D7.1 - INITIAL CPSWARM ABSTRACTION LIBRARY

Deliverable ID	**D7.1**
Deliverable Title	**Initial CPSwarm Abstraction Library**
Work Package	**WP7 - Deployment Toolchain**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**2018-07-11**
Status	**Final**
Lead Editor	Gianluca Prato **(ISMB)**
Main Contributors	Bálint Jánvári **(SLAB),** Etienne Brosse **(SOFTEAM),** Artiza Elosegui **(TTTECH),** Angel Soriano **(ROBOTNIK),** Gianluca Prato **(ISMB),** Omar Morando **(DIGISKY)**

Published by the CPSwarm Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731946.

Document History

Version	Date	Author(s)	Description
0.01	2018-05-07	Gianluca Prato (ISMB)	First Draft with TOC
0.02	2018-06-18	Gianluca Prato (ISMB)	TOC update
0.03	2018-06-25	Gianluca Prato (ISMB)	Added contents related to Section 3 and 4
0.04	2018-06-26	Gianluca Prato (ISMB)	Integrated contributions from SOFTEAM, ROBOTNIK, TTTECH and SLAB
0.05	2018-07-02	Artiza Elosegui (TTTECH)	Harmonized TTTECH contributions with D2.2
0.06	2018-07-06	Omar Morando (DIGISKY)	Added DIGISKY contributions in Section 6
1.0	2018-07-11	Gianluca Prato (ISMB)	Integrated comments from revisions (DIGISKY, ROBOTNIK)

Internal Review History

Review Date	Reviewer	Summary of Comments
2018-07-10	Angel Soriano (ROBOTNIK)	Approved with minor comments.
2018-07-10	Omar Morando (DIGISKY)	Approved with few comments.

Table of Contents

1 Executive Summary

This deliverable, "D7.1 Initial CPSwarm Abstraction Library", gives a complete report of the design phase and first implementation of CPSwarm Abstraction Library. Particular attention has been given to justify the selected approach in order to fulfil the project requirements. In the first part of the document a survey of common CPS platforms and an analysis of common ways to provide robotic adaptation libraries is performed. Then, a detailed description of the selected design for the library is presented. Finally, the deliverable outlines the current implementations towards M18 Review.

This deliverable reports on the results of Task 7.1 - CPSwarm Abstraction Library.

2 Introduction

Autonomous robots are complex systems that require the interaction between numerous heterogeneous components (software and hardware). Because of the general complexity of robotic applications and the diverse range of hardware, many software libraries have been developed to promote the integration of new technologies hiding the complexity of low-level hardware. Furthermore, the development of these libraries has been pushed by the desire to improve software quality, ease the reuse of robotic software infrastructures across multiple research efforts, and to reduce production costs. A survey on some of the most popular robotic libraries has been used to assemble a base knowledge that drove the design and implementation of the Initial CPSwarm Abstraction Library.

2.1 Document Organization

The document is organized as follows: firstly, Section 3 introduces the current state-of-art of CPS abstraction libraries and middlewares. Then, Section 4 presents the software design of the CPSwarm Abstraction Library and describes the link of this component with the rest of the workbench. Section 5 is dedicated to the description of the Communication Library, developed to let each CPS communicate among them and with external tools. Finally, Section 6 reports all the implementation realized till M18 with focus on the use case scenarios.

2.2 Related documents

ID	Title	Reference	Version	Date
[D2.2]	Final Vision Scenarios and Use Case Definition	D2.2	1.0	M16
[D5.3]	Updated CPSwarm Modelling Tool	D5.3	1.0	M18
[D7.2]	Final CPSwarm Abstraction Library	D7.2		M32

3 CPS Abstraction Libraries State-of-Art

This section provides an overview of hardware Abstraction Libraries which aims to identify the major solutions in the Cyber Physical System context. Furthermore, in order to identify the best solution for the initial implementation of the Abstraction Library, an examination of common robotic systems that shall be supported by the workbench has been carried out.

3.1 CPS main categories

During the first months of Task 7.1 particular attention was given to the identification of the different categories of CPS that should be targeted by the CPSwarm project. In the following, we mainly found two possible ways to categorize the robotic systems giving answer to two different questions:

1. In which application this CPS is employed?
2. How the CPS execute his tasks?

3.1.1 Categorization by application

- **Industrial CPSs**

  Industrial CPSs are robots used in an industrial manufacturing environment. Usually these are articulated arms specifically developed for such applications as welding, material handling, painting and others.

- **Domestic CPSs**

  CPSs used at home. This type of robots includes many quite different devices such as robotic vacuum cleaners, robotic pool cleaners, sweepers, gutter cleaners and other robots that can do different chores. Also, some surveillance and telepresence robots could be regarded as household robots if used in that environment.

- **Medical CPSs**

  Robots used in medicine and medical institutions. They mainly include surgery-assistant robots such as telemanipulators that allows the surgeon to perform the normal movements associated with the surgery whilst the robotic arms carry out those movements using end-effectors and manipulators to perform the actual surgery on the patient.

- **Military CPSs**

  CPSs used in military. This type of robots includes bomb disposal robots, different transportation robots, reconnaissance drones. Often robots initially created for military purposes can be used in law enforcement, search and rescue and other related fields.

- **Entertainment CPSs**

  These are robots used for entertainment. This is a very broad category. It starts with toy robots (e.g. the very famous Robosapien[1]) or robots realized just for fun and robots made for competition.

- **Space CPSs**

  This category includes robots used on the International Space Station, robotic arms such as Canadarm[2] that was used on the Space Shuttle orbiters to deploy, maneuverer and capture payloads, as well as Mars rovers and other robots used for space exploration' missions

3.1.2 Categorization by mobility

A second way to categorize CPSs is based on how they can or cannot move while they are executing their main tasks. Based on this consideration, 5 major classes have been identified.

[1] https://wowwee.com/robosapien-x
[2] https://en.wikipedia.org/wiki/Canadarm

- **Stationary CPSs**

  Stationary CPSs are robots that work without changing their positions. Referring the robot as "stationary" does not mean that the robot is not moving. What "stationary" means is that the base of the robot does not move during operation.

  This kind of robots generally manipulate their environment by controlling the position and orientation of an end-effector. Stationary CPS category includes robotic arms, Cartesian/Gantry robots, cylindrical robots, spherical robots, SCARA robots and parallel robots.

Figure 1 - Stationary arm CPS

- **Wheeled CPSs**

  Wheeled robots are robots which change their positions with the help of their wheels. Wheeled motion for a robot can be achieved easily in mechanical terms and its cost is pretty low. Additionally, control of wheeled movement is generally easier.

