UPANZI
DIGITAL PUBLIC INFRASTRUCTURE
NETWORK

Pepper Robot Tour for Digital Experience Center

D3.3 Software Installation Manual (ROS2)

Due date: **07/08/2026**
Submission Date: **07/08/2026**
Revision Date: **04/09/2026**

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 D3.3 is the Software Installation Manual (ROS2) for `pepper4dec`, the Pepper Robot Tour software stack developed for the Upanzi Digital Experience Center (DEC), a spin-off of the Culturally Sensitive Social Robotics for Africa (CSSR4Africa) project. This living document describes the process for installing, configuring, and running the ROS2 packages that make up the system on a physical Pepper robot, either natively on Ubuntu 22.04 or inside the project's Docker image. It covers hardware and software prerequisites, workspace setup, per-node Python virtual environments, model file placement, package configuration, bringing up the robot over NAOqi, and running the individual components as well as the full tour system. It will be updated throughout the project to reflect the current capabilities of the system.

Contents

1 Introduction

This manual provides step-by-step instructions for installing, configuring, and running the DEC Pepper Robot Tour software, distributed as the `pepper4dec` repository. The system is built on **ROS2 Humble Hawksbill** running on **Ubuntu 22.04**, and controls a physical Pepper robot equipped with an Intel RealSense depth camera and, for autonomous navigation, a Unitree L2 lidar.

Pepper role-plays as a digital guide at the DEC, coordinating speech, gestures, dialogue, perception, and navigation to walk visitors through the fictional Upanzi Republic's Digital Public Infrastructure (DPI) booths – from biometric enrollment (MOSIP) to financial transactions (MIFOS) and subsidy validation (UPMS). The software stack is organized as a set of ROS2 lifecycle nodes coordinated by a central behavior controller, following the naming and packaging conventions described in Deliverable D3.2 Software Engineering Standards Manual.

This document is intended for researchers and engineers who need to deploy, test, or extend the DEC system. It covers:

- Hardware and software prerequisites.
- Two installation methods: a containerized Docker build, and a manual native install.
- Setting up the per-node Python virtual environments required by the Python packages.
- Downloading and placing the perception model files.
- An overview of every package in the system, and how each is configured.
- Bringing the Pepper robot up over NAOqi, and running the software – individual nodes, the full tour system, and the SLAM/navigation stack.
- Troubleshooting common installation and runtime issues.

This is a living document. It will be updated at each project milestone to reflect new packages, configuration changes, and updated installation procedures.

2 Prerequisites

2.1 Hardware Requirements

- Pepper robot (SoftBank Robotics / United Robotics Group), connected to the workstation over Wi-Fi on the same local network; see SoftBank Robotics (2024) for hardware specifications.
- Workstation running **Ubuntu 22.04 LTS** (x86_64).
- **Intel RealSense** depth camera (RGB + aligned depth + IMU), mounted on Pepper's sensor rig or on a stand for bench testing. Required by the perception nodes and by the RTAB-Map SLAM/localization profile.
- **Unitree L2 lidar**, required by `pepper_slam` and `pepper_navigation` for lidar-inertial odometry (FAST-LIO) and 2D mapping (SLAM Toolbox). Nothing in the navigation stack works without it.
- **NVIDIA GPU (CUDA)** – optional. Accelerates the ONNX-based perception nodes (`face_detection`, `age_gender_detection`, `person_detection`) and Whisper transcription in `speech_event`; all fall back to CPU automatically if no GPU is present. Note that navigation is deliberately kept CPU-light so that it also runs on an on-board Jetson-class computer.

