Module Interface Specification for ProgName

Team #, Team Name
Student 1 name
Student 2 name
Student 3 name
Student 4 name

March 18, 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 [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

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 ProgName.

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 (-∞, ∞)

The specification of ProgName 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, ProgName 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	
Behaviour-Hiding Module	Config Loading Module Data Loading Module Model Module Checkpointing Module Training Module Inference Module Loss Module Evaluation Module Optimization Module Data Processing Module
Software Decision Module	Plotting Module PyTorch Module Logging Module

Table 1: Module Hierarchy

6 MIS of Training Module

6.1 Module

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
trainEpoch	Dataloader, Model, Loss, Optimizer	-	-
train	Dataloader, Model, Loss, Optimizer	-	-

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Environment Variables

File System

6.4.3 Assumptions

6.4.4 Access Routine Semantics

train():

- transition: Saving model as a checkpoint on the file system.
- output: N/A
- exception: N/A

6.4.5 Local Functions

Is trainEpoch a local function? It updates the model using the optimizer, dataset, and loss function.

7 MIS of Inference Module

7.1 Module

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
infer??	Dataloader, Model	-	-

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

Screen

7.4.3 Assumptions

7.4.4 Access Routine Semantics

infer():

- transition: Displaying the predicted bounding boxes in the LiDAR pointcloud on the screen.
- output: N/A
- exception: N/A

7.4.5 Local Functions

Not yet done

8 MIS of Model Module

8.1 Module

8.2 Uses

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	Configuration File	-	-
forward	Camera Images, LiDAR Data	Bounding Boxes	-

8.4 Semantics

8.4.1 State Variables

Model parameters, Hyperparameters

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

8.4.4 Access Routine Semantics

init():

- transition: Sets the hyperparameters according to the configuration file.
- output: N/A
- exception: N/A

forward():

- transition: N/A
- output: Outputs predicted bounding boxes for the input Camera Images and LiDAR Pointcloud.
- exception: N/A

8.4.5 Local Functions

Not yet done

9 MIS of Loss Module

9.1 Module

9.2 Uses

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	Configuration File	-	-
forward	Predicted Bounding Boxes, Ground Truth Bounding Boxes	Loss "Value"	-

9.4 Semantics

9.4.1 State Variables

Hyperparameters

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

9.4.4 Access Routine Semantics

init():

- transition: Sets the hyperparameters according to the configuration file.
- output: N/A
- exception: N/A

forward():

- transition: N/A
- output: Outputs the loss between the predicted and ground truth bounding boxes.
- exception: N/A

9.4.5 Local Functions

Not yet done

10 MIS of Data Loading Module

10.1 Module

10.2 Uses

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	Configuration File	-	-

Iterator stuff?

10.4 Semantics

10.4.1 State Variables

Dataset, Dataloader

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

10.4.4 Access Routine Semantics

init():

- transition: Sets up the dataset and the data loader using the configuration.
- output: N/A
- exception: N/A

10.4.5 Local Functions

Not yet done

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.

11 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

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

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

4. 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?

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

6. 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)