Module Interface Specification for ESF

Alaap Grandhi

April 17, 2025

1 Revision History

Date	Version	Notes
Mar 21	1.0	First Draft
Mar 22	1.1	Added units/type
Apr 17	2.0	Incorporated Professor Smith's feedback

2 Symbols, Abbreviations and Acronyms

See SRS Documentation

2.1 Table of Symbols

The following table of symbols is copied over from the SRS document to clarify symbols used in this document. Not all symbols in here show up in this document but they are put here nonetheless for consistency with the SRS.

symbol	unit	description
f_x	$\mathbb{R}_{\geq 0}$ pixels	The focal length of the given camera in the x direction.
f_y	$\mathbb{R}_{\geq 0}$ pixels	The focal length of the given camera in the y direction.
c_x	$\mathbb{R}$ pixels	The principal point of the given camera in the x direction.
c_y	$\mathbb{R}$ pixels	The principal point of the given camera in the y direction.
s	$\mathbb{R}$ pixels	The skew of the given camera representing how far from perpendicular the x and y directions are.
$\mathbf{K}$	$\mathbb{R}^{3\times 3}$ pixels	The intrinsic matrix of the given camera representing how 3D points are projected onto the 2D image plane.
n_p	$\mathbb{R}_{\geq 0}$	The number of pixels in a given input image.
$\mathbf{P_i}$	$\mathbb{R}^{2\times n_p}$ pixels	The set of x,y coordinates for each of the n_p pixels in a given image.
$\mathbf{P_c}$	$\mathbb{R}^{3\times n_p}$ m	The set of back-projected x,y,z coordinates for each of the n_p pixels in a given image. This is just X_c, Y_c, and Z_c concatenated.
$\mathbf{X_c}$	$\mathbb{R}^{n_p}$ m	The set of back-projected x coordinates for each of the n_p pixels in a given image.
$\mathbf{Y_c}$	$\mathbb{R}^{n_p}$ m	The set of back-projected y coordinates for each of the n_p pixels in a given image.
$\mathbf{Z_c}$	$\mathbb{R}^{n_p}$ m	The set of back-projected z coordinates for each of the n_p pixels in a given image.
D	$\mathbb{R}^3 \to \mathbb{R}^3$	The distortion model of the given camera (accounts for lens effects).
$\mathbf{P_w}$	$\mathbb{R}^{3\times n_p}$ m	The set of back-projected, world reference frame x,y,z coordinates for each of the n_p pixels in a given image.
$\mathbf{T_{c,w}}$	$\mathbb{R}^{4\times 4}$	The extrinsics matrix that maps x,y,z points from the given camera's frame of reference to the world reference frame.
$\mathbf{R_{c,w}}$	$\mathbb{R}^{3\times 3}$	The rotation matrix between the given camera's frame of reference and the world reference frame.
$\mathbf{t_{c,w}}$	$\mathbb{R}^3$	The translation vector between the given camera's frame of reference and the world reference frame.

