CSSR for

Culturally Sensitive Social Robotics for Africa

D4.1 Sensor Test

Due date: **1/10/2023**
Submission Date: **26/03/2024**
Revision Date: **n/a**

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

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

Responsible Person: **Yohannes Haile** Revision: **1.0**

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 D4.1 aims to create a robust suite of unit tests to verify that sensor data is successfully acquired on each sensor topic. The key components of this deliverable include a ROS node called `sensorTest`, a report documenting the development process, requirements refinement and specifying functional characteristics, and a user manual on how to build and launch the module. The interface design encompasses input, output, and control data. Suitable data structures are specified, and coding adheres to software engineering standards.

Contents

1 Introduction

This deliverable is dedicated to ensuring the optimal functionality of the sensors on the Pepper robot (or simulator), with a primary focus on verifying their ability to successfully acquire data from their designated topics. Illustrated in Figure 1, the diverse array of sensors on Pepper plays a critical role in shaping its perception and interactive capabilities.

At the core of Pepper's visual perception are the dual RGB cameras, functioning as its primary set of visual sensors. These cameras capture images with a broad range of resolutions, offering Pepper a comprehensive view from two distinct fields of vision. Working in tandem with the RGB cameras, the depth sensor enriches Pepper's spatial understanding by providing essential information about the distance to objects in its environment. The microphones are equipped to give Pepper auditory perception capabilities. This feature is pivotal for speech recognition, sound detection, and localization which facilitates effective communication with users.

Supplementing Pepper's tactile interaction capabilities are the touch sensors distributed in the head and hand sensor areas. These sensors enhance the robot's ability to engage naturally with humans and its surroundings, fostering interactive and intuitive experiences. The sonar sensor further contributes to Pepper's environmental awareness by gauging distances to nearby objects, proving integral to tasks such as navigation, obstacle avoidance, and maintaining spatial awareness during movement.

For Pepper's physical coordination, the gyroscope and accelerometer sensors are used to detect changes in orientation and acceleration respectively. This information is vital for the robot to maintain balance and coordination across a spectrum of activities. Additionally, the laser sensor introduces another layer to Pepper's environmental perception, emitting laser beams and measuring their reflections.

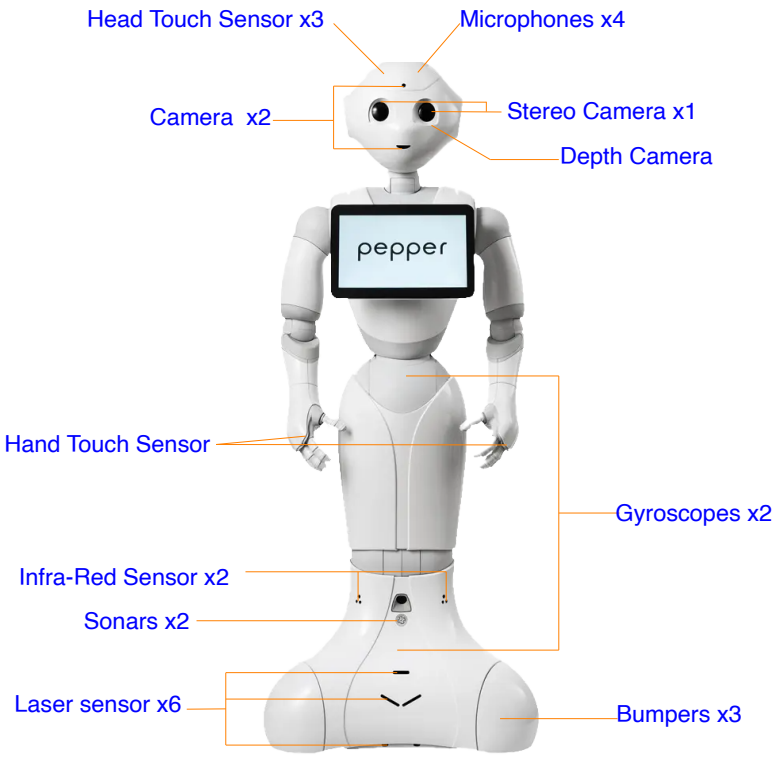


Figure 1: Pepper robot sensors

The ROS-NAOqi driver provides a bridge between ROS(Robot Operating System) and NAOqiOS, the operating system of Aldebaran's Nao, Pepper, and Romeo robots [1]. The driver publishes all sensor and actuator data as well as basic diagnostic information for battery and temperature. It also subscribes to RViz and cmd_vel for teleportation. The ROS-NAOqi Interface comprises ROS nodes that interact with NAOqi via the NAOqi messaging API. These nodes can subscribe to and publish ROS topics, facilitating communication with other ROS nodes.

2 Requirements Definition

The sensor test provides a systematic process aimed at understanding and documenting the functionality needs that the module must fulfill. This deliverable is important in identifying the specific users expectations, ensuring that the module will be capable of performing the precise sensor tests in various scenarios and environments. The test encompasses a thorough examination of the intended functionality, the user's needs, and the project's objectives to ensure that the module will meet the requirements of the project.

