAI, GUAN, YONG, TAN, JINDONG, & SHAO, ZILI. 2016. RT-ROS: A real-time ROS architecture on multi-core processors. *Future Generation Computer Systems*, **56**, 171–178. 18

WEISER, MARK. 1993a. Hot topics-ubiquitous computing. *Computer*, **26**(10), 71–72. 16

WEISER, MARK. 1993b. Some computer science issues in ubiquitous computing. *Communications of the ACM*, **36**(7), 75–84. 16

YANG, CHENGUANG, WANG, XINGJIAN, LI, ZHIJUN, LI, YANAN, & SU, CHUN-YI. 2017. Teleoperation control based on combination of wave variable and neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, **47**(8), 2125–2136. 17

YARP. *YARP: Yet Another Robot Platform*. https://www.yarp.it/. [Online; accessed February-2019]. 18

YOSHIDA, KAZUYA. 2009. Achievements in space robotics. *IEEE Robotics & Automation Magazine*, **16**(4). 17

YU, YANGGUANG, WANG, XIANGKE, ZHONG, ZHIWEI, & ZHANG, YONGWEI. 2017. ROS-based UAV control using hand gesture recognition. *Pages 6795–6799 of: 29th Chinese Control And Decision Conference (CCDC)*. IEEE. 18

ZAMANI, MARYAM, & VICSEK, TAMAS. 2017. Glassy nature of hierarchical organizations. *Scientific Reports*, **7**(1), 1382. 43, 44, 45

ZEILENGA, KURT. 2006 (6). *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*. The Internet Society. Network Working Group. 5, 11, 32

ZHANG, WEISHAN, ZHAO, DEHAI, XU, LIANG, LI, ZHONGWEI, GONG, WENJUAN, & ZHOU, JIEHAN. 2016. Distributed embedded deep learning based real-time video processing. *Pages 001945–001950 of: IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 14

ZHONGYUAN, GUO, WENJING, YANG, MINGLONG, LI, XIAODONG, YI, ZHONGXUAN, CAI, & YANZHEN, WANG. 2018. ALLIANCE-ROS: A software framework on ROS for fault-tolerant and cooperative mobile robots. *Chinese Journal of Electronics*, **27**(3), 467–475. 18