CSSR for

Culturally Sensitive Social Robotics for Africa

D4.2.1 Person Detection and Localization

Due date: **31/12/2023**
Submission Date: **21/02/2025**
Revision Date: **20/06/2025**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **Yohannes Haile** Revision: **1.4**

Project funded by the African Engineering and Technology Network (Afretec) Inclusive Digital Transformation Research Grant Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Executive Summary

Deliverable D4.2.1 focuses on the development of a ROS node that detects and localizes people under various conditions. This deliverable includes the implementation of a ROS node called `personDetection`, accompanied by a comprehensive report documenting the development process, refinement of requirements, and a detailed specification of the node's functional characteristics. Additionally, it provides a user manual with clear instructions on building and launching the ROS node. The design of the interface covers input, output, and control data, with suitable data structures and code that adhere to the software engineering standards established by the project. The functionality of the `personDetection` node is thoroughly tested and validated using various test cases, including scenarios with different lighting conditions, occlusions, and varying distances between the robot and the user. The node is also tested on the Pepper robot to confirm its reliability and real-time performance, ensuring it meets the intended objectives effectively.

Contents

1 Introduction

This document outlines the development and implementation of a ROS node for person detection and localization. The primary goal of this node is to enhance the interaction capabilities of the Pepper robot, allowing it to identify and track people within its environment, which serves as the behavior controller to initiate interaction as specified in behavior specification.

The deliverable includes a detailed report documenting the complete software development life cycle for the person detection and localization module. The requirements definition process is thoroughly covered, ensuring that all functional necessities are carefully aligned with the project's objectives. This section also highlights any identified misalignment or challenges that may arise during the development process and how they are addressed. The module design section provides an in-depth description of the person detection localization functionality, covering critical aspects such as input, output, and control data.

The operation of the module is guided by parameters defined in a configuration file, which is structured as a list of key-value pairs in the `person_detection_configuration.json` file. This configuration allows for flexible and scalable customization of the module's behavior to suit various operating conditions and requirements. Furthermore, the document emphasizes the importance of robust design principles to ensure the module's reliability and performance in real-time applications.

2 Requirements Definition

The person detection and localization module is designed to meet the following requirements, ensuring seamless integration with the Pepper robot and its ROS-based ecosystem. The key requirements for the module are as follows:

Person Detection

- Detect people in the robot's field of view using an RGB image as input.
- Detect and localize all people in the field of view when multiple people are present.
- Localize each person by determining their position in the image and drawing bounding boxes.
- Identify the centroid coordinates of each bounding box.
- Determine the depth (distance from Pepper robot) of all the people detected.

Person Labeling and Consistency

- Assign unique labels to each detected person (e.g., "Person 1").
- Maintain consistent labeling of the same person across consecutive image frames, provided the spatial displacement is within a defined tolerance.
- Reassign new labels to reappearing persons if not detected for a configurable number of images.

Configurable Parameters

- Allow customization through a configuration file (`person_detection_configuration.json`)

Input/Output Specifications

- **Input**: RGB-D image from a robot camera or external camera.
- **Output**: Annotated images with bounding boxes and labeled person records published to the `/personDetection/data` topic.

Verbose Mode

- Provide optional diagnostic output and visual debugging through an OpenCV window.

Misalignment of the module

Due to the poor camera present on the robot camera it necessitated the use of external camera. Hence, the depth information provided by the Pepper's camera is low quality hence the depth information (distance of the people from the camera) isn't accurate. In addition, this module doesn't support the simulator.

3 Module Specifications

The person detection module, implemented as a ROS node named personDetection, is designed to detect people within Pepper robot's field of view and determine their location in the image frame of reference. The module provides labeled, color-coded bounding boxes around each detected person and tracks these label across successive images for coherent detection.

The inputs for this module is an RGB image from the robot's camera or external camera (Intel RealSense D435i), the depth iamge from the robot's depth sensors or an external RGB-D camera (Intel Realsense D435i).