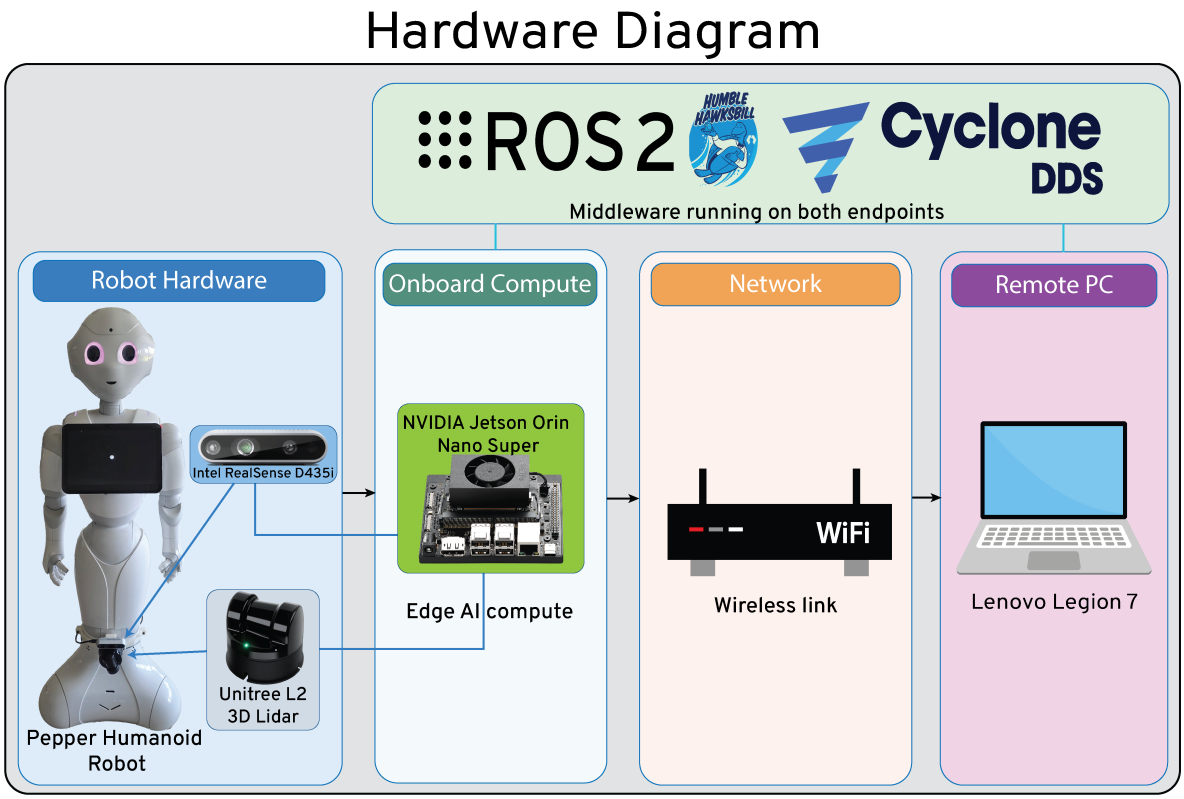

Figure 1: Pepper4DEC hardware setup: the Pepper robot, RealSense camera, and Unitree L2 lidar around the workstation running the software stack.

2.2 Software Prerequisites

- **ROS2 Humble Hawksbill** – Installation guide.
- **Python 3.10** (the interpreter ROS2 Humble targets on Ubuntu 22.04).
- **colcon** build tools and **rosdep**.
- **Git** for cloning the repository and its third-party dependencies.
- **Docker** and **Docker Compose** – only required for the containerized installation (Method 1 in Section 3).

3 Setting up the Development Environment

There are two ways to set up the `pepper4dec` development environment: a containerized install using Docker (Method 1), and a manual native install directly on Ubuntu 22.04 (Method 2).

Important: The Docker image (Method 1) bundles ROS2 Humble, every system dependency, all three per-node Python virtual environments, and a full `colcon build` of the workspace, so it is the fastest way to get a working system. Method 2 is intended for developers who need a native install – for example, to attach a debugger, or because Docker's device passthrough does not suit their setup.

3.1 Method 1: Docker Installation

The repository ships a multi-stage `Dockerfile` and a `docker-compose.yml` that builds the ROS2 workspace and all three Python virtual environments in parallel, then assembles them into a single runtime image.

Important: GPU passthrough into the container (used by the ONNX perception nodes and Whisper transcription) requires the **NVIDIA Container Toolkit** to be installed and configured on the host *before* the first `docker compose build/up` – it is not installed by anything in this repository. Install it via the official `nvidia.github.io/libnvidia-container` apt repository, then run `sudo nvidia-ctk runtime configure runtime=docker` and restart Docker (`sudo systemctl restart docker`). Without this step, `docker-compose.yml`'s GPU device reservation fails at container start. Skip this if running CPU-only (perception nodes fall back to CPU automatically).

1. Clone the repository.

```
git clone https://github.com/yohatad/pepper4dec.git
cd pepper4dec
```

2. Copy the example environment file and edit it as needed (ROS domain ID, GPU selection, and the LLM API key used by `conversation_manager`).

```
cp .env.example .env
```

3. Build the image. This clones the NAOqi driver, the lidar-localization packages (`lio_localization`, `fast_lio`, `point_lio`, `l2lidar_node`, `fastlio_lc_pgo`), and BehaviorTree.CPP/BehaviorTree.ROS2; installs `naoqi_libqi`/`naoqi_libqicore` and the Pepper/NAO mesh packages from apt; builds all workspace packages with `colcon`; and builds the three per-node Python virtual environments from frozen lockfiles.

```
docker compose build
```

4. Run the default service, which launches the full tour system (`dec_system.launch.py`, Section 6.5).

```
docker compose up pepper4dec
```

5. Optional profiles are available for visualization and live development:

```
# Adds an rviz2-only container
docker compose --profile viz up

# Live-mounts the repository source over the image, for iterative
    development
docker compose --profile dev up
```

Note: The container uses host networking (required for ROS2 DDS multicast discovery) and runs in privileged mode to access the RealSense USB device, the Unitree L2 lidar, and the audio devices. No venv is activated inside the container shell by default – each Python node resolves its own interpreter from `docker/venv_map.sh` at launch time, described in Section 4.

