SPRING

GRANT AGREEMENT N. 871245

Deliverable D7.3

Initial Software Architecture for SPRING-ARI

Due Date: 31/08/2021
Main Author: Sara Cooper
Contributors: Séverin Lemaignan, Anaël Le Bihan, Luca Marchionni
Dissemination: Public Deliverable

This project has received funding from the European Union's Horizon 2020 Research and Innovation

Programme under Grant Agreement No. 871245.

DOCUMENT FACTSHEET

Deliverable no.	Annex to D7.3: Initial Software Architecture for SPRING-ARI
Responsible Partner	PAL
Work Package	WP7: WP Robot Customisation and Software Integration
Task	T7.3: Preliminary Software Integration Cycle
Version & Date	V4, 29/06/2022
Dissemination level	[X] PU (public) [ ] CO (confidential)

CONTRIBUTORS AND HISTORY

Version	**Editor**	**Date**	**Change Log**
1	PAL	06/08/2021	First Draft
2	PAL	07/09/2021	Partner contributions added
3	PAL	07/06/2022	Revision based on Reviewers feedback
4	PAL	29/06/2022	Revision after external reviewers request

APPROVALS

Authors/editors	PAL: Sara Cooper, Luca Marchionni, Séverin Lemaignan
Task Leader	PAL
WP Leader	PAL

TABLE OF CONTENTS

Executive Summary

Deliverable 7.3 concludes Task 7.3 on preliminary software integration cycle. The goal of this task is to perform the preliminary integration cycle that corresponds to T1.3, providing a basic system with the first development stage of all building blocks.

The document will explain:

- SPRING-ARI robot brief software architecture
- Preliminary SPRING-ARI API and modules, highlighting current status
- Coding and integration guidelines, indicating the SPRING Gitlab repository, provided dockers, documentation, etc.
- Continuous integration system and unit tests procedures

Contents of Deliverable

The aim of this deliverable is to provide a first version of the software architecture of the SPRING-ARI Robot, considering the user-cases defined as part of WP1, and the modules being developed between WP2 to WP6.

Specifically, PAL has:

- Provided coding and integration guidelines to partners, specifically regarding the ISO of the robotic platform
- Set up and maintain a continuous integration system, as well as develop software tools to automate as much as possible the software integration
- Coordinated and supervised the initial integration cycle

The other partners of the consortium have integrated their respective modules and applications.

To this end, this report is a high-level description of the main modules of the SPRING-ARI robot, and refers to the associated source code repository where the work has been carried out. This task is followed by Task 7.4 and Task 7.5, which aim to achieve an intermediate and complete software integration of the system, respectively.

The document is structured as follows. In Section 1 the General Architecture of the SPRING-ARI Robot, Section 2 the SPRING-ARI Modules and API. In Section 3 the quality assurance of the software will be highlighted, including the software and continuous integration system.
Finally, conclusions are presented in Section 4.

1.SPRING-ARI General Architecture

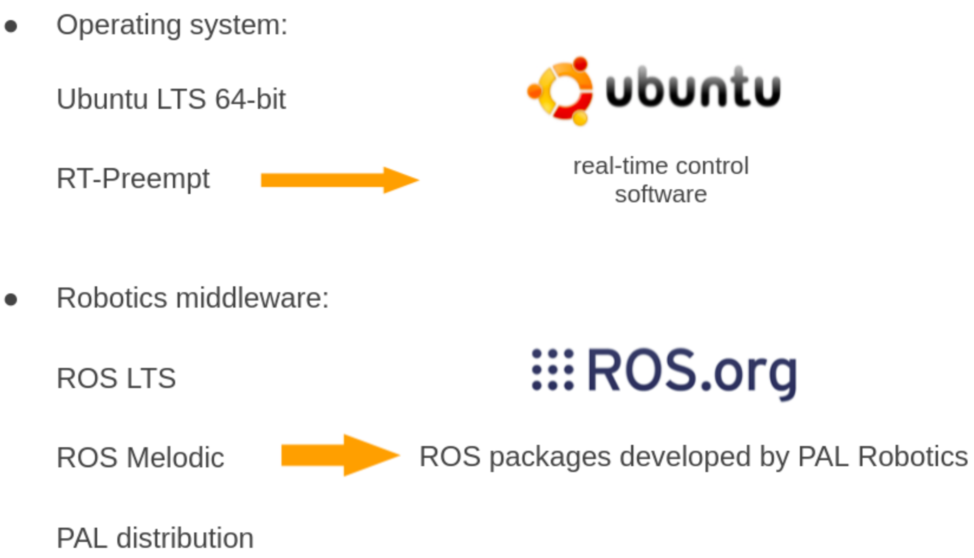
The SPRING-ARI robot has a component-based architecture that instantiates the robotic application by encapsulating different functionalities. Before explaining the different modules, the core software architecture of ARI will be outlined.

ARI Robot Software Architecture

The software architecture of the onboard computer of ARI has been designed in order to accommodate:

- Third party OS, middlewares and software packages
- PAL proprietary software and ROS packages
- Developer software and ROS packages

Figure 1 presents the software that is installed out-of-the-box in the robot.



- Operating system:

  Ubuntu LTS 64-bit

  RT-Preempt → real-time control software

- Robotics middleware:

  ROS LTS

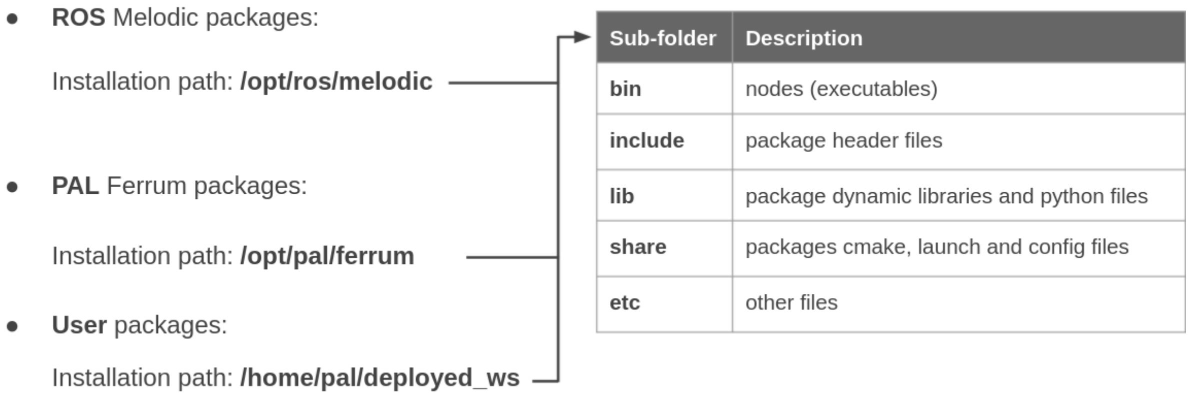
  ROS Melodic → ROS packages developed by PAL Robotics

  PAL distribution

Figure 1: Off-the-shelf software of the robot

All the software referred to in Figure 1 is read-only protected in the SSD of the onboard computer of the robot in order to prevent corruption of the core system. More precisely, any change done in the system out of the **/home/pal** folder will not be persistent after a computer reboot. Nevertheless, a procedure is provided in order to do system modifications in a persistent way so that partners will be able to install their own software packages.

Regarding the ROS software packages, three levels are defined in the robot computer, which are shown in Figure 2.



- **ROS** Melodic packages:

  Installation path: **/opt/ros/melodic**

- **PAL** Ferrum packages:

  Installation path: **/opt/pal/ferrum**

- **User** packages:

  Installation path: **/home/pal/deployed_ws**

Sub-folder	Description
bin	nodes (executables)
include	package header files
lib	package dynamic libraries and python files
share	packages cmake, launch and config files
etc	other files

Figure 2: ROS software in the robot

Next a brief description of some of the most relevant ARI components are described, on top of which the SPRING-ARI software modules are developed. For more details please refer to the ARI Robot Manual, shared with all partners as part of D7.2: http://docs.pal-robotics.com/manuals/0.0.x/manuals/ari/lang/en_GB/index_ari_manual.html, or the ARI training videos accessible as part of SPRING-ARI robot training: https://youtu.be/2QlUOMy6nPk

1. Mapping, localization and autonomous navigation

ARI is provided with a customised ROS navigation stack, where it uses its front torso RGB-D camera to map the environment using Visual SLAM. Once a map is built, it navigates autonomously using particle-filter based localization and motion planning, that considers dynamic obstacles detected. This is relevant mainly for WP2 modules. Fig 3 shows the navigation architecture of the ARI robot:

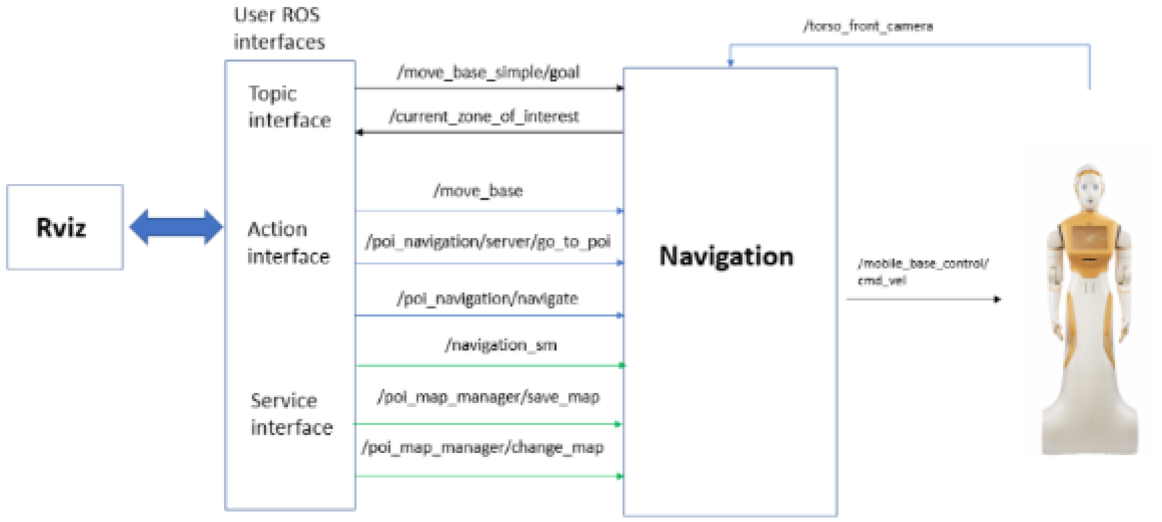


Figure 3: ARI Robot Navigation architecture using ROS

As can be seen, the user can communicate with the navigation software using ROS (Robotics Operating System: http://wiki.ros.org/) actions and services.

