Grant Agreement No. 611373

FP7-ICT-2013-10

D3.3. A Petri net-based online mission replanner with diver symbol inputs

Due date of deliverable: 30/06/2016
Actual submission date: 30/06/2016

Start date of project: 01 January 2014 Duration: 36 months

Organization name of lead contractor for this deliverable: CNR

Version 1

Dissemination level		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contents

1 Outline of the deliverable

This deliverable describes the design, implementation and validation of a Petri net-based high level mission control layer for the robotic agents, devoted to the planning, execution and supervision of the cooperative action that have to be undertaken by the robotic platforms in order to support diver operations. The deliverable is mostly related to the results obtained in WT3.3.

The deliverable reports the general description of the mission controller architecture and its main characteristics with respect of state tracking and inter-task conflict resolution. Then the customization of the architecture with respect of the envisioned validation experiment is described, as well as the integration phase with the robotic platforms’ control systems and gesture recognition modules.

2 Mission controller architecture

The CADDY mission control system is a modular framework designed and developed with the aim of managing the state tracking, task activations and reference generation that fulfills the requirements in order to support the diver operations.
As depicted in Fig. 1, the mission controller acts as a cognitive planning and supervision system taking as input the decoded gesture sequence executed by the diver, in turn generating proper task activation actions in such a way to trigger the required capabilities needed to provide the requested support.

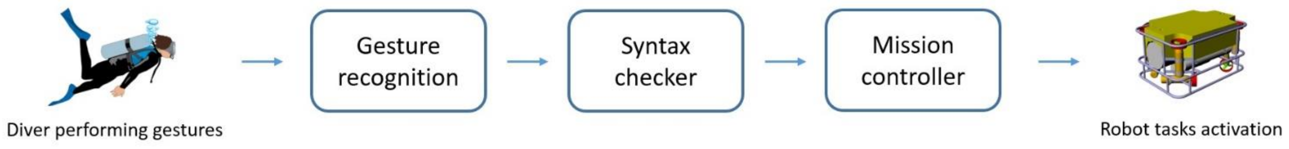

Fig.1. Main modules of high-level CADDY functionalities

The main goal of the mission control module is to abstract the robotic capabilities at logical level and manage the activation of the real robotic primitives, while at the same time resolving possible conflicts that may arise when selecting the robotic capabilities to be triggered.
In the CADDY logical framework, three classes of executable units have been identified and shown in Fig. 2:
functional primitives represent the macro-actions that the robotic platform has to carry out in order to support the diver operation and that are strictly related to the current functional mode (slave, guide, observer);
high-level logical tasks are the interface between the primitives and the operative task provided by robot. This logical task set is common in the overall architecture and will provide the required functionalities activating the proper low-level tasks that are currently made available by the employed robotic platform;
low-level robotic tasks are the actual implemented autonomous functionalities on the target robot, e.g. speed regulators, heading and depth controller, etc. Depending on the low-level task availability, the CADDY compliant mission control system will properly select which high-level functionalities can be activated allowing, in turn, the enabling of the required primitives to fulfil the mission operations.

The primitives are linked to the support action that the diver can require by means of the gesture-based language: once a gesture or a complex sequence of gestures is recognized and validated, it is sent to the mission controller that will activate the proper primitive to start the support operation.
Each primitives activates as set of high-level tasks that represent the logical functionalities required to fulfill the required operation.
The high-level logical tasks activate in turn a set of robotic tasks which enable and execute the physical operations on the real robot devoted to the support of the diver operations.

As an example, Fig. 3 reports the case of activation of the "Follow me" primitive; such a primitive requires the system to turn on the following functionalities: "go_to_depth" to reach and maintain a desired depth (i.e. the same of the diver); "go_to_2D_point_fa" is the procedure to track a 2D point (the diver position) for fully-actuated (fa) platforms, that generates proper horizontal velocities for point tracking; "turn_towards" enabling the auto-heading capability to always look towards the diver.
In turn, each of the high-level task has to be linked with one or more low-level tasks in order to physically execute the required actions:

- "go_to_depth" requires the activation of a depth_controller;
- "go_to_2D_point_fa" requires the activation of surge and sway velocity regulators;
- "turn_towards" enables the auto-heading controller.

