CSSR for

Culturally Sensitive Social Robotics
for Africa

D5.5.4 Robot Navigation

Due date: **26/02/2025**
Submission Date: **26/02/2025**
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: **Birhanu Shimelis Girma**

Revision: **1.0**

Executive Summary

Deliverable D5.5.4 focuses on developing a software module for Robot Navigation, enabling the Pepper robot to autonomously traverse its environment while considering both static and dynamic obstacles. This module integrates path planning algorithms, including Breadth-First Search (BFS), Dijkstra, and A*. Navigation is executed by identifying waypoints along the planned path and controlling the robot's locomotion from waypoint to waypoint.

A key feature of this module is the incorporation of culturally sensitive proxemics, derived from the knowledge base created in Deliverable D1.2 Rwandan Cultural Knowledge. This functionality ensures that the robot maintains appropriate social distances when navigating around humans, enhancing its adaptability and acceptance in human-centered environments.

The deliverable outlines a software development process that includes requirements definition, module specification, implementation, implementation, and unit testing. These phases are comprehensively documented, adhering to the methodologies established in Deliverable D3.2. The navigation module integrates with the robot localization system developed in Deliverable D4.2.4, ensuring path execution within the robot's environment.

This integration is critical for navigation, as the robot continuously updates its pose with real-time data from the robotLocalization node. The requirement definition specifies input parameters control data, and output velocities, ensuring compatibility with the physical robot. All coding activities adhere to the software engineering standards to produce maintainable and reliable code.

The deliverable concludes with unit testing. The test evaluates that the developed module meets its objectives and delivers reliable navigation performance for the Pepper robot.

Contents

1 Introduction

This document describes the development and implementation of the robotNavigation ROS node, a component enabling the Pepper robot to navigate in the environment while considering both static and dynamic obstacles. This module is designed to compute optimal paths, avoid obstacles, and execute smooth locomotion towards the goals, ensuring safe and efficient navigation.

As part of Task 5.5.4, the robotNavigation module integrates fundamental path planning and localization techniques, leveraging algorithms such as Breadth-First Search (BFS), Dijkstra's Algorithm, and A* for trajectory computation. A key feature of this module is its adherence to culturally sensitive proxemics (social distance) , derived from the Rwandan Cultural Knowledge Base (as outlined in Deliverable D1.2 [1]). This ensures that the robot maintains appropriate social distances when interacting with humans, improving its usability in real-world settings. The navigation system operates in physical environment.

This report provides a comprehensive overview of the robotNavigation node, covering module specifications, implementation details, interface design, executing the node and unit testing. The subsequent sections will elaborate on the key functional components of the module, including path planning, and waypoint-based locomotion ensuring clarity in system design and execution.

2 Requirements Definition

The robot navigation module provides the Pepper robot with the ability to navigate autonomously through an environment containing static inanimate obstacles and dynamic obstacles such as humans [2]. This deliverable ensures that the module meets user expectations, enabling the robot to compute the shortest path to a target destination and safely move from waypoint to waypoint. The navigation system operates in various scenarios of physical environment.

The module computes the shortest path from the robot's current position to a specified target position and orientation using Breadth First Search or Dijkstra's algorithm or the A* algorithm. The computed path should include waypoints, which are identified using either equidistant waypoint selection or high path curvature waypoint selection techniques. These methods ensure smooth and efficient navigation along the planned path while considering the robot's physical capabilities and the complexity of the environment.

The module also requires obstacle avoidance mechanisms. It must augment a pre-defined metric workspace map with additional obstacles, including human obstacles detected in the robot's field of view. The size and extent of these human obstacles must adhere to culturally sensitive proxemics, as defined in Deliverable D1.2 African Modes of Social Interaction. This ensures that the robot respects personal space and cultural norms during navigation.

