

**Freie Universität Berlin**

---

Department of Mathematics and Computer Science

Dahlem Center for Machine Learning and Robotics

**Deep Neural Network Architectures For Semantic  
Segmentation Of Images And Cross-Modal Label  
Transfer To Point Clouds Applied In An  
Autonomous Driving System**

**Adrian Widera**

A thesis submitted in partial fulfillment of the requirements for the degree of

**Master in Computer Science**

Examiner: Prof. Dr. Daniel Göhring

Examiner: Prof. Dr. Dr. hc. habil. Raúl Rojas

Berlin, 03.12.2018

© Copyright by Adrian Widera 2018.  
All Rights Reserved.

# Abstract

Interest for autonomous driving is growing around the globe. Current production car's assist systems already show some conditional driving automation. Technological advancements allow to achieve higher levels of driving automation, up to autonomous performing of entire dynamic driving tasks. Nonetheless, the save deployment of autonomous vehicles beyond research laboratory environments in real traffic on public roads necessitates further developments, which makes technologies around autonomous vehicles an area of active scientific research. A robust comprehensive environmental perception and understanding are basic requirements for derived actions and save driving behavior. It can only be achieved through the combination of different multi-modal sensing technologies and according data fusion.

An analysis of current trends shows that camera and LiDAR sensing technologies in combination with deep artificial neural network architectures and semantic segmentation in the context of autonomous driving form a set of current and challenging topics, equally interesting from an industrial as well as research perspective, to be addressed in this master thesis in computer science.

With the MIG (Made In Germany), the Free University Berlin maintains an adequately equipped autonomous vehicle platform serving as research platform with permission to operate in real world traffic.

As part of this thesis a deep neural network architecture framework suitable for semantic segmentation is integrated in the MIG platform. Furthermore an appropriate deep artificial neural network for pixelwise semantic segmentation of the vehicle's camera images is selected and implemented. Eventually a system is developed and implemented into the MIG that performs cross-modal transfer of pixelwise semantic labels from 2D images to corresponding 3D point clouds generated by a LiDAR scanner. All implementations and their underlying technologies are then assessed concerning their suitability in the context of autonomous driving.

# Declaration

## Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 03.12.2018

A. Widera



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Scope . . . . .	1
1.2	Motivation . . . . .	1
1.3	Goals . . . . .	3
1.4	Structure . . . . .	4
<b>2</b>	<b>Fundamentals</b>	<b>5</b>
2.1	Conventions And Definitions . . . . .	5
2.2	Autonomous Vehicle . . . . .	7
2.3	ROS - Robot Operating System . . . . .	11
2.4	Digital Image . . . . .	14
2.5	Artificial Neural Network (ANN) . . . . .	15
2.6	Convolutional Neural Network (CNN) . . . . .	22
2.7	Point Cloud . . . . .	27
2.8	Spatial Descriptions And Transformations . . . . .	28
<b>3</b>	<b>Approach</b>	<b>31</b>
3.1	System Overview And Main Components . . . . .	31
3.2	SegNet - CNN For Semantic Segmentation Of Images . . . . .	33
3.3	ROS Caffe Package - ROS CNN Support . . . . .	35
3.4	ROS Node awid_SemSegImg - Semantic Segmentation of 2D Images . . . . .	37
3.5	ROS Node awid_SemSegPnt - Label Transfer To 3D Point Clouds . . . . .	42
<b>4</b>	<b>Evaluation</b>	<b>48</b>
4.1	Dataset For Semantic Segmentation Evaluation . . . . .	48
4.2	SemSegImg Node Evaluation . . . . .	52
4.2.1	Image 2D Quantitative Evaluation Methodology . . . . .	53
4.2.2	Image 2D Qualitative Evaluation Methodology . . . . .	55
4.2.3	SemSegImg Resource Consumption Assessment . . . . .	57
4.2.4	SemSegImg Segmentation Assessment . . . . .	59
4.3	SemSegPnt Node Evaluation . . . . .	64
4.3.1	Point Cloud 3D Evaluation Methodology . . . . .	64
4.3.2	SemSegPnt Resource Consumption Assessment . . . . .	67
4.3.3	SemSegPnt Alignment And Segmentation Assessment . . . . .	68
<b>5</b>	<b>Conclusion</b>	<b>75</b>
5.1	Conclusive Summary . . . . .	75
5.2	Future Work . . . . .	78
<b>A</b>	<b>SegNet - Deep CNN for Semantic Segmentation</b>	<b>80</b>
<b>B</b>	<b>KITTI Autonomous Driving Platform</b>	<b>84</b>

# List of Figures

2.1	MIG Autonomous Vehicle - Overview Sensor Positioning [33, Image]	7
2.2	MIG Autonomous Vehicle - Positions of Integrated SatCams	8
2.3	ROS Computation Graph Level - Publish/Subscribe Mechanism	11
2.4	ROS Filesystem Level - Recommended Catkin Workspace Layout	12
2.5	Digital Image Representation - RGB Image, Channels And Numerical Array	14
2.6	Artificial Neuron - Computational Unit - Model And Structure	15
2.7	Rectified Linear Unit (ReLU) - Activation Function	16
2.8	3-Layer Fully Connected Feedforward ANN	17
2.9	3-Layer Fully Connected Feedforward ANN With Cost Function	18
2.10	Network Learning Progress And Prediction Accuracy [61]	21
2.11	CNN - 2D Convolution	22
2.12	CNN - Sparse Connectivity And Parameter Sharing	23
2.13	CNN - Increased Receptive Field And Spatial Context In Deeper Layers	24
2.14	CNN - 3-Dimensional Computing Units	24
2.15	CNN - 3D Convolution	25
2.16	MIG Autonomous Vehicle - RViz Visualized Point Cloud And Camera Images	27
2.17	MIG Autonomous Vehicle - Frames Attached To Base Link, Velodyne and Front SatCam	29
2.18	Pinhole Camera Model Geometry [64]	29
3.1	Implementation Approach - Overview System And Components	32
3.2	SegNet Architecture Illustration, adapted from [28]	33
3.3	SegNet Encoder - Consecutive Convolution Layers	34
3.4	ROS CAFFE Package Installer - UML Activity Diagram	36
3.5	SemSegImg Node Implementation - UML Activity Diagram	37
3.6	SemSegImg Node Implementation - UML Class Diagram	38
3.7	SemSegPnt Node Implementation - UML Activity Diagram	43
3.8	SemSegPnt Node - Callback Function Extensive Option Implementation - UML Activity Diagram	46
4.1	awid_KITTLSemSeg Dataset - Folder Structure And Content	50
4.2	awid_KITTLSemSeg Dataset - SegNet Class Occurrence Summary	51
4.3	awid_KITTLSemSeg Dataset - SegNet Class Pixel Distribution	51
4.4	IoU Python Script Implementation - UML Activity Diagram	54
4.5	SemSegImg Node - Qualitative Evaluation - Images For Visual Comparison	56
4.6	awid_KITTLSemSeg Dataset - SegNet Class Precision Recall IoU Summary	59
4.7	awid_KITTLSemSeg Dataset - SegNet Class $IoU_{Image}$ Score Summary	60
4.8	SegNetImg Node - Evaluation Image Set - awid_KITTLSemSeg Frames 105 (left) And 7 (right) - Highest $IoU_{Image}$ And Highest FN In SegNet Class <i>Sign</i>	61
4.9	SemSegImg Node - Evaluation Image Set - awid_KITTLSemSeg Frames 81 (left) And 32 (right)	62

4.10	SemSegImg Node - Evaluation Image Set - awid_KITTLSemSeg Frames 115 (left) And 92 (right) - Highest And 2nd Highest FP In SegNet Class <i>Fence</i> . . . . .	63
4.11	SemSegPnt Node - Qualitative Evaluation - Images For Visual Comparison	66
4.12	KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 27 . .	68
4.13	KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 84 . .	68
4.14	KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 78 . .	69
4.15	MIG SatCam RGB Color Image With Point Cloud Overlay . . . . .	69
4.16	MIG SatCam Overlay Images - Point Cloud Alignment . . . . .	70
4.17	SemSegPnt Node - SemSeg-Colored Point Clouds - awid_KITTLSemSeg Dataset Frames 5 (left) and 82 (right) . . . . .	71
4.18	SemSegPnt Node - SemSeg-Colored Clouds - awid_KITTLSemSeg Frames 66 (left) and 107 (right) - Average And Highest $\overline{IoU}_{Image}$ For Class <i>Vehicle</i>	72
4.19	SemSegPnt Node - MIG Overlay Image And SegNet-Colored Point Cloud .	73
4.20	SemSegPnt Node - MIG SatCam Images And SegNet-Colored Point Clouds - Label-Casting And Occlusion . . . . .	74
A.1	SegNet Graph Drawing - Key Elements Of The Network Architecture . . .	83
B.1	KITTI Autonomous Vehicle - Sensor Setup [90] . . . . .	84
B.2	KITTI Autonomous Vehicle - ROS TF Frame Tree . . . . .	85

# List of Tables

2.1	Notation Overview . . . . .	5
2.2	MIG Autonomous Vehicle - SatCam Camera Characteristics . . . . .	9
2.3	MIG Autonomous Vehicle - Velodyne LiDAR Scanner Characteristics . . . . .	9
2.4	MIG Autonomous Vehicle - Computer Specification Outline . . . . .	10
2.5	Overview Coordinate Spaces . . . . .	28
3.1	Target System Environment Characteristics . . . . .	35
3.2	SemSegImg Node - Command Line Argument Options Overview . . . . .	39
3.3	SemSegImg Node Command - Line Argument Options Defaults . . . . .	40
3.4	SemSegPnt Node - Command Line Argument Options Overview . . . . .	42
3.5	SemSegPnt Node - Command Line Argument Options Defaults . . . . .	44
4.1	Class Mapping - KITTI i.e. Cityscapes $\leftrightarrow$ SegNet . . . . .	52
4.2	SemSegImg Node - Execution Duration And Memory Consumption . . . . .	58
4.3	Class Mapping - KITTI Tracklet Label $\rightarrow$ SegNet . . . . .	64
4.4	SemSegPnt Node - Execution Time And Memory Consumption . . . . .	67
4.5	SemSegPnt Node - Quantitative Evaluation - Sample Frames Class <i>Vehicle</i> . . . . .	70
A.1	SegNet Network Visualization Color Scheme . . . . .	80
A.2	SegNet Class Color Scheme . . . . .	80
A.3	SegNet Layers Part 1 - Input And Encoders . . . . .	81
A.4	SegNet Layers Part 2 - Decoders And Classifier . . . . .	82
B.1	KITTI Autonomous Vehicle - Left Color Camera Characteristics . . . . .	85
B.2	KITTI Autonomous Vehicle - Velodyne LiDAR Scanner Characteristics . . . . .	85

# Chapter 1

## Introduction

### 1.1 Scope

The presented work is a master thesis in computer science. It is prepared in the Intelligent Systems and Robotics Group of the Free University of Berlin as partial fulfillment of the requirements for the degree of Master in Computer Science. As such it represents an academic work that focuses on technology and implementation. In particular the following topics are addressed:

- autonomous driving systems
- computer vision
- cameras and 2D images
- LiDAR and 3D point clouds
- deep artificial neural network architectures
- semantic segmentation
- multi-modal sensor fusion
- cross-modal class label transfer

Even in awareness of their importance, this work is not a scientific treatment of business, economics, legal, social, health or other related research areas.

