System Verification and Validation Plan for ESF

Alaap Grandhi

April 17, 2025

Revision History

Date	Version	Notes
February 27	1.0	Initial version of the VnV Plan
April 17	2.0	Updated according to Professor Smith's feedback

Contents

List of Tables

This document will describe the Verification and Validation (VnV) plan for the ESF project. It will outline the tests and procedures that will be taken to ensure the integrity of the code as well as to ensure that implementation meets the requirements described in the Software Requirements and Specifications (SRS) document (Grandhi (2025)).

1 General Information

This section provides general information about the ESF project.

1.1 Summary

The ESF project will involve constructing a library that enables users to both train and test multi-sensor 3D object detection networks on public autonomous driving datasets. Expanding upon the capabilities of the OpenPCDet repository (Team (2020)), it will serve as a foundation for further research into the field by presenting a method for LiDAR-Camera fusion-based object detection that better addresses alignment issues faced by prior methods.

The training and inference (testing) portions of ESF correspond to two different use-cases that require different considerations. During training, the library is intended to allow an experienced machine learning and perception engineer to train a LiDAR-Camera fusion network on an autonomous driving dataset of their choosing. During inference, the library is intended to allow a user or a greater system to predict the set of dynamic objects in an autonomous vehicle's surroundings using a trained model and sensor input.

1.2 Objectives

The primary objectives is to build confidence in the correctness of the software. As a routinely used and validated open-source repository, the base functionality of the OpenPCDet repository (Team (2020)) is considered to be error-free and thus its correctness is outside the scope of this document/project. The installation of the software and the extensions made to it are the components for which correctness will be verified.

Verifying the accuracy of ESF relative to existing baselines implemented in OpenPCDet (Team (2020)) is the secondary objective. Since the software

is intended to serve as a foundation for future research, its relative accuracy compared to state-of-the-art baseline methods will serve as a metric to judge how much it contributes to research in the field.

A tertiary objective is ensuring the usability of the software. Since the library will be extended and used for research purposes, it is important to minimize the effort needed to understand and modify the code as needed.

1.3 Relevant Documentation

The other relevant design documents as well as their relations to this document are listed as follows:

- SRS (Grandhi (2025)): Describes the problem the software solves, the software's goals, the relevant theoretical models, and the requirements for potential solutions. This document thereby establishes the setting and mathematical tools necessary for the validation and verification methods described in this document.
- More documents will be added as they are completed.

2 Plan

This section describes the approaches that will be taken to verify and validate the different documents and steps involved in the design process. Namely, it will state the members of the VnV team and will describe the steps that will be taken to verify the SRS (Grandhi (2025)), design, VnV plan, implementation. It will also describe the automating testing tools that will be used.

2.1 Verification and Validation Team

Name	Role	Description	Deliverables
Alaap Grandhi	Author	Write the VnV plan, carry out the testing detailed in it, fill in the VnV report, and validate the software and documents against the requirements during development.	PS&G, SRS, VnV plan, MG, MIS, Updated SRS, Updated VnV plan, Updated MG, Updated MIS, VnV report, Reflect and Trace document, Code
Spencer Smith	Project Supervisor/Course Instructor	Review and provide feedback on the VnV plan and report. Additionally, provide insight into better testing methods.	PS&G feedback, SRS feedback, VnV plan feedback, MG feedback, MIS feedback
Matthew Giamou	Masters Supervisor	Provide advice on the project in general.	SRS Comments/Guidance, MIS Comments/Guidance, Code Guidance and Suggestions
Bo Liang	Domain Expert Reviewer	Review and provide feedback on the VnV plan and report.	VnV plan github issues feedback, MG github issues feedback, MIS github issues feedback

Table 1: VnV team members and their responsibilities

2.2 SRS Verification Plan

The SRS will be verified in a manner that is inspired and informed by Liu (insert cite here). Specifically, I will verify my SRS through the following steps:

First, I will review my SRS document as a whole by doing the following:

1. Going through the document using the SRS checklist on my own and taking my own set of notes
2. Sharing the document with my Domain Expert and allowing them to create their own notes
3. Sitting down with my Domain Expert and asking a set of open-ended questions to my Domain Expert to see if the document was clear to them:
    - What general problem do you think SRS aims to address?
    - How could the object detection task focused on in the SRS aid Autonomous Driving?
    - What is your understanding of the difference between the training and inference pipelines?
    - What do you think some important characteristics of a solution would be here?
    - How did the mathematical notation used in the SRS affect the reading experience?
    - Beyond these, other questions will be asked based on their responses to the above
4. Comparing our notes on the document to create unified more complete set of notes

By separating my note-taking process from my Domain Expert's note-taking process, I will not run the risk of accidentally influencing or biasing their understanding of the document with my own thought process. This way, I can get a better idea of the SRS's merit as a standalone document.

Following this high-level review, I plan on going through the SRS line-by-line with my Masters Supervisor (Dr. Matthew Giamou). As someone with a strong history of publishing robotics-focused research, he should be able to

guide me on ways to make my SRS more useful for the research community I am targeting (since this project aims to empower future research). I will center this line-by-line review around the following questions:

1. Does the mathematical notation used align with standards in the field?
2. Does the problem outlined in the introduction reflect an actual gap in the field?
3. Do the project scope and system constraints seem reasonable given past research in the field?
4. Would it be reasonable to assume that the average computer-vision robotics conference attendee possesses the characteristics of the intended reader?
5. Do the training and inference pipelines outlined in the general system description seem reasonable?
6. Could you see researchers using this software?
7. Is all of the terminology defined correctly?
8. Do the assumptions made generally hold for autonomous robotics?
9. Do the mathematical models presented seen correct?
10. Are each of the instance models presented relevant to the task at hand?
11. Is the bounding box loss function a sufficient signal for training the model?
12. Does the mAP IOU-based evaluation metric accurately reflect the usefulness of the software?
13. Do the functional requirements align with prior research in the field?
14. Are the nonfunctional requirements sufficient for gauging the quality of the software?
15. Are there any forseeable likely or unlikely changes that I seem to be missing?

Once both of these have been conducted, I plan on taking suggestions for changes from my Domain Expert to refine my document. Specifically, this feedback will be provided in the form of Github issues. Following the creation of such issues, I will review the document with these suggestions in mind and will make changes where appropriate (or explain why such a change is unnecessary if not). Then, the Github issues will be responded to and closed. Due to time constraints, a prototype will not be used for SRS verification.

2.3 Design Verification Plan

To verify the design, I have constructed the following checklist detailing the aspects of the design that the Domain Expert Reviewer and optionally the Masters and Project Supervisors (if they have time) will evaluate during regular discussions:

- Does the design explicitly or implicitly address all requirements described in the SRS document? (Grandhi (2025))
- Are the interfaces between modules of the design clear and in-line with the modular structure of the OpenPCDet repository? (Team (2020))
- Given the overall design description, can the Domain Expert Reviewer accurately roughly predict the downstream code structure?
- Are any aspects of the design ambiguous?
- Does the design leverage existing functionality from the OpenPCDet repository (Team (2020)) where applicable?
- Do all of the design components contribute towards meeting a requirement?
  - If not, is this due to a redundancy in the design or something missing in the SRS document? (Grandhi (2025))

2.4 Verification and Validation Plan Verification Plan

Manual tests and the overall structure of the VnV plan will be reviewed by the Project Supervisor and the Domain Expert Reviewer. This will be done

to both ensure that the document adheres to the template and to ensure that the tests presented sufficiently cover the requirements described in the SRS document (Grandhi (2025)). Automated tests will be verified by mutation testing where possible.

