Module Guide for 2D Localizer

Aliyah Jimoh

April 16, 2025

1 Revision History

Date	Version	Notes
2025/03/19	1.0	Initial Draft
2025/04/18	1.1	Implement Feedback

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
2D Localizer	2D Localization Solution
AC	Anticipated Change
CLI	Command-Line Interface
DAG	Directed Acyclic Graph
GTSAM	Georgia Tech Smoothing and Mapping
GUI	Graphical User Interface
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in scientific computing (SC), where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format of the initial input data.

AC3: The integration of different sensors.

AC4: The format of the visualization method.

AC5: The types of sensor data used during simulation and testing.

AC6: The evaluation methods for accuracy and uncertainty estimation.

AC7: The format for user interaction.

AC8: The accuracy of the visualization method.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: This system will always be designed for 2D localization.

UC3: The Georgia Tech Smoothing and Mapping (GTSAM) library will be implemented as a modelling language.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: GTSAM Module

M3: Input Format Module

M4: Simulation Module

M5: Output Module

M6: Localization Module

M7: Control Module

M8: Plotting Module

M9: Accuracy Evaluation Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Simulation Module Output Module Localization Module Control Module Accuracy Evaluation Module
Software Decision Module	GTSAM Module Plotting Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *2D Localizer* means the module will be implemented by the 2D Localizer software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Input Format Module (M3)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: 2D Localizer

Type of Module: Abstract Data Type

7.2.2 Simulation Module (M4)

Secrets: The simulated variables of the sensor measurements.

Services: Computes the beacon and camera based on the coordinates given.

Implemented By: 2D Localizer

Type of Module: Abstract Object

7.2.3 Output Module (M5)

Secrets: The display table of the estimated pose (output).

Services: Displays and updates the estimated pose on a table.

Implemented By: Tkinter

Type of Module: Abstract Object

7.2.4 Localization Module (M6)

Secrets: The mathematical models used to calculate the estimated pose.

Services: Computes estimated position based on the sensor fusion measurements

Implemented By: GTSAM

Type of Module: Abstract Object

7.2.5 Control Module (M7)

Secrets: The system's order in running modules and how users provide input.

Services: Handles the user interaction and manages the execution of the other modules.

Implemented By: 2D Localizer

Type of Module: Abstract Object

7.2.6 Plotting Module (M8)

Secrets: The visualization technique used to display the localization data.

Services: Generates visual representation of pose estimate with the beacon and fiducial marker coordinates on the map.

Implemented By: Matplotlib

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 GTSAM Module (M2)

Secrets: The Georgia Tech Smoothing and Mapping (GTSAM) library by Dellaert and Contributors (2022)

Services: Creates a wrapper for the GTSAM library to be used in 2D Localizer

Implemented By: 2D Localizer

Type of Module: Library

7.3.2 Accuracy Evaluation Module (M9)

Secrets: The accuracy analysis of the pose estimate.

Services: Assesses accuracy through computing the Fisher Information Matrix(FIM) and evaluating the Cramér-Rao Lower Bound (CRLB).

Implemented By: 2D Localizer

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M3, M7
R2	M1, M3, M7
R3	M1, M3, M7
R4	M3
R5	M5, M6
R6	M9
R7	M5, M8
R8	M5, M8
R9	M8

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M3
AC3	M6
AC4	M5
AC5	M??
AC6	M9
AC7	M7
AC8	M8

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

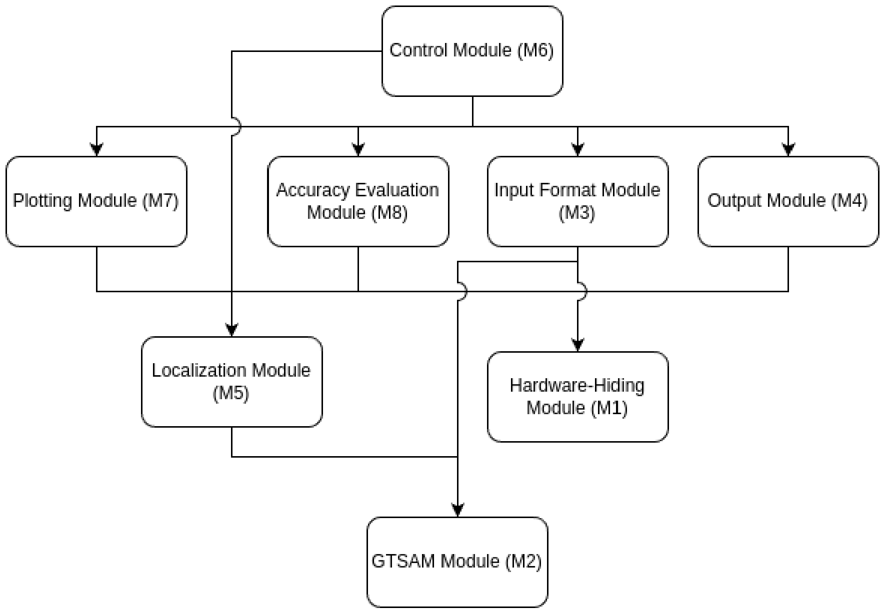

Figure 1: Use Hierarchy Among Modules

10 User Interfaces

10.1 Graphical User Interface (GUI)

The users will have their results shown on a table with the Output Module (M5) and a visual plot with the Plotting Module (M8).

10.2 Command-Line Interface (CLI)

2D Localizer provides a simple CLI for users to insert their variables in the system. This is done by having them modify user_input.yaml which is a file that is set up to be read by the Input Format Module (M3) in the Control Module (M7). To run the system, users type './run.sh' which executes M7.

11 Timeline

Table 4 shows the schedule of tasks needed to be completed by the Verification and Validation (VnV) team members referred to in Section 3.1 of the VnV Plan document

Task	**Assigned Member**	**Due By**
Place the results for M9 in an interface	Author	March 22, 2025
Modify inputs for M3	Author	March 24, 2025
Review modules and present feedback	Domain Expert	March 25, 2025
Modify modules for final documentation	Author	April 18, 2025

Table 4: Module Timeline

References

Frank Dellaert and GTSAM Contributors. borglab/gtsam, May 2022. URL https://github.com/borglab/gtsam).

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.