$\mathbb{P}$	$\mathbb{R}^{n_c}$	A predicted probability distribution over the set of n_c discrete classes.
α_{gd}	$\mathbb{R}$	A hyperparameter for gradient descent that controls how large each gradient update is.
L	$\mathbb{R}^{n_\theta} \to \mathbb{R}$	A loss function that maps the parameters of a model to a single estimated loss value.
n_θ	$\mathbb{R}_{\geq 0}$	The number of learnable parameters for a given model.
θ	$\mathbb{R}^{n_\theta}$	The learnable parameters for a given model, where the number of parameters is n_θ.
TP	$\mathbb{N}_0$	The number of true positives for a given classification task. That is, the number of positive predictions that corresponded to positive ground truth.
FP	$\mathbb{N}_0$	The number of false positives for a given classification task. That is, the number of positive predictions that corresponded to a negative ground truth.
FN	$\mathbb{N}_0$	The number of false negatives for a given classification task. That is, the number of positive ground truths that were not predicted to be positive.
$\mathbf{d_c}$	$\mathbb{R}^{n_p}$ m	The estimated depth values for each of the pixels in a given image.
g	$\mathbb{R}_{\geq 0}$	The number of scalar values/parameters used to define a bounding box (dataset-dependent). For the nuScenes dataset, this is 10 corresponding to the 3D center position, height, width, length, and a quaternion angle representation.
n_b	$\mathbb{R}_{\geq 0}$	The number of ground truth bounding boxes.
$n_{\hat{b}}$	$\mathbb{R}_{\geq 0}$	The number of predicted bounding boxes.
n_c	$\mathbb{R}_{\geq 0}$	The number of object classes (typically 3 corresponding to cars, pedestrians, and cyclists).
$\mathbf{B}$	$\mathbb{R}^{g \times n_b}$	The set of ground-truth bounding box attributes (i.e. center, height, width, etc.), where there are n_b ground truth bounding boxes.
$\hat{\mathbf{B}}$	$\mathbb{R}^{g \times n_{\hat{b}}}$	The set of predicted bounding box attributes, where there are $n_{\hat{b}}$ predicted bounding boxes.
$\hat{\mathbf{C}}_{\text{dist}}$	$\mathbb{R}^{n_c \times n_{\hat{b}}}$	The set of predicted bounding box class probability distributions.
$\hat{\mathbf{c}}$	$\mathbb{N}_0^{n_{\hat{b}}}$	The set of predicted bounding box classes.
$\mathbf{c}$	$\mathbb{N}_0^{n_b}$	The set of ground truth bounding box classes.
α_{fl}	$\mathbb{R}$	A hyperparameter for controlling the influence of the focal loss term.

γ	$\mathbb{R}$	A hyperparameter for controlling the effect of easy examples on the focal loss. In practice, this is class specific and is set to the inverse class frequency.
β_1	$\mathbb{R}$	A hyperparameter for changing the inertia of the first moment in ADAM.
β_2	$\mathbb{R}$	A hyperparameter for changing the inertia of the second moment in ADAM.
τ_{IoU}	$\mathbb{R}$	An IoU threshold that is used to determine whether a predicted and a ground truth bounding box sufficiently overlap to be considered paired.
h	$\mathbb{R}_{\geq 0}$	The height of a given input image.
w	$\mathbb{R}_{\geq 0}$	The width of a given input image.
n_l	$\mathbb{R}_{\geq 0}$	The number of points in an input lidar pointcloud.
$\mathcal{I}$	$\mathbb{R}^{h \times w}$	An input image with height h and width w.
$\mathcal{P}$	$\mathbb{R}^{3 \times n_l}$	An input lidar pointcloud with n_l points.
n_{cams}	$\mathbb{R}$	The number of cameras on the autonomous vehicle (and thus the number of images in each multi-view image).

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**
2.1 Table of Symbols . . . ii

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **1**

6 MIS of Training Module **3**
6.1 Module . . . 3
6.2 Uses . . . 3
6.3 Syntax . . . 3
6.3.1 Exported Constants . . . 3
6.3.2 Exported Access Programs . . . 3
6.4 Semantics . . . 3
6.4.1 State Variables . . . 3
6.4.2 Environment Variables . . . 4
6.4.3 Assumptions . . . 4
6.4.4 Access Routine Semantics . . . 4

7 MIS of Inference Module **5**
7.1 Module . . . 5
7.2 Uses . . . 5
7.3 Syntax . . . 5
7.3.1 Exported Constants . . . 5
7.3.2 Exported Access Programs . . . 5
7.4 Semantics . . . 5
7.4.1 State Variables . . . 5
7.4.2 Environment Variables . . . 5
7.4.3 Assumptions . . . 5
7.4.4 Access Routine Semantics . . . 5

8 MIS of Evaluation Module **7**
8.1 Module . . . 7
8.2 Uses . . . 7
8.3 Syntax . . . 7
8.3.1 Exported Constants . . . 7
8.3.2 Exported Access Programs . . . 7
8.4 Semantics . . . 7