The navigation module is designed to integrate with other systems. It acquires the robot's current pose from the robotLocalization node, and uses the workspace map generated in Deliverable D5.5.3. The module then publishes the planned path and velocity commands to the /cmd_vel topic, enabling the robot's locomotion.

The module operates in two modes, normal mode and verbose mode. In normal mode, it executes navigation without additional logging or visualization. In verbose mode, it logs data published to topics and displays diagnostic information, such as the configuration space map and planned path, in an OpenCV window. This dual-mode functionality allows for detailed debugging and analysis during development and testing.

To initiate navigation, the module accepts a goal pose specified by the coordinates x, y and an orientation θ in the workspace frame of reference. Once the goal is received, the module calculates the required path and outputs a sequence of forward and angular velocities, which are published to the /cmd_vel topic. Additionally, the configuration space map and the planned path are rendered graphically for analysis.

Operationally, the module is compatible with both the physical Pepper robot and a simulated environment using the Pepper simulator. It reads platform-specific parameters, such as sensor and actuator topics, from a configuration file named robotNavigationConfiguration.ini. This ensures that the module can be easily adapted to different platforms without altering its core logic.

From a non-functional perspective, the module is designed to compute the shortest path to ensure real-time navigation. The robot must reach its target position within a predefined tolerance ($\pm 0.1\,\mathrm{m}$ for position and $\pm 5°$ for orientation). The module must also demonstrate a high success rate (e.g., $\geq 95\%$) in avoiding obstacles during navigation. To ensure maintainability, the code adheres to the software engineering standards outlined in Deliverable D3.2, and internal documentation clearly describes the module's functionalities, parameters, and interfaces.

In its intended use, the module must navigate static environments using a pre-defined workspace map. The robot must also dynamically adjust its behavior when humans or other obstacles enter its path, ensuring safe and efficient navigation.

3   Module Specification

The Robot Navigation module is designed to facilitate autonomous navigation for the Pepper robot by computing paths, avoiding obstacles, and ensuring cultural proxemics are respected. This module integrates path planning algorithms such as Breadth-First Search (BFS), Dijkstra, and A* to determine the optimal trajectory between waypoints while maintaining real-time awareness of the environment.

The navigation process follows a structured approach: First, the module computes the shortest path from the robot's current position to the specified goal (x, y, θ) using a selected algorithm. Once the path is determined, it is discretized into intermediate waypoints to facilitate smooth navigation. The system checks its configuration space to account for static obstacles, ensuring safe movement. Finally, the robot executes the planned trajectory by adjusting its velocity and rotation commands, allowing it to reach the target destination efficiently.

