CSSR for Africa

Culturally Sensitive Social Robotics
for Africa

D4.2.2 Face and Mutual Gaze 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: **1.3**

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 D4.2.2 focuses on the development of a ROS node that detects and localizes human faces under various conditions and determines whether mutual gaze is established between the Pepper robot and the human user through head pose estimation. This deliverable includes the implementation of a ROS node called `faceDetection`, 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 `faceDetection` 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 face detection, localization, and mutual gaze detection using head pose estimation for the Pepper robot. The primary goal of this node is to enhance the interaction capabilities of the Pepper robot, allowing it to identify and track human faces within its environment, which serves as a cornerstone for creating natural and intuitive interactions. The mutual gaze detection functionality further strengthens this interaction by leveraging head pose estimation to identify moments when users establish eye contact with the robot—an essential component of engaging and socially aware behavior.

The deliverable includes a detailed report documenting the complete software development life cycle for the face 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 face detection and mutual gaze 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 `face_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 face and mutual gaze 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:

Face Detection

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

Face Labeling and Consistency

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

Gaze Direction Estimation

- Analyze head pose to estimate gaze direction, enabling the detection of mutual gaze between the robot and the user.

Configurable Parameters

- Allow customization through a configuration file (`face_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 face records published to the `/faceDetection/data` topic.

Verbose Mode

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

Misalignment of the module

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

3 Module Specifications

The face detection module, implemented as a ROS node named faceDetection, is designed to detect faces within Pepper robot's field of view and determine their location and gaze direction in the image frame of reference. The module provides labeled, color-coded bounding boxes around each detected face and tracks these label across successive images for coherent detection. The module doesn't perform face recognition but ensure consistency in labeling based on spatial proximity and configurable tolerance settings.

The inputs for this module are 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 are annotated RGB image with bounding boxes around detected faces and an array of records is published with the following message `faceDetection/data` topic:

- Face label representing as number
- 3D image coordinates of the bounding box centroid
- True/False value determining whether a mutual gaze is established.
- Width and Height of the bounding box.

The module utilizes two methods for face detection: `MediaPipe` and a `YOLO` (You Only Look Once) [1] based face detection model. Each method is optimized for different scenarios, giving the flexibility to the user to select between these two algorithms based on their specific requirements.

`MediaPipe` is ideal for face detection within shorter distances and when processing is limited to CPU-based computing. It efficiently detects facial landmarks, including key points such as the distance between the eyes, nose, and mouth, which are essential for inferring the 3D orientation of the head pose. However, its performance and detection range are limited when faces are located farther from the camera.

On the other hand, the `YOLO-based model` is a deep learning approach designed to detect human heads at greater distances, leveraging GPU acceleration for robust and accurate face detection, even in challenging environments. Once a face is detected, the model applies `SixDRep (6D Rotation Representation for Unconstrained Head Pose Estimation)` [2] to determine the head's orientation. This method offers a representation of the head's rotation across six degrees of freedom, allowing the system to accurately assess mutual gaze.

By providing the option to choose between MediaPipe and YOLO, the module ensures flexibility across various settings. MediaPipe can be selected for lightweight, close-range scenarios, while YOLO can be chosen for long-range detection and environments where GPU resources are available.

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

A unit test is developed to cover various scenarios, including multiple faces, partial occlusion, variable lighting conditions, and label reassignment when faces disappear. The tests is 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 is 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 below 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.

Algorithms

MediaPipe

MediaPipe is open-source framework developed by Google that provides efficient solution for real-time computer vision application. Among the various capablitlies, MediaPipe can be utilized for head pose estimation by leveraging its face detection and face landmark modules. The process involves detecting key facial landmarks, such as the eyes, nose and mouth, to determine the orientation of the head in 3D space. MediaPipe's Face Mesh module identifies 468 distinct facial landmarks with high precision, allowing for robust tracking of head movements. Once these landmarks are detected, they are used as input to compute the head's rotation and translation relative to the camera coordinate system. Figure 1 shows the landmarks detected on the human face.