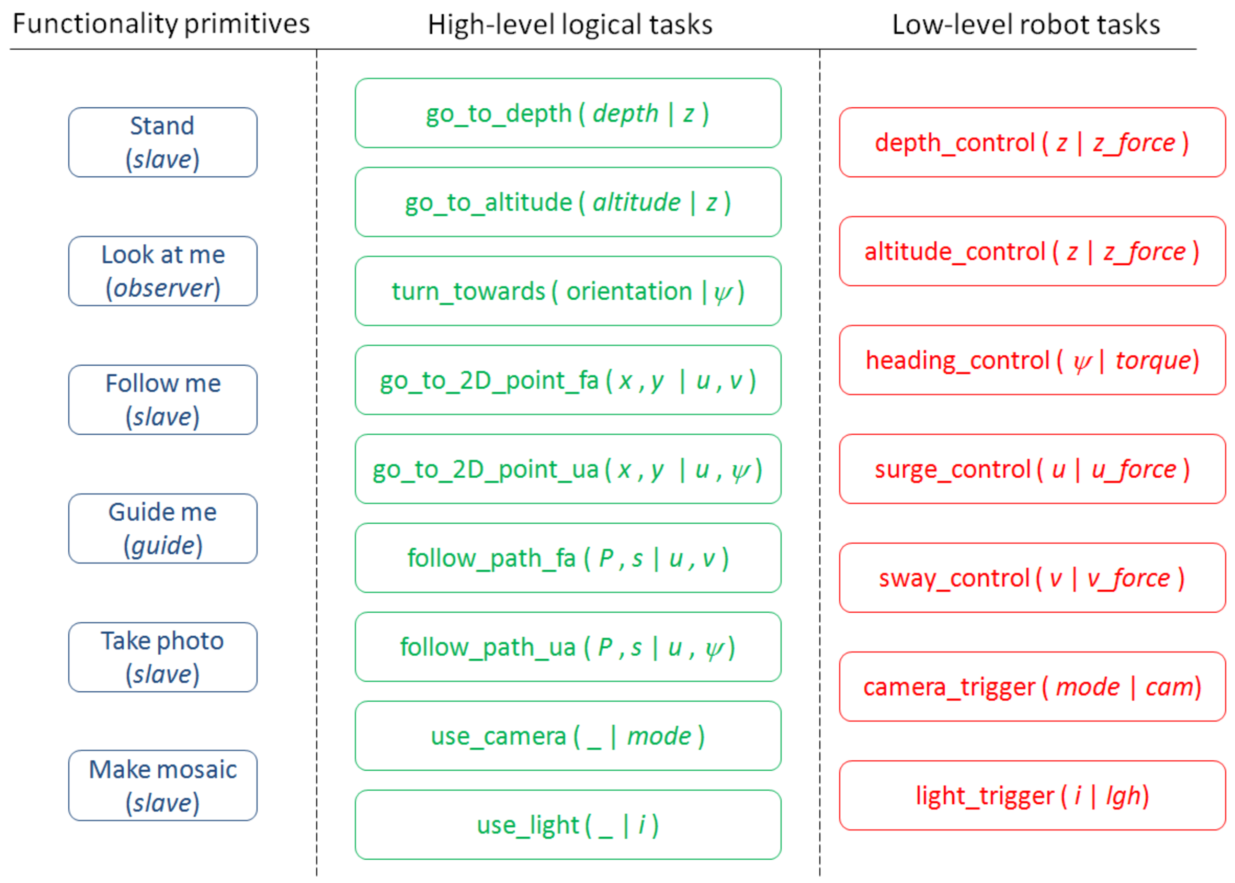
Functionality primitives
High-level logical tasks
Low-level robot tasks
Stand (*slave*)
Look at me (*observer*)
Follow me (*slave*)
Guide me (*guide*)
Take photo (*slave*)
Make mosaic (*slave*)
go_to_depth (*depth* | *z*)
go_to_altitude (*altitude* | *z*)
turn_towards (orientation | *ψ*)
go_to_2D_point_fa (*x* , *y* | *u* , *v*)
go_to_2D_point_ua (*x* , *y* | *u* , *ψ*)
follow_path_fa (*P* , *s* | *u* , *v*)
follow_path_ua (*P* , *s* | *u* , *ψ*)
use_camera (_ | *mode*)
use_light (_ | *i*)
depth_control (*z* | *z_force*)
altitude_control (*z* | *z_force*)
heading_control (*ψ* | *torque*)
surge_control (*u* | *u_force*)
sway_control (*v* | *v_force*)
camera_trigger (*mode* | *cam*)
light_trigger (*i* | *lgh*)

Fig. 2. Task classification in the CADDY logical framework

The logical links between the high- and low-level layers are set on the basis of the input/output variables: the output generated by a high-level task is the input for one or more low-level tasks, e.g. "go_to_2D_point_fa" generates the *u* and *v* speed reference signals that feed the low-level speed controllers.

