UPANZI
DIGITAL PUBLIC INFRASTRUCTURE
NETWORK

Pepper Robot Tour for Digital Experience Center

D5.1 Robot Actuator Migration

Due date: **01/05/2026**
Submission Date: **06/05/2026**
Revision Date: **15/08/2026**

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

Duration: **12 months**

Responsible Person: **Yohannes Haile**

Revision: **1.6**

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 5 (WP5) addresses the migration of the actuator control software modules from the CSSR4Africa project from ROS1 to ROS2 Humble, as part of the DEC Pepper Robot Tour project. The objective is to ensure full compatibility with the ROS2 framework while improving upon the original ROS1 implementations where necessary. This deliverable covers the complete migration and redesign of the following actuator-related behavior modules:

- Animate Behavior package
- Gesture Execution package
- Overt Attention package
- Text-to-Speech package

Each of these packages provides a distinct capability in Pepper's behavioral repertoire: lifelike idle animations, purposeful named gestures, visual attention-driven head orientation, and neural text-to-speech synthesis. Together they constitute the actuator control layer through which higher-level behavior orchestration commands the robot's joints, base, and speech synthesizer.

Contents

1 Introduction

This document describes the migration and redesign of Work Package 5 actuator control modules from ROS1 to ROS2 Humble as part of the DEC Pepper Robot Tour project. The CSSR4Africa project originally developed these modules under ROS1 Noetic using the `rospy` API and `catkin` build system; this deliverable ports them to ROS2 Macenski et al. (2022). Three of the four packages (Animate Behavior, Gesture Execution, Overt Attention) were subsequently rewritten a second time from their initial `rclpy`/`ament_python` port into C++ (`rclcpp`/`ament_cmake`) lifecycle nodes; Text-to-Speech remains a Python `rclpy` package.

Work Package 5 covers the behavioral output capabilities of the robot, including smooth idle animation, execution of named gesture primitives, visual attention and head orientation, and text-to-speech synthesis. These modules were substantially redesigned during migration to take advantage of new neural synthesis backends, a multi-node attention architecture, Bézier-based motion interpolation, and LED animation, all improvements over the original ROS1 implementations.

The migration encompasses the following packages:

- **Animate Behavior**: Generates smooth, randomized lifelike animations for Pepper's upper body, hands, and base. The package controls both LED eye animations and gestural body movements using exponential joint-angle smoothing. It exposes a ROS2 action server so that a behavior controller can trigger and monitor idle animation sequences. Its ROS1 predecessor is specified in CSSR4Africa Deliverable D5.2, Animate Behaviour Subsystem.
- **Gesture Execution**: Executes named gesture primitives (e.g., *welcome*, *wave*, *shake*) on Pepper using Bézier-interpolated joint trajectory playback. Gesture definitions are loaded from a YAML file and executed via a ROS2 action server. The package includes inverse kinematics utilities for computing deictic pointing directions. Its ROS1 predecessor is specified in CSSR4Africa Deliverable D5.5.1.1, Gesture Execution.
- **Overt Attention**: Drives Pepper's head toward visually salient targets in a biologically plausible manner. The package implements a three-node architecture: a Boolean Map Saliency Zhang and Sclaroff (2013) node computes bottom-up visual attention from the camera stream; an attention controller node (the `overt_attention` node) arbitrates between face detections and saliency peaks using a priority scheme with Inhibition of Return (IOR) Itti and Koch (2001); and a Visualization node overlays attention state on the camera image for debugging. Its ROS1 predecessor is specified in CSSR4Africa Deliverable D5.3, Attention Subsystem.
- **Text-to-Speech**: Synthesizes and plays speech on Pepper's speakers. The package supports five synthesis backends ranging from Pepper's on-board NAOqi Aldebaran Robotics (2024) engine to neural Kokoro-82M Hexgrad (2025) and ElevenLabs ElevenLabs (2024) cloud synthesis. A ROS2 action server interface blocks until speech playback completes, and the speech-recognition microphone is muted during playback so the robot does not transcribe its own speech. Its ROS1 predecessor is specified in CSSR4Africa Deliverable D5.5.2.4, Integrated Text to Speech Conversion.

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`, from XML launch files to Python-based ROS2 launch files, and from ROS1 message definitions to ROS2 interface packages defined in `dec_interfaces`. Custom action definitions (`dec_interfaces/action/Gesture` and `dec_interfaces/action/TTS`) replace the ROS1 service interfaces used in the original CSSR4Africa implementation.

2 Requirements Definition

This section defines the migration requirements for Work Package 5 (WP5). The primary requirement is that each migrated ROS2 node must preserve or improve upon 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` or `rclcpp` and conform to the ROS2 (Humble) API.
- Build system files must be migrated from `catkin` to `ament_python` (Python packages) or `ament_cmake` (C++ packages), with `package.xml` updated to format 3.
- Launch files must be rewritten as Python-based ROS2 launch files, replacing ROS1 XML `.launch` files.
- Custom action definitions must be provided via the `dec_interfaces` package 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 that represent intentional improvements documented in this deliverable.
- Gesture primitives and motion profiles must be preserved. Where interpolation logic has been revised to use Bézier curves (Gesture Execution), the improvement is documented as an intentional redesign.
- The attention system must retain the ability to orient Pepper's head toward detected persons. The redesigned multi-node architecture is an intentional improvement over the single-node ROS1 implementation.

Module-Specific Requirements

- **Animate Behavior**: The node must generate randomized idle animations covering the upper body, hands, arms, and base rotation of the Pepper robot. It must use exponential joint-angle smoothing to ensure fluid motion, and must publish joint angle commands at a configurable rate. LED eye animations must be controllable independently of body gesture animations. The node must expose a ROS2 action server interface for behavior controller integration.
- **Gesture Execution**: The node must load gesture definitions from a YAML file and execute them on request via a ROS2 action server. Motion must be interpolated using Bézier curves to guarantee smooth joint trajectories. The node must validate all joint angles against Pepper's safety limits before execution, and must publish RViz2 visualization markers for debugging. Inverse kinematics utilities must support deictic pointing gestures.