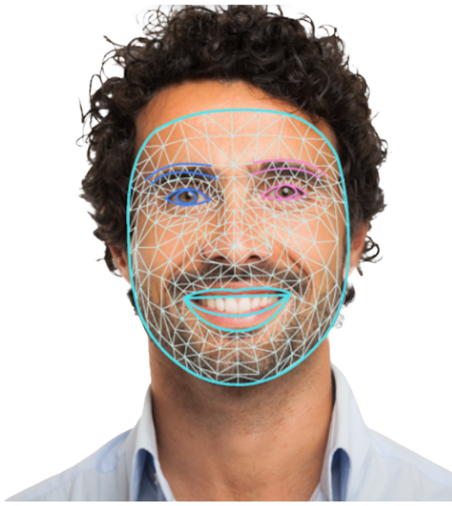

Figure 1: MediaPipe Face land marks [3].

By fitting a 3D face model to the detected 2D landmarks using Perspective-n-Point(PnP), it calculates Euler angles (yaw, pitch, and roll) to represent the head's orientation.

SixDRepNet

SixDRepNet is a deep learning-based model designed specifically for head pose estimation. Unlike traditional methods like MediaPipe, which rely on 2D-to-3D correspondences and facial landmarks, SixDRepNet takes a direct regression approach. It predicts the yaw, pitch, and roll angles directly from input images, without requiring explicit 3D landmark annotations or predefined face models. The process begins with YOLO, which detects the face in the input image and generates a bounding box. The detected face region is then cropped and resized to the required input dimensions of 224 × 224 pixels for the head pose estimation model. The preprocessed face image is fed into SixDRepNet, which outputs a 6D rotation representation. This representation is converted to a rotation matrix and used to compute the head's yaw, pitch, and roll angles.

Centroid Tracker

For tracking faces across frames, the centroid tracker is used together with MediaPipe. The tracker ensures the detected faces remains consistently tracked even as they move or momentrily disappear. Mediapipe detects facial landmarks and provides the bounding boxes around faace in each frame, while the centroid Tracker assigns and maintains unique IDs for each detected face. It calculates the centroid of the bounding boxes and tracks it across consecutive frames by measuring the Euclidean distance to match centroids. If a match is found, the corresponding face ID is updated; otherwise, a new ID is assigned. The tracker handles cases where faces temporarily disappear by keeping track of missed detection and deregistering them only after a set threshold of consecutive frames.

Algorithm 1 Centroid Tracker Algorithm

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

if O_{t-1} is empty **then**
  for all centroid $c \in C_t$ **do**
    Register c as a new object with unique ID
  end for
else
  Compute distance matrix D between C_t and O_{t-1}
  Match centroids using nearest-neighbor approach
  for all matched pair (o, c) **do**
    Update object o with new centroid c
    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 centroids in C_t **do**
    Register centroid c as a new object with unique ID
  end for
end if
return updated objects O_t

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 face 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 [4].

Algorithm 2 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 **??** illustrates the complete process of SORT tracking.

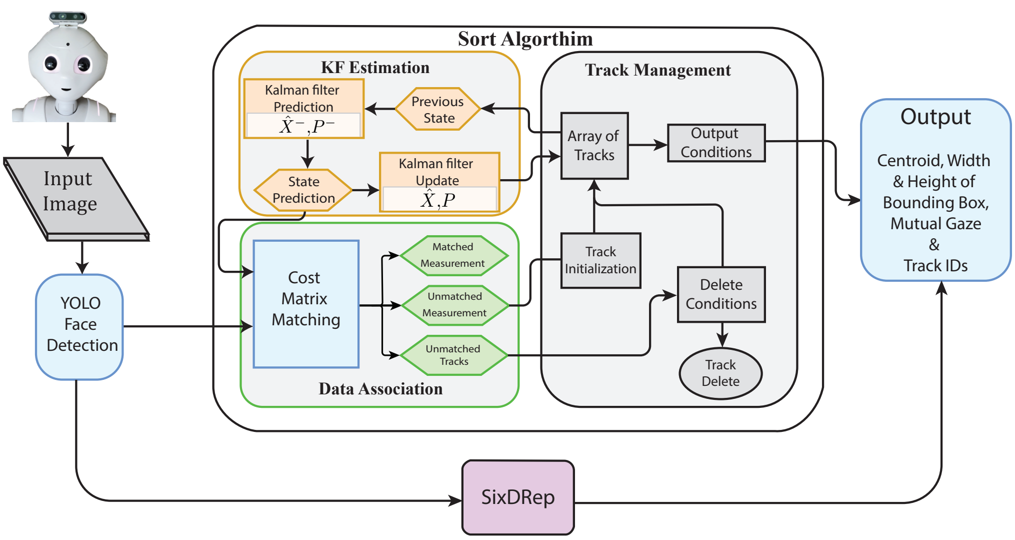

Figure 2: SORT Diagram.