As seen from the diagram in Figure 1, the Pepper robot is equipped with a variety of sensors that are essential for its operation. For this deliverable, the camera sensors will be tested to ensure that they can capture images and videos. The depth sensor will be tested to ensure that it can capture depth information. The microphone sensor will be tested to ensure that it can capture audio data. The IMU sensor will be tested to ensure that it can capture orientation and acceleration data. The sonar sensor will be tested to ensure that it can capture distance data. The laser sensor will be tested to ensure that it can capture distance data. The odometry sensor will be tested to ensure that it can capture odometry data. The joint states sensor will be tested to ensure that it can capture joint state data for the joint actuator of the robot. Exceptionally, the loudspeaker, even though it is not a sensor, will be tested to ensure that it can speak out the text provided to it.

Misalignment of the Module

Due to the unavailability of the head touch, hand touch, and bumper sensors, the module will not be able to test these sensors. Given the current status of the project, we have opted to suspend work on it, as we have not identified a direct application for these sensors. However, should the future direction of our project necessitate the integration of these sensors due to their potentially critical role, we will undertake efforts to ensure their functionality.

In regards to the sensor test for the simulator, the microphone sensor, IMU, stereo camera, and loudspeaker test will not be conducted as they are not available in the simulator. Furthermore, the Gazebo simulator will raise errors when testing for the camera is conducted in parallel mode. Therefore, the camera sensors should be tested in sequential mode.

3 Module Specifications

To validate the functionality of the sensors, the sensor test module is designed to execute a series of tests on the sensors. The sensor test module is a ROS node that subscribes to the sensor topics and verifies that the sensors can capture data. The sensor test module is designed to test the following

- Back Sonar
- Front Sonar
- Bottom Camera
- Front Camera
- Depth Camera
- Stereo Camera ***(not available in the simulator)***
- Laser Sensor
- Microphone ***(not available in the simulator)***
- Joint State
- Odometry
- IMU ***(not available in the simulator)***
- Speech (Loudspeaker) ***(not available in the simulator)***

All of the sensors will be tested for 10 seconds, whereas the microphone sensor will be tested for a total of 40 seconds. The microphone sensor will be tested for a total of 40 seconds where each microphone will be tested for 10 seconds. For the loudspeaker, a text will passed on it to speak out. The sensor test will be conducted in two modes: parallel and sequential. In the parallel mode, all the sensors will be tested at the same time, whereas in the sequential mode, the sensors will be tested one after the other. The sensor test will be conducted for both the robot and the simulator.

To test the sensors, the sensor test module will read various configuration files that contain the necessary information for the test. The configuration files include the sensor test configuration file, the sensor test input file, and the topics file. The sensor test configuration file contains the platform to be tested, the mode of the test, and the topics file for the robot and simulator. The sensor test input file contains the sensors to be tested. The topics file contains the list of topics for the robot and simulator. The sensor test module will publish the results of the test to the sensor test output file. Furthermore, the sensor test module will record the video and audio data for the camera and microphone sensors respectively, and store them in the data directory as specified by the user in the `sensorTestImplementation.cpp` file.

4 Interface Design

Source Code

The source code for conducting sensor tests is structured into two primary components: `sensorTestApplication` and `sensorTestImplementation`. The sensorTestImplementation component encapsulates all the essential functionality required for executing comprehensive sensor tests. This includes all of the tests for the sensors that consist of the bottom, front, and stereo camera, depth camera, sonar, laser, odom, microphone, speech, and joint states. In addition to sensor testing capabilities, this component is also equipped with the functionality to process various files critical for the testing process, such as configuration files, input files, and topics files.

On the other hand, the sensorTestApplication invokes those functions for the testing process. It is tasked with executing functions defined within the sensorTestImplementation, effectively managing the sensor test operations.

Here is the file structure of the pepper interface tests package:

```
pepper_interface_tests
├── config
│   ├── actuatorTestConfiguration.ini
│   ├── actuatorTestInput.ini
│   ├── sensorTestInput.ini
│   └── sensorTestConfiguration.ini
├── data
│   ├── pepperTopics.dat
│   ├── sensorTestOutput.dat
│   └── simulatorTopics.dat
├── include
│   └── pepper_interface_tests
│       ├── actuatorTestInterface.h
│       └── sensorTestInterface.h
├── launch
│   ├── actuatorTestLaunchRobot.launch
│   ├── sensorTestLaunchRobot.launch
│   └── interfaceTestLaunchSimulator.launch
├── src
│   ├── actuatorTestApplication.cpp
│   ├── actuatorTestImplementation.cpp
│   ├── sensorTestApplication.cpp
│   └── sensorTestImplementation.cpp
├── README.md
├── CMakeLists.txt
└── package.xml
```

Configuration File

The operation of the `sensorTest` node is determined by the contents of the configuration file that contains a list of key-value pairs as shown below.

The configuration file is named `sensorTestConfiguration.ini`

Table 1: Configuration file for the sensor test.

