UPANZI
DIGITAL PUBLIC INFRASTRUCTURE
NETWORK

Pepper Robot Tour for Digital Experience Center

D4.1 Robot Sensing Migration

Due date: **01/03/2026**
Submission Date: **01/06/2026**
Revision Date:

Start date of project: **01/07/2026**

Duration: **12 months**

Responsible Person: **Yohannes Haile**

Revision: **1.0**

Project funded by Upanzi Network Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants	
RE	Restricted to a group specified by the consortium	
CO	Confidential, only for members of the consortium	

Executive Summary

Work Package 4 (WP4) addresses the migration of all software modules from Work Package 4 of the CSSR4Africa project from ROS1 to ROS2. The objective is to ensure full compatibility with the ROS2 framework while making necessary changes needed to improve up on the original ROS1 implementations. This work package includes the complete migration of all modules developed in WP4, encompassing the following deliverables:

- Person Detection and Localization Module
- Face Detection and Mutual Gaze Localization Module
- Speech Event package (comprising the Speech Event node and the Speech Localization node)

Robot Localization will be developed directly for ROS2 and hence is not included in this migration deliverable. Sensor Tests isn't included in this deliverable to make the migration process more manageable considering the number of modules that need to be migrated also considering the number of individuals working on the entire project. Furthermore, unit tests for each nodes isn't included as well as the migration process is focused on the main functionality of the nodes.

Contents

1 Introduction **4**

2 Requirements Definition **5**

3 Migration Overview **7**

4 Module Specifications **8**

5 Module Design **9**

6 Implementation **13**

References **23**

Principal Contributors **24**

Document History **26**

1 Introduction

This document describes the migration of the Work Package 4 software modules from ROS1 to ROS2 as part of the CSSR4Africa project. The CSSR4Africa project aims to develop culturally sensitive social robots capable of interacting naturally with people in African contexts, using the Pepper robot as the primary platform. The overall system architecture for the CSSR4Africa project is defined in CSSR4Africa Deliverable D3.1 [1]. Work Package 4 covers the perception capabilities of the robot, including person detection and localization, face and mutual gaze detection, sound detection and localization, and speech event detection. These modules were originally developed for ROS1 (Noetic) and are being migrated to ROS2 (Humble) to align with the project's long-term software strategy and to benefit from the improved reliability, security, and tooling offered by ROS2.

The migration encompasses the following deliverables:

- **D4.2.1**: Person Detection and Localization — detects and localizes objects in the robot's field of view using RGB-D camera input. The module uses YOLOv11 for detection and allows the user to specify which person class(es) to track (e.g., person, or any class supported by YOLO). Tracking is performed using BYTETrack.
- **D4.2.2**: Face and Mutual Gaze Detection and Localization — detects faces, tracks them across frames, and determines whether mutual gaze is established between the robot and a user.
- **D4.3.2**: Speech Event — the Sound Detection and Localization module (D4.2.3) and the Speech Event module (D4.3.2) have been consolidated into a single ROS2 package. The package provides two nodes: the *Speech Recognition* node and the *Speech Localization* node. The Speech Recognition node replaces the original keyword-spotting approach with a full speech recognition and localization-in-time pipeline based on Voice Activity Detection (VAD) and Automatic Speech Recognition (ASR), detecting speech onset and offset and producing verbatim transcriptions. In addition, it comprises of the sound detection signal processing to reduce noise. The Speech Localization node estimates the horizontal direction of any sound source using SRP-PHAT beamforming on Pepper's four-microphone array.

Each module has been rewritten to conform to the ROS2 API, replacing ROS1 constructs with their ROS2 equivalents. This includes migrating from `rospy` to `rclpy`, from `catkin` to `ament_python` build system, from `.launch` XML files to Python-based ROS2 launch files, and from ROS1 `.msg` definitions to ROS2 interface packages. The migrated nodes maintain the same functional specifications as their ROS1 counterparts, ensuring continuity in system behavior while operating within the ROS2 ecosystem.

2 Requirements Definition

This section defines the migration requirements for Work Package 4 (WP4). The primary requirement is that each migrated ROS2 node must preserve the functional behavior of its ROS1 counterpart. In addition, the migration must satisfy the following technical and structural requirements.

ROS2 Compatibility

- All nodes must be implemented using `rclpy` and conform to the ROS2 (Humble) API.
- Build system files must be migrated from `catkin` to `ament_python`, with `package.xml` updated to format 3.
- Launch files must be rewritten as Python-based ROS2 launch files, replacing ROS1 XML `.launch` files.
- Custom message types must be redefined as ROS2 interface packages using `rosidl`.

Functional Preservation

- Each migrated node must subscribe to and publish on topics that are functionally equivalent to those in the original ROS1 implementation or has improved functionality. For example, the face detection node (D4.2.2) now relies on person detections from the person detection module (D4.2.1) for improved tracking accuracy, which is an intentional change to enhance performance.
- The detection algorithms and output data structures for each module must be preserved. Where tracking logic has been revised to improve accuracy (e.g., D4.2.2), those improvements are documented as intentional changes within this migration.

