Module Guide for Attitude Check

Adrian Sochaniwsky

March 3, 2024

1 Revision History

Date	Version	Notes
March 3, 2024	1.0	Initial draft

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Attitude Check	Explanation of program name
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 SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the from 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: Add online gyroscope bias compensation.

AC2: Add magnetic disturbance compensation.

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. Currently, there are no unlikely changes.

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: Matrix Math Module

M2: Quaternion Module

M3: Input Verification Module

M4: Initial Quaternion Estimator w/o Mag Module

M5: Initial Quaternion Estimator w Mag Module

M6: Estimate w/o Mag Module

M7: Estimate w Mag Module

M8: Control Module

Level 1	Level 2
Behaviour-Hiding Module	Control Module Input Verification Module Initial Quaternion Estimator w/o Mag Module Initial Quaternion Estimator w Mag Module Estimate w/o Mag Module Estimate w Mag Module
Software Decision Module	Matrix Math Module Quaternion 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. *Attitude Check* means the module will be implemented by the Attitude Check 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 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: –

7.1.1 Input Verification Module (M3)

Secrets: N/A

Services: Verifies input data is within the limits described in the SRS.

Implemented By: Attitude Check

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.1.2 Initial Quaternion Estimator w/o Mag Module (M4)

Secrets: The algorithm for computing a quaternion from a single accelerometer and gyroscope measurement.

Services: Calculates an initial orientation quaternion from accelerometer and gyroscope data.

Implemented By: Attitude Check

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.1.3 Initial Quaternion Estimator w Mag Module (M5)

Secrets: The algorithm for computing a quaternion from a single set of accelerometer, gyroscope, and magnetometer measurements.

Services: Calculates an initial orientation quaternion from accelerometer, gyroscope, and magnetic data.

Implemented By: Attitude Check

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.1.4 Estimate w/o Mag Module (M6)

Secrets: The algorithm for computing an estimated orientation using sequential accelerometer and gyroscope data.

Services: Calculates orientation estimates from accelerometer and gyroscope data.

Implemented By: Attitude Check

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.1.5 Estimate w Mag Module (M7)

Secrets: The algorithm for computing an estimated orientation using sequential accelerometer, gyroscope, and magnetometer data.

Services: Calculates orientation estimates from accelerometer, gyroscope, and magnetometer data.

Implemented By: Attitude Check

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.1.6 Control Module (M8)

Secrets: The algorithm for managing the program.

Services: Provides the main program.

Implemented By: Attitude Check

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.2 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: –

7.2.1 Matrix Math Module (M1)

Secrets: Algorithms and functions for matrix math including multiplication, inverses, etc.

Services: Performs matrix math.

Implemented By: Eigen (Guennebaud et al., 2010)

7.2.2 Quaternion Module (M2)

Secrets: Data structure for quaternion data.

Services: Provide a quaternion object and its associated mathematical operations, multiplication, inverses, etc.

Implemented By: Attitude Check

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
R2	M4, M5, M1, M2
R3	M1, M2, M3, M5, M7, M8
R4	M1, M2, M3, M4, M6, M8
R5	M2

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M6, M7
AC2	M7

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.

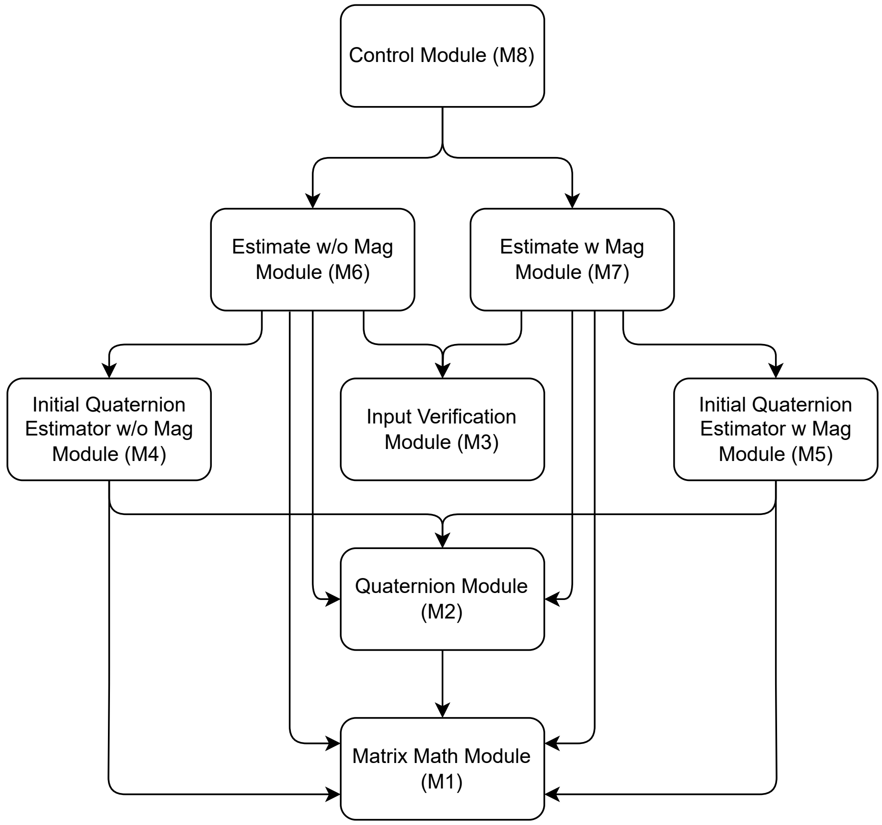

Figure 1: Use hierarchy among modules

References

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

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.