The configuration space map is used to define accessible areas within the robot's environment. This map is essential for identifying free space, avoiding obstacles, and computing feasible navigation paths. The workspace is represented as an occupancy grid, where white regions indicate navigable areas, and darker regions correspond to obstacles or restricted zones. A sample configuration space map is shown in Figure 1.
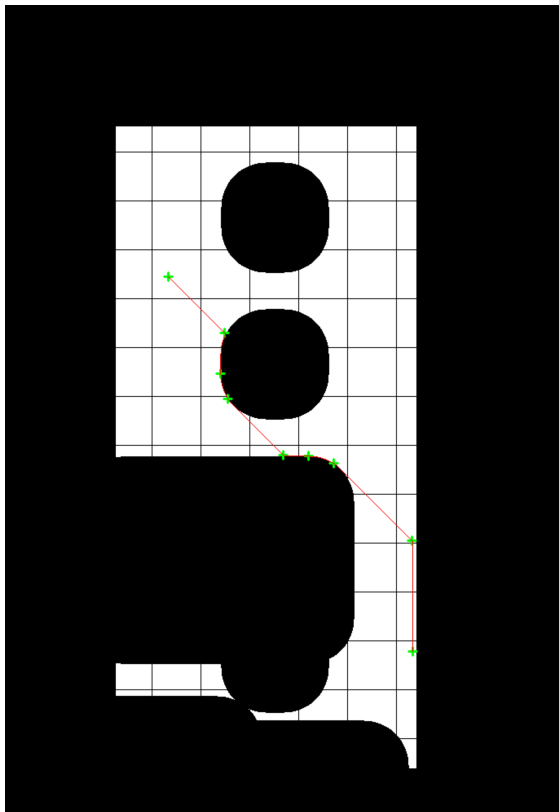
Figure 1: The configuration space map highlights areas that are navigable and restricted, ensuring safe path planning.
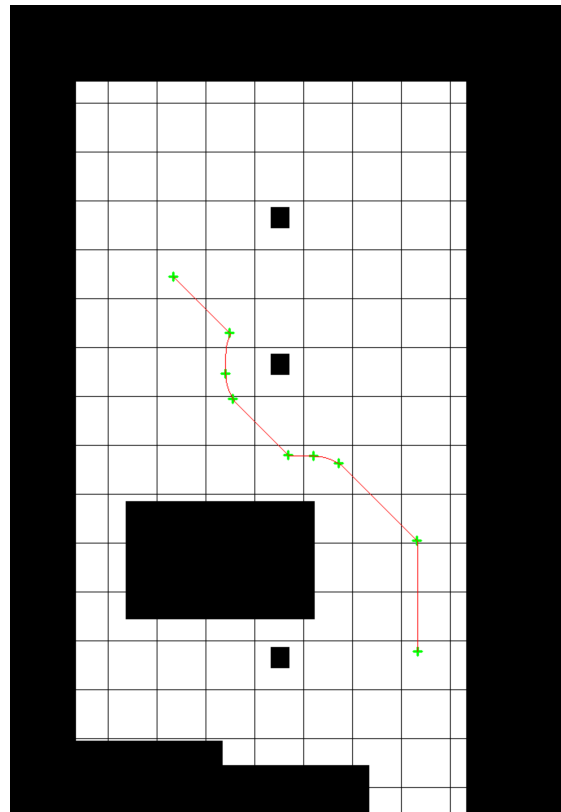
Figure 2: Visualization of the waypoint-based navigation approach. The robot follows the computed waypoints to reach its goal.

To ensure efficient navigation, the module breaks the computed path into discrete waypoints. Each waypoint serves as an intermediate target, guiding the robot smoothly toward its final destination. The robot continuously updates its position and recalculates waypoints when necessary. The map of the waypoints,

depicted in Figure 2, illustrates a sample navigation path with sequential waypoints.

The navigation module supports static map navigation, where the robot operates on a predefined map with known obstacles. To execute a navigation task, the module requires an input, which is a goal pose (x, y, θ) provided through a ROS service request. The processing phase involves path computation, waypoint selection, and obstacle detection to ensure safe movement. Finally, the output consists of a sequence of velocity commands that actuate the robot's wheels, guiding it toward the desired destination.

The robot receives sensory data from the robotLocalization node and adjusts its navigation accordingly. The final execution involves publishing velocity commands to the `/cmd_vel` topic, enabling movement toward the desired goal while adhering to safety constraints.

4 Implementation

4.1 File Organization

The source code for executing the robot navigation functionality is structured into three primary components: robotNavigationApplication, robotNavigationImplementation, and robotNavigationInterface. The robotNavigationImplementation component encapsulates the core functionality required for robot navigation, including tasks such as path planning, obstacle avoidance, and waypoint management. It supports various algorithms, including BFS, Dijkstra, and A*, and integrates features for cultural proxemics constraints, such as maintaining social distances. This component also processes critical files such as configuration files, map data, and topic definitions to enable smooth navigation.

The robotNavigationApplication serves as the entry point for the navigation node, managing the execution of functions defined in the implementation and interface components. It initializes the ROS node, sets up parameters, and orchestrates the navigation operations. The robotNavigationInterface defines the abstract layer and function declarations that facilitate communication between the application and implementation layers, ensuring modularity and consistency in the codebase. This file structure promotes clean separation of concerns, making the navigation system easier to maintain, extend, and debug while supporting scalable integration with other system components.

