CSSR for

Culturally Sensitive Social Robotics for Africa

D5.5.3 Environment Map Generation

Due date: **21/03/2025**
Submission Date: **04/04/2025**
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**

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 D5.5.3 Environment Map Generation focuses on the development of a software module that generates metric workspace and configuration space maps of the environments used in the project's two use case scenarios. These maps include non-symbolic metric data, enabling the Pepper robot to perform path planning and navigate during human-robot interactions. The deliverable focuses primarily on map generation from a priori CAD data.

The software module, implemented as a ROS node named `mapGeneration`, produces a map that captures the physical layout of environments using pre-defined geometric data to construct the map. This method generates non-symbolic metric maps visualizable as an image. The module also generates configuration space maps through image dilation, providing essential data for robot navigation planning.

The development process followed a structured approach that included requirement definition, module specification, implementation, and unit testing. Each phase adhered to the software engineering standards established in Deliverable D3.2, ensuring maintainability and reliability. The configurabiliy of the module through the `mapGenerationConfiguration.ini` file facilitates its operation in different environments.

Unit testing confirmed the module's functionality in CAD mode, with tests executed on the physical robot. The tests verified the accuracy of map generation, and the effectiveness of configuration space computation. Overall, this deliverable provides a flexible mapping solution that integrates seamlessly with other components of the system architecture.

Contents

1 Introduction

This deliverable represents the output of Task 5.5.3, which aims to develop a software module for environment map generation. The module enables the Pepper robot to navigate and make meaningful deictic gestures by providing map that includes physical layout information.

Environment mapping is a fundamental capability for autonomous robots operating in human-centered spaces. In the context of the CSSR4Africa project, the map serves dual purposes: allowing physical navigation through spaces and supporting meaningful human-robot interaction through references to objects in the environment. The map generation module bridges these requirements by producing a map with metric precision.

The module supports map generation through a CAD-based approach, which uses a priori data, typically from Computer-Aided Design (CAD) files, to construct the map. The advantage of this method is that it can produce detailed maps without the need for the robot to physically explore the environment. It relies on structured geometric descriptions of the environment to generate workspace and configuration space maps. A workspace map representing the physical environment and a configuration space map derived through image dilation that accounts for the robot's physical dimensions.

The implementation of the `mapGeneration` ROS node adheres to the software engineering standards documented in Deliverable D3.2. It integrates with other system components, like robot localization, robot navigation, and particularly the robot mission interpreter, which relies on the generated maps for mission execution. The module's development followed a systematic software engineering process, ensuring maintainability, and integration with the broader system architecture.

This document details each phase of the development process, from requirements definition through to unit testing, providing a comprehensive overview of the module's functionality, design decisions, and implementation details. The subsequent sections elaborate on the technical aspects of the mapping approach, and the integration with other system components.

2 Requirements Definition

The environment map generation module fulfills the functional needs of the users and meets the requirements outlined in the work plan. This section defines the specific requirements that guided the development of the module.

The module must support map generation using a CAD-based approach with a priori environment data. The generated maps include non-symbolic metric data visualizable as images, and configuration space maps derived through image dilation.

The configuration space generation functionality converts workspace maps to configuration space maps, applies image dilation using a structuring element modeling the robot's base, and ensures the configuration space accurately represents robot navigability. Integration capabilities include publishing map data on appropriate ROS topics, supporting data access by other modules in the system, and maintaining compatibility with the Robot Mission Interpreter.

The module supports both the normal operation mode and verbose mode for diagnostics and debugging. In terms of platform compatibility, the module works with both the physical Pepper robot and the Pepper simulator.

Non-functional requirements include performance specifications, where the module generates maps with sufficient resolution for navigation (typically 0.05-0.1 meters per pixel). Reliability requirements ensure the module generates consistent maps across multiple executions with the same input data, handles edge cases such as complex geometries and overlapping objects, and recovers gracefully from errors in input data.

The module provides an intuitive interface for map visualization, includes comprehensive error messages for troubleshooting, and supports configurability through parameters. It follows the coding standards outlined in Deliverable D3.2, includes comprehensive internal documentation, and implements modular design for future extensions. Compatibility requirements ensure the module generates maps in formats compatible with ROS navigation stack and supports standard message types for map data.

3 Module Specification

The module specification defines the functional capabilities of the environment map generation module, detailing the input to output data transformations, expected input data, output formats, and configuration parameters, as well as the algorithms and data structures used for map generation.