Module-Specific Requirements

- **D4.2.1 — Person Detection and Localization**: The migrated node must detect and localize objects in the robot's field of view using YOLOv11, with the target person class configurable by the user (e.g., person, or any class supported by YOLO). Tracking must be performed using BYTETrack, and the node must publish bounding box coordinates and depth estimates over the corresponding ROS2 topic.
- **D4.2.2 — Face and Mutual Gaze Detection and Localization**: The migrated node must detect faces, estimate head pose, and publish mutual gaze status and bounding box data over the corresponding ROS2 topic. SORT tracking has been removed; face tracking now relies on person detection from the person detection module (D4.2.1, person class) for improved tracking accuracy. A model warm up step must be included to pre-load detection models into memory before processing begins.
- **D4.3.2 — Speech Event Package (D4.2.3 + D4.3.2)**: The Sound Detection and Localization module (D4.2.3) and the Speech Event module (D4.3.2) are delivered as two primary ROS2 nodes within the same package.

  - The *Speech Recognition* node perform speech recognition that is it detects speech onset and offset using Voice Activity Detection, and then produce a verbatim transcription of the detected utterance using Automatic Speech Recognition. It must operate either as a ROS2 action server — returning the transcription as the action result when queried — or in a continuous standalone mode that publishes transcriptions to a topic. The original keyword-spotting approach had insufficient accuracy and has been replaced with a Silero VAD + Faster-Whisper pipeline.
  - The *Speech Localization* node must detect sound events in the audio stream and publish the estimated horizontal direction (azimuth) of the sound source, along with a confidence score, using SRP-PHAT beamforming on Pepper's four-microphone planar array. In the ROS1 implementation, it was limited to frontal field of view of the robot in addition it utilized GCC-PHAT for localizing the speech.

Configurable Parameters

- Each node must load its operating parameters from its respective YAML configuration file. Parameters are declared as ROS2 node parameters and loaded via a `--params-file` argument at launch time.
- Verbose mode must remain supported across all modules, providing optional diagnostic output and visual debugging.

Misalignment and Scope Exclusions

- Unit tests for each module are not included in this deliverable; the migration is focused on the functional nodes.
- The D4.1 Sensor Tests and D4.2.4 Robot Localization modules are excluded from this migration. Robot Localization will be developed natively for ROS2 using Self Localization and Mapping (SLAM).
- The simulator is not supported by the migrated modules; all nodes target the physical Pepper robot with the Intel RealSense D435i camera.

3 Migration Overview

This section summarises the systematic changes applied across all modules during the ROS1-to-ROS2 migration.

Common Migration Changes

Module	Migration Approach	Redesigned?
D4.2.1 Person Detection	Direct port with API updates; tracker upgraded to BYTETrack; Migrated to use YOLOv11 replacing YOLOv8	Partial
D4.2.2 Face & Mutual Gaze	Direct port; SORT removed, tracking delegated to D4.2.1	Partial
D4.2.3 & D4.3.2 – Speech Event (combines Sound Detection and Speech Event from ROS1)	The standalone Sound Detection module (D4.2.3) is removed; its functionality is absorbed into the unified `speech_event` package. The Speech Recognition node was fully redesigned with a VAD + ASR pipeline (Silero VAD + Faster-Whisper) and a ROS2 Action-based interface. The Speech Localization node was ported using SRP-PHAT beamforming.	Yes

Table 1: Per-module migration summary.

4 Module Specifications

This section describes the expected behaviour and functional requirements of each module delivered in Work Package 4.

D4.2.1 — Person Detection and Localization

The Person Detection and Localization node (originally specified in CSSR4Africa Deliverable D4.2.1 [**?**]) must detect and localise people in the robot's field of view in real time using an RGB-D camera. It is expected to accept a configurable target class list, so that the operator can restrict detection to `person` or any other COCO class. Detections must be associated with persistent track identities across consecutive frames so that downstream modules can follow individuals over time. For each tracked object, the node must estimate a 3D position (pixel centroid plus depth) and publish it for consumption by other modules such as face detection. The node must support both the Intel RealSense D435i and Pepper's built-in cameras, selectable via a configuration parameter, and must operate in verbose mode for diagnostic purposes when required.

D4.2.2 — Face and Mutual Gaze Detection and Localization

The Face and Mutual Gaze Detection node (originally specified in CSSR4Africa Deliverable D4.2.2 [**?**]) must detect faces in the robot's camera feed, estimate head pose for each detected face, and determine whether mutual gaze is established between the robot and the user. It is expected to consume person detections from D4.2.1 to associate faces with tracked individuals, removing the need for an independent tracking module. The node must publish, for each detected face, the bounding box, estimated head pose angles, and a mutual-gaze flag. Mutual gaze must be determined by comparing the absolute values of yaw, pitch, and roll against a configurable angular threshold. The node must support verbose mode for debugging.