The file structure of the robot navigation node in the `cssr_system` package is organized as follows:

```
cssr_system
└── robotNavigation
    ├── config
    │   └── robotNavigationConfiguration.ini
    ├── data
    │   ├── robotTopics.dat
    │   ├── simulatorTopics.dat
    │   └── navigationOutput.dat
    ├── include
    │   └── robot_navigation
    │       ├── robotNavigationInterface.h
    │       └── moveTo.h
    ├── launch
    │   ├── robotNavigationLaunchRobot.launch
    │   └── robotNavigationLaunchSimulator.launch
    ├── msg
    │   └── Goal.msg
    ├── src
    │   ├── robotNavigationApplication.cpp
    │   └── robotNavigationImplementation.cpp
    ├── srv
    │   └── set_goal.srv
    ├── README.md
    ├── CMakeLists.txt
    └── package.xml
```

4.2 Configuration Parameters

The operation of the `robotNavigation` node is determined by the contents of a configuration file, `robotNavigationConfiguration.ini` that contain a list of key-value pairs as shown below in Table 1 [3].

Table 1: Configuration Parameters for robotNavigation node.

Key	Values	Effect
map	scenarioOneMap.dat	Specifies the workspace map file.
pathPlanning	BFS, Dijkstra, A*	Specifies the algorithm for path planning.
socialDistance	true, false	Enables/disables cultural proxemics constraints.
robotTopics	pepperTopics.dat	Sensor/actuator topics for the physical robot.
simulatorTopics	simulatorTopics.dat	Sensor/actuator topics for the simulator.
verboseMode	true, false	Enables verbose output with terminal logs and OpenCV displays.

Input File

There is no specific input data file required for the robot navigation node. The navigation goals are processed dynamically based on service or action requests received from client nodes.

Output File

There is no output data file generated by the robot navigation node. Instead, the outcome of the navigation operation, such as successful goal completion or failure details, is communicated back to the invoking client through the respective response mechanism. Additionally, diagnostic and status messages are logged to the console, with the verbosity controlled by the verboseMode key in the configuration file. This ensures that navigation operations are both responsive to real-time inputs and informative for debugging or monitoring purposes.

Topics File

For the robot navigation node, a curated list of topics for the robot is maintained in dedicated topics files. These file is stored in the .dat format and contain key-value pairs where each key represents the name of an actuator, and the corresponding value specifies the associated topic. The topics file for the robot is named robotTopics.dat. This file ensures proper communication and data flow between the node and its respective hardware, providing a structured approach to topic management and facilitating operation in different execution contexts.

Topics Subscribed

This node subscribes to one topic, published by the robotLocalization node, which provides the pose of the robot, as summarized in Table 2.

Table 2: Topic Subscribed to by the robotNavigation node.

Topic	Node	Platform
/robotLocalization/pose	robotLocalization	Physical robot

Topics Published

The robotNavigation node publishes velocity commands to control the robot's movement, as summarized in Table 3.

Services Supported

This node provides and advertizes a server for a service /robotNavigation/set_goal to request navigation to a given goal position and orientation. It uses a custom message to specify the pose with the x

Table 3: Topics Published by the robotNavigation node.

Topic	Actuator	Platform
/cmd_vel	WheelFL, WheelFR, WheelB	Physical robot

and y coordinates, and the angle of rotation θ about the z axis. If the navigation request is successful, the service response is "1"; if it is unsuccessful, it is "0". The service is called by the behaviorController node, as summarized in Table 4.

Table 4: Services Provided and Called.

Service	Message Value	Effect
/robotNavigation/set_goal	$< x >< y >< \theta >$	Define navigation goal pose.