Key	Value	Description
`platform`	`simulator` or `robot`	Specifies the platform to be tested. The platform can be set to either `simulator` or `robot`.
`robotTopics`	`robotTopics.dat`	Specifies the name of the robot topics file. The robot topics file contains the list of topics for the robot.
`simulatorTopics`	`simulatorTopics.dat`	Specifies the name to the simulator topics file. The simulator topics file contains the list of topics for the simulator.
`mode`	`parallel` or `sequential`	Specifies the mode of the test. The mode can be either `parallel` or `sequential`. The parallel mode runs all the tests in parallel. The sequential mode runs all the tests sequentially.

Input File

The input file is used to specify which sensors to test by using the sensor name as the key and `True` or `False` as the value. The sensor name must be the same as the sensor name in the topics file.

The input file is named `sensorTestInput.ini`

Output Data File

The output data file is used to store the results of the sensor test. The output data file is written in the .dat file format. The data file is written as the sensor name and the result of the test. The output result is dependent upon the sensors that are being tested. The detail of the output file is shown in the Result section.

The output data file is named `sensorTestOutput.dat`

Furthermore, the recorded video and audio files are stored in the `data` directory. The recorded video and audio files are named `frontCameraOutput.mp4`, `bottomCameraOutput.mp4`, `stereoCameraOutput.mp4`, `depthCameraOutput.mp4`, and `microphoneOutput.wav`

Topics File

For the test, a selected list of the topics for the robot and simulator 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 sensor name and the value is the topic

The topics file for the robot is named `robotTopics.dat` and the topics file for the simulator is named `simulatorTopics.dat`

Topics Subscribed

The sensorTest node subscribes to the following topics:

Table 2: Topics subscribed by the sensorTest node.

Topic	Sensor	Platform
/naoqi_driver/camera/bottom/image_raw	Bottom Camera	robot
/naoqi_driver/camera/depth/image_raw	Depth Camera	robot
/naoqi_driver/camera/front/image_raw	Front Camera	robot
/naoqi_driver/camera/stereo/image_raw	Stereo Camera	robot
/naoqi_driver/audio	Microphone	robot
/naoqi_driver/imu/base	IMU Sensor	robot
/naoqi_driver/sonar/front	Sonar Sensor	robot
/naoqi_driver/sonar/back	Sonar Sensor	robot
/naoqi_driver/laser	Laser	robot
/naoqi_driver/odom	Odometry	robot
/joint_states	Joint States	robot
/pepper/sonar_back	Back Sonar	simulator
/pepper/sonar_front	Front Sonar	simulator
/pepper/camera/bottom/image_raw	Bottom Camera	simulator
/pepper/camera/depth/image_raw	Depth Camera	simulator
/pepper/camera/front/image_raw	Front Camera	simulator
/pepper/laser_2	Laser	simulator
/pepper/odom	Odometry	simulator
/joint_states	Joint States	simulator

Topics Published

As explained earlier, the speech topic is listed under the sensor topics because its topic is available under the NAOqi driver sensor topics. The sensorTest node publishes the following topics:

Table 3: Topics published by the sensorTest node.

Topic	Actuator	Platform
/speech	Speakers	robot

Launch File

The launch file is used to launch the sensor test. `sensorTestLaunchRobot.launch` file is used for the robot and `sensorTestLaunchSimulator.launch` for the simulator. The launch file has the following parameters:

- `robot_ip`: specifies the IP address of the robot.
- `roscore_ip`: specifies the IP address of the roscore.
- `network_interface`: specifies the network interface name.

A default value is provided for each parameter in the launch file. For the robot_ip, the default value is `172.29.111.230` and for the network_interface, the default value is `eth0`. The roscore_ip parameter can be specified by the user as identified using `ifconfig` command. But the default value works hence the user can skip specifying the roscore_ip parameter. If the user has a different network_interface name, the user can change the default value specified in the launch file.

Three nodes are launched in the launch file: `naoqi_driver_node`, `naoqiAudio`, and `naoqiAudioPublisher`. The `naoqi_driver_node` is primarily responsible for publishing sensor data from the robot to various topics, excluding audio data. For audio data acquisition, the `naoqiAudio` node is introduced. This node leverages the Python SDK to capture audio data directly from the robot. Due to compatibility issues, as the Python SDK operates under Python 2 which is not directly integrable with ROS-Noetic which runs Python 3. The `naoqiAudioPublisher` node bridges this gap through the use of web sockets, facilitating the transfer of audio data from the SDK to ROS. This node plays a crucial role in processing the audio data by deinterleaving it and then publishing the processed audio to the `/naoqi_driver/audio` topic.

The diagram below shows the data flow of the actuator test.