3.1 Functional Overview

The `mapGeneration` module transforms input data from CAD files into a structured representation of the environment with metric information. This transformation includes processing input data to generate a workspace map, applying image dilation to create a configuration space map, saving maps and object data to a file.

3.2 Input Data Specification

The module accepts input parameters including map dimensions (width and height in millimeters), obstacle file (containing geometric descriptions of obstacles), parameters file (containing environment parameters), map resolution (meters per pixel), and output filenames for workspace map, and configuration space map.

3.3 Algorithms and Data Structures

The CAD-based map generation uses a geometric processing algorithm that parses geometric primitives from the input file, converts to occupancy grid representation, and uses rasterization algorithms for different geometric shapes (Bresenham's [1] line algorithm for rectangle edges, flood-fill for interior regions, circle rasterization for circular objects, and polygon filling for complex shapes).

The configuration space generation uses an image dilation algorithm that converts the occupancy grid to a binary image, defines a structuring element based on robot dimensions (circular element for omnidirectional base, size determined by `robotRadius` parameter), applies morphological dilation operation, and converts the dilated image back to an occupancy grid. The implementation uses OpenCV for efficient image processing, applies the `cv::dilate` function with appropriate kernel, and handles edge conditions at map boundaries.

The module is structured with primary classes including `MapGenerator` (main class that orchestrates the mapping process), `CADMapBuilder` (implements CAD-based map generation), and `ConfigSpaceGenerator` (generates configuration space from the workspace map).

4 Implementation

This section details the implementation of the environment map generation module, including file organization, configuration, and core functionality.

File Organization

The file structure of the map generation module in the `cssr_system` package is organized as follows:

```
cssr_system
└── mapGeneration
    ├── config
    │   └── mapGenerationConfiguration.ini
    ├── data
    │   ├── obstacles.txt
    │   ├── parameters.txt
    │   └── mapGenerationInput.txt
    ├── include
    │   └── mapGeneration
    │       └── mapGeneration.h
    ├── launch
    │   └── mapGenerationLaunchRobot.launch
    ├── src
    │   ├── mapGenerationApplication.cpp
    │   └── mapGenerationImplementation.cpp
    ├── README.md
    └── CMakeLists.txt
```

Configuration Parameters

The operation of the `mapGeneration` node is determined by the contents of a configuration file, `mapGenerationConfiguration.ini`, that contains a list of key-value pairs as shown below in Table 1.

Table 1: Configuration Parameters for mapGeneration node.

Key	Values	Effect
`mode`	`CAD, SLAM`	Specifies the map generation approach
`verboseMode`	`true, false`	Enables/disables diagnostic output
`resolution`	decimal value	Specifies map resolution in meters per pixel
`robotRadius`	decimal value	Specifies the robot base radius
`inputFile`	string	The input file name

Input Files

For CAD-based map generation, the module requires `mapGenerationInput.txt` (containing basic parameters including map dimensions, filenames, and resolution), `obstacles.txt` (defining geometric primitives such as rectangles).

The input files include:
`mapGenerationInput.txt`:

```
680 993                      # Map dimensions in millimeters (width, height)
obstacles.txt                # File containing obstacle definitions
parameters.txt               # File containing environment parameters
environmentMap.png           # Output filename for workspace map
configurationSpaceMap.png    # Output filename for configuration space map
```

`obstacles.txt`:

```
OBSTACLE   2.5775 3.12  2.315 1.44    # Format: TYPE X Y WIDTH HEIGHT
OBSTACLE   1.3  0.45  2.60 0.90
OBSTACLE   2.20  0.30  4.40 0.60
OBSTACLE   3.31  1.925  0.22 0.25
```

Output File

The workspace map represents the physical environment in PNG image format and as a ROS OccupancyGrid map representation of free/occupied space, and saved to a file as specified in `mapGenerationInput.txt`.

The configuration space map accounts for robot dimensions, also in PNG image format and as a ROS OccupancyGrid map representation of the robot navigable space, and saved to a file as specified in `mapGenerationInput.txt`.

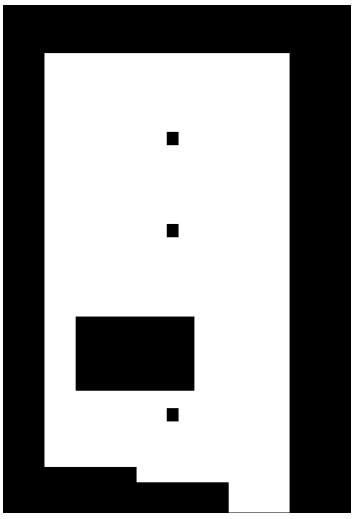

Figure 1: A workspace map representing the physical environment in PNG image format.