Services Called

This node calls the following two services, as summarized in Table 5.

Table 5: Services Provided and Called.

Service	Message Value	Effect
/knowledgeBase/query	To be defined	Extract required knowledge from the cultural knowledge base
/robotLocalization/reset_pose	reset	Reset the pose of the robot using absolute localization

The type of variable that is passed as an argument to the /knowledgeBase/query service has not yet been defined. This will be done when the node that services and advertizes these services are fully specified. Similarly, the type of service call return value has not yet been defined. Again, this will be done when the node that services and advertizes these services are fully specified

5 Executing Robot Navigation

The `robotNavigation` node is responsible for controlling the autonomous movement of the robot within a predefined environment. It receives target destinations, computes optimal paths, and ensures obstacle avoidance while navigating towards the goal. This section outlines the detailed steps required to configure, launch, and test the `robotNavigation` node.

5.1 Environment Setup

Before executing the node, ensure the following dependencies are installed and configured correctly.

Install all necessary dependencies by navigating to the workspace and running:

```
cd ~/workspace/pepper_rob_ws
```

```
rosdep install --from-paths src --ignore-src -r -y
```

Clone the CSSR4Africa repository into the robot's workspace (if not already cloned):

```
cd ~/workspace/pepper_rob_ws/src
```

```
git clone https://github.com/cssr4africa/cssr4africa.git
```
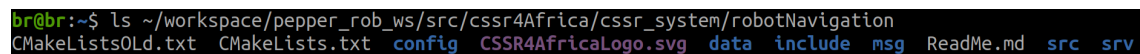
Build the package and source the environment:

```
cd ~/workspace/pepper_rob_ws
```

```
catkin_make
```

```
source devel/setup.bash
```

Verify the package is correctly placed in the workspace:

```
ls ~/workspace/pepper_rob_ws/src/cssr4Africa/cssr_system/robotNavigation
```


Figure 3: Screenshot of expected output showing the files and folders in the robotNavigation node from the executed command

Configure the Node

The `robotNavigation` node is configured using pre-defined map and setting files. The configuration parameters are set in the `robotNavigationConfiguration.ini` listed in Table 1. If necessary, update the configuration values in the configuration file before starting the node.

Starting the Node

> **NOTE**
>
> Ensure that the robot localization node is running. If not, execute the following command:
>
> ```
> rosrun cssr_system robotLocalization
> ```

To start the `robotNavigation` Node, execute the following command:

```
rosrun cssr_system robotNavigation
```

This will initialize the navigation system and display status messages indicating the node running.

```
br@br:~$ rosrun cssr_system robotNavigation
[ INFO] [1740552240.236375253]: Robot Navigation Goal Server Ready

Environment Map File: scenarioOneEnvironmentMap.dat
Configuration Map File: scenarioOneConfigMap.dat
Path Planning Algorithm: A*
Social Distance Mode: true
Robot Topics: pepperTopics.dat
Topics Filename:
Debug Mode: true
Subscribing to odom
POSITION_TOLERANCE:        0.30
POSITION_TOLERANCE_GOAL:   0.20
ANGLE_TOLERANCE_ORIENTING: 0.05
ANGLE_TOLERANCE_GOING:     0.20
POSITION_GAIN_DQ:          0.50
ANGLE_GAIN_DQ:             0.50
POSITION_GAIN_MIMO:        0.35
ANGLE_GAIN_MIMO:           0.30
MIN_LINEAR_VELOCITY:       0.03
MAX_LINEAR_VELOCITY:       0.40
MIN_ANGULAR_VELOCITY:      0.05
MAX_ANGULAR_VELOCITY:      1.00
CLEARANCE:                 0.30
SHORTEST_PATH_ALGORITHM    0
ROBOT_AVAILABLE            1
[ INFO] [1740552242.963732286]: Robot Navigation Node Running...
[ INFO] [1740552252.963739308]: Robot Navigation Node Running...
```

Figure 4: Screenshot of the output of running the robotNavigation node.