8.4.1 State Variables . . . 7
8.4.2 Environment Variables . . . 7
8.4.3 Assumptions . . . 7
8.4.4 Access Routine Semantics . . . 7
8.4.5 Local Functions . . . 7

9 MIS of Logger Module **8**

10 MIS of Model Module **9**
10.1 Template . . . 9
10.2 Uses . . . 9
10.3 Syntax . . . 9
10.3.1 Exported Constants . . . 9
10.3.2 Exported Access Programs . . . 9
10.4 Semantics . . . 9
10.4.1 State Variables . . . 9
10.4.2 Environment Variables . . . 9
10.4.3 Assumptions . . . 9
10.4.4 Access Routine Semantics . . . 9
10.4.5 Local Functions . . . 10

11 MIS of Loss Module **11**
11.1 Template . . . 11
11.2 Uses . . . 11
11.3 Syntax . . . 11
11.3.1 Exported Constants . . . 11
11.3.2 Exported Access Programs . . . 11
11.4 Semantics . . . 11
11.4.1 State Variables . . . 11
11.4.2 Environment Variables . . . 11
11.4.3 Assumptions . . . 11
11.4.4 Access Routine Semantics . . . 11
11.4.5 Local Functions . . . 12

12 MIS of Optimizer Module **13**
12.1 Template . . . 13
12.2 Uses . . . 13
12.3 Syntax . . . 13
12.3.1 Exported Constants . . . 13
12.3.2 Exported Access Programs . . . 13
12.4 Semantics . . . 13
12.4.1 State Variables . . . 13
12.4.2 Environment Variables . . . 13

12.4.3 Assumptions . . . 13
12.4.4 Access Routine Semantics . . . 13
12.4.5 Local Functions . . . 14

13 MIS of Plotting Module **15**
13.1 Module . . . 15
13.2 Uses . . . 15
13.3 Syntax . . . 15
13.3.1 Exported Constants . . . 15
13.3.2 Exported Access Programs . . . 15
13.4 Semantics . . . 15
13.4.1 State Variables . . . 15
13.4.2 Environment Variables . . . 15
13.4.3 Assumptions . . . 15
13.4.4 Access Routine Semantics . . . 15
13.4.5 Local Functions . . . 16

14 MIS of Checkpoint Module **17**
14.1 Module . . . 17
14.2 Uses . . . 17
14.3 Syntax . . . 17
14.3.1 Exported Constants . . . 17
14.3.2 Exported Access Programs . . . 17
14.4 Semantics . . . 17
14.4.1 State Variables . . . 17
14.4.2 Environment Variables . . . 17
14.4.3 Assumptions . . . 17
14.4.4 Access Routine Semantics . . . 17
14.4.5 Local Functions . . . 18

15 MIS of Data Module **19**
15.1 Module . . . 19
15.2 Uses . . . 19
15.3 Syntax . . . 19
15.3.1 Exported Constants . . . 19
15.3.2 Exported Access Programs . . . 19
15.4 Semantics . . . 19
15.4.1 State Variables . . . 19
15.4.2 Environment Variables . . . 19
15.4.3 Assumptions . . . 19
15.4.4 Access Routine Semantics . . . 19
15.4.5 Local Functions . . . 20

16 MIS of Equivariant Layers Module **21**
16.1 Module . . . 21
16.2 Uses . . . 21
16.3 Syntax . . . 21
16.3.1 Exported Constants . . . 21
16.3.2 Exported Access Programs . . . 21
16.4 Semantics . . . 21
16.4.1 State Variables . . . 21
16.4.2 Environment Variables . . . 22
16.4.3 Assumptions . . . 22
16.4.4 Access Routine Semantics . . . 22
16.4.5 Local Functions . . . 22