D4.3.2 — Speech Event Package

The Speech Event package (originally developed across CSSR4Africa Deliverable D4.2.3 and CSSR4Africa Deliverable D4.3.2) must deliver two complementary audio-processing capabilities within a single ROS2 package. The standalone Sound Detection module (D4.2.3) is removed; sound detection signal processing is instead absorbed into this package.

The *Speech Recognition* node must replace the original keyword-spotting approach with a full speech recognition pipeline that detects speech onset and offset (using Voice Activity Detection) and produces verbatim transcriptions (using Automatic Speech Recognition). It must support two operating modes: an action server mode, in which transcription is triggered on demand by another module, and a continuous standalone mode, in which the node monitors audio indefinitely and publishes transcriptions as they occur. Audio processing must be enable/disable-able at runtime via a service interface.

The *Speech Localization* node must estimate the horizontal azimuth of any sound source from Pepper's four-microphone array and publish it continuously. Estimates must include a confidence score, and results falling below a configurable confidence threshold must be discarded. The node must run in parallel with the Speech Recognition node, both consuming the same raw audio stream from Pepper's microphone array.

5 Module Design

This section describes how each module is implemented at the code level, covering node architecture, data flow, and key algorithmic components.

D4.2.1 — Person Detection and Localization

Node Architecture

The package follows a two-file split that is used consistently across all modules in this work package. `person_detection_application.py` is the ROS2 entry point: it initialises `rclpy`, instantiates the node class defined in `person_detection_implementation.py`, spins it, and handles graceful shutdown on `KeyboardInterrupt`. All detection, tracking, and publishing logic lives in `person_detection_implementation.py`.

At initialisation the implementation class reads the YAML configuration file, declares all parameters as ROS2 node parameters, selects the correct camera topics based on the `camera` and `useCompressed` parameters, and loads the YOLOv11 ONNX model into an ONNX Runtime inference session. A warm-up pass over a zero-valued dummy input is run immediately after model loading to pre-load weights into memory and avoid first-frame latency. The `ByteTrack` tracker from the `supervision` library is also instantiated at this point, and synchronised RGB and depth subscribers are created using `ApproximateTimeSynchronizer`.

Per-Frame Data Flow

Each time a synchronised pair of colour and depth images arrives, the callback in the implementation class executes the following pipeline:

1. The incoming `sensor_msgs/Image` (or `CompressedImage`) is decoded into an OpenCV BGR array using `cv_bridge`.

2. The BGR array is converted to RGB, resized to the model's required input resolution, normalised to $[0, 1]$, transposed from HWC to CHW layout, and wrapped in a batch dimension to produce a float32 tensor of shape $[1, 3, H, W]$.

3. The tensor is passed to the ONNX Runtime session. The raw output is post-processed to extract bounding boxes in $[x_1, y_1, x_2, y_2]$ format, class probabilities, and confidence scores. Detections below `confidenceThreshold` and those not in `targetClasses` are discarded.

4. The surviving detections are wrapped in an `sv.Detections` object and passed to `tracker.update_with_detections()`. BYTETrack first matches high-confidence detections to existing tracks by IoU (Hungarian algorithm), then re-evaluates lower-confidence detections against still-unmatched tracks. Detections that receive a `tracker_id` are kept; transient detections without a stable track are dropped.

5. For each tracked bounding box, the pixel centroid $(\bar{u}, \bar{v})$ is computed as the midpoint of the box. A region of interest (ROI) of 10% of the box width and height centred on the centroid is extracted from the depth image. All valid (non-zero, finite) depth readings inside the ROI are averaged and converted from millimetres to metres to produce the z-coordinate of the 3D centroid.

6. The results are packed into a `dec_interfaces/PersonDetection` message and published on `/personDetection/data`. If `verboseMode` is enabled, annotated debug images are published to `/personDetection/debug` and `/personDetection/depth_debug`.

Camera Abstraction

The node supports three camera sources selected at configuration time. When `camera` is set to `realsense`, the node subscribes to the RealSense colour and pixel-aligned depth topics; the RealSense driver is launched with `align_depth.enable: True` so that every depth pixel corresponds exactly to its colour-image counterpart. When `camera` is set to `pepper`, the node subscribes to the NAOqi driver's front and depth camera topics. Any other value leaves no camera driver started, allowing the node to consume data replayed from a ROS2 bag file.

D4.2.2 — Face and Mutual Gaze Detection and Localization

Node Architecture

The package follows the same two-file pattern as D4.2.1. `face_detection_application.py` is the entry point and handles ROS2 lifecycle. `face_detection_implementation.py` contains the node class, which hosts two ONNX-based models — a `YOLOONNX` face detector and a `SixDrepNet` head-pose estimator — and subscribes to both the camera stream and the person detections published by D4.2.1.