5 Implementation

File Organization

The source code for conducting face detection, mutual gaze detection, and localization is structured into three primary components: `face_detection_application`, `face_detection_implementation`, and `face_detection_tracking`. The `face_detection_implementation` component encapsulates all the essential functionality required for executing both face detection and mutual gaze detection, utilizing `MediaPipe` and `SixDRepNet`. The `face_detection_tracking` component, on the other hand, manages tracking functionality by employing either the Centroid Tracker or `SORT` (Simple Online and Realtime Tracking). Additionally, the face 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 `face_detection_application` component serves as the entry point, invoking the main functions to run the face detection node and executing the functions defined within `face_detection_implementation`.

Figure 3 shows the file structure of the face detection package.

```
cssr_system
├── face_detection
│   ├── config
│   │   └── face_detection_configuration.json
│   ├── data
│   │   └── pepper_topics.dat
│   ├── launch
│   │   └── face_detection_launch_robot.launch
│   ├── models
│   │   ├── face_detection_goldYOLO.onnx
│   │   └── face_detection_sixdrepnet360.onnx
│   ├── msg
│   │   └── face_detection_msg_file.msg
│   ├── src
│   │   ├── face_detection_application.py
│   │   ├── face_detection_implementation.py
│   │   └── face_detection_tracking.py
│   ├── face_detection_requirements_x86.txt
│   └── README.md
├── CSSR4AfricaLogo.svg
├── CMakeLists.txt
└── Package.xml
```

Figure 3: File structure of the face detection system.

UML Diagram for the Face and Mutual Gaze Detection and Localization Module

The UML diagram provides a clear structural representation of the Face and Mutual Gaze Detection and Localization Module, illustrating the relationships between its core components. It highlights inheritance, where FaceDetectionNode serves as the base class, extended by MediaPipe and SixDRepNet for specialized face detection and head pose estimation. Associations between tracking components such as Sort, CentroidTracker, and TrackerUtils emphasize how detected faces are tracked using Kalman filtering and centroid-based methods. Composition relationships are depicted, showing that SixDRepNet integrates YOLOONNX for face detection as an essential part of head pose estimation, ensuring a modular and scalable system.

Figure 4 shows the UML diagram of face_detection_implementation.py.