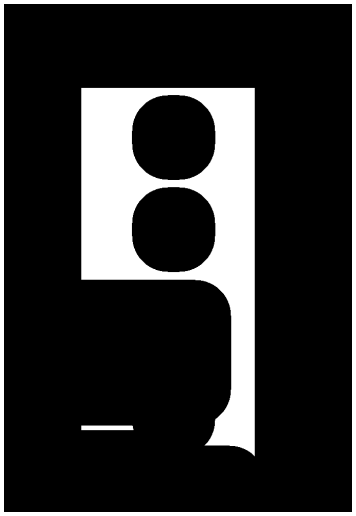

Figure 2: A configuration space map accounting for robot dimensions, also in PNG image format and an OccupancyGrid map representation of the robot navigable space.

5 Executing Environment Map Generation

This section provides a comprehensive guide on setting up, configuring, and executing the `mapGeneration` node on the physical Pepper robot.

5.1 Environment Setup

Before executing the node, several setup steps must be completed to ensure all dependencies are installed and configured correctly. First, all necessary dependencies are installed by navigating to the workspace and running:

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

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

- If the CSSR4Africa repository has not been cloned yet, this must be done by:

  ```
  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/mapGeneration
  ```

```
br@br:~$ ls ~/workspace/pepper_rob_ws/src/cssr4africa/cssr_system/mapGeneration
CMakeLists.txt  config  data  include  README.md  src
```

Figure 3: Expected output showing the files and folders in the mapGeneration node

5.2 Interacting with the Node

Starting the Node

To start the `mapGeneration` node directly, the following command is used:

```
rosrun cssr_system mapGeneration
```

```
br@br:~$ rosrun cssr_system mapGeneration
[ INFO] [1743595066.716507027]: /mapGeneration: v1.0
                                This project is funded by the African Engineering and Technology Network (Afretec)
                                Inclusive Digital Transformation Research Grant Programme.
                                Website: www.cssr4africa.org
                                This program comes with ABSOLUTELY NO WARRANTY.
[ INFO] [1743595066.718297319]: /mapGeneration: startup.
Package directory: /home/br/workspace/pepper_rob_ws/src/cssr4africa/cssr_system
Configuration parameters:
mode: CAD
verboseMode: false
inputFile: mapGenerationInput.txt
resolution: 0.050000
robotRadius: 0.300000
Successfully saved environment map to /home/br/workspace/pepper_rob_ws/src/cssr4africa/cssr_system/mapGeneration/data/environmentMap.png
Successfully saved navigation map to /home/br/workspace/pepper_rob_ws/src/cssr4africa/cssr_system/mapGeneration/data/configurationSpaceMap.png
Map generation completed successfully.
Press any key to terminate the program ...
```

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

Verifying the Results

After map generation is complete, the results should be verified by checking that the output files have been created in the specified location `cssr_system/mapGeneration/data`, inspecting the workspace map and configuration space map for accuracy.

6 Unit Test

Unit testing verifies that the module meets its specifications and functions correctly in various scenarios. This section details the testing approach, setup, and results for the environment map generation node.

It's important to note that the mapGeneration node has been designed to operate independently of the physical robot hardware. Users can generate environment maps and configuration space maps without needing to connect to a Pepper robot. Since the CAD-based map generation relies solely on geometric data provided in the input files, no sensor data or robot connectivity is required. This makes the node useful for pre-planning environments before deploying the robot, and enables development and testing on any computer with ROS installed.

6.1 File Organization and Its Purposes

The unit tests for the map generation module are organized in the following directory structure:

```
unit_tests
├── mapGenerationTest
│   ├── config
│   │   └── mapGenerationTestConfiguration.ini
│   ├── data
│   │   ├── test_obstacles.txt
│   │   ├── test_parameters.txt
│   │   ├── test_input.txt
│   │   └── test_output.logs
│   ├── include
│   │   └── mapGenerationTest
│   │       └── mapGenerationInterfaceTest.h
│   ├── launch
│   │   └── mapGenerationLaunchTestHarness.launch
│   ├── src
│   │   ├── mapGenerationTestDriver.cpp
│   │   ├── mapGenerationTestApplication.cpp
│   │   └── mapGenerationTestImplementation.cpp
│   ├── CMakeLists.txt
│   └── README.md
├── CMakeLists.txt
└── package.xml
```

The purpose of each file in this structure is to support testing of the map generation node. The configuration file sets test parameters, the data file provides input for the tests, the header file defines interfaces for testing, the source files implement the test cases, and the launch files enable easy execution of tests.