The thesis does not allow and thus this text is not intended to be exhaustive on all topics discussed. Nonetheless, all topics necessary to follow the discussion within the scope of this master thesis are addressed directly to a reasonable level of detail. In addition, references to further information are provided.

### 1.2 Motivation

Interest for *autonomous driving* is growing around the globe [1] and has never before attracted so much publicity as in recent months [2, p. 2]. Advanced technologies to enable features like *ADAS* (*Advanced Driver Assistance System*) have rapidly arrived on the market and their future deployment is expected to accelerate [3, p. 5]. Cars providing partial driving automation according to the *Society of Automotive Engineers' SAE Level 2* [4, Table 1] like acceleration, breaking and steering assistance are common on today's streetscape. Even automated driving systems (ADS) that offer SAE Level 3 conditional or SAE Level 4 high driving automation are beyond research laboratory environments and are increasingly

deployed on public roads [5]. These vehicles are capable of autonomously performing entire dynamic driving tasks under certain conditions with driver intervention only as safety fallback [4, Table 1].

To be aware of its surroundings, *autonomous vehicles (AVs)* deploy the combination of different sensor systems, mainly based on radar, cameras and LiDAR. Only the combination of all three technologies enables sufficient comprehensive environmental sensing for higher level automatic driving systems [6]. Weaknesses of one technology are compensated by the strengths of another. Radar systems are proven and robust and as such already widely deployed in current cars.

Advancements in camera related technologies, especially in computer vision and vision-based road scene understanding, facilitate a growing application of cameras in the automotive sector. A recent study on search engine volume queries [7] evidences that cameras are the biggest consumer-led trend in the automotive category. The European New Car Assessment Program (Euro NCAP), known for its star ratings, includes test procedures for generally camera-based features like Lane Support Systems (LSS) [8]. Such safety assist systems become increasingly important [9, p. 4] and a necessity to achieve high Euro NCAP star ratings. Governmental regulations in the USA [10] even mandate the deployment of automotive cameras for new cars by May 1, 2018.

Especially when it comes to exact distance measurements and environmental mapping, which are essential for collision-free navigation, other technologies are superior. A key sensing technology in autonomous driving cars is *LiDAR (Light Detection and Ranging)*, a method that measures distance to objects by illuminating those with laser light and measuring the return time of the reflected light. It was used by the winner of the 2005 DARPA (Defence Advanced Research Project Agency) Urban Challenge and by all finishing teams in 2007. LiDAR is a comparatively new technology in the automotive market. However, experts believe that LiDAR in combination with radar and camera systems will ultimately become the approach favored by many future AV players [11]. Multi-modal data fusion gives a far better basis for scene understanding and provides the AV with information to make educated decisions.

Although some autonomous driving technologies are mature enough to be conditionally used in the public, the general market availability of autonomous driving cars is expected not before the mid to late 2020s [2, p.20], [5]. Towards higher levels of autonomy further developments regarding comprehensive environmental perception and derived resulting behavior are essential. Technologies around AVs are an area of active scientific research with a continuously increasing number of publications and an uninterrupted evolution of citations, both summing in the last few years [12]. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) is ranked 5th in the list of Google scholar top publications in the category *Engineering & Computer Science* [13]. Nearly all of the most cited CVPR publications [14] are concerned with *deep artificial neural network architectures*. The 4 top-cited publications [15], [16], [17], [18] have over 5000 and as such more than twice as many citations than the following papers. Half of those publications [17], [18] are addressing *semantic segmentation* which is the partitioning of an image into coherent semantically meaningful parts. Semantic segmentation is considered an important step towards comprehensive road scene understanding [19] and thus for autonomous driving.

The above leads to the conclusion that camera and LiDAR sensing technologies in combination with deep artificial neural network architectures and semantic segmentation in the context of autonomous driving form a set of current and challenging topics, equally interesting from an industrial as well as research perspective, to be addressed in a master thesis in computer science.

## 1.3 Goals

The goals of this thesis are derived from the set of topics concluded in the motivation above. To adequately address this set of topics within the scope of this thesis the following goals are aimed at:

1. Integration of a deep artificial neural network framework suitable for semantic segmentation into a real world autonomous driving system.
2. Development and implementation of pixelwise semantic segmentation of 2D color camera images, based on a deep artificial neural network architecture, in a real world autonomous driving system.
3. Development and implementation of a system, for cross-modal transfer of pixelwise semantic labels from 2D images to 3D point clouds generated by a LiDAR-based system, in a real world autonomous driving system.

To elaborate on the summarized goals: Firstly an autonomous driving system should be empowered to utilize deep neural network technology. Secondly 2D images generated by cameras of an autonomous vehicle are supposed to be pixelwise semantically segmented by utilizing a deep neural network. Thirdly the pixelwise labels of 2D images are planned to be transferred to label another modality, the 3D points generated by an autonomous vehicle's LiDAR system. The first goal provides the key enabling technology for the second one. The third goal represents a concrete application of utilizing the output generated by a semantic segmentation architecture such as the one described in the second goal. The design and implementation of the semantic segmentation of the 2D images as well as the cross-modal label transfer to 3D points, should be deployable in a real world autonomous driving system.

## 1.4 Structure

The general structural approach throughout this thesis is to introduce concepts and terms, provide an illustrative example and finally reference sources of further information. Regarding the chapters and their contents this thesis is structured as follows:

*Chapter 1 - Introduction* – defines the scope of the thesis, introduces its motivation and the therefrom derived goals to be satisfied through the course of this work. In addition, this chapter outlines the structure of the thesis including a short description of each chapter’s contents.

*Chapter 2 - Fundamentals* – introduces and explains terms, technologies and concepts applied in subsequent chapters. Its sections present conventions, color schemes and fundamental definitions used throughout this work. It informs about the autonomous vehicle used as target platform for implementation to satisfy the previously formulated goals. The chapter further introduces Artificial Neural Networks and Convolutional Neural Networks focusing on specific architectural concepts to allow comprehension of selection and evaluation of a suitable neural network architecture for implementation. Finally an overview of spatial descriptions, transformations and the deployed camera model are provided.

*Chapter 3 - Approach* – explains the development and implementation of a system and its components to satisfy the previously formulated goals. The chapter informs about SegNet, the selected neural network architecture for semantic segmentation of images and explains the implementation to integrate it into the autonomous vehicle target platform. The remaining section covers the implementation of the cross-modal label transfer from 2D images to 3D points.

*Chapter 4 - Evaluation* – describes the compilation of a dataset to assess the implementation and corresponding underlying technologies. After introducing the appropriate methodologies the results of a quantitative and qualitative evaluation of the implemented system are presented and discussed.

*Chapter 5 - Conclusion* – summarizes the accomplished work and the key findings gathered in this thesis. Furthermore it presents directions for future work.

*Appendix A* – provides additional details on the SegNet architecture and defines color schemes for network visualization as well as SegNet classes. It lists dimensions, calculated memory requirements and the number of network parameters for each of the network’s layers and depicts graph drawings of the SegNet architecture’s key building blocks.

*Appendix B* – informs about the KITTI autonomous vehicle configuration and the characteristics of its color camera and LiDAR scanner. Data captured with this vehicle platform is used in the evaluation sections in this work.



## Chapter 2

# Fundamentals

This chapter introduces and explains terms, technologies and concepts applied in subsequent chapters. Its sections present conventions, color schemes and fundamental definitions used throughout this work. It informs about the configuration and characteristics of the autonomous vehicle used as target platform for implementation to satisfy the previously formulated goals and the platform's mainly addressed components, such as the cameras, the LiDAR scanner and deployed software frameworks. The chapter further introduces Artificial Neural Networks, their principle components, architectures as well as training and inference. Convolutional Neural Networks are presented with particular coverage of specific architectural concepts to allow comprehension of selection and evaluation of a suitable neural network architecture for implementation. Finally an overview of spatial descriptions, transformations and the deployed camera model are provided.

### 2.1 Conventions And Definitions

This section gives an overview of notations, general definitions and naming conventions used throughout this thesis. Table 2.1 shows commonly deployed types, their notation and a short description.

Type	Notation	Description
scalar	<i>x</i>	italic, lowercase letter
vector	<b>x</b>	italic, bold typeface, lowercase letter
matrix	<b>X</b>	bold typeface, uppercase letter
tensor	<b>X</b>	math sans serif font, uppercase
averaged value	$\bar{x}$	overline
described term	<i>term</i>	italic in normal text
file name	<code>name_of_file.end</code>	typewriter font
command line text	<code>cm_text</code>	typewriter font
class names	<i>class name</i>	math mode

Table 2.1: Notation Overview

To better identify items that are generated or specifically modified as part of this thesis, like scripts, source code files, paths or datasets, these are prefixed with *awid\_*, as long as compliance with other naming conventions is maintained and contextual understanding facilitated. The source code files, reports and scripts created and described in this work are placed on a git repository [20].

Source code written in C++ follows the ROS C++ Style Guide [21], Python code the Python Enhancement Proposal PEP8 - Style Guide for Python Code [22]. The source code

contains Doxygen-style documentation [23]. Overview diagrams are provided according to the Object Modeling Group’s Unified Modeling Language UML 2.5 [24]. File and path names within a ROS (Robot Operating System) [25] workspace, introduced in Section 2.3, comply with the ROS Enhancement Proposal REP 144 - ROS Package Naming [26].