  These reasons make wheeled robots one of the most frequently seen robots. Single wheeled robots, mobile ball robots, two-wheeled robots, three-wheeled robots, four-wheeled robots, multi-wheeled robots and tracked robots are examples of wheeled robots.

Figure 2 - Simple four-wheeled robot

- **Legged CPSs**

  Legged robots are mobile robots, similar to wheeled robots, but their locomotion methods are more sophisticated and complicated compared to their wheeled counterparts. As their name suggests they use their legs to control their locomotion and they perform much better than wheeled robots on uneven terrain.

  Despite the cost and complexity of production is high for these robots, their advantages on uneven terrain makes these robots indispensable for most applications. One-legged robots, two-legged robots, three-legged robots, four-legged robots, six-legged robots and multi-legged robots are examples of this robot class.

Figure 3 - Six legs walking robot

- **Underwater CPSs**

  Swimming CPSs are robots which move underwater. These robots are generally inspired by fish and they usually use their fin-like actuators to maneuverer in water.

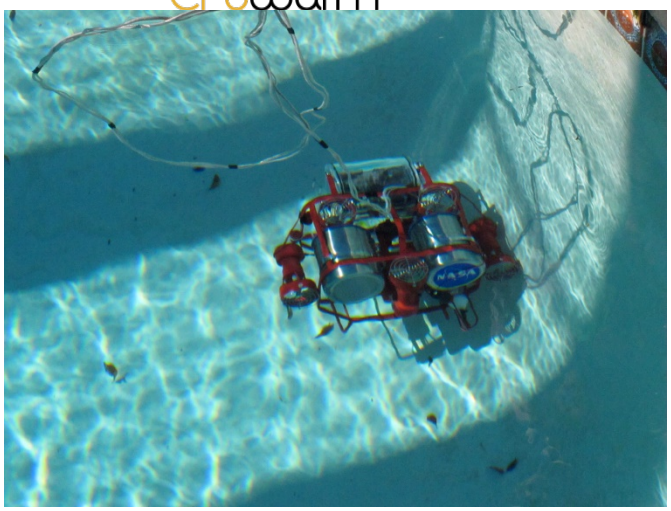

Figure 4 - Underwater CPS

- **Flying CPSs**

  Flying robots are robots that float and maneuverer on air using their plane-like or bird/insect-like wings, propellers or balloons. Examples of these robots are airplane robots, bird/insect inspired wing flapping robots, propeller based multicopters and balloon robots.

Figure 5 - Quadrocopters Unmanned Aerial Vehicle

3.2 Main Abstraction Libraries Overview

The concept of Abstraction Library is often strictly interconnected to that of middleware. The middleware solution, in fact, responds well to the necessity of aggregating the large number of sub-systems from which a robotic system is generally composed. Middleware could be imagined as a layer that lies between the application code and the run-time infrastructure. Middleware generally consists of a library of functions and enables a number of applications to use these functions from the common library rather than re-create them for each application [1].

In the next paragraphs there will be a short presentation of some of the most famous middlewares that are commonly used in the robotic context. The aim of this analysis is not to have an in-depth knowledge of the different implementations but to focus on the adopted solutions in order to achieve some common objectives like simplifying the development process, integrate different platforms and guarantee the reusability of the software. More details on the different robotic framework can be found in [2] and [3].

3.2.1 OROCOS

OROCOS[3] stands for Open RObot Control Software. OROCOS provides a toolchain to create real-time applications using modular and runtime configurable components. OROCOS is composed by 4 main C++ libraries:

- **OROCOS Real-Time Toolkit** (RTT): represents the core of OROCOS and provides the infrastructure and the functionalities to build robotics applications.
- **OROCOS Components Library** (OCL): this library provides some ready to use component to interface with hardware devices to control various types of robot as well as to record and display data flows and to debug the system.
- **Kinematics and Dynamics Library** (KDL): this library allows the modelling and development of kinematic chains.
- **Bayesian Filtering Library** (BFL): provides an application independent framework for inference in Dynamic Bayesian Networks, i.e., recursive information processing and estimation algorithms based on Bayes' rule.

The main objective of this middleware, since his first releases, is to provide a hard-efficient real-time ecosystem. For this reason, many possible extensions suggested by the community have been ignored in order to preserve this basic concept and letting these services been accessible through the integration of external projects.

3.2.2 Orca

Orca[4] is an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems. The project's main goal is to promote software reuse in robotics [4]. Orca tries to achieve this goal focusing on three concepts:

- Enable software reuse by defining a set of commonly-used interfaces.
- Simplify software reuse by providing libraries with a high-level convenient API.
- Encourage software reuse by maintaining repository of components.

Orca adopts a Component-Based Software Engineering approach without applying any additional constraints and uses a commercial open-source library for communication and interface definition. Moreover, it provides tools to simplify component development but make them strictly optional to maintain full access to the underlying communication engine and services.

While for the earlier versions of Orca the communication engine was based on CORBA[5] (like in OROCOS), due to the complexity of his system, it was then replaced by the Internet Communication Engine[6] (ICE).

3.2.3 ROS

ROS[7] is defined as an open-source meta-operating system, highlighting that it wants to be something more than a simple middleware. It provides the services expected from an operating system, including hardware abstraction, low-level hardware control, IPC facilities and package management. Its primary goal, as for Orca, is to enable code reuse in robotics research and development and to release ready to use software packages for a large set of common tasks. ROS provides also a community website for sharing and collaboration, and repositories to store and distribute software packages.

[3] http://www.orocos.org/
[4] http://orca-robotics.sourceforge.net/
[5] http://www.corba.org/
[6] https://zeroc.com/products/ice
[7] http://www.ros.org/

to use ROS, users create stand-alone programs called Nodes[8], each denoted as *n*, with custom or existing ROS packages. Each node *n* can execute robot commands, update sensor information or process and forward incoming data. To exchange information with other nodes, each node must connect to a ROS master. Through the master, each node can locate and communicate with another node through message-passing in three different methods:

1. Topics[9]
2. Services[10]
3. Actions[11]

The ROS ecosystem now consists of tens of thousands of users worldwide, working in domains ranging from table top hobby projects to large industrial automation systems. Thanks to this big community of developers, ROS is continuously updated (last release was on May 2018) and many bridges have been implemented in order to integrate other robotic projects such as OROCOS and MAVLink[12].

