System Verification and Validation Plan for PCD: Partially Covered Detection of Obscured People using Point Cloud Data

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

April 4, 2025

Revision History

Date	Version	Notes
Nov 5, 2024	Rev 0	Initial Draft
April 4, 2025	Rev 1	Revision 1
March 27, 2025	Rev 1	Revision 1
March 27, 2025	Rev 1	Revision 1
March 27, 2025	Rev 1	Revision 1
March 29, 2025	Rev 1	Revision 1
March 29, 2025	Rev 1	Revision 1
April 4, 2025	Rev 1	Revision 1
April 4, 2025	Rev 1	Revision 1

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

All symbols, abbreviations, and acronyms can be found within section E.1 (Glossary) of the SRS Report.

This document outlines the Verification and Validation plan for our software project. The purpose of this plan is to increase confidence in our software by ensuring it meets its requirements and performs as expected. This plan will list our objectives and processes related to verification and validation of our system and our roadmap for doing so.

2 General Information

2.1 Summary

The software being tested is our Partially Covered Detection (PCD) system. The system leverages depth and RGB layers to form a combined coloured point cloud, which is then analyzed to accurately detect individuals even when they are not fully visible. Detection can occur in a live setting through a Kinect, or using offline file captures.

2.2 Objectives

The primary objective of this VnV plan is to ensure the system's correctness and performance, verifying both its functional and non-functional requirements. This involves testing the system's ability to accurately identify partially obscured individuals and demonstrate that it operates efficiently within its environment. Additionally, the plan aims to ensure that the system implementation matches the project specifications. Testing within dark environments is considered out of scope, as RGB data cannot be captured in such settings. Furthermore, the project assumes that any external libraries used have already been verified by their respective implementation teams.

2.3 Challenge Level and Extras

The challenge level for this project is advanced, as agreed upon with our assigned TA. A User Manual and Design Thinking additions are our included extras.

2.4 Relevant Documentation

The following documents are relevant to the Verification and Validation efforts for this project. Each document listed below provides information supporting the VnV process, ensuring that the system meets its requirements and operates as intended.

1. Problem Statement and Goals: This document outlines the primary objectives and challenges of the project. It provides a clear understanding of the problem being addressed and the goals to be achieved, and is vital for defining the scope and focus of the VnV proceedings.

2. Development Plan: This plan includes risks related to the project prior to conducting a formal hazard analysis. It is important to verify the risks outlined in the document relating to the software system have been mitigated.

3. SRS Report: The Software Requirements Specification (SRS) report defines the functional and non-functional requirements of the system. It includes a brief baseline for VnV efforts, providing the criteria against which the system's correctness and performance will be evaluated.

4. Hazard Analysis: This document identifies potential hazards and risks associated with the system. It is essential for the VnV process as it helps in prioritizing the testing efforts to address the most critical risks and ensure the system's safety and reliability.

5. Module Interface Specification: The Module Interface Specification (MIS) describes the interfaces between different modules of the system. It is relevant to the VnV efforts as it ensures that the interactions between modules are correctly implemented and function as intended.

6. Module Guide: The Module Guide (MG) provides detailed descriptions of each module's design and implementation. It is used in the VnV process to verify that each module meets its design specifications and integrates seamlessly with other modules.

7. VnV Report: This report documents the results of the VnV activities, including the tests performed, issues identified, and their resolutions. It provides a comprehensive evaluation of the system's compliance with its requirements and serves as a record of the VnV efforts.

3 Plan

This section describes the overall plan for the verification and validation of our system. It includes the work breakdown of each member of the verification and validation team. This section also outlines the plans for the verification of our SRS, Design, and VnV. Furthermore, it details the plans for the implementation of these verification strategies as well as the implementation of the testing tools and the software validation plan.