6.2 Test Environment Setup

The unit tests are designed to validate CAD-based map generation functionality. Before executing the tests, the testing environment must be properly configured. All necessary dependencies must be installed, the CSSR4Africa repository must be cloned into the workspace if not already present, and the workspace must be built and sourced.

6.3 Test Cases

The unit tests are designed to validate specific functionality of the map generation node across different scenarios. Each test case focuses on different aspects of the map generation process:

Test Case 1: Basic CAD Map Generation (TestCADMapGeneration) This test verifies that the node correctly generates a workspace map from a basic test obstacles file. It checks for proper parsing of obstacle

data from input files, correct rendering of rectangular obstacles on the map, correct map dimensions based on input specifications, and Proper boundary handling.

Expected Output: A workspace map showing rectangular obstacles as specified in the test_obstacles.txt file. The resulting map should match the reference image shown in Figure 5.

Test Case 2: Configuration Space Generation (TestConfigSpaceGeneration) This test validates the dilation algorithm used to create the configuration space. It tests the generation of configuration spaces with varying robot radii (0.1m, 0.2m, 0.3m, 0.5m, and 0.8m), correct application of structuring elements based on robot dimensions, progressive increase in obstacle area as robot radius increases, and consistent handling of map boundaries during dilation.

Expected Output: Three configuration space maps with progressively larger dilated obstacles, corresponding to the different robot radii. Figure 6,7, and 8 shows the comparison between the workspace map and a configuration space map with at a different robot radius.

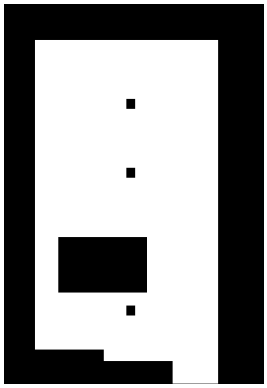

Figure 5: Workspace map showing the obstacles from TestCADMapGeneration

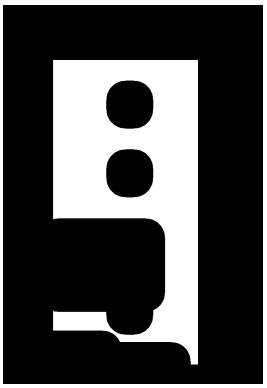

Figure 6: Configuration space map with 0.1m robot radius

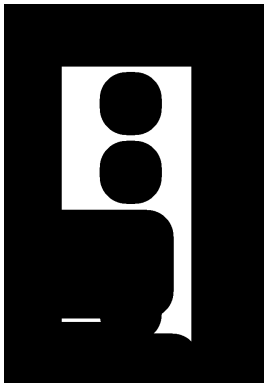

Figure 7: Configuration space map with 0.3m robot radius

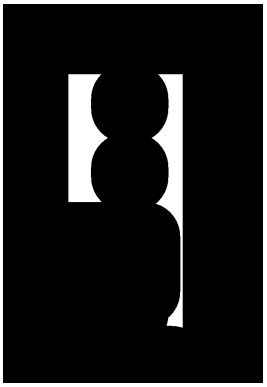

Figure 8: Configuration space map with 0.5m robot radius

Each test case generates both workspace and configuration space maps, which can be found in the test output directory after test execution. The test logs (shown in Appendix I) provide detailed information about each test's execution and verification steps.

6.4 Executing the Map Generation Unit Test

To run the unit tests, the provided launch file is used:

```
roslaunch unit_tests mapGenerationLaunchTestHarness.launch
```

Alternatively, individual tests can be run directly:

```
rosrun unit_tests mapGenerationTest
```

6.5 Test Results

The results of the unit tests confirm that the map generation node meets all specified requirements for CAD-based mapping. The node successfully generates workspace maps, and computes configuration spaces. Performance is within expected parameters, ensuring the module can operate efficiently in real-world conditions.

All test cases (TestCADMapGeneration, and TestConfigSpaceGeneration) have passed, indicating that the node functions correctly according to its specifications. Detailed test logs are included in Appendix I, providing a comprehensive record of all test executions.