[8] http://wiki.ros.org/Nodes
[9] http://wiki.ros.org/Topics
[10] http://wiki.ros.org/Services
[11] http://wiki.ros.org/actionlib
[12] https://mavlink.io/en/

4 CPSwarm Abstraction Library

Developing software to control intelligent robots is challenging in multiple ways: robot setups typically involve wildly different pieces of hardware running on different platforms [5]. The CPSwarm Abstraction Library (AL) should guarantee the support for controlling several types of sensors such as ultrasonic range sensors, cameras, GPS systems or actuators as grippers, motors and servos. Furthermore, it should raise the level of abstraction from a platform-dependent point view to a more application-oriented perspective. Programming robots, in general, is a complicated task that requires experts with advanced technical and methodological knowledge of the robotics domain. In this sense, the CPSwarm AL provides facilities to easily develop high-level routines, shifting the focus of the developer from what it should code in order to let the CPS execute a particular action, to just describe how the CPS should behave in order to complete a high-level task or reach an application-specific goal.

To summarize, therefore, the CPSwarm Abstraction Library covers two different roles inside the project: the former is to provide a set of CPS-specific adaptation libraries in order to access platform-specific information of a robotic system in a standard and coherent way. The latter is to provide support for the development of algorithms using a Model-Driven approach that can be deployed on CPSs using an automated code generation process.

To achieve both these purposes, we organized the AL as a composition of tree main overlapping components called Levels:

- Functions Level
- Sensing/Actuation Level
- Drivers Level

In Figure 6 is presented the complete structure of the Abstraction Library.

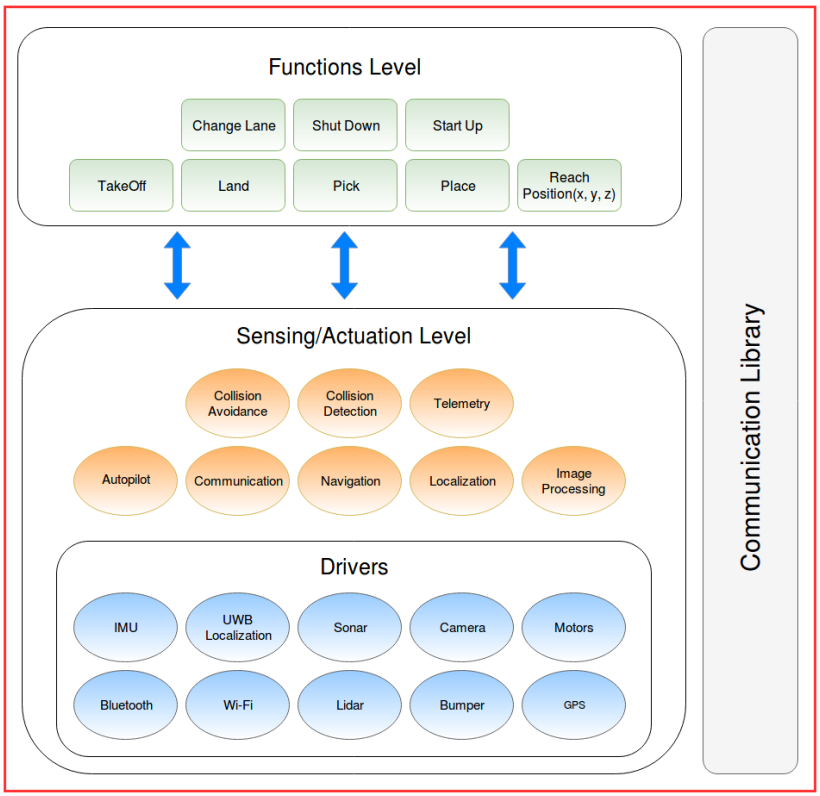

Figure 6 - The CPSwarm Abstraction Library

Alongside these 3 modules, a fourth component has been introduced: the Communication Library. The Communication Library is transversal to the rest of the AL as it provides a full stack of communication facilities. It is responsible to manage and coordinate all communication of a single CPS to others CPSs or to external tools ensuring an appropriate level of security. Section 5 has been dedicated to the presentation of this component.

In next three paragraphs the role of the different levels will be detailed following a bottom-up prospect in order to better highlight the interconnections among them.

4.1 Drivers Level

All the software libraries that are responsible to enable the other computer programs to access the hardware functions are gathered at this level. This component constitutes the fundament of the whole Abstraction Library. In the course of the project, the library will be enriched with all the drivers that are typically mounted on common robotic systems (starting from the CPS involved in the three use case scenarios), such as:

- Sensor
    - Accelerometer
    - Camera
    - Distance Sensor
    - GPS
    - Receiver (e.g. Wi-Fi, Bluetooth, etc...)
    - Lidar
    - Ultra-Wideband Indoor Localization
    - Range Finder
    - Inertial Measurement Unit (IMU)
    - Infrared Sensor
- Actuator
    - Brake
    - Motor (linear or rotational)
    - Speaker
    - LED
    - Gripper

4.2 Sensing/Actuation Level

This level is responsible to provide sensors information and to control the CPS using his actuators. While the Drivers Level has a direct connection with the hardware, this level is a pure software component that contributes to supply a first degree of abstraction to realize complex functionalities provided by the above level. Depending on the available computational power, memory and hardware resources, this level can be present as an independent component or incorporated inside the Driver Level.

4.3 Functions Level

At this stage there is a set of high-level functionalities corresponding to complex routines that a CPS can execute involving a set of sensors and actuators. Each function interacts with the below level sending specific commands and requesting for specific sensors information.

The role of this level is quite central for the actual implementation of the CPSwarm workbench. As presented in D5.3, during the first half of the project the focus was mainly concentred on the modelling of new behaviours in form of Finite State Machines (FSM). In this regard, each function can constitute a base building block to define a new FSM. A single State can be associated to a specific function of the AL that will be executed while the state is active.

Therefore, the presence of this level supply to the workbench two essential features:

- The application of model-driven techniques based on the design of FSM that will speed up the development process to realize new CPSs behaviours.
- The mapping of robot's functionalities to specific software modules that can guarantee the reusability of those functionalities. In this way, the effort to develop new CPS applications will be significantly reduced letting the developers re-use some existing solution and focusing their attention to more application-specific problems.

