

System Verification and Validation Plan for 2D Localizer

Aliyah Jimoh

April 17, 2025

Revision History

Date	Version	Notes
24/02/2025	1.0	Initial Draft
03/03/2025	1.1	Feedback Implementation

Contents

1	Symbols, Abbreviations, and Acronyms	iv
2	General Information	1
2.1	Summary	1
2.2	Objectives	1
2.3	Challenge Level and Extras	1
2.4	Relevant Documentation	1
3	Plan	2
3.1	Verification and Validation Team	2
3.2	SRS Verification Plan	2
3.3	Design Verification Plan	3
3.4	Verification and Validation Plan Verification Plan	3
3.5	Implementation Verification Plan	3
3.6	Automated Testing and Verification Tools	4
3.6.1	Code Guidelines	4
3.6.2	GTSAM Integration	4
3.7	Software Validation Plan	4
4	System Tests	5
4.1	Tests for Functional Requirements	5
4.1.1	Sensor Estimation Tests	5
4.1.2	Input Testing	7
4.1.3	Visual Testing	8
4.2	Tests for Nonfunctional Requirements	9
4.2.1	Estimation Accuracy	9
4.3	Traceability Between Test Cases and Requirements	11
5	Unit Test Description	11
5.1	Unit Testing Scope	11
5.2	Tests for Functional Requirements	11
5.2.1	Module 3: Input Format	11
5.2.2	Module 2	13
5.3	Tests for Nonfunctional Requirements	13
5.3.1	Module ?	13
5.3.2	Module ?	14

5.4	Traceability Between Test Cases and Modules	14
6	Appendix	15
6.1	Symbolic Parameters	15
6.2	Usability Survey Questions?	15

List of Tables

1	Verification and Validation Team	2
2	Traceability Matrix Showing the Connections Between Re- quirements and System Tests	11

List of Figures

1	Example Factor Graph from Dellaert (2012)	4
[Remove this section if it isn't needed —SS]		

1 Symbols, Abbreviations, and Acronyms

Symbol	Description
2D	Two-Dimensional
2D Localizer	2D Localization Solution
CRLB	Cramér-Rao Lower Bound
FM	Fiducial Marker
GTSAM	Georgia Tech Smoothing and Mapping
MG	Module Guide
MIS	Module Interface Specification
MLE	Maximum Likelihood Estimation
PEP8	Python Enhancement Proposal 8
SE(2)	Special Euclidean Group in 2D
SRS	Software Requirements Specification
T	Test
VnV	Verification and Validation

For a full list of symbols, abbreviations, and acronyms used, refer to section 1 in the [SRS](#) document.

This document shows the verification and validation plan of the 2D Localizer program. This plan starts with general information that talks about 2D Localizer in section 2. The plan and system tests involved with this software is explained in sections 3 and 4.

2 General Information

2.1 Summary

This document examines the verification and validation plan of 2D Localizer. This software is used to help accurately locate mobile robots in a provided 2D map given the measurements and coordinates of the sensors and fiducial markers (FMs).

2.2 Objectives

The objective of this plan is to validate the accuracy of this program to build confidence in the software's reliability and performance. This plan also aims to satisfy all the requirements the System Requirements Specification ([SRS](#)) document outlined.

2.3 Challenge Level and Extras

This system has an advanced research level which can be seen from the implementation and the topic. Setting up the robot's movement, accurately displaying the trajectory, coordinating the sensors' measurements, and finding a way to animate the output would definitely add difficulty to this system, however, this is not a niche topic meaning that there are papers or libraries available to draw inspiration from.

2.4 Relevant Documentation

The documentation relevant to the 2D Localizer includes the Problem Statement since it explains the proposed software, the [SRS](#) which talks about the requirements needed to properly use the system, the Verification and Validation ([VnV](#)) report that goes through the tests and plans for the system, and the Module Guide ([MG](#)) and Module Interface Specification ([MIS](#)) for the design.