If, as a second exemplificative case, an under-actuated platform is employed, it implies in turn that the "sway_speed" controller is not available (due to the under-actuation, e.g. of a rudder based vehicle). The unavailability of this latter low-level task reflects on the inhibition of the "go_to_2D_point_fa". Anyway to fulfil the "follow me" primitive requirements, the system can automatically swith to the "go_to_2D_point_ua" that can drive the robotic platform towards the desired point generating proper surge velocity and heading signals. The activation of the "go_to_2D_point_ua" task goes in conflict with the "turn_towards" one, given the generation of the ψ reference signals by both the tasks. Detecting this logical conflict, as depicted in Fig. 4, the system deactivates the execution of the "turn_towards" (that, by user definition, has a lower priority with respect to the "go_to_2D_point_ua" in relation to the "follow me" primitive).

For the automatic selection, activation and inter-task conflict management, a Petri net based execution control system has been developed. The system is configured by means of a set of configuration files that specify, on one side, the capabilities of the robot in terms of autonomous tasks and, on the other side, the set of high level functionalities that the CADDY system has to provide for the diver support. A real-time Petri net engine models the logical interconnections among the tasks and primitives and, depending on the specific actions commanded by the diver, automatically handle the activation/deactivation of the proper task sets.

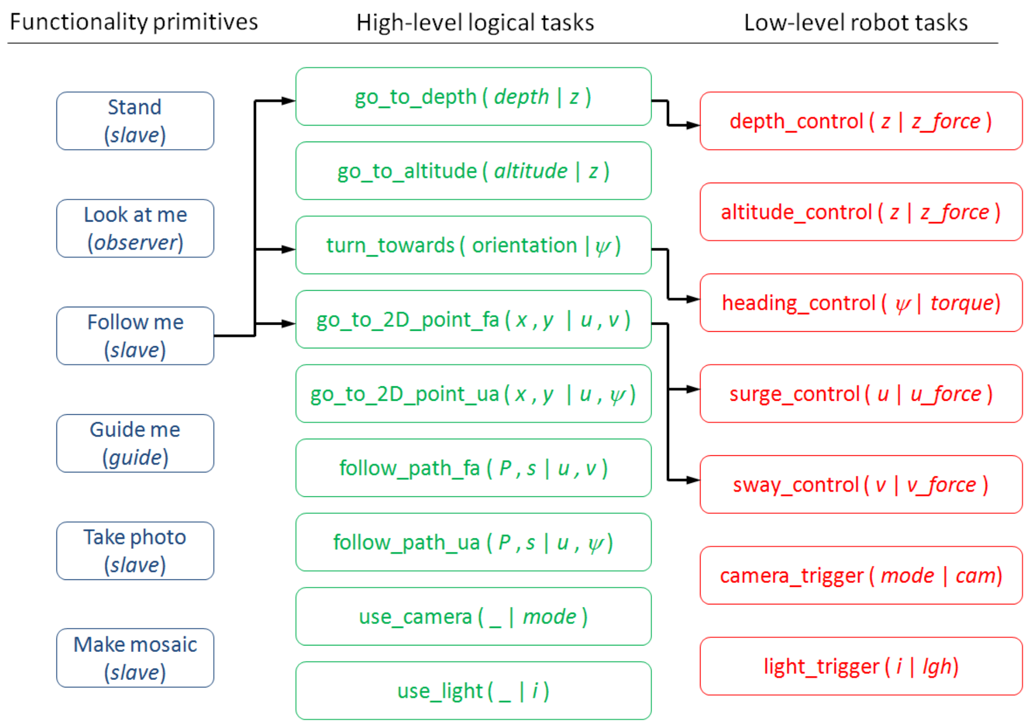
Functionality primitives
High-level logical tasks
Low-level robot tasks
Stand (slave)
Look at me (observer)
Follow me (slave)
Guide me (guide)
Take photo (slave)
Make mosaic (slave)
go_to_depth (depth | z)
go_to_altitude (altitude | z)
turn_towards (orientation | ψ)
go_to_2D_point_fa (x , y | u , v)
go_to_2D_point_ua (x , y | u , ψ)
follow_path_fa (P , s | u , v)
follow_path_ua (P , s | u , ψ)
use_camera (_ | mode)
use_light (_ | i)
depth_control (z | z_force)
altitude_control (z | z_force)
heading_control (ψ | torque)
surge_control (u | u_force)
sway_control (v | v_force)
camera_trigger (mode | cam)
light_trigger (i | lgh)