4.4 ROS-based implementation description

For the first implementation of the CPSwarm Abstraction Library the Consortium agreed on the use of ROS as a base framework. Before giving details on how ROS has been used to implement the AL, a short overview of the reasons that drove the decision to adopt it will be presented. ROS was selected for main reasons:

1. ROS is open-source. Most of the core packages are released under a BSD license. Furthermore, other licenses are commonly used in the community packages, such as the Apache 2.0 license, the GPL license, the MIT license, and even proprietary licenses. Each package in the ROS ecosystem is required to specify a license. Thanks to this, the Consortium could use the many of the developed packages as starting point for the implementation and then extend and modify them according to specific project necessities.
2. Inter-platform operability: ROS message-passing means that is possible to work in an environment where different components and subsystems can run be written with different languages. In this way, it is possible to select the most suitable language for each component considering specific requirement that a single component could have in respect of the Level it belongs to. For instance, a component in the Driver Level could be written with C code in order to guarantee the right execution speed, while a component of the Functions Level could be implemented using Java or Python that can support a more abstract environment to ease the code development.
3. Modularity: the modular design of ROS can guarantee enough flexibility to the system and easing the reuse of the code. Furthermore, it fit well the structure designed for the CPSwarm AL based on overlapping levels that communicate with each other.
4. ROS is directly supported by some simulation tools such as Gazebo and Stage, easing the deployment of code realized in a simulated environment on an actual CPSs.
5. Finally, ROS is generally considered as a standard *de facto* among the robotic community and is well supported by many CPS platforms including those that will be used in the three use case scenarios.

For this decision, also the restrictions of ROS regarding the real-time execution and the difficulty to be installed on low-resource system has been taken under consideration. While, for this implementation of the AL this aspect represents a limit, the Consortium was aware of the fact that a new version of ROS, called ROS 2, is under heavy development in order to guarantee embedded systems support and real-time execution. Of course, not all of what has been currently developed could be directly re-used, but could work as a starting point for future developments. Moreover, the ROS community is working to ease the migration process from ROS 1 package to ROS2.

For the implementation of the three Levels of the AL architecture common ROS conventions have been followed. All data coming from the sensors are continuously streamed through the ROS topics facilities. Following a single sender multiple receiver arrangement, each component in the stack that is interested in

processing a particular type of data (e.g. the current position, the speed, …) can subscribe to the dedicated topic to receive periodic updates. This method can be applied to implement virtual sensor: a processing module can collect information coming from different sources, elaborate them and finally re-publish the result on a new topic. The publish and subscribe mechanism provided by ROS is used also by the Communication Library in order to gather the telemetry information and provide them to the Monitoring Tool.

The dispatch of direct commands and requests for specific information are managed through the ROS Services. Indeed, the client-server scheme fits well to manage this request/reply interaction between two specific components.

Finally, for components that belong to the Function Level a mixed approach has been preferred:

- Short running tasks, such as moving up and an elevator or letting a drone take off, can be activated using Services.
- For long running and computational expensive operations (e.g. path following or a target research), the ROS actionlib package has been used. In fact, ROS Actions provide a feedback mechanism that is very useful to check the current status of a long activity during the execution.

4.5 Integration and correlation with CPSwarm Workbench Components

The Abstraction Library is strictly interconnected with many of the modules that define the CPSwarm toolchain. The goal of this chapter is to present the link between the Abstraction Library and the different components of the workbench.

4.5.1 Modelling Tool

To be used during the modelling of swarm member behaviour, a UML representation of the CPSwarm Abstraction library has been produced. A **model library** or **library** in a Modelling project is a set of non-modifiable model elements that is required for the development of a project, packaged in a single artefact. To properly model the behaviour of swam member, many functionalities from the CPSwarm Abstraction Library can be used either as atomic action (i.e. these actions won't be decomposed further) of a whole model behaviour. These required model elements are brought into the CPSwarm modelling project via a **model library**. The intention is not to edit or modify the CPSwarm Abstraction Library model, which is why the models provided by libraries are **read-only**.

Figure 7 depicts part of CPSwarm Abstraction Library under Modelling Tool. Two hardware components are defined - *Controller* and *Elevator* - with a Data Type - *2D Pose*. Both Component provided set of functionalities model as Operation with their respective parameters.

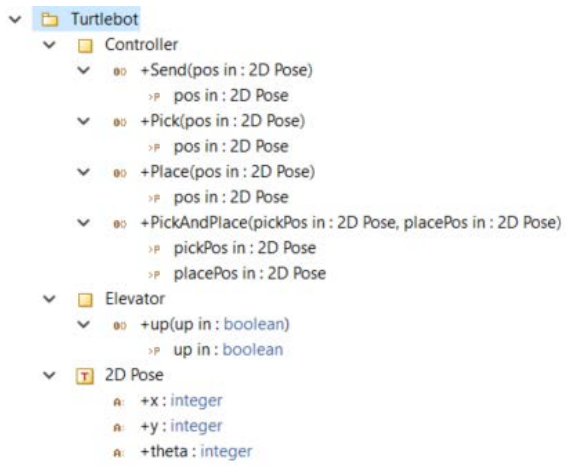

Figure 7 - CPSwarm Abstraction Library Under Modelling Tool

4.5.2 Code Generator

In the first half of the project the Consortium decided to focus on the implementation of swarm device behaviour algorithms modelled as Hierarchical Finite State Machines (HFSM). Each state of the state machine can be associated with a high-level functionality provided by the Abstraction Library or with an optimized algorithm coming from the Optimization Tool. In both cases, the role of the Code Generator is to serve as a "glue" level between the platform-independent algorithms realized using the Modelling Tool and the Abstraction Library that provides a set of APIs to access the functionalities of the CPS.

As is possible to see in Figure 8, the Code Generator has a central role in the toolchain from the modelling part and the deployment of the code on the actual CPS relying on the functionalities provided by the Abstraction Library.