Data structures commonly applied within this work are *images* and *point clouds*, typically represented as arrays, vectors or according files, as described in Sections 2.4 and 2.7. Essence data is usually organized into channels such as RGB, L or XYZ, each representing an according property. *RGB* channels indicate a color image and can be interpreted as real world color information or as class label, whereby each color represents a class according to a predefined color scheme. Color schemes applied in this thesis relate to *SegNet* [27][28], a deep network architecture detailed in Section 3.2, and to the *KITTI* [29] or the *Cityscapes* [30] road scene datasets. The *L* channel of an image carries classification information whereby the pixel value directly represents the class ID. *Classification* means the assignment of a label from a set of classes to a set of inputs, like for example assigning a certain class to each pixel of an image. *Semantic segmentation* is realized by classifying each pixel of an image and thus partitioning the image into coherent semantically meaningful parts. The term *semantic* is used for instances with according element-wise class labels. The labeling can originate from a manual annotation process, generating *ground truth*, or from an automatic processing like a deep network, generating *predicted* aka *inferred* instances as output. Semantic segmentation is commonly abbreviated with *SemSeg*, ground truth with *gt* and predicted with *pred*.

Therefore, in a `SegNetColored.png` image file the color represents a class according to the SegNet color scheme, whereas a `SegNetRGBL.png` image file’s RGB channels contain either real world or semantic color information and the L channel values represent the class ID of each corresponding pixel.

Several color schemes are deployed. In visualizations of neural network architectures the input layer is colored red, the output blue, layers inbetween green. The color scheme applied for visualizations of network graphs is defined in Table A.1 in Appendix A. Class-assignment represented as colors in images follows the color scheme defined in Table A.2 for SegNet and the one shown in Table 4.1 for KITTI or Cityscapes.

The used citation style is IEEE (Institute of Electrical and Electronic Engineers) Style [31] based on the Chicago Manual of Style [32].

## 2.2 Autonomous Vehicle

The autonomous vehicle project at the Intelligent Systems and Robotics Group of the Freie Universität Berlin, Germany started in the year 2006. The mission since then has been to research and develop autonomous and assistance systems.

For that purpose several platforms, especially autonomous cars, have been built. In 2007 a team participated in the DARPA (Defence Advanced Research Project Agency) Grand Urban Challenge with a modified Dodge Grand Caravan named "Spirit of Berlin" and reached the semi-finals. With funding from the BMBF, the German Federal Ministry of Education and Research, the AutoNOMOS Labs were formally founded in 2009 and the autonomous car platform "*MadeInGermany*" (MIG) has been built as successor of the "Spirit of Berlin". The MIG is based on a Volkswagen Passat Variant 3C. In 2011 an electrically powered Mitsubishi i-MiEV has been added.

For this thesis the MIG is used as target system because its technical setup suits best and broad experience has been gathered over the years. It is equipped with drive-by-wire technology and several different sensors allowing the car to be controlled by the AutoNOMOS Software framework which runs on computers connected to the car. Due to extensive simulation and testing of driving scenarios at the former airport Tempelhof, it was possible to obtain in 2011 a special permit to drive the MIG autonomously on public roads in real traffic, although the safety concept requires monitoring by a human safety driver and a safety observer. The MIG drove many thousand kilometers autonomously since then, including a 2400 km long-distance drive from the USA to Mexico City.

Figure 2.1 illustrates the MIG and the arrangement of some of it's sensors. The sensors addressed mainly in this thesis are the *Technica Engineering (TE) SatCam* RGB color cameras and the *Velodyne HDL-64E S2* LiDAR scanner.

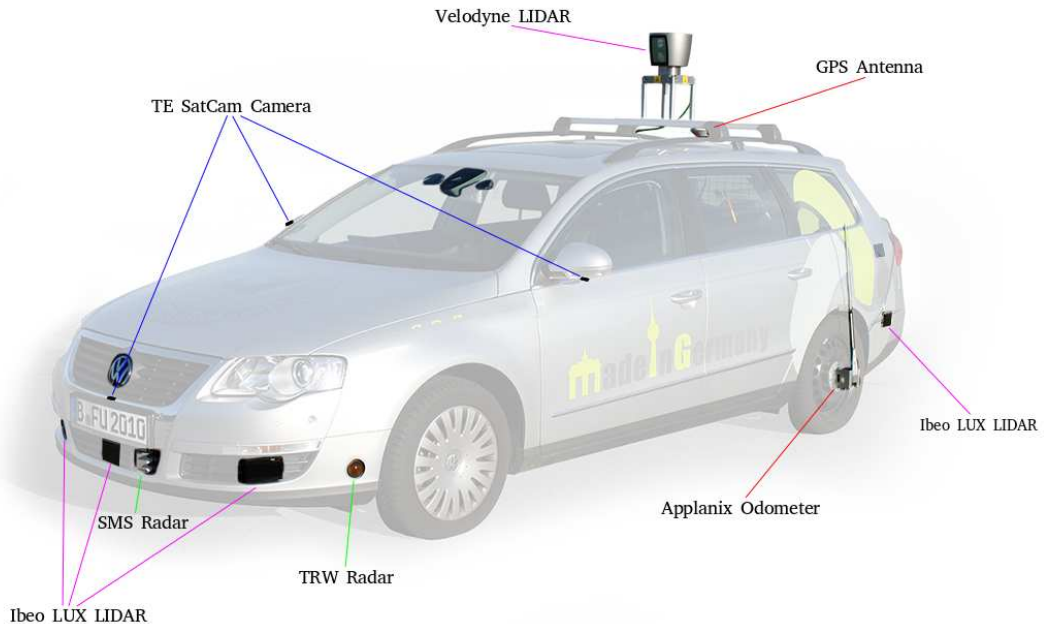


Figure 2.1: MIG Autonomous Vehicle - Overview Sensor Positioning [33, Image]

There are 4 TE SatCam RGB color cameras integrated in the chassis of the car. As shown in Figure 2.2 the cameras are located in the front, in the back and one in each exterior rear mirror on the sides of the car. The cameras are connected to the car via *BroadR-Reach automotive Ethernet* [34] and communicate with the AutoNOMOS Software framework through the *camera driver* developed and described in [35]. The basic characteristics of

the camera system integrated in the MIG are listed in Table 2.2. They are based on specifications provided by the component suppliers [36], [37] and reflect the setup and configuration used in this thesis. Of course it is possible to operate the components of the camera system in other modes as well.

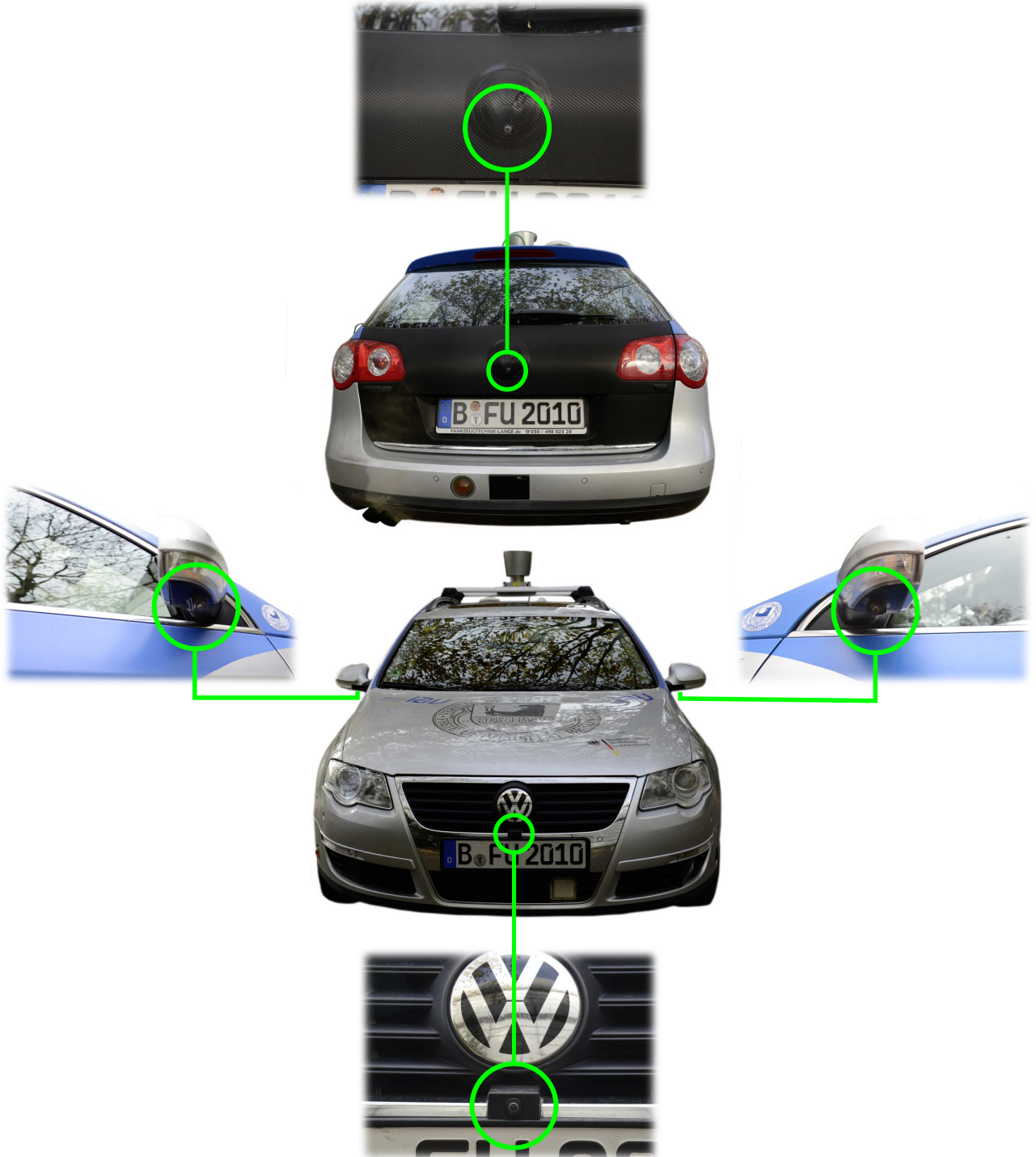


Figure 2.2: MIG Autonomous Vehicle - Positions of Integrated SatCams

Camera Feature	Characteristic
Imaging Sensor	OV10635 Family [36]
Horizontal Field Of View (hFOV) [deg]	190
Vertical Field Of View (vFOV) [deg]	118
Spatial Output Resolution [px]	1280 x 800
Output Frame Rate [fps]	30
Output File Format	8-bit RGB JPG
Physical Connection	IEEE 802.3bw-2015 [34]
Video Stream Transport Protocol	IEEE 1722 [38]
Lens Size ["]	1 / 2.7
Scan Mode	Progressive
Shutter	Rolling Shutter
Pixel Size [ $\mu\text{m}$ ]	4.2 x 4.2
Image Area [ $\mu\text{m}$ ]	5510.4 x 3418.8
Dynamic Range [dB]	115
Max S/N Ratio [dB]	39
Sensitivity [mV / lux-sec]	3650

Table 2.2: MIG Autonomous Vehicle - SatCam Camera Characteristics

Whereas the cameras output 2D images the Velodyne LiDAR scanner generates 3D point clouds. It is mounted on the roof of the car as depicted in Figure 2.1. Specifications according to the applied mode of operation are listed in Table 2.3. Analogous to the cameras, the specifications are based on data provided by the manufacturer in [39] and [40] and represent the configuration used in this work.