3.1 Verification and Validation Team

Table 1: Verification and Validation Team Members Table

Name	**Role(s)**	**Responsibilities**
Harman Bassi	Lead test developer, Test developer, Manual tester	Lead the test development process. Create automated tests for backend code. Main verification and reviewer of system/unit tests.
Matthew Bradbury	Test developer, Manual tester, Code Verifier	Create automated tests for backend code. Manually test human detection algorithm functionality. Ensure source code follows project coding standard. Verification reviewer for the Hazard Analysis and SRS.
Kyen So	Test developer, Manual tester, Code Verifier	Create automated tests for backend code. Manually test human outline manager functionality. Ensure source code follows project coding standards. Main verification reviewer for the Verification and Validation document.
Tarnveer Takhtar	Test developer, Manual tester	Create automated tests for backend code. Manually test Kinect manager and ensure proper functionality. Verification Reviewer of Hazard Analysis and SRS
Dr. Gary Bone	Supervisor, SRS validator, Final reviewer	Make sure SRS meets requirements of the project, Validate code functionality. Because Dr. Bone is the supervisor of this project, he can verify that the project is functioning as expected.

3.2 SRS Verification Plan

The current plan to verify our SRS involves incorporating both self-review and peer-review feedback, used in tandem with notes from our TA and a final read-over with our supervisor. Our team will first do a quick read-through of each other's sections and provide feedback for changes in the form of comments on the issue. We will then incorporate the feedback we receive from our peers in another group, delivered to us via separate Github issues. These issues will be assigned to a single member of the team, who will have the responsibility of finishing and closing it. Additionally, we will create issues related to the feedback we received from our TA and work on adding the corresponding changes to our SRS. Finally, after incorporating all the feedback received, we will host a meeting with Dr. Bone. In this meeting, the team will walk Dr. Bone through our SRS and get his opinion on any final changes that need to be made to our requirements. Issues will be created to address the requested changes.

3.3 Design Verification Plan

Like the SRS, the plan for design verification includes a team review, a peer review, and a TA review. Like the SRS, the team review will involve team members reviewing another team member's section and commenting on changes that should be made. The peer review will similarly consist of another team adding Github issues to our repository and getting assigned to a team member. At that time, the specified team member will complete and close the issue. Additionally, we will incorporate issues raised by our TA.

3.4 Verification and Validation Plan Verification Plan

Similarly to the previous plans, this one will consist of a team review, a peer review, and a TA review, as well as a review from our supervisor. Just as the previous plans, the team, peer, and TA review will be conducted in the same way as previously mentioned. Additionally, for this verification plan, we will also be including our supervisor, Dr. Bone. The team will schedule a meeting with him and go through specific parts of the document. These parts will be our plan for what automated tests we are implementing, and Dr. Bone will have the final word on any improvements or changes we should make before proceeding.

3.5 Implementation Verification Plan

For our implementation verification plan, we will perform the corresponding set of system tests from 4.1.1, 4.1.2 or 4.1.4 in every version of the code that makes changes to either the human detection component or the 3D space estimation component. This allows us to verify that our changes continues to meet our functional requirements. For major version updates, we will perform our entire test suite of 4.1 and 4.2, ensuring that all of our functional and non-functional requirements are met and our software is working as intended. Despite most of our tests being manual, many of them aren't particularly time consuming.

Additionally, for every major version update, a code walkthrough will be performed by the developers and Dr. Gary Bone so that we can find out if there are any discrepancies between what had been implemented and the stakeholder's needs.

3.6 Automated Testing and Verification Tools

For the automated testing, cppunit will be used as it provides a simple and portable way to unit test the system. When it comes to doing coverage testing it would be best to use GCov because it is compatible with VScode and easier to set up compared to other applications. The main coverage focus for the project would be MC/DC coverage because it is a good coverage test for complicated decisions which the PCD system will have to make. Linters are also a good tool to help ensure all the code meets a certain standard that is respected within the field. For this project, Clang-Tidy will be used because it is compatible with VS code and meets the standards within the industry for C++.

3.7 Software Validation Plan

For software validation, we will be providing Dr. Bone with bi-weekly written updates on the progress of the project. These updates will serve as a brief validation that our software matches the aforementioned requirements outlined in the SRS. These smaller written checkups serve as an iterative way to validate the software against the requirements by constantly getting input from Dr. Bone about new code. Larger releases, such as code milestones or demos, will be accompanied by a meeting with Dr. Bone instead of a

written update. These meetings will be lead by the main developers of the corresponding section and serve as the main form of software validation. The team will also consider peer feedback and TA/prof feedback from the demo to determine if the software accomplishes its goals.