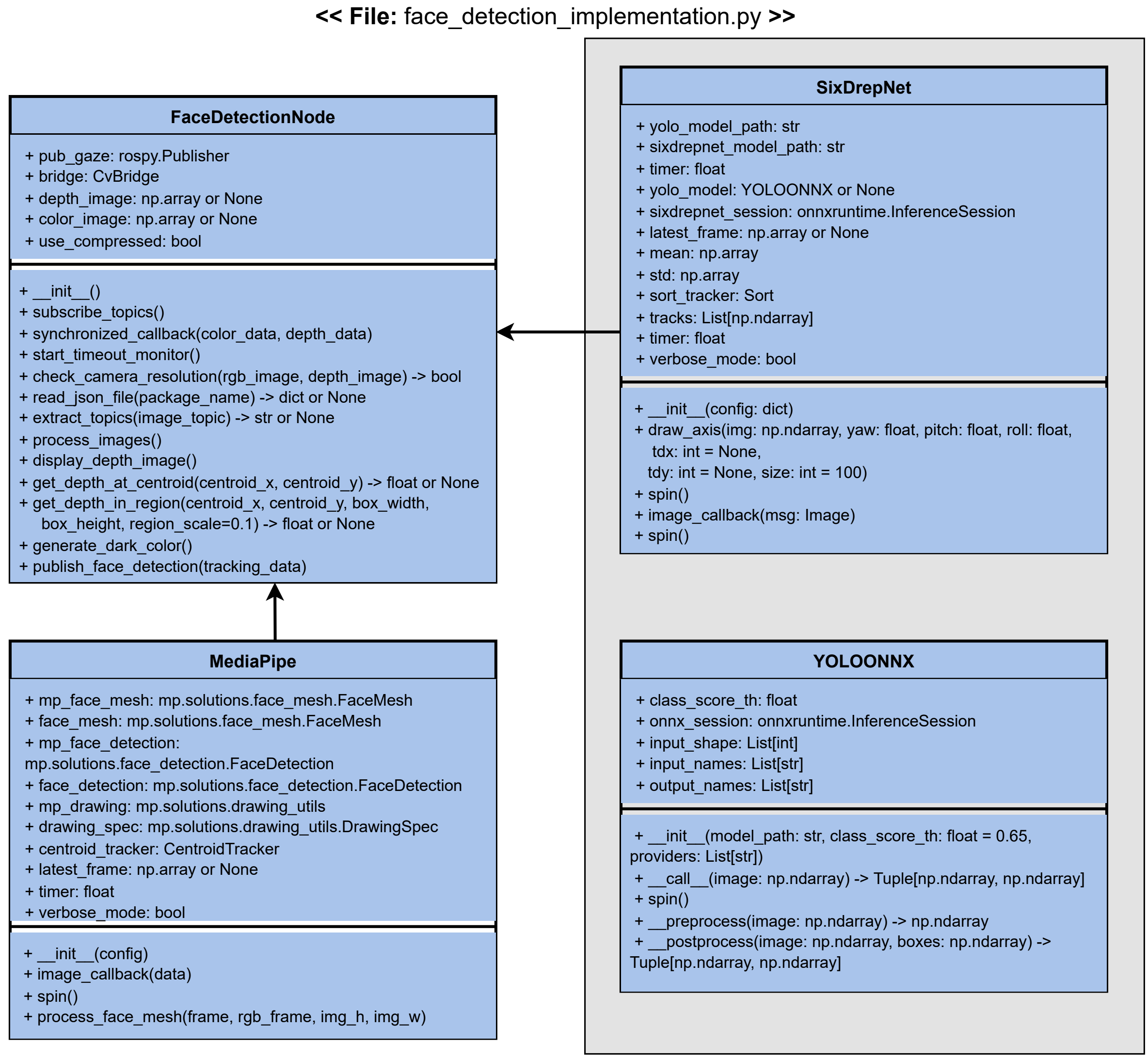

Figure 4: Face detection implementation UML.

Figure 5 shows the UML diagram of face_detection_tracking.py.