In order to visualise the mapping and localization pipeline using Rviz, ROS visual interface, the ARI robot has a customised Rviz interface. With this it is possible to build and localise within a map using ORB SLAM (https://github.com/raulmur/ORB_SLAM2), and then send navigation goals. Advanced options are offered such as creating Points of Interests for the robot to move to, Virtual Obstacles e.g. areas to avoid, etc. (Figure 4).

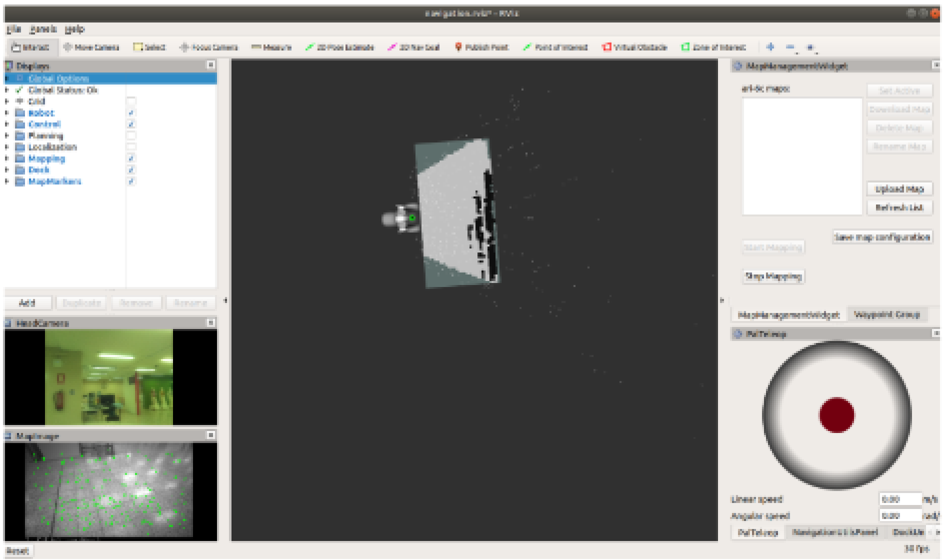


Figure 4: ARI Robot Navigation Rviz Map Editor

2. Perception

The cameras of the SPRING-ARI robot are described in D7.2. Figure 5 illustrates some of the camera outputs of the robot, such as the front RGB-D camera of the torso, its pointcloud, and the back fisheye camera. This section is relevant for WP2, WP3, WP4, WP5 and WP6 modules involving perception.

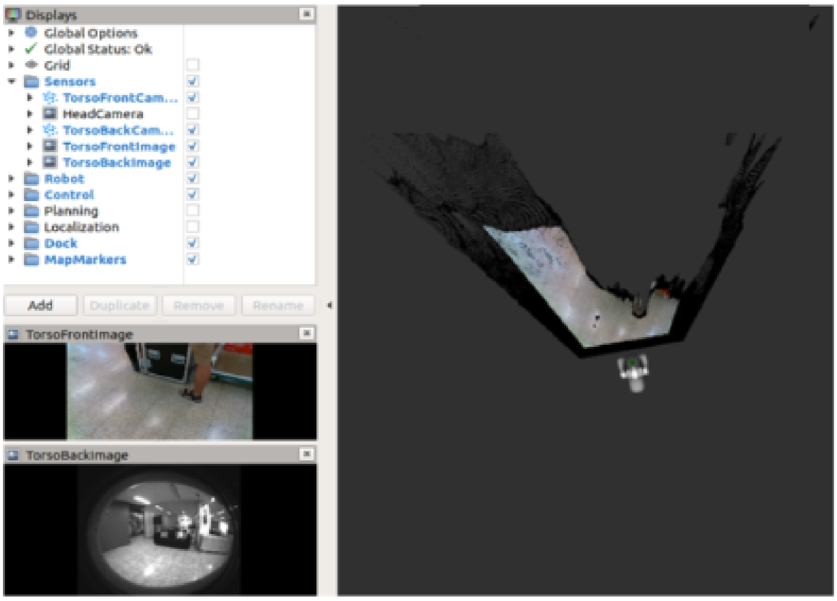


Figure 5: SPRING-ARI Robot camera outputs

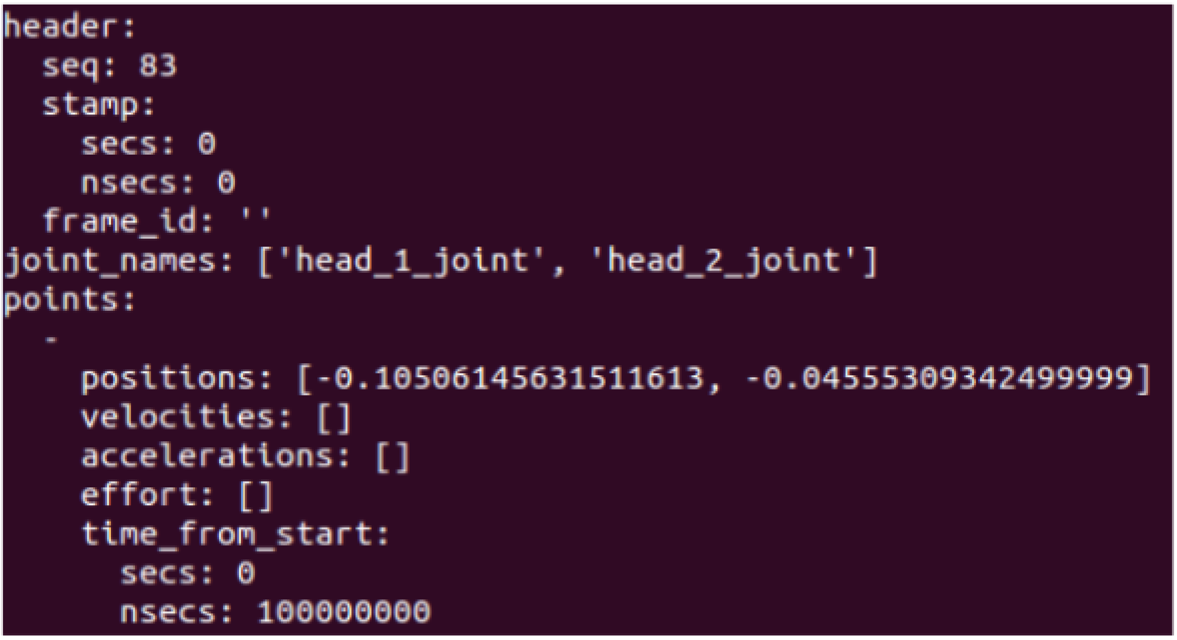
3. Low-level control system

The SPRING-ARI has 14 degrees of freedom, including 4 joints in each arm, 1 in each hand, 2 in the head, and 2 for the base. For their control it makes use of ros_control (http://wiki.ros.org/ros_control)

In total, the controllers that can be moved are:

- arm_left_controller: 4 joints
- arm_right_controller: 4 joints
- hand_left_controller: 1 joint
- hand_right_controller: 1 joint
- head_controller: 2 joints

For instance, Figure 6 shows how the joint_trajectory_controller package can be used to control the head position, an open source ROS package that takes as input joint space trajectories and executes them.

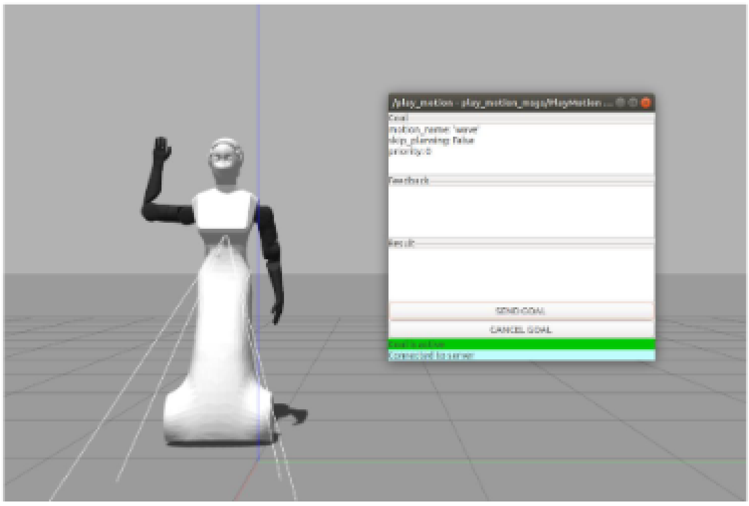


```
header:
  seq: 83
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
joint_names: ['head_1_joint', 'head_2_joint']
points:
  -
    positions: [-0.10506145631511613, -0.045553093424999999]
    velocities: []
    accelerations: []
    effort: []
    time_from_start:
      secs: 0
      nsecs: 100000000
```

Figure 6: Publishing head positions to the head_controller using the joint_trajectory_controller

Additionally the arm controller provides a safe version, which performs self collision check before executing each trajectory increasing safety of the robot motions.