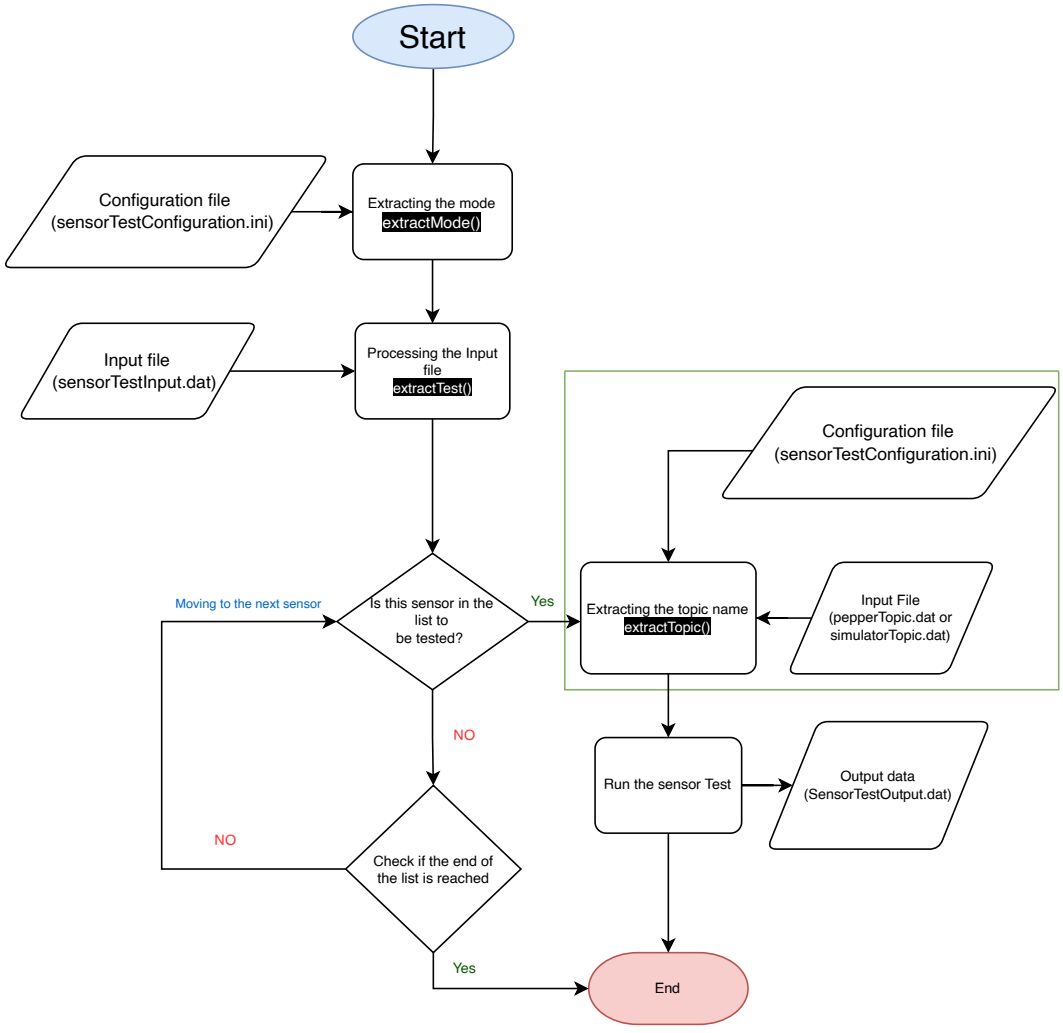


Figure 2: Data Flow Diagram of the Sensor Test.

5 Module Design

For testing the physical robot, the NAOqi driver is used to subscribe to the sensor topics. The driver provides a hardware interface to connect to Aldebaran Nao, Romeo, and Pepper robot. The module first reads the configuration file to get the mode of the test, the platform to be tested, and the topics file for the robot. The module then reads the input file to get the sensors to be tested. The module then reads the topics file to get the topics for the robot. The module then subscribes to the topics for the robot and tests the sensors.

For the sonar sensor, the module subscribes to the sonar topics. The module then verifies that the sonar sensor can capture distance data. The module then logs the distance data to the output data file. For testing the sonar it uses `sensor_msgs/Range` message type. This message type represents a standardized way to express distance measurements typically gathered from range-sensing devices such as sonar, infrared(IR) sensors, and lidar.

For the camera sensors, the module subscribes to the camera topics. The module then verifies that the camera sensor can capture image data. The module then logs the image data to the output data file. For testing the camera sensors, it uses the `sensor_msgs/Image` message type. The video data is recorded and stored in the data directory using the `cv::VideoWriter` class from the OpenCV library.

For the depth camera sensor, the module subscribes to the depth camera topic. The module then verifies that the depth camera sensor can capture image data. The module then logs the image data to the output data file. For testing the depth camera sensor, it uses the `sensor_msgs/Image` message type. The video data is recorded and stored in the data directory using the `cv::VideoWriter` class from the OpenCV library.

For the laser sensor, the module subscribes to the laser topics. The module then verifies that the laser sensor can capture distance data. The module then logs the distance data to the output data file. For testing the laser sensor, it uses the `sensor_msgs/LaserScan` message type. This message type represents a 1D scan from a planar laser range-finder.