The outputs for this module is annotated RGB image with bounding boxes around detected people and an array of record is published with the following message `personDetection/data` topic:

- Person label representing as number
- 3D image coordinates of the bounding box centroid
- Width and height of the bounding box

`YOLO` is a deep learning framework optimized for real-time object detection, and when tailored for person detection it accurately identifying human figures in diverse environments. Once persons are detected, the system integrates SORT (Simple Online and Realtime Tracking) for tracking across frames. SORT uses techniques like Kalman filtering and the Hungarian algorithm to maintain consistent identities for each detected individual, even as they move or occlude one another. This combination between YOLO's rapid detection and SORT's efficient tracking provides a robust solution for real-time monitoring for Pepper robot.[1]

If verbosemode is set to True in the configuration file, an OpenCV window displays the detected person's bounding box. Each detected person is assigned a unique label. This provides real-time visualization and tracking for person detection.

A unit test is developed to cover various scenarios, including multiple people, partial occlusion, variable lighting conditions, and label reassignment when the person disappear. The tests are conducted using a driver-stub test platform, which utilizes recorded color and depth images stored in the data folder. Additionally, the unit tests can be executed directly on the physical robot to validate real-world performance.

4 Module Design

Image Input

The primary input for the ROS node be the Intel RealSense camera mounted on top of Pepper's head. As an alternative, the Pepper camera can also be used by configuring the camera parameter in the configuration file. However, as noted in section 2 the depth camera has very low quality. The Intel RealSense camera provides both RGB and depth images at various resolutions and frame rates, which can be customized through the launch file parameters. Table 1 outlines the available resolution and frame rate configurations for the Intel RealSense camera.

Format	Resolution	Frame Rate (FPS)	Comment
Z [16 bits]	1280x720	6, 15, 30	Depth
	848x480	6, 15, 30, 60, 90	
	640x480	6, 15, 30, 60, 90	
	640x360	6, 15, 30, 60, 90	
	480x270	6, 15, 30, 60, 90	
	424x240	6, 15, 30, 60, 90	
YUY2 [16 bits]	1920x1080	6, 15, 30	Color Stream from RGB camera (Camera D415 & D435/D435i)
	1280x720	6, 15, 30	
	960x540	6, 15, 30, 60	
	848x480	6, 15, 30, 60	
	640x480	6, 15, 30, 60	
	640x360	6, 15, 30, 60	
	424x240	6, 15, 30, 60	
	320x240	6, 30, 60	
	320x180	6, 30, 60	

Table 1: Stream Configurations for Depth and Color for Intel RealSense D435i. See the official datasheet: Intel RealSense D400 Series Datasheet.

Algorithm

YOLO (You look only Once)

The YOLO algorithm is an deep learning method for real-time object detection. It uses a single convolutional neural network to predict multiple bounding boxes and class probabilities for each object in the frame. The model divides the image into a grid and makes predictions for each cell simultaneously, allowing it to detect multiple objects quickly and accurately. YOLO's architecture significantly reduces computation time, making it suitable for real-time applications. During post-processing, overlapping boxes are filtered using non-max suppression to ensure precision and reduce false positives. When employed for person detection specifically, YOLO efficiently isolates human figures in complex environments, enabling fast and reliable identification even in dynamic scenes. In our implementation, we are using the YOLOV8s model, a state-of-the-art variant optimized for enhanced speed and accuracy. YOLOV8s retains the core strengths of the original YOLO framework while offering improvements in efficiency and detection performance. YOLOV8 comes with five variants based on the number of parameter nano(n), small(s), medium(m), large(l), and extralarge(x). We are employ-

ing the small(s) variant which is designed for resource-constrained environments and delivers faster inference.