2.5 Implementation Verification Plan

The implementation will simply be verified through semi-regular code walkthroughs with my Domain Expert. These walkthroughs will consist of me going through my MIS modules one-by-one and displaying these side-by-side with the code sections corresponding to them. By explaining the code that is meant to realize each module in a step-by-step manner, my Domain Expert will be able to tell me if my code seems incorrect or does not match its specification at any point. Additionally, the tests described in Section 3 will serve as a basis for verifying that the implementation meets the requirements detailed in the SRS document. (Grandhi (2025))

2.6 Automated Testing and Verification Tools

There are a few different automated tools that will be used for testing. Since the library will be written in Python (the OpenPCDet library (Team (2020)) is written in Python), PyFlakes will be used as an error linter to discover simple logical errors. Linters like flake8 that enforce style convention will not be used as the base OpenPCDet repository (Team (2020)) does not adhere to any particular style convention (it is designed with research and quick development as the primary objective) and it would be out of scope to refactor that. Additionally, the automated unit tests below will be implemented in pytest and will be set to automatically run on push using github actions. Code coverage will not be considered for this project since by nature different components are meant to be called according to the training configuration passed in and it would overly tedious to routinely test over all possible input configurations.

2.7 Software Validation Plan

In order to validate my Software, I plan on running it over the evaluation set of NuScenes and saving a set of visualizations for each input. Specifically,

these visualizations will display input pointclouds alongside predicted bounding boxes. From there, my Domain Expert and I will visually inspect these to see if they provide a sufficient and accurate representation of the objects in the Autonomous Vehicle's surroundings. Validation of the requirements themselves will only briefly be covered during discussions with my Masters Supervisor.

3 System Tests

This section details a set of tests that shall be sufficient to ensure that ESF meets the requirements detailed in the SRS document. Additional tests may be added as the SRS document is refined.

3.1 Tests for Functional Requirements

The subsections below detail the tests that will be used to ensure that ESF meets the functional requirements.

3.1.1 Input Format Testing

These tests will ensure that in inference, the system is able to process all the necessary file formats for the camera images and LiDAR pointclouds.

Camera Image Input Testing

1. test-camera-valid1→4

   Control: Automatic

   Initial State: The inference pipeline initialized with a pre-defined trained model and configuration loaded. A clean, conforming LiDAR pointcloud in the pcd.bin format from the NuScenes dataset is also expected to have already been loaded (this will be handpicked from NuScenes/v1.0-trainval/samples/LIDAR_TOP and put into the test's directory).

   Input: A valid (subject to input constraints) multiview camera image set stored in each of the file formats specified in requirement R1 (one test case for file format).

Output: The inference pipeline should run as expected without errors and output a set of predicted bounding boxes. The actual values for these bounding boxes do not matter as long as the shape of the output is as expected and the constraints on the values (i.e. height of a bounding box is non-negative) hold.

Test Case Derivation: For all the file formats specified in requirement R1, the inference pipeline should run without issues. All other aspects are kept constant between the tests to ensure that the test can detect any issues with specific image file formats.

How test will be performed: This test will be automatically run through pytest. The configuration, pretrained model, LiDAR pointcloud will all be manually generated beforehand and then automatically accessed by pytest. The same set of valid multiview camera images will simply be converted to the different file types beforehand to be automatically accessed by pytest on github push.

2. test-camera-invalid

   Control: Automatic

   Initial State: The inference pipeline initialized with a pre-defined trained model and configuration loaded. A clean, conforming LiDAR pointcloud in the pcd.bin format from the NuScenes dataset is also expected to have already been loaded (this will be handpicked from NuScenes/v1.0-trainval/samples/LIDAR_TOP and put into the test's directory).

   Input: A valid (subject to input constraints) multiview camera image set stored in a file format that is not expected to be handled by the program (a binary image).

   Output: A invalid-input error is expected to be thrown by the program.

   Test Case Derivation: The inference pipeline should not even try to run for any file format not specified in requirement R1. This is to test for cases where images are input in unforeseen or invalid formats, and makes sure that the system's response to such input types is well-defined.

   How test will be performed: This test will be automatically run through pytest on github push in much the same way as the valid tests. The

only difference is that it will load a set of multiview camera images stored in a binary format.

3. test-camera-small

   Control: Automatic

   Initial State: The inference pipeline initialized with a pre-defined trained model and configuration loaded. A clean, conforming LiDAR point-cloud in the pcd.bin format from the NuScenes dataset is also expected to have already been loaded (this will be handpicked from NuScenes/v1.0-trainval/samples/LIDAR_TOP and put into the test's directory).