17 MIS of OpenPCDet Layers Module **23**

18 MIS of PyTorch Module **24**

19 MIS of Config Module **25**
19.1 Template . . . 25
19.2 Uses . . . 25
19.3 Syntax . . . 25
19.3.1 Exported Constants . . . 25
19.3.2 Exported Access Programs . . . 25
19.4 Semantics . . . 25
19.4.1 State Variables . . . 25
19.4.2 Environment Variables . . . 25
19.4.3 Assumptions . . . 25
19.4.4 Access Routine Semantics . . . 25
19.4.5 Local Functions . . . 26

20 MIS of Data Processing Module **27**
20.1 Module . . . 27
20.2 Uses . . . 27
20.3 Syntax . . . 27
20.3.1 Exported Constants . . . 27
20.3.2 Exported Access Programs . . . 27
20.4 Semantics . . . 27
20.4.1 State Variables . . . 27
20.4.2 Environment Variables . . . 27
20.4.3 Assumptions . . . 27
20.4.4 Access Routine Semantics . . . 28
20.4.5 Local Functions . . . 28

21 Appendix **30**

3 Introduction

The following document details the Module Interface Specifications for ESF: Equivariant Learning-based Camera-LiDAR 3D object detection in Autonomous Driving.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/alaapgrandhi/equivariant-sensor-fusion.

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 ESF.

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 ESF 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, ESF 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.

In addition to the above, str is used to denote a sequence of chars and * is used to denote the convolution operator.

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 Module
	Data Module
	Model Module
	Checkpoint Module
	Training Module
	Inference Module
	Loss Module
	Evaluation Module
	Optimization Module
	Data Processing Module
	Equivariant Layer Module
	OpenPCDet Layer 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

1. PyTorch Dataloader (PyTorch Documentation)
2. PyTorch Optimizer (PyTorch Documentation)
3. PyTorch Loss (PyTorch Documentation)
4. Model (10)
5. Checkpoint (14)
6. Logger (9)

6.3 Syntax

6.3.1 Exported Constants

- NUM_EPOCHS (R): The number of epochs to train for. For now, this constant equals 30, but this may be refined in future versions.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
setup	-	-	-
train	-	-	-
eval	-	-	-

6.4 Semantics

6.4.1 State Variables

- evalLoader (PyTorch Dataloader)
- trainLoader (PyTorch Dataloader)
- optim (Optimizer ADT)
- model (Model ADT)
- loss (Loss ADT)

6.4.2 Environment Variables

6.4.3 Assumptions

6.4.4 Access Routine Semantics

setup():

- transition: Creates the optim, model, and loss abstract objects by calling their respective constructors (For the Optimizer 12, Model 10, and Loss 11 Modules respectively). The trainLoader abstract object is created by calling the Data module's 15 constructor with isEval set to False and then calling the getDataloader access program. The same process is followed for the evalLoader, but with isEval set to True. References to these are then stored in the corresponding state variables.
- output: N/A
- exception: N/A

train():

- transition: The model is trained on the trainLoader's data for NUM_EPOCHS epochs using the ADAM optimizer described here. In this process, the evalLoader, trainLoader, and the loss are simply accessed, while the optim and model objects are modified according to the training procedure.
- output: The loss values for each batch are logged to the filesystem using the global Logger 9.
- exception: N/A

eval():

- transition: N/A
- output: The mAP metrics for the current model are collected using the Evaluation Module's 8 eval access program. Only the evalLoader and model are passed in here since no checkpoint needs to be loaded. These metrics are logged to the filesystem using the global Logger 9. Additionally, a checkpoint of the current model is saved to the file system using the Checkpoint Module's 14 saveCheckpoint exported access program (the CkptPath is up to the code writer's discretion as it is based on their local file setup).