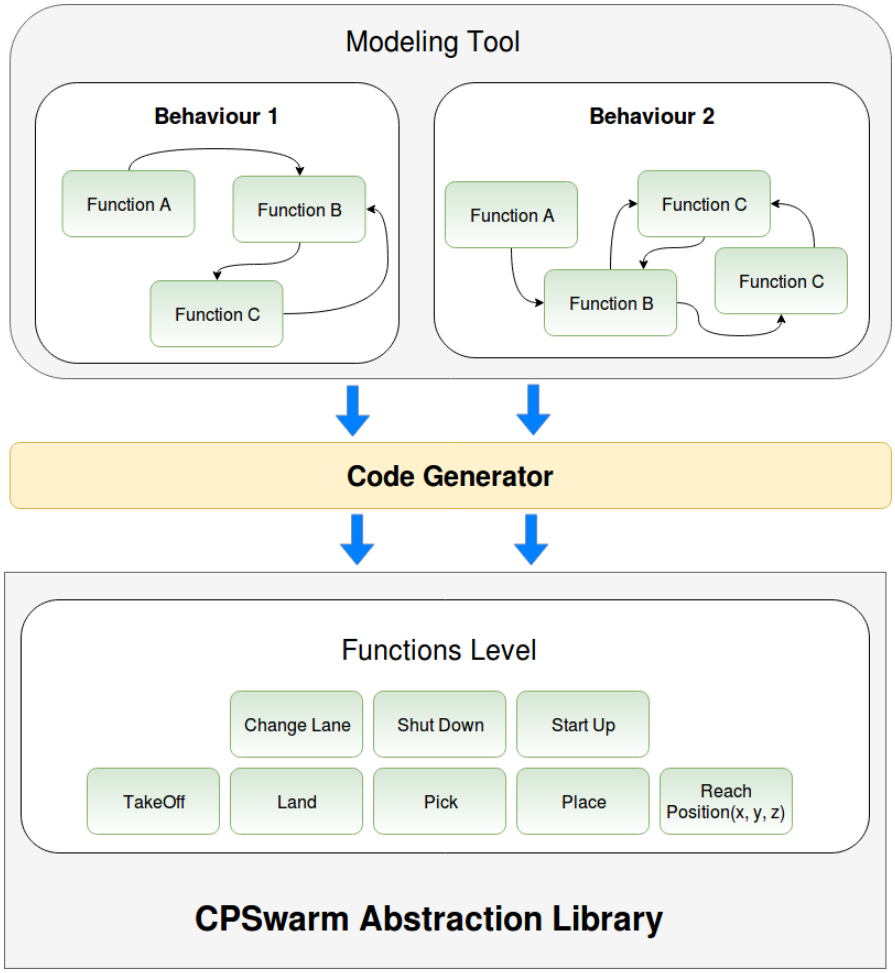

Figure 8 - Interconnection between different workbench's components

4.5.3 Monitoring Tool

The communication between the Monitoring Tool and the Abstraction Library is intermediated by the Communication Library. Exploiting the discovery functionality provided by the Communication Library, the events and topics of the swarm members can be discovered. Using this information, the interface of the Monitoring Tool can be dynamically generated to let the user to send events and to filter the data coming from the swarm member keeping only the one that the user has selected to see.

The list of commands that can be sent to the agent is exposed to the Monitoring Tool through the discovery feature of the Communication Library. Sending commands from the Monitoring Tool we can influence the behaviour of a specific CPS triggering events that start the execution of a particular sequence of actions.

5 Communication Library

The Communication Library provides a unified interface tools that swarm members can use to interact with each other. It is the duty of the library to ensure that all communications happen with the desired reliability, security level and latency.

After evaluating the requirements established by our core use cases and the design goals of a swarm in general, we concluded that interactions have a well-defined set of primitives and actions:

- Swarm members need to be discoverable on the network
- Events and commands need to be sent and received
- Parameters need to be remotely adjustable
- Telemetry needs to be sent back to operators and other subscribers

Our aim is to provide a stable API for all tools that abstracts away the physical layer and the authentication scheme used. To do this, a pluggable architecture was designed for the Communication Library, which separates the logical layer responsible for implementing these primitives and the endpoint implementation capable of sending individual messages over the network. This extensible infrastructure makes it possible to add support for new low-level protocols, physical layers and security schemes without affecting the rest of the system. As a first step, the Zyre[13] protocol was integrated with the library, but other endpoint types will be supported as the project progresses - in order to better support mesh networking and the deterministic wireless networks designed by TTTech.

Tools and any custom software developed by users can link against the public C++ API of the library, and can participate in the swarm on equal terms with other swarm members. While swarm management tools such as those integrated with the CPSwarm Workbench seem to be the primary target for this API, IoT devices can also run custom software capable of participating in the swarm in order to realize IoT2Swarm interactions and to offer a physical interface for operators.

Since in the end all three of our use cases involve ROS, special care was taken to integrate well into a ROS environment - which manifests in a ROS node capable of bridging native ROS IPC facilities with the primitives supported by the Communications Library. Among other things, this makes it possible to adjust the parameters of ROS packages unrelated to the CPSwarm project and to observe through telemetry the topics published by any of the ROS components. Since a typical ROS based use case will involve many third-party packages (for example in order to support hardware components), this will, in many cases, spare the user the effort of having to use a separate facility to interact with their topics and parameters or to write such a bridge manually.

Implementing security features is the responsibility of individual endpoint implementations. The Zyre protocol supports (as a beta feature) a CURVE based encryption and authentication scheme, which provides basic confidentiality and authentication between swarm members. We also plan to build or integrate a protocol capable of supporting more advanced features such as authorization of commands and selective encryption. End users may also implement any scheme they think would better fit their use case without disrupting any of the other components thanks to the pluggable architecture.

While it is not the primary goal of the library to provide real-time or near real-time performance, such an endpoint type can in theory be developed. We aim to benchmark the performance of the implementations that will be provided with the final release to help end users evaluate whether they fit their requirements. The current Zyre based implementation suggests a 10ms overhead on a request-reply type exchange over standard IP based networks.

[13] https://github.com/zeromq/zyre

6 Developments for Use Case Scenarios

The goal of this chapter is to describe all the developments related to the use case scenarios with focus on the three different levels. For all activities that will not be completed in M18, a description of the current understanding and planned implementations is presented instead.

6.1 Search and Rescue Scenario

Description of work from Official Documentation (D2.1): "We will consider heterogeneous swarms of ground robots/rovers and UAVs to conduct certain missions in the surveillance of critical infrastructure e.g., industrial or power plants as well as in Search and Rescue (SAR) tasks.". Starting from this assumption, we have made 5 compact UAVs and 3 ground robots equipped with the following components:

- n. 1 autopilot based on PX4 flight stack and PixHawk board, with 10DOF IMU
- n. 1 on-board computer, a Quad-core Cortex-A7 and Ubuntu 16.04
- n. 3 ultrasonic range finder for collision avoidance
- n. 1 camera with built-in ARM 32bits processor for computer vision (target tracking)
- n. 1 Ultra-Wideband (UWB) module for precise positioning