For the microphone sensor, the module subscribes to the audio topic. The module then verifies that the microphone sensor can capture audio data. For testing the microphone sensor, it uses a custom message type named `naoqi_driver/AudioCustomMsg`. This message type represents audio data for the four microphones of the Pepper robot. The audio data is recorded and stored in the data directory. The audio is recorded using `.wav` format.

For the joint state sensor, the module subscribes to the joint state topics. The module then verifies that the joint state sensor can capture joint state data. The module then logs the joint state data to the output data file. For testing the joint state sensor, it uses the `sensor_msgs/JointState` message type. This message type represents the state of the robot's joints.

For the odometry sensor, the module subscribes to the odometry topics. The module then verifies that the odometry sensor can capture odometry data. The module then logs the odometry data to the output data file. For testing the odometry sensor, it uses the `nav_msgs/Odometry` message type. This message type represents the odometry data for the robot.

For the IMU sensor, the module subscribes to the IMU topics. The module then verifies that the IMU sensor can capture IMU data. The module then logs the IMU data to the output data file. For testing the IMU sensor, it uses the `sensor_msgs/Imu` message type. This message type represents the IMU data for the robot.

For the speech, the module publishes the text to the speech topic. The module then verifies that the speech can speak out the text. The module then logs the text to the output data file. For testing the speech, it uses the `std_msgs/String` message type that represents a string message.

6 Executing the Sensor Test

To start the sensor test, the user must first install the necessary software packages as outlined in Deliverable D3.3. Furthermore, you can refer to how to get the parameters for the robot IP, roscore IP, and network interface from Deliverable D3.3. Referring to the interface section, the user must set the platform to be tested, and specify which mode to run the test. Then using key-value pairs, the user must specify which sensors to test by using the sensor name as the key and `True` or `False` as the value.

To launch the sensor test, the user must run the following command:

```
# Launch the sensors for the physical robot
roslaunch pepper_interface_tests sensorTestLaunchRobot.launch \
robot_ip:=<robot_ip> roscore_ip:=<roscore_ip> \
network_interface:=<network_interface_name>

# Launch the interfaceTestLaunchSimulator for the simulator
roslaunch pepper_interface_tests interfaceTestLaunchSimulator.launch
```

The above commands will launch the sensor test for the robot and simulator.

It's important to note that the tests for actuators and sensors are designed to function autonomously, allowing for the execution of sensor tests without necessitating the robot's activation. However, when conducting these tests in a sleep state, the robot's camera will orient downwards due to its sitting posture. To circumvent this and achieve a frontal camera perspective, users have the option to first awaken the robot. This can be achieved by launching the actuator file, thereby transitioning the robot to an upright position. Following this, conducting the sensor test will yield a direct camera feed, ensuring optimal test conditions and results.

```
# Run the actuator test
roslaunch pepper_interface_tests actuatorTestLaunchRobot.launch \
robot_ip:=<robot_ip> roscore_ip:=<roscore_ip> \
network_interface:=<network_interface_name>
```

The `sensorTest` node will then run the test and store the result in the output data file. To run the test, the user must run the the following command:

```
# Run the sensor test
rosrun pepper_interface_tests sensorTest
```

6.1 Bottom, Front, and Stereo Camera Test

The tests for the bottom, front, and stereo cameras are designed to evaluate the operational status of these essential sensors. By subscribing to the respective topics for each camera, the test confirms the continuous data publication, ensuring the cameras are functioning correctly. Utilizing the OpenCV library, the tests offer a visual confirmation by displaying images captured from the bottom, front, and stereo cameras. Each image is presented in a dedicated window, labeled with the corresponding sensor's name, allowing a visualization process. These images are displayed for 10 seconds. The test for these cameras will also log the width and height of the image.

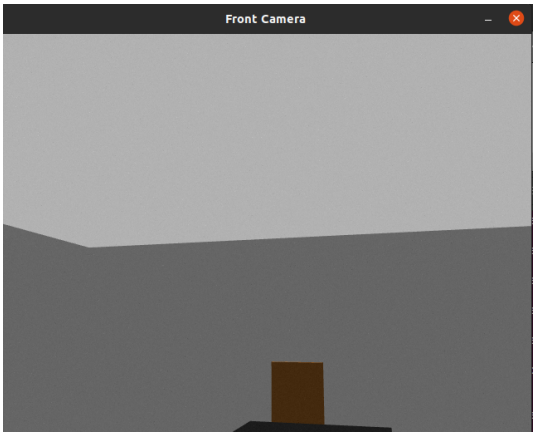
It's important to note the distinction in camera availability between physical and simulator robots. Specifically, the stereo camera is exclusive to the physical robot, with no equivalent functionality in the simulator environment.

To initiate these tests, first, verify that the input file correctly enables the cameras by setting the key-value pairs for the front, bottom, and stereo cameras to `True`. Once confirmed, execute the designated command to launch the sensor tests on the robot.

The subsequent visualization results will provide a clear comparison between the capabilities of both the physical and simulator robots, highlighting the operational status and functionality of the tested cameras. This approach ensures a comprehensive assessment of the camera sensors, catering to both physical and simulated environments.