7 MIS of Inference Module

7.1 Module

7.2 Uses

- PyTorch Dataloader (PyTorch Documentation)
- PyTorch Tensor (PyTorch Documentation)
- Checkpoint (14)
- Model (10)
- Logger (9)

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
setupData	imagePath (str), lidarPath (str)	-	-
makePred	ckptPath (str)	-	-
inference	boundingBoxesGT (PyTorch Tensor $R^{g \times n_b}$)	-	-

7.4 Semantics

7.4.1 State Variables

- imageTens (PyTorch Tensor $R^{n_{cams} \times 3 \times h \times w}$)
- lidarTens (Pytorch Tensor $R^{3 \times n_l}$)
- boundingBoxesPred (PyTorch Tensor $R^{g \times n_b}$)
- model (Model ADT)

7.4.2 Environment Variables

7.4.3 Assumptions

7.4.4 Access Routine Semantics

setupData(imagePath (str), lidarPath (str)):

- transition: This exported access program uses the Data Processing Module's 20 process access program to load the data and save it to the imageTens and lidarTens state variables.
- output: N/A
- exception: N/A

makePred(ckptPath (str)):

- transition: This exported access program first uses the Checkpoint Module's 14 loadCheckpoint access program with model and ckptPath as inputs to load the saved checkpoint into the model (changes the model's weights). This model is then run on the imageTens and lidarTens state variables to generate an output that is saved into the boundingBoxesPred state variable.
- output: N/A
- exception: N/A

inference(boundingBoxesGT (PyTorch Tensor $R^{g \times n_b}$)):

- transition: N/A
- output: This module uses the Plotting Module's 13 plot access program to display the lidar pointcloud alongside the predicted and ground truth bounding boxes. This is to allow for visual inspection of the predictions.
- exception: N/A

8 MIS of Evaluation Module

8.1 Module

8.2 Uses

- PyTorch Dataloader (PyTorch Documentation)
- Model (10)
- Checkpoint (14)

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
eval	evalLoader (PyTorch Dataloader), model (Model Module 10), ckpt (Checkpoint Module 14)	metrics (Named Tuple of (str, R))	-

8.4 Semantics

8.4.1 State Variables

8.4.2 Environment Variables

8.4.3 Assumptions

8.4.4 Access Routine Semantics

eval(evalLoader (PyTorch Dataloader), model (Model Module), ckpt (Checkpoint Module)):

- transition: N/A
- output: This function will run the input model over the evaluation loader and collect the mAP statistics over this set of data. These statistics will then be returned as output. If a checkpoint is passed in, the model will first load the given checkpoint. Details of the mAP calculation can be seen in the SRS's IM4.
- exception: N/A

8.4.5 Local Functions

9 MIS of Logger Module

This module is simply used to represent a wrapper around a global Tensorboard logger (PyTorch tutorial on using Tensorboard) that can be used in any module to log metrics to a logging file. It interacts with the file system environment variable and can be used to visualize logs after saving them.

This module interacts with the File System environmental variable, saving log files to it.

10 MIS of Model Module

10.1 Template

10.2 Uses

- Is a subclass of PyTorch Module (PyTorch Documentation)
- PyTorch Tensor (PyTorch Documentation)
- Equivariant Layers (16)
- OpenPCDet Layers (17)
- Configuration (19)

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
Model	-	Model	-
forward	imgTens (PyTorch Tensor $R^{n_{cams} \times 3 \times h \times w}$), lidarTens (PyTorch Tensor $R^{n_l \times 3}$)	boundingBoxesPred (PyTorch Tensor $R^{g \times n_{\hat{b}}}$)	-

10.4 Semantics

10.4.1 State Variables

- pcdetLayers (OpenPCDet Layers Module 17)
- eqLayers (Equivariant Layers Module 16)

10.4.2 Environment Variables

10.4.3 Assumptions

10.4.4 Access Routine Semantics

Model():

- transition: This function will load the model-related configuration from the Config module 19 and will then create the OpenPCDet and Equivariant Layers accordingly. References to these layers will be stored in the pcdetLayers and eqLayers state variables.