LiDAR Scanner Feature	Characteristic
Scanner Model	Velodyne HDL-64E S2
Number of Lasers	64
Number of Detectors	64
Rotations per Minute [rpm]	600
Rotations per Second [hz]	10
Vertical Field of View (vFOV) [deg]	+2 to -24.8
Horizontal Field of View (hFOV) [deg]	360
Total Points per Revolution	133,333
Horizontal Angular Resolution (Azimuth) [deg]	0.17283
Vertical Angular Resolution (Altitude) [deg]	0.41875
Range [m]	up to 120

Table 2.3: MIG Autonomous Vehicle - Velodyne LiDAR Scanner Characteristics

As already mentioned, all of the MIG's autonomous driving system components are eventually connected to a computer running the AutoNOMOS Software framework. The specifications of the computer used in this thesis are outlined in Table 2.4. The computer is a notebook connected to the MIG via Ethernet running Ubuntu Linux and the Robot Operating System (ROS). ROS serves as basis for the AutoNOMOS Software framework which is adapted and extended with own ROS packages as needed. Since ROS plays a key role in developing and implementing a working solution to the problem stated at the beginning, the dedicated section 2.3 provides an initial overview to ROS and some of its key concepts.

Computer Feature	Characteristic
Computer Model	Dell XPS 15 9560
Processor (CPU)	i7-7700HQ @ 2.8GHz
System Memory (RAM)	32GB DDR4, 2400 MHz
Storage	SK Hynix 1TB PCIe SSD
Graphics Card (GPU)	Nvidia GeForce GTX 1050
GPU Memory (VRAM)	4GB GDDR5
Ethernet	Dell DA200 Adapter
Operating System	Ubuntu 16.04.3 LTS (Xenial)
Linux Kernel	4.4.0-130-generic(x86_64)
Robot Operating System (ROS) Version	Kinetic Kame

Table 2.4: MIG Autonomous Vehicle - Computer Specification Outline

More information about the autonomous vehicles described above are available in the book [41], on the Autonomos Systems GmbH website [autonomos-systems.de/en/](https://autonomos-systems.de/en/) or in the many publications and theses accessible at the [website](#) of the Intelligent Systems and Robotics Research Group in the Department of Mathematics and Computer Science at the Free University Berlin.



## 2.3 ROS - Robot Operating System

The *Robot Operating System (ROS)* [25] is a collection of flexible software frameworks and a unified approach for writing robot applications. It consists of tools, libraries, and conventions and provides robotics middleware, hardware abstraction, device drivers, visualizers, message-passing, package management, and more. ROS was initiated at the Stanford Artificial Intelligence Lab and is nowadays maintained by the Open Source Robotics Foundation (OSRF). The Robot Operating System (ROS) is becoming the de facto standard programming approach for modern robotics [42, p. XIX].

The ROS architecture is divided into several levels of concepts. One of these levels of concepts is the *ROS Computation Graph*, representing a ROS system as peer-to-peer network of independent ROS processes each receiving data, performing some computation and passing data and results to other programs. A ROS process is represented as *node*, a running instance of a ROS program that uses ROS's middleware to communicate with other processes along the graph's edges. Communication among nodes is done by sending and receiving *messages*. The fundamental communication method to exchange messages are topics combined with a publish/subscribe mechanism as shown in Figure 2.3. A *topic* is a named and strongly typed message bus. To send i.e. publish data, a node *advertises* a topic and the according message type to the ROS master. The *ROS master* provides central registration and lookup for all nodes within the graph. Nodes register details of the messages they provide as well as those they subscribe to at the ROS master. A node's *publisher* then publishes the messages to the advertised topic. A node's *subscriber*, interested in receiving messages, subscribes to the corresponding topic at the ROS master. The master acting as a name service in the Computation Graph ensures that publisher and subscriber find each other. The messages itself are then exchanged directly between publisher and subscriber. Every time a subscriber receives a message published by a publisher under a topic it subscribed to, the subscriber's *callback function* is invoked and processes the newly received message. Whereas the just described publish/subscribe mechanism follows a many-to-many communication paradigm, ROS also offers *services*, representing a bi-directional one-to-one request/reply mechanism. The *server* is a node offering a service that it advertises. A *client* sends a request message to the server which is answered with an according reply message. Establishment of the communication between server and client is facilitated by the master as described above.

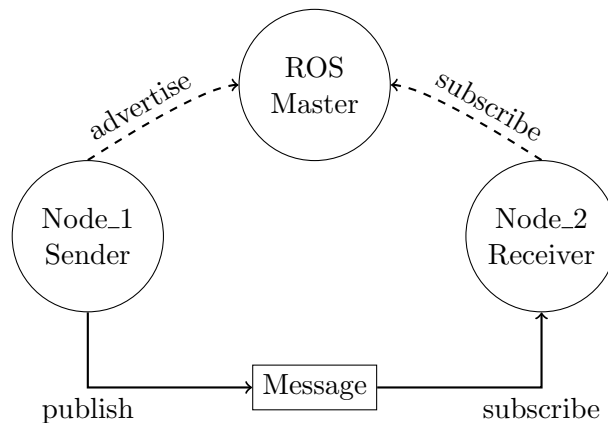


Figure 2.3: ROS Computation Graph Level - Publish/Subscribe Mechanism

Another level of concepts is the *ROS Filesystem level* which specifies the organization of folders and files. The root folder for ROS-related developments is the *workspace*. ROS has an official custom build system called *catkin* that extends the functionality of the usual CMake workflow by combining CMake macros and Python scripts. A *catkin workspace* is a folder structure with a recommended layout as specified in *ROS Enhancement Proposal* REP-128 [43]. It is further organized into several spaces, i.e. source space, build space, development space, install space, and contains according subfolders and for the build process required files as depicted in Figure 2.4.

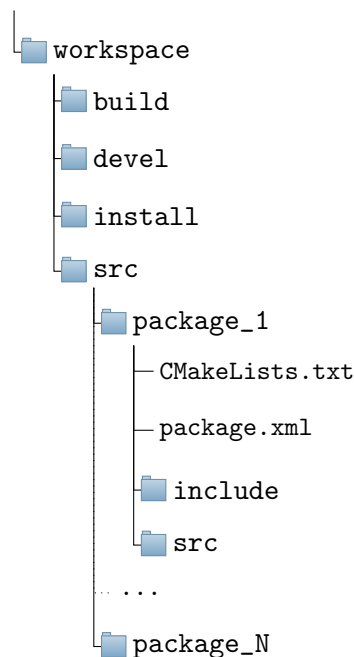


Figure 2.4: ROS Filesystem Level - Recommended Catkin Workspace Layout

The source space is located in the workspaces subfolder `/src` and contains the packages. Software in ROS is organized in *packages* which are the most atomic build item in ROS. They are a structure of folders and files representing for example a node. In its main folder each catkin package contains a *package manifest* in form of a `package.xml` file as specified in REP-140 [44] as well as a `CMakeLists.txt` file describing how to build the code and where to install it. Sometimes packages related to each other are grouped together in a *metapackage*.

ROS installations already include powerful internal software libraries as packages and a wide set of tools. Some of those used within this thesis are shortly introduced hereafter to provide an initial overview. All nodes in this work are implemented in C++ which necessitates usage of *roscpp*, the C++ implementation of ROS, and according C++ *APIs* (*Application Programming Interfaces*).

The *Transform Library* (*tf*) [45] keeps track of multiple coordinate frames over time and maintains their relationship in a tree structure buffered in time. It offers listeners to wait for transforms to become available and functions to calculate transforms between timestamped coordinate frames.

The *Image Geometry Library* enables ROS to interpret images geometrically by using parameters from the camera's *Camera Info* messages that contain a camera's meta-information such as the pixel resolution or projection matrix. It further provides mathematical camera models like the pinhole camera model.



ROS comes with tools for recording, playback and visualization of *.bag* files which is a file format for storing ROS message data. The *rosbag* tool allows to subscribe to ROS topics and store the serialized message data in the same representation that is used in the network transport layer of ROS. A later playback of such a bag file simulates the message feeds as if published at time of the recording.

The tool *RViz* is a powerful 3D visualizer providing a *GUI (Graphical User Interface)* to interactively visualize and analyze combined ROS messages such as camera images, 3D LiDAR point clouds or coordinate transform trees.

Indepth coverage of ROS is available through the project's official online documentation at [wiki.ros.org](http://wiki.ros.org) and in numerous further sources such as books like [46], [47], [42], [48].

## 2.4 Digital Image

A *digital image* is represented as array of numbers. Common digital cameras output two-dimensional raster images with  $X$  rows,  $Y$  columns and  $Z$  channels. Each picture element or *pixel* represents the numerical intensity value  $f(x, y)$  at the discrete spatial coordinate  $(x, y)$  with  $x = 0, 1, 2, \dots, X-1$  and  $y = 0, 1, 2, \dots, Y-1$ . Pixels of gray scale images consist of just one intensity value for each spatial location. In the case of multi-channel images, such as the RGB color image in Figure 2.5, pixels consist of one value for each individual channel which results in three values, one for each channel of the RGB image. Whereas a two-dimensional array or matrix  $\mathbf{I}$  is sufficient to represent a gray scale image, three-dimensional arrays or tensors  $\mathbf{I} \in \mathbb{R}^{X \times Y \times Z}$  will be used for three-channel color images with  $Z$  denoting the channels. For normalized images the intensity values have the range  $[0..1]$  with  $f(x, y) \in \mathbb{R}$ . For integer images the range depends on the bit depth  $k$  dedicated to each individual image channel and comprises  $[0..2^k - 1]$  with  $f(x, y) \in \mathbb{N}_0$ . In a normalized gray scale image the lowest intensity value 0.0 is interpreted as black, the highest 1.0 as white. Any value in between represents an according gray tone.