4 System Tests

System tests are divided into tests for functional and non-functional requirements. Each functional and non-functional requirement we have will be sufficiently tested to ensure that these requirements are properly implemented and integrated into our system.

4.1 Tests for Functional Requirements

Each subsection below covers a functional requirement of the system. All functional requirements are accounted for and each have its own tests such that all functional requirements are met.

4.1.1 Human Detection Testing

Live Tests

1. Live Full body, Uncovered Test (FT11)

   Control: Manual

   Initial State: Kinect is properly set up whilst facing an environent without any objects that may obstuct the view of a person. Software is running in realtime and is not detecting anyone in frame. A person is ready to walk into frame.

   Input: Realtime PCD from Kinect

   Output: The software recognizes that there is a human in frame in realtime.

   Test Case Derivation: The software is expected to first not detect any humans while no one is in frame. Once the human is fully in frame, the software is expected to recognize that someone is now in frame in order to sastify the requirement of human detection.

How test will be performed: The test begins with an environment free of any objects that may obstruct the view of a person and the software running in realtime. The tester will ensure that the software doesn't detect any human while no one is in frame. Then a human will be instructed to walk into frame. The output of the software will be inspected and analyzed once the human is fully in frame.

2. Live Upper Body, Uncovered Test (FT12)

   Control: Manual

   Initial State: Kinect is properly set up whilst facing an environent without any objects that may obstuct the view of a person. Software is running in realtime and is not detecting anyone in frame. A person is ready to walk into frame close to the kinect such that only their upper body is visible.

   Input: Realtime PCD from Kinect

   Output: The software recognizes that there is a human in frame in realtime.

   Test Case Derivation: The software is expected to first not detect any humans while no one is in frame. Once the top half of the human is fully in frame, the software is expected to recognize that someone is now in frame in order to sastify the requirement of human detection.

   How test will be performed: The test begins with an environment free of any objects that may obstruct the view of a person and the software running in realtime. The tester will ensure that the software doesn't detect any human while no one is in frame. Then a human will be instructed to walk into frame close to the kinect such that only their upper body is visible. The output of the software will be inspected and analyzed once the human is fully in frame.

3. Live Human Partially Covered by Another Human Test (FT13)

   Control: Manual

   Initial State: Kinect is properly set up whilst facing an environment without any objects that may obstruct the view of a person. Software is

running in realtime and is not detecting anyone in frame. Two humans are ready to walk into frame.

Input: Realtime PCD from Kinect

Output: The software correctly recognizes and distinguishes 2 humans separately in realtime

Test Case Derivation: The software is expected to first not detect any humans while no one is in frame. Once both humans are fully in frame and one human is partially covering the other, the software is expected to recognize that there are two humans in frame and be able to distinguish each human in order to satisfy the requirement of human detection.

How test will be performed: The test begins with an environment free of any objects that may obstruct the view of a person and the software running in realtime. The tester will ensure that the software doesn't detect any human while no one is in frame. Then, both humans will be instructed to walk into frame with one human partially covering the other. The output of the software will be inspected and analyzed once both humans are fully in frame and in the correct position.

4. Live Human Partially Covered by Another Object Test (FT14)

   Control: Manual

   Initial State: Kinect is properly set up whilst facing an environment with an object placed that is large enough to partially obstruct the view of a person. Software is running in realtime and is not detecting anyone in frame. A person is ready to walk into frame.

   Input: Realtime PCD from Kinect

   Output: The software recognizes that there is a human in frame in realtime.

   Test Case Derivation: The software is expected to first not detect any humans while no one is in frame. Once the human is fully within the frame and is partially covered by the object, the software is expected to recognize that there is a human behind the object in order to satisfy the requirement of human detection.

How test will be performed: The test begins in an environment that contains an object big enough to partially obstruct a person from the Kinect's view, such as a chair or table, and with the software running in realtime. The tester will ensure that the software doesn't detect any human while no one is in frame. Then, a human will be instructed to walk into frame and stand behind the object. The output of the software will be inspected and analyzed once the human is in the correct position.