   Input: A multiview camera image set with a height of one and a width of one (too small) stored in the valid JPEG format.

   Output: A invalid-input error is expected to be thrown by the program.

   Test Case Derivation: The inference pipeline should not even try to run for an image that is does not match the input constraint. This is to test for cases where improperly-sized images are input, and makes sure that the system's response to such input types is well-defined.

   How test will be performed: This test will be automatically run through pytest on github push in much the same way as the valid tests. The only difference is that it will load a set of multiview camera images of size 1x1.

4. test-camera-wrongnumviews

   Control: Automatic

   Initial State: The inference pipeline initialized with a pre-defined trained model and configuration loaded. A clean, conforming LiDAR point-cloud in the pcd.bin format from the NuScenes dataset is also expected to have already been loaded (this will be handpicked from NuScenes/v1.0-trainval/samples/LIDAR_TOP and put into the test's directory).

   Input: A multiview camera image set with ten views (where NuScenes expects six) stored in the valid JPEG file format.

Output: A invalid-input error is expected to be thrown by the program.

Test Case Derivation: The inference pipeline should not even try to run for a multiview image with a different number of views than the training dataset (NuScenes). This is to test for cases where the wrong amount of image views are input, and makes sure that the system's response to such input types is well-defined.

How test will be performed: This test will be automatically run through pytest on github push in much the same way as the valid tests. The only difference is that it will load a set of multiview camera images with ten views instead of the normal six.

LiDAR Pointcloud Input Testing

1. test-lidar-valid1→4

   Control: Automatic

   Initial State: The inference pipeline initialized with a pre-defined trained model and configuration loaded. A clean, conforming set of multiview camera images in the JPEG format from the NuScenes Dataset is also expected to have already been loaded (this set will be handpicked from NuScenes/v1.0-trainval/samples/CAM_* with one corresponding image from each CAM folder).

   Input: A valid (subject to input constraints) LiDAR pointcloud stored in each of the file formats specified in requirement R2 (one test case for each format).

   Output: The inference pipeline should run as expected without errors and output a set of predicted bounding boxes. The actual values for these bounding boxes do not matter as long as the shape of the output is as expected and the constraints on the values (i.e. height of a bounding box is non-negative) hold.

   Test Case Derivation: For all the file formats specified in requirement R2, the inference pipeline should run without issues. All other aspects are kept constant between the tests to ensure that the test can detect any issues with specific image file formats.

   How test will be performed: This test will be automatically run through pytest. The configuration, pretrained model, multiview camera image

set will all be manually generated beforehand and then automatically accessed by pytest. The same LiDAR pointcloud will simply be converted to the different file types beforehand to be automatically accessed by pytest on github push.

2. test-lidar-invalid

   Control: Automatic

   Initial State: The inference pipeline initialized with a pre-defined trained model and configuration loaded. A clean, conforming set of multiview camera images in the JPEG format from the NuScenes Dataset is also expected to have already been loaded (this set will be handpicked from NuScenes/v1.0-trainval/samples/CAM_* with one corresponding image from each CAM folder).

   Input: A LiDAR pointcloud stored in the binary file format (since this format is not one of the ones specified in R2).

   Output: A invalid-input error is expected to be thrown by the program.

   Test Case Derivation: The inference pipeline should not even try to run for any file format not specified in requirement R2. This is to test for cases where pointclouds are input in unforeseen or invalid formats, and makes sure that the system's response to such input types is well-defined.

   How test will be performed: This test will be automatically run through pytest on github push in much the same way as the valid tests. The only difference is that it will load a LiDAR pointcloud stored in a binary format.

3.1.2 Tests for Correct Optimization

The test below will ensure that ESF is being optimized correctly.

Model Training Consistency Testing

1. test-consistent

   Control: Manual

Initial State: A set of pre-configured baseline methods will be set up for training. Additionally, the proposed method built on the OpenPCDet (Team (2020)) platform will be set up for training.

