UPANZI
DIGITAL PUBLIC INFRASTRUCTURE
NETWORK

Pepper Robot Tour for Digital Experience Center

D3.1 System Architecture

Due date: **28/02/2026**
Submission Date: **28/02/2026**
Revision Date: **02/06/2026**

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

Duration: **12 months**

Responsible Person: **Yohannes Haile**

Revision: **1.2**

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

Deliverable D3.1 specifies the system architecture for the Pepper Robot Tour at the Digital Experience Center (DEC) of the Upanzi Network at Carnegie Mellon University Africa. The system is implemented on **ROS2 Humble** and follows a modular, component-based software engineering approach using the publish-subscribe model and the ROS2 Managed Node (Lifecycle) pattern for graceful start-up and shutdown.

The architecture identifies two primary operational subsystems. The **Robot Sensing subsystem** (WP4) encompasses perception and attention modules: YOLO-based person detection, SixDrepNet-based face and mutual-gaze detection, VAD-aware speech recognition with an action-server interface, and a saliency-driven overt attention controller. The **Robot Behaviors subsystem** (WP5) encompasses the action and coordination modules: the BehaviorTree.CPP v4 Behavior Controller that orchestrates the full tour, a RAG-based Conversation Manager, a gesture execution module, an animated background behavior module, a text-to-speech module, and the RTAB-Map + Nav2 navigation and localization stack.

All inter-node communication is defined using custom `dec_interfaces` action, message, and service types together with standard ROS2 message types. Robot actuator commands use `naoqi_bridge_msgs` to interface with the Pepper hardware. Configuration for every node is loaded from YAML files at launch time without recompiling any package. The Behavior Controller acts as the central coordinator, consuming perceptual data from WP4 modules and issuing commands to all WP5 action servers through an XML-defined behavior tree.

Contents

1 Introduction

This document describes the system architecture for the Pepper Robot Tour for Upanzi's Digital Experience Center (DEC). The DEC system is developed as a spin-off of the **CSSR4Africa** (Culturally Sensitive Social Robotics for Africa) project and builds on its ROS1-based software stack, introducing targeted adaptations and new modules to meet the requirements of an autonomous, Pepper-driven tour environment at CMU-Africa. The codebase is maintained in the `pepper4dec` repository and is structured as a ROS2 workspace under `dec_system/`.

The architecture integrates the outputs of all work packages (WP1–WP6) into a coherent operational system. This document provides:

- A description of all subsystems and their constituent modules (ROS2 nodes and packages).
- Interface specifications for each module, including verified topic names, message types, services, and actions derived from the source code.
- A description of the data and control flow between subsystems.
- The software engineering standards and infrastructure applied across the system.

2 Requirements Definition

Composability and Modularity

- Each subsystem must be implemented as a collection of independent ROS2 nodes with clearly defined interfaces, enabling modules to be developed, tested, and replaced in isolation.
- All nodes must be independently launchable, and the full system must be startable from a single top-level launch file in the `dec_launch` package.

Communication Standards

- All inter-node communication must use ROS2 topics (for streaming data), services (for request–reply interactions), or actions (for long-running goal-oriented tasks), as appropriate to the interaction pattern.
- Custom message, service, and action types must be defined in the `dec_interfaces` package using `rosidl`.
- Pepper hardware commands must use `naoqi_bridge_msgs` types to interface with the NAOqi driver.

Configuration

- Each node must load all operating parameters from a YAML configuration file supplied via `--params-file` at launch time.
- Topic names that vary by deployment (robot vs. laptop, Pepper camera vs. RealSense) must be resolved from a shared `pepper_topics.yaml` data file rather than hard-coded.

Lifecycle Management

- All primary nodes must implement the ROS2 Managed Node lifecycle (`configure` → `activate` → `deactivate` → `cleanup`).
- Resource-intensive initialisation (model loading, database connections) must occur in `on_configure`; topic subscriptions must be created in `on_activate`.

3 System Overview

The DEC system is organized into the subsystems listed in Table 1. Table 2 enumerates all ROS2 packages in the `pepper4dec` repository.