A normalized single channel digital image is expressed in equation form as

$$f(x, y) = \begin{bmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, Y-1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, Y-1) \\ \vdots & \vdots & & \vdots \\ f(X-1, 0) & f(X-1, 1) & \cdots & f(X-1, Y-1) \end{bmatrix}, \quad (2.1)$$

or equivalently in matrix notation as

$$\mathbf{I} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,Y-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,Y-1} \\ \vdots & \vdots & & \vdots \\ a_{X-1,0} & a_{1,1} & \cdots & a_{X-1,Y-1} \end{bmatrix}. \quad (2.2)$$

Each array element represents an intensity value  $f(x, y)$ . The according spatial location  $(x, y)$  within the image is given by the indices of the array. Multi-channel images consist of several such arrays, usually one for each channel.

For computational purposes an image channel can also be represented as vector  $\mathbf{v}$  which is formed by successively concatenating either all columns of the matrix to form one column vector of size  $XY \times 1$  or all rows to form a respective row vector of size  $1 \times XY$ .



Figure 2.5: Digital Image Representation - RGB Image, Channels And Numerical Array

Further details on digital image representations are given in [49, ch. 2.4.2].

## 2.5 Artificial Neural Network (ANN)

*Artificial neural network* (ANN) is the formal name for *neural network* (NN) which is a form of computation inspired by the structure and function of the brain [50]. It consists of artificial neurons connected by directed weighted connections. These networks can be trained to solve complex tasks by providing examples.

This section gives an overview of some basic principles of ANNs necessary to further discuss and understand SegNet. For a comprehensive treatment of the topics please consult [51], [52], [53], [54].

### Artificial Neuron

*Artificial neurons* are the elementary components of artificial neural networks. Since they are majorly simplified models of its biological counterparts, it might be preferred to call them *computing units* rather than neurons [51]. The mathematical model and structure of a basic computing unit are depicted in Figure 2.6.

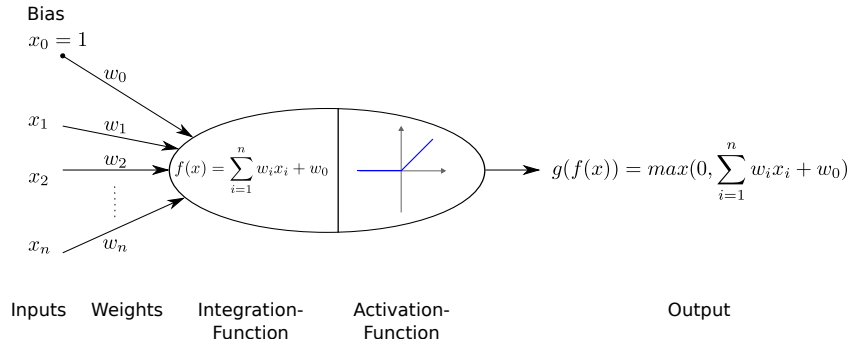


Figure 2.6: Artificial Neuron - Computational Unit - Model And Structure

Within the context of this thesis, each artificial neuron receives  $n$  inputs  $x_i$  which are individually weighted with an associated factor  $w_i$ . All except one of these inputs are either input data fed into a network or outputs from other computing units. The one exceptional input is the *bias*, a constant  $x_0 = 1$  with assigned weight  $w_0 \in \mathbb{R}$ . It is solely connected to the associated unit, thus independent of any other unit. The computing unit then combines the received information to a single value, usually by adding all its inputs. Hence, it has an *input function* or *integration function*  $f(x)$  calculating the sum of weighted inputs:

$$f(x) = \sum_{i=1}^n w_i x_i + w_0, \quad (2.3)$$

or more explicitly

$$f(x) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + w_0, \quad (2.4)$$

or equivalently in matrix notation with a column vector  $\mathbf{x}$  as input

$$f(x) = \mathbf{w}^T \mathbf{x}. \quad (2.5)$$

This sum is then used as argument of an *activation function*  $g$ , which is applied element-wise to calculate the neurons final *output*  $g(f(x))$ . The activation function acts as threshold element and the threshold can effectively be controlled through the bias. The unit

is activated i.e. generates an output only in case the threshold has been exceeded. It is common to choose an activation function, which adds non-linearity and limits or squashes the units output to a desired range. The three major types of activation functions introducing non-linearities that are used in practice are sigmoid, hyperbolic tangent and have-wave rectifier [55, ch.1, p.13]. The *half-wave rectifier*  $f(x) = \max(0, x)$ , shown in Figure 2.7, is the default recommended [53, ch. 6, p. 174] and presently most popular [56, p. 438] non-linear activation function for modern neural networks. A computing unit with a half-wave rectifier as activation function is referred to as *Rectified Linear Unit*, or just *ReLU*.

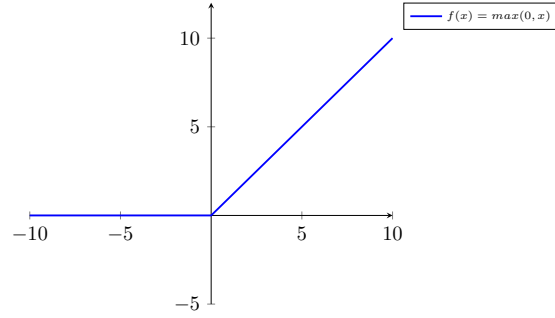


Figure 2.7: Rectified Linear Unit (ReLU) - Activation Function

In general a *computing unit* represents a simple function, mapping an  $n$ -dimensional input to a single output value. A ReLU, an artificial neuron composed of an integration function calculating the weighted sum of inputs as just described and a half-wave rectifier as activation function can be formulated as

$$g(f(\mathbf{x})) = \begin{cases} x, & \text{if } \sum_{i=1}^n w_i x_i > w_0, \\ 0, & \text{otherwise} \end{cases}, \quad (2.6)$$

$$x, w \in \mathbb{R},$$

$$i = 1, \dots, n.$$

With respect to classification, another important activation function is the *softmax function*, or normalized exponential function. It squashes a  $K$ -dimensional input of real values to a  $K$ -dimensional output of real values each in the range  $[0, 1]$ , whereby all output values combined add up to 1. Thus, the softmax functions output represents a probability distribution over a variable with  $K$  mutually exclusive categories or classes.

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^K e^{x_k}}, \quad (2.7)$$

$$x \in \mathbb{R},$$

$$i, k = 1, \dots, K.$$

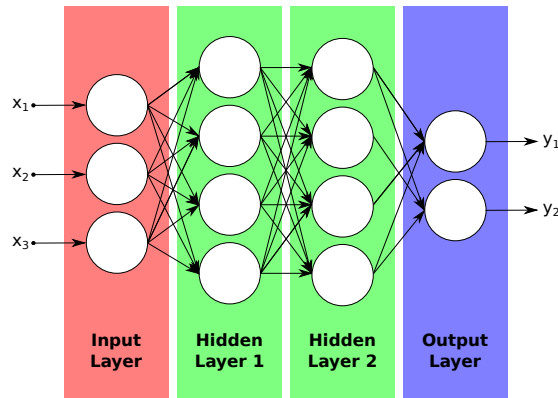
Consequently a strong, confident prediction for a class is given with a probability close to 1, whereas several simultaneous classes with similar probabilities give a weak prediction.

## ANN Architectures

A *network architecture* is a tuple  $(I, N, O, E)$  consisting of a set  $I$  of input sites, a set  $N$  of computing units, a set  $O$  of output sites and a set  $E$  of weighted directed edges. A directed edge  $E$  is a tuple  $(u, v, w)$  whereby  $u \in I \cup N$ ,  $v \in N \cup O$  and  $w \in \mathbb{R}$  [51, ch. 6, p. 125]. It is a chain of function compositions or eventually a composite function.

To allow for simple and efficient matrix vector operations the computing units of ANNs are commonly arranged in *layers*. These layers consist of mutually exclusive subsets  $N_l$  of the set of computing units  $N$ . Thus, any computing unit belongs to solely one layer. Computing units are only connected to units of adjacent layers, not to units within the same layer, and not to units beyond directly adjacent layers. A layer is *fully connected* if all units of a layer are connected to all units of the adjacent layer. From a network architectural point of view there are three basic types of layers. An *input layer* receives the inputs to the network. The last layer which is returning the final output of the network is the *output layer*. Any layers between input and output are *hidden layers*. Networks consisting of  $l$  layers are called  $(l-1)$ -layer networks since the input layer is not counted by convention. Neural networks with a higher number of hidden layers are referred to as *deep networks*.

As already mentioned, directed weighted edges  $E$  are connecting computing units of adjacent layers. All edges between layers have the same direction. An edge  $E$  connecting a unit's output  $u$  in layer  $N_l$  to another unit's input  $v$  in layer  $N_{l+1}$  has the associated weight  $w_{uv}$ . This excludes cyclic connections and results in an architecture referred to as *feedforward* network. Such an ANN can also be represented as *directed acyclic graph* (DAG). An example of a 3-layer fully connected feedforward network is given in Figure 2.8. It is composed of an input layer with 3 inputs, two 4-unit hidden layers and one output layer with two units. Showing the biases of the units is omitted for the purpose of visual clarity.



3-Layer Fully Connected Feedforward ANN with Three Inputs, Two 4-Unit Hidden Layers and One 2-Unit Output Layer

Figure 2.8: 3-Layer Fully Connected Feedforward ANN

The combination and organization of multiple computing units in a network architecture allows the modeling of complex nonlinear mappings. According to the *universal approximation theorem* [57], [58] a feedforward neural network with at least one hidden layer consisting of enough hidden units can approximate any continuous function in  $\mathbb{R}^n$  [53, ch. 6.4.1].

## ANN Training And Inference

A typical application of ANNs is *classification* which is the assignment of a label from a set of classes to a set of inputs, like for example assigning a certain class to each pixel of an image. An ANN is a mathematical model in form of a composite *network function*  $\Phi$  that maps an  $n$ -dimensional input to an  $m$ -dimensional output  $\mathbb{R}^n \mapsto \mathbb{R}^m$ . Correct mapping is determined by choosing the right set of parameters for the network function. In the case of feedforward ANNs with a non-dynamic, fixed network architecture this means finding a proper set of weights. Starting with some initial set of weights and successively adjusting them to find a proper combination is referred to as *training* or *learning*.

A significant feature of ANNs is that they can learn from examples which is called *supervised learning*. This requires a *training set*  $\{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_s, \mathbf{t}_s)\}$ , which is a collection of labeled input sets  $(\mathbf{x}_i, \mathbf{t}_i)$ , consisting of  $s$  ordered pairs of an  $n$ -dimensional input vector  $\mathbf{x}_i$  and an  $m$ -dimensional label output vector  $\mathbf{t}_i$ . By knowing the expected target output  $\mathbf{t}_i$  for the provided input  $\mathbf{x}_i$ , the network function's weights  $\mathbf{W}$  can be adjusted accordingly, thus the network can learn the model from data.