Fig. 3. Example of "follow me" primitive activation for fully-actuated robot

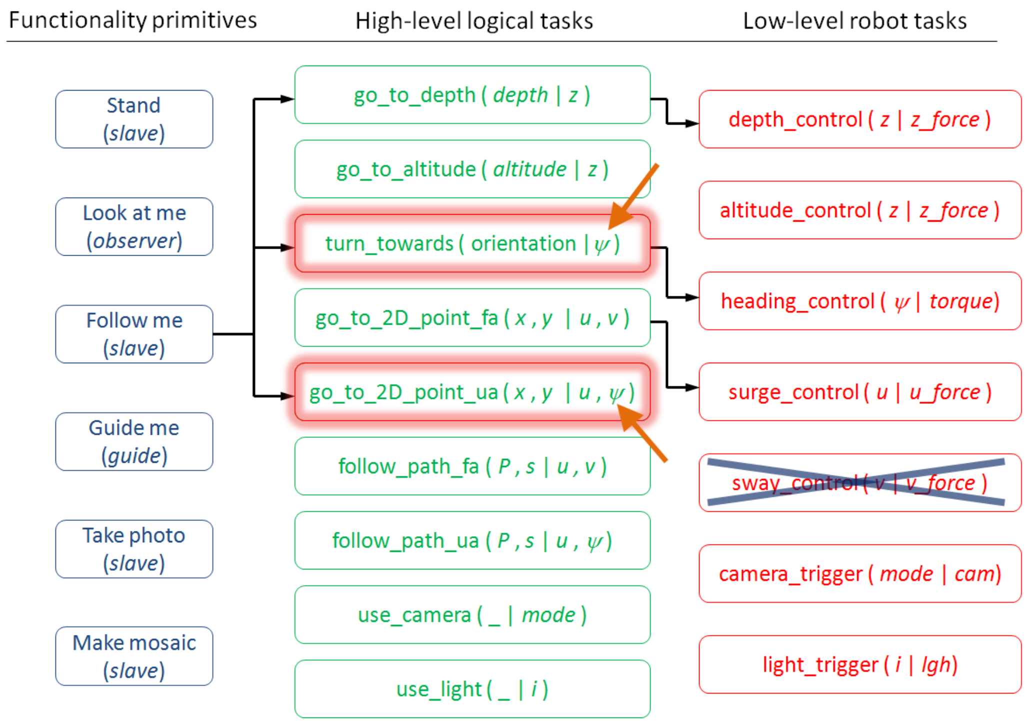
Functionality primitives
High-level logical tasks
Low-level robot tasks
Stand (slave)
Look at me (observer)
Follow me (slave)
Guide me (guide)
Take photo (slave)
Make mosaic (slave)
go_to_depth (depth | z)
go_to_altitude (altitude | z)
turn_towards (orientation | ψ)
go_to_2D_point_fa (x , y | u , v)
go_to_2D_point_ua (x , y | u , ψ)
follow_path_fa (P , s | u , v)
follow_path_ua (P , s | u , ψ)
use_camera (_ | mode)
use_light (_ | i)
depth_control (z | z_force)
altitude_control (z | z_force)
heading_control (ψ | torque)
surge_control (u | u_force)
~~sway_control (v | v_force)~~
camera_trigger (mode | cam)
light_trigger (i | lgh)

Fig. 4. Example of "follow me" primitive activation for under-actuated robot and inter-task conflict management

Each mission controller related task is logically represented by a simple Petri Net which define the state of activation of the task. As shown in Fig. 5, two *places* represent the two state of the task: *idle* and *running*. Two other places represent the start and stop events, while two *transitions* define the task net evolution rules. The idle place is initially marked, representing an initial condition of deactivated task.