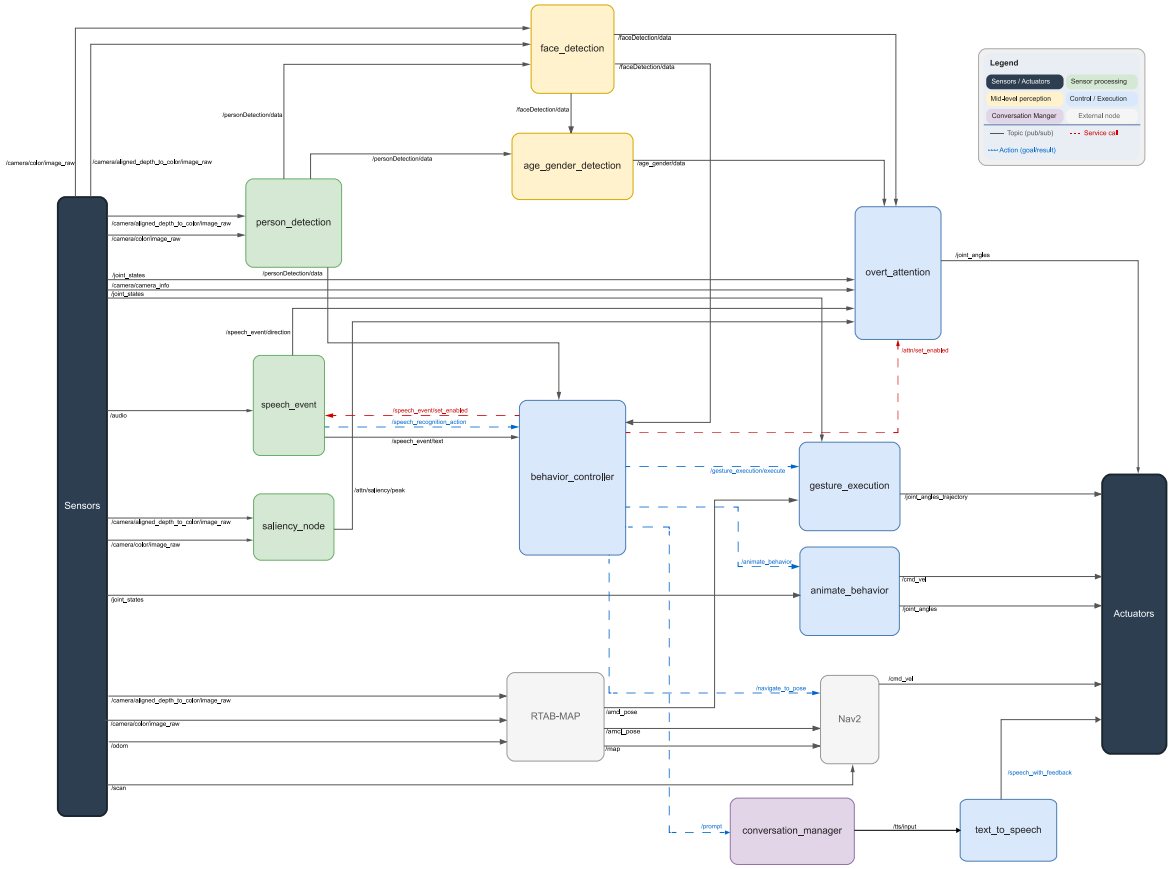

Figure 1: DEC ROS2 system architecture overview diagram showing the Robot Sensing (WP4) and Robot Behaviors (WP5) subsystems, their interconnections, and the data/control flow between modules.

Work Package	Subsystem	Description
WP1	Map & Knowledge Base	DEC environment map, location knowledge base, and cultural knowledge base.
WP2	Interaction Scenario	Use case scenarios, behavior specifications, and the XML behavior tree.
WP3	Systems Engineering	System architecture (this document), software standards, and installation manual.
WP4	Robot Sensing	Person detection, face/gaze detection, speech recognition, and overt attention.
WP5	Robot Behaviors	Behavior controller, conversation manager, gesture execution, animated behavior, text-to-speech, and navigation.
WP6	Use Case Evaluation	Demonstration deployment, visitor studies, and evaluation pipelines.

Table 1: DEC system subsystems by work package.