To update the weights towards an optimum, it is inevitable to define a measure of difference between the expected output  $\mathbf{t}_i$  versus the actually observed output  $\mathbf{y}_i = \Phi(\mathbf{W}_i, \mathbf{x}_i)$  of the network. This is achieved by means of a *cost function*  $C(\mathbf{t}, \mathbf{y})$  that quantifies this error. With the commonly used squared error, the cost function can be defined as

$$C = \sum_{i=1}^s \sum_{j=1}^m \frac{1}{2} \|\mathbf{y}_{ij} - \mathbf{t}_{ij}\|^2. \quad (2.8)$$

Here the index  $i$  references the sample of the training set and the index  $j$  the component of the vector. Hence  $\mathbf{y}_{ij}$  is the  $j$ -th component of the network's output vector of the  $i$ -th sample of the training set. The cost function represents the sum of squared errors for each component of the network's output vector accumulated over the complete training set which is essentially the *network's total error*. Applied to the ANN shown in Figure 2.8 the cost function according to Equation 2.8 is  $C(\mathbf{t}, \mathbf{y}) = \sum_{i=1}^s (\frac{1}{2}(\mathbf{y}_{i1} - \mathbf{t}_{i1})^2 + \frac{1}{2}(\mathbf{y}_{i2} - \mathbf{t}_{i2})^2)$ . Figure 2.9 shows the network extended by this cost function.

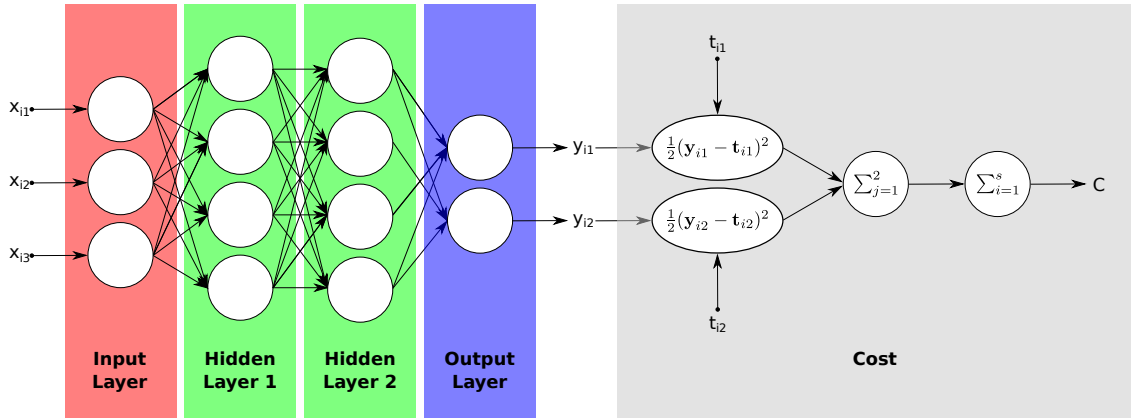


Figure 2.9: 3-Layer Fully Connected Feedforward ANN With Cost Function

Minimizing the cost function minimizes the total error of the network. Therefore, besides a measure for difference, it is necessary to define means of how to properly adjust the weights towards minimizing the network's cost function. For this purpose, iterative gradient-based optimizers are used that drive the cost function to a sufficiently low error value. This requires to calculate the partial derivative of the cost function with respect to each weight in the network.

A function's *gradient*  $\nabla f$  is the vector of partial derivatives of that function's variables.

$$\nabla_{\mathbf{x}} f = \nabla f(\mathbf{x}) = \text{grad}(f(\mathbf{x})) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}. \quad (2.9)$$

The derivative indicates the rate of change or *sensitivity* of a function with respect to a variable's infinitesimally small change  $h$  at position  $x$ .

$$\begin{aligned} \frac{df(x)}{dx} &= \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \\ \Leftrightarrow f(x+h) &= f(x) + h \frac{df(x)}{dx} \end{aligned} \quad (2.10)$$

The gradient of the cost function with regard to its weights  $\frac{\partial C}{\partial w_i}$ , shows what effect the adjustment of each individual weight  $w_i$  of the network has on changing the total error of the network. The negative gradient vector averaged over all samples of the training set indicates the direction of steepest descent in weight space. By adjusting the cost function's weights vector in the direction of this averaged negative gradient vector, as formulated in Equation 2.11, it is possible to decrease the network's total error.

$$\begin{aligned} \mathbf{w}_{new} &= \mathbf{w} - \eta \frac{1}{s} \sum_{i=1}^s \frac{\partial C}{\partial w_i} \\ &= \mathbf{w} - \eta \frac{1}{s} \sum_{i=1}^s \nabla C(\mathbf{w}) \end{aligned} \quad (2.11)$$

The amount of adjustment during each update step is controlled by the *step size* or *learning rate*  $\eta$ . The procedure is repeated until the loss function converges to a sufficiently low value. The just described optimization method of repeatedly computing the gradient and then updating the parameters is known as *gradient descent*. Instead of calculating the gradients for all samples of the complete training set to get the network's total error before updating the parameters, it is common to update the parameters already after calculating the error for just a subset called a *batch* or *mini-batch* of the complete training set. Averaging over randomly shuffled subsets of the training set rather than all samples before updating the parameters, allows to update the parameters more frequently with less accuracy and results in much faster convergence and therefore speeds up the learning process. Updating the parameters each time after calculating the gradient for a single sample, which means the batch size equals one sample, is called *stochastic gradient descent* (SGD). One *epoch* is reached once every sample of the training set has been used during the training.

As described above, the network is a composite function of composite functions which essentially consist of simple functions represented by computing units. One function's output is another function's input. Consequently the *chain rule* of calculus, formulated in Equation 2.12 for scalars and in Equation 2.13 in generalized vector notation, must be applied to compute the networks gradient.

$$z = g(f(x)) = f(y) \frac{dz}{dy} = \frac{dz}{dy} \frac{dy}{dx} \quad (2.12)$$

The generalized chain rule in vector notation for  $\mathbf{x} \in \mathbb{R}^m$ ,  $\mathbf{y} \in \mathbb{R}^n$  and  $\mathbf{z} \in \mathbb{R}$  is

$$\nabla_{\mathbf{x}} z = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T \nabla_{\mathbf{y}} z, \quad (2.13)$$

where  $\left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T$ , the transposed  $n \times m$  Jacobian matrix of  $f(\mathbf{x})$ , is multiplied with the gradient  $\nabla_{\mathbf{y}} z$ . For a detailed description and a further generalization of the gradient to tensors please refer to [53, ch. 6.5.2].

In this context the question regarding the differentiability of a ReLU's activation function given in Equation 2.6 might occur. It is not differentiable but sub-differentiable at  $x = 0$  and defined as

$$g'(f(\mathbf{x})) = \begin{cases} 1, & \text{if } \sum_{i=1}^n w_i x_i > w_0 \\ 0, & \text{otherwise} \end{cases}. \quad (2.14)$$

Differentiability of ReLU activation functions is covered in [53, ch. 6.3]. A thorough treatment of backpropagation in combination with gradient descent as learning algorithm is given in [59], [51, ch. 7], [53, ch. 6.5], [60, lecture 4].

A simple and computationally efficient algorithm to compute gradients of expressions through recursive application of the chain rule is *backpropagation* [59]. It consists of two phases.

The first is the *feedforward* phase or *forward propagation* in which an input  $\mathbf{x}$  is fed into the network and processed layer by layer from input layer  $N_1$  successively to the next adjacent hidden layer  $N_2$  until the output layer  $N_l$  and the cost function, resulting in a scalar cost value  $C$ . All units within a layer are computed in parallel, units in different layers are computed sequentially. The outputs of all units are stored where  $o_u$  denotes the output of unit  $u$ . Each output is forward propagated along an edge  $E$  and multiplied with the associated weight  $w_{uv}$ . The weighted output  $o_u w_{uv}$  serves as input to a unit in the next layer as described above and formulated in Equation 2.3. It can be interpreted as input into a subnetwork starting at edge  $E(u, v, w)$ .

In the second phase, the *backpropagation*, the gradient of the cost function with respect to the inputs is calculated and stored. As just described, the inputs are  $o_u w_{uv}$ . Since  $o_u$  is constant during backpropagation, the gradient of the cost function with respect to just the weights is calculated. Applying Equation 2.9 leads to

$$\frac{\partial C}{\partial w_{uv}} = o_u \frac{\partial C}{\partial o_u w_{uv}}. \quad (2.15)$$

The constant 1 is fed into the network's output layer  $N_l$  and backpropagated through the network in opposite direction, from output through hidden layers to input layer  $N_1$ . All received backpropagated inputs  $o_v w_{uv}$  to a node are added, then multiplied by the node's gradient and finally passed on to the connected nodes of the previous adjacent layer  $N_{l-1}$ . This local application of the chain rule is repeated for each node, layer by layer, until the input layer has been reached and thus the gradient of the cost function with respect to its weights has been calculated.

Then gradient descent is used to update the weights according to Equation 2.11. Training therefore consists of repeating the backpropagations forward phase to calculate the network's error, the backward phase to calculate the cost function's gradient with respect to the weights and gradient descent to reduce the network's error through updating the weights until the cost value is sufficiently low.



The learning progress can be evaluated by plotting the cost after each epoch as depicted in Figure 2.10 (a). Overall the cost should decay over time which generally happens exponentially [61]. How well the network's parameters have been determined to correctly represent the desired network function, is evaluated e.g. after each epoch through the resulting *prediction accuracy*, which is the fraction of correct predictions over all available samples in a set or some other suitable metric. Besides calculating the prediction accuracy for the training set, it is also calculated for the *validation set*, which is a collection of labeled inputs, just like the *training set* described above. However, the network has not directly been exposed to the samples of the validation set during training and therefore those samples can be used to tune hyperparameters and to evaluate how well the network can predict the correct label for an unknown input, which means how well it can *generalize*. One of the major challenges in designing and training ANNs is to find an optimal fit to represent the model well, yielding good generalization. The inherent high capacity of deep networks, due to the high number of parameters i.e. computing units, requires measures to avoid *overfitting*. If the capacity of the network is too high compared to the provided training data and it has been trained for too many epochs, the network might have learned the training samples including noise rather than learning the model. In that case the accuracy for the training set is high, whereas the validation set accuracy is low and the difference between both indicates the amount of overfitting. The relation between training accuracy, validation accuracy and overfitting is shown in Figure 2.10 (b). The network suffers from *underfitting* if it has not been trained properly or its capacity is too low to represent the model. Continuing low accuracy for training set as well as validation set during training indicates underfitting.