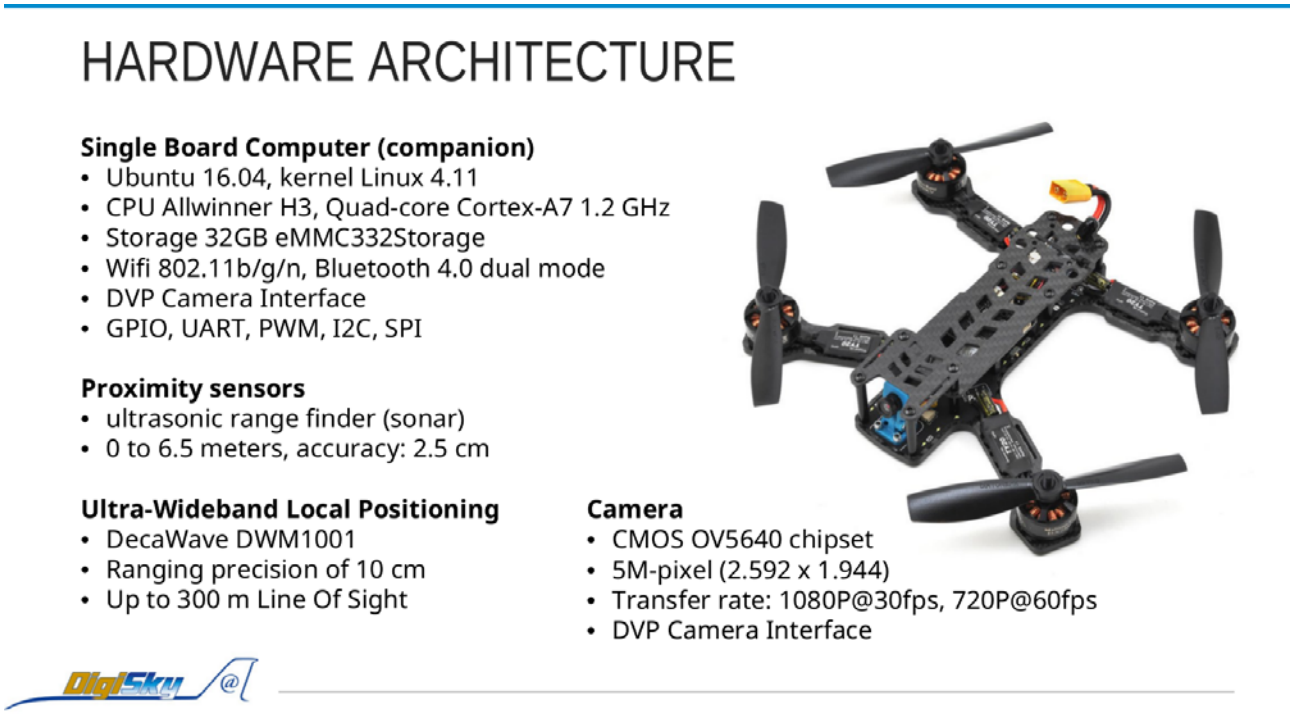

Figure 9 - SAR drones' hardware architecture

This architecture allows us to implement the following features:

- autonomous flight with dynamic waypoint and target location
- a complete offboard control via companion computer to send basic commands: takeoff, landing, movements, target position
- obstacle detection and avoidance using ultrasonic sensors in three directions: front, left, right
- visual target detection using monocular camera

- precise positioning using UWB module with an accuracy of 15 cm
- real-time telemetry based on Wi-Fi connection

6.1.1 MavROS/MAVLink Integration

In order to ensure easy integration of the different features of the drones, an operating system Ubuntu 16.04, ROS Kinetic and MavROS[14] are installed on the companion computer.

The communication layer between autopilot and companion computer is based on MAVLink[15] or Micro Air Vehicle Link, a protocol for communicating specific for small unmanned vehicle. It is designed as a header-only message marshalling library. MAVLink was first released early 2009 by Lorenz Meier under LGPL license. It is used mostly for communication between a Ground Control Station (GCS) and Unmanned vehicles, and in the inter-communication of the subsystem of the vehicle. It can be used to transmit commands, the orientation of the vehicle, its GPS location, speed etc.

MavROS is an extendable communication node for ROS of the MAVLink protocol. Thanks to its use, it is possible to integrate the swarming algorithm and the movement commands using simple ROS nodes. For example, in the CPSwarm/SAR package there are two main ROS nodes: "cpswarm_sar_uav" which provides the swarming behavior for the drones and "cpswarm_sar_ugv" which provides the swarming behavior for the rovers. Some topics are specifically developed to control robots in the scenario:

- <target_found>, <target_lost>, <target_rescued> related to the target status
- <range_right>, <range_front>, <range_left> for collision avoidance
- <arming>, <takeoff>, <landing>, <set_target_position> for the movement

6.1.2 Other functionalities needed for SAR scenario

There are two other important aspects that have been implemented in the ROS system for the SAR scenario:

- **Identification of the target though computer vision.**

  In SAR scenario, drones fly over the inspection area searching the target (during the demo we will use an AprilTag marker). We use a monocular camera equipped with a built-in ARM Cortex processor able to implement a complete computer vision system with a good frame-rate. The camera has a 5M-pixel (2.592x1.944) sensor with a CMOS OV7725 chipset. We developed a dedicated algorithm based on OpenCV for the marker detection and tracking, that sends target coordinates to the companion computer over a ROS node.

- **Collision avoidance system.**

  Based on 3 ultrasonic range finders rotated of 120° each of them, the system allows to identify a front or side obstacle starting from a distance of about 8 meters and to calculate the best trajectory/behavior to avoid collision. In a swarm configuration it is essential to have an effective system able to detect obstacles, considering that the flight of each drone is totally autonomous.

[14] http://wiki.ros.org/mavros
[15] https://mavlink.io/en/

6.2 Logistic Scenario

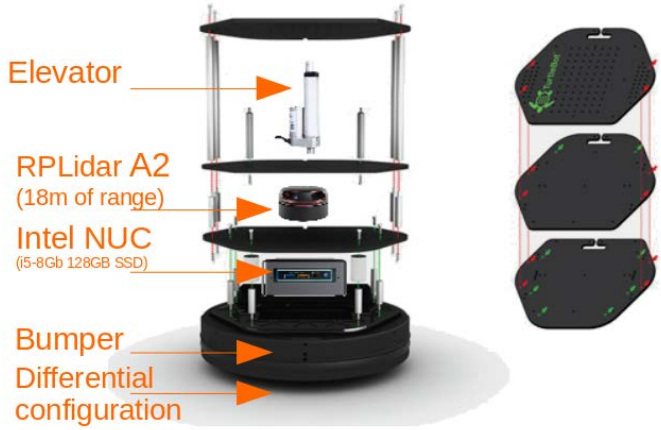

Figure 10 - Turtlebot architecture

For the logistic scenario 3 Turtlebots with elevators have been built from scratch. The robots have installed a RPLidar A2 with 18 meters of range, an Intel NUC with i5 processor, 8GB of RAM and a 128GB of SSD. The Figure 10 shows the components integrated with the robot.