For more technical detail regarding the 2 RGB camera equipped on the robot, refer to the Pepper 2D camera technical detail. Regarding the stereo camera, refer to the Pepper stereo camera technical detail.

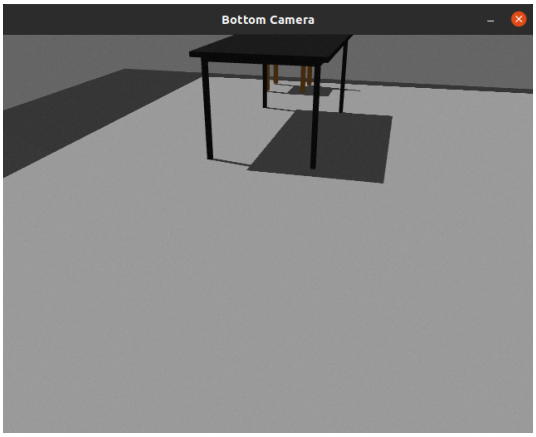
(a) Front camera Image for Physical Robot.



(b) Front camera Image for the Simulator.

Figure 3: Front Camera Image Result.

(a) Bottom camera Image for Physical Robot.



(b) Bottom camera Image for the Simulator.

Figure 4: Bottom Camera Image Result

As seen from the figure 5, the stereo camera is displaying the pair of images captured by the two cameras side by side.

Figure 5: Stereo Camera Image for the Physical Robot.

The robot is equipped with dual cameras: a bottom camera and a front camera, capable of capturing RGB format images that support multiple resolutions: `QQVGA(160x120)`, `QVGA(320x240)`, and `VGA(640x480)`. The selection of the desired resolution can be configured by modifying the file `src/naoqi_driver/share/boot_config.json` settings. You can specify the resolution by using specifying the integer accordingly to the once specified as `"_comment": "QQVGA = 0, QVGA = 1, VGA = 2"`. This allows for flexible adaptation of image resolution based on the requirements of specific tasks or applications. The FPS parameter can be used to adjust the frame rate of the robot but as the number of sensor increase the frame rates that could be achieved decrease. During the testing, the highest frame rate that could be achieved was 1.3 FPS. However, after turning off most of the sensors except the front camera, the odometry, and the joint state, the frame rate increased to 3.0 FPS. Therefore, this still could be limiting for some applications that require high frame rates.

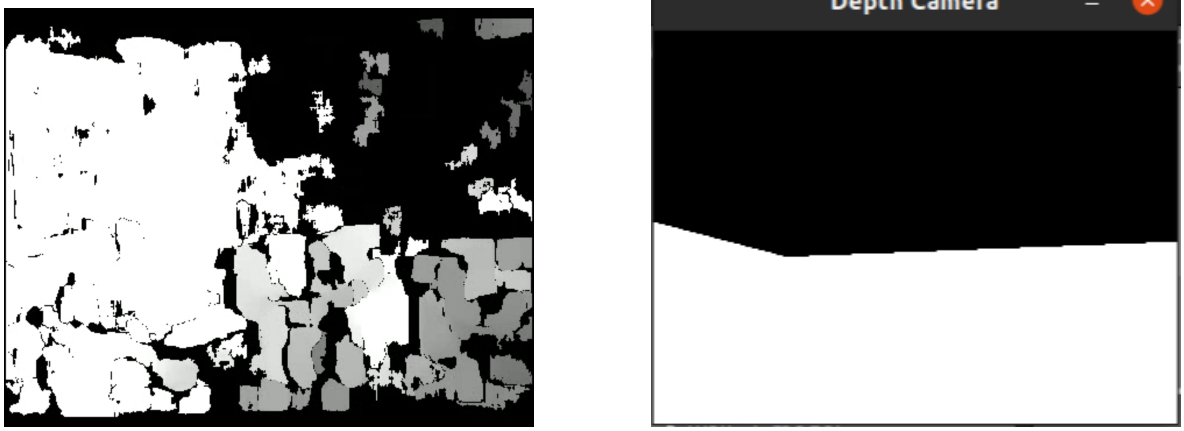
6.2 Depth Camera Test

The depth camera test rigorously assesses the sensor's functionality by subscribing to its topic to confirm data flow and visualizing the output using the OpenCV library. Depth images are displayed in grayscale, with pixel intensity indicating the proximity to the camera, showcased in a window for 10 seconds. The depth sensor, which employs stereo camera technology, demonstrates limited efficacy when used at distances greater than one meter. This limitation significantly impacts tasks such as environment map generation (D5.5.3) that rely on Simultaneous Localization and Mapping (SLAM) techniques. The inherent constraints of the depth camera's calibration process at extended ranges pose challenges to achieving accurate and reliable results. Therefore, it is imperative to explore alternative methodologies to fulfill the requirements of such tasks effectively.

Similarly to test the depth sensor for the physical robot or simulator, the user must set the key-value pair for the depth camera to `True` in the input file. Then execute sensorTest node to run the test.