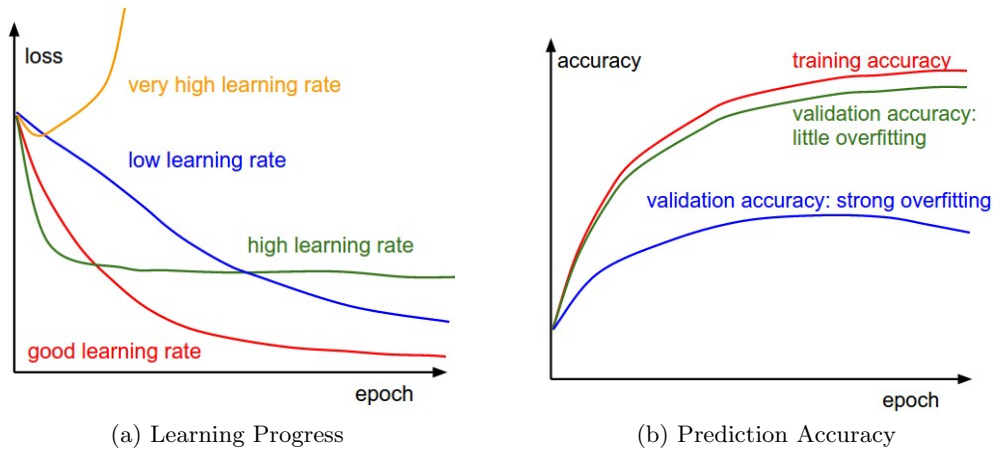


Figure 2.10: Network Learning Progress And Prediction Accuracy [61]

There are endless techniques to monitor and control the training of neural networks, especially to prevent overfitting. A thorough treatment of different techniques is given in [61] and [62], some techniques beyond basic gradient descent are described in [55, ch. 1, ch. 4].

After the network has been trained, it is deployed to run *inference* using the previously learned parameters to process unknown inputs. To assess the networks final prediction and generalization performance after the completed training phase, yet another set of labeled inputs, the *test set* is used. In contrast to the validation set which is used during training to tune the hyperparameters, the test set is only used once at the end, after the training phase. Besides a suitable architecture and representative data sets, proper initialization and especially means to prevent overfitting are important to get a well performing deep neural network.

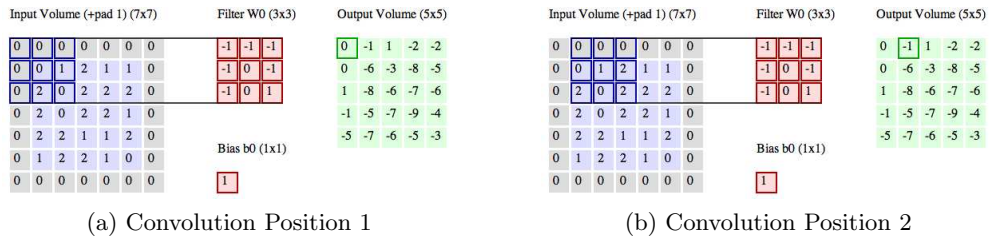
## 2.6 Convolutional Neural Network (CNN)

A *convolutional neural network* (CNN), also called *ConvNet*, is a neural network that uses convolution in place of general matrix multiplication in at least one of its layers [53, ch. 9]. The main building blocks of CNNs in this thesis are input, convolutional, ReLU, pooling and fully connected layers. Further distinguishing features of CNNs are the use of computing units arranged in 3 dimensions, local connectivity and parameter sharing which will all be described in the following paragraphs. Where it seems suitable for clarity reasons, concepts will be explained in lower dimensions first before extending those to meet the volumetric nature of CNN computing units.

ConvNets are designed to process input data with grid-like topology such as images represented in form of numerical arrays as described in Section 2.4. Convolution, denoted with an asterisk  $*$ , is a mathematical operation. For a two-dimensional input like an image  $\mathbf{I}$ , it is defined as

$$\mathbf{I}'(x, y) = \mathbf{W} * \mathbf{I} = \sum_m \sum_n \mathbf{W}(m, n) \mathbf{I}(x - m, y - n). \quad (2.16)$$

Convolving an input  $\mathbf{I}$  with a *kernel*  $\mathbf{W}$  results in a so-called *feature map*  $\mathbf{I}'$ . The  $M$  rows of the kernel  $\mathbf{W}$  are indexed by  $m$  and the  $N$  columns by  $n$ , everything else follows Section 2.4 for now. Thus, *convolution* is the process of generating a activation or feature map by moving a 180°-rotated filter mask, i.e. kernel, over an image and computing the sum of element-wise products at each pixel location. Depending on the filter used, different features of the input are detected and consequently different feature maps are generated. The filter's coefficients are the learnable weights and thus the filters can be trained to detect certain features. Convolving an input image with different filters generates different feature maps with different detected basic features such as edges. Resulting feature maps from a previous layer serve as inputs to convolutions of successive layers, generating feature maps representing more complex features, motifs, objects and so on. The filter mask typically has square dimensions with  $M = N$  and an odd number of rows and columns. To apply the filter mask to each pixel of the image, a *padding* of  $\frac{M-1}{2}$  rows has to be added to the top and to the bottom of the image as well as a padding of  $\frac{N-1}{2}$  columns to the left and right. It is common to use rows and columns of zeros and referring to it as *zero-padding*. Figure 2.11 depicts two of the  $5 \times 5 = 25$  filter positions of a two-dimensional convolution of a zero-padded (gray) input image (blue) with a kernel (red) and the resulting activations or feature map (green). It is a modified version of the convolution demo from [63].



Input (Blue):  $\mathbf{I}=5 \times 5$ , Zero-Padding (Gray):  $\mathbf{P}=1$ , Kernel (Red):  $\mathbf{W}=3 \times 3$ , Bias (Red):  $\mathbf{B}=1 \times 1$ , Feature Map (Green):  $\mathbf{O}=5 \times 5$  [63, Generated With Modified JavaScript Code]

Figure 2.11: CNN - 2D Convolution

Typical kernel sizes are  $3 \times 3$ ,  $5 \times 5$  or  $7 \times 7$  and thus comparatively small with regard to the size of the image or input from a previous layer. The sum of element-wise products, i.e. dot products, calculated during convolution just involves spatially limited image patches of the

same size as the kernel. This *local connectivity* means that computing units in convolution layers of CNNs are not fully connected to the previous layer but only to regional patches. Regional patches on the input layer represent the *receptive field* of connected computing units in succeeding deeper layers. This *sparse connectivity* is one of the main ideas of CNNs and saves computational resources by tremendously reducing the amount of parameters to calculate and store. Using the same kernel coefficients and bias at each spatial position, that means for all the convolution operations within the same feature map or layer, is a form of *parameter sharing* which further reduces the storage requirements. Figure 2.12 exemplifies the concept of sparse connectivity and the significant reduction of connections, and therefore parameters, in the CNN. Compared to the fully connected layers of the ANN, the units of the hidden layers of the depicted CNN are each connected through a one-dimensional kernel of size 3 which significantly reduces the parameters from 112 to 64, even for such a small dummy network. With parameter sharing this further reduces to 22 parameters. For images from one of the SatCam cameras in the MIG each fully connected unit of the first hidden layer would have  $(1280 \times 720 \times 3) \text{ weights} + 1 \text{ bias} = 2\,764\,081$  parameters whereas using a convolution with a  $3 \times 3$  kernel results in  $3 \times 3 + 1 = 10$  parameters per unit.

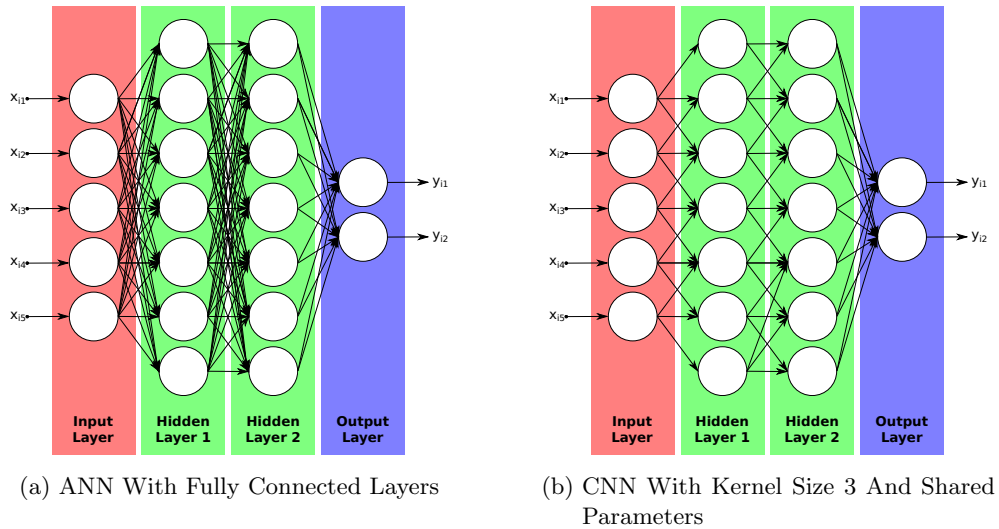


Figure 2.12: CNN - Sparse Connectivity And Parameter Sharing

The size of the receptive field on the input layer is mainly determined by the kernel size and the amount of successive dense convolutions. The larger the kernel size and/or the more successive convolutions, the larger the computing units receptive field and consequently the bigger the spatial context. As shown in figure 2.13 the receptive field increases for computing units in deeper layers of CNNs. Whereas the receptive field of the gray marked unit in hidden layer 1 comprises 3 units of the input layer, the receptive field of the gray unit in hidden layer 2 increased to a size of 5 units.

The increase of the receptive field towards neurons in deeper layers is even larger if layers include strided convolution or pooling. *Stride* is the step size or amount of displacement of the filter mask over the image array between each application of the filter operation. For example convolution with a stride of 2 means moving the kernel 2 pixels on the input before the convolution is applied. Consequently the resulting output has half the spatial size in the dimension in which the stride of 2 has been used.