At startup, the node loads the YAML configuration, instantiates the two ONNX Runtime sessions (`face_detection_goldYOLO.onnx` and `face_detection_sixdrepnet360.onnx`), and creates subscribers for the synchronised RGB-D stream and for `/personDetection/data`.

Per-Frame Data Flow

On each synchronised image pair the following steps are executed:

1. The colour image is decoded and passed to the GoldYOLO face detector. GoldYOLO is a lightweight YOLO variant optimised for face detection; it runs as an ONNX Runtime session and produces bounding boxes with confidence scores for all faces in the frame. Detections below the configured confidence threshold are discarded.

2. Each surviving face bounding box is cropped from the colour image and resized to 224×224 pixels to match the SixDRepNet input requirement.

3. The cropped patch is passed to SixDRepNet, which outputs a 6D rotation representation. The node converts this representation to a 3×3 rotation matrix and then decomposes it into Euler angles: yaw (left–right rotation), pitch (up–down tilt), and roll (in-plane rotation). A face is flagged as in mutual gaze when $|\text{yaw}|$, $|\text{pitch}|$, and $|\text{roll}|$ all lie within the `sixdrepnetHeadposeAngle` threshold.

4. Detected faces are associated with tracked persons from D4.2.1 using the Hungarian algorithm applied to the intersection-over-union (IoU) between face bounding boxes and person bounding boxes. This removes the need for an independent face tracker: face identity is inherited from the person track ID assigned by BYTETrack in D4.2.1.

5. For each associated face, the depth at the face centroid is sampled from the aligned depth image using the same ROI-averaging approach as in D4.2.1.

6. Results are packed into a `dec_interfaces/FaceDetection` message — containing bounding box, depth, track ID, head pose angles, and mutual-gaze flag — and published on `/faceDetection/data`.

D4.3.2 — Speech Event Package

Package Overview

The `speech_event` package hosts three ROS2 nodes that share the same audio source. Both subscribe to `/naoqi_driver/audio` (`naoqi_bridge_msgs/AudioBuffer`), which carries interleaved four-channel audio from Pepper's microphone array at 48 kHz. Each `AudioBuffer` message contains 4096 samples per channel (≈85 ms of audio). The two nodes operate in separate threads and do not share state.

Speech Recognition Node

The `speech_recognition` node is implemented in `speech_event_implementation.py`. Its design centers on a VAD state machine that gates a transcription pipeline.

Audio ingestion. On each `/audio` callback, the interleaved four-channel PCM buffer is deinterleaved and the first channel is extracted as a mono signal. The 48 kHz signal is downsampled to 16 kHz using a polyphase resampler (`scipy.signal.resample_poly`) before any further processing.

Voice Activity Detection. The 16 kHz audio is segmented into 512-sample chunks and fed to an `OnnxWrapper` that wraps the Silero VAD ONNX model. Silero VAD is a compact recurrent neural network that outputs, for each chunk, the probability that speech is present. A two-threshold hysteresis state machine governs the speech/silence boundary: a speech segment begins when the probability exceeds `speech_threshold` (default 0.7), and ends only after the probability remains below `neg_threshold` (default 0.35) for at least `min_silence_duration_ms` (default 800 ms). A pre-speech buffer of `pre_speech_buffer_ms` (default 200 ms) is prepended to each detected segment to avoid clipping the onset. Segments shorter than `min_speech_duration` (0.3 s) are discarded; segments longer than `max_speech_duration_s` (10 s) are truncated.

Transcription. Once a complete speech segment is confirmed, the 16 kHz buffer is submitted to a `ThreadPoolExecutor` to avoid blocking the ROS2 executor. The transcription is performed by Faster-Whisper (a CTranslate2 reimplementation of OpenAI Whisper), which runs with `float16` compute type on CUDA when available. The model ID (`deepdml/faster-whisper-large-v3-turbo-ct2` by default) is configurable via `whisper_model_id`. When `noise_cleaning_enabled` is `true`, the confirmed speech buffer is first passed through the `SpeechDenoiser` class (implemented in `speech_event_denoiser.py`), which applies bandpass filtering, harmonic notch filtering, and Wiener-filter spectral subtraction using a pre-recorded noise profile (`data/noise_profile.npy`) before the buffer is submitted for transcription.

Operating modes. The `action_server` parameter selects between two modes. In action server mode, the node exposes a `dec_interfaces/action/SpeechRecognition` action server on `/speech_recognition_action`; audio processing is active only while a goal is held, and the transcription is returned as the action result. In standalone/continuous mode, the

node processes audio indefinitely and publishes each transcription to `/speech_event/text`. A `std_srvs/SetBool` service on `/speech_event/set_enabled` allows audio processing to be paused and resumed at runtime in either mode. When `vad_always_active` is `true`, the VAD state machine continues running even while no action goal is active, so that speech onset is not missed at the start of a goal.