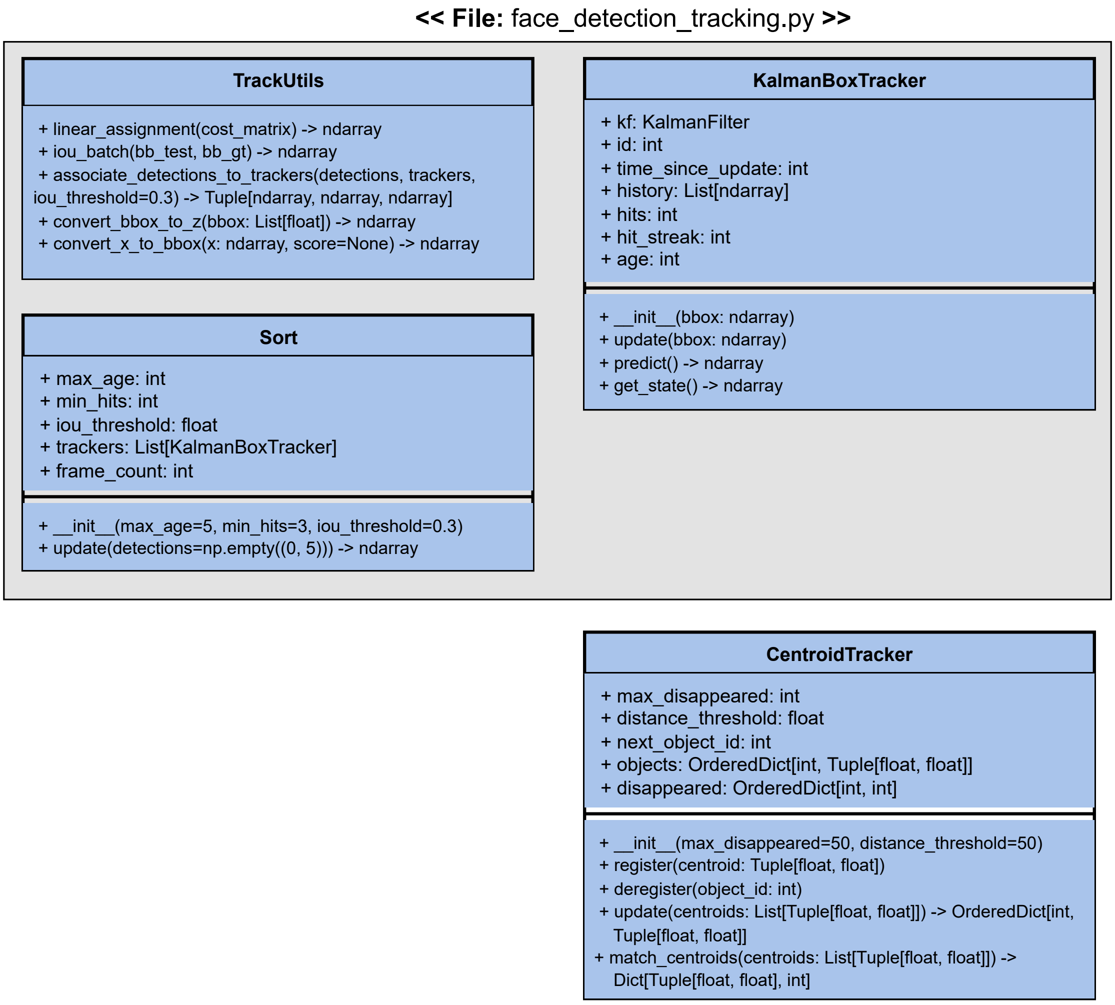

Figure 5: Face detection tracking UML.

Configuration File

The operation of the face detection node is determined by the contents of the configuration file that contains a list of key-value pairs as shown on the Table 2. The configuration file is named `face_detection_configuration.json`.

Key	Value	Description
`algorithm`	`mediapipe or sixdrep`	Specifies which algorithm to use.
`useCompressed`	`true or false`	Specifies to use compressed image or raw images.
`mpFacedetConfidence`	`<number>`	Specifies the confidence threshold for the MediaPipe face detection algorithm.
`mpHeadposeAngle`	`<number>`	Specifies the maximum angular deviation (in degrees) for MediaPipe head pose estimation.
`centroidMaxDistance`	`<number>`	Specifies the maximum allowed distance (in pixels) between centroids for tracking continuity.
`centroidMaxDisappeared`	`<number>`	Specifies the maximum number of frames a centroid can disappear before being considered lost.
`sixdrepnetConfidence`	`<number>`	Specifies the confidence threshold for the SixDRepNet pose estimation algorithm.
`sixdrepnetHeadposeAngle`	`<number>`	Specifies the maximum angular deviation (in degrees) for SixDRepNet head pose estimation.
`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 face detection node.

Input File

There is no input file the face detection node.

Output File

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

Launch File

The launch file `face_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`).
- `network_interface`: specifies the network interface name (default: `wlp0s20f3`).
- `roscore_ip`: IP address of the ROS master (default: `127.0.0.1`)
- `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 `/faceDetection/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.

Models

The face detection node uses two models for face detection and head pose estimation. The models are stored in the `models` directory. The models are shown in Table 3.

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

Table 3: Models used by the face detection node.

Topics File

For the test, a selected list of the topics for the robot is stored in the topics file. The topic files are written in the .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`.

Topics Subscribed

The face 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 face detection node.

Topics Published

The face detection node publishes to the topics shown in Table 5.

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

Table 5: Topics published by the face detection node.