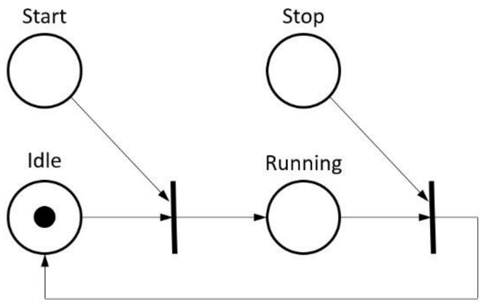

Fig. 5. Basic task representation in the Petri Net based mission controller

The Petri Nets representing the tasks are then connected through suitable arcs joining the *running* state of a logical layer to the *start* place of the following layer, as depicted in Fig.6. In such a way a cascade activation of the task chain is carried out.

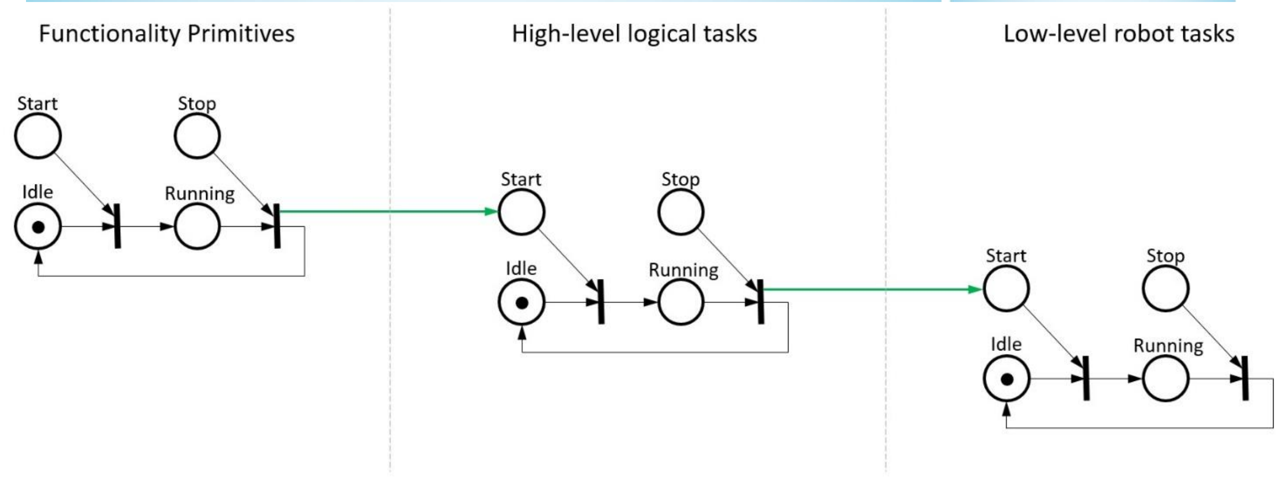

Fig. 6. Cascade-layer tasks connection

With such a proposed scheme, two main issues remain to be handled: the mutual exclusive activation of the functionality primitives and the conflict management in high- and low-level tasks execution.
For the issue of primitives mutual exclusion it is sufficient to insert an additional control place at primitives level, in such a way that only one single primitive can switch to running state, as shown in Fig. 6.
The inter-task conflict issue is raised from the fact that different tasks can generate output for the same output variable; the abovementioned example shows that the activation of the two high-level tasks “turn_towards” and “go_to_2D_point_ua” bring to a conflict due to the generation of the same output variable “ψ”. To resolve the conflicts, a constraint place is added between two or more conflict tasks whenever a same output variable is shared among those tasks (see Fig. 6). The presence of this new constraint place allows the mutual exclusion of the conflicting task execution and providing a consistent state of the mission net.