5.2 Interacting with the Node

To check the robot's current pose, execute the following command.

```
rostopic echo /robotNavigation/pose
```

This will output the robot's current position and orientation.

Resetting the Pose

To manually reset the robot's pose, use the following service call:

```
rosservice call /robotLocalization/reset_pose <x_position> <y_position> <theta
    >
```

```
br@br:~/workspace/pepper_rob_ws$ rosservice call /robotLocalization/reset_pose 2.0 6.6 0.0
success: True
```

Figure 5: Screenshot of the rosservice call to reset the position responds a success message of true.

Sending a Goal

To command the robot to navigate to a specific position, use:

```
rosservice call /robotNavigation/set_goal goal_x goal_y goal_theta
```

```
Example:
rosservice call /robotNavigation/set_goal 2.0 6.6 0.0
```

If the navigation is successful, a confirmation message will be displayed.

```
br@br:~/workspace/pepper_rob_ws$ rosservice call /robotNavigation/set_goal 3.0 6.6 0.0
navigation_goal_success: 1
```

Figure 6: Screenshot of the rosservice call to set goal for navigation responds a success message of 1.

6 Unit Test

Testing is a crucial step in the software development lifecycle, ensuring that each component of the system performs as expected. The `robotNavigation` node requires validation to confirm that path planning, goal execution, and obstacle avoidance work reliably under various conditions. Unit testing is employed to assess these functionalities independently, following the structured testing guidelines outlined in Deliverable D3.5 [4], System Integration and Quality Assurance. The `robotNavigation` node comprises several key operations, including path planning, waypoint generation, pose retrieval, and velocity command execution. To ensure that these components function correctly, a unit test is developed using the GoogleTest framework. These tests isolate functions and verify their expected behavior under different scenarios.

6.1 File Organization and Its Purposes

```
unit_tests
└── robotNavigationTest
    ├── config
    │   └── robotNavigationTestConfiguration.ini
    ├── data
    │   ├── navigationUnitTestLogs.log
    │   └── test_goals.dat
    ├── include
    │   └── robotNavigationTest
    │       └── robotNavigationTestInterface.h
    ├── launch
    │   └── robotNavigationTestLaunch.launch
    ├── msg
    │   └── Goal.msg
    ├── src
    │   ├── robotNavigationDriver.cpp
    │   ├── robotNavigationTestApplication.cpp
    │   └── robotNavigationTestImplementation.cpp
    ├── srv
    │   ├── NavigationTest.srv
    │   └── set_goal.srv
    ├── CSSR4AfricaLogo.svg
    ├── CMakeLists.txt
    └── README.md
├── CMakeLists.txt
└── package.xml
```

The `robotNavigationTest` folder is structured to support unit testing of the Robot Navigation unit test Node. The `src` directory contains `robotNavigationDriver.cpp`, which serves as the driver for test execution, `robotNavigationTestApplication.cpp`, which manages the test flow, and `robotNavigationTestImplementation.cpp`, which provides the core test logic.

The `config` directory houses the `robotNavigationTestConfiguration.ini` file, which defines test parameters such as goal positions and navigation constraints, allowing customization of test

cases. The `data` folder contains `test_goals.dat`, which stores predefined navigation goals for the tests, and `navigationUnitTestLogs.log`, which captures the test execution logs, recording timestamps and goal completion statuses for debugging and validation.

6.2 Test Environment Setup

Before executing the robot navigation units tests, the testing environment must be properly configured.