Package	WP	Role
`person_detection`	WP4	YOLOv11-based person detection and tracking from RGB-D camera streams.
`face_detection`	WP4	Face detection, SixDrepNet head-pose estimation, and mutual-gaze detection.
`speech_event`	WP4	VAD-based speech recognition with action-server and topic interfaces; optional sound-source localization.
`overt_attention`	WP4	Saliency- and face-driven head-tracking attention controller.
`behavior_controller`	WP5	Central mission interpreter; executes the XML behavior tree at 50 Hz via BehaviorTree.CPP v4.
`conversation_manager`	WP5	RAG-based dialogue module using ChromaDB + LLM; streams TTS incrementally.
`gesture_execution`	WP5	Executes iconic, deictic, bow, and nod gestures via inverse-kinematics-computed joint trajectories.
`animate_behavior`	WP5	Manages looping background animated behavior sequences and LED animations.
`text_to_speech`	WP5	TTS synthesis with multiple backends (naoqi_ros, Kokoro, ElevenLabs).
`pepper_navmap`	WP5	RTAB-Map SLAM for localization; Nav2 for global path planning and obstacle avoidance.
`dec_interfaces`	Infra	Custom ROS2 message, service, and action type definitions.
`dec_launch`	Infra	System-level launch files and startup configurations.

Table 2: ROS2 packages in the `pepper4dec` repository.

The Behavior Controller (WP5) acts as the central system coordinator. It subscribes to perceptual data from WP4, calls WP5 action servers to execute tour steps, and calls services to control attention, animation, and speech listening modes. All hardware commands reach the Pepper robot via `naoqi_bridge_msgs` topics consumed by the NAOqi driver.

4 Robot Sensing Subsystem (WP4)

The Robot Sensing subsystem provides the perceptual capabilities required for the DEC tour. It comprises four modules. These modules were migrated from ROS1 to ROS2 as part of DEC Deliverable D4.1 (Robot Sensing Migration) [1], building on the original implementations from CSSR4Africa Deliverable D4.2.1 [2], CSSR4Africa Deliverable D4.2.2 [3], and CSSR4Africa Deliverable D4.3.2 [4].

4.1 Person Detection

ROS2 node: `personDetection` **Package:** `person_detection` **Language:** Python (Lifecycle)

The Person Detection module performs real-time detection and tracking of visitors using a YOLOv11 ONNX model applied to RGB-D image streams. The camera source (RealSense or Pepper's built-in camera) and whether to use compressed topics are resolved from `pepper_topics.yaml` at runtime. Each detected person is assigned a persistent tracking ID and described by centroid pixel coordinates, metric depth, bounding box dimensions, and detection confidence.

Topic	Message Type	Description
`/camera/color/image_raw_custom`	`sensor_msgs/Image`	RGB image stream (resolved from `pepper_topics.yaml`).
`/camera/aligned_depth_to_color/image_raw_custom`	`sensor_msgs/Image`	Aligned depth image stream for metric depth estimation.

Table 3: Person Detection — Topics Subscribed.

Topic	Message Type	Description
`/personDetection/data`	`dec_interfaces/PersonDetection`	Per-frame list of detected persons with tracking IDs, centroids, bounding boxes, confidence scores, and depth.
`/personDetection/debug`	`sensor_msgs/Image`	Debug visualisation image with bounding box overlays.
`/personDetection/depth_debug`	`sensor_msgs/Image`	Depth debug visualisation image.

Table 4: Person Detection — Topics Published.

4.2 Face and Mutual Gaze Detection

ROS2 node: `faceDetection` **Package:** `face_detection` **Language:** Python (Lifecycle)

The Face Detection module performs real-time multi-face detection, SixDrepNet-based head-pose estimation, and mutual-gaze evaluation. It subscribes to the `/personDetection/data` stream (optionally, when `require_person_detection` is enabled) to crop face regions, and directly to the RGB-D camera topics otherwise. For each detected face it reports a persistent tracking ID, pixel-space centroid, bounding box, and a boolean mutual-gaze flag. Optional age and gender estimation is also supported.