Speech Localization Node

The `speech_localization` node is implemented in `speech_event_localization.py`. It estimates the horizontal azimuth of a sound source using Steered Response Power with Phase Transform (SRP-PHAT) beamforming via the `pyroomacoustics` library.

Microphone geometry. Pepper's four microphones are modelled as a planar array with known XYZ coordinates. This geometry is passed to `pyroomacoustics` to compute the theoretical inter-microphone time delays of arrival for each candidate direction.

SRP-PHAT computation. On each audio callback, the 48 kHz four-channel buffer is passed through a Hann-windowed STFT with FFT size 1024. For each candidate azimuth θ in a grid of 36 uniformly spaced angles, the algorithm evaluates

$$P(\theta) = \sum_{m \neq n} \int_{f_{\min}}^{f_{\max}} \frac{G_{mn}(f)}{|G_{mn}(f)|} e^{j2\pi f \tau_{mn}(\theta)} \, df$$

where $G_{mn}(f)$ is the cross-power spectral density between microphones m and n and $\tau_{mn}(\theta)$ is the theoretical delay for direction θ at the speed of sound $c = 343$ m/s. The integration is restricted to a configurable band (default 500–2500 Hz) to avoid low-frequency noise and spatial aliasing. The azimuth that maximizes $P(\theta)$ is selected as the estimated source direction.

Confidence and filtering. A confidence score $(\text{peak} - \mu)/\mu$ (where μ is the spatial spectrum mean) is computed for each estimate. Estimates below `confidence_threshold` are discarded. An RMS amplitude gate (`intensity_threshold`) pre-filters silent frames before the SRP-PHAT computation. Optional circular-mean smoothing over a configurable history window reduces frame-to-frame jitter.

Published outputs. The node publishes the estimated azimuth (`std_msgs/Float32`), a unit direction vector (`geometry_msgs/Vector3Stamped`), the confidence score, an estimated source pose (`geometry_msgs/PoseStamped`), and an RViz arrow marker (`visualization_msgs/Marker`) for visualization.

Audio Recorder Node

The `audio_recorder` node is implemented in `speech_event_recorder.py`. It is a utility node intended for development and data collection: it subscribes to `/audio` and writes the incoming `naoqi\_bridge\_msgs/AudioBuffer` stream to disk as a multi-channel WAV file. When `split_channels` is `true`, one mono WAV file is written per microphone channel. The output path prefix, maximum recording duration, and channel-splitting behaviour are all configurable via the `audio_recorder` parameter group in `speech_event_configuration.yaml`. The node is not required for normal robot operation and is excluded from the main launch file.

6 Implementation

D4.2.1 — Person Detection and Localization

File Organization

Here is the file structure of the person detection package:

```
dec_system
└── person_detection
    ├── config
    │   └── person_detection_configuration.yaml
    ├── data
    │   └── pepper_topics.yaml
    ├── launch
    │   └── person_detection_launch_robot.launch.py
    ├── models
    │   └── person_detection_yolov11m.onnx
    ├── person_detection
    │   ├── __init__.py
    │   ├── person_detection_application.py
    │   └── person_detection_implementation.py
    ├── resource
    │   └── person_detection
    ├── package.xml
    ├── setup.cfg
    ├── setup.py
    ├── requirements.txt
    └── README.md
```

Figure 1: File structure of the person detection package.

Configuration File

The operation of the person detection node is controlled by the YAML configuration file `person_detection_configuration.yaml`. The available key-value pairs are listed below.

Launch File

The launch file `person_detection_launch_robot.launch.py` reads the `camera` value from the YAML configuration file at launch time and conditionally starts the appropriate camera driver:

- If `camera` is set to `realsense`, the `realsense2_camera_node` is launched with a 640×480 colour and depth profile at 15 fps, depth alignment enabled, and BEST_EFFORT QoS on image topics.
- If `camera` is set to `pepper`, the `naoqi_driver_node` is launched with the robot IP, port, and network interface specified in the launch file defaults.

Key	Value	Description
`camera`	`realsense` or `pepper`	Selects the camera source. Determines which ROS topics to subscribe to.
`useCompressed`	`true` or `false`	Whether to subscribe to compressed image topics. Compressed depth (compressedDepth) is used for RealSense; not supported for Pepper.
`imageTimeout`	`<number>`	Time (seconds) after which a warning is logged if no images have been received.
`verboseMode`	`true` or `false`	Enables diagnostic logging, OpenCV debug windows (if a display is available), and publication of debug image topics.
`confidenceThreshold`	`<number>`	Minimum YOLOv11 detection confidence (0.0–1.0) required to pass a detection to the tracker.
`targetClasses`	`[<string>, ...]`	List of COCO class names to track (e.g., `[person]`). Use `[all]` to track all 80 classes.
`trackThreshold`	`<number>`	ByteTrack activation threshold: minimum confidence for a detection to initialise or update an active track.
`trackBuffer`	`<integer>`	Number of consecutive frames a track may remain unmatched before being removed.
`matchThreshold`	`<number>`	IoU threshold used by ByteTrack when matching detections to existing tracks.
`frameRate`	`<integer>`	Expected video frame rate (fps), used by ByteTrack to scale track buffer timing.