Additionally, for non-verbal behaviour generation such as producing new movements for the arms, for instance useful for modules developed in WP6, ARI comes with the play_motion_builder package that allows to create new motions that the play_motion package can then execute. Some available gestures it can do right now include waving, shaking hands, or bowing the head.



motion_name: 'wave'
skip_planning: false
priority: 0

Goal is active
Connected to server

Figure 7: Play motion package to play pre defined motions

4. Text to speech

The ARI robot uses ACAPELA (https://www.acapela-group.com/) to produce speech, with the option to install any of its available voices. In order to trigger it a ROS Action server is available named /tts. Figure below shows an example of how to send action goal to the server to make the robot talk saying the sentence “Hello world” in English:

```
rostopic pub /tts/goal pal_interaction_msgs/TtsActionGoal "header:
      seq: 0
      stamp:
      secs: 0
      nsecs: 0
      frame_id: ''
      goal_id:
      stamp:
            secs: 0
            nsecs: 0
      id: ''
      goal:
      text:
            rawtext:
            text: 'Hello world'
            lang_id: 'en_GB
            speakerName: ''
      wait_before_speaking: 0.0"
```

5. Face Recognition

Face and emotion recognition are implemented on top of the Verilook Face SDK provided by Neurotechnolgy.

The ROS package implementing facial perception subscribes to */head_front_camera/image_raw* image and processes this topic at 3 Hz in order to provide the following information:

- Multiple face detection
- 3D position estimation
- Gender classification with confidence estimation
- Face recognition with matching confidence
- Facial attributes: eye position and expression
- Emotion confidences for six basic emotions

As part of SPRING, in WP4, face perception modules will be developed and therefore will replace the existing module of the robot, endowing it with advanced abilities such as mask detection and gaze detection.

6. Speech Recognition

The ARI robot uses Google Cloud Speech API to process speech and extract text and keywords in many languages supported by the Google cloud service. The recognized text can then be used for activating functionalities or answer questions asked by the robot. As part of the SPRING project, WP3 will provide a new speech recognizer that will enhance speech recognition capabilities of ARI.

2.SPRING-ARI Modules and API

This chapter describes the preliminary SPRING modules that will be developed as part of WP1 to WP7, including their inputs/outputs to ensure suitable integration with ARI Robot's architecture and requirements of associated modules. Note it is a working version that will be updated in D7.4 and D7.5.

Figure 8 outlines the general draft of the different modules and their association to their respective packages.

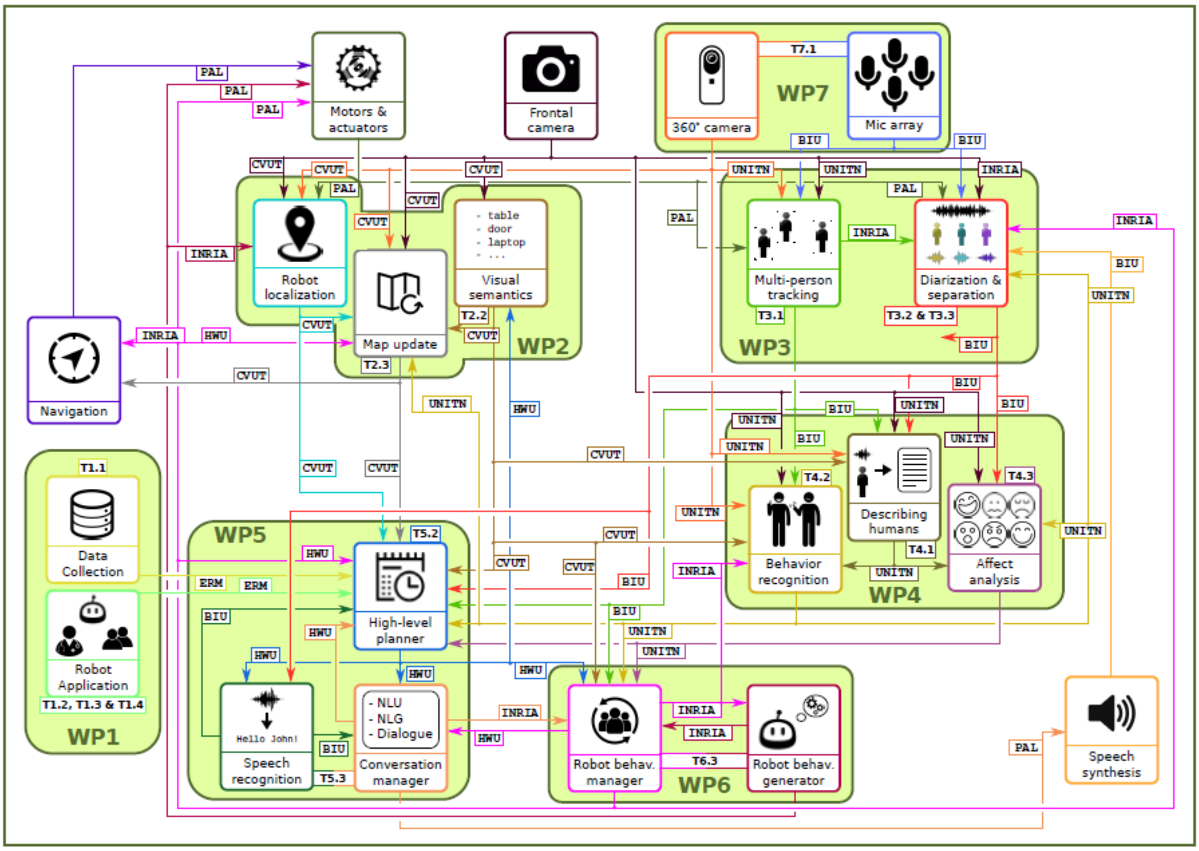


Figure 8: Relational diagram of software modules

The following table specifically summarises the breakdown of ROS nodes per work package:

WORK PACKAGE	**SOFTWARE MODULES**
WP1 - Experimental validation *(lead: ERM)*	• user input via the tablet • non-user application input
WP2 - Environment Mapping, Self-Localisation and Simulation *(lead: CVUT)*	• monocular object localisation • multi-view object localisation • visual self-localisation

	● SLAM ● 2D occupancy map
WP3 - Robot audio-visual perception of humans *(lead: BIU, with INRIA contribution)*	● multi-body tracker ● people's state fusion ● speech diarisation and separation ● sound source localisation ● ASR ● speaker recognition
WP4 - Multi-Modal Human Behaviour Understanding *(lead: UNITN)*	● body pose estimation ● biometrics (id, age, gender) ● activity recognition ● user gaze estimation ● user attention estimation ● group estimation (f-formations) ● user expression recognition
WP5 - Multi-User Spoken Conversations with Robots *(lead: HWU)*	● multi-party ASR ● dialogue manager ● robot verbal utterance generation ● high-level robot social state ● high-level task planning
WP6 - Learning Robot Behaviour *(lead: INRIA)*	● robot navigation ● robot non-verbal behaviours generation ● robot high-level action state
WP7 - Robot Customization and Software Integration *(lead: PAL)*	● low-level input/output robot drivers ● standard ROS messages ● overall architecture integration

Next each module will be briefly described, including their inputs and outputs, followed by ROS based SPRING API. Note that some are better defined than others and will be continuously updated. The input and outputs of each modules have been translated to ROS msgs, actions and services, with a preliminary version stored inside **spring_msgs** metapackage (https://gitlab.inria.fr/spring/wp7_ari/spring_msgs) and available to all partners. Thanks to this duplication of msgs is avoided and it is ensured additional ones are added easily. Some of these are available in the Appendix.

WP1 (ERM): Experimental Validation

2 main modules are subdivided in order to suitably collect and manage data during experiments (T1.2), collect user feedback and validate the technology (T1.3, T1.4, T1.5).

Due to their dependency on other WP modules, its more specific API will be defined as part of D7.4.

1. User-related signals (coming from the tablet).
2. Application-related signals (coming from the evolution of the application).