Also, the light carts that robots have to pick have been constructed from four wooden legs and a methacrylate board as Figure 11 shows:

Figure 11 - Light cart

Related with the software, each robot has installed the operating system Ubuntu 16.04, ROS Kinetic, ROS navigation stack and Robotnik ROS packages.

In Robotnik ROS packages there are some methods developed specifically for the logistic scenario. They can be classified in two kinds: simple functionalities and complex functionalities.

6.2.1 Simple functionalities

Two basic functionalities related with the logistic environment have been developed:

6.2.1.1 Send robot below a cart

This method has been developed as a ROS action. It receives an input with the position where the cart is located. Once the action is called, the robot moves to the position of the cart and tries to find the legs of the

cart using the laser scan. Then, when it is found, the robot moves below the cart. The following figure shows this process:

Robotnik ROS packages:

1- Simple methods:

1.1- Send robot below a cart (by action via ROS, via MQTT, via REST).

Input: 2D pose of the cart (x, y, theta).

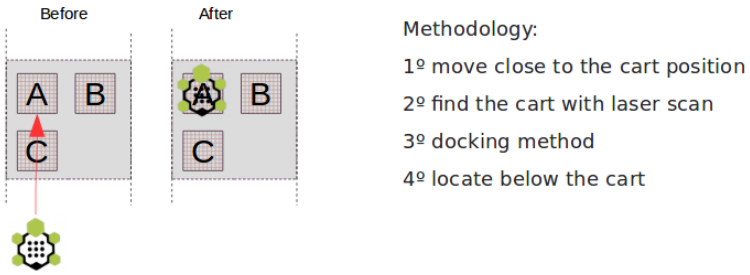

Figure 12 - Send robot below a cart

6.2.1.2 Move the elevator up or down

To give the robots the ability to pick things up, a linear motor has been integrated inside the software architecture. In this case, Robotnik ROS package offers another action to move the elevator up or down. The input of this ROS action is a boolean variable to indicate if the action is move up or move down. The Figure 13 shows the action:

1.2- Move the elevator up or down (an action via ROS, via MQTT, via REST).

Input: boolean.

Figure 13 - Move up or down the elevator

6.2.2 Complex functionalities

Other methods more complex have been developed to offer a more abstract view of the actions that robots can carry out. These complex functionalities are the concatenation of simple methods. These functionalities are:

- **Send a robot below a cart and move up the elevator**

  This action receives as an input the location of the cart. After calling this action the robot will move to the cart location and then will move up the elevator to pick the cart.

- **Send a robot to any place and move down the elevator**

  Literally it is the same previous action but oriented to leave what it has picked.

- **Send a robot below a cart, move up the elevator, send robot to another place and move down the elevator**

This action is the typically pick and place operation. It receives two inputs: the position where the cart has been picked and the position where the cart must be placed.

6.3 Automotive Scenario

In the framework of the CPSwarm project, since the autonomous vehicles should communicate with each other without wires while they run on the road, the challenge is to apply the know-how of the wired TTEthernet on a wireless environment. TTEthernet is a scalable technology and allows development of critical system parts according to fail-safe or fail-operational application requirements.

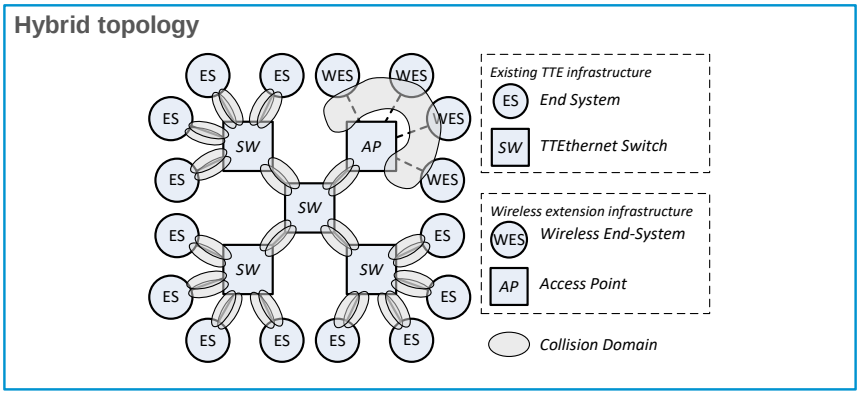

Figure 14 - TTEthernet topology

The main difference between the wired and wireless links is that wireless constitute a single collision domain when the stations are in range, whereas the wired links are full-duplex.

In order to support integration of applications with different real-time and safety requirements in a single network, TTEthernet supports three different traffic classes:

- time-triggered (TT) traffic - is sent in a time-triggered way, i.e. each TTEthernet sender node has a transmit schedule, and each TTE-Switch has a receive and forward schedule. This traffic is sent over the network with constant communication latency and small and bounded jitter.
- rate-constrained (RC) traffic - is sent with a bounded latency and jitter ensuring lossless communication. Each TTEthernet sender node gets a reserved bandwidth for transmitting messages with the RC traffic. No clock synchronization is required for RC message exchange.
- best-effort (BE) traffic - traffic with no timing guarantees. BE traffic class compatible with the IEEE 802.3 standard Ethernet traffic.

6.3.1 Challenges of the automotive scenario

1. Wireless communication

   The communication from the leading vehicle to the follower vehicles, and also among all the vehicles, must be mandatorily wireless since it is not possible to have a wire among vehicles when they are running in a realistic situation.

2. Real-Time communication

   Real-time communication is compulsory to give response to the safety requirements, for example, when breaking. Network communication technology must use time scheduling to bring deterministic real-time communication.

3. Low reliability communication

   Real circumstances like harsh weather conditions, obstacles or presence of other wireless signals may decrease the reliability of wireless transmissions and can compromise real-time communication requirements. Considering that the quality of the wireless channels varies with the time, frequency

and location, it is possible to increase reliability by finding better times, frequencies and locations to transmit and/or by performing retransmissions, while still observing deadlines.

6.3.2 Wireless Driver integration

The wireless driver under development will meet all the requirements of the car convoy scenario mentioned above. The driver will be based on the IEEE 802.11 standard. As of today, wireless drivers compliant with IEEE 802.11 make use of the listen before transmitting medium access control (MAC) principle in an attempt to avoid collisions between transmissions. However, this principle lacks real-time behaviour, since transmissions can be postponed indefinitely waiting for the wireless medium to be cleared.