SORT (Simple Online and Realtime Tracker)

The SORT algorithm is a lightweight multi-object tracking method that combines Kalman filtering for motion prediction and the Hungarian algorithm for data association. The process begins with detecting a person using an Intel RealSense camera mounted on Pepper's head. YOLO is employed for head detection, after which the Kalman filter predicts the motion of detected objects in subsequent frames. To associate new detections with existing tracks, SORT utilizes the Hungarian algorithm with Intersection over Union (IoU) as the matching criterion. Once matches are found, the Kalman filter updates its state with the latest information. Tracks that do not find a match are marked as lost and eventually deleted, while new detections initiate new tracks.[2]

Algorithm 1 SORT Algorithm

Require: Detected bounding boxes B_t at time t, tracked objects O_{t-1} from time $t-1$
Ensure: Updated object IDs and bounding boxes O_t

if O_{t-1} is empty **then**
  for all bounding box $b \in B_t$ **do**
    Register b as a new object with a unique ID
  end for
else
  Predict new positions of tracked objects using the **Kalman filter**
  Compute the cost matrix D using **Intersection over Union (IoU)** between B_t and predicted objects
  Solve the assignment problem using the **Hungarian algorithm**
  for all matched pairs (o, b) **do**
    Update object o with new bounding box b
    Reset disappearance counter for o
  end for
  for all unmatched objects in O_{t-1} **do**
    Increment disappearance counter
    if counter exceeds threshold **then**
      Deregister the object
    end if
  end for
  for all unmatched bounding boxes in B_t **do**
    Register bounding box b as a new object with a unique ID
  end for
end if
return updated objects O_t

Figure 1 illustrates the complete process of SORT tracking.

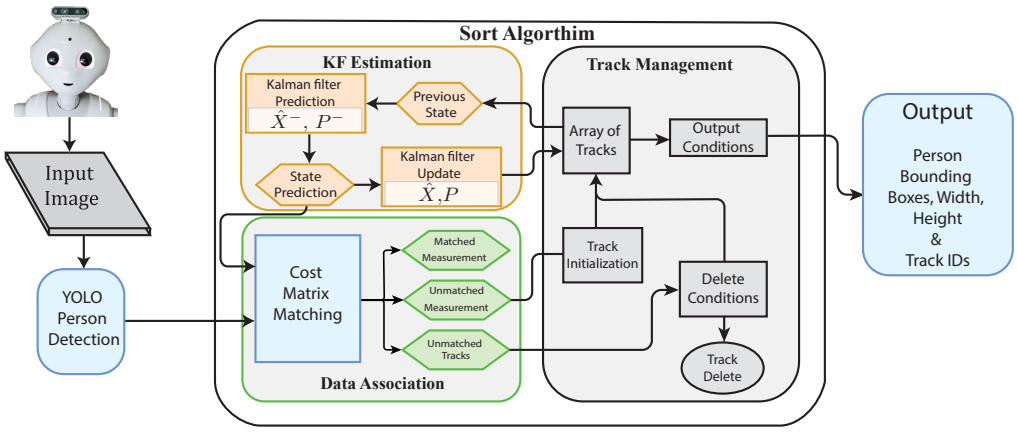

Figure 1: SORT Diagram.

5 Implementation

File Organization

The source code for conducting person detection and localization is structured into three primary components: `person_detection_application`, `person_detection_implementation`, and `person_detection_tracking`. The `person_detection_implementation` component encapsulates all the essential functionality required for executing YOLO based person detection. The `person_detection_tracking` component, on the other hand, manages tracking functionality by employing `SORT` (Simple Online and Realtime Tracking). Additionally, the person detection system is equipped with the capability to process various files critical for testing, such as configuration files, input files, and topic files. Meanwhile, the `person_detection_application` component serves as the entry point, invoking the main functions to run the person detection node and executing the functions defined within `person_detection_implementation`.

Figure 2 shows the file structure of the person detection package.

```
cssr_system
├── person_detection
│   ├── config
│   │   └── person_detection_configuration.json
│   ├── data
│   │   └── pepper_topics.dat
│   ├── launch
│   │   └── person_detection_launch_robot.launch
│   ├── models
│   │   └── person_detection_yolov8s.onnx
│   ├── msg
│   │   └── person_detection_msg_file.msg
│   ├── src
│   │   ├── person_detection_application.py
│   │   ├── person_detection_implementation.py
│   │   └── person_detection_tracking.py
│   ├── person_detection_requirements_x86.txt
│   └── README.md
├── CSSR4AfricaLogo.svg
├── CMakeLists.txt
└── Package.xml
```

Figure 2: File structure of the person detection system.