(Priority High (3) → Low (1))

API Name	Handler	Description	Prior.
User signal (ERM)	Robot Application	Input from the user through the tablet, handled by the "Robot Application." An example could be: "user-signal": {"user1": ["start explanation"]}	1
Application signal (ERM)	Robot Application	Input from the robot application not triggered by a user. Example: "application -signal": {"user1": ["explanation ended"]}	1

WP2 (CVUT): Environment Mapping, Self-Localisation and Simulation

The modules under this WP2 are aimed to develop models, representations and learning algorithms for building and updating a map from the environment, and simulating audio-visual data according to the semantic and behavioural patterns of it. Specifically including:

1. The Visual Robot localization module, that performs robot self-localisation from vision T2.1
2. An audio-visual data simulator providing training data for machine learning T2.2
3. The Visual Semantics Module, that achieves language-driven robot self-localisation T2.3
4. Online Map Update Module, to update current map of the environment with behavioural and semantic information T2.4

(Priority High (3) → Low (1))

API Name	Handler	Description	Prior.	Status

Object Localization (CVUT) with a single camera	Visual Semantics	Visually detected objects: class and localisation and size in the image. Use: this module would run only upon demand. Two examples: ● the robot wants to self-localise, ● when the dialogue module requires to detect objects. **Input:** ● Rectified RGB-D image front fish-eye: [/torso_front_camera/depth/image_rect_raw] sensor_msgs/CompressedImage OR RGB image from head camera: [/head_front_camera/image_raw/compressed] ● Camera info (size, calibration, etc). /torso_front_camera/depth/camera_info [sensor_msgs/CameraInfo] OR /head_front_camera/camera_info [sensor_msgs/CameraInfo] **Output [once per object detected]:** ● object_class [string] ● confidence [float] ● 6D position/pose of object w.r.t. camera -> geometry_msgs/PoseStamped.msg ● 3D bounding box (example ROS msgs: sensor_msgs/RegionOfInterest bounding_box) uint32 x_offset uint32 y_offset uint32 height uint32 width bool do_rectify	2	The module is still in development phase
Object Localization (CVUT) with multiple camera	Visual Semantics	In addition to the definition/input/output of the previous module, additional needs are: **Input:** ● Images from all cameras. ● Transformation between cameras [provided by the ROS TF library: http://wiki.ros.org/tf].	2	Not started

Visual robot self-localization (CVUT)	Robot Localization	Instantaneous position/orientation of the robot in the 2D map. **Input:** ● torso RGB-D camera image [/torso_front_camera/depth/image_rect_raw] ● back RGB-D camera image **Output:** ● Instantaneous position/orientation of the robot (6D) geometry_msgs/PoseStamped.msg ● To be fused (?) with current topic that ORB SLAM is publishing localisation information: /loc_orb/pose [geometry_msgs/PoseWithCovarianceStamped] ● Confidence (face for the time being). Float	3	The module is available as a client server with pythonic ROS client which sends a query image to server/workstation where it is processed and reads the output. But it hasn't been tested with ARI yet
Map & Updates (CVUT)	Map	Environment map and any map updates. **Input:** ● Instantaneous position/orientation of the robot (6D). From Visual Robot Self-localization module. Topic. /loc_orb/pose [geometry_msgs/PoseWithCovarianceStamped] ● 6D position/pose of object w.r.t. camera [given by TF package] -> tf change between /object TF location and /torso_front_camera/ OR ● 6D position/pose of object w.r.t. world (for experimental purposes of CVUT)[given by TF package]. TF change between /object TF location and /world **Output** (http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/OccupancyGrid.html): ● current map (inloc format?) ● coarse 3D point cloud ● floor plan ● some visual panoramas	2	The module is still in development phase

2D occupancy map	Map	Occupancy of the environment map **Input:** ● map [from Map & Updates API] projected to 2D ground ● Obstacle detection/avoidance from ARI (where are the obstacles around ARI in the ground floor). ● /map [nav_msgs/OccupancyGrid] or /move_base/local_costmap/costmap [nav_msgs/OccupancyGrid] with indicated obstacles ● People's position in the ground floor (WP3) **Output:** ● occupancy grid map [nav_msgs/OccupancyGrid]-> e.g. right now ORB SLAM publishes on topic /map, with map information in topic /map_metadata	1	Module not started yet

WP3 (BIU): Robot audio-visual perception of humans

It aims to provide two main modules:

1. Multi-Person Tracking module, that uses auditory and visual raw data to detect, localise and track multiple speakers (T3.1)
2. Diarisation & Separation and the speech Recognition modules, extracting the desired speaker(s) from a speech dynamic mixture and recognising the speech utterances from the separated sources, for a static (T3.2) and a moving (T3.3) robot

How to acquire audio suitably from the ARI robot is one of the matters under discussion.

API Name	Handler	Description	Prior.	Status

People position (INRIA)	Visual Person Tracking	Position (2D/3D) of people in the environment, and their velocity. Input: ● RGB camera images (front torso and back torso (?)): [/front_camera/fisheye/image_raw/compressed] and [/rear_camera/fisheye/image_raw/compressed] ● Depth images [/torso_front_camera/depth/color/points] ● Robot position within the map (WP2) Output [for each tracked person] ● Header (with timestamp and the frame of reference of the detection) ● tracking ID [string/int] ● 3D position [geometry_msgs/Point] ● 3D velocity [geometry_msgs/Point] ● 2D bounding box [——] ● visual representation (re-ID) ● confidence [float] ● last update [float] (-1 if never seen, 0 if visible, else last time seen) See below msgs: TrackedPerson2d.msg	3	The module has been tested initially using RGB images captured by a USB Realsense D435 camera connected to a Jetson TX2 module and wirelessly transmitted to a Laptop for processing. Tests were performed directly from Linux and also within ROS. With ARI, tracking was tested at their office on its front RGB fisheye camera and on the two fisheye cameras from the rear Realsense T265. Processing of wireless-transmitted images was carried out on a laptop.
People's state (INRIA)	Audio-Visual Person Description	Fusion of all modules of the WP Input ● For each tracked person [ID + 3D position + visual representation] ● For each sound source [DoA + ASR + Separated speech + Diarisation + Voice recognition] Output ● Tracked person augmented with audio information (to be better defined)		

Speech diarisation & separation (BIU)	Diarization & Separation	Input: ● Raw audio input [16bit]. RawAudioData.msg (/audio/raw_audio) Output: ● Up to three separated speech utterances string[] utterances (Utterances.msg) ● Binary output for each speaker (active/inactive) /tts/feedback topic will be publishing continuously when the robot is saying something	3	CNN-based multiple speaker tracking. Tested with simulations and with real people moving on an arc. BIU worked with their own linear microphone array mounted in free-space (no body reflections). Implemented in Python 3.
Sound Localization (BIU)	Audio Person Tracking	Localization of sound sources. Input: ● Raw audio input [16bit] RawAudioData.msg (/audio/raw_audio) Output [for each detected source]: ● Direction of arrival (DoA) (in the coordinate frame of the microphone array). [DOAResult.msg]. float32 angle #angle with respect to mic array frame.	3	LCMV beamformer controlled by speakers' activity and direction of arrival of sources in the scene. The control is CNN-based.
ASR (BIU)	Automatic Speech Recognition	Input: ● Separated and enhanced speech signals [see two blocks above] ● Speaker/Tracking ID (for local consistency) Output: ● Transcription (in French). string[] recognition_results		Cloud service French ASR. We evaluated 8 speakers each uttering 100 sentences recorded at the hospital in a quiet environment. Test conditions are not specified. We evaluated: Google, Amazon, IBM and Azure and stand-alone Kaldi. Google and Azure are the only services

				giving useful results. We have also added our noise and acoustics to the recordings and we are currently evaluating the results. We plan to evaluate on-prem solution by NVIDIA.
Speaker Recognition (BIU)	Recognise the speaker within a database	Input: ● Raw audio data RawAudioData.msg (/audio/raw_audio), see below ● Separated speech utterances (see above) ● Binary speaker activity (active/inactive) Output [for each input utterance]: ● ID of a person in the database OR ID of a recently heard person		

WP4 (UNITN): Multi-Modal Human Behaviour Understanding

As part of WP4 the goal is to develop models and algorithms to recognise and interpret high-level human behaviours, with three main modules:

1. Human description (T4.1)
2. Behaviour Recognition (T4.2)
3. Affect Analysis (T4.3)

API Name	Handler	Description	Prior.	Status

Body Pose (UNITN)	Describing Humans	3D skeleton (at least upper body and head orientation) Input: ● Torso Front ELP Camera [/front_camera/fisheye/image_raw/compressed] and [/front_camera/fisheye/camera_info] ● Head front RGB camera [/head_front_camera/image_raw/compressed] ● Positions and bounding boxes in the front image of tracking (WP3) [vision_msgs/BoundingBox2D or 3D] Output [for each detected pose]: ● Header (with timestamp and the frame of reference of the detection ● 2D pose of body points: [Nose, Neck, RShoulder, RElbow, RWrist, LShoulder, LElbow, LWrist, MidHip, RHip, RKnee, RAnkle, LHip, LKnee, LAnkle, REye, LEye, REar, LEar, LBigToe, LSmallToe, LHeel, RBigToe, RSmallToe, RHeel] [geometry_msgs/Point32] For more detail, check ros_openpose for msgs outputs ● TrackingID (from tracking - WP3) [int] (negative values correspond to unassigned poses)	3	Will be integrated during 9/2021 At the moment only 2D pose is estimated.

People's characteristics (UNITN)	Describing Humans	Role, age, etc. Input: • Head front RGB camera [/head_front_camera/image_raw/compressed] • Positions and bounding boxes in the front image of tracking (WP3) [vision_msgs/BoundingBox2D or 3D?] Output [for each detected face]: • identity [string] • age [int] • age_confidence [float] • mask [bool] • mask_confidence [float] • gender • gender_confidence [float] • TrackingID (from tracking - WP3) [int] (negative values correspond to unassigned poses) • Face bounding box 2D See FaceDetection.msg and FaceDetections.msg in Appendix of WP4.		Will be integrated during 9/2021

Activity recognition (UNITN)	Behavior Recognition	Input: ● Torso Front ELP Camera [/front_camera/fisheye/image_raw/compressed] and [/front_camera/fisheye/camera_info+ ● Head front RGB camera [/head_front_camera/image_raw/compressed] ● Bounding boxes (from tracking WP3) corresponding to these frames. ● Use Gaze (from WP4) Output [for each tracked person]: ● activity [string] ● activity_confidence [float] ● actor ID [int] ● co_actor ID [int] (ID of the person involved in the interaction) See ActivityDetection.msg and ActivityDetections.msg in Appendix of WP4.	3	Pending to decide the set of activities.