Appendix I: Unit Test Logs

```
[2025-03-31 08:27:36] ========================================================================
[2025-03-31 08:27:36] === New Map Generation Test Run Started at 2025-03-31 08:27:36 ===
[2025-03-31 08:27:36] ========================================================================
[2025-03-31 08:27:36] Initializing Map Generation Node: PASSED
[2025-03-31 08:27:36] ------------------------------------------------------------------------
[2025-03-31 08:27:36] Test Case: MapGenerationTest.EmptyMapCreation
[2025-03-31 08:27:36] Loaded radius1: 0.100000m
[2025-03-31 08:27:36] Loaded radius2: 0.200000m
[2025-03-31 08:27:36] Loaded radius3: 0.300000m
[2025-03-31 08:27:36] Loaded radius4: 0.500000m
[2025-03-31 08:27:36] Loaded radius5: 0.800000m
[2025-03-31 08:27:36] Loading test configuration file...
[2025-03-31 08:27:36] Generating workspace map...
[2025-03-31 08:27:36] [SUCCESS] Workspace map generated successfully
[2025-03-31 08:27:38] Verifying map dimensions (680x993): PASSED
[2025-03-31 08:27:38] Verifying obstacle placement: PASSED
[2025-03-31 08:27:38] ------------------------------------------------------------------------
[2025-03-31 08:27:38] Test Case: MapGenerationTest.MapWithObstacles
[2025-03-31 08:27:38] Loaded radius1: 0.100000m
[2025-03-31 08:27:38] Loaded radius2: 0.200000m
[2025-03-31 08:27:38] Loaded radius3: 0.300000m
[2025-03-31 08:27:38] Loaded radius4: 0.500000m
[2025-03-31 08:27:38] Loaded radius5: 0.800000m
[2025-03-31 08:27:38] Loading complex obstacles file...
[2025-03-31 08:27:38] Generating workspace map...
[2025-03-31 08:27:38] [SUCCESS] Workspace map generated successfully
[2025-03-31 08:27:40] Verifying complex obstacle placement: PASSED
[2025-03-31 08:27:40] ------------------------------------------------------------------------
[2025-03-31 08:27:40] Test Case: MapGenerationTest.ConfigurationSpaceGeneration
[2025-03-31 08:27:40] Loaded radius1: 0.100000m
[2025-03-31 08:27:40] Loaded radius2: 0.200000m
[2025-03-31 08:27:40] Loaded radius3: 0.300000m
[2025-03-31 08:27:40] Loaded radius4: 0.500000m
[2025-03-31 08:27:40] Loaded radius5: 0.800000m
[2025-03-31 08:27:40] Loading test workspace map...
[2025-03-31 08:27:40] Generating configuration spaces with different robot radii...
```

```
[2025-03-31 08:27:40] Testing with radius 0.100000m...
[2025-03-31 08:27:41] [SUCCESS] Configuration space generated for radius 0.100000m
[2025-03-31 08:27:41] Testing with radius 0.200000m...
[2025-03-31 08:27:43] [SUCCESS] Configuration space generated for radius 0.200000m
[2025-03-31 08:27:43] Testing with radius 0.300000m...
[2025-03-31 08:27:44] [SUCCESS] Configuration space generated for radius 0.300000m
[2025-03-31 08:27:44] Testing with radius 0.500000m...
[2025-03-31 08:27:45] [SUCCESS] Configuration space generated for radius 0.500000m
[2025-03-31 08:27:45] Testing with radius 0.800000m...
[2025-03-31 08:27:47] [SUCCESS] Configuration space generated for radius 0.800000m
[2025-03-31 08:27:47] Comparing radius 0.100000m vs 0.200000m: 501404 vs 538581 obstacle pixels: PASSED
[2025-03-31 08:27:47] Comparing radius 0.200000m vs 0.300000m: 538581 vs 573427 obstacle pixels: PASSED
[2025-03-31 08:27:47] Comparing radius 0.300000m vs 0.500000m: 573427 vs 631380 obstacle pixels: PASSED
[2025-03-31 08:27:47] Comparing radius 0.500000m vs 0.800000m: 631380 vs 673789 obstacle pixels: PASSED
[2025-03-31 08:27:47] Comparing dilation results: PASSED
[2025-03-31 08:27:47] Verifying dilation effect: PASSED
[2025-03-31 08:27:47] ------------------------------------------------------------------------
[2025-03-31 08:27:47] ========================================================================
[2025-03-31 08:27:47] === Map Generation Test Run (CAD Mode) Completed at 2025-03-31 08:27:47 ===
[2025-03-31 08:27:47] === All tests PASSED ===
[2025-03-31 08:27:47] ========================================================================
```

References

[1] J. E. Bresenham. Algorithm for computer control of a digital plotter. 4(1):25–30. Conference Name: IBM Systems Journal.

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.
04 April 2025.