Offline Tests

1. Offline Full Body, Uncovered Test (FT15)

   Control: Manual

   Initial State: The system is running in offline mode which allows the user to upload a .pcd file.

   Input: .pcd file containing the full-body of a human unobstructed by any objects. (Full Body 1.pcd)

   Output: The software recognizes that there is a human in frame

   Test Case Derivation: Given a .pcd file input, the software is expected to be able to analyze the data in the file. Since the .pcd file contains the full-body of a human, the software is expected to recognize that there is a human in the frame in order to satisfy the requirement of human detection.

   How test will be performed: The test begins by first obtaining a .pcd file that contains the full-body of a human unobstructed by any objects. Then, the .pcd file is uploaded to the software. The output of the software will be inspected and analyzed by the tester.

2. Offline Upper Body, Uncovered Test (FT16)

   Control: Manual

   Initial State: The system is running in offline mode which allows the user to upload a .pcd file.

   Input: .pcd file containing only the upper body of a human unobstructed by any objects. (PartiallyObstructedHalfBodyVisible.pcd)

Output: The software recognizes that there is a human in frame

Test Case Derivation: Given a .pcd file input, the software is expected to be able to analyze the data in the file. Since the .pcd file contains the upper body of a human, the software is expected to recognize that there is a human in the frame in order to satisfy the requirement of human detection.

How test will be performed: The test begins by first obtaining a .pcd file that contains only the upper body of a human unobstructed by any objects. Then, the .pcd file is uploaded to the software. The output of the software will be inspected and analyzed by the tester.

3. Offline Human Partially Covered by Another Human Test (FT17)

   Control: Manual

   Initial State: The system is running in offline mode which allows the user to upload a .pcd file.

   Input: .pcd file containing 2 humans with one human partially covering the other. (PartiallyCoveredByAnotherHuman.pcd)

   Output: The software correctly recognizes and distinguishes 2 humans separately

   Test Case Derivation: Given a .pcd file input, the software is expected to be able to analyze the data in the file. Since the .pcd file contains 2 humans with 1 partially covering the other, the software is expected to recognize and distinguish each human in order to satisfy the requirement of human detection.

   How test will be performed: The test begins by first obtaining a .pcd file that contains 2 humans with 1 partially covering the other. Then, the .pcd file is uploaded to the software. The output of the software will be inspected and analyzed by the tester.

4. Offline Human Partially Covered by Another Object Test (FT18)

   Control: Manual

   Initial State: The system is running in offline mode which allows the user to upload a .pcd file.

Input: .pcd file containing a human who is partially covered by an object such as a chair or a table (PartiallyObstructedOnlyHeadVisible.pcd)

Output: The software recognizes that there is a human in frame

Test Case Derivation: Given a .pcd file input, the software is expected to be able to analyze the data in the file. Since the .pcd file contains a human partially covered by an object, the software is expected to recognize that there is a human in frame in order to satisfy the requirement of human detection.

How test will be performed: The test begins by first obtaining a .pcd file that contains a human partially covered by an object such as a chair or table. Then, the .pcd file is uploaded to the software. The output of the software will be inspected and analyzed by the tester.

4.1.2 Location Prediction

Live Tests

1. Outside of Frame(FT21)

Control: Manual

Initial State: Kinect is properly set up whilst facing an environment without any objects that may obstruct the view of a person. Software is running in realtime and is not detecting anyone in frame. A person is ready to show a hand for the software to detect, while standing off screen.

Input: Realtime PCD from Kinect

Output: The software locates where the human should be and draws an arrow point to the approximate off screen location.

Test Case Derivation: The software would first expect to detect no one within the empty environment. Then when the human shows a skin point while standing out of frame, the software detect the skin point in frame. It will then determine the location of region growing, identify that its off screen and have an arrow point in the general direction.

How test will be performed: The test will first have the empty environment and then have a human ready to have their hand appear

in frame (while the person is still out of frame). The tester would then ensure that the screen does not detect any humans in the frame. Then the human would then reach out in front of the sensor making sure that their hand is the only visible skin point in front of the Kinect. The tester would then check the screen to see that the system draws an arrow pointing the direction of where the human should be standing.