User Gaze (UNITN)	Behavior Recognition	Extract the gaze of the people. Input: ● Torso Front ELP Camera [/front_camera/fisheye/image_raw/compressed] and [/front_camera/fisheye/camera_info+ ● Head front RGB camera [/head_front_camera/image_raw/compressed] ● Bounding boxes (from tracking WP3) with header specifying the timestamp and frame of reference ● Relative position of the Bounding Box w.r.t. the original image Output: ● eyes located [bool] ● left / right eye x and y coordinates (w.r.t. bounding box) [int] ● center of eyes 3D estimate (w.r.t. the camera) [geometry_msgs/Point32] ● gaze_orientation [geometry_msgs/Point32] See GazeDetection.msg in Appendix of WP4.	3	

User Attention (UNITN)	Behavior Recognition	Identify whom each user attends to with a confidence score and timestamp of start and end, no matter whether speaking or not, for example, Input: ● Active speakers (from WP3) ● User Gaze (from WP4) ● Bounding boxes (from tracking WP3) ● Torso front RGB image (?) ● Head front RGB image (?) Output [per tracked person]: ● attention_confidence to each of the players in the room (including the robot) [float] ● timestamp ● detection_id See Attention.msg and Attentions.ms in Appendix of WP4.	3	
F-formation (UNITN)	Behavior Recognition	Type, spaces. Input: ● Bounding boxes (from tracking WP3) ● 3D Pose Estimation (from WP4) ● Torso front RGB image Output: ● List of group assignment per each TrackedID		

Non-verbal behavior (UNITN)	Behavior Recognition	Emotion recognition. The available emotions are: ANGER, CONTEMPT, DISGUST, FEAR, HAPPINESS, SADNESS, SURPRISE Input [for each person tracked] ● Tracking ID [int] ● Face Torso front RGB image ● Separated speech utterances (WP3) Output: ● string expression ● float32 expression_confidence See EmotionDetected.msg and EmotionsDetected.msg in Appendix of WP4.		

WP5 (HWU): Multi-User Spoken Conversations with Robots

The overall objective of this WP is to develop techniques for multi-user conversation involving a robot and multiple humans and the overall robot task planning.

Specifically, the outcome will be:

1. An initial baseline system for multi-user dialogue to facilitate data collection T5.1.

2. A high-level task planner to connect the overall robot goal, with the low-level goals and the current status of the environment T5.2.

3. A multi-user conversational system trained on collected data T5.3.

API Name	Handler	Description	Prior.	Status
User Spoken Utterances	Multi-party Automatic Speech Recognizer	Required Inputs ● Audio speaker diarisation and extraction (WP3). (speaker ID, location, focus of attention, audio) Outputs ● asrMessage.	3	

Dialogue State	Conversation Manager	Required Inputs ● ASR Message. ● Interaction Message Outputs ● Dialogue state (list of people in the conversation, intent, entities, topic, turn, last one to speak, last robot spoke to, last spoke to robot, ...)		
Robot Verbal Utterance	Conversation Manager	The text the robot should say. Output ● TTS Message in the ROS msg form required by PAL (WP7)	3	
Robot Social State	High-level Planner (Decision Making)	Required Inputs ● Visual robot self-localization. (WP2) ● Scene description (WP2) ● User A/V tracking (WP3) ● Human description (WP4) ● Dialogue state (WP5) ● Robot behaviour (WP6) ● Robot sensors (WP7) Outputs ● Robot Social or Interaction State (e.g. what the robot is doing) ● Interaction Message (who we are interacting with)		

Robot Task	High-level Planner or Behaviour Manager	Required Inputs ● Visual robot self-localisation. (WP2) ● Scene description (WP2) ● User A/V tracking (WP3) ● Human description (WP4) ● Dialogue state (WP5) ● Robot Social State. Outputs ● Robot action to behaviour generator (must return (bool) success)	3	

WP6 (INRIA): Learning Robot Behaviour

The goal of this WP is to provide a non-verbal Behaviour Manager and Robot Non-verbal Behaviour Generation modules allowing to synthesise robot behaviour and to choose the appropriate non-verbal actions.

API Name	**Handler**	**Description**	**Prior.**	**Status**

Robot Navigation Plan (INRIA)	Robot Behavior Generator	This API exposes the robot navigation module : target location, position of the person the robot is following... Required input: ● WP3 - Visual Person Tracking ● WP4 - People status (pose, F-formations, user attention, activity recognition...) ● WP2 - 2D Occupancy map ● WP2 - Object detections ● WP2 - Visual robot self-localization ● Robot status + Target person ID (int) or Target F-formation id (int) Interacting API: ● WP5 - Targeted addressee. Outputs: ● Target location/orientation (Point/Angle): geometry_msgs/PoseStamped target_pose ● Estimated time to reach target - distance to target (time delta) ● ARI Motor API: MoveBase.action to send goal target_pose, and check status Target goal should be fed directly to Move Base action.		

Robot non-verbal behaviour (INRIA)	Robot Behavior Generator	This API exposes the robot non-verbal behaviour: pan and tilt actions, robot gestures. Required input: ● WP6 - Robot status ● WP5 - Dialogue state and Robot Social State ● WP4 - Person non-verbal behaviour and Body Pose ● WP3 - Speech diarisation & separation Outputs: ● ARI Motor API [sent through /play_motion ROS action or joint trajectory controllers of arm_left, arm_right, head]. Should be of type trajectory_msgs/JointTrajectory. See Appendix in WP6 for more details. ● Gesture description + status (idle/moving). To use /play_motion (below) and FollowJointTrajectory Action state (if active, robot moving; if succeeded/aborted, finished moving)		

High-level robot status (INRIA)	High-level Planner	Contains the high-level action the robot is currently executing (e.g., approaching a person, quitting discussion group). Required input: ● WP3/WP4 - Person id list / position / status Interaction: ● WP5 - High level planner Outputs: ● Robot status (following a person / joining a group / escorting a person to a location / idle / participating in a discussion)[ROS Action]. Example: string action string[] parameters uint32 priority ● Target person ID (int) or Target F-formation (int)		

WP7 (PAL): Robot Customization and Software Integration

Modules for WP7 include ARI robot's software architecture itself and ROS packages, messages, services and actions that are necessary to interface its sensors and existing modules with those developed in WP1 to WP6. Some of which most relevant are access to its cameras, microphone, and joints, defined in the robot's manual.

ARI cameras

For most modules raw images are needed. Main camera topics (excluding camera info):

Back RGB-D camera:

/torso_back_camera/fisheye1/image_raw
/torso_back_camera/fisheye1/image_raw/compressed

Front RGB-D camera:

/torso_front_camera/aligned_depth_to_color/image_raw
/torso_front_camera/aligned_depth_to_infra1/image_raw
/torso_front_camera/color/image_raw
/torso_front_camera/color/image_raw/compressed
/torso_front_camera/depth/color/points
/torso_front_camera/depth/image_rect_raw
/torso_front_camera/infra2/image_rect_raw

Head RGB camera:

/head_front_camera/image_raw/compressed

Front and back torso fisheye cameras:

/front_camera/fisheye/image_raw/compressed
/rear_camera/fisheye/image_raw/compressed

Microphone-array (audio input)

ARI uses a ReSpeaker Array MicV2.0, with the respeaker ros package that is used to acquire audio input for WP3 modules.

/sound_direction [std_msgs/Int32] -> result of DoA
/sound_localization [geometry_msgs/PoseStamped] -> result of DoA as pose
/audio [AudioData.msg] -> raw audio
/is_speeching [boolean] -> speech detected yes/no
/speech_audio [AudioData.msg] -> audio data while speaking

Note that as part of the project and WP3 work adjustments will be made to how audio is retrieved to ensure high-quality capture.

Motors and joints

ARI's joints may be controlled using the joint_trajectory_controller, of interest for WP4 and WP6 modules. Specifically ARI has 3:

- arm_left_controller
- arm_right_controller
- head_controller

For each controller:

/arm_left_controller/command (trajectory_msgs/JointTrajectory)

- Topic interface to move the left arm

/arm_left_controller/follow_joint_trajectory (control_msgs/FollowJointTrajectoryAction)

- Action interface to move the left arm
- In the case of action -> action state will indicate if robot is idle/moving: state
- pending: goal has yet to be processed by the action server
- active: goal is currently being processed by the action server
- succeeded: goal successfully achieved
- aborted: goal was terminated

Coordinate changes

Provided by ROS's TF package (http://wiki.ros.org/tf). Using a ROS TF Listener it is possible to read the transformation between a *source_frame* and a *target_frame.*
Example using a terminal:

```
rosrun tf tf_echo /head_front_camera_link /torso_front_camera_link
```

```
rosrun tf tf_echo /torso_front_camera_link /head_front_camera_link (backward
transformation)
```

