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

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

January 17, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]
[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for Partially Covered Detection (PCD) software. The sections in this document describes each module in our software and how each module interacts with each other. Additional information and documentation can be found in System Requirement Specifications (SRS).

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/takhtart/PCD.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by PCD: Partially Covered Detection of Obscured People using Point Cloud Data.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
PointXYZRGBA	$\mathbb{P}$	point cloud data in the PCL Library

The specification of PCD uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, PCD uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

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 MIS of Kinect Stream

[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

6.1 Module

[Short name for the module —SS]

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

7 MIS of Application Control

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

7.1 Module

[Short name for the module —SS]

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

7.4 Semantics

7.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

7.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

7.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

8 MIS of Input Data Read

8.1 Module

reader

8.2 Uses

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
reader	-	-	-

8.4 Semantics

8.4.1 State Variables

- Choice (Integer): Choice that the user made on what mode to run the program in.

8.4.2 Environment Variables

- std::cin: Used to get the users input on what choice they want to make.

8.4.3 Assumptions

The user provides a valid input ($\mathbb{Z}$) corresponding with the correct option

8.4.4 Access Routine Semantics

main():

- transition: Converts the input data into the data structure used by the Input Processing Module **??**

8.4.5 Local Functions

None.

9 MIS of Input Classifier

9.1 Module

classify

9.2 Uses

- Bounding Box Display 11
- Point Cloud Data Structures 12

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
clssify	(PointCloudT::Ptr)cloud, (PointCloudT::Ptr) cloudfiltered		-

9.4 Semantics

9.4.1 State Variables

- *cloudFiltered = *personCloud : updated cloud value for the data set
- personCloud (PointCloudT::Ptr) : the cliuster that is being identified as a human within the frame.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

- The cloud has been properly filtered to allow for a good reading to quickly identify the human on screen.

9.4.4 Access Routine Semantics

classify(cloud, cloudfiltered)(): transition: This module will take the the filtered values and filter down further to just identify the human within the frame and only leave the data points connected to that person.

9.4.5 Local Functions

None.

10 MIS of Input Classifier Ranking

10.1 Module

ranking

10.2 Uses

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
ranking	dataPoint($\mathbb{P}$)	-	-

10.4 Semantics

10.4.1 State Variables

- weights ($\mathbf{P}^n$): An array of size n containing the ordered weights of the pcd points

10.4.2 Environment Variables

None.

10.4.3 Assumptions

- The input point cloud data is valid.
- The classification strategy is implemenented correctly to be able to order the weights

10.4.4 Access Routine Semantics

ranking(dataPoint)():

- transition: This module will take in the dataPoint and add it to the list and order it into the array.

10.4.5 Local Functions

None.

11 MIS of Bounding Box Display

11.1 Module

boundingBox

11.2 Uses

- Input Classifer Module

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
boundingBox	humancloud($\mathbb{P}$), minpt($\mathbb{P}$), maxpt($\mathbb{P}$)	-	-

11.4 Semantics

11.4.1 State Variables

- thickness (**R**): An float with the thickness of the box
- $scale_factor$ (**R**): Factor that adjusts the box size.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

- The input cloud points are valid to provide an accuurate drawing of the box.

11.4.4 Access Routine Semantics

boundingBox(humancloud,minpt,maxpt)(): transition: This module will take in inputed data points and add draw out a box using the max and pin points provided.

11.4.5 Local Functions

None.

12 MIS of Point Cloud Data Structures

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

12.1 Module

[Short name for the module —SS]

12.2 Uses

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

12.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

13 MIS of Input Processing

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

13.1 Module

[Short name for the module —SS]

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

13.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

13.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

13.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

13.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

14 MIS of Command Line Interface

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

14.1 Module

[Short name for the module —SS]

14.2 Uses

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

14.4 Semantics

14.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

14.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

14.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

14.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

14.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

15 MIS of Graphical User Interface

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

15.1 Module

[Short name for the module —SS]

15.2 Uses

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

15.4 Semantics

15.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

15.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

15.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

15.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

15.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

16 Appendix

[Extra information if required —SS]

Appendix — Reflection

1. Why is it important to create a development plan prior to starting the project?

   It is important to create a development plan prior to starting the project as it allows most of the heavy lifting behind project planning to be done before any work has started. It allows for expectations and workflow to be clearly defined before any issues arise. It also creates a document that can be referenced at other times to avoid confusion.