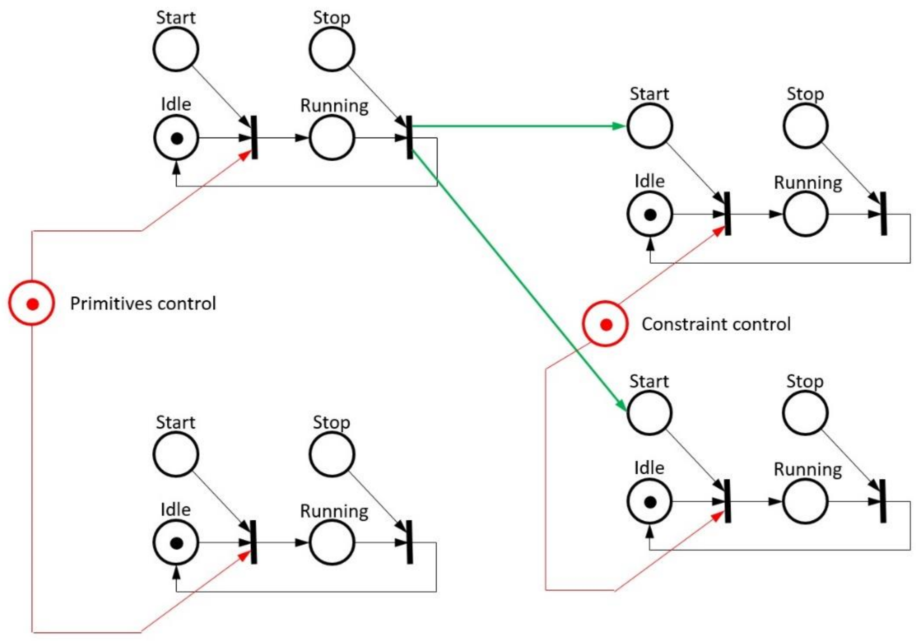

Fig. 6. Mutual exclusive and conflict-free mission control Petri Net

3 Validation experiment framework

The envisioned final validation experiment sets the goal of demonstrating the main advanced capabilities in supporting the diver operations through the exploitation of a highly autonomous robotic system.
With respect to the CADDY framework and provided functionalities, the experiment is composed by the following three main phases:
the robotic system, in guide mode, navigates at constant distance from the diver performing observation activity. From the ground station, a new point of interest is fed to the robotic team in order to guide the diver to the requested position. Performing the "pointer experiment", the underwater buddy starts guiding the diver towards the selected point;
the diver stops its motion towards the point and initiates a gesture-based communication with the buddy requiring the robot to stop the current guide activity and start a "go to boat and carry equipment" action. In this phase the underwater robot will surface and move to a specific base location in order to collect some specific equipment unit and then returning at the operational point close to the diver;
the guide function towards the point of interest is recovered and carried out until the desired location is reached by the diver.

Given the envisioned experiment plan, the functionalities required are "guide_me" and "go_ and_carry"; if needed, two additional primitives can be added to complement the functionality set: "idle" (no action is executed) and "emergency" (to trigger the robot to signal a problem or to immediately surface).

The defined primitives are linked to a set of logical tasks that indicates the activation of the related high-level modules which allow the execution of the required actions. In particular the following tasks are selected:
"go_to_fa" – 2D position tracking in full-actuated configuration;
"go_to_ua" - 2D position tracking in under-actuated configuration;
"pointer" – the control scheme to perform the pointer experiment
"go_to_depth" – automatic control for desired depth maintenance
"turn_towards" – auto-heading controller

A set of low-level robot tasks are also defined and triggered by the activation of the high-level ones. If needed, they can activate specific modules or procedure on-board the actual robot.
For the specific case, the following robot tasks are defined: "depth_control", "heading_control", "surge_control", "sway_control"," x_control", "y_control".
The activation of the tasks is automatically triggered by the mission control Petri net that, on the basis of the requested action, computes the correct sequence of task execution.
The activation of the primitives can be commanded through the topics:

/buddy/mission_controller/command/<primitive_name>

Setting the value: 0 -> stop , 1 -> start

The mission control Petri net computes the task activation sequence and the result is published into the following topics (the values are again: 0 -> stop , 1 -> start):

/buddy/mission_controller/primitives/<primitive_name>

/buddy/mission_controller/high_level_tasks/<high_level_task_name>
/buddy/mission_controller/robot_tasks/<robot_task_name>

Proper reference signals have to be generated and consumed by different modules.

"guide_me" mode - A continuous generation of the references for position, orientation and depth for the buddy has to be carried out.
The depth reference will be always the same as the diver depth.
The position reference of buddy is function of the current diver position & orientation and it is computed with respect to the desired point of interest to be reached.
Orientation reference is computed with respect to the current position of diver, in order to orient buddy always to face the diver.

"go_and_carry" mode - A set of action has to be sequentially executed.