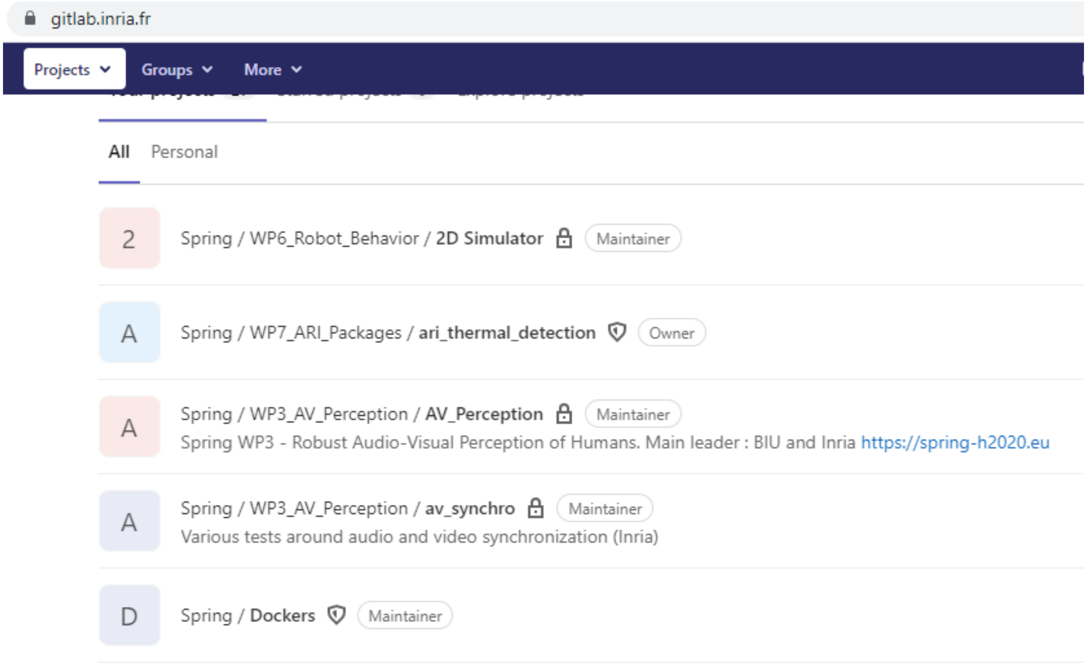
4. Quality Assurance

This section describes the main software and continuous integration guidelines as well as how the software modules and results are stored and updated.

Gitlab

The source code and results of the SPRING-ARI software will be stored and maintained for 4 years minimum past project ending. Specifically at SPRING partner INRIA's private Gitlab (https://gitlab.inria.fr/), which while being internal accepts all invites received to the administrators.

Each Work-Package (WP) has its own sub-group repository, where different projects are added. And each WP has a leader, which is in charge of their repository (Figure 9).



2 Spring / WP6_Robot_Behavior / **2D Simulator** Maintainer

A Spring / WP7_ARI_Packages / **ari_thermal_detection** Owner

A Spring / WP3_AV_Perception / **AV_Perception** Maintainer
Spring WP3 - Robust Audio-Visual Perception of Humans. Main leader : BIU and Inria https://spring-h2020.eu

A Spring / WP3_AV_Perception / **av_synchro** Maintainer
Various tests around audio and video synchronization (Inria)

D Spring / **Dockers** Maintainer

Figure 9:SPRING Gitlab repository

- WP1 User Application: https://gitlab.inria.fr/spring/wp1_user_application
- WP2 Mapping and Localisation: https://gitlab.inria.fr/spring/wp2_mapping_localization
- WP3 AV Perception: https://gitlab.inria.fr/spring/wp3_av_perception
- WP4 Behavior: https://gitlab.inria.fr/spring/wp4_behavior
- WP5 Spoken Conversation: https://gitlab.inria.fr/spring/wp5_spoken_conversations
- WP6 Robot Behavior: https://gitlab.inria.fr/spring/wp6_robot_behavior
- WP7 ARI: https://gitlab.inria.fr/spring/wp7_ari

Software Integration Guidelines

As part of software integration the established GitHub Flow model will be used. Despite the name, it works with any git infrastructure, in our case, gitlab.

At all times, the main (or master) branch must be usable, or at least compilable even if it's non-functional. New features or bug fixes are done in separate branches, merge requests are submitted by the developers and approved by each WP leader (maintainer) after review.

If this workflow is not followed and some breaking changes are committed to the main branch, other partners' progress may be affected if they cannot compile or execute their code.

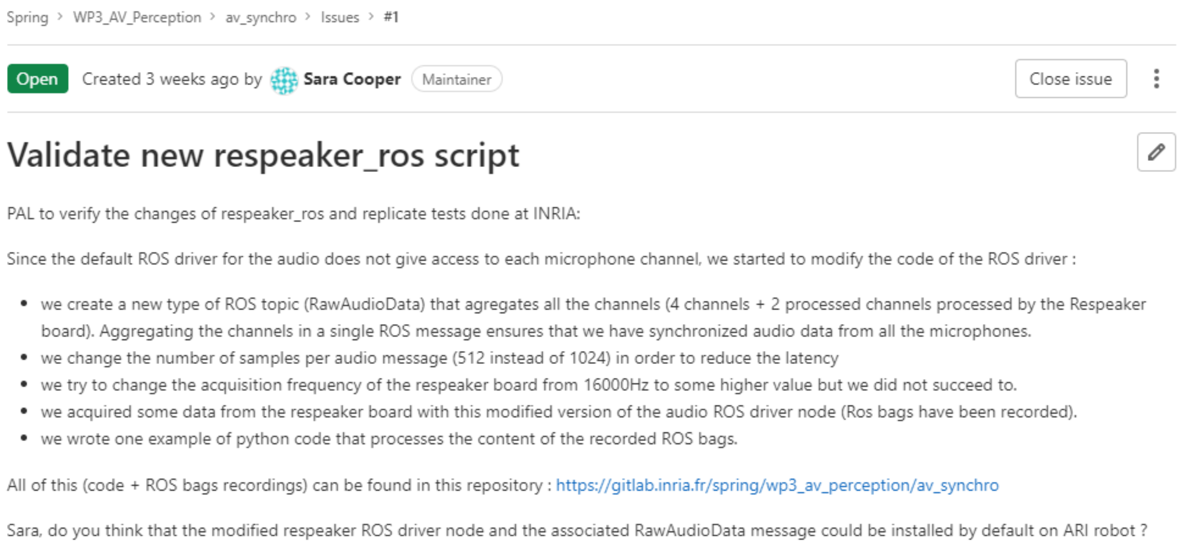
Specifically, here we will have the following procedure, as part of Tasks 7.3, 7.4 and 7.5 of software integration.

- Add new feature/code in a new branch -> work on it, add unit tests, and ensure it works on your ARI **-> each WP developer/maintainer**

- If as the developer works there are integration questions, problems, Issues are raised assigned to PAL (WP7 leader), directing to the line of code you want to check, as mentioned above.

- Make sure that the code of the branch compiles properly and that it has been tested on the robot, before assigning a merge request

- Once it is verified, do a merge request, assigned to PAL, who validate it before it is merged. This means that the following will be checked:
    - 1. Package is deployed correctly onto the robot. If not PAL will raise an Issue in the respective repo.
    - 2. It passess the tests, continuous integration pipeline successful. If not, PAL raises an Issue. In parallel, if developer does not know how to test something specifically, use same means to explain feature, in order to provide tips
    - 2. Validate API. **PAL will check it is publishing the topics and msgs it should, contrasting it with either the SPRING API document mentioned in Section III, ensuring the module t**hat it subscribes to all needed components. If there are points of improvement or need to contact with another partner/module,it will ibe checked through Issues.
    - 3. Test: PAL will finally check the functionality on their ARI. E.g. face recognition that it successfully recognises person. For this it is important that all MR have a description on brief description of inputs/outputs of the respective package, and how to run it. If there are errors or the performance can be improved somehow, PAL will raise an issue (Figure 10).


Spring › WP3_AV_Perception › av_synchro › Issues › #1
Open
Created 3 weeks ago by Sara Cooper
Maintainer
Close issue

Validate new respeaker_ros script

PAL to verify the changes of respeaker_ros and replicate tests done at INRIA:

Since the default ROS driver for the audio does not give access to each microphone channel, we started to modify the code of the ROS driver :

- we create a new type of ROS topic (RawAudioData) that agregates all the channels (4 channels + 2 processed channels processed by the Respeaker board). Aggregating the channels in a single ROS message ensures that we have synchronized audio data from all the microphones.
- we change the number of samples per audio message (512 instead of 1024) in order to reduce the latency
- we try to change the acquisition frequency of the respeaker board from 16000Hz to some higher value but we did not succeed to.
- we acquired some data from the respeaker board with this modified version of the audio ROS driver node (Ros bags have been recorded).
- we wrote one example of python code that processes the content of the recorded ROS bags.

All of this (code + ROS bags recordings) can be found in this repository : https://gitlab.inria.fr/spring/wp3_av_perception/av_synchro

Sara, do you think that the modified respeaker ROS driver node and the associated RawAudioData message could be installed by default on ARI robot ?

Figure 10: Raising Issues using Gitlab

- Once above 4 points are met, PAL merges the branch with master

Unit Testing

To make the WP leaders' work easier, we encourage the implementation of Unit Tests that automatically check that the code compiles and works as expected. Before submitting a MR all developer should ensure the tests pass.

How can I test my program does what it should?