Input: A sufficiently small training dataset that has been predetermined to produce similar training accuracies across the baseline methods (corresponding to an mAP standard deviation below a threshold that will be decided on with my Domain Expert).

Output: When all the models are trained and subsequently evaluated on the same small training dataset, the proposed method shall not fall below the mean mAP of the baseline methods by more than a given absolute threshold that will be decided on with my Domain Expert.

Test Case Derivation: There is no true way to test if the combined bounding box is exactly minimized considering the randomness inherent in machine learning techniques and floating point computation. Rather, since the baselines are assumed to roughly minimize the bounding box loss on the training dataset, consistency with their results should be sufficient for verifying this. Additionally, by evaluating on the same dataset that was trained on, we can ensure that the ADAM optimization is minimizing the loss for the training subset.

How test will be performed: This test will be manually run due to the time-intensive nature of model training. Moreover, it will not be regularly run but rather only run when major changes to ESF are made. It will also not be run until the model is in a trainable state as a whole and the other tests have all passed.

Note that since ADAM will be directly imported from pytorch's implementation, its validity will not be verified. Rather, the code walkthroughs mentioned above in section 2.5 will aid in verifying that it has been used correctly.

3.1.3 Tests for Visualization

The below test will verify that the visualization produced during the inference phase is correct.

Visualization Testing

1. test-viz-correct

   Control: Automatic

   Initial State: Any components aside from the inputs that are needed for the visualization pipeline to run.

   Input: A set of pre-configured bounding boxes will be loaded alongside a predetermine LiDAR pointcloud. Additionally, a set of viewing positions and resolutions to view the produced pointclouds from.

   Output: The output pointcloud visualizations with bounding boxes should look identical to pre-generated verified visualizations of the same scene and bounding boxes. These pre-generated visualizations will be created in the future and carefully examined to ensure correctness before setting up the test.

   Test Case Derivation: This is really just using the method of manufactured solutions to verify that the visualization pipeline is working as expected without bugs. A correct and matching visualization here does not guarantee a correctly functioning visualization pipeline but it is a necessary condition.

   How test will be performed: This test will automatically be run through pytest with the LiDAR pointcloud, bounding boxes, and verified visualizations being generated beforehand. Matplotlib.testing.compare.compare_images will be run to compare each generated visualization with the corresponding verified visualization.

3.2 Tests for Nonfunctional Requirements

The subsections below will detail the tests that will ensure that ESF meets the nonfunctional requirements.

3.2.1 Tests for Accuracy

The test below will ensure that ESF is at least relatively as accurate as the BEVFusion baseline (Liang et al. (2022)).

Accuracy Testing

1. test-mAP

   Type: Manual

   Initial State: The BEVFusion model will be loaded with the BEVFusion configuration currently present in the OpenPCDet repository (Team (2020)). Additionally, ESF will be loaded with some chosen configuration.

   Input/Condition: A training and validation set from the nuScenes dataset (Caesar et al. (2020)).

   Output/Result: After both models have been trained on the nuScenes training set (Caesar et al. (2020)), their mAP metrics on the validation set will be returned. The proposed solution's validation mAP should not fall below the BEVFusion mAP by more than a threshold that will be determined with my Domain Expert.

   How test will be performed: Since model training is time-intensive, this test will mainly be conducted whenever major changes to ESF are made. The threshold will only serve as a guideline and after each run of this test, a discussion will be done with my Domain Expert and Masters Supervisor to determine if the result is satisfactory.

While Waymo (Sun et al. (2020))could be tested for as well, it is a much larger dataset that would take quite long to train on. Moreover, evaluating this test for the nuScenes dataset (Caesar et al. (2020)) alone should provide almost as much insight as evaluating on both (since in existing research, better performing models tend to do better across both datasets).

3.2.2 Tests for Understandability

Understandability Testing Rather than use a fixed test for understandability, it will be tested closer to the end of the project through code walkthroughs. Specifically, I will provide a demonstration of how to use the library to my Domain Expert and optionally Project and Masters supervisors (if they have time). I will ask them the following questions afterwords to gauge how easy to understand the code is.