1. Buddy sets position keeping to maintain current 2D position;
2. Buddy sets autoheading to set a desired orientation;
3. Buddy sets depth reference to zero and surfaces;
4. Buddy switches to wifi comm mode in order to maintain connection with MedusaS and ground station;
5. Position reference is changed to "virtual boat" location;
6. As the "virtual boat" location is reached within a specified threshold, buddy waits for a "go back" signal (we can command the action through a specific topic, that can be triggered by a timer or by human notification);
7. Go back to starting 2D position;
8. As the initial position is reached, "guide_me" mode is triggered and buddy goes to correct depth, orientation and position in order to continue the guide operation of the diver towards the desired location.

"idle" mode – no action, no references
"emergency" - signaling alarm state

In the following, Table 1 resumes the functional modes, running modules, topics published and consumed.

Mode:	Running module:	Subscribed to:	Published:
guide_me	mission_controller	/caddy/mission_controller/command/<primitive_name>	/caddy/mission_controller/high_level_tasks/<high_level_task_name>
			/caddy/mission_controller/robot_task/<robot_task_name>
	pointer_experiment	/caddy/mission_controller/high_level_tasks/pointer	/caddy/pose_req
		/caddy/buddy/position_filt_ekf	/caddy/speed_req
		/caddy/diver/position_filt_ekf	
go_and_carry	mission_controller	/caddy/mission_controller/command/<primitive_name>	/caddy/mission_controller/high_level_tasks/<high_level_task_name>
		/caddy/buddy/position_filt_ekf	/caddy/mission_controller/robot_task/<robot_task_name>
	re-surface	/caddy/buddy/position_filt_ekf	/caddy/pose_req
	go-to-point	/caddy/buddy/position_filt_ekf	/caddy/pose_req
			/caddy/speed_req
	pointer_experiment	/caddy/mission_controller/high_level_tasks/pointer	/caddy/pose_req
		/caddy/buddy/position_filt_ekf	/caddy/speed_req
		/caddy/diver/position_filt_ekf	
mosaicking	mission_controller		/caddy/follow_section/enable
	follow_section	/caddy/buddy/position_filt_ekf	/caddy/pose_req
			/caddy/speed_req
idle	mission_controller	/caddy/mission_controller/command/<primitive_name>	/caddy/mission_controller/high_level_tasks/<high_level_task_name>
			/caddy/mission_controller/robot_task/<robot_task_name>
emergency	mission_controller	/caddy/mission_controller/command/<primitive_name>	
			/caddy/mission_controller/high_level_tasks/<high_level_task_name>
			/caddy/mission_controller/robot_task/<robot_task_name>

Table.1. Modules and topics specification

4 Results of the integration workshop

During an integration workshop, carried out in the period 2016 May 16-20 in Zagreb (Croatia), the mission controller architecture has been integrated and tested within the surface and underwater control systems and the gesture recognition modules.

The integration of the mission controller with the vehicle architectures was focused on the interconnection between the logical layer of the mission control module with the execution level of the specific vehicles (MedusaS and Buddy) in order to trigger the execution of the specific guidance and control tasks in function of the selected functional mode.
Moreover, having two separate mission controllers running onboard the surface and underwater platforms, a synchronization mechanism has to be provided in such a way to maintain a consistent state between the two mission modules. To solve this issue, a suitable latch-logic is coded into the mission controller and triggered via specific information dispatch through the communication layer.
Furthermore, during specific actions like the surfacing of the underwater platform, the acoustic tracking behavior of the surface vehicle has to be dynamically changed (e.g. from buddy + diver tracking to diver-only tracking). Such issue is solved by the mission controller generating proper flags in function of the specific functional state.

On the other side, the mission control module of the underwater platform is connected to the gesture recognition and interpretation modules. The recognized, interpreted and validated gestures or complex sequences are forwarded to the mission controller that in turn activates the proper functional modalities to comply with the diver requests.

The overall integration of the mission controller architecture was successfully tested during different experiments in pool environment during the integration workshop.

5 Conclusions

This deliverable has mainly described the development activity carried out during the WT3.3 and its related successful results.
A flexible and modular mission control architecture has been developed and integrated with different robotic platforms. The mission controller is capable of consistently change the set of running tasks in order to comply with the online diver requests that are issued via hand gestures.
The overall framework has been customized and tested in pool environment for the envisioned validation experiment scheduled for the next trials.