Topic	Message Type	Description
/personDetection/data	dec_interfaces/ PersonDetection	Person bounding boxes used to crop face regions (optional mode).
/camera/color/image_raw_custom	sensor_msgs/Image	RGB image stream (resolved from pepper_topics.yaml).
/camera/aligned_depth_to_color/image_raw_custom	sensor_msgs/Image	Aligned depth image for metric face distance.

Table 5: Face Detection — Topics Subscribed.

Topic	Message Type	Description
/faceDetection/data	dec_interfaces/ FaceDetection	Per-frame list of detected faces with tracking IDs, centroids, bounding boxes, and mutual-gaze flags.
/faceDetection/debug	sensor_msgs/Image	Debug visualisation with face bounding boxes.
/faceDetection/depth_debug	sensor_msgs/Image	Depth debug visualisation.

Table 6: Face Detection — Topics Published.

4.3 Speech Event

ROS2 node: speechEvent **Package:** speech_event **Language:** Python (Lifecycle)

The Speech Event module provides VAD-based speech recognition with a post-VAD noise reduction stage. It subscribes to the Pepper microphone audio stream on /naoqi_driver/audio (configurable), applies a voice activity detector to isolate speech segments, and transcribes them using the configured ASR backend. Two interfaces are exposed to downstream consumers: a ROS2 action server (/speech_recognition_action) for blocking goal-based listening requests from the Behavior Controller, and a publisher (/speech_event/text) for standalone topic-based consumption. A SetBool service allows the Behavior Controller to mute the recognizer during TTS playback. Optional sound-source localization is also supported.

Topic	Message Type	Description
/naoqi_driver/audio	naoqi_bridge_msgs/ AudioBuffer	Raw microphone audio stream from the Pepper robot (topic configurable via parameter).

Table 7: Speech Event — Topics Subscribed.

Topic	Message Type	Description
/speech_event/text	std_msgs/String	Transcribed visitor utterance, published after VAD isolation.
/speech_event/vad_speech_prob	std_msgs/Float32	Real-time VAD speech probability for monitoring.

Table 8: Speech Event — Topics Published.

Interface	Type	Description
/speech_recognition_action	dec_interfaces/action/ SpeechRecognition	Action server. Goal: wait (max seconds to listen). Result: transcription (empty string if timeout). Feedback: status (waiting / speech / transcribing).
/speech_event/set_enabled	std_srvs/ SetBool	Service. Enable or disable the speech recognizer (called by Behavior Controller before and after TTS playback).

Table 9: Speech Event — Action Server and Service.

4.4 Overt Attention

ROS2 node: `overtAttention` **Package:** `overt_attention` **Language:** Python (Lifecycle)

The Overt Attention module controls Pepper's head-tracking behavior using a saliency-driven unified attention system. It fuses face detection data, visual saliency peaks, and camera geometry to compute target head-joint angles, which are published on `/joint_angles` for the NAOqi driver. The Behavior Controller enables and disables the attention system via the `/attn/set_enabled` service.

Topic	Message Type	Description
`/faceDetection/data`	`dec_interfaces/ FaceDetection`	Face centroids and gaze flags used to compute head-tracking targets.
`/attn/saliency_peak`	`std_msgs/ Float32MultiArray`	Visual saliency peak coordinates from the saliency node.
`/camera/color/camera_info`	`sensor_msgs/ CameraInfo`	Camera intrinsics used to convert pixel targets to joint angles.
`/joint_states`	`sensor_msgs/ JointState`	Current head joint positions for smooth interpolation.

Table 10: Overt Attention — Topics Subscribed.

Topic	Message Type	Description
`/joint_angles`	`naoqi_bridge_msgs/ JointAnglesWithSpeed`	Target yaw and pitch joint angles commanded to Pepper's head via the NAOqi driver.
`/attn/target_angles`	`geometry_msgs/ Vector3`	Computed target angles published for monitoring and visualization.

Table 11: Overt Attention — Topics Published.

Service	Type	Description
`/attn/set_enabled`	`std_srvs/ SetBool`	Enable or disable the attention system. When disabled, the head returns to the default pose.

Table 12: Overt Attention — Service Provided.

5 Robot Behaviors Subsystem (WP5)

The Robot Behaviors subsystem provides all action, coordination, and navigation capabilities required to execute the DEC tour.