As mentioned earlier, the depth camera is reconstructed from the stereo camera. For more technical details regarding the depth camera, refer to the Pepper 3D camera technical detail.

The depth image shown in Figure 6a is an image captured by the stereo camera shown in Figure 5 which is converted to a depth image. Similarly, the Depth image for the simulator is shown in Figure 6b. However, for both the physical robot and simulator, the depth image doesn't provide a very clear image that can be used for depth measurement.



(a) Depth Camera Image for Physical Robot.

(b) Depth Camera Image for Simulator.

Figure 6: Depth Camera Result.

6.3 Front and Back Sonar Test

The sonar test is designed to capture and log the sensor's data both on the terminal for immediate observation and into an output file named `sensorTestOutput.dat` for references. During the test, the node will subscribe to the sonar topics for both the front and sonar sensors. To test the sonar sensor, the user must set the key-value pair for the front sonar and back sonar to `True` in the input file. Then execute the sensorTest node to run the test. The sonar sensor is capable of detecting objects within a range. Key metrics such as the range for object proximity is recorded. Upon receiving data, the test outputs a series of messages that detail:

Table 4: Sonar Test Output Data.

Data	Description
`Frame ID`	An identifier for the sensor data frame, ensuring that the data can be accurately correlated with its source and time of capture.
`Field of View`	The angular extent of the observable world that the sonar sensor captures at any given moment.
`Min Range`	The shortest distance at which the sonar sensor can reliably detect objects.
`Max Range`	The furthest distance within which the sensor can effectively perceive objects.
`Range`	The actual distance measured by the sonar sensor to the detected object, providing a directed indication of the object's proximity.

Furthermore, please refer to the Pepper sonar technical detail for more technical detail such as frequency, resolution, vertical, and horizontal field of view.

6.4 Laser Test

The laser test is designed to capture and log the sensor's data both on the terminal for immediate observation and into an output file name `sensorTestOutput.dat` for references. During the test, the node will subscribe to the laser topic and record the range data. To test the laser sensor, the user must set the key-value pair for the laser sensor to `True` in the input file. Then execute the

sensorTest node to run the test. The laser sensor is capable of detecting objects within a range. Upon receiving data, the test outputs a series of messages that detail:

Table 5: Laser Test Output Data.

Data	Description
`Frame ID`	An identifier for the sensor data frame, ensuring that the data can be accurately correlated with its source and time of capture.
`Start and End Angles of the Scan`	The scan covers an angular range field of view that is essential for comprehensive environmental scanning and obstacle detection.
`Angular Distance between Measurements`	This metric reflects the sensor's angular resolution, determining the fineness of the scan and the sensor's ability to discern features.
`Time between Measurements and Scans`	It indicates the time interval between consecutive measurements and scans, influencing the sensor's ability to capture dynamic environmental changes.
`Minimum and Maximum Range Values`	The sensor is capable of detecting objects within a range defining the effective operational distance for reliable data acquisition.
`Range Data`	This array of range data points captures the distances measured across the sensor's field of view, offering a dense dataset for analysis.

For more technical details regarding the laser sensor, refer to the Pepper laser technical detail that shows the frame rate, horizontal and vertical field of view, and the resolution of the laser sensor.

6.5 Microphone Test

The microphone test is designed to capture and play the audio data from the robot's microphone. The test will record the audio data as `.wav` file and play the audio data. For the test, the user can speak or play a sound to the robot's microphone which is located at the top of the robot's head. The test will record the audio data for the four microphones for 10 seconds each as `.wav` file and saves the audio file in the `data` directory to do a system call to play the audio data. The user has the option to keep or delete the audio data. If the user wants to keep the audio data, the user can set the `deleteFile` key to `False` in the implementation file. Then compile the code and run the test. To test the microphone sensor, the user must set the key-value pair for the microphone sensor to `True` in the input file. Then execute the sensorTest node to run the test.

Note:

> The test is only available for the physical robot. The audio starts recording from the rearLeft microphone - rearRight microphone - frontLeft microphone - frontRight microphone. Ensure that the volume for your computer is set to a reasonable level to hear the audio recorded.

The pepper robot has four microphones located at the top of the robot's head as shown in Figure 7.

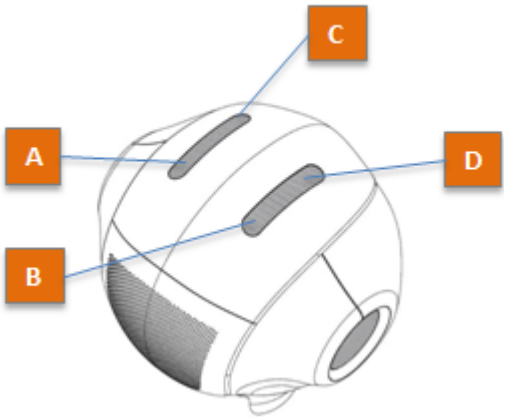


Figure 7: Pepper Microphone. Source: Aldebaran

Table 6: Pepper Microphone Part and Name.