2. Legs Covered by Desk (FT22)

   Control: Manual

   Initial State: Kinect is properly set up whilst facing an environment without any objects that may obstruct the view of a person. Software is running in realtime and is not detecting anyone in frame. A person is ready to walk into frame.

   Input: Realtime PCD from Kinect

   Output: The software locates the human outlines the whole human, while assuming where the legs should be.

   Test Case Derivation: The software would first expect to detect no one within the empty environment. Then when the human walks into frame, but only with the upper body being visible. The software is then able to recognize that there is a human in frame and draw a general box around them. This box will be a size of a general human. The human should be outlined as a whole and the software would assume where the missing parts would be.

   How test will be performed: The test will first have the empty environment. The tester would then ensure that the screen does not detect any humans in the frame. Then the human would walk in front of the sensor making sure that only their upper body is shown to the Kinect. The tester would then check the screen to see if the system outlines the human. The outline should assume the parts that are hidden and draw out a gaint box at the shape of a full visble human.

Offline Tests

1. Outside of Frame (FT23)

Control: Manual

Initial State: The system is running in offline mode which allows the user to upload a .pcd file.

Input: A file of a human that only has their hand in frame. (Out of Screen.pcd)

Output: The software locates where the human should be and draws an arrow point to the approximate off screen location.

Test Case Derivation: The file uploaded should have the hand of the human visbile within the environment and so when the file is uploaded it is expected that the system is able to detect that there is a skin point in the frame and draw an arrow pointing outward to where the human should be standing. This is because the system is able to see the hand as a skin point and know where it is region growing to.

How test will be performed: The correct file is uploaded into the system and the result should be displayed on screen. There should be an arrow pointing off to the side of the frame to where the human is standing. The screen should display this for the tester, who will check the file and match with the screen.

2. Legs Covered by Desk (FT24)

Control: Manual

Initial State: The system is running in offline mode which allows the user to upload a .pcd file.

Input: File of a human that is only showing the upper body, with their legs blocked by a desk or object. (PartiallyObstructedHalfBodyVisible.pcd)

Output: The software locates the human outlines the whole human, while assuming where the legs should be.

Test Case Derivation: The system reads the uploaded file and is expected to display a box around the whole body of the human because that is able to see the skin points and determine that the region growing is in frame. It is then able to draw a general human sized box that would assume the location of the whole human.

How test will be performed: The file gets uploaded in the offline mode. The system then processes the file and outputs the outline around the human. The outline should include where the software assumes the legs to be. The screen should display this for the tester, who will check the file and match with the screen.

4.1.3 Offline Processing

File Format Tests

1. .pcd File Test (FT31)

   Control: Manual

   Initial State: The system is running in offline mode which allows the user to upload a file.

   Input: Any .pcd file (take from Test Folder)

   Output: The software is able to read and analyze the data from the uploaded .pcd file and notifies the user that the file has been successfully uploaded

   Test Case Derivation: The software should be able to analyze and read the data from any given .pcd file. It should recognize that the correct file format has been uploaded.

   How test will be performed: The system will be in offline mode and the tester will upload a .pcd file. The feedback of the software will be inspected by the tester.

2. Incorrect File Format Test (FT32)

   Control: Manual

   Initial State: The system is running in offline mode which allows the user to upload a file.

   Input: Any file that isn't a .pcd file such as a .jpg file

   Output: The software is unable to read and analyze the data from the uploaded file and notifies the user that the file upload was unsuccessful

Test Case Derivation: The software should be not able to analyze and read the data from a file format that isn't .pcd. It should recognize that the incorrect file format has been uploaded.

How test will be performed: The system will be in offline mode and the tester will upload a file that isn't a .pcd file. The feedback of the software will be inspected by the tester.

4.1.4 Body Pose Variation Handling

1. Person not facing the sensor (FT41)

   Control: Manual

   Initial State: The system is running in real time with no object obstructing the view of the sensor. The person is facing away from the sensor.

   Input: Realtime PCD from Kinect

   Output: The system should be detected and their head, torso ,and limbs will be outlined on screen.

   Test Case Derivation: The outcome should be the same as the person looking at the sensor. It should not change the outcome and so the system is expected to behave the same as before and outline the human.