- **Overt Attention**: The *saliency_node* must compute bottom-up visual saliency from the camera stream using Boolean Map Saliency (BMS) and publish detected salient peak locations. The *overt_attention* node must arbitrate between face detection input and saliency peaks according to a configurable priority scheme, apply Inhibition of Return to suppress recently visited locations, and publish smoothed head joint angle commands. The *attention_visualization* node must overlay attention state on the camera image. The system must be enable/disable-able at runtime via a `std_srvs/SetBool` service.
- **Text-to-Speech**: The node must accept text strings on a ROS2 topic and synthesize speech using one of five configurable backends. It must expose a ROS2 action server that blocks until playback is complete. It must mute the speech-recognition microphone for the duration of playback so the robot does not transcribe its own 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.

Scope Exclusions

- Unit tests for each module are not included in this deliverable; the migration is focused on the functional nodes.
- The simulator is not supported; all nodes target the physical Pepper robot.
- The Actuator Tests module and the Robot Mission Interpreter are excluded from this deliverable. The Behavior Controller is addressed in a subsequent deliverable.

3 Migration Overview

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

Common Migration Changes

Table 1: Per-module migration summary.

Module	Migration Approach	Redesigned?
Animate Behavior	Direct port with ROS2 API updates; LED animation control added; exponential joint smoothing introduced; ROS2 action server interface added.	Partial
Gesture Execution	Direct port; motion interpolation upgraded from linear to Bézier curves; inverse kinematics utilities added for pointing; ROS2 action server interface added.	Partial
Overt Attention	Fully redesigned as a three-node architecture. Boolean Map Saliency replaces simple face-tracking; Inhibition of Return mechanism added; priority-based attention arbitration replaces threshold-based switching.	Yes
Text-to-Speech	Fully redesigned; NAOqi on-board TTS is one of five configurable backends. Neural synthesis (Kokoro-82M, ElevenLabs) added. Microphone muting during playback added. ROS2 action server interface added.	Yes

4 Module Specifications

This section describes the expected behaviour and functional requirements of each module delivered in Work Package 5.

Animate Behavior

The Animate Behavior node must generate continuous, randomized, lifelike idle animations for the Pepper robot when it is not executing a specific task. Animations must cover the full upper body (shoulders, elbows, wrists), hands, and base rotation. The node must produce smooth, non-jerky joint-angle motion transitions. LED eye animations must be controllable via a separate configuration group, producing a wave-like brightening and dimming effect across Pepper's eye LEDs. The node must support multiple animation modes selectable at launch: *all* (body and base), *body*, *hands*, *arms*, *rotation*, and *home* (return to rest pose). Gesture timing must be randomized within a configurable interval range to avoid repetitive or mechanical-looking motion. The node must expose a ROS2 action server so that a higher-level behavior controller can trigger, monitor, and cancel animation sequences.

Gesture Execution

The Gesture Execution node must execute named gesture primitives on demand via a ROS2 action server. Gesture definitions (comprising joint names, angle keyframes in radians, and associated timestamps) must be loaded from a YAML file at startup, allowing new gestures to be added without modifying source code. Motion between keyframes must be interpolated using Bézier curves to ensure smooth, human-like trajectories. The node must validate all commanded joint angles against Pepper's kinematic safety limits before execution and must reject unsafe goals with an appropriate action result. For deictic pointing gestures, the node must compute a target joint configuration using inverse kinematics relative to the robot's current pose obtained from the localization module. RViz2 visualization markers must be published during execution for debugging and demonstration purposes.

Overt Attention

The Overt Attention package must drive Pepper's head toward visually salient targets in real time, mimicking biological attention behavior. The package is delivered as three coordinated ROS2 nodes.

The *saliency_node* must process the camera image stream to compute a bottom-up visual saliency map using the Boolean Map Saliency (BMS) algorithm. It must detect and publish the top-N salient peak locations in the image, with configurable downsampling resolution and processing rate.

The *overt_attention* node must arbitrate between face detections received from the face detection module (D4.2.2) and salient image locations received from the saliency node. Face detections must take priority over saliency; among multiple faces, an engaged face (determined by mutual gaze from D4.2.2) must receive a configurable priority bonus. The node must apply Inhibition of Return (IOR), a biologically motivated suppression mechanism that temporarily reduces the salience of recently attended locations to prevent the robot's gaze from repeatedly returning to the same spot. Head movement commands must be exponentially smoothed before publication to reduce jitter.

The *attention_visualization* node must overlay attention targets, saliency maps, and IOR state on the camera image for diagnostic purposes.

Text-to-Speech

The Text-to-Speech node must receive text strings and synthesize audible speech on Pepper's speakers. The node must support five interchangeable synthesis backends selected at configuration time: *naoqi_ros* (Pepper's on-board ALTextToSpeech), *kokoro_local* (Kokoro-82M neural synthesis played on the laptop), *kokoro_pepper* (Kokoro-82M synthesis streamed to Pepper's speakers), *elevenlabs_local* (ElevenLabs cloud synthesis played on the laptop), and *elevenlabs_pepper* (ElevenLabs cloud synthesis streamed to Pepper's speakers). For Pepper-targeted backends, two playback methods must be supported: *file* (WAV transferred to robot via SCP and played via ALAudioPlayer) and *stream* (raw PCM chunks sent via NAOqi AudioDevice). The node must expose a ROS2 action server that blocks until playback completes, enabling callers to wait for speech to finish before proceeding. For the duration of playback, the node must mute the speech-recognition microphone (via the `/speech_event/set_enabled` service) so that the robot does not transcribe its own speech.