1. How do you think the code works at a high level?

2. If you had to describe what x (to be determined after implementation) class of submodules does, how would you do so?

3. If you had to learn how to use this code with nothing but the documentation present in the repository and the code itself, how long do you think it would take you to do so?

Afterwords, I will test further test their understanding of the code by letting them play around with it and getting them to perform the following tasks with assistance as necessary.

1. Train a model using x (to be determined after implementation) configuration.

2. Evaluate a pretrained model on the nuScenes validation dataset (Caesar et al. (2020)).

3. Modify the configuration for this pretrained model to change the post-processing prediction range.

Through the code walkthrough and their subsequent interaction with it, I hope to get at least a rough idea of how understandable the code is.

3.2.3 Tests for Installability

Installability Testing Since I implemented the code, it would not really make sense to test the installability of ProgName myself. This is because I am familiar with the bugs that I faced when initially installing the code and thus would be able to troubleshoot on the fly where the user might not be able to. Rather, I plan to sit down with either my domain expert or Masters Supervisor (Dr. Matthew Giamou) and have them perform the following steps with minimal guidance:

1. Clone the repository from github to get a fresh copy by running "git clone URL"

2. Create a conda environment from the provided environment.yml file by running "conda env create -f environment.yml" from the environment subfolder

3. Setup the repository using OpenPCDet's setup.py file by running "python setup.py develop" from the base directory

4. Follow OpenPCDet's dataset setup guide for the NuScenes Dataset as explained here
   (a) Since the full dataset is too large, I would tell them to install the mini version instead
   (b) If they do not want to have to create a NuScenes account to download the dataset, they can simply download the mini set by running "wget https://d36yt3mvayqw5m.cloudfront.net/public/v1.0/v1.0-mini.tgz"
   (c) The pip install step in OpenPCDet's dataset setup guide should be ignored as it is already installed in the conda environment
   (d) The multi-modal data infos should be generated rather than the lidar-only infos
5. Install the pretrained BEVFusion model and place it in the pretrained_models folder
6. Install the Pretrained NuImages Swin-Transformer and place it in the pretrained_models folder
7. Evaluate this pretrained model on the nuScenes mini validation dataset (Caesar et al. (2020)) (I will add a guide for this in the Github's README file)

If they are able to perform all these steps without errors (or with minimal errors that a researcher in the field would be able to troubleshoot), then I would consider this test to have passed. Otherwise, the steps that fail would provide insight into how the installability of the software could be improved.

3.2.4 Tests for Performance

Performance Testing Since the amount of time inference and training runs take to execute are highly dependant on factors such as GPU, ram, hyperparameter, and background-processes. As such, rather than try to compare runtimes between different methods or compare to existing papers (which run on much more powerful setups with 4 to 8 GPUs), I aim to simply print the amount of time the model takes to for each training batch and each inference batch. With this information, I plan on having a discussion with my Masters' Supervisor (Dr. Matthew Giamou) to discuss if the results seem reasonable.

3.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4	NFR1	NFR2
Camera Image Input Testing	X					
LiDAR Pointcloud Input Testing		X				
Model Training Consistency Testing			X			
Visualization Testing				X		
Accuracy Testing					X	
Understandability Testing						X

Table 2: Traceability Matrix Showing the Connections Between Requirements and Tests

4 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

4.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

4.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

4.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

4.2.2 Module 2

...

4.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

4.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.3.2 Module ?

...

4.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

Alaap Grandhi. System requirements specification. https://github.com/alaapgrandhi/equivariant-sensor-fusion/tree/main/docs/SRS/SRS.pdf, 2025.

Tingting Liang, Hongwei Xie, Kaicheng Yu, Zhongyu Xia, Zhiwei Lin, Yongtao Wang, Tao Tang, Bing Wang, and Zhi Tang. Bevfusion: A simple and robust lidar-camera fusion framework. *Advances in Neural Information Processing Systems*, 35:10421–10434, 2022.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020.

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

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

5.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]