5.1 Behavior Controller

ROS2 node: `behaviorController` **Package:** `behavior_controller` **Language:** C++ (Lifecycle)

The Behavior Controller is the central system orchestrator. It is a ROS2 Lifecycle Node that loads an XML behavior tree at startup and ticks it at 50 Hz once activated, using BehaviorTree.CPP v4 with the `behaviortree_ros2` bridge. A companion plain `rclcpp::Node` (`behavior_controller_bt`) is used for all BT action and service clients. A `KnowledgeManager` singleton loads the environment and cultural knowledge bases from YAML files, supplying tour waypoints, gesture targets, and speech phrases at runtime. Live tree visualization is supported via Groot2.

The registered BT node types are: `AnimateBehavior`, `StopAnimateBehavior`, `SetOvertAttention`, `SetSpeechListening`, `Gesture`, `Navigate`, `SpeechRecognition`, `ConversationManager`, `SpeechWithFeedback`, `TTS`, `CheckFaceDetected`, and `ListenForSpeech`.

Topic	Message Type	Description
`/faceDetection/data`	`dec_interfaces/FaceDetection`	Visitor presence and mutual-gaze status, consumed by `CheckFaceDetected` and `ListenForSpeech` BT nodes.
`/speech_event/text`	`std_msgs/String`	Transcribed visitor utterances, consumed by `ListenForSpeech` BT node.

Table 13: Behavior Controller — Topics Subscribed.

Action / Service Called	Type	Description
/speech_recognition_action	dec_interfaces/action/ SpeechRecognition	Listen for visitor speech for up to wait seconds (SpeechRecognition BT node).
/conversation_manager	dec_interfaces/action/ ConversationManager	Submit visitor utterance for RAG-based response and intent classification (ConversationManager BT node).
/tts	dec_interfaces/action/ TTS	Synthesise and play a speech utterance, blocking until playback completes (TTS BT node).
animate_behavior	dec_interfaces/action/ AnimateBehavior	Start a looping background animation for the specified duration (AnimateBehavior BT node).
/gesture_execution/execute	dec_interfaces/action/ Gesture	Execute a named gesture (iconic, deictic, bow, nod) (Gesture BT node).
/navigate_to_pose	nav2_msgs/action/ NavigateToPose	Send a navigation goal to the Nav2 stack (Navigate BT node).
/naoqi_driver/speech_with_feedback	naoqi_bridge_msgs/action/ Say	Direct NAOqi TTS with word-timing feedback (SpeechWithFeedback BT node).
animate_behavior/stop	std_srvs/ Trigger	Immediately halt the current background animation (StopAnimateBehavior BT node).
/attn/set_enabled	std_srvs/ SetBool	Enable or disable the overt attention system (SetOvertAttention BT node).
/speech_event/set_enabled	std_srvs/ SetBool	Mute or unmute speech recognition during TTS playback (SetSpeechListening BT node).

Table 14: Behavior Controller — Actions and Services Called.

5.2 Conversation Manager

ROS2 node: conversationManager **Package:** conversation_manager **Language:** Python (Lifecycle)

The Conversation Manager provides natural dialogue capability through a Retrieval-Augmented Generation (RAG) pipeline backed by ChromaDB and an OpenAI-compatible LLM. When the Behavior Controller submits a visitor utterance as an action goal to /conversation_manager, the module retrieves relevant context from the Upanzi knowledge base, streams each answer sentence to /tts/input for immediate playback, and returns the full response text, classified intent, and confidence score as the action result.

Action Server	Type	Description
/conversation_manager	dec_interfaces/action/ ConversationManager	Goal: prompt (visitor utterance). Result: response, intent, confidence. Feedback: status (searching / generating).

Table 15: Conversation Manager — Action Server.

Topic	Message Type	Description
/tts/input	std_msgs/String	Answer sentences streamed incrementally to the Text-to-Speech module for immediate playback while the LLM is still generating.

Table 16: Conversation Manager — Topics Published.