3 Plan

This section describes the tests made for the 2D Localizer system. This begins by mentioning the VnV team in section 3.1, then followed by the SRS Verification Plan in 3.2, the Design Verification Plan in section 3.3, the VnV Plan Verification Plan in section 3.4, the Implementation Verification Plan in section 3.5, the Automated Testing and Verification tools in section 3.6, and the Software Validation Plan in section 3.7.

3.1 Verification and Validation Team

This section shows the members of the VnV team. They are shown in Table 1 along with what document they contributed in and their roles.

Name	Document	Role
Aliyah Jimoh	All	Author
Dr. Spencer Smith	All	Instructor Reviewer
Kiran Singh	SRS VnV	Domain Expert Reviewer
Dr. Matthew Giamou	Problem Statement	Supervisor Reviewer

Table 1: Verification and Validation Team

3.2 SRS Verification Plan

The SRS document for 2D Localizer will be verified through the following steps:

1. The initial review will be performed by the assigned reviewers (Dr. Spencer Smith and Kiran Singh) and will use the [SRS Checklist](#) as a guide.
2. The reviewers will give feedback through creating issues in the GitHub repository.

3. The author (Aliyah Jimoh) will apply the feedback to the document and address issues that may not be applied.
4. The author will discuss with the supervisor (Dr. Matthew Giamou) and ask for their input.

3.3 Design Verification Plan

The design for 2D Localizer will be verified through the following steps:

1. The design documents, the MG and MIS, will be reviewed by the assigned reviewers (Kiran Singh and Dr. Spencer Smith) and will use both checklists ([MG Checklist](#) and [MIS Checklist](#)) as a guide.
2. The reviewers will give feedback through creating issues in the GitHub repository.
3. The author (Aliyah Jimoh) will apply the feedback to the document and address issues that may not be applied.

3.4 Verification and Validation Plan Verification Plan

The VnV plan for 2D Localizer will be verified through the following steps:

1. The VnV plan will be reviewed by the assigned reviewers (Kiran Singh and Dr. Spencer Smith) and will use the [VnV Checklist](#) as a guide.
2. The reviewers will give feedback through creating issues in the GitHub repository.
3. The author (Aliyah Jimoh) will apply the feedback to the document and address issues that may not be applied.

3.5 Implementation Verification Plan

The implementation of 2D Localizer will be verified by the following techniques:

- **Static Verification**

The author will perform a code walkthrough the domain expert (Kiran Singh) and go through different tests to check the accuracy. The author

will then do a user test case in a presentation while explaining the tests made and performed. Section 4 describes system-level tests derived from SRS functional requirements that are implemented through code in Section 5 and validated through dynamic testing.

- **Dynamic Testing**

The unit tests for 2D Localizer are discussed in section 5

3.6 Automated Testing and Verification Tools

3.6.1 Code Guidelines

For Python implementation, the 2D Localization software will follow the coding style guidelines outlined in Python Enhancement Proposal 8 (PEP8). To enforce this, the Flake8 linter will be used to ensure that code meets PEP8’s guidelines and for continuous integration in GitHub Actions.

3.6.2 GTSAM Integration

Georgia Tech Smoothing and Mapping ([GTSAM](#)) by [Dellaert and Contributors \(2022\)](#) is a library that implements sensor fusion for robotics using factor graphs. These graphs consist of two types of nodes: variables (e.g., robot poses and sensor measurements) and factors, which represent probabilistic constraints between them. 2D Localizer will be using this library as a modelling language and would need to run unit tests to verify if GTSAM is properly integrated.

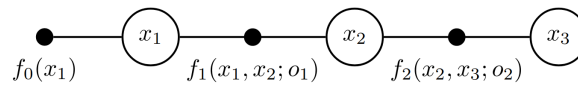


Figure 1: Example Factor Graph from [Dellaert \(2012\)](#)

3.7 Software Validation Plan

To help validate the accuracy of the localization algorithm, 2D Localizer will use 2D data sets to test. If a data set cannot be found, alternative test cases

and feedback will be discussed with the supervisor.

4 System Tests

This section goes over the different system tests 2D Localizer will go through to satisfy its requirements.

