Module Guide for Mechatronics

Team #20, OpenASL
Robert Zhu zhul49
Zifan Meng mengz17
Jiahui Chen chenj194
Kelvin Huynh huynhk12
Runze Zhu zhur25
Mirza Nafi Hasan hasanm21

April 6, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	Everyone - Initial MG Draft
April 05, 2023	2.0	Everyone - Rev 1

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
Mechatronics	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 (**?**). 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 implementation of a more advanced motion tracking algorithm so that continuous motions with faster speed can be captured.

AC2: The formatting of the normalized coordinates input into the Classification Module.

AC3: The format of export normalized coordinates.

AC4: Allow for static image input to enable training using larger datasets.

AC5: Increase the area that the recognition boundary encloses for recognizing more complicated gestures.

AC6: Modify keypoint classification to check multiple points instead of single point for more complex motion.

AC7: Change from static classifier set to a dynamic classifier set that is able to expand the number of gestures.

AC8: The implementation of a more suitable machine learning model for better performance.

AC9: The implementation of reading a translated script into audio output through Raspberry Pi.

AC10: The hardware tracking devices on which the the software is running was replaced with a Raspberry Pi with the camera from a PC.

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: The csv format to another file type.

UC2: The output to just TTS instead of both TTS and video.

UC3: Capture full body and facial recognition for sign language.

UC4: The hardware design is not likely to change once it is finished.

UC5: Training method for the machine learning model.

UC6: The goal for the ASL translator is to translate the user's hand gestures to the corresponding English words or phrases.

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: Gesture Detection Module

M2: Coordinate Normalization Module

M3: Coordinate Export Module

M4: Video Capture Module

M5: Video Analysis Module

M6: Data Processing Module

M7: Data Collection Module

M8: Machine Learning Module

M9: User Interface Module

M10: Hardware Hiding Module

Level 1	Level 2
Hardware-Hiding Module	Video Capture Module Hardware Hiding Module
Behaviour-Hiding Module	User Interface Module Data Processing Module - Communicates with ML module with data from coordinate normalization module Data Collection Module - Communicates with ML module to update dataset Coordinate Export Module - Read data from video capture and stores into file Gesture Detection Module - Controller (ties everything together)
Software Decision Module	Video Analysis Module - requires data to be used Machine Learning Module Coordinate Normalization 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

7.0.1 Video Capture Module (M4)

Secrets: The algorithm used to track hand movement through any camera lens

Services: Allows the progrom to interface with an external recording devices

Implemented By: TensorFlow

7.1 Hardware Hiding Modules (M10)

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 of PC or Raspberry Pi

7.2 Behaviour-Hiding Module

7.2.1 Motion Tracking Module (M1)

Secrets: The algorithm by which the software detects the hand gestures performed in ASL

Services: Receives the video from the camera as input, and locates the different hand joints to return the coordinates of the hand position

Implemented By: Python

7.2.2 Coordinate Export Module (M3)

Secrets: The algorithm used to create a dataset for the machine learning model

Services: Provides access to normalized coordinate data

Implemented By: Python

7.2.3 Data Processing Module (M6)

Secrets: The algorithm used to return index/classifier closest to the normalized coordinates

Services: Provides communication between the motion tracking module and the machine learning module by storing the gathered information in the dataset

Implemented By: Python

7.2.4 Data Collection Module (M7)

Secrets: Gathers data points from the user and formats the structure into input for machine learning

Services: Combines classifier label and coordinate data for application use

Implemented By: Python

7.2.5 User Interface Module (M9)

Secrets: The algorithm and dataset used to translate text-to-speech

Services: This module serves as the UI interface that contains methods to convert the sign language gesture into audio output

Implemented By: Python

7.3 Software Decision Module

7.3.1 Coordinate Normalization Module (M2)

Secrets: The algorithm to restructure the coordinates gathered from users into a value between 1 and -1

Services: Standardizes the coordinate data allowing for different resolutions in cameras to be used

Implemented By: Python

7.3.2 Video Analysis Module (M5)

Secrets: The algorithm by which the hand joints and classifier are displayed for the user

Services: Based on the data from the Motion Tracking Module (M1), the algorithm places a joint overlay over the user's hand as a visual tracker for all 20 joints

Implemented By: Python

7.3.3 Machine Learning Module (M8)

Secrets: The data structure in which classifier label data is paired with normalized coordinate data

Services: Automates classification of sign language based on input coordinate data

Implemented By: Python

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
GFR1	M1, M4
GFR2	M1, M3
GFR3	M1, M5
GFR4	M1, M2
GFR5	M1, M2
GFR6	M1, M5
GFR7	M1, M4, M5
GFR8	M7
MLFR1	M2
MLFR2	M3
MLFR3	M4
MLFR4	M6, M9, M10
MLFR5	M4
NFR1	M5, M6, M8
NFR2	M9
NFR3	M9
NFR4	M8
NFR5	M10
NFR6	M1, M4, M9

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M4
AC5	M5
AC6	M6
AC7	M7
AC8	M8
AC9	M9
AC10	M10

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.

