Module Guide for Image Feature Correspondences for Camera Calibration

Kiran Singh

March 17, 2025

1 Revision History

Date	Version	Notes
2025-03-19	1.0	Initial Release

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
BRIEF	Binary Robust Independent Elementary Featu
DAG	Directed Acyclic Graph
FAST	Features from Accelerated Segment Test
IFCS	Image Feature Correspondence Software
M	Module
MG	Module Guide
ORB	Oriented Fast and Rotated Brief
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Image Feature Correspondences for Camera Calibration	Program to identify and retrive regions of interest within the scene of camera imagery
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 (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by **?**, 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 (**?**). 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 constraints on the input parameters.

AC4: The format of the output data.

AC5: The constraints on the output results.

AC6: The algorithm used to reduce noise within the imagery.

AC7: The algorithm used to identify keypoints in an image.

AC8: The algorithm used to define features within an image.

AC9: The algorithm used to compare features between images.

AC10: The implementation of the data structure used to hold identified feature matches.

AC11: The relationships between what methods of feature identification, description, and comparison are compatible with each other.

AC12: The user wants to modify the metrics to be stored alongside a candidate matches of features.

AC13: The user wants to visualize the outputs of a given processing operation.

AC14: The user wants to save a snapshot of the output from any given processing operation.

[Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters. —SS]

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: The goal of the system is to identify and compare features identified within different images.

UC3: The methods of feature detection can be executed using parameters defined in the input parameters module.

UC4: The methods used to compare features can be executed using parameters defined in the input parameters module.

UC5: The user interface is changed from a .config file and command line arguments to a novel user GUI.

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. Modules are numbered by **M** followed by a number.

M1: Hardware Hiding Module

M2: Input Format Module

M3: Specification Parameters Module

M4: Output Format Module

M5: Output Verification Module

M6: Control Module

M7: Image Smoothing Module

M8: Keypoint Detection Module

M9: Feature Descriptor Module

M10: Feature Matching Module

M11: Image Data Structure Module

M12: Image Plot Module

M13: Feature Match Data Module

M14: Dataframe Structure Module

M15: ORB Data Structure Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	M2
	M3
	M4
	M5
	M6
	M7
	M8
	M9
	M10
Software Decision Module	M11
	M12
	M15
	M13
	M14

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 **?**. 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. *IFCS* means the module will be implemented by the Image Feature Correspondences for Camera Calibration 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 (M2)

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: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Specification Parameters Module (M3)

Secrets: The default values and software limits for constraints on input variables.

Services: Read access for the input checks against the parameters in M2.

Implemented By: IFCS

Type of Module: *Record* [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.3 Output Format Module (M4)

Secrets: The format of the output data and the structure in which it is contained.

Services: Outputs the results of the matched features, including the source image and feature identifiers.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.4 Output Verification Module (M5)

Secrets: Contains the algorithms used to check uniqueness of feature matches and their identifiers.

Services: Reviews the set of matches image features and flags any that are repeated or share the same point of origin.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.5 Control Module (M6)

Secrets: The procedure to run the program.

Services: Oversees execution of the overall program.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.6 Image Smoothing Module (M7)

Secrets: The Kernel parameters used to smooth the image.

Services: Defines the smoothing kernel using the parameters in the Input Format module.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.7 Keypoint Detection Module (M8)

Secrets: The methods used to define features within images.

Services: Defines the available methods used to detect features using the parameters in the Input Format module.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.8 Feature Descriptor Module (M9)

Secrets: The methods of defining features within images.

Services: Defines the available methods used to describe features using the parameters in the Input Format module.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.9 Feature Matching Module (M10)

Secrets: The methods used to compare features within images.

Services: Defines the available methods used to compare image features using the parameters in the Input Format module.

Implemented By: IFCS

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.3 Software Decision Module

7.3.1 Image Data Structure Module (M11)

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: OpenCV library

7.3.2 Image Plot Module (M9)

Secrets: The data structure of the image to be displayed.

Services: Defines the methods used to generate and display an image as a figure.