- output: This function will return a reference to self.
- exception: N/A

forward():

- transition: N/A
- output: This function will run the input data (LiDAR and Images) through the OpenPCDet and Equivariant layers to produce a set of predicted bounding boxes. These predicted bounding boxes will then be returned.
- exception: N/A

10.4.5 Local Functions

11 MIS of Loss Module

11.1 Template

11.2 Uses

- Is a subclass of PyTorch Loss (PyTorch Documentation)
- PyTorch Tensor (PyTorch Documentation)
- Configuration (19)

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
Loss	-	Loss	-
forward	boundingBoxesPred (PyTorch Tensor $R^{g \times n_b}$), boundingBoxesGT (PyTorch Tensor $R^{g \times n_b}$)	Loss Value (PyTorch Tensor R)	-

11.4 Semantics

11.4.1 State Variables

- hyperparameters (Sequence of R)

11.4.2 Environment Variables

11.4.3 Assumptions

11.4.4 Access Routine Semantics

Loss():

- transition: This method will set the loss module's hyperparamters according to the loss-related configuration from the Config module 19.
- output: This method will output a reference to self.
- exception: N/A

forward(boundingBoxesPred (PyTorch Tensor $R^{g \times n_b}$), boundingBoxesGT (PyTorch Tensor $R^{g \times n_b}$)):

- transition: N/A
- output: This method will use the predicted and ground truth bounding boxes to regress and output a loss value. Details of the loss function to be implemented can be seen in the SRS's IM2.
- exception: N/A

11.4.5 Local Functions

12 MIS of Optimizer Module

12.1 Template

12.2 Uses

- Is a wrapper around the PyTorch ADAM Optimizer (PyTorch Documentation)
- PyTorch Parameter (PyTorch Documentation)
- Configuration (19)

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
Optimizer	modelParams (Sequence of PyTorch Parameter)	Optimizer	-
step	-	-	-
zero_grad	-	-	-

12.4 Semantics

12.4.1 State Variables

- hyperparameters (Sequence of R)
- modelParameters (Sequence of PyTorch Parameter)

12.4.2 Environment Variables

12.4.3 Assumptions

12.4.4 Access Routine Semantics

Optimizer(modelParams (Sequence of PyTorch Parameter)):

- transition: This method will set the optimizer module's hyperparameters according to the optimizer-related configuration from the Config module 19. It will also save a reference to passed in model parameters in the modelParameters state variable. Details of the ADAM Optimizer can be seen in the SRS's IM3.
- output: This method will return a reference to self.

- exception: N/A

step():

- transition: This method will update the modelParameters based on their gradients, following the ADAM optimization method. Details of the ADAM Optimizer can be seen in the SRS's IM3.
- output: N/A
- exception: N/A

zero_grad():

- transition: This method will clear all gradients for the modelParameters.
- output: N/A
- exception: N/A

12.4.5 Local Functions

13 MIS of Plotting Module

13.1 Module

13.2 Uses

- Open3D Oriented Bounding Boxes (Open3D Documentation)
- Open3D Pointclouds (Open3D Documentation)
- Open3D in general (Open3D Documentation)
- PyTorch Tensor (PyTorch Documentation)

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
plot	lidarTens (PyTorch Tensor $R^{n_l \times 3}$), boundingBoxesGT (PyTorch Tensor $R^{g \times n_b}$), boundingBoxesPred (PyTorch Tensor $R^{g \times n_{\hat{b}}}$)	-	-

13.4 Semantics

13.4.1 State Variables

13.4.2 Environment Variables

- Computer Screen (2D sequence of RGB tuples)

13.4.3 Assumptions

13.4.4 Access Routine Semantics

plot():

- transition: The LiDAR pointcloud is first converted into an Open3D Pointcloud. Then, the bounding boxes are converted into the Open3d OrientedBoundingBox type. Finally, these are all visualized together in a single 3D plot using Open3D's draw_geometries function. This outputs a plot and thus updates the computer screen environment variable.
- output: N/A