2. In your opinion, what are the advantages and disadvantages of using CI/CD?

   Employing CI/CD allows for better issue tracking and rollbacks. Utilizing CI/CD gives the opportunity for teams to better track individual issues and commits, leading to increased awareness and visibility on workflow issues. It also allows for easier time rolling back to a previous version in case something goes wrong. Some disadvantages are with the conceptual depth and speed. Ensuring a specific workflow and constant PR reviews can slow things down as contributors have to make sure that they are following the workflow properly and have to wait for PR reviews (when necessary). Furthermore, it is more effort to set up, both in the codebase and conceptually. The process has to be talked through and understood by all team members.

3. What disagreements did your group have in this deliverable, if any, and how did you resolve them?

   Our group mainly debated how to set up our GitHub workflows. Initially, we considered using individual branches for each issue, but we ultimately decided on a more streamlined approach with two revision branches and separate forks. This allows us to effectively manage pull requests for merging feature changes into the codebase. We reached this agreement after discussing the benefits of clarity and collaboration in our development process.

4. What went well while writing this deliverable?

   This deliverable allowed for the group to be able to discuss standard goals vs stretched goals. Everyone contributed their ideas and we were able to come to a clear conclusion when it came to the goals and problem statement of the project. It also allowed us all to see where the project is headed and how we should properly prepare ourselves so that we can achieve our goals.

5. What pain points did you experience during this deliverable, and how did you resolve them?

The biggest pain point during this deliverable was being able to decide which goals were too ambitious and outside our design scope. Some goals such as the aspect of outlining the human in the environment were broken up. There was a deep discussion on how to properly decide the goals, but at the end the group got a better understanding of the project.

6. How did you and your team adjust the scope of your goals to ensure they are suitable for a Capstone project (not overly ambitious but also of appropriate complexity for a senior design project)?

   Because our project is presented by a professor, the team already had a pretty clear understanding of the goals that needed to be achieved for our project to be considered a success. The only issue was trying to ensure that the goals can be properly broken down so that a goal that seemed a bit ambitious could be broken into something that seems doable. For example, the human detection is broken into the Minimal and viable product goal and then also extended into the stretch goal. This was an important discussion that the group had to ensure that we deemed our goals to be doable.

7. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

   Every member of the group needs to acquire knowledge on Computer Vision and understanding how pcd files operate. Overall these are big topics and so the basics should be acquired by every member, but some specifics would be broken down to different parts for each member. Tarnveer and Matthew would be assigned to understanding how to track people on screen and map them on the screen. Harman and Kyen would be working on reading in the pcd files and understanding the PCL. The PCL will explore on aspects of boxing the points on screen. Tarnveer would also be responsible for understanding the real time coding aspect in c++. Matthew would also be assigned to acquire knowledge on improving the human outline based on better data. Harman will be assigned to understanding how to cancel out noise from the data set. Kyen will have to understand how to find the person based off the pcd and understand how to properly read the files.

8. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

One approach for acquiring the knowledge would be to use the provided documentation for the libraries (PCL and OpenCV). This would provide a good fundamental understanding for the important aspects of the project.

Another approach would be to just watch videos on the specific topics and try to understand from there. This would be able to provide a more visual explanations for the topics.

Tarnveer: use documentation and videos Matthew: use documentation and videos Kyen: use documentation Harman: use documentation and videos

9. What went well while writing this deliverable?

   The deliverable was straightforward. We had a rough idea of the main hazards within our project and tried to make sure that we covered the main scope. The document writing was split between all of us. The document is pretty straightforward and we as a group were able to talk over the different sections and divide up the work.