Table 2: Configuration file key-value pairs for the person detection node.

- If `camera` is set to any other value, no camera driver is started; this allows the node to be used with data replayed from a ROS2 bag file.

In all cases the `person_detection` node is started after the camera driver.

Topics Subscribed

The person detection node subscribes to the following topics:

Camera	Topic Name	Message Type
`RealSenseCameraRGB`	`/camera/color/image_raw`	`sensor_msgs/Image`
`RealSenseCameraRGB` (Compressed)	`/camera/color/image_raw/compressed`	`sensor_msgs/CompressedImage`
`RealSenseCameraDepth`	`/camera/aligned_depth_to_color/image_raw`	`sensor_msgs/Image`
`RealSenseCameraDepth` (Compressed)	`/camera/aligned_depth_to_color/image_raw/compressed`	`sensor_msgs/CompressedImage`
`PepperFrontCamera`	`/naoqi_driver/camera/front/image_raw`	`sensor_msgs/Image`
`PepperDepthCamera`	`/naoqi_driver/camera/depth/image_raw`	`sensor_msgs/Image`

Table 3: Topics subscribed by the person detection node.

The colour and depth topics are subscribed using an `ApproximateTimeSynchronizer` with

a synchronisation slop of 0.1 s (RealSense) or 5.0 s (Pepper) to account for timing differences between the two streams.

Topics Published

Topic Name	Message Type	Description
`/personDetection/data`	`dec_interfaces/PersonDetection`	Array of tracked objects with track IDs, class names, confidence scores, 2D+depth centroids, and bounding box dimensions.
`/personDetection/debug`	`sensor_msgs/Image`	Annotated RGB debug image (published when `verboseMode` is true).
`/personDetection/depth_debug`	`sensor_msgs/Image`	Colourised depth debug image (published when `verboseMode` is true).

Table 4: Topics published by the person detection node.

D4.2.2 — Face and Mutual Gaze Detection and Localization

File Organization

The source code for conducting face detection, mutual gaze detection, and localization is structured into two primary components: `face_detection_application` and `face_detection_implementation`. The `face_detection_implementation` component encapsulates all the essential functionality required for executing face detection and mutual gaze detection using `SixDRepNet` for head pose estimation. Tracking is no longer handled by a dedicated tracking module; instead, face tracking relies on person detections received from the person detection node (D4.2.1), removing the need for Centroid Tracker or SORT. The `face_detection_application` component serves as the entry point, invoking the main functions to run the face detection node and executing the functions defined within `face_detection_implementation`.

Here is the file structure of the face detection package:

Configuration File

The operation of the face detection node is determined by the contents of the configuration file that contains a list of key-value pairs as shown on the table below. The configuration file is named `face_detection_configuration.yaml`.

Input File

There is no input file for the face detection node.

Output File

There is no output file for the face detection node. The node uses OpenCV to display the detected faces with bounding boxes and labels, and the mutual gaze detection.

```
dec_system
└── face_detection
    ├── config
    │   └── face_detection_configuration.yaml
    ├── data
    │   └── pepper_topics.yaml
    ├── launch
    │   └── face_detection_launch_robot.launch.py
    ├── models
    │   ├── face_detection_goldYOLO.onnx
    │   └── face_detection_sixdrepnet360.onnx
    ├── face_detection
    │   ├── __init__.py
    │   ├── face_detection_application.py
    │   ├── face_detection_implementation.py
    │   └── age_gender_detection.py
    ├── resource
    │   └── face_detection
    ├── package.xml
    ├── setup.cfg
    ├── setup.py
    ├── requirements.txt
    └── README.md
```

Figure 2: File structure of the face detection package.

Launch File

The launch file `face_detection_launch_robot.launch.py` reads the `camera` value from `face_detection_configuration.yaml` at launch time and conditionally starts the appropriate camera driver together with both the `person_detection` and `face_detection` nodes. It exposes one launch argument:

- `launch_camera`: controls whether the camera driver is started by this launch file. Set to `true` (default) when running on the physical robot; set to `false` when replaying data from a ROS2 bag file.

When `launch_camera:=true` and `camera` is `realsense` in the configuration file, the `realsense2_camera_node` is started with a 640×480 colour and depth profile at 15 fps, with depth alignment and time synchronisation enabled, and BEST_EFFORT QoS on image topics. When `camera` is `pepper`, the `naoqi_driver_node` is launched instead with the robot connection parameters embedded in the launch file. The `person_detection` node is always launched first so that face-to-person associations are available when the `face_detection` node starts.