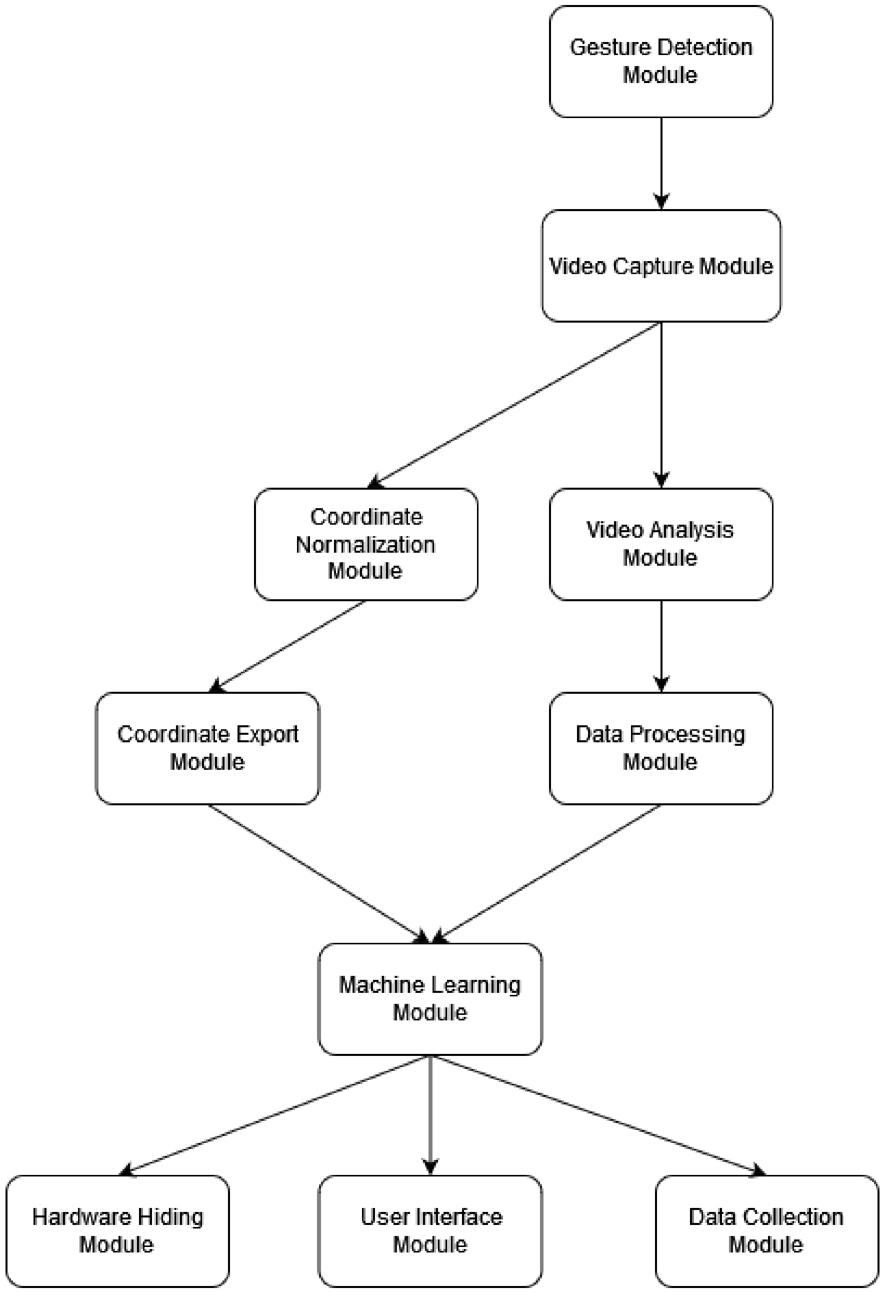

Figure 1: Use hierarchy among modules