5 Module Design

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

Animate Behavior

Node Architecture

The package follows the C++ two-file split used consistently across the compiled modules in this work package. `animate_behavior_application.cpp` is the ROS2 entry point: it initialises `rclcpp`, constructs the `AnimateBehaviorNode` lifecycle node class defined in `animate_behavior_implementation.cpp`, and spins it. All animation logic, timing state, and publishing resides in `animate_behavior_implementation.cpp`.

As a lifecycle node, initialisation is split across transitions. On `configure`, the node reads its ROS2 parameters, builds the per-limb joint definitions (names, soft limits, home pose, randomisation factors), and creates its lifecycle publishers (`/joint_angles`, `/cmd_vel`), the `/animate_behavior` action server, the `/animate_behavior/stop` service, and the `/naoqi_driver/run_led` action client. On `activate`, it activates the lifecycle publishers, subscribes to `/joint_states`, starts the animation and feedback timers, and, if `led_enabled`, kicks off the face-LED cascade animation. `deactivate` stops any active animation, cancels the timers, and destroys the joint-state subscription; `cleanup` tears down the publishers, action server, stop service, and LED client.

A fast animation timer (30 Hz by default, `gesture_update_rate`) drives the joint smoothing loop; a gesture scheduling timer triggers the selection of the next random gesture after a randomized interval drawn from [`gesture_interval_min`, `gesture_interval_max`].

Animation Data Flow

An `/animate_behavior` action goal (behavior type, range, duration) starts an animation; while active, on each animation tick the node executes the following steps:

1. The current joint state is read from the most recently received `/joint_states` message.
2. A target joint configuration is selected based on the requested behavior type and a randomly chosen gesture primitive from the mode's gesture pool.
3. Each animation step advances the joints smoothly from their current state toward the selected target configuration, using exponential smoothing (`gesture_smoothing_factor`) and a configured motion speed (`gesture_motion_speed`), producing fluid, non-jerky transitions rather than step changes.
4. The resulting joint-angle motion is packed into a `naoqi_bridge_msgs/JointAnglesWithSpeed` message and published to `/joint_angles`.
5. If base rotation is enabled for the current mode, a `geometry_msgs/Twist` velocity command is published to `/cmd_vel` at the configured rotation interval.
6. Elapsed-time feedback is reported to the action client on a separate feedback timer. The animation ends when the requested duration elapses, the goal is cancelled, or the `/animate_behavior/stop` service is called, at which point velocity and LEDs are zeroed.

LED Animation

LED animations run independently of joint motion, driven via an action client to the NAOqi driver's `/naoqi_driver/run_led` action server. When `led_enabled` is `true`, the node sends LED colour commands to Pepper's eye LEDs, producing a bottom-to-top brightening wave (`led_white_step` delay per layer) followed by a top-to-bottom darkening wave (`led_dark_step` delay per layer), a full-white hold period (`led_white_hold`), and a dark pause (`led_dark_pause`) before the next cycle begins.

Gesture Execution

Node Architecture

The package follows the same C++ two-file pattern as Animate Behavior. `gesture_execution_application.cpp` is the ROS2 entry point and `gesture_execution_implementation.cpp` contains the `GestureExecutionNode` lifecycle node class. An additional module, `gesture_pepper_kinematics.cpp`, provides inverse kinematics helper functions for deictic pointing gestures. A separate `gesture_test_visualization.cpp` tool provides a standalone RViz2 visualization utility for testing gesture trajectories offline.

On `configure`, the node loads gesture definitions from `data/gesture.yaml` and the robot topic mapping from `data/pepper_topics.yaml` (both file paths are hardcoded in the code; see Configuration File, below), initialises kinematics/home-pose state, and creates its lifecycle publishers and the `dec_interfaces/action/Gesture` action server on `/gesture_execution`. On `activate`, it subscribes to `/joint_states` and `/localization` (the fused robot pose used for deictic pointing); `deactivate` destroys those subscriptions, and `cleanup` tears down the publishers and action server.

Gesture Execution Data Flow

When an action goal is received, the node executes the following pipeline:

1. The requested gesture name is looked up in the loaded gesture dictionary. If the gesture is not found, the action is immediately aborted.
2. The keyframe sequence (joint names, angle arrays in radians, timestamps) is extracted from the gesture definition. Deictic, bow, and nod gestures are instead computed directly: deictic gestures resolve a target joint configuration via inverse kinematics against the current `/localization` pose.
3. A Bézier curve is fitted through the keyframe angles for each joint to produce a smooth continuous trajectory. This replaces the linear interpolation used in the ROS1 implementation and eliminates velocity discontinuities at keyframe boundaries.
4. All waypoints on the Bézier trajectory are validated against Pepper's kinematic joint limits. If any waypoint violates a limit, the goal is rejected before execution begins.
5. Joint angle commands are published as a `naoqi_bridge_msgs/JointAnglesTrajectory` message to `/joint_angles_trajectory`, progressing along the Bézier trajectory at the speed profile encoded in the gesture definition's timestamps. Elapsed-time feedback messages are sent to the action client on a separate timer during execution.

6. On completion, the action result is returned to the caller.

RViz2 markers (`visualization_msgs/Marker`, published to `/gesture_execution/visualization`) are published throughout execution of deictic gestures, displaying the target point, shoulder position, and pointing vector for diagnostic purposes.

Overt Attention

Package Overview

The `overt_attention` package hosts three independent ROS2 nodes, all defined in `overt_attention_interface.h` and implemented across `overt_attention_saliency.cpp`, `overt_attention_implementation.cpp`, and `overt_attention_visualization.cpp` (C++, not the original Python port). Only the attention controller, `OvertAttentionNode`, is a lifecycle node; `SaliencyNode` and `VisualizationNode` are plain nodes. All three are launched together by `attention_system.launch.py`; the RealSense camera driver can optionally be co-launched via `realsense_camera.launch.py`.