Models

The face detection node uses two models for face detection and head pose estimation. The models are stored in the `models` directory. The models are:

Key	Value	Description
`camera`	`realsense`, `pepper`	Selects the camera source. Controls which image topics are subscribed to.
`useCompressed`	`true` or `false`	Whether to subscribe to compressed image topics.
`verboseMode`	`true` or `false`	Enables diagnostic logging and OpenCV debug display windows.
`imageTimeout`	`<number>`	Timeout (seconds) after which a warning is logged if no images are received.
`sixdrepnetConfidence`	`<number>`	Detection confidence threshold (0.0–1.0) for the YOLO face detector used by SixDRepNet.
`sixdrepnetHeadposeAngle`	`<number>`	Maximum angular deviation (degrees) in yaw, pitch, and roll for a face to be classified as in mutual gaze.
`requirePersonDetection`	`true` or `false`	When `true`, the node subscribes to `/personDetection/data` and uses person tracks from D4.2.1 to associate faces. When `false`, the node runs independently without person detection input.
`objectDetectionTimeout`	`<number>`	Maximum age (seconds) of person detection data before it is considered stale and discarded.

Table 5: Configuration file key-value pairs for the face detection node.

Models	Description
`face_detection_goldYOLO.onnx`	YOLO-based face detection model.
`face_detection_sixdrepnet360.onnx`	SixDRepNet head pose estimation model.

Table 6: Models used by the face detection node.

Topics File

For the test, a selected list of the topics for the robot is stored in the topics file. The topic files are written in the .yaml file format. The data file is written in key-value pairs where the key is the camera and the value is the topic. The topics file for the robot is named `pepper_topics.yaml`.

Topics Subscribed

The face detection node subscribes to the following topics:

Camera	Topic Name	Message Type
RealSenseCameraRGB	/camera/color/image_raw	sensor_msgs/Image
RealSenseCameraRGB (Compressed)	/camera/color/image_raw/compressed	sensor_msgs/ CompressedImage
RealSenseCameraDepth	/camera/aligned_depth_to_color/image_raw	sensor_msgs/Image
RealSenseCameraDepth (Compressed)	/camera/aligned_depth_to_color/image_raw/compressed	sensor_msgs/ CompressedImage
PepperFrontCamera	/naoqi_driver/camera/front/image_raw	sensor_msgs/Image
PepperDepthCamera	/naoqi_driver/camera/depth/image_raw	sensor_msgs/Image

Table 7: Topics subscribed by the face detection node.

Topics Published

The face detection node publishes the following topics:

Topic Name	Message Type	Description
/faceDetection/data	dec_interfaces/FaceDetection	An array of face records containing face labels, 3D image coordinates of the bounding box, width and height of the bounding box, and a boolean value indicating mutual gaze detection.

Table 8: Topics published by the face detection node.

D4.3.2 — Speech Event Package

File Organization

Here is the file structure of the speech event package:

```
dec_system
└── speech_event
    ├── config
    │   └── speech_event_configuration.yaml
    ├── data
    │   ├── noise_profile.npy
    │   └── pepper_topics.yaml
    ├── resource
    │   └── speech_event
    ├── speech_event
    │   ├── __init__.py
    │   ├── speech_event_application.py
    │   ├── speech_event_denoiser.py
    │   ├── speech_event_implementation.py
    │   ├── speech_event_localization.py
    │   └── speech_event_recorder.py
    ├── package.xml
    ├── requirements.txt
    ├── setup.cfg
    ├── setup.py
    └── README.md
```

Figure 3: File structure of the speech event package.

Configuration File

All three nodes share a single configuration file `speech_event_configuration.yaml`. Parameters are grouped under the ROS2 node name of each executable. The key-value pairs for the *Speech Recognition* node (`speech_recognition`) are listed in Table 9, and those for the *Speech Localization* node (`speech_localization`) are listed in Table 10.

Key	Value	Description
`microphone_topic`	`/audio`	ROS2 topic name for the raw audio input stream.
`input_sample_rate`	`48000`	Sample rate (Hz) of the incoming audio from Pepper's microphone array.
`sample_rate`	`16000`	Target sample rate (Hz) after downsampling, required by Silero VAD and Whisper.
`device`	`cuda` or `cpu`	Inference device for Faster-Whisper. Use `cuda` for GPU acceleration.

