Module Guide for PCD: Partially Covered Detection of Obscured People using Point Cloud Data

Team #14, PCD
Tarnveer Takhtar
Matthew Bradbury
Harman Bassi
Kyen So

January 16, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Gr
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computin
SRS	Software Requireme
PCD: Partially Covered Detection of Obscured People using Point Cloud Data	Explanation of prog
UC	Unlikely Change
[etc. —SS]	[... —SS]

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 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 constraints on the input parameters.

AC3: The constraints on the output results.

AC4: The classification model used to determine the presence of a human.

[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, Kinect, and Keyboard, Output: File, Screen).

UC2: There will always be a source of input data external to the software.

UC3: Output data will always be displayed to the output device.

UC4: The goal of the system is to accurately detect and identify presence of a human.

UC5: The format of the initial input data.

UC6: The format of the input parameters.

UC7: The format of the final output data.

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

M2: Kinect Stream

M3: Application Control

M4: Input Data Read

M5: Input Classifier

M6: Input Classifier Ranking

M7: Bounding Box Display

M8: Point Cloud Data Structures

M9: Input Processing

M10: Command Line Interface

M11: Graphical User Interface

Level 1	**Level 2**
Hardware-Hiding Module	Kinect Stream
Behaviour-Hiding Module	Application Control Input Data Read Input Classifier Input Classifier Ranking Bounding Box Display
Software Decision Module	Point Cloud Data Structures Input Processing Command Line Interface Graphical User Interface

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.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

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. *PCD: Partially Covered Detection of Obscured People using Point Cloud Data* means the module will be implemented by the PCD: Partially Covered Detection of Obscured People using Point Cloud Data 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

7.1.1 Kinect Stream Module (M2)

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

Services: Sets up the connection needed for the Kinect and handles providing our other modules with the continuous output of the Kinect.

Implemented By: kinect2_grabber

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 Application Control Module (M3)

Secrets: The algorithm for the overall flow of the program.

Services: Provides the main program.

Implemented By: PCD

Type of Module: Abstract Data Type

7.2.2 Input Data Read Module (M4)

Secrets: The format, constraint, and structure of the input data.

Services: Prompts the user to select between offline and live input types. Then reads in either the offline file or live Kinect stream and checks that it is valid. Converts the input data into the data structure used by the Input Processing Module M9.

Implemented By: PCD

Type of Module: Abstract Data Type

7.2.3 Input Classifier Module (M5)

Secrets: The model used for classification given the input point cloud.

Services: Defines the strategies that make up the model used to determine human(s) on the pre-processed point cloud given by the Input Processing ModuleM9.

Implemented By: PCD

Type of Module: Abstract Data Type

7.2.4 Input Classifier Ranking Module (M6)

Secrets: The weights and biases that provide the decision of the classification strategy.

Services: Stores data of the highest performing classification strategy for the given point cloud. This is to be used by the Input Classifier Module M5

Implemented By: PCD

Type Of Model: Abstract Data Type

7.2.5 Bounding Box Display (M7)

Secrets: Derives cluster calculated vertices to draw bounding box

Services: Draws a bounding box around a detected human, classified by the Input Classification Module M5

Implemented By: PCD

Type Of Model: Abstract Data Type

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 Point Cloud Data Structures (M8)

Secrets: Uses data structures for storing and organizing various forms of point cloud data.

Services: Provides point cloud data structures for use in other modules, such as Input Processing M9 and Graphical User Interface M11.

Implemented By: PCL

Type of Module: Library

7.3.2 Input Processing (M9)

Secrets: Applies algorithms for downsampling, noise removal, and data filtering on point cloud data.

Services: Provides methods for preprocessing point cloud data, utilizing structures from the Point Cloud Data Structures Module M8, to prepare data for analysis and visualization (used by Command Line Interface M**??** and Graphical User Interface Modules M11).

Implemented By: PCL

Type of Module: Library

7.3.3 Command Line Interface (M10)

Secrets: Handles user input through command-line text-based commands.

Services: Processes command-line input, triggering actions in the Input Processing Module M9 and Graphical User Interface Module M11.

Implemented By: OS

Type of Module: Interface

7.3.4 Graphical User Interface (M11)

Secrets: Provides methods for rendering and interacting with 3D point cloud data in a visual interface.

Services: Displays 3D point cloud data and allows user interaction, using structures from the Point Cloud Data Module M8 and processed data from the Input Processing Module M9.

Implemented By: PCL

Type of Module: Library

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**
F411	M4, M5, M6, M9, M8
F412	M7, M11
F413	M4, M10, M11, M5, M6, M9, M8
F414	M4, M5, M6, M9, M8
F415	M1, M2
NF431	M1, M2, M9
NF432	M9, M5, M6, M7
NF433	M5, M6, M7

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M4
AC3	M7
AC4	M5

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.

[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]

References

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.