4.1 Tests for Functional Requirements

This subsection looks at the system tests that will help satisfy the functional requirements outlined in the SRS (Section 5.1)

4.1.1 Sensor Estimation Tests

Range-Only Pose Estimation

1. F-RO-01

Requirements Addressed: R3, R5

Control: Automatic

Initial State:

- Known beacon positions
- Robot starts at unknown location

Input: A set of noisy range measurements $\tilde{\mathbf{d}}_j$ with each set containing the number of beacons that whose range detected it. The map size is given along with the initial estimate position of the robot. All inputs are derived from a predefined YAML configuration file.

Output: The estimated robot pose $\hat{\mathbf{x}}$ represented by (x, y, θ) , corresponding to the robot's position and heading in the environment.

Test Case Derivation: SRS Section IM2

How test will be performed: The tester will run the program with a valid input YAML file containing beacon positions, trajectory data, and noise configuration. The system will estimate a pose using only range measurements. The test is considered a pass if:

- A valid pose estimate is returned (finite values)
- The pose lies within the bounds of the map dimensions

SE(2) Transformation Estimation

1. F-SE2-01

Requirements Addressed: R2, R5

Control: Automatic

Initial State:

- Robot starts with a known pose T_{wr}
- Fiducial markers are placed at fixed positions T_{wf}

Input: Transformation matrices T_{wf}, T_{fr}

Output: Computed transformation must satisfy:

$$T_{wr} = T_{wf}T_{fr}$$

Test Case Derivation: SRS Section IM3

How test will be performed: The tester will configure an input YAML file where the robot has line-of-sight to fiducial markers (FMs). The system will use this to refine its pose estimate. The test passes if:

- A pose is returned without error
- Estimated heading shows consistent improvement compared to range-only results
- The result remains finite and within bounds

Range + SE(2) Sensor Fusion Test

1. F-SF-01

Requirements Addressed: R5, R6

Control: Automatic

Initial State:

- Robot starts at unknown pose
- Beacons and fiducial markers placed in test environment

Input: Range measurements + SE(2) transformation constraints

Output: Estimated pose T_{wr} and \hat{x}

Test Case Derivation: F-RO-01 and F-SE2-01

How test will be performed: The tester will provide input files with both fiducial and range data, and compare the results against a previous run using only range data. The test passes if:

- The fused system produces a pose estimate without failure
- Accuracy improves or remains consistent as measured by the CRLB matrix.

4.1.2 Input Testing

Input Validation

1. F-IN-01

Requirements Addressed: R4

Control: Automatic

Initial State:

- Uninitialized or corrupted input files.

Input: Malformed YAML or invalid numeric data in CSV files.

Output: System-generated error message or exception

How test will be performed: The tester will intentionally modify the input YAML and/or CSV files to introduce syntax errors, negative or NaN values, or structurally invalid fields. The system should raise an error and refuse to proceed. The test passes if:

- An informative error is shown
- The system halts without crashing or proceeding with incorrect data

4.1.3 Visual Testing

2D Map Overlay of Estimated Poses

1. F-MO-01

Requirements Addressed: R1, R2, R7, R9

Control: Automatic

Initial State:

- Environment 2D map and sensor locations (beacons, fiducials) are known.
- Robot follows a predefined trajectory.

Input:

- Ground truth trajectory T_{wr}
- Estimated trajectory from localization output \hat{x} , generated by a GTSAM-based factor graph optimization.
- Sensor positions and detections.

Output: The visualization correctly displays:

- Estimated trajectory \hat{x} based on GTSAM's factor graph optimization.
- Sensor placements (beacons, fiducial markers).

Test Case Derivation: The estimated pose \hat{x} is computed using Maximum Likelihood Estimation (MLE), as described in SRS Section IM2.

How test will be performed: The tester will launch the visualization after localization completes. The test passes if:

- The map is displayed without error
- All landmarks are plotted correctly
- The trajectory follows the expected path smoothly
- The robot's position updates dynamically.

Real-Time Pose Display

1. F-RT-01

Requirements Addressed: R8

Control: Manual

Initial State: Live localization process is running

Input: Pose estimates that are generated over time

Output: Updated GUI table showing (x, y, θ) at each position.