1. Install all necessary dependencies:

```
cd ~/workspace/pepper_rob_ws
rosdep install --from-paths src --ignore-src -r -y
```

2. Clone the CSSR4Africa repository into the workspace (if not already cloned):

```
cd ~/workspace/pepper_rob_ws/src
```

```
git clone https://github.com/cssr4africa/cssr4africa.git
```

3. Build the workspace and source the environment:

```
cd ~/workspace/pepper_rob_ws
```

```
catkin_make
```

```
source devel/setup.bash
```

The test execution is controlled using the configuration file `robotNavigationTestConfiguration.ini`, located in the `config` directory of the unit test package. This file defines the test parameters, including position tolerance, angle tolerance, and social distance.

6.3 Test Cases

The test cases evaluate the ability of the `robotNavigation` node to process navigation requests and reach predefined goals.

1. SetGoalServiceAvailable: Verifies that the `/robotNavigation/set_goal` service is available before sending navigation commands.

2. SendNavigationGoalFromFile: Reads a list of predefined goals from the `test_goals.dat` file and sends them sequentially. The robot's response is validated against expected success values.

6.4 Executing the Robot Navigation Unit Test

To start the unit test, use the following command:

```
cd ~/workspace/pepper_rob_ws
```

```
rosrun unit_tests robotNavigationTest
```

Upon execution, the test sequence is initiated, and results are logged in the `navigationUnitTestLogs.log` file.

```
br@br:~$ rosrun unit_tests robotNavigationTest
[ INFO] [1740559980.973002174]: Program: Robot Navigation unit Test Application
[ INFO] [1740559980.973066109]: This project is funded by the African Engineering and Technology Network (Afretec)
[ INFO] [1740559980.973082442]: Inclusive Digital Transformation Research Grant Programme.
[ INFO] [1740559980.973102174]: Website: www.cssr4africa.org
[ INFO] [1740559980.973116078]: This program comes with ABSOLUTELY NO WARRANTY.
[ INFO] [1740559980.973129147]: robotNavigation: start-up.
[ INFO] [1740559980.973141184]: robotNavigation: subscribed to /robotLocalization/pose
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from RobotNavigationTest
[ RUN      ] RobotNavigationTest.SetGoalServiceAvailable
[       OK ] RobotNavigationTest.SetGoalServiceAvailable (302 ms)
[ RUN      ] RobotNavigationTest.RunNavigationTests
ERROR: transport error completing service call: unable to receive data from sender, check sender's logs for details
[       OK ] RobotNavigationTest.RunNavigationTests (81027 ms)
[----------] 2 tests from RobotNavigationTest (81329 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (81329 ms total)
[  PASSED  ] 2 tests.
```

Figure 7: Screenshot of the unit test command running.

6.5 Test Results

The results of the unit tests are categorized based on success and failure conditions. If a test case passes, the corresponding success message is logged. Otherwise, an error message is generated with relevant debugging details. The test reports are attached in Appendix I.