- exception: N/A

13.4.5 Local Functions

14 MIS of Checkpoint Module

14.1 Module

14.2 Uses

- PyTorch Module (PyTorch Documentation)
- PyTorch State Dictionary (PyTorch Documentation)

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
saveCheckpoint	model (PyTorch Module 18), CkptPath (str)	-	-
loadCheckpoint	model (PyTorch Module 18), CkptPath (str)	-	-

14.4 Semantics

14.4.1 State Variables

14.4.2 Environment Variables

- File System

14.4.3 Assumptions

14.4.4 Access Routine Semantics

saveCheckpoint():

- transition: The input model's state dictionary is saved to a pytorch tensor file at the input path.
- output: N/A
- exception: N/A

loadCheckpoint():

- transition: The input model's parameters are set using the state dictionary saved at the input path.

- output: N/A
- exception: N/A

14.4.5 Local Functions

15 MIS of Data Module

15.1 Module

15.2 Uses

1. PyTorch Dataset and Dataloader (PyTorch Documentation)
2. Configuration (19)

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
Data	isEval (boolean)	Data	-
getDataloader	-	loader (PyTorch Dataloader)	-

15.4 Semantics

15.4.1 State Variables

- dataset (PyTorch Dataset)

15.4.2 Environment Variables

1. File System

15.4.3 Assumptions

15.4.4 Access Routine Semantics

Data():

- transition: This function will load the data-related configuration from the Config module 19 and will then create the dataset accordingly. This dataset will either be the NuScenes (Caesar et al. (2020)) or the Waymo (Sun et al. (2020)) dataset for now and will be stored in the dataset state variable. If isEval is false, the training portion of the dataset will be loaded. If isEval is true, the evaluation portion of the dataset will be loaded.
- output: This function will return a reference to self.
- exception: N/A

getDataloader():

- transition: N/A
- output: This function will use the dataset state variable to create a PyTorch Dataloader which will then be returned.
- exception: N/A

15.4.5 Local Functions

16 MIS of Equivariant Layers Module

This module is meant to represent the equivariant layers I will be adding into the OpenPCDet repository for the novel part of my project. To provide context for the reader, equivariant layers refer to neural network layers that preserve input transformations when generating output features. For example, a rotated image of a cat will result in similarly rotated output features (when compared to features generated from an upright image of a cat). This work by Cohen et Al. (Cohen and Welling (2016a)) more clearly explains this concept.

This module will encapsulate two different types of equivariant layers. The first of these is steerable kernel equivariant CNNs (Cohen and Welling (2016b)) extended to include both 2D and 3D symmetries using the escnn repository (Cesa et al. (2022)). The second of these is equivariant transformer networks (Tai et al. (2019)). Existing OpenPCDet layers can essentially then be directly swapped out for these equivariant alternatives.

16.1 Module

16.2 Uses

- PyTorch Parameter (PyTorch Documentation)
- Steerable CNNs (Cesa et al. (2022))
- Equivariant Transformers (Tai et al. (2019))

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
forward	-	-	-

16.4 Semantics

16.4.1 State Variables

- layerParameters (Sequence of PyTorch Parameter)

16.4.2 Environment Variables

16.4.3 Assumptions

16.4.4 Access Routine Semantics

init():

- transition: The parameters for the layer are randomly initialized according to a uniform distribution around 0 (saved in the layerParameters state variable).
- output: N/A
- exception: N/A

forward():

- output:
  - **For the CNN case:** Since the CNN case is simpler, the math relating to it can be shown here itself. The result y of this formula is returned:

    $y[m,n](k,x) = \sum_{i=-w}^{w} \sum_{j=-h}^{h} k[w+i,h+j]x[m+i,n+j] \rightarrow y(k,x) = k * x$
    such that if $x_2 = gx_1$ for some g in the group of considered transformations, $y(x_2) = \rho(g)y(x_1)$ for a single function ρ.