How test will be performed: The tester will observe the GUI during runtime. The test passes if:

- The table updates with each new pose
- Updates occur at consistent intervals
- No freezes or lag are observed

4.2 Tests for Nonfunctional Requirements

4.2.1 Estimation Accuracy

Accuracy Validation

1. NF-ACC-01

Requirements Addressed: R6, NFR1

Type: Nonfunctional, Dynamic, Automatic

Initial State: Final pose estimate has been computed

Input/Condition:

- Sensor data (real or simulated).
- Estimated robot poses \hat{x} , generated using a GTSAM-based factor graph optimization.
- Ground truth trajectory.

Output/Result:

- The Root Mean Squared Error (RMSE) of the estimated poses \hat{x} must satisfy the accuracy constraint in SRS Section 5.2, Requirement NFR1.

- The estimated pose covariance must not exceed the theoretical Cramér-Rao Lower Bound (CRLB) (SRS Section TM2).

How test will be performed: The tester will run the system with a test configuration containing known ground truth data and noise-defined sensor measurements. The system will estimate poses using factor graph optimization.

- RMSE satisfies the required accuracy threshold
- The CRLB matrix is finite and does not exceed the expected theoretical bound.

4.3 Traceability Between Test Cases and Requirements

	F-RO-01	F-SE2-01	F-SF-01	F-MO-01	NF-ACC-01
R1	X	X	X	X	
R2	X		X		X
R3	X		X		
R4					
R5				X	

Table 2: Traceability Matrix Showing the Connections Between Requirements and System Tests

5 Unit Test Description

This section describes how the unit tests verify software components in 2D Localizer as described in the MIS. Unit testing is conducted dynamically using the pytest framework and is focused on verifying that each function behaves correctly under normal and invalid conditions. Each test corresponds to one or more functional or nonfunctional requirements described in the SRS.

All test cases can be found in the [test](#) directory of the repository, with descriptive names and in-code docstrings referencing their Test IDs.

5.1 Unit Testing Scope

The focus of unit testing is on modules M3 (Input Format), M6 (Localization), M8 (Plotting), and M9 (Accuracy Evaluation).

Modules like M2 (GTSAM) and M7 (Control Module) are not explicitly tested at the unit level since they act primarily as libraries or orchestration logic.

5.2 Tests for Functional Requirements

5.2.1 Module 3: Input Format

Module 3 handles parsing and validation of user-provided inputs, including YAML files, sensor positions, and measurement arrays. Tests confirm input

correctness, formatting, and rejection of invalid files using both black-box and white-box strategies.

1. `test_invalid_yaml` (from `test_input_file.py`)

Type: Functional, Dynamic, Automatic

Initial State: User provides a malformed input file (e.g., YAML with syntax error or NaNs in CSV)

Input: Invalid YAML or malformed CSV file

Output: Raised exception or error message

Test Case Derivation: From R4 - the system must reject malformed files

How test will be performed: Automatic test using PyTest

2. `test_map_image` (from `test_inputs.py`)

Type: Functional, Dynamic, Automatic

Initial State: User provides a YAML file with a valid map path

Input: Valid YAML file referencing a .png or .jpg map

Output: File existence confirmed

Test Case Derivation: From R4 - the system must reject malformed files

How test will be performed: Automatic test using PyTest

3. `test_coordinates` (from `test_inputs.py`)

Type: Functional, Dynamic, Automatic

Initial State: YAML and CSV files include sensor coordinates

Input: 2D coordinate arrays

Output: Parsed as float arrays of shape (N, 2)

Test Case Derivation: From R2

How test will be performed: Automatic test using PyTest

4. test_range_measurements (from test_inputs.py)

Type: Functional, Dynamic, Automatic

Initial State: Range matrix and trajectory arrays provided

Input: 2D range array ($T \times B$), trajectory array ($T \times 3$)

Output: Dimensions and type validity confirmed

Test Case Derivation: From R3

How test will be performed: Automatic test using PyTest

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. 2012.

URL <https://api.semanticscholar.org/CorpusID:131215724>.

Frank Dellaert and GTSAM Contributors. borglab/gtsam, May 2022. URL

<https://github.com/borglab/gtsam>).

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]