UML Diagram for the Person Detection and Localization Module

The UML diagram provides a clear structural representation of the Persons Detection and Localization Module, illustrating the relationships between its core components. It highlights inheritance, where PersonDetectionNode serves as the base class, extended by YOLOv8 for specialized person detection.

Associations between tracking components such as Sort, and TrackerUtils emphasize how detected people are tracked using Kalman filtering.

Figure 3 show the UML diagram of person_detection_implementation.py

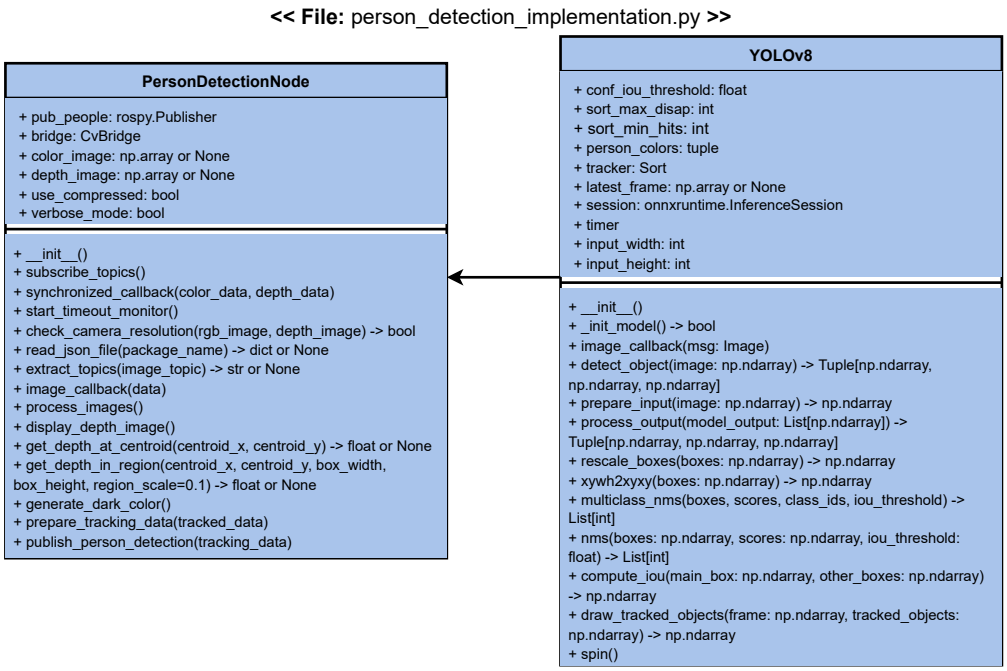

Figure 3: Person detection implementation UML.