Our approach to guarantee fairness in the transmissions between vehicles consists on scheduling the points in time when every vehicle is able to perform such transmissions, following a time-division multiple access (TDMA) approach. These instants are uniquely assigned to every vehicle so that collisions due to concurrent transmissions do not happen. For the schedule to be followed properly, a common time notion should exist between the vehicles. To solve this issue, a time synchronization protocol between the vehicles will be deployed.

Granting the access to the transmission medium is not enough for the transmissions to properly arrive the destination vehicle, since reliability might be compromised. A common way of increasing reliability is performing retransmissions, so that the same data is sent several times. Retransmissions can be performed at different points in time or use different frequencies or physical paths. Our first attempt will be to perform transmissions at different points in time while still considering deadlines, and evaluate how good the solution performs for the vehicle environment.

The abstraction library can interface the wireless deterministic driver via a specific API. This work is currently ongoing. The API will either include normal sockets and addition configuration through standard driver calls, or alternatively, it will use custom API that wraps the sockets in the same way as the TTEthernet. Both options have 2 different traffic classes: best effort and time-triggered.

6.3.3 Integrated Behaviours

In the automotive scenario, when part of the route is common for two or more vehicles, they can create a platoon responding to the swarm intelligence. When they run in a platoon, the leading vehicle has autonomous driving capability and prescribes the actions and decisions (i.e. navigation, decision on take-over manoeuvres, sequencing manoeuvres, lane change etc.) for the follow-up vehicles. The follow-up vehicles have autonomous driving capability and environmental awareness. They follow the leading vehicle´s actions.

The mission describes the goal of the vehicle, i.e., where is it and where it should end. The optimization requested will be done on the route needed to execute the mission with the least cost. When two or more vehicles are traveling behind each other the cost of the road is reduced by 20% making platooning as preferred solution. The goal of the optimization will be to find out the best route for every vehicle. Therefore, the behaviours will be:

- The shortest path algorithm for each vehicle, from start position to final destination, responding to evolved or swarm algorithm provided by the CPSwarm workbench.
- Join/leave the platoon, responding to evolved or predictive algorithm provided by the CPSwarm workbench.

Some vehicles might create a platoon while others will only follow the shortest path as fully autonomous vehicles. The vehicles that will create the platoon in the common route, will select their role (either leading vehicle or following vehicle) dynamically based on evolutionary algorithms.

When they run in a platoon, the vehicles will be running with a given speed and with a given distance among them. The behaviours respond to situations of the real life, for example, when an accident occurs in the trajectory of the vehicles. In such a case, they are able to take decisions and the behaviours will be:

- Situation 1: Lane change.

The road has multiple lanes and the leading vehicle changes the trajectory to the next lane on the left. The follow-up vehicles follow the leading vehicle always keeping the platoon configuration. There is no speed change. This is an event sent by the leading vehicle.

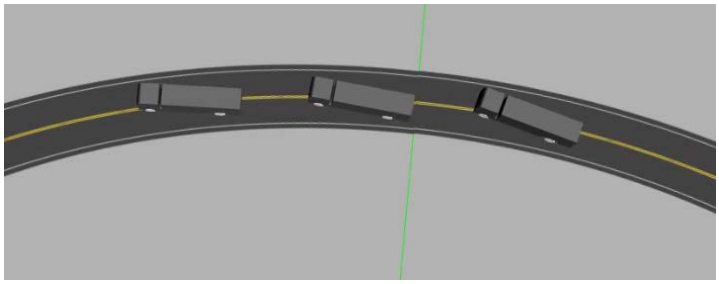

Figure 15 - Platoon configuration

- Situation 2: Emergency breaking.

  The road has only one lane and the leading vehicle reacts to the obstacle on the road by breaking until it completely stops. The follow-up vehicles break after the leading vehicle without collision. They must stop by keeping a minimum "safety" distance among them. The event is sent by the leading vehicles, instructing all vehicles to break.

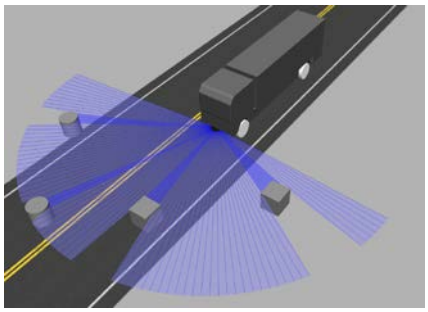

Figure 16 - Obstacles detection for emergency breaking

7 Conclusions

This deliverable has presented the work done in Task 7.1 to design and implement the initial version of the CPSwarm Abstraction Library. As this task will continue till M32, the Library will be further enriched with new components in order to support other hardware technologies (sensors and actuators) and to provide new high-level functionalities needed for the final implementation of the use case scenarios. The new modification will be documented in later deliverable D7.2, *Final CPSwarm Abstraction Library*.

Acronyms

Acronym	Explanation
API	Application Programming Interface
CPS	Cyber Physical System
SCARA	Selective Compliance Assembly Robot Arm
ROS	Robot Operating System
SOEnvO	Simulation and Optimization Environment Orchestrator
MAVLink	Micro Air Vehicle Link
OROCOS	Open RObot Control Software
ICE	Internet Communication Engine
CORBA	Common Object Request Broker Architecture
GPS	Global Positioning System
IMU	Inertial Measurement Unit
AL	Abstraction Library
FSM	Finite State Machine
HFSM	Hierarchical Finite State Machines
GPL	General Public License
UML	Unified Modeling Language
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-process Communication
RAM	Random Access Memory
SSD	Solid-state Drive
MQTT	Message Queue Telemetry Transport
REST	Representational State Transfer

List of figures

References

[1] Ceriani, S.; Miglianvacca, M. Middleware in Robotics. Internal Report for "Advanced Methods of Information Technology for Autonomous Robotics".

Available online: http://home.deib.polimi.it/gini/AdvancedRobotics/docs/CerianiMigliavacca.pdf

[2] Mohamed, N., Al-Jaroodi, J. and Jawhar, I. Middleware for Robotics: A Survey. 2008 IEEE Conference on Robotics, Automation and Mechatronics.

[3] Elkady, Ayssam & Sobh, Tarek. (2012). Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. Journal of Robotics. 2012. 10.1155/2012/959013.

[4] Makarenko, A., Brooks, A., Kaupp, T.: Orca: Components for Robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006), Workshop on Robotic Standardization (2006).

[5] Markus Rickert and Andre Gaschler. Robotics Library: An object-oriented approach to robot applications. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 733-740, Vancouver, BC, Canada, Sep. 2017.