  - **For the Transformer case:** The math here is more complicated and so I will simply refer the reader to the paper by Tai et. al (Tai et al. (2019)). The output of this layer is returned. Generally though for attention function f, the following can be written similar to the CNN case: $x_2 = gx_1 \rightarrow y(k,x_2) = \rho(g)y(k,x_1)$

16.4.5 Local Functions

17 MIS of OpenPCDet Layers Module

This module is simply used to represent the existing layers in the OpenPCDet library (Team (2020)). Rather than highlight specific relevant layer types and modules, I would encourage the reader to read through the BEVFusion (Liang et al. (2022)) configuration file (BEVFusion Config File) and associated code sections (Model-Related OpenPCDet Code).

18 MIS of PyTorch Module

This module is simply used to represent the PyTorch library referenced in the other modules. The main list of components that are used from this library is as follows:

- PyTorch Tensor (PyTorch Documentation)
- PyTorch Module (PyTorch Documentation)
- PyTorch Dataloader (PyTorch Documentation)
- PyTorch Optimizer (PyTorch Documentation)
- PyTorch Loss (PyTorch Documentation)
- PyTorch State Dictionary (PyTorch Documentation)
- PyTorch Parameter (PyTorch Documentation)

19 MIS of Config Module

19.1 Template

19.2 Uses

19.3 Syntax

19.3.1 Exported Constants

- CONFIG_PATH (str): The standard filesystem path pointing to where the configuration file is. For now this is set to "./checkpoint" but the idea is to have it overridden through a command-line argument where necessary.

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
Config	-	Config	-
getModelConfig	-	-	-
getDataConfig	-	-	-
getOptimConfig	-	-	-
getLossConfig	-	-	-

19.4 Semantics

19.4.1 State Variables

- modelConfig (Sequence of R)
- dataConfig (Sequence of R)
- optimConfig (Sequence of R)
- lossConfig (Sequence of R)

19.4.2 Environment Variables

- File System

19.4.3 Assumptions

19.4.4 Access Routine Semantics

Config():

- transition: This method loads the yaml configuration file stored at the predefined constant path CONFIG_PATH. Then, the dictionary loaded from this JSON file is split into model, data, optimizer, and loss sections. Each of these is saved to the corresponding state variable for this module. A sample configuration file can be seen here.
- output: This method returns a reference to self.
- exception: N/A

getModelConfig():

- transition: N/A
- output: This method returns the configuration parameters for the model module 10.
- exception: N/A

getDataConfig():

- transition: N/A
- output: This method returns the configuration parameters for the data module 15.
- exception: N/A

getOptimConfig():

- transition: N/A
- output: This method returns the configuration parameters for the optimizer module 12.
- exception: N/A

getLossConfig():

- transition: N/A
- output: This method returns the configuration parameters for the loss module 11.
- exception: N/A

19.4.5 Local Functions

20 MIS of Data Processing Module

20.1 Module

20.2 Uses

- PyTorch Tensor (PyTorch Documentation)

20.3 Syntax

20.3.1 Exported Constants

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
process	imgPath (str), lidarPath (str)	imgTens (PyTorch Tensor $R^{n_{cams} \times 3 \times h \times w}$), lidarTens (PyTorch Tensor $R^{n_l \times 3}$)	-

20.4 Semantics

20.4.1 State Variables

20.4.2 Environment Variables

20.4.3 Assumptions

20.4.4 Access Routine Semantics

process():

- transition: N/A
- output: Loads the multiview images and LiDAR pointcloud from their respective files and converts them into a standard PyTorch Tensor format that can be used by the model. The Tensor form of the multiview images and the LiDAR pointcloud are returned.
- exception: N/A

20.4.5 Local Functions

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.

Gabriele Cesa, Leon Lang, and Maurice Weiler. A program to build E(N)-equivariant steerable CNNs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=WE4qe9xlnQw.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016a.

Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016b.

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.

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.

Kai Sheng Tai, Peter Bailis, and Gregory Valiant. Equivariant Transformer Networks. In *International Conference on Machine Learning*, 2019.

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

21 Appendix

May be added in the future.