continued on next page

Table 9 – *continued from previous page*

Key	Value	Description
compute_type	float16	Numerical precision for Faster-Whisper inference (float16 or int8).
language	en	Language code passed to Whisper for transcription.
whisper_model_id	<string>	Faster-Whisper model identifier (e.g., deepdml/faster-whisper-large-v3-turbo-ct2).
speech_threshold	0.7	VAD probability threshold above which speech onset is declared.
neg_threshold	0.35	VAD probability threshold below which silence accumulates toward segment end.
min_silence_duration_ms	400	Minimum silence duration (ms) required to close a speech segment.
min_speech_duration	0.3	Minimum speech segment duration (s); shorter segments are discarded.
max_speech_duration_s	10.0	Maximum speech segment duration (s); longer segments are truncated.
pre_speech_buffer_ms	200	Audio lookback (ms) prepended to each segment to avoid onset clipping.
intensity_threshold	0.001	RMS amplitude gate; frames below this value are discarded before VAD.
action_server	true or false	When true, the node runs as a ROS2 action server; when false, it publishes transcriptions continuously.
vad_always_active	true or false	When true, VAD runs even while no action goal is active.
noise_cleaning_enabled	true or false	Enables the SpeechDenoiser pipeline before Whisper transcription.
noise_profile_path	<path>	Path to a .npy file containing a pre-recorded noise magnitude spectrum.
noise_alpha	0.5	Blending weight (0.0–1.0) between the rolling noise estimate and the static profile.

Table 9: Configuration parameters for the speech recognition node.

Key	Value	Description
microphone_topic	/audio	ROS2 topic name for the raw audio input stream.
sample_rate	48000	Sample rate (Hz) of the incoming audio.
speed_of_sound	343.0	Speed of sound (m/s) used to compute inter-microphone time delays.
nfft	1024	FFT size for the Hann-windowed STFT used in SRP-PHAT.
angular_resolution	36	Number of candidate azimuth angles uniformly distributed over 360°.

continued on next page

Table 10 – *continued from previous page*

Key	Value	Description
freq_range_min	500	Lower bound (Hz) of the frequency band used in SRP-PHAT integration.
freq_range_max	2500	Upper bound (Hz) of the frequency band used in SRP-PHAT integration.
num_chunks_for_localization	6	Number of audio chunks accumulated per localization estimate.
update_rate_hz	2.0	Target rate (Hz) at which localization estimates are published.
confidence_threshold	0.15	Minimum confidence score for an azimuth estimate to be published.
intensity_threshold	0.001	RMS amplitude gate; silent frames are rejected before SRP-PHAT.
enable_smoothing	true or false	Enables circular-mean temporal smoothing over the azimuth history.
smoothing_window	5	Number of consecutive estimates over which the circular mean is computed.

Table 10: Configuration parameters for the speech localization node.

Topics Subscribed

All three nodes in the package subscribe to the same audio topic: /naoqi_driver_audio

Node	Topic Name	Message Type
speech_recognition	/naoqi_driver/audio	naoqi_bridge_msgs/AudioBuffer
sound_localization	/naoqi_driver/audio	naoqi_bridge_msgs/AudioBuffer
audio_recorder	/naoqi_driver/audio	naoqi_bridge_msgs/AudioBuffer

Table 11: Topics subscribed by the speech event package nodes.

Topics Published

Topic Name	Message Type	Description
/speech_event/vad_speech_prob	std_msgs/Float32	Real-time Silero VAD speech probability for each 512-sample chunk.
/speech_event/text	std_msgs/String	Verbatim transcription published in standalone (continuous) mode.
/speech_localization/azimuth	std_msgs/Float32	Estimated horizontal azimuth (degrees) of the sound source.
/speech_localization/direction	geometry_msgs/Vector3Stamped	Unit direction vector of the estimated sound source.
/speech_localization/confidence	std_msgs/Float32	Confidence score for the current azimuth estimate.
/speech_localization/source_pose	geometry_msgs/PoseStamped	Estimated pose of the sound source in the robot frame.
/speech_localization/visualization	visualization_msgs/Marker	RViz arrow marker for visualizing the sound direction.

Table 12: Topics published by the speech event package nodes.

Action Server and Service Interfaces

Interface	Name	Description
Action Server	/speech_recognition_action	dec_interfaces/action/SpeechRecognition action; accepts a goal with a configurable maximum wait duration and returns the verbatim transcription as the action result.
Service	/speech_event/set_enabled	std_srvs/SetBool service; pauses (false) or resumes (true) audio processing at runtime in either operating mode.

Table 13: Action server and service interfaces of the speech recognition node.

References

[1] CSSR4Africa Deliverable D3.1: System Architecture. Technical report, Carnegie Mellon University Africa, 2025. Revision 2.6. Available online: https://cssr4africa.github.io/deliverables/CSSR4Africa_Deliverable_D3.1.pdf.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Yohannes Haile, Carnegie Mellon University Africa.

References

[1] CSSR4Africa Deliverable D3.1: System Architecture. Technical report, Carnegie Mellon University Africa, 2025. Revision 2.6. Available online: https://cssr4africa.github.io/deliverables/CSSR4Africa_Deliverable_D3.1.pdf.

Document History

Version 1.0

First draft.
Yohannes Haile.
01 June 2026.