5.3 Gesture Execution

ROS2 node: gestureExecution **Package:** gesture_execution **Language:** Python (Lifecycle)

The Gesture Execution module computes inverse-kinematics-based joint trajectories for Pepper's arms, hands, and body to produce expressive gestures. It exposes a single action server (/gesture_execution/execute) that accepts a gesture type (iconic, deictic, bow, or nod), gesture name, duration, and an optional 3D target coordinate for deictic pointing. Joint trajectories are published on /joint_angles_trajectory for the NAOqi driver. The module also subscribes to /joint_states for current positions and to /robotLocalization/pose for deictic target resolution.

Topic	Message Type	Description
/joint_states	sensor_msgs/ JointState	Current joint positions used as the starting point for trajectory generation.
/robotLocalization/pose	geometry_msgs/ Pose2D	Current robot pose, used to resolve 3D deictic target coordinates into joint angles.

Table 17: Gesture Execution — Topics Subscribed.

Topic	Message Type	Description
/joint_angles_trajectory	naoqi_bridge_msgs/ JointAnglesTrajectory	Joint trajectory commands sent to Pepper's arms and body via the NAOqi driver.
/gesture_execution/visualization	visualization_msgs/ Marker	RViz marker for debugging deictic target positions.

Table 18: Gesture Execution — Topics Published.

Action Server	Type	Description
/gesture_execution/execute	dec_interfaces/action/ Gesture	Goal fields: gesture_type (iconic / deictic / bow / nod), gesture_name, gesture_duration (ms), location_x/y/z (for deictic). Feedback: elapsed_seconds.

Table 19: Gesture Execution — Action Server.

5.4 Animated Behavior

ROS2 node: animateBehavior **Package:** animate_behavior **Language:** Python (Lifecycle)

The Animated Behavior module manages looping background animation sequences and LED cascades that give Pepper a lively, idle appearance. It exposes an action server (animate_behavior)

that accepts a behavior type (`All`, `body`, `arms`, `hands`, or `idle`) and an optional duration. Joint angles are smoothed and published on `/joint_angles` for the NAOqi driver. The Behavior Controller cancels an active goal or calls `animate_behavior/stop` to halt the animation immediately.

Topic	Message Type	Description
`/joint_states`	`sensor_msgs/JointState`	Current joint positions, used as the starting point for smooth animation interpolation.

Table 20: Animated Behavior — Topics Subscribed.

Topic	Message Type	Description
`/joint_angles`	`naoqi_bridge_msgs/JointAnglesWithSpeed`	Smoothed joint angle commands published at ˜30 Hz to produce idle body animations.
`/cmd_vel`	`geometry_msgs/Twist`	Base velocity commands for rotation-type animations.

Table 21: Animated Behavior — Topics Published.

Interface	Type	Description
`animate_behavior`	`dec_interfaces/action/AnimateBehavior`	Action server. Goal: `behavior_type`, `selected_range` (0–1), `duration_seconds` (0 = infinite). Feedback: `current_limb`, `gestures_completed`, `elapsed_time`.
`animate_behavior/stop`	`std_srvs/Trigger`	Service. Immediately halts the active animation and returns the robot to a neutral pose.
`/naoqi_driver/run_led`	`naoqi_bridge_msgs/action/RunLed`	Action client. Drives LED cascade wave animations on Pepper's eye rings.

Table 22: Animated Behavior — Action Server, Service, and Action Client.

5.5 Text-to-Speech

ROS2 node: `textToSpeech` **Package:** `text_to_speech` **Language:** Python

The Text-to-Speech module converts text to speech and plays it through the Pepper robot's speakers. Multiple synthesis backends are supported: `naoqi_ros` (publishes plain text to `/speech` for the NAOqi driver), `kokoro_local`, `kokoro_pepper`, `elevenlabs_local`, and `elevenlabs_pepper`. The active backend is selected via the YAML configuration file. The module exposes an action server (`/tts`) that blocks until playback is fully complete, making it safe to sequence in a behavior tree. Barge-in via VAD is honoured: if the visitor speaks during playback, remaining sentences are dropped and the action returns success.

Action Server	Type	Description
`/tts`	`dec_interfaces/action/TTS`	Goal: `text` (string, may contain multiple sentences). Result: `success`, `message`. Feedback: `status` (`queuing` / `speaking`).

Table 23: Text-to-Speech — Action Server.