If the workspace source changes, rebuild inside the running `pepper4dec-dev` container rather than rebuilding the image:

```
colcon build --symlink-install
source install/setup.bash
```

3.2 Method 2: Manual Installation

3.2.1 Installing ROS2 Humble

1. Update the system and install the base tools.

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl git wget build-essential cmake
```

2. Follow the official ROS2 Humble installation instructions for Ubuntu 22.04 (Debian package install), which registers the ROS2 apt repository, installs the signing key, and installs `ros-humble-desktop`.

```
sudo apt install ros-humble-desktop
```

3. Install colcon, rosdep, and the remaining Python build tools.

```
sudo apt install -y python3-colcon-common-extensions python3-vcstool
    \
python3-rosdep python3-pip python3-dev python3-venv python3-
    setuptools

sudo rosdep init
rosdep update
```

4. Source ROS2 automatically in every new shell.

```
echo "source /opt/ros/humble/setup.bash" >> $HOME/.bashrc
source $HOME/.bashrc
```

3.2.2 Installing System Dependencies

Install the Nav2, SLAM, and perception system packages, the NAOqi driver's C++ dependencies (including the pinned NAOqi/mesh libraries, installed from apt rather than built from source – see the note after the workspace clone step below), GTSAM (needed by the optional `fastlio_lc_pgo` loop-closure package), and the native libraries used by the perception and audio nodes.

```
sudo apt install -y \
  ros-humble-nav2-bringup ros-humble-nav2-map-server ros-humble-nav2-amcl
    \
  ros-humble-nav2-controller ros-humble-nav2-planner ros-humble-nav2-
   behaviors \
  ros-humble-nav2-bt-navigator ros-humble-nav2-lifecycle-manager \
  ros-humble-nav2-costmap-2d ros-humble-nav2-msgs \
  ros-humble-slam-toolbox ros-humble-rtabmap-ros ros-humble-realsense2-
   camera \
  ros-humble-message-filters ros-humble-cv-bridge ros-humble-image-
   transport \
  ros-humble-geometry-msgs ros-humble-sensor-msgs ros-humble-std-msgs \
  ros-humble-std-srvs ros-humble-robot-state-publisher \
  ros-humble-joint-state-publisher \
  ros-humble-naoqi-libqi ros-humble-naoqi-libqicore \
  ros-humble-nao-meshes ros-humble-pepper-meshes \
  ros-humble-gtsam net-tools \
  libyaml-cpp-dev libopencv-dev libomp-dev libeigen3-dev libboost-all-dev
    \
  libgl1-mesa-glx libglib2.0-0 libudev-dev libusb-1.0-0 \
  libportaudio2 portaudio19-dev libsndfile1 espeak-ng
```

3.2.3 Creating the Workspace and Cloning Repositories

Following the workspace layout defined in Deliverable D3.2 Software Engineering Standards Manual, the `pepper4dec` repository is cloned into the workspace's source tree as `dec_system`, alongside the third-party NAOqi, BehaviorTree, and lidar-localization packages it depends on. `naoqi_libqi`, `naoqi_libqicore`, and the Pepper/NAO mesh packages are *not* cloned here – they come from the pinned apt packages installed in Section 3.2.2, which avoids rebuilding these stable, unchanging packages from source on every build.

```
# Create the workspace
mkdir -p $HOME/dec_ws/src
cd $HOME/dec_ws/src

# Clone the DEC software stack
git clone https://github.com/yohatad/pepper4dec.git dec_system

# Clone the NAOqi ROS2 driver
git clone --depth 1 https://github.com/yohatad/naoqi_driver2.git
   naoqi_driver2

# Clone the fork of naoqi_bridge_msgs carrying the SetAudioVolume,
# StopAudio and UnloadAudioFile services that dec_system depends on
git clone --depth 1 https://github.com/yohatad/naoqi_bridge_msgs2.git
   naoqi_bridge_msgs
```

```
# Clone BehaviorTree.CPP and its ROS2 bindings, used by
   behavior_controller.
# BehaviorTree.CPP stays source-built rather than using the
# ros-humble-behaviortree-cpp apt package: that package has a packaging
   bug
# on Jammy where its library lands at a multiarch path
# (lib/x86_64-linux-gnu/libbehaviortree_cpp.so) that its own CMake export
# config does not look in, so anything depending on it (BehaviorTree.ROS2
   )
# fails to configure.
git clone --depth 1 https://github.com/BehaviorTree/BehaviorTree.CPP.git
git clone --depth 1 https://github.com/BehaviorTree/BehaviorTree.ROS2.git