Appendix I: Unit Test Logs

```
[2025-02-26 10:08:10] ================================================================================
[2025-02-26 10:08:10] === New Robot Navigation Test Run Started at 2025-02-26 10:08:10 ===
[2025-02-26 10:08:10] ================================================================================
[2025-02-26 10:08:10] Initializing Robot Navigation Node: PASSED
[2025-02-26 10:08:10] --------------------------------------------------------------------------------
[2025-02-26 10:08:10] Test Case 1: Goal to (5.0, 6.6, 0.0)
[2025-02-26 10:08:10] Sending goal...
[2025-02-26 10:08:31] [SUCCESS] Goal reached: (5.000000, 6.600000, 0.000000)
[2025-02-26 10:08:31] Goal execution successful: PASSED
[2025-02-26 10:08:31] --------------------------------------------------------------------------------
[2025-02-26 10:08:31] Test Case 2: Goal to (5.0, 2.0, 270.0)
[2025-02-26 10:08:31] Sending goal...
[2025-02-26 10:09:03] [SUCCESS] Goal reached: (5.000000, 2.000000, 270.000000)
[2025-02-26 10:09:03] Goal execution successful: PASSED
[2025-02-26 10:09:03] --------------------------------------------------------------------------------
[2025-02-26 10:09:03] Test Case 3: Goal to (5.0, 6.6, 90.0)
[2025-02-26 10:09:03] Sending goal...
[2025-02-26 10:09:53] [SUCCESS] Goal reached: (5.000000, 6.600000, 90.000000)
[2025-02-26 10:09:53] Goal execution successful: PASSED
[2025-02-26 10:09:53] --------------------------------------------------------------------------------
[2025-02-26 10:09:53] Test Case 4: Goal to (2.0, 6.6, 0.0)
[2025-02-26 10:09:53] Sending goal...
[2025-02-26 10:10:19] [SUCCESS] Goal reached: (2.000000, 6.600000, 0.000000)
[2025-02-26 10:10:19] Goal execution successful: PASSED
[2025-02-26 10:10:19] --------------------------------------------------------------------------------
[2025-02-26 10:10:19] ================================================================================
[2025-02-26 10:10:19] === Robot Navigation Test Run Completed at 2025-02-26 10:10:19 ===
[2025-02-26 10:10:19] ================================================================================

[2025-02-26 10:53:01] ================================================================================
[2025-02-26 10:53:01] === New Robot Navigation Test Run Started at 2025-02-26 10:53:01 ===
[2025-02-26 10:53:01] ================================================================================
[2025-02-26 10:53:01] Initializing Robot Navigation Node: PASSED
[2025-02-26 10:53:01] --------------------------------------------------------------------------------
[2025-02-26 10:53:01] Test Case 1: Goal to (3.2, 6.6, 0.0)
[2025-02-26 10:53:01] Sending goal...
```

```
[2025-02-26 10:53:10] [SUCCESS] Goal reached: (3.200000, 6.600000, 0.000000)
[2025-02-26 10:53:10] Goal execution successful: PASSED
[2025-02-26 10:53:10] ------------------------------------------------------------------------
[2025-02-26 10:53:10] Test Case 2: Goal to (3.2, 6.0, 270.0)
[2025-02-26 10:53:10] Sending goal...
[2025-02-26 10:53:11] [SUCCESS] Goal reached: (3.200000, 6.000000, 270.000000)
[2025-02-26 10:53:11] Goal execution failed: FAILED
[2025-02-26 10:53:11] ------------------------------------------------------------------------
[2025-02-26 10:53:11] Test Case 3: Goal to (2.0, 6.0, 180.0)
[2025-02-26 10:53:11] Sending goal...
[2025-02-26 10:53:28] [SUCCESS] Goal reached: (2.000000, 6.000000, 180.000000)
[2025-02-26 10:53:28] Goal execution successful: PASSED
[2025-02-26 10:53:28] ------------------------------------------------------------------------
[2025-02-26 10:53:28] Test Case 4: Goal to (2.0, 6.6, 0.0)
[2025-02-26 10:53:28] Sending goal...
[2025-02-26 10:54:22] [SUCCESS] Goal reached: (2.000000, 6.600000, 0.000000)
[2025-02-26 10:54:22] Goal execution failed: FAILED
[2025-02-26 10:54:22] ------------------------------------------------------------------------
[2025-02-26 10:54:22] ========================================================================
[2025-02-26 10:54:22] === Robot Navigation Test Run Completed at 2025-02-26 10:54:22 ===
[2025-02-26 10:54:22] ========================================================================
```

References

[1] https://cssr4africa.github.io/deliverables/CSSR4africa_deliverable_d1.2.pdf.

[2] Work plan | culturally sensitive social robotics for africa.

[3] https://cssr4africa.github.io/deliverables/CSSR4africa_deliverable_d3.2.pdf.

[4] https://cssr4africa.github.io/deliverables/CSSR4africa_deliverable_d3.5.pdf.

Principal Contributors

The main authors of this deliverable are as follows:
Birhanu Shimelis Girma, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Birhanu Shimelis Girma.
17 December 2024.