10. What pain points did you experience during this deliverable, and how did you resolve them?

    The biggest pain point was probably discussing what would be some assumptions we had to make, but we were able to come to an agreement by communicating our points of why or why not.

11. Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (ones you thought of while doing the Hazard Analysis), how did they come about?

    All the risks were mainly thought of before the deliverable. We knew that we needed to ensure that the offline file is the correct format and that the system needs to make sure it is working with a Kinect sensor and not something else.

    The privacy risk was something we thought of at the informal interview.

12. Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least 2 other types of risk in software products. Why are they important to consider?

    Could be some performance risks and making sure that the performance of the software meets the goals/requirements for the project. Another risk could be in terms of privacy. The application is capturing sensitive data and so its important on how the application handles this data.

13. What went well while writing this deliverable?

    This deliverable was relatively painless and straightforward. We had already started thinking about testing plans during our SRS deliverable, specifically section S.6. In this section we detailed a brief VnV plan, including system testing and unit testing.

This set up the basic outline for this deliverable, it being an extension of what we already wrote/thought about.

14. What pain points did you experience during this deliverable, and how did you resolve them?

    One pain point we had during this deliverable was editing requirements from the SRS. When creating the traceability matrix, we noticed that some of the requirements from the SRS document were overlapping or in the wrong spot. We held a meeting as a group to sort this out and reach a consensus on which requirements should stay, should be changed, or should be deleted.

15. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

    In order to properly complete the verification and validation of our project, some skills will need to Be acquired. Firstly, we will have to familiarize ourselves with cppunit, as majority of our testing knowledge from previous courses is in Java or Python. Additionally, because of this, we will have to learn how to use GCov in order to accurately figure out our code coverage. On the topic of code coverage, we will also need to brush up on our coverage definitions that were learnt in our testing course. Finally, we will need to implement linters to check our code on github before merge.

16. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

    For these skills, there are a few ways to approach acquiring the knowledge. With new skills, utilizing Youtube and online tutorials are a good way to quickly learn the basics and proper implementation of new techniques. For older skills, i.e. ones that we have learnt previously but haven't used in a while, we can go back to old projects/lectures and relearn the information.

    Matthew will find online tutorials to learn about cppunit. This is because he has no experience with creating automated testing in c++, and needs to start off by learning the basics.

    Tarnveer will find online tutorials to learn about cppunit and c++ linters on Github. This is for the same reason as above; he has no experience implementing testing in c++ or adding linters to Github for PRs.

    Harman will go back to our old 3SO3 notes in order to relearn MC/DC coverage. This is because he had implemented MC/DC coverage and checks in that course, but in

Java. Since he has implemented this before, he is familiar with the content and should be relatively painless to relearn the content.

Kyen will find online tutorials for learning about GCov. For similar reasons as Matthew and Tarnveer, this is because he has no prior experience with this specific coverage tool.

17. What went well while writing this deliverable?

    Everyone on the team was on track with their sections of the assignment and we were able to thick of better ways to break up some modules to make more sense.

18. What pain points did you experience during this deliverable, and how did you resolve them?

    Getting used to the new year and so it was a slow start trying to get back into the flow, but once we started working it came back.

19. Which of your design decisions stemmed from speaking to your client(s)or a proxy (e.g. your peers, stakeholders, potential users)? For those thatwere not, why, and where did they come from?

    Most of the module break up comes from talking to our client becuase they helped us focus on their vision for the project but making the inputs and specific variables were all done independently.

20. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

    For now no real document had to be changed becuase the structure for this assignment was thought of before through the many client meets. This allowed for a strong structure.

21. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

    With unlimited resources the ability to capture better imaging with the kinect would allow for a faster and more precise human detection algorithm. Maybe also being able to better maximize the human detection to better fit a humaniod shape.

22. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

    Other design implications would just involve taking a different approach to creating the algorithm. The issue with for example a solution that does not use hue or skin color is limiting our ability to full captalize on the fact that the sensor picks up RGB as well.