Figure 4 show the UML diagram of person_detection_tracking.py

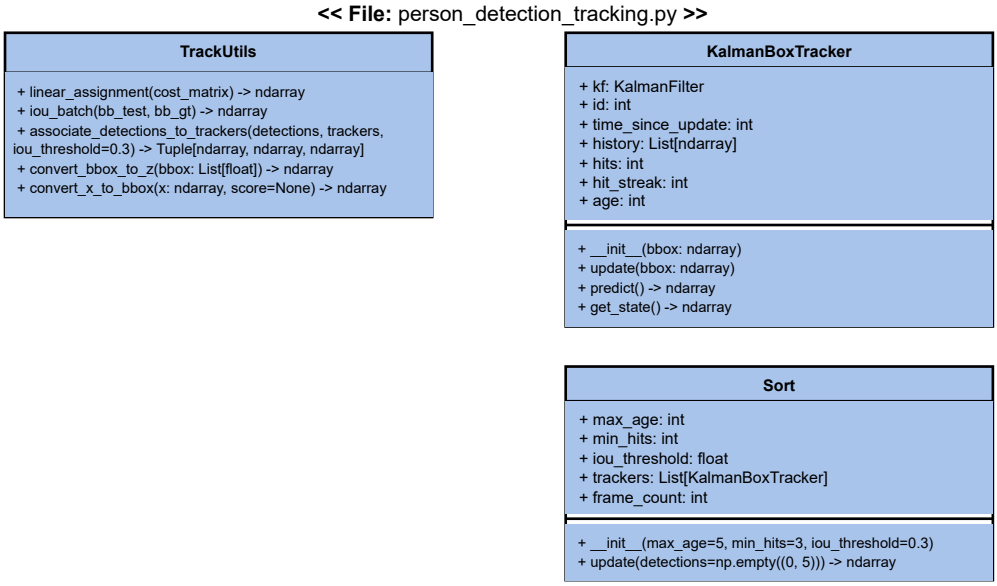

Figure 4: Person detection tracking UML.

Configuration File

The operation of the person detection node is determined by the contents of the configuration file that contains a list of key-value pairs as shown in Table 2.

The configuration file is named `person_detection_configuration.json`.

Key	Value	Description
`useCompressed`	`true or false`	Specifies to use compressed image or raw images.
`confidenceThreshold`	`<number>`	Specifies the confidence threshold for the YOLO person detection algorithm.
`sortMaxDisappeared`	`<number>`	Specifies the maximum number of frames an object can disappear for SORT tracker before being removed.
`sortMinHits`	`<number>`	Specifies the minimum number of consecutive hits required for SORT tracker initialization.
`sortIouThreshold`	`<number>`	Specifies the Intersection over Union (IoU) threshold for SORT tracker associations.
`imageTimeout`	`<number>`	Timeout (seconds) for shutting down the node after video ends.
`verboseMode`	`true or false`	Specifies whether diagnostic data is to be printed to the terminal and diagnostic images are to be displayed in OpenCV windows.

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

Input File

There is no input file the person detection node.

Output File

There is no output file the person detection node. The node using OpenCV to display the detected people with bounding boxes and labels.

Models

The person detection node uses one model for detecting people. The model is stored in the `models` directory.

Model	Description
`person_detection_yolov8s.onnx`	YOLO-based person detection model.

Table 3: Models used by the person detection node.

Topics File

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

and the value is the topic. The topics file for the robot is named `pepper_topics.dat`.

Launch File

The launch file `person_detection_launch_robot.launch` is designed to initialize either Pepper's front camera or the Intel RealSense camera based on the specified configuration. It declares several parameters that can be customized to match your network settings and camera choice:

- `pepper_robot_ip`: specifies the IP address of the Pepper robot (default: `172.29.111.230`).
- `pepper_robot_port`: specifies the communication port for Pepper (default: `9559`).
- `roscore_ip`: IP address of the ROS master (default: `127.0.0.1`)
- `network_interface`: specifies the network interface name (default: `wlp0s20f3`).
- `namespace`: sets the ROS namespace for the naoqi driver (default: `naoqi_driver`).
- `camera`: selects the camera source; set to `pepper` for Pepper's front camera or `realsense` for the Intel RealSense camera (default: `realsense`).

The file sets the parameter `/personDetection/camera` to the chosen camera and conditionally launches the corresponding nodes. If the `camera` parameter is set to `pepper`, the launch file starts the `naoqi_driver` node using the provided IP, port, network interface, and namespace. Conversely, if `camera` is set to `realsense`, it includes the RealSense camera launch file with specified parameters for image resolution, frame rate, and depth alignment. Users can adjust these default values to suit their specific hardware configurations.

Topics Subscribed

The person detection node subscribes to the topics shown in Table 4.

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

Table 4: Topics subscribed by the person detection node.

Topics Published

The person detection node publishes the topics shown in Table 5.

Topic Name	Message Type	Description
`/personDetection/data`	`personDetection/msg_file`	`An array of records containing person labels, 3D image coordinates of the bounding box, and width and height of the bounding box`

Table 5: Topics published by the person detection node.