- Using mocks: google tests (https://github.com/google/googletest.) or google mock (https://github.com/google/googlemock). For example:
    - Check that a service that initiates face enrolment is started as it should
- Using rosbags: http://wiki.ros.org/rosbag. For example:
    - Test face detection by getting rosbag input from cameras to make sure it successfully detects a face
    - Test speech recognition by inputting to the test rosbag of /audio topic

The following documentation for guidelines of unit testing in ROS has been taken into account to ensure all code uploaded to Gitlab is tested thoroughly:

http://wiki.ros.org/Quality/Tutorials/UnitTesting
https://www.theconstructsim.com/how-to-test-your-ros-programs/

Continuous Integration

Continuous integration has been enabled in existing repositories, any new repositories must be manually configured.
To enable CI we require:

- A valid **.gitlab-ci.yml** file must be present in the root directory of the project.
- A properly configured **gitlab-runner**.

The **gitlab-ci.yml** file defines the test structures and commands that the gitlab-runner should execute. It can be extended for different behaviors in specific branches or on special gitlab events such as merge requests.

The gitlab runner represents the slave node where the tests will be executed. You can have as many as you need.
INRIA has provided a shared gitlab runner for all work packages, should it be frequently overloaded by the amount of code changes, additional runners could be easily added.

Because the repositories are ROS, we have integrated a framework for continuous integration with ROS called industrial_ci. This framework is easily extendable and should be able to cover all SPRING use cases.

By the end of the project, we aim at:

- having all Merge Requests for each WP’s repo automatically compiled and unit-tested before authorising the Merge Request to be merged to the main git branch;

- having (manually triggered) **integration** tests where the full architecture is instantiated (using *docker compose* with WP-produced docker images, see below). While we might not achieve complete integration testing with real-world data, mock data will be used to validate the data flow between the modules, with known expected results.

As such, we aim at 100% **functional** coverage of the architecture by the end of the project.

Dockers

Docker is a container platform provider where users can share and upload/downlaod applications from the cloud. An architectural overview is shown in Figure 11. In docker, the physical computer (the Host) runs and virtualizes applications (that are developed, deployed and run with Containers), which are stored in the cloud. A container is launched by running an image and is defined by a Dockerfile. An image is an environment that includes everything needed to run an application, i.e. the code, a runtime environment, libraries, environment variables, and configuration files. The user can pull and push images from a remote repository by means of a terminal (Client).

PAL Robotics ARI robot has provided a base docker image running on ROS Melodic and Ubuntu 18.04, shared with all SPRING partners through a secured gitlab.com/pal-robotics/ account. Like this each partner can pull the base docker and work in a common environment, thus solving the problems of diversity of platforms. Instructions to use the dockers with ROS are those coinciding with http://wiki.ros.org/docker/Tutorials/Docker.

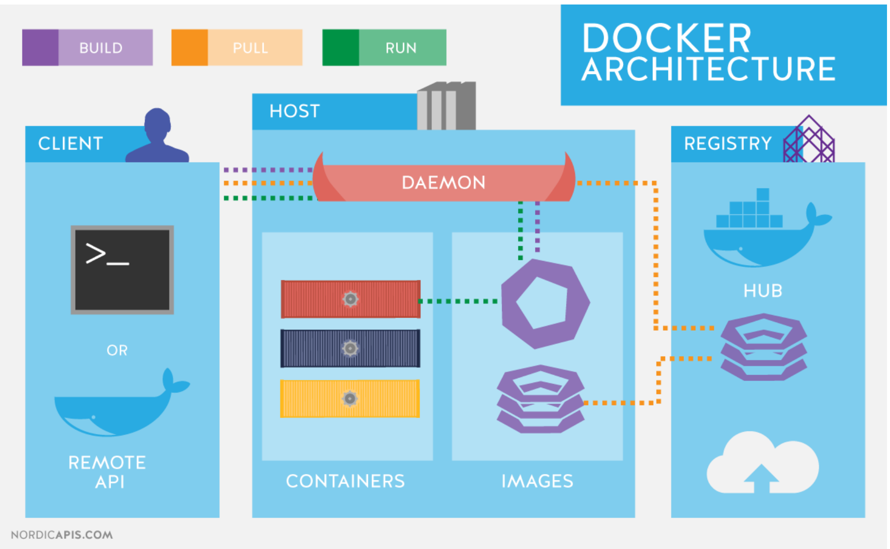


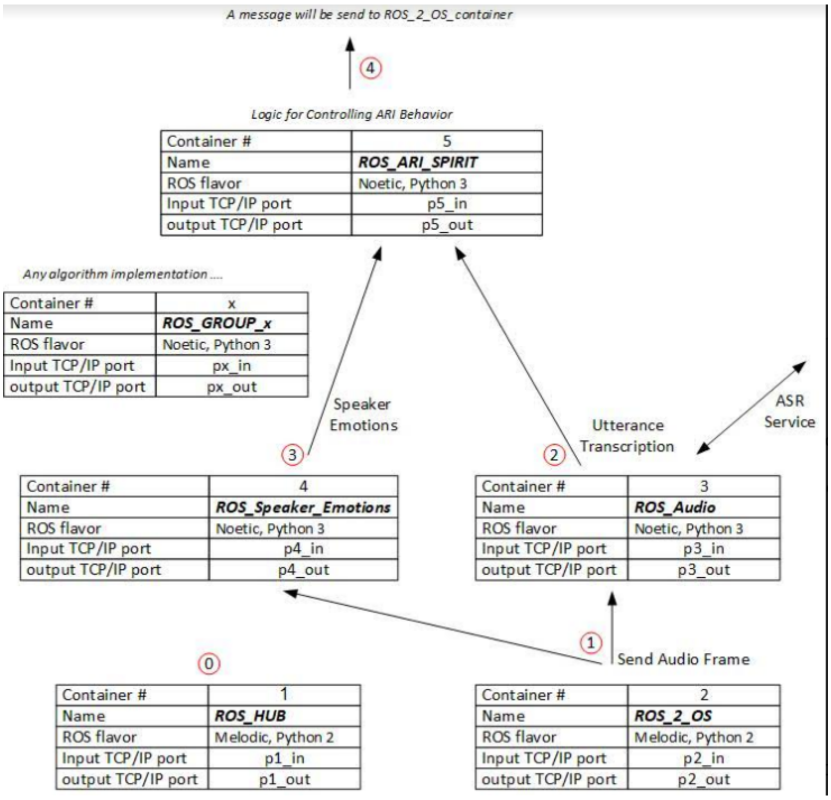
Figure 11: Docker architecture

The different SPRING modules are developed in different programming languages (e.g. Python 2 and 3, for instance) and using different packages, in order to ensure interoperability, it is necessary that the SPRING-ARI supports both ROS Melodic and ROS Noetic.Our proposed solution is to use Dockers both in development and in the robot, to mix both ROS systems. On a development PC with ROS Noetic installed, the ARI simulation can be started using its base Docker image as described in the first section.

Once the setup is configured and a container is started, the ARI simulation inside the docker can be run and the ROS interfaces are made accessible from the Noetic installation.

On the robot, which is installed with ROS Melodic, the opposite might be needed. Each module must be provided inside a docker image with its required environment, and will start a container from this image inside the robot, which will be able to interact with the ROS Melodic system.

Figure 12 shows an example of the structure, where each partner works with their docker and the needed ROS versions. Each module or WP containers will be stored inside the container-registry of its respective repository in gitlab, e.g. https://gitlab.inria.fr/spring/wp2_mapping/container_registry


A message will be send to ROS_2_OS_container
4
Logic for Controlling ARI Behavior
Any algorithm implementation....
Speaker
Emotions
3
Utterance
Transcription
2
ASR
Service
1
Send Audio Frame
0

Container #	5
Name	***ROS_ARI_SPIRIT***
ROS flavor	Noetic, Python 3
Input TCP/IP port	p5_in
output TCP/IP port	p5_out

Container #	x
Name	***ROS_GROUP_x***
ROS flavor	Noetic, Python 3
Input TCP/IP port	px_in
output TCP/IP port	px_out

Container #	4
Name	***ROS_Speaker_Emotions***
ROS flavor	Noetic, Python 3
Input TCP/IP port	p4_in
output TCP/IP port	p4_out

Container #	3
Name	***ROS_Audio***
ROS flavor	Noetic, Python 3
Input TCP/IP port	p3_in
output TCP/IP port	p3_out

Container #	1
Name	***ROS_HUB***
ROS flavor	Melodic, Python 2
Input TCP/IP port	p1_in
output TCP/IP port	p1_out

Container #	2
Name	***ROS_2_OS***
ROS flavor	Melodic, Python 2
Input TCP/IP port	p2_in
output TCP/IP port	p2_out

Figure 12: Each module or software to be wrapped in a docker container

During software integration, there are two main reasons why each partner should update the docker image of the gitlab repository:

1. The base SPRING-ARI Image might be changed time to time, due to robot software upgrade
2. A SPRING partner has finished a package/feature or modified the Dockerfile and needs to be tested and used by other partners

See the Appendix for details on docker updates.

5.Conclusions

The preliminary structure of the SPRING-ARI software modules and API have been outlined in this document, followed by software and continuous integration guidelines set up.

These as well as the source code will be updated between the writing of this deliverable and autumn in relation to achieving Milestone 4.

Appendix

SPRING-ARI ROS API

In this appendix the main ROS mgs, actions, services and topics that will be considered are outlined subdivided by Work-Package (WP). Some are used by the ARI robot already, while those new ones for SPRING are added as [new]. Note this will be an evolving API and updated in the subsequent deliverables D7.4 and D7.5.