# Clone the lidar-inertial odometry and localization packages used by
# pepper_slam/pepper_navigation for the Unitree L2. fast_lio and
   point_lio
# each carry git submodules (ikd-Tree, and for point_lio also IKFoM) that
    a
# plain clone will NOT fetch -- omitting --recurse-submodules produces a
# CMake "No SOURCES given to target" failure at build time.
git clone --depth 1 https://github.com/yohatad/lio_localization.git
git clone --depth 1 --recurse-submodules --shallow-submodules \
  https://github.com/yohatad/FAST_LIO_ROS2.git fast_lio
git clone --depth 1 --recurse-submodules --shallow-submodules \
  https://github.com/yohatad/point_lio_ros2.git point_lio
git clone --depth 1 https://github.com/yohatad/l2lidar_node.git

# Optional: loop-closure pose-graph-optimization backend for FAST-LIO
git clone --depth 1 https://github.com/yohatad/fastlio_lc_pgo.git
```

Note: `lio_localization`, `fast_lio`, `point_lio`, `l2lidar_node`, and `fastlio_lc_pgo` are only needed for the lidar-based `nav_profile` options (Section 6.6) – `fastlio_loc` (the default), `rtabmap_loc`, and `amcl` all require `fast_lio`; `fastlio_loc` additionally requires `lio_localization`. The `legacy` profile needs none of them. If you have no Unitree L2 lidar yet, skip this block and pass `nav_profile:=legacy` when launching (Section 6.5).

3.2.4 Resolving Dependencies and Building the Workspace

```
cd $HOME/dec_ws

# Install any remaining system dependencies declared by the cloned
   packages
sudo apt update
rosdep install --from-paths src --ignore-src -r -y --skip-keys "python3-
   pip"

# Build the workspace. colcon build fans out to one process per available
# core by default; on a memory-constrained machine this can exhaust RAM
   when
```

```
# several heavy C++ translation units compile concurrently -- cap it with
# --parallel-workers N and MAKEFLAGS="-jN" if the build gets OOM-killed.
export I_AGREE_TO_NAO_MESHES_LICENSE=1
export I_AGREE_TO_PEPPER_MESHES_LICENSE=1
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release

# Source the workspace, and add it to every new shell
source install/setup.bash
echo "source $HOME/dec_ws/install/setup.bash" >> $HOME/.bashrc
```

After installing, you will have the following workspace structure:

```
dec_ws
├── build
├── install
├── log
├── src
│   ├── dec_system
│   │   ├── animate_behavior
│   │   ├── behavior_controller
│   │   ├── conversation_manager
│   │   ├── dec_common
│   │   ├── dec_interfaces
│   │   ├── dec_launch
│   │   ├── face_detection
│   │   ├── gesture_execution
│   │   ├── overt_attention
│   │   ├── pepper_navigation
│   │   ├── pepper_slam
│   │   ├── person_detection
│   │   ├── speech_event
│   │   └── text_to_speech
│   ├── naoqi_driver2
│   ├── naoqi_bridge_msgs
│   ├── BehaviorTree.CPP
│   ├── BehaviorTree.ROS2
│   ├── lio_localization
│   ├── fast_lio
│   ├── point_lio
│   ├── l2lidar_node
│   └── fastlio_lc_pgo
└── .venvs
    ├── sound
    ├── conversation
    └── tts_virtual_env
```

Figure 2: Directory structure of the `dec_ws` workspace after installation.

4 Setting up the Python Virtual Environments

Three of the fourteen `dec_system` packages are pure Python (`conversation_manager`, `speech_event`, `text_to_speech`); the rest are C++ and need no Python environment at all. These three packages cannot share a single virtual environment: `speech_event` requires `numpy==2.2.6` while `conversation_manager` requires `numpy==1.24.4`, and each pulls in a different, mutually incompatible build of PyTorch. Each package is therefore given its own dedicated virtual environment, all created under `$HOME/dec_ws/.venvs/` so that each package's `scripts/` launcher (invoked transparently by `ros2 run`) can resolve the correct interpreter through `$HOME/dec_ws/.venvs/venv_map.sh`.

Package	Virtual environment	Notable dependencies
`speech_event`	`.venvs/sound`	`torch`, `faster-whisper`, `onnxruntime`, `pyroomacoustics`
`conversation_manager`	`.venvs/conversation`	`chromadb`, `sentence-transformers`, `openai`
`text_to_speech`	`.venvs/tts_virtual_env`	`kokoro`, `soundfile`, `sounddevice`

Table 1: Per-node Python virtual environments.

4.1 speech_event

The speech pipeline (voice-activity detection, Whisper transcription, and sound localization) needs its own PyTorch build pinned to a specific CUDA version.