Topic	Message Type	Description
/speech	std_msgs/String	Plain-text utterances published to the NAOqi driver for on-board synthesis (naoqi_ros backend only).

Table 24: Text-to-Speech — Topics Published (naoqi_ros backend).

5.6 Navigation and Localization

Package: pepper_navmap **Language:** C++ / Python (ROS2 wrappers)

The Navigation and Localization module integrates two components. **RTAB-Map** provides simultaneous localization and mapping using the Intel RealSense RGB-D camera, publishing the odom→map transform. **Nav2** (ROS2 Navigation Stack) uses the RTAB-Map-generated map and AMCL-based localization to plan and execute collision-free paths. The Behavior Controller sends navigation goals via the Nav2 BasicNavigator API, which calls the standard /navigate_to_pose action server. Pre-built maps are stored in pepper_navmap/map/ and loaded at launch.

Topic	Message Type	Description
/camera/color/image_raw_custom	sensor_msgs/Image	RGB image for RTAB-Map visual odometry and loop closure.
/camera/aligned_depth_to_color/image_raw_custom	sensor_msgs/Image	Aligned depth image for 3D point cloud construction.
/pepper/odom	nav_msgs/Odometry	Wheel odometry from Pepper, fused with visual odometry by RTAB-Map.

Table 25: Navigation and Localization — Topics Subscribed.

Topic	Message Type	Description
/map	nav_msgs/OccupancyGrid	Occupancy grid map published by RTAB-Map.
/tf	tf2_msgs/TFMessage	Transform tree including odom→map frame.
/cmd_vel	geometry_msgs/Twist	Velocity commands issued by Nav2 to Pepper's mobile base.

Table 26: Navigation and Localization — Topics Published.

Action Server	Type	Description
/navigate_to_pose	nav2_msgs/action/NavigateToPose	Standard Nav2 action server. Accepts a geometry_msgs/PoseStamped goal; returns on arrival or failure. Called by the Behavior Controller via BasicNavigator.

Table 27: Navigation and Localization — Action Server.

References

[1] DEC Deliverable D4.1: Robot Sensing Migration. Technical report, Carnegie Mellon University Africa, 2026. Revision 1.0. Available online: https://cssr4africa.github.io/deliverables/CSSR4Africa_Deliverable_D4.1.pdf.

[2] CSSR4Africa Deliverable D4.2.1: Person Detection and Localization. Technical report, Carnegie Mellon University Africa, 2025. Revision 2.1. Available online: https://cssr4africa.github.io/deliverables/CSSR4Africa_Deliverable_D4.2.1.pdf.

[3] CSSR4Africa Deliverable D4.2.2: Face & Mutual Gaze Detection and Localization. Technical report, Carnegie Mellon University Africa, 2025. Revision 1.5. Available online: https://cssr4africa.github.io/deliverables/CSSR4Africa_Deliverable_D4.2.2.pdf.

[4] CSSR4Africa Deliverable D4.3.2: Speech Event. Technical report, Carnegie Mellon University Africa, 2025. Revision 1.5. Available online: https://cssr4africa.github.io/deliverables/CSSR4Africa_Deliverable_D4.3.2.pdf.

Principal Contributors

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

Yohannes Haile, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Yohannes Haile.
28 February 2026.

Version 1.1

Updated to reflect actual ROS2 package structure from the `pepper4dec` repository. Added Overt Attention, Animated Behavior, and Text-to-Speech module specifications.
Yohannes Haile.
01 May 2026.

Version 1.2

All interface tables corrected from verified source code.
Updated topic names: `/joint_angles`, `/joint_angles_trajectory`, `/speech_event/text`, `/attn/set_enabled`, `/speech_event/set_enabled`
Updated message types: `naoqi_bridge_msgs/JointAnglesTrajectory`, `naoqi_bridge_msgs/JointAnglesWithSpeed`.
Updated action servers: `/gesture_execution/execute`, `animate_behavior`, `/tts`, `/speech_recognition_action`.
Updated service names and camera topics (`/camera/color/image_raw_custom`, `/camera/aligned_depth_to_color/image_raw_custom`) throughout.
Yohannes Haile.
01 June 2026.