   How test will be performed: The person would stand in the open of the environment and face away from the sensor. The system will then outline the person's body parts and display the results on the screen. The tester can then verify if it is the correct output.

2. Person sitting on chair (FT42)

   Control: Manual

   Initial State: The system is running in real time with the chair obstructing the part of the person in the frame. The person is sitting facing away from the sensor.

   Input: Realtime PCD from Kinect

   Output: The system should be detected and their head and torso will be outlined on screen.

Test Case Derivation: Because the person is sitting it should not affect the outcome of the human. The chair is blocking part of the human and only the head and torso would be visible so that is the only thing that the sensor should be able to pick up.

How test will be performed: The person would sit in the frame facing away from the sensor. The system is then able to detect the part of the human visible and outline the head and torso. The tester is able to see this on the screen and verify the outcome.

3. Poking Head In and Out (FT43)

   Control: Manual

   Initial State: The system is running in real time with an object fully obstructing the person.

   Input: Realtime PCD from Kinect.

   Output: The system should be detected and their head whenever it is poked out.

   Test Case Derivation: The system is updating in real time and so it should be able to display the head outline whenever the sensor picks it up.

   How test will be performed: The person would be hidden behind the objects and then would repeatedly poke their head out. The system will then outline the head whenever it is in frame. The tester will observe this on the screen and see if the head is outlined at the correct times.

4.1.5 Integration with Kinect Sensor

Live Tests

1. Live Connection Test (FT51)

   Control: Manual

   Initial State: Kinect is turned on and connected to the computer that the software is running from.

   Input: Disconnection and reconnection to Kinect

Output: When the kinect is disconnected ,the system recognizes that the Kinect is disconnected and notifies the user. Once the kinect is connected again, the system recognizes that the kinect is connected and notifies the user

Test Case Derivation: The software should be able to integrate seamlessly with the Kinect meaning that if the connection to the Kinect is severed, the software should be aware and notify the user. Once the Kinect is reconnected, the system will immediately be aware that the Kinect is connected and notify the user making the connection seamless.

How test will be performed: The test begins with the Kinect already connected to the system. The tester would check to see that the Kinect is connected properly and then disconnect the connection. The tester will reconnect the Kinect to the computer that the software is running from and inspect the connection status of the Kinect in the system.

2. Live Kinect Data Test (FT52)

   Control: Manual

   Initial State: Kinect is turned on and connected to the computer that the software is running from.

   Input: Realtime PCD from Kinect

   Output: Stable stream of realtime PCD from Kinect visible to the user in the software

   Test Case Derivation: The software should be able to read PCD from the Kinect in realtime

   How test will be performed: The test begins with the Kinect already connected to the system and facing any environment. The tester would make hand gestures in front of the kinect sensor and then inspect to see if the PCD that the software reads accurately represents what is happening inside the frame of the Kinect in realtime.

4.1.6 Realtime Processing

1. Poking Head Out Test(FT61)

Control: Manual

Initial State: The system is running with an object fully obstructing the person.

Input: Realtime PCD from Kinect.

Output: The system should be detected and their head whenever it is poked out within a certain amount of time.

Test Case Derivation: The system should be updating in real time and so it should be able to display the head outline under a second.

How test will be performed: The person would be hidden behind the objects and then would repeatedly poke their head out. The system will then outline the head whenever it is in frame.The tester will measure the time by setting up a function to see how long the system takes to outline the person (from read input to having the coordinates for outline). If the time meets the minimal requirement to be considered a real time system.

4.2 Tests for Nonfunctional Requirements

Each subsection below covers a nonfunctional requirement of the system. All nonfunctional requirements are accounted for and each have its own tests such that all nonfunctional requirements are met.

4.2.1 Usability

1. Download Required Software Test (NFT11)

Control: Manual

Initial State: The user unfimilar with the project and no required downloads already on PC/Laptop.

Input: The user is given the Setup.md (link), a laptop without any necessary downloads, and the Kinect.

Output: By the end of the test the user should have the kinect working in a live scenario with all necessary technical components.

Test Case Derivation: The User is given all necessary technical elements to set up the software and they will be timed. A reasonable set up time would be 10-15 mins without download time being included.