6 Running the Person Detection Node

To run the person detection node, the user must first install the necessary software packages as outlined in Deliverable 3.3. The required packages are listed in the `person_detection_requirements.txt` file. The user can follow the README file in the person detection package to install the required packages. Referring to the implementation section of this deliverable report, the user must set the configuration file to the desired parameters. Using the key-value pair, the user can set the camera, algorithm, confidence threshold, and other parameters. The user can then run the person detection node by executing the following command in the terminal:

```
# Launch either Pepper Camera or RealSense Camera from the launch file
$ roslaunch cssr_system person_detection_launch_robot.launch camera:=
  pepper
# or
$ roslaunch cssr_system person_detection_launch_robot.launch camera:=
  realsense
```

```
# Activate the virtual environment:
source cssr4africa_face_person_detection_env/bin/activate
```

```
# Run the person detection node
$ rosrun cssr_system person_detection_application.py
```

If the user has set the verboseMode to True in the configuration file, the person detection node displays the detected people with bounding boxes and labels in an OpenCV window.

7 Unit Test

The unit test is designed to validate the person detection node's functionality under various scenarios, including multiple people, occlusions, and varying lighting conditions. The test can be performed using a driver-stub test platform, which utilizes recorded color and depth images stored in the data folder as a rosbag file. The unit test can also be executed directly on the physical robot to validate real-world performance.

The person detection unit test file structure is as shown in Figure 5

```
unit_test
├── person_detection_test
│   ├── config
│   │   └── person_detection_test_configuration.json
│   ├── data
│   │   ├── person_detection_test_input_single_person.bag
│   │   ├── person_detection_test_input_multiple_people.bag
│   │   ├── person_detection_test_input_lighting_1.bag
│   │   └── person_detection_test_input_lighting_2.bag
│   ├── launch
│   │   ├── person_detection_test_launch_robot.launch
│   │   └── person_detection_test_launch_test_harness.launch
│   ├── msg
│   │   └── person_detection_test_msg_file.msg
│   ├── src
│   │   ├── person_detection_test_application.py
│   │   └── person_detection_test_implementation.py
│   └── README.md
├── CSSR4AfricaLogo.svg
├── CMakeLists.txt
└── Package.xml
```

Figure 5: File structure of the person detection unit test.

The test cases for the person detection node that are going to be evaluated are as shown in Table 6.

Test Case	Description
Single Person Detection	Verify the person detection node's ability to detect and localize a single person in the image frame, as well as evaluate the distance at which the person is detected.
Multiple Person Detection	Validate the person detection node's capability to detect and localize multiple people in the image frame.
Person Tracking	Test the person detection node's tracking functionality by tracking a person across multiple frames.
Occlusion Handling	Evaluate the person detection node's performance in handling partial occlusions of person.
Lighting Conditions	Test the person detection node's robustness under varying lighting conditions.

Table 6: Test cases for person detection node evaluation.

Configuration File

The configuration file for the person detection unit test is named `person_detection_test_configuration.json` and contains the key-value pairs shown in Table 7.

Key	Value	Description
`algorithm`	`sixdrep or mediapipe`	Specifies the algorithm used for face detection and head pose estimation.
`useCompressed`	`true or false`	Specifies to use compressed image or raw images.
`saveVideo`	`true or false`	Specifies whether to save the output video of the test.
`saveImage`	`true or false`	Specifies whether to save individual image frames from the test.
`videoDuration`	`<number>`	Specifies the duration (in seconds) for which the video is saved.
`imageInterval`	`<number>`	Specifies the time interval (in seconds) at which images are captured and saved.
`recordingDelay`	`<number>`	Delay (seconds) before recording starts.
`maxFrameBuffer`	`<number>`	Maximum number of frames to store in buffer.
`verboseMode`	`true or false`	Specifies whether detailed logs and diagnostic images are displayed during execution.

Table 7: Configuration file key-value pairs for the person detection test.

Note: Valid values for `bag_file` include: `single_person`, `multiple_people`, `lighting_1`, `lighting_2`.

Input File

The node takes recorded RGB and depth video saved as rosbag file as an input.

Output File

The node has the option to save a recorded video and/or image with the bounding box and mutual gaze determined.