Implemented By: OpenCV library

7.3.3 Feature Match Data Module (M9)

Secrets: The algorithms used to identify matched features.

Services: Provides data structure and algorithms to compare features between images to identify candidate matches.

Implemented By: OpenCV library

7.3.4 Dataframe Structure Module (M9)

Secrets: The algorithms used to identify matched features.

Services: Provides data structure and algorithm to compare features between images to identify candidate matches.

Implemented By: OpenCV library

7.3.5 ORB Data Structure Module (M9)

Secrets: The algorithms that implement specialized keypoint detection and feature assignments as oriented-FAST and rotated BRIEF.

Services: Provides data structure and algorithms to perform combined FAST and BRIEF.

Implemented By: Pandas library

7.3.6 Etc.

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	M1, M2
R2	M1, M2
R3	M1, M2
R4	M1, M2
R5	M7
R6	M8, M15
R7	M9, M15
R8	M10
R9	M7
R10	M6, M8, M12
R11	M6, M9, M12
R12	M6, M10, M13, M12
R13	M5
R14	M4, M13
R15	M4, M5, M14

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M4
AC5	M5
AC6	M7
AC7	M8
AC8	M9
AC9	M10
AC10	M14
AC11	M6, M15
AC12	M13

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

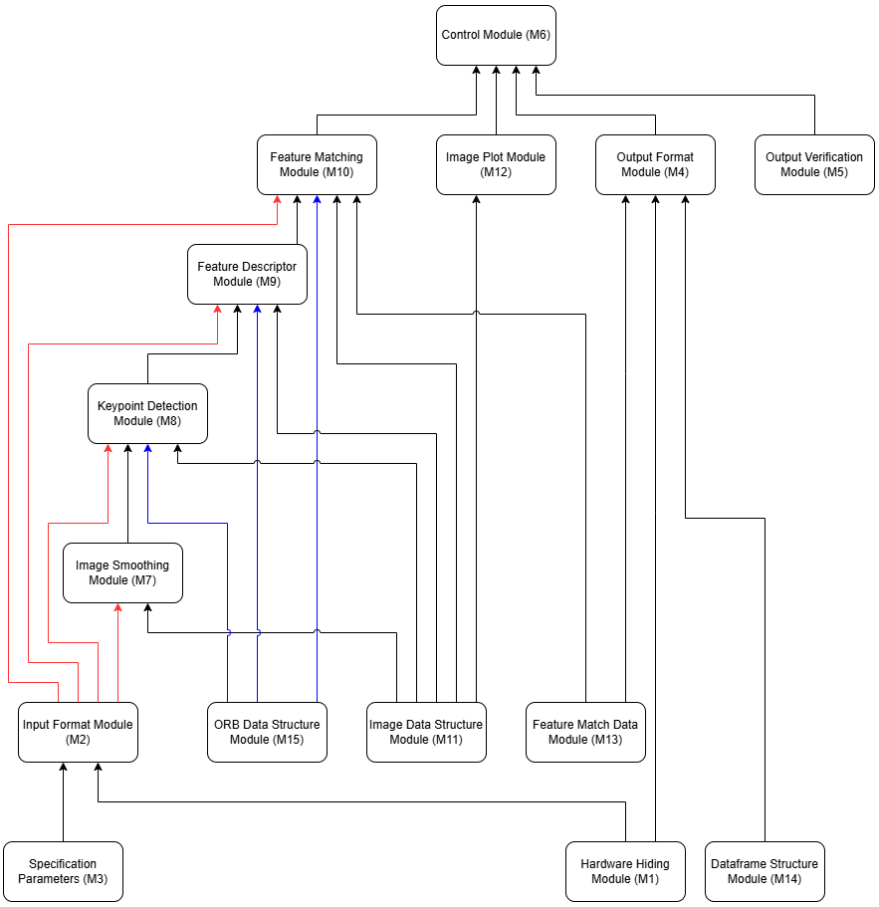

Figure 1: Use hierarchy among modules

10 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

11 Design of Communication Protocols

[If appropriate —SS]

12 Timeline

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]