Part	Name
A	`MicroRL_sensor`
B	`MicroRR_sensor`
C	`MicroFL_sensor`
D	`MicroFR_sensor`

The custom message type `naoqi_driver/AudioCustomMsg` records the audio data for the four microphones. The message type will have four data fields for the audio data representing the four microphones.

The microphones are crucial for Deliverable D4.2.3 (Sound Detection and Localization) and Deliverable D4.3.2 (Speech Event) as it is used to perform automatic speech recognition.

For more technical details regarding the microphone sensor, refer to the Pepper microphone technical detail which outlines the location of the four microphones, the frequency range, and the sensitivity of the microphones.

6.6 Joint State Test

The joint state test is designed to assess the functionality and accuracy of the joint state sensors by subscribing to the joint state topic and recording the joint state data. The sensor test will record the status of each joint, including the position, velocity, and effort. The test will output the joint state data to the terminal and save the joint state data to the output data file named `sensorTestOutput.dat`. To test the joint state sensor, the user must set the key-value pair for the joint state sensor to `True` in the input file. Then execute the sensorTest node to run the test.

Upon receiving data, the test outputs a series of messages that detail:

Table 7: Joint State Test Output Data.

Data	Description
`Name`	The name of the joint for which the data is recorded, providing a clear reference for the specific joint being analyzed.
`Position`	The current position of the joint, indicating the angle at which the joint is currently positioned. The position is measured in radians.
`Velocity`	The velocity of the joint, reflecting the rate at which the joint is moving. The velocity is measured in radians per second.
`Effort`	The effort exerted by the joint, representing the force or torque applied to the joint. The effort is measured in Newton-meters.

6.7 Odometry Test

The odometry test is designed to assess the functionality and accuracy of the odometry sensor by subscribing to the odometry topic and recording the odometry data. The sensor test will record the odometry data, including the position, orientation, and linear and angular velocity. The test will output the odometry data to the terminal and save the odometry data to the output data file named `sensorTestOutput.dat`.

To test the odometry sensor, the user must set the key-value pair for the odometry sensor to `True` in the input file. Then execute the sensorTest node to run the test.

Upon receiving data, the test outputs a series of messages that detail:

Table 8: Odometry Test Output Data.

Data	Description
`Timestamp`	The time at which the odometry data was recorded, providing a reference for the data's temporal context.
`Frame of Reference`	The frame in which the odometry data is reported, ensuring that the data is accurately correlated with the robot's position and orientation.
`Child Frame ID`	The ID of the frame that is moving, typically the base frame of the robot.
`Pose`	The robot's position and orientation in the form of a pose message (geometry_msgs/PoseWithCovariance), including both the pose and the covariance of the pose estimate.
`Twist`	The robot's velocity in both linear and angular terms (geometry_msgs/TwistWithCovariance), including the velocity and the covariance of the velocity estimate.

6.8 IMU Sensor Test

The IMU sensor test is designed to assess the functionality and accuracy of the gyroscope and accelerometer sensors by subscribing to the IMU topic and recording the IMU data. The sensor test will record the IMU data, including the linear acceleration, angular velocity, and orientation. The test will output the IMU data to the terminal. Furthermore, you can observe the IMU sensor publishes the data to two topics: `/naoqi_driver/imu/base` and `/naoqi_driver/imu/torso`. Both of the topics publish the IMU sensor data from different frames of reference. For the test, the sensor test will subscribe to the `/naoqi_driver/imu/base` topic.

To test the IMU sensor, the user must set the key-value pair for the IMU sensor to `True` in the input file. Then execute the sensorTest node to run the test.

The test will output the IMU data to the terminal. In addition, upon receiving data, the test outputs a series of messages that detail:

Table 9: IMU Sensor Test Output Data.

Data	Description
`Orientation`	The sensor's current orientation in space, provided as a quaternion (x, y, z, w).
`Angular Velocity`	The rate of rotation around the sensor's x, y, and z axes, typically in radians per second.
`Linear Acceleration`	The acceleration vector along the sensor's x, y, and z axes, excluding gravity, usually in meters per second squared.
`Covariance Matrices`	Estimates the noise and uncertainty of the orientation, angular velocity, and linear acceleration measurements, crucial for algorithms that fuse sensor data for more accurate state estimation.

For more technical details regarding the IMU sensor, refer to the Pepper IMU technical detail.

6.9 Speech Test

The speech test is designed to assess the functionality of speakers by publishing a speech command to the robot. The test will publish the speech in the form of a string message to the speech topic. The text input provided for the speaker is `"This is the CSSR4Africa speaker test."`.

To test the speech, the user must set the key-value pair for the speech to `True` in the input file. Then execute the sensorTest node to run the test.

Note: The test is only available for the physical robot. Ensure that the volume of the robot is set to a reasonable level to hear the speech.

For more technical details regarding the speaker, refer to the Pepper speaker technical detail.

References

[1] Pepper technical specifications. http://doc.aldebaran.com/2-5/family/pepper_technical/index_pep.html.

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.
26 March 2024.