```
cd $HOME/dec_ws
python3 -m venv .venvs/sound
source .venvs/sound/bin/activate

pip install --upgrade pip
pip install torch==2.5.1+cu121 torchaudio==2.5.1+cu121 \
  --index-url https://download.pytorch.org/whl/cu121
pip install -r src/dec_system/speech_event/requirements.txt

deactivate
```

4.2 conversation_manager

```
cd $HOME/dec_ws
python3 -m venv .venvs/conversation
source .venvs/conversation/bin/activate

pip install --upgrade pip
pip install -r src/dec_system/conversation_manager/requirements.txt

deactivate
```

Set the LLM API key and endpoint used by `conversation_manager` as environment variables (the default endpoint and model target DeepSeek):

```
export LLM_API_KEY=your_api_key_here
export LLM_BASE_URL=https://api.deepseek.com/v1
export LLM_MODEL=deepseek-chat
```

4.3 text_to_speech

```
cd $HOME/dec_ws
python3 -m venv .venvs/tts_virtual_env
source .venvs/tts_virtual_env/bin/activate

pip install --upgrade pip
pip install kokoro soundfile sounddevice scipy

# Only required for the elevenlabs_local / elevenlabs_pepper backends
pip install elevenlabs

deactivate

# Kokoro's phonemiser needs espeak-ng, installed at the system level
sudo apt-get install espeak-ng
```

Note: The `file` playback method of `text_to_speech` (used to play synthesized audio through Pepper's on-board `ALAudioPlayer`) SCPs the WAV file to the robot first, and therefore requires a passwordless SSH key already installed on Pepper. The default `stream` playback method has no such requirement.

5 Downloading Model Files

The perception nodes load ONNX models that are not committed to the repository and must be downloaded and placed manually before the corresponding node will start.

Package	Expected model file(s), under `<package>/models/`
`person_detection`	`person_detection_yolov11m.onnx`
`face_detection`	`face_detection_goldYOLO.onnx`, `face_detection_sixdrepnet360.onnx` (Ruiz et al., 2022), `face_detection_mivolo_agegender.onnx`
`speech_event`	`silero_vad.onnx` (already committed under `models/`; no download required)

Table 2: Perception model files.

```
# Place the person detection model
cp person_detection_yolov11m.onnx \
  $HOME/dec_ws/src/dec_system/person_detection/models/

# Place the face detection models
cp face_detection_goldYOLO.onnx face_detection_sixdrepnet360.onnx \
  face_detection_mivolo_agegender.onnx \
  $HOME/dec_ws/src/dec_system/face_detection/models/
```

Both `face_detection` and `person_detection` fall back to CPU inference automatically if no CUDA-capable GPU is available; the `whisper_model_id` used by `speech_event` is instead downloaded on first run directly through the `faster-whisper` package, and the Kokoro-82M weights used by `text_to_speech` are likewise pulled by the `kokoro` pip package at runtime.

6 Installing and Running the Pepper4DEC Software

6.1 Package Overview

Once built, the `src/dec_system` directory of the workspace contains the following packages, grouped by function.

Package	Language	Description
Core Control		
`behavior_controller`	C++	BehaviorTree.CPP mission interpreter that orchestrates the tour
`animate_behavior`	C++	Idle / social animated behavior action server
`conversation_manager`	Python	RAG (Lewis et al., 2020) + LLM dialogue manager
`gesture_execution`	C++	Deictic/iconic gesture action server (Bézier + IK)
`speech_event`	Python	VAD, Whisper ASR, and sound-source localization
`text_to_speech`	Python	Multi-backend speech synthesis and playback
Perception & Attention		
`face_detection`	C++	Face detection, head pose, mutual gaze, age/gender estimation
`person_detection`	C++	YOLOv11 (Ultralytics, 2024) person detection and ByteTrack (Zhang et al., 2022) (SORT-family (Wojke et al., 2017)) multi-person tracking
`overt_attention`	C++	Visual attention controller (saliency + face + audio)
Navigation & Localization		
`pepper_slam`	C++	RTAB-Map / SLAM Toolbox / FAST-LIO mapping and localization
`pepper_navigation`	C++	Nav2 stack (planning, obstacle avoidance, keepout zones)
Infrastructure & Utilities		
`dec_launch`	C++	System launch files and startup configurations
`dec_interfaces`	C++	Custom ROS2 message, service, and action definitions
`dec_common`	C++	Shared C++ utilities: camera lifecycle base class, ByteTrack tracker

Table 3: Pepper4DEC software packages.

The full workspace layout resulting from installation is shown in Figure 2 in Section 3.2.4.

6.2 Configuration

Each C++ package stores its ROS2 parameters in a YAML file under its own `config/` directory, loaded through the package's launch file. Parameters can be inspected or overridden at runtime with `ros2 param get/set`.

6.2.1 Example: Person Detection Configuration

`person_detection/config/person_detection_configuration.yaml` controls the camera source, detection confidence, and ByteTrack tracking parameters:

```
person_detection:
  ros__parameters:
    camera: "pepper"
    use_compressed: false
    image_timeout: 2.0
    verbose_mode: false
    confidence_threshold: 0.6
    target_classes:
      - person
    track_threshold: 0.45
    track_buffer: 30
    match_threshold: 0.8
    frame_rate: 30
```

6.2.2 Example: Face Detection Configuration

`face_detection/config/face_detection_configuration.yaml` gates the SixDRepNet (Ruiz et al., 2022) head-pose and mutual-gaze thresholds, and whether the node requires `person_detection` to be running:

```
face_detection:
  ros__parameters:
    use_compressed: false
    camera: "pepper"
    verbose_mode: false
    image_timeout: 2.0
    sixdrepnet_confidence: 0.90
    sixdrepnet_headpose_angle: 10.0
    require_person_detection: true
    person_detection_timeout: 0.5
    prioritize_face_depth: true
```

6.2.3 Example: Behavior Controller Configuration

`behavior_controller/config/behavior_controller_configuration.yaml` selects which BehaviorTree XML mission file to run, and points to the culture and environment knowledge bases it consults while running it. Like every other package's configuration file, it is loaded as a standard ROS2 parameters file through the launch file's `parameters=[...]`, so it must be wrapped under the node's name and `ros__parameters`:

```
behavior_controller:
  ros__parameters:
    scenario_specification: "asr_cm_tts_pipeline"
    culture_knowledge_base: "cultureKnowledgeBase.yaml"
    environment_knowledge_base: "decEnvironmentKnowledgeBase_short.yaml"
    verbose_mode: true
```

The mission XML files themselves live in `behavior_controller/data/` (for example `asr_cm_tts_pipeline.xml` and `dec_Tour.xml`); see `behavior_controller/data/XML_USE.md` for the tree-authoring conventions.

6.2.4 Example: Text-to-Speech Configuration

`text_to_speech/config/text_to_speech_configuration.yaml` selects the synthesis backend and playback method described in Section 4.3:

```
text_to_speech:
  ros__parameters:
    engine: "naoqi_ros"
    playback_method: "stream"
    naoqi_speech_topic: "/speech"
    voice: "af_bella"
    sample_rate: 24000
```

6.3 Bringing up the Pepper Robot

Communication with the physical robot goes through the NAOqi ROS2 driver, `naoqi_driver2`, cloned in Section 3.2.3. It requires the robot's IP address, which Pepper announces by voice on boot (or on a single press of the chest button), and, for some nodes, the network interface name of the workstation, found with `ifconfig` or `ip addr`.

1. Ensure the workstation and the Pepper robot are on the same Wi-Fi network.
2. Open a new terminal and bring up the NAOqi driver.

```
ros2 launch naoqi_driver naoqi_driver.launch.py nao_ip:=<robot_ip>
```

For manual bring-up during development, individual perception nodes can also connect directly by `qi` URL, for example:

```
--qi-url=tcp://<robot_ip>:9559 --roscore_ip=127.0.0.1 \
--network_interface=<network_interface_name>
```

6.4 Running Individual Nodes

Each package can be launched independently for development and testing:

```
# Perception
ros2 launch person_detection person_detection.launch.py
ros2 launch face_detection face_detection.launch.py
ros2 launch overt_attention attention_system.launch.py

# Speech and dialogue
ros2 launch speech_event speech_event.launch.py
ros2 launch text_to_speech text_to_speech.launch.py
ros2 launch conversation_manager conversation_manager.launch.py

# Gesture and animation
ros2 launch gesture_execution gesture_execution.launch.py
ros2 run animate_behavior animate_behavior

# Behavior orchestration
ros2 launch behavior_controller behavior_controller.launch.py
```

Note: Use `ros2 launch`, not a bare `ros2 run`, for any node whose `config/` YAML must be loaded as ROS2 parameters (`gesture_execution` and `behavior_controller` above) – the package's own launch file points `parameters=[...]` at that YAML. A bare `ros2 run` starts the node with no parameters loaded at all.

A convenience launch file also brings up the speech recognition → conversation manager → behavior controller pipeline together (the NAOqi driver from Section 6.3 must already be running):