6 Running the Face Detection Node

To run the face detection node, the user must first install the necessary software packages as outlined in Deliverable 3.3. The required packages are listed in the `face_detection_requirements_x86.txt` file. The user can follow the README file in the face detection package to install the required packages. Refering to the implementation section of this deliverble 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 face detection node by executing the following command in the terminal:

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

Source the python environment you setup for face_detection.

```
# Activate the virtual environment:
$ source $HOME/workspace/pepper_rob_ws/face_person_detection/bin/activate
```

```
# Run the face detection node
$ rosrun cssr_system face_detection_application.py
```

If the user has set the verbose mode to True in the configuration file, the face detection node displays the detected faces with bounding boxes and labels, as well as the mutual gaze detection in an OpenCV window. The user can then interact with the Pepper robot to establish mutual gaze and observe the system's response.

7 Unit Test

The unit test is designed to validate the face detection node's functionality under various scenarios, including multiple faces, 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 face detection unit test file structure is as follows:

```
unit_test
├── face_detection_test
│   ├── config
│   │   └── face_detection_test_configuration.json
│   ├── data
│   │   ├── face_detection_test_input_single_face.bag
│   │   ├── face_detection_test_input_multiple_faces.bag
│   │   ├── face_detection_test_input_mutual_gaze.bag
│   │   ├── face_detection_test_input_lighting_1.bag
│   │   └── face_detection_test_input_lighting_2.bag
│   ├── launch
│   │   ├── face_detection_test_launch_robot.launch
│   │   └── face_detection_test_launch_test_harness.launch
│   ├── msg
│   │   └── face_detection_test_msg_file.msg
│   ├── src
│   │   ├── face_detection_test_application.py
│   │   └── face_detection_test_implementation.py
│   └── README.md
├── CSSR4AfricaLogo.svg
├── CMakeLists.txt
└── Package.xml
```

Figure 6: File structure of the face detection unit test.

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

Test Case	Description
Single Face Detection	Verify the face detection node's ability to detect and localize a single face in the image frame, as well as evaluate the distance at which the face is detected.
Multiple Face Detection	Validate the face detection node's capability to detect and localize multiple faces in the image frame.
Face Tracking	Test the face detection node's tracking functionality by tracking a face across multiple frames.
Mutual Gaze Detection	Confirm the face detection node's ability to detect mutual gaze between the robot and the user.
Occlusion Handling	Evaluate the face detection node's performance in handling partial occlusions of faces.

Test Case	Description
Lighting Conditions	Test the face detection node's robustness under varying lighting conditions.

Table 6: Test cases for face detection node evaluation (continued across pages).

Configuration File

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

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

Table 7: Configuration file key-value pairs for the face detection test (continued across pages if needed).

Note: Valid values for `bag_file` include: `single_face`, `multiple_faces`, `mutual_gaze`, `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 `face_detection_test_launch_robot.launch` is designed to support testing the face 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_face`).

- `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 `face_detection_test_launch_test_harness.launch` launches the face_detection node and face_detection_test node that runs the unit test for based on configuration file in the face_detection_test.

Topics Subscribed

The face 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 face detection test node.

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

Running Face and Mutual Gaze 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 face_detection by setting the camera to
  realsense, pepper or video.
$ roslaunch unit_test face_detection_test_launch_robot.launch camera:=
  realsense
# or
$ roslaunch unit_test face_detection_test_launch_robot.launch camera:=
  pepper
# or
$ roslaunch unit_test face_detection_test_launch_robot.launch camera:=
  video
```

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

```
# Run the face detection node
$ roslaunch unit_test face_detection_test_launch_robot.launch
```

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

[3] Google AI. Mediapipe face landmarker: Real-time face detection and landmark detection. https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker, 2024. Accessed: February 15, 2025.

[4] 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.
Yohannes Haile.
21 February 2025.

Version 1.1
Changed the notation in Figure 2.
Updated the configuration Table 2.
Updated the UML diagram on Figure 4.
Updated the file structure in the face_detection in Figure 3.
Updated the file structure in the face_detection_test in Figure 6.
Updated the Topics published table in Table 5.
Updated command for the launch files. (Page 18).
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 face_detection.
Added input file, output file, running the face detection unit test, launch file and Topics subscribed for the unit test.
Removed future tense in the report.
Yohannes Haile.
29 April 2025.

Version 1.2
Fixed typos.
David Vernon.
16 June 2025

Version 1.3
Added explicit references to the table and figures.
Yohannes Haile.
20 June 2025