Saliency Node

`SaliencyNode` (`overt_attention_saliency.cpp`) implements the Boolean Map Saliency (BMS) algorithm to compute a bottom-up visual saliency map from the camera stream.

Image preprocessing. On each camera frame, the incoming `sensor_msgs/Image` (or `CompressedImage`) is decoded and downsampled to a fixed resolution (160×120, no longer YAML-configurable) to reduce computation. When `use_depth_weighting` is enabled, depth values from the aligned depth stream (also subscribed, raw or compressed depending on `use_compressed`) are used to reduce the weight of objects that are very close (below `depth_min_m`) or very far (above `depth_max_m`) from the robot.

Saliency computation. The BMS algorithm decomposes the image into Boolean maps across multiple thresholds, applies flood-fill-based background suppression, and combines them into a single saliency map. The top-N peaks (`num_peaks`, default 5) are extracted with a fixed minimum inter-peak distance constraint (50 pixels, no longer YAML-configurable) to avoid reporting redundant nearby detections. Peaks below `min_peak` threshold are discarded. The saliency map is recomputed at `process_hz` (default 1 Hz).

Published outputs. Peak locations are packed into a `std_msgs/Float32MultiArray` and published to `/overt_attention/saliency_peak`. When `publish_map` is `true`, the annotated saliency map image is also published.

Attention Controller Node

`OvertAttentionNode` (`overt_attention_implementation.cpp`) arbitrates between competing attention targets and commands Pepper's head. It is launched under the runtime node name `overt_attention`, set explicitly by `attention_system.launch.py` (overriding the class's constructor-default node name, which is also `overt_attention`). On `configure` it creates its lifecycle publishers, the `/overt_attention/set_enabled` service, and subscriptions to face detections, saliency peaks, camera info, and joint states; `activate`/`deactivate` (de)activate the publishers, and `cleanup` tears everything down.

Priority arbitration. On each update cycle, the node evaluates available attention candidates in the following priority order: (1) faces detected in mutual gaze, weighted by `engaged_priority_bonus`; (2) all other detected faces; (3) salient image peaks above `saliency_min_score`. The highest-priority candidate is selected as the current attention target unless a face-switch cooldown (`face_switch_cooldown`) or saliency-switch cooldown (`saliency_min_cooldown`) is active. If no face has been received for longer than `face_timeout` (default 2.0 s), the node falls back to saliency.

Inhibition of Return. Each time the robot attends to a location, that location is added to an IOR list with a full suppression weight of `ior_max_suppression` (default 0.9). The suppression weight decays exponentially with half-life `ior_half_life` (default 3 s). Any attention candidate within `ior_radius_deg` (default 15°) of a suppressed location has its effective priority reduced by the current IOR weight. Entries that decay below a fixed cleanup threshold (0.05, no longer YAML-configurable) are removed, and at most 20 suppressed locations (also fixed) are tracked at once.

Head movement. The selected target image coordinates are converted to head yaw and pitch angles using camera intrinsics received on `/camera/color/camera_info` (or the Pepper-camera equivalent) and the current joint state. Commands below `min_angular_change_deg` (default 2°) are suppressed to prevent jitter. Remaining commands are smoothed with an exponential filter (α = `target_smoothing_alpha`, default 0.4) and clamped to joint limits before being published as `naoqi_bridge_msgs/JointAnglesWithSpeed` to `/joint_angles`: fixed limits when tracking a face ($\pm$ its own yaw/pitch bounds), or the YAML-configurable `saliency_yaw_lim`/`saliency_pitch_up`/`saliency_pitch_dn` when tracking a saliency peak.

Visualization Node

`VisualizationNode` (`overt_attention_visualization.cpp`) subscribes to the camera image, camera info, face detections, saliency peaks, and the current attention target, and produces an annotated overlay image showing tracked faces (always with mutual-gaze engagement status; this is no longer a togglable option), saliency peaks, and the current head target. The overlay is published as `sensor_msgs/Image` to `/overt_attention/visualization`.

Text-to-Speech

Node Architecture

The package follows the two-file split and remains a Python `rclpy` package (unlike the other three modules in this work package, which were subsequently rewritten in C++). `text_to_speech_application.py` is the ROS2 entry point and `text_to_speech_implementation.py` contains `TextToSpeechNode`, an `rclpy.lifecycle.LifecycleNode`.

The constructor declares all parameters (so they survive repeated configure/cleanup cycles) but otherwise does no setup. On `configure`, the node reads the parameter values, instantiates the selected synthesis backend, and creates a subscriber for `/text_to_speech/input`, a publisher for `/text_to_speech/speaking`, a client for the `/speech_event/set_enabled` service (microphone muting), and registers a `dec_interfaces/action/TTS` action server on `/text_to_speech`.

Synthesis Pipeline

When a text string is received (either via topic or action goal), the following pipeline is executed:

1. The text string is passed to the active synthesis backend. For the *naoqi_ros* backend, the text is forwarded directly to Pepper's on-board ALTextToSpeech via the `/speech` topic; playback duration is estimated from the character count and `chars_per_second`. For the *kokoro* and *elevenlabs* backends, the text is sent to the respective synthesis engine and the resulting audio is returned as a PCM buffer.

2. For **_pepper* backends, the PCM buffer is delivered to Pepper's speakers. In *file* mode, the buffer is saved as a WAV file, transferred to the robot via SCP, and played via ALAudioPlayer. In *stream* mode, raw PCM chunks are sent via the NAOqi AudioDevice interface. The `stream_volume` parameter scales the PCM amplitude before streaming.

3. For the duration of playback, the node disables the speech-recognition microphone via the `/speech_event/set_enabled` service and re-enables it once playback completes, so that the robot does not transcribe its own speech.

4. The `/text_to_speech/speaking` topic is published as `true` during playback and `false` once playback completes or is interrupted.

5. If a TTS action goal is active, the action result is returned once playback finishes, indicating whether the utterance completed normally or was interrupted by a stop request.

6 Implementation

Animate Behavior

File Organization

Here is the file structure of the animate behavior package:

```
animate_behavior
├── config
│   └── animate_behavior_configuration.yaml
├── data
│   └── pepper_topics.yaml
├── include/animate_behavior
│   └── animate_behavior_interface.h
├── launch
│   └── animate_behavior.launch.py
├── src
│   ├── animate_behavior_application.cpp
│   └── animate_behavior_implementation.cpp
├── CMakeLists.txt
├── package.xml
└── README.md
```

Figure 1: File structure of the animate behavior package.

Configuration File

The operation of the animate behavior node is controlled by `animate_behavior_configuration.yaml`. Parameters are declared under the `animate_behavior` node namespace. The available key-value pairs are listed below.

Table 2: Configuration parameters for the animate behavior node.

Key	Value	Description
`verbose_mode`	`true` or `false`	Enables diagnostic logging output.
`led_enabled`	`true` or `false`	Enables the LED wave animation on Pepper's eye LEDs.
`led_white_step`	`0.06`	Delay (s) between successive LED layers during the brightening wave.
`led_dark_step`	`0.04`	Delay (s) between successive LED layers during the darkening wave.
`led_fade_duration`	`0.10`	Fade duration (s) applied to each LED layer transition.
`led_white_hold`	`2.0`	Duration (s) to hold all-white before beginning the darkening wave.
`led_dark_pause`	`0.2`	Pause (s) while fully dark before starting the next brightening wave.
`gesture_update_rate`	`30.0`	Animation timer frequency (Hz) driving the animation loop.
`gesture_smoothing_factor`	`0.15`	Exponential smoothing coefficient applied each animation tick when advancing joints toward their target.
`gesture_motion_speed`	`0.08`	ALMotion speed parameter used for animated joint moves.
`gesture_interval_min`	`2.5`	Minimum time (s) between successive gesture selections.
`gesture_interval_max`	`4.5`	Maximum time (s) between successive gesture selections.

Key	Value	Description
gesture_rotation_interval	5.0	Time (s) between base rotation commands.

Topics Subscribed

Table 3: Topics subscribed by the animate behavior node.

Topic Name	Message Type	Description
/joint_states	sensor_msgs/JointState	Current joint positions used as the starting point for exponential smoothing.

Topics Published, Service, and Action Interfaces

Table 4: Topics published, service, and action interfaces of the animate behavior node.

Interface	Name	Description
Publisher	/joint_angles (naoqi_bridge_msgs/JointAnglesWithSpeed)	Smoothed target joint-angle commands published at gesture_update_rate Hz.
Publisher	/cmd_vel (geometry_msgs/Twist)	Base rotation velocity command published at gesture_rotation_interval intervals.
Service	/animate_behavior/stop (std_srvs/Trigger)	Immediately stops the current animation and zeroes velocity and LEDs.
Action Server	/animate_behavior (dec_interfaces/action/AnimateBehavior)	Runs a gesture/rotation/LED animation for a requested behaviour type, range, and duration, reporting elapsed-time feedback.
Action Client	/naoqi_driver/run_led (naoqi_bridge_msgs/action/RunLed)	Drives the cascading face-LED animation.

Gesture Execution

File Organization

Here is the file structure of the gesture execution package:

```
gesture_execution
├── config
│   └── gesture_execution_configuration.yaml
├── data
│   ├── gesture.yaml
│   └── pepper_topics.yaml
├── include/gesture_execution
│   ├── gesture_execution_interface.h
│   └── gesture_pepper_kinematics.h
├── launch
│   └── gesture_execution.launch.py
├── src
│   ├── gesture_execution_application.cpp
│   ├── gesture_execution_implementation.cpp
│   ├── gesture_pepper_kinematics.cpp
│   └── gesture_test_visualization.cpp
├── test
│   └── test_pepper_kinematics.cpp
├── CMakeLists.txt
├── package.xml
└── README.md
```

Figure 2: File structure of the gesture execution package.

Configuration File

The operation of the gesture execution node is controlled by the YAML configuration file `gesture_execution_configuration.yaml`, which currently contains a single key, `verboseMode`. The gesture-definition and topic-mapping file paths are fixed in the code at `data/gesture.yaml` and `data/pepper_topics.yaml`.

Table 5: Configuration parameters for the gesture execution node.

Key	Value	Description
`verboseMode`	`true` or `false`	Enables detailed diagnostic logging during gesture execution.

Gesture Definition File

Named gesture primitives are defined in `data/gesture.yaml`. Each entry specifies joint names, a sequence of joint angle keyframes (in radians), and corresponding timestamps (in seconds). The currently defined gestures are:

Table 6: Named gesture primitives defined in `gesture.yaml`.

Gesture	Description
`welcome`	Raises both arms in a welcoming pose. Controls left arm, right arm, and leg joints across 5 s with a 3-keyframe trajectory.
`wave`	Animates the right arm in a waving motion. Cycles the elbow-roll joint between approximately 0.52 and 1.31 rad across 9 keyframes over 8.2 s.
`shake`	Extends the right arm forward and oscillates it in a handshake motion across 4 keyframes over 11 s.

Topics Subscribed

Table 7: Topics subscribed by the gesture execution node.

Topic Name	Message Type	Description
`/joint_states`	`sensor_msgs/JointState`	Current joint positions used for safety validation and IK initialisation.
`/localization`	`nav_msgs/Odometry`	Fused `map→base_footprint` robot pose used by the inverse kinematics utilities for deictic pointing.

Topics Published and Action Server

Table 8: Topics published and action server of the gesture execution node.

Interface	Name	Description
Publisher	`/joint_angles_trajectory`	`naoqi_bridge_msgs/JointAnglesTrajectory`: joint-angle trajectory waypoints published during gesture execution.
Publisher	`/gesture_execution/visualization`	`visualization_msgs/Marker`: RViz2 marker showing the target point, shoulder position, and pointing vector for deictic gestures.
Action Server	`/gesture_execution`	`dec_interfaces/action/Gesture`: accepts a gesture name goal, provides elapsed-time feedback, and returns a success/failure result.

Overt Attention

File Organization

Here is the file structure of the overt attention package:

```
overt_attention
├── config
│   └── overt_attention_configuration.yaml
├── data
│   └── pepper_topics.yaml
├── include/overt_attention
│   └── overt_attention_interface.h
├── launch
│   ├── attention_system.launch.py
│   └── realsense_camera.launch.py
├── src
│   ├── overt_attention_application.cpp
│   ├── overt_attention_implementation.cpp
│   ├── overt_attention_saliency.cpp
│   ├── overt_attention_utilities.cpp
│   └── overt_attention_visualization.cpp
├── CMakeLists.txt
├── package.xml
└── README.md
```

Figure 3: File structure of the overt attention package.

Configuration File

All three nodes share a single configuration file `overt_attention_configuration.yaml`. A shared `/**` block applies to all nodes; per-node parameters are grouped under the respective node name.

Shared Parameters These parameters live in the `/**` block and apply to all three nodes.

Table 9: Shared configuration parameters for all overt attention nodes.

Key	Value	Description
`use_compressed`	`false`	Whether to subscribe to compressed image topics.
`camera_type`	`pepper`	Selects the camera source (`pepper` or `realsense`); determines which image topics are used.

Saliency Node Parameters These parameters are declared under the `saliency_node` namespace.

Table 10: Configuration parameters for the saliency node. Downsample resolution, overlay transparency, and minimum peak spacing were removed as tunable parameters and are now fixed implementation details (160×120, 0.4, and 50 px respectively).

Key	Value	Description
publish_map	true	Publishes the annotated saliency map image.
use_depth_weighting	true	Weights saliency peaks by depth, deprioritising very near or far objects.
depth_min_m	0.3	Minimum depth (m) below which objects receive minimum weighting.
depth_max_m	10.0	Maximum depth (m) above which objects receive minimum weighting.
depth_weight_min	0.2	Minimum weight applied to out-of-range depth objects.
min_peak	0.5	Minimum saliency score for a peak to be reported.
num_peaks	5	Maximum number of salient peak locations to publish per frame.
process_hz	1.0	Rate (Hz) at which the saliency map is recomputed.

Attention Controller Node Parameters These parameters are declared under the overt_attention namespace in the shared configuration file.

Table 11: Key configuration parameters for the attention controller node. Fixed face-tracking joint limits, the same-face angular threshold, and IOR cleanup bookkeeping are no longer tunable: only the three saliency-tracking joint limits remain YAML-configurable (the face-tracking equivalents had never actually been changed from their code defaults), and the same-face threshold parameter was removed entirely (face identity is matched by track ID, not angular distance, so the parameter was never read).

Key	Value	Description
start_enabled	true	Whether to begin processing on startup without waiting for a service call.
move_to_default_on_disable	true	Whether the head returns to the default pose when attention is disabled.
default_yaw	0.0	Head yaw (rad) to return to when attention is disabled.
default_pitch	−0.2	Head pitch (rad) to return to when attention is disabled.
default_move_speed	0.1	Speed used when moving to the default pose.
saliency_yaw_lim	1.2	Maximum yaw (rad) allowed when tracking a saliency peak.
saliency_pitch_up	0.3	Maximum upward pitch (rad) when tracking a saliency peak.
saliency_pitch_dn	−0.3	Maximum downward pitch (rad) when tracking a saliency peak.
face_timeout	2.0	Time (s) without a face detection before falling back to saliency.
engaged_priority_bonus	2.0	Priority multiplier for faces detected in mutual gaze.
face_switch_cooldown	1.0	Minimum time (s) before switching attention to a different face.
prefer_closer_faces	true	Whether nearer faces are prioritised over farther ones.
max_face_distance	5.0	Maximum face distance (m) considered for tracking.
min_angular_change_deg	2.0	Minimum head movement (degrees) below which a command is suppressed.
target_smoothing_alpha	0.4	Exponential smoothing coefficient for head angle commands.
saliency_min_score	0.30	Minimum saliency score for a peak to be considered as an attention candidate.
saliency_min_cooldown	1.5	Minimum time (s) before switching attention to a different saliency target.
saliency_max_dwell	3.0	Maximum time (s) attention may dwell on a single saliency target.
switch_score_ratio	1.4	Minimum score ratio required to switch to a new saliency target.
same_target_threshold_deg	5.0	Angular threshold (degrees) below which two saliency peaks are treated as the same target.
enable_ior	true	Enables Inhibition of Return.
ior_max_suppression	0.9	Maximum priority suppression weight applied immediately after attending a location.
ior_half_life	3.0	Time (s) for IOR suppression weight to decay to half its initial value.
ior_radius_deg	15.0	Angular radius (degrees) within which IOR suppression is applied.

Visualization Node Parameters The `attention_visualization` YAML block previously declared `publish_overlay`, `publish_markers`, and `show_metrics`, none of which the node actually read; it has since been corrected to match the node's real parameters, shown below. A fourth parameter the node used to read, `show_engagement`, was removed rather than fixed: the engagement glow/label was found to always draw unconditionally regardless of its value, so it was a no-op knob.

Table 12: Configuration parameters for the visualization node.

Key	Default	Description
`camera_type`	`pepper`	Selects the camera source for the visualization image stream (also settable via the shared block).
`show_face_ids`	`true`	Overlays each tracked face's ID on the debug image.
`show_depth`	`true`	Overlays each tracked face's depth estimate.

Topics Subscribed

Table 13: Topics subscribed by the overt attention package nodes.

Node	Topic Name	Message Type	Description
`saliency_node`	`/camera/color/image_raw` (or compressed)	`sensor_msgs/Image` / `CompressedImage`	RGB camera stream for saliency computation; topic resolved from `pepper_topics.yaml` per `camera_type`.
`saliency_node`	`/camera/aligned_depth_to_color/image_raw` (or compressed)	`sensor_msgs/Image` / `CompressedImage`	Depth stream used for depth-weighted saliency when `use_depth_weighting` is enabled.
`overt_attention`	`/face_detection/data`	`dec_interfaces/msg/FaceDetection`	Face detections from D4.2.2, including mutual gaze flags.
`overt_attention`	`/joint_states`	`sensor_msgs/JointState`	Current head joint positions for angle computation.
`overt_attention`	`/overt_attention/saliency_peak`	`std_msgs/Float32MultiArray`	Salient peak locations from the saliency node.
`overt_attention`	`/camera/color/camera_info`	`sensor_msgs/CameraInfo`	Camera intrinsics used to convert pixel coordinates to head yaw/pitch angles.
`attention_visualization`	`/camera/color/image_raw` (or compressed)	`sensor_msgs/Image` / `CompressedImage`	Camera stream for the debug overlay image.
`attention_visualization`	`/camera/color/camera_info`	`sensor_msgs/CameraInfo`	Camera intrinsics, used for consistency with the attention controller node's angle computation.
`attention_visualization`	`/face_detection/data`	`dec_interfaces/msg/FaceDetection`	Face detections overlaid on the debug image.
`attention_visualization`	`/overt_attention/saliency_peak`	`std_msgs/Float32MultiArray`	Saliency peaks overlaid on the debug image.
`attention_visualization`	`/overt_attention/target_angles`	`geometry_msgs/Vector3`	Current attention target overlaid on the debug image.

Topics Published and Service

Table 14: Topics published and service interface of the overt attention package.

Node	Interface	Name / Type	Description
`overt_attention`	Publisher	`/joint_angles` (`naoqi_bridge_msgs/JointAnglesWithSpeed`)	Smoothed head joint angle commands.
`overt_attention`	Publisher	`/overt_attention/target_angles` (`geometry_msgs/Vector3`)	Current attention target as yaw/pitch/confidence.
`saliency_node`	Publisher	`/overt_attention/saliency_peak` (`std_msgs/Float32MultiArray`)	Top-N salient peak locations.
`saliency_node`	Publisher	`/overt_attention/saliency_map/compressed` (`sensor_msgs/CompressedImage`)	Annotated saliency map image, published when `publish_map` is true.
`attention_visualization`	Publisher	`/overt_attention/visualization` (`sensor_msgs/Image`)	Annotated camera image with attention overlay.
`overt_attention`	Service	`/overt_attention/set_enabled` (`std_srvs/SetBool`)	Enable or disable attention processing at runtime.

Text-to-Speech

File Organization

Here is the file structure of the text-to-speech package:

```
text_to_speech
├── text_to_speech
│   ├── __init__.py
│   ├── text_to_speech_application.py
│   └── text_to_speech_implementation.py
├── config
│   └── text_to_speech_configuration.yaml
├── data
│   └── pepper_topics.yaml
├── launch
│   └── text_to_speech.launch.py
├── scripts
│   └── text_to_speech
├── resource
│   └── text_to_speech
├── test
│   ├── test_flake8.py
│   └── test_pep257.py
├── manual_tests
│   └── test_play_audio.py
├── package.xml
├── requirements.txt
├── setup.cfg
├── setup.py
└── README.md
```

Figure 4: File structure of the text-to-speech package.

Configuration File

The operation of the text-to-speech node is controlled by the YAML configuration file `text_to_speech_configuration.yaml`. The available key-value pairs are listed below.

Table 15: Configuration parameters for the text-to-speech node.

Key	Value	Description
`engine`	`<string>`	Synthesis backend: `naoqi_ros`, `kokoro_local`, `kokoro_pepper`, `elevenlabs_local`, or `elevenlabs_pepper`.
`playback_method`	`stream` or `file`	Delivery method for `*_pepper` backends. `stream` sends raw PCM via NAOqi AudioDevice; `file` transfers WAV via SCP.
`naoqi_speech_topic`	`/speech`	Topic used by the `naoqi_ros` backend to forward text to Pepper's ALTextToSpeech.

Key	Value	Description
chars_per_second	12.0	Estimated speaking rate (characters/s) for duration estimation in naoqi_ros mode.
speech_padding_s	0.5	Extra time (s) added to the estimated duration before declaring playback complete in naoqi_ros mode.
voice	af_bella	Kokoro voice identifier used when a kokoro backend is selected.
sample_rate	24000	Output sample rate (Hz) for Kokoro synthesis.
output_device	-1	Output audio device index for the kokoro_local/elevenlabs_local backends (-1 selects the system default).
elevenlabs_api_key	<string>	ElevenLabs API key; required when an elevenlabs backend is selected.
elevenlabs_voice_id	<string>	ElevenLabs voice ID.
elevenlabs_model	eleven_turbo_v2_5	ElevenLabs synthesis model identifier.
elevenlabs_stability	0.83	Voice stability (0.0–1.0); higher values reduce expressiveness variation.
elevenlabs_similarity_boost	0.75	Voice similarity boost (0.0–1.0); higher values keep closer to the reference voice.
elevenlabs_style	0.3	Style exaggeration (0.0–1.0).
elevenlabs_speed	1.05	Speaking speed multiplier (0.7–1.2).
stream_volume	2.0	PCM amplitude multiplier applied before streaming to Pepper's speakers.

Topics Subscribed

Table 16: Topics subscribed by the text-to-speech node.

Topic Name	Message Type	Description
/text_to_speech/input	std_msgs/String	Text string to synthesize.

Topics Published and Action Server

Table 17: Topics published, action server, and service/action clients of the text-to-speech node.

Interface	Name	Description
Publisher	/text_to_speech/speaking	std_msgs/Bool: true during active playback; false when idle.
Publisher	/speech	std_msgs/String: text forwarded to Pepper's onboard NAOqi TTS (naoqi_ros backend only).
Action Server	/text_to_speech	dec_interfaces/action/TTS: accepts a text string goal; blocks until playback completes; returns a success or failure result.
Service Client	/speech_event/set_enabled	std_srvs/SetBool: mutes/unmutes the speech-recognition microphone during playback.
Service Client	/naoqi_driver/load_audio_file /naoqi_driver/unload_audio_file	naoqi_bridge_msgs/srv/LoadAudioFile, UnloadAudioFile: upload and free synthesised WAV audio on the robot (*_pepper backends, file playback method).
Service Client	/naoqi_driver/send_audio_buffer	naoqi_bridge_msgs/srv/SendAudioBuffer: streams raw PCM audio buffers to the robot (stream playback method).
Action Client	/naoqi_driver/play_audio	naoqi_bridge_msgs/action/PlayAudio: plays an uploaded audio file on Pepper's speakers.

References

Aldebaran Robotics (2024). NAOqi developer documentation. http://doc.aldebaran.com/2-5/naoqi/index.html. Accessed: August 2026.

ElevenLabs (2024). ElevenLabs text to speech. https://elevenlabs.io. Accessed: August 2026.

Hexgrad (2025). Kokoro-82M: An open-weight text-to-speech model. https://huggingface.co/hexgrad/Kokoro-82M. Accessed: August 2026.

Itti, L. and Koch, C. (2001). Computational modelling of visual attention. *Nature Reviews Neuroscience*, 2(3):194–203.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66).

Zhang, J. and Sclaroff, S. (2013). Saliency detection: A Boolean map approach. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 153–160.

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.
01 May 2026.

Version 1.1

Updates to the topic, service, and action names.
Updates to the formatting of the tables.
Yohannes Haile.
09 July 2026.

Version 1.2

Cross-checked the whole deliverable against `pepper4dec`: Animate Behavior, Gesture Execution, and Overt Attention were rewritten from their initial Python port into C++ `rclcpp_lifecycle::LifecycleNode`s after this deliverable was first written, so Module Design, Implementation, and every affected file, topic, and config reference were updated (Text-to-Speech remains Python).
Corrected swapped and dead topics: Animate Behavior publishes `/joint_angles` and Gesture Execution `/joint_angles_trajectory` (previously reversed), and Gesture Execution subscribes to `/localization` (`nav_msgs/Odometry`) and publishes an RViz `Marker`, not the dead `/robotLocalization/pose` and `MarkerArray`.
Completed the Animate Behavior and Overt Attention interfaces: added the missing services, action servers and clients, the saliency-map publisher, the depth, camera-info, and visualization subscriptions, and the shared `camera_type` parameter.
Completed the configuration tables: added Animate Behavior's two missing keys, the attention controller's 15 undocumented parameters, the Saliency Node's `overlay_alpha`, and Text-to-Speech's `output_device` key.
Completed the Text-to-Speech interfaces and file structure: added its service and action clients, corrected the launch file to `text_to_speech.launch.py`, and added the `scripts/text_to_speech` entry point.
Corrected the Gesture Definition File keyframe counts (`welcome` has 3, `shake` has 4).
Yohannes Haile.
07 August 2026.

Version 1.3

Synced the Overt Attention section with the codebase's parameter cleanup (35 to 29 tunable parameters, landed after v1.2 above was written): removed the saliency node's downsample resolution, overlay transparency, and peak spacing; the attention controller's face-tracking joint limits, `same_face_threshold_deg` (removed entirely, never read), and IOR cleanup/max-locations pair; all now fixed in code rather than YAML-configurable. Corrected the visualization node's parameter table, which v1.2 had already flagged as stale (`publish_overlay`/`publish_markers`/`show_metrics`, never read) to the current, now-accurate `camera_type`/`show_face_ids`/`show_depth` (`show_engagement` was removed, not fixed: the engagement overlay was found to always draw regardless of its value).
Yohannes Haile.
08 August 2026.

Version 1.4

Corrected Text-to-Speech's file structure: `tests/test_play_audio.py` was renamed to `manual_tests/test_play_audio.py` (it is a hardware-dependent manual script, not part of the automated lint suite), and the package's `test/` directory (`test_flake8.py`, `test_pep257.py`) was added to the file tree.
Yohannes Haile.
13 August 2026.

Version 1.5

Corrected the attention controller's runtime node name from `unified_attention_node` to `overt_attention` throughout (Module Specification, Module Design, and both interface tables), and standardized its descriptive label to "Attention Controller Node".
Reworked every table to break across page boundaries with a repeated header (`xltabular`) instead of being pushed whole to the next page, moved the captions to the top, and fixed the resulting table numbering.
Yohannes Haile.
14 August 2026.

Version 1.6

Re-checked the configuration tables against `pepper4dec`: removed the `gestureDescriptors` and `robotTopics` keys from the Gesture Execution configuration table and prose, as both were dropped from `gesture_execution_configuration.yaml` and the node's code. The node now reads only `verboseMode`, and the gesture-definition and topic-mapping file paths are fixed in the code.
Added references to the CSSR4Africa ROS1 predecessor deliverables for each module: Animate Behaviour Subsystem (D5.2), Attention Subsystem (D5.3), Gesture Execution (D5.5.1.1), and Integrated Text to Speech Conversion (D5.5.2.4).
Populated the References section with citations for the third-party technologies named in the document: ROS2, NAOqi, Kokoro-82M, ElevenLabs, Boolean Map Saliency, and Inhibition of Return.
Yohannes Haile.
15 August 2026.