```
ros2 launch dec_launch asr_cm_pipeline.launch.py nao_ip:=<robot_ip>
```

6.5 Running the Full Tour System

The complete system – perception, localization, animation, gesture, speech, dialogue, navigation, and behavior orchestration – is brought up with a single top-level launch file:

```
source $HOME/dec_ws/install/setup.bash
ros2 launch dec_launch dec_system.launch.py

# Pick a different Nav2 localization profile
ros2 launch dec_launch dec_system.launch.py nav_profile:=rtabmap_loc

# Perception + behavior only, without the Nav2 stack (lio_localization
   still
# comes up standalone in this case, so /localization/pose is still
   published)
ros2 launch dec_launch dec_system.launch.py enable_navigation:=false
```

This launch file starts, in order, the attention/perception subsystem, animation, gesture execution, speech event, text-to-speech, conversation manager, the navigation stack (disable with `enable_navigation:=false`), and the behavior controller; it finishes by starting a `nav2_lifecycle_manager` node that sequences every lifecycle node through `configure` → `activate` in dependency order. `nav_profile` selects which of `pepper_navigation`'s four Nav2 bringups to use:

nav_profile	**Nav2 bringup**	**Localization**
`fastlio_loc` (default)	`pepper_nav2_fastlio_loc.launch.py`	FAST-LIO + prior-map 3D ICP (`lio_localization`)
`rtabmap_loc`	`pepper_nav2_rtabmap_loc.launch.py`	RTAB-Map localization mode against a prior `.db`
`amcl`	`pepper_nav2_amcl.launch.py`	AMCL over a flattened `/scan`, on FAST-LIO odometry
`legacy`	`pepper_navigation.launch.py`	AMCL on raw wheel odometry; publishes no `/localization/pose`

Table 4: `dec_system.launch.py`'s `nav_profile` options.

Note: `dec_system.launch.py` does not itself start the NAOqi driver or the L2 lidar driver – bring these up separately first, as described in Section 6.3 and Section 6.6 respectively. Every profile except `legacy` requires `fast_lio` to already be built (Section 3.2.3); `fastlio_loc` additionally requires `lio_localization` and a pre-built prior map (Section 6.6).

6.6 Running SLAM and Navigation

`pepper_slam` provides three interchangeable mapping/localization backends, and `pepper_navigation` provides a matching Nav2 launch file for each:

```
# Start the Unitree L2 lidar driver first (external package)
ros2 launch l2lidar_node l2lidar.launch.py

# Build a new map with SLAM Toolbox (2D lidar mapping)
ros2 launch pepper_slam slam_toolbox.launch.py

# Or build a map with RTAB-Map (RGB-D mapping)
ros2 launch pepper_slam rtabmap_base.launch.py

# Run FAST-LIO or Point-LIO odometry on the Unitree L2 (needed by the
# fastlio_loc/rtabmap_loc/amcl nav profiles above)
ros2 launch pepper_slam fastlio_odometry.launch.py
ros2 launch pepper_slam pointlio_odometry.launch.py

# Localize against a prior map with FAST-LIO + ICP, and run Nav2
ros2 launch pepper_navigation pepper_nav2_fastlio_loc.launch.py

# Or localize with RTAB-Map, and run Nav2
ros2 launch pepper_navigation pepper_nav2_rtabmap_loc.launch.py
```

Pre-built occupancy grid maps (`.pgm`/`.yaml`) are kept under `pepper_navigation/map/` for reuse across sessions.

6.7 Running with Docker Compose

The Docker installation in Section 3.1 launches the full tour system (`dec_system.launch.py`, default `nav_profile`) by default via `docker compose up pepper4dec`. To pass different launch arguments or run a different launch file inside the container:

```
docker compose run --rm pepper4dec \
  /bin/bash -c "source /opt/pepper4dec_setup.sh && \
  ros2 launch dec_launch dec_system.launch.py nav_profile:=rtabmap_loc"
```

7 Troubleshooting

This section covers installation and startup problems found by building the system from scratch on a fresh machine, beyond the version-specific bugs already fixed upstream.

`PackageNotFoundError` / `get_package_share_directory()` failures at launch

Nearly always a missing sibling clone. Cross-check against the clone list in Section 3.2.3: `lio_localization` and `fast_lio` are required by every `nav_profile` except `legacy` (Section 6.5); `point_lio` and `fastlio_lc_pgo` are only needed if you explicitly launch `pepper_slam`'s Point-LIO or loop-closure launch files; `l2lidar_node` is needed to bring up the physical lidar at all (Section 6.6).

CMake error: `No SOURCES given to target`, while building `fast_lio` or `point_lio`

These two packages carry git submodules (`ikd-Tree`, and for `point_lio` also `IKFoM`) that a plain `git clone` does not fetch. Re-clone with `--recurse-submodules --shallow-submodules` as shown in Section 3.2.3, or run `git submodule update --init --recursive` inside an already-cloned copy.

`rosdep`/CMake cannot find GTSAM, building `fastlio_lc_pgo`

`fastlio_lc_pgo` links against GTSAM but does not declare it as a rosdep-resolvable dependency. Install it explicitly before building: `sudo apt install ros-humble-gtsam` (already included in the system dependencies list, Section 3.2.2).

Docker: GPU device reservation fails at container start

The NVIDIA Container Toolkit is not installed. See the note in Section 3.1.

`rtabmap` segfaults shortly after startup, no RealSense attached

Observed on an RGB-D-less bench setup with `nav_profile:=rtabmap_loc`. Plausibly `rtabmap` mishandling a total absence of RGB-D data rather than a `pepper4dec` bug; attach a RealSense camera and re-test before assuming otherwise.

Bare `ros2 run` starts a node with none of its configured parameters

Some nodes (`gesture_execution`, `behavior_controller`) load their `config/` YAML only when launched through their own launch file's `parameters=[...]`. Use `ros2 launch <package> <package>.launch.py`, not a bare `ros2 run`, unless you pass `--ros-args --params-file` yourself (see the usage comment at the top of each package's configuration YAML).

References

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474.

Ruiz, N., Chong, E., and Rehg, J. M. (2022). Sixdrepnet: 6d rotation representation for head pose estimation. In *European Conference on Computer Vision (ECCV)*.

SoftBank Robotics (2024). Pepper technical specifications. http://doc.aldebaran.com/2-5/family/pepper_technical/index_pep.html.

Ultralytics (2024). Ultralytics yolo11. https://github.com/ultralytics/ultralytics. Accessed: August 2026.

Wojke, N., Bewley, A., and Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. *arXiv preprint arXiv:1703.07402*.

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., and Wang, X. (2022). Bytetrack: Multi-object tracking by associating every detection box. In *European Conference on Computer Vision (ECCV)*.

Version History

Version 1.0

Initial release, replacing the earlier placeholder draft (which referenced a non-existent `upanzi-dec` repository and invented package names) with an installation manual verified against the actual `pepper4dec` source tree.
Documented both the Docker and manual native installation methods, the three per-node Python virtual environments and why they cannot be merged, model file placement, the full package overview, worked configuration examples for four representative packages, NAOqi bring-up, and running the system as individual nodes, the full tour, and the SLAM/navigation stack.
Adopted the `dec_ws`/`dec_system` workspace layout defined in Deliverable D3.2 Software Engineering Standards Manual for consistency across the two documents.
Yohannes Haile.
07 August 2026.

Version 1.1

Re-verified the entire manual against the current `pepper4dec` source tree (branch `cpp`) after a round of upstream fixes and a lidar/localization stack addition.
Corrected the Behavior Controller configuration example to the wrapped `ros__parameters` format it now actually requires, and added the five previously-undocumented sibling dependencies (`lio_localization`, `fast_lio`, `point_lio`, `fastlio_lc_pgo`, `l2lidar_node`) to the clone list and workspace tree, including the `--recurse-submodules` flag `fast_lio`/`point_lio` need for their bundled `ikd-Tree`/`IKFoM` submodules.
Switched Method 2's NAOqi/mesh packages from source clones to the pinned apt packages, matching the Docker image, and added the `gtsam`/`net-tools` system dependencies it also installs explicitly.
Added an NVIDIA Container Toolkit prerequisite note for Docker's GPU passthrough, corrected the Docker and `docker compose run` default-service description (now the full tour system, not RTAB-Map RealSense), added the `nav_profile` launch argument table, and fixed a dead SLAM launch-file reference.
Switched the `gesture_execution`/`behavior_controller` individual-node examples from a bare `ros2 run` to `ros2 launch`, since only the latter loads their configured parameters.
Added a Troubleshooting section covering the issues found during this re-verification.
Added the hardware diagram (Figure 1) after the Hardware Requirements list, and fixed two genuine margin-overflow spots (an unbroken `nav2_lifecycle_manager` identifier and a table cell) by switching them from to , which allows breaking at underscores.
Corrected the Package Overview table's person_detection citation, which cited the 2017 YOLO9000 paper for what is actually a YOLOv11-based module; now cites Ultralytics YOLO11, and cites the ByteTrack paper directly alongside the existing SORT-family citation.
Added the missing Retrieval-Augmented Generation citation to the same table's conversation_manager row.
Yohannes Haile.
31 August 2026.

Version 1.2

Renamed the deliverable to "Software Installation Manual (ROS2)" to disambiguate it from any future non-ROS2 installation documentation, and renamed the source/PDF files and containing

folder to match.
Yohannes Haile.
04 September 2026.