How test will be performed: The User will be given all necessary components to run the software and just told to follow a set of instructions. They will be timed during the whole process and the timer will pause between downloads.

2. Upload a File Test (NFT12)

   Control: Manual

   Initial State: The system is running in offline mode. A file with an object partially obstructing the person will be uploaded.

   Input: A .pcd file that user is uploading

   Output: The file is uploaded without errors and the system displays the expected results.

   Test Case Derivation: The process would make sure that the User understands how to operate the other setting of the software.

   How test will be performed: The user is given a file to upload into the offline mode. So they start up the software and click the offline mode and upload the file. The sytem will then display the results.

4.2.2 Reliability

1. Same File Test (NFT21)

   Control: Automated

   Initial State: The system is running in offline mode. A file with an object partially obstructing the person will be uploaded.

   Input: A offline file with an object partially obstructing the person which will be uploaded 10 times.

   Output: The system predicts the location of the human and outlines the human with only a 5-10% difference between each run.

   Test Case Derivation: Because the file is the same, uploading it multiple different times should result in the system predicting the the

person's location within the screen similarly each time. In the one file uploaded the human will be partially covered and so each time it is uploaded the coordinates for the outline will be checked with the first run and it should fall into a range that is only 5-10 percent off of previous upload.

How test will be performed: This will be done by having a file be uploaded to the system 10 times in a loop and then checked if the coordinates off each upload fall into the expect range for the test to be a success.

4.2.3 Accuracy

1. Live Data Test (NFT31)

Control: Manual

Initial State: Kinect is properly set up whilst facing an environment with an object placed that is large enough to partially obstruct the view of a person. Software is running in realtime and is detecting the person in frame.

Input: Multiple Changed Positions

Output: The system outlines the approximate location of the human and the person testing would provide a rating from 1-10 each time. The final ratings are then averaged.

Test Case Derivation: The system is expected to produce an outline for the human that is relatively close to what can be seen on screen. The system should only be able to predict the human with a given range around the person standing in frame. The average for the 10 tests should be greater than 80%.

How test will be performed: The person would hold a certain position and then the tester would manually check if the outlined area fits into a certain range by viewing the screen. The person would then change their position and the tester does the same process over again. This would be repeated 10 times to measure accuracy. Each postition will be given a rating out of 10 (1 - completly wrong and 10 - on the person eact location) and then the final results will all be averaged. The score should be above 80% to be considered a pass.

4.3 Traceability Between Test Cases and Requirements

Table 2: Test and Requirements Traceability Matrix - See Section G.4 In SRS Report.

Tests	Requirement
(FT11)	[F411]
(FT12)	[F411]
(FT13)	[F411]
(FT14)	[F411]
(FT15)	[F411]
(FT16)	[F411]
(FT17)	[F411]
(FT18)	[F411]
(FT21)	[F412]
(FT22)	[F412]
(FT23)	[F412]
(FT24)	[F412]
(FT31)	[F413]
(FT32)	[F413]
(FT41)	[F414]
(FT42)	[F414]
(FT43)	[F414]
(FT51)	[F415]
(FT52)	[F415]
(FT61)	[NF416]
(NFT11)	[NF431]
(NFT12)	[NF431]
(NFT21)	[NF432]
(NFT22)	[NF432]
(NFT31)	[NF433]

5 Code Quality Tests

This section is included to outline the tests that will be run to ensure that the code is of high quality and matches industry standards.

5.1 Linting

1. Linting (CQT11)

   Control: Automated

   Initial State: The system has a C/C++ Linter hosted on Github Actions and has the Google C++ styling clang configuration file included.

   Input: Source Code

   Output: The system indicates which files do not match the Google C++ styling if the test fails. Otherwise, the test passes.

   Test Case Derivation: The system is expected to read all of the source files with the relevant C++ extensions (.cpp, .c, .hpp, .h, etc.) and ensure it matches the Google C++ styling as defined in the configuration file. The system should return either a successful or failed result and include thorough information of the automated test in either case.

   How test will be performed: This test will run automatically when performing changes on git through a git push or a git merge.