Launch File

The launch file `person_detection_test_launch_robot.launch` is designed to support testing the person detection node with various input sources: a live feed from Pepper's front camera, the Intel RealSense camera, or a recorded rosbag video. It provides several configurable arguments to customize the test environment:

- `camera`: selects the camera input source; set to `pepper` for Pepper's camera, `realsense` for the RealSense camera, or `video` to use a recorded rosbag (default: `video`).
- `bag_file`: specifies which bag file to play; only used when `camera=video` (default: `single_person`).
- `robot_ip`: IP address of the Pepper robot (default: `172.29.111.230`).
- `roscore_ip`: IP address of the ROS master (default: `127.0.0.1`).
- `robot_port`: communication port for the Pepper robot (default: `9559`).
- `network_interface`: name of the network interface for ROS communication (default: `wlp0s20f3`).
- `namespace`: ROS namespace for the naoqi driver (default: `naoqi_driver`).

Depending on the selected input method, the launch file performs the following:

- If `camera` is set to `realsense`, it launches the RealSense camera driver with pre-configured resolution and frame rate settings.
- If `camera` is set to `pepper`, it launches the `naoqi_driver` node to stream Pepper's camera data.
- If `camera` is set to `video`, it plays a specified bag file from the `unit_test` package in a loop.

This setup allows flexible testing of the face detection node using live or recorded data sources with consistent parameters across different hardware.

The launch file `person_detection_test_launch_test_harness.launch` launches the person_detection node and person_detection_test node that run the unit test based on configuration file in the person_detection_test.

Topics Subscribed

The person detection test node subscribes to the topics shown in Table 8.

Camera	Topic Name	Message Type
RealSenseCameraRGB	/camera/color/image_raw	sensor_msgs/Image
RealSenseCameraDepth	/camera/aligned_depth_to_color/image_raw	sensor_msgs/Image
PepperFrontCamera	/naoqi_driver/camera/front/image_raw	sensor_msgs/Image
PepperDepthCamera	/naoqi_driver/camera/depth/image_raw	sensor_msgs/Image

Table 8: Topics subscribed by the person detection test node.

.

In addition it subscribes to /personDetection/data to draw the bounding box and save the video in the data folder.

Running Person Detection and Localization Unit Test

The user can execute the following commands in the terminal to run the unit test for person detection node.

```
# Launch unit test for person_detection by setting the camera to
  realsense, pepper or video.
$ roslaunch unit_test person_detection_test_launch_robot.launch camera
  :=realsense
# or
$ roslaunch unit_test person_detection_test_launch_robot.launch camera
  :=pepper
# or
$ roslaunch unit_test person_detection_test_launch_robot.launch camera
  :=video
```

```
# Activate the virtual environment:
source cssr4africa_face_person_detection_env/bin/activate
```

```
# Run the person detection node
$ roslaunch unit_test person_detection_test_launch_robot.launch
```

The user should validate the node's performance by visually inspecting the images displayed through OpenCV.

References

[1] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.

[2] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *arXiv preprint arXiv:1703.07402*, 2017.

Principal Contributors

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

Yohannes Haile, Carnegie Mellon University Africa.
David Vernon, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Natasha Mutangana.
10 January 2024.

Version 1.1

Complete rewrite of the deliverable.
Yohannes Haile.
21 February 2025.

Version 1.2

Changed the notation in Figure 1.
Changed the file structure tree in Figure 2.
Updated the UML diagram on Figure 3.
Updated the configuration Table 2.
Updated the file structure in the person_detection in Figure 3.
Updated the file structure in the person_detection_test in Figure 6.
Updated the Topics subscribed in Table 4.
Added width and height of the bounding box.
Updated the launch file for person_detection(Page 13).
Removed speaker option from the configuration file Table 7.
Updated the name of the rosbag files to use (Page 20).
Added width and height in the message field for the msg_file for the person_detection.
Added input file, output file, running the unit test launch file and Topics subscribed for the unit test.
Removed future tense in the report.
Yohannes Haile.
29 April 2025.

Version 1.3

Fixed typos and added explicit references to tables.
David Vernon.
16 June 2025.

Version 1.4

Added explicit references to the figures.
Yohannes Haile.
20 June 2025.