WP2

- [new]DetectObject.msg -> corresponding to output of Object Localization with a Single Camera

```
std_msgs/Header header
# class: The respective class type of the found object
string object_class

# confidence: how sure you are it is that object and not another one
# It is between 0 and 1 and the closer to 1 it is better
float32 confidence

# pose: 6D position/pose of the object w.r.t camera
geometry_msgs/PoseStamped pose

# bounding_box: The region of the image, where the object is found
sensor_msgs/RegionOfInterest bounding_box
```

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

- [new]DetectedObjectsArray.msg -> save as above but stores the objects in an array

  std_msgs/Header header

  spring_msgs/DetectedObject[] objects

- map_msgs/OccupancyGridUpdate -> to update Occupancy Grid map

Related services:

- pal_navigation_msgs/SaveMap.srv -> to save an occupancy grid map.
- pal_navigation_msgs/ListMaps.srv -> to list available maps of the robot
- pal_navigation_msgs/RenameMap.srv -> rename existing map
- nav_msgs/SetMap.srv -> set a new map together with the initial pose
- nav_msgs/GetMap.srv -> get the map as an OccupancyGrid
- [new] spring_msgs/UpdateMap.srv -> update map. Uses msgs: map_msgs/OccupancyGridUpdate
- [new] spring_msgs/VisualRobotLoc.srv -> returns estimated pose of the robot as PoseWithCovarianceStamped message and its confidence
- [new] DetectedObjects.srv -> detect objects in the camera stream
- [new] DetectObjectsRegion.srv -> detect object given a region of interest and object label

Related actions:

- /move_base [move_base_msgs/MoveBaseGoal for goal message) -> send navigation goal within a map (PoseStamped message in cartesian)

Related topics (excluding cameras):

- /loc_orb/pose [geometry_msgs/PoseWithCovarianceStamped] -> pose returned by ORB_SLAM localization -> input
- /map [nav_msgs/OccupancyGrid] -> the occupancy grid map that was produced using ORB SLAM is published in this topic.
- /move_base/local_costmap/costmap [nav_msgs/OccupancyGrid] -> occupancy grid indicating obstacles in the map, as computed by the robot's obstacle avoidance system
- /map_metadata [nav_msgs/MapMetaData]. Return occupancy grid map information

WP3

- [new] spring_msgs/TrackedPerson2d.msg -> currently detected person

```
uint64          track_id                # unique identifier of the target, consistent over
time
geometry_msgs/PointStamped   position3D          # person 3D position
geometry_msgs/PointStamped   velocity3D  # person 3D velocity
sensor_msgs/RegionOfInterest bounding_box        # bounding box for the detected
person
float32         confidence              # confidence
```

- [new] spring_msgs/TrackedPersons2d.msg -> array of currently detected people

```
# Message with all 2d box in image of currently tracked people

Header header

spring_msgs/TrackedPerson2d[] detections # all person that are currently being
tracked
```

- [new]DOAResult.msg -> indicates direction of arrival angle

```
# Angle with respect to the microphone array frame
float32 angle
```

- [new]Utterances.msg -> array of string utterances:

```
string[] utterances
```

Related topics:

- [new] /audio/raw_audio [RawAudioData.msg] -> publishes audio from ARI's ReSpeaker (respeaker_ros) using all the 6 channels.

```
Header header
int8   nb_channel
int32  rate
int32  format
int32  sample_byte_size
int32  nb_frames
int16[] data
```

- [new] `/spring_recognizer/words`:Publishes the result and the confidence value of real time speech recognition [SpeechResult.msg]
-
- [new] `/spring_recognizer/sentences`: Publishes the final result and confidence value of speech recognition
-
- [new] `/spring_recognizer/direction_of_arrival`: Publishes the estimation of the sound source position for a given utterance [DOAResult.msg]

Related actions:

- Tts.action -> given an input text, produce speech output. Interesting for this API for example to check /tts/status to check if the robot is speaking or not (speakers active or not)

WP4

Related ROS msgs:

- [new] FaceDetection.msg

  # Person recognition

  string identity
  float32 identity_confidence

  # Gender recognition

  string gender
  float32 genderConfidence

  # Age recognition

  int age
  float32 age_confidence

  # Tracking ID

  int tracked_id

```
# Face bounding box

sensor_msgs/RegionOfInterest bounding_box
```

- [new] FaceDetections.msg

  ```
  Header header

  spring_msgs/FaceDetection[]  faces

  # Optional transformation between the camera frame and a certain parent frame
  geometry_msgs/TransformStamped camera_pose
  ```

- [new] ActivityDetection.msg

  ```
  # Indicates activity recognized for the tracked person

  string activity
  float32 activity_confidence
  int actor_id
  int co_actor_id
  ```

- [new] ActivityDetections.msg

  ```
  Header header
  spring_msgs/ActivityDetection[]  activities
  ```

- [new] GazeDetection.msg

  ```
  # Outputs eye position and gaze information, if found

  bool eyesLocated

  int32 leftEyeX
  int32 leftEyeY
  int32 rightEyeX
  int32 rightEyeY

  geometry_msgs/Point32 position #centre of eyes 3D estimate

  geometry_msgs/Point32 gaze_orientation #gaze orientation

  sensor_msgs/RegionOfInterest bounding_box #bounding box from FaceDetection
  ```

- [new] Attention.msg

  ```
  # Estimation of user attention

  int32 tracked_id
  float32 attention_confidence
  ```

- [new] Attentions.msg

  ```
  #Array indicating the attention for each detected user

  Header header
  int32 detection_id
  spring_msgs/Attention[] attentions
  ```

- [new] EmotionDetected.msg

  ```
  # List available emotions detected

  int32 tracked_id

  #Facial expressions and their confidence
  string EXPRESSION_ANGER="anger"
  string EXPRESSION_CONTEMPT="CONTEMPT"
  string EXPRESSION_DISGUST="disgust"
  string FEAR="fear"
  string HAPPINESS="happiness"
  string SADNESS="sadness"
  string SURPRISE="surprise"
  string expression
  float32 expression_confidence
  ```

- [new] EmotionsDetected.msg

  ```
  Header header

  spring_msgs/EmotionDetected[]  emotions
  ```

Related topics:

- /pal_face/faces [pal_detection_msgs/FaceDetections] -> message with the regions of interest classified as faces.
- /pal_face/debug [sensor_msgs/Image[-> debug image with detected faces (bounding box included)

Related services:

- /pal_face/recognizer -> Recognizer.srv
- /pal_face/set_database

WP5

Related ROS msgs:

[new] **asrMessage.msg:**

```
asrUser Speaker;                #who spoke from speaker diarization & separation
asrUser[] Listener;             #list of users inside the conversation.
asrResult Utterance;
time start;
time end; #or duration
```

[new] **asrUser.msg:**

```
string UUID;            #UUID should be same in all messages that refer to same user.
location;                                 #User position and focus of attention messages
user_focus_of_attention;        #from wp3 tracking, should have a confidence, etc.
float confidence;       #confiden of user identification
```

[new] **asrResult.msg:**

```
boolean final;          #whether the results is final or partial result.
string text;
wordResult[] words;
float confidence;
```

[new] **wordResult.msg:**

```
string word;
float confidence;
```

[new] **dialogueState.msg:**

```
string interactionID;
dialogueMsg dialogue
dialogUser lastSpoke
dialogUser lastRobotSpoke
dialogUser lastSpokeRobot
```

[new] **dialogueMsg.msg**:

```
string intent;
string entities;
```

```
        dialogueTask task;
        string turn;
[new] dialogueTask.msg:
        string taskID;
        string command;   # what is send to the dialogue arbiter.
[new] dialogueUser.msg:
        string UUID;
        time time_of_user_dialogue_event;
        #string text;  #optional

[new] interactionMsg.msg::
        time start;
        bool active;
        string ID;  #ID of interaction
        groupInteractionMsg interactionGroup
        dialogueState lastDialogue;
[new] groupInteractionMsg.msg:
        userDescription[] participants;  #participants in the interaction group, [1 .. *]

[new] robotSocialState.msg:
        bool inConversation;
        bool inNavigation;
        interactionMsg activeInteraction;
        robotDescriptionMsg robotStatus
[new] robotDescriptionMsg.msg:
        robotTarget;
        robotAction;
        robotGoal;
        robotLocation;
        robotGazeDirection;
        RobotBattery;
```

WP6

Related msgs:

- [new] NavigationGoal.msg

```
# Robot navigation target location

geometry_msgs/PoseStamped target_pose

float32 estimated_time
```

Related actions:

PlayMotion will be used to trigger different robot gesture-based behaviours.

- PlayMotion.action -> sends a motion previously generated

  string motion_name

  bool skip_planning

  int32 priority

- MotionInfo.msg -> information on the joints that the motion executes

  string name

  string[] joints

  duration duration

- ListMotions.srv -> returns list of currenty loaded motions that can be played by play_motion

  MotionInfo[] motions

- isAlreadyThere.srv -> checks if the robot joint state matches the first point of a given motion. Robot joint positions will be checked against the values found in the motion's firtst point, given a tolerance

  goal: string motion_name

  float32 tolerance (in radians)

  result: bool already_there

Docker updates

Procedure to update the dockers should be the following:

```
docker login registry.gitlab.inria.fr
docker pull registry.gitlab.inria.fr/wp2_mapping/sciroc/mydocker

# Update your local private_gitlab containing the latest Dockerfile
```

```
cd <wp2_mapping>
git pull --rebase
# Build your image
docker build .
# Tag the pulled docker with your personal docker name
docker tag
registry.gitlab.inria.fr/wp2_mapping/sciroc/mydocker<docker_remote_path_name>
# Upload your new image to gitlab.inria wp2_mapping
docker push <docker_remote_path_name
```