*Pooling* replaces the regional output of the network. Commonly used is *max-pooling* which returns just the maximum output of a region. Therefore max-pooling over spatial regions produces invariance to translations to a certain extend. Pooling over outputs of convolu-

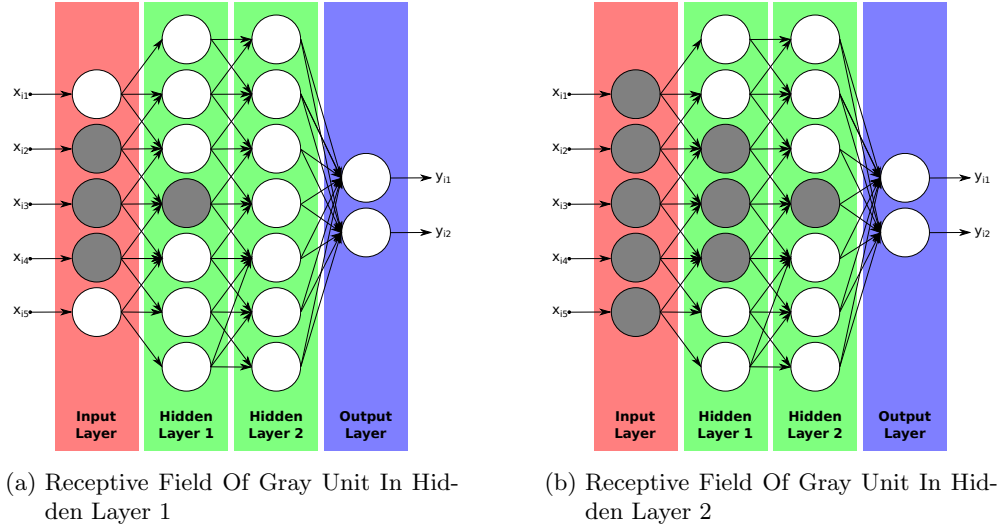


Figure 2.13: CNN - Increased Receptive Field And Spatial Context In Deeper Layers

tions produces invariance to further transformations. Pooling with a stride  $> 1$  is used for non-linear down-sampling. Applying strided max-pooling to a region of  $2 \times 2$  pixels with a stride of 2 results in an output which is horizontally and vertically half the size of the input and consists of output pixels representing just the pixel with the highest value of each according  $2 \times 2$  input region.

As mentioned in the beginning of this section, computing units of ConvNets are arranged in three dimensions, width  $X$  and height  $Y$  being the two spatial ones, and depth  $Z$  adding the volumetric dimension. Whereas the depth of the network is described by the amount of layers it consists of, the depth of a CNN computing unit is given by e.g. the number of channels of an image or the amount of filters or feature maps. As outlined in Section 2.4, color images usually consist of three channels. An according input volume (red) of a color image with a spatial resolution of  $X_1 \times Y_1$  pixels  $\times Z_1$  channels is visualized in Figure 2.14, together with a filter (saturated red) with dimensions  $M \times N \times Z_1$  and a convolution layer (green) with the dimensions  $X_2 \times Y_2 \times Z_2$ .

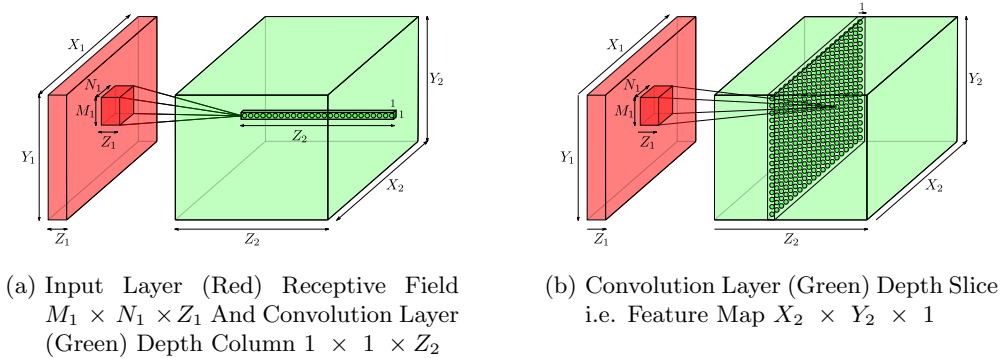
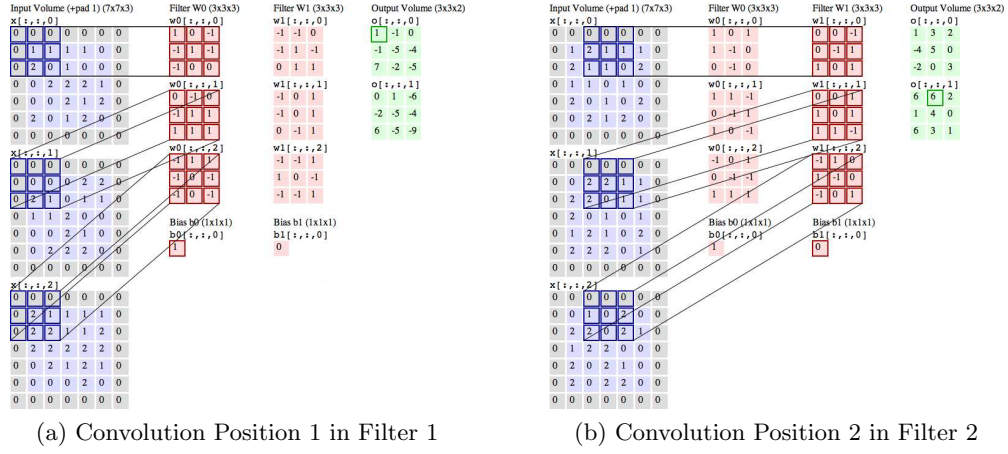


Figure 2.14: CNN - 3-Dimensional Computing Units

The shown input layer has a depth of  $Z_1 = 3$  and the convolution layer has a depth of  $Z_2 = 25$ , thus 25 resulting feature maps. The stack of different filters used to extract different features from an input resulting in the according amount of feature maps along the outputs depth axis  $Z_2$  is called *filter bank*. Figure 2.14 (a) shows a depth column of the

convolution layer (ConvLayer). Each computing unit of a *depth column* in the convolution layer is connected to the the same receptive field, that means only to a local spatial region of units in the input layer, but always to the full depth of the input layer, e.g. to all channels of the image. Connections and according computations always cover the entire depth of the input volume. The applied filter coefficients, i.e. weights and bias, change along the depth of the output volume. Each depth slice, illustrated in Figure 2.14 (b), represents a feature map. Parameter sharing assumed, the receptive field changes but the same coefficients are applied for each dot product within the same *depth slice*.

To appropriately cover the CNN computing unit's volumetric arrangement, Equation 2.16 is simply extended with an index representing the additional dimension and an according summation. This and other modified equations to mathematically express e.g. processing batches of inputs of including the stride are given in [53, ch. 9.5]. Figure 2.15 shows an extended illustration of the convolution demo from Figure 2.11, now with a three-channel input layer, a convolution layer with two three-dimensional filters and a stride of 2.



Input (Blue):  $I=5 \times 5 \times 3$ , Zero-Padding (Gray):  $P=1$ , Kernel (Red):  $W=3 \times 3 \times 3$ , Bias (Red):  $B=1 \times 1 \times 1$ , Feature Map (Green):  $O=3 \times 3 \times 2$  [63, Modified JavaScript Source Code]

Figure 2.15: CNN - 3D Convolution

Following the notation above, the dimensions of a convolution layer's output volume in a CNN are calculated with the following equations:

$$X_2 = \frac{X_1 - N + 2P}{S} + 1, \quad (2.17)$$

$$Y_2 = \frac{Y_1 - M + 2P}{S} + 1, \quad (2.18)$$

$$Z_2 = \text{number of filters.} \quad (2.19)$$

Padding is treated as spatial parameter and is not effected by the additional dimension. It is equally applied to all spatial slices of an input volume. To not reduce the spatial dimensions of the output of a convolution with a stride of  $S = 1$ , the amount of padding on each spatial side is calculated as

$$P_X = \frac{N - 1}{2}, \quad (2.20)$$

$$P_Y = \frac{M - 1}{2}. \quad (2.21)$$

The applicable stride is constraint such that

$$X - N + 2P \bmod S \stackrel{!}{=} 0, \quad (2.22)$$

$$Y - M + 2P \bmod S \stackrel{!}{=} 0. \quad (2.23)$$

This leads to the following equations for calculating  $par_{FC}$ , the number of parameters in an FC layer. Substitution of Equations 2.17 and 2.18 into 2.24 results in Equation 2.26.

$$par_{FC} = \sum_{N_l} (X_2 * Y_2 * Z_2) * (N * M * X_1 + 1) \quad (2.24)$$

$$= \sum_{N_l} (X_{N_{l+1}} * Y_{N_{l+1}} * Z_{N_{l+1}}) * (N_{Filter} * M_{Filter} * X_{N_l} + 1_{Bias}) \quad (2.25)$$

$$= \sum_{N_l} \left[ \left( \frac{X_{N_l} - N_{Filter} + 2P}{S} + 1 \right) * \left( \frac{Y_{N_l} - M_{Filter} + 2P}{S} + 1 \right) * Z_{N_{l+1}} * (X_{N_l} * Y_{N_l} * Z_{N_l} + 1_{Bias}) \right] \quad (2.26)$$

With parameter sharing, as described in Section 2.6, the same weights and biases are used for all units of a depth slice. When using parameter sharing the number of parameters in a layer  $par_{PS}$  is calculated as formulated in Equation 2.27.

$$par_{PS} = \sum_{N_l} \left[ Z_{N_{l+1}} * (X_{N_l} * Y_{N_l} * Z_{N_l} + 1_{Bias}) \right] \quad (2.27)$$

The memory requirements  $mem$  for each layer of the network are calculated with Equation 2.28, whereby  $TypeSize$  represents the memory size of each parameter in bytes. Thus a 32-bit parameter has a  $TypeSize$  of 4. The memory requirements are one of the major limiting factors, since especially deep network architectures deploy a comparatively huge amount of parameters.

$$mem = \sum_{N_l} X_{N_l} * Y_{N_l} * Z_{N_l} * TypeSize \quad (2.28)$$

Training and inference of CNNs are performed as described in Section 2.5. The gradients are backpropagated through the CNN and all parameters, i.e. weights in the filter banks, are updated during training.

Regarding the architecture, CNNs commonly consist of building blocks of type input, convolution, ReLu, pooling and FC which are all explained in this chapter. An example CNN is detailed including its architecture in Section 3.2.



## 2.7 Point Cloud

A *point cloud* is a set of data points in space. Within the scope of this thesis, points and hence point clouds are defined in  $\mathbb{R}^3$ , a three-dimensional Euclidean space formed by three real Cartesian coordinates. Each point represents a unique location in this Euclidean space specified by an ordered triplet of coordinates (x,y,z). The point clouds inspected, originated from LiDAR scanners which send out pulsed laser light and measure the pulses reflected