6 Unit Test Description

Our testing strategy for creating our unit tests is to select modules from the MIS that have the highest priority such as Input Processing and Point Cloud Data Structures. These modules represent important logics that we must ensure is reliable. Within our modules, we created test cases for the helper functions that deal with conversions and computations. Specifically, the functions that cannot be visibly tested and thus not covered by our manual tests will be tested through unit tests.

6.1 Unit Testing Scope

The other modules that are not outlined below are outside of the scope of unit tests due to the size of the functions within the modules or their low priority. Furthermore, the other modules are best tested through manual testing and live observations, which has been covered by our system tests.

6.2 Tests for Functional Requirements

6.2.1 Module Input Data Read

The Test used to cover Input Data Read is to ensure that the data transferred from the Kinect Sensor is correct to ensure that any filtering done will be done on a correct cv::Mat.

1. Test for convertPCLtoOpenCV (UT11)

   Type: Automatic

   Initial State: The test is correctly implemented within the test suite.

   Input: Sample PCL Point Cloud

   Output: The test checks that the RGB values, Depth Values, and Cloud Size are mapped correctly between file types (one to one) and returns an OK.

   Test Case Derivation: The test should be able to compare the sample cloud with the converted OpenCV image containers (cv::Mat) contaning rgb and depth values to check that the sample PCL point cloud has been properly converted to OpenCV.

   How test will be performed: Run the Unit Test suite through the unittest.cpp file and it will automatically display if the test passes through an OK output.

6.2.2 Module Point Cloud Data Structure

There are two tests here that will cover the Point Cloud Data Structure module one with a valid cloud and one with an invalid cloud. These two tests will cover the main scope of the clustering to ensure the module runs as expected.

1. Test for valid_cluster (UT21)

   Type: Automatic

   Initial State: The test is correctly implemented within the test suite.

   Input: Sample PCL cloud with a valid cluster

Output: The test correctly assesses the cluster to be valid and returns an OK

Test Case Derivation: The test should be able to confirm that the cluster that has been extracted is valid so that the correct human clustering can be determined.

How test will be performed: Run the Unit Test suite through the unittest.cpp file and it will automatically display if the test passes through an OK output.

2. Test for invalid_cluster (UT22)

   Type: Automatic

   Initial State: The test is correctly implemented within the test suite.

   Input: Sample PCL cloud with an invalid cluster

   Output: The test correctly assesses the cluster to be invalid and returns an OK

   Test Case Derivation: The test should be able to confirm that the cluster is invalid to make sure that it does not take in invalid human clustering.

   How test will be performed: Run the Unit Test suite through the unittest.cpp file and it will automatically display if the test passes through an OK output.

6.2.3 Module Input Processing

The two tests below will cover the Input Processing module by covering the main filtering methods of the module. Testing to ensure that voxel downsampling and noise removal is functioning properly will ensure that the cloud is filtered before the subsequent processing methods are run.

1. Test for voxel_downsampling (UT31)

   Type: Automatic

   Initial State: The test is correctly implemented within the test suite.

   Input: Sample PCL cloud with predetermined points

Output: Sample PCL cloud is downsampled and contains less points than the original cloud. The test returns an OK.

Test Case Derivation: The test should be able to confirm that the PCL cloud was properly downsampled and contains less points than the original cloud.

How test will be performed: Run the Unit Test suite through the unittest.cpp file and it will automatically display if the test passes through an OK output.

2. Test for removeNoise (UT32)

   Type: Automatic

   Initial State: The test is correctly implemented within the test suite.

   Input: Sample PCL cloud with noise added

   Output: The noise that was added to the sample PCL cloud is removed. The test checks that the noise has been removed and returns an OK.

   Test Case Derivation: The test should be able to confirm that the PCL cloud has had all of the added noise removed.

   How test will be performed: Run the Unit Test suite through the unittest.cpp file and it will automatically display if the test passes through an OK output.

6.3 Traceability Between Test Cases and Modules

Table 3: Module and Test Case Tracing

Test ID	Modules
UT11	M4
UT21	M8
UT22	M8
UT31	M9
UT32	M9

See section 7 in the Module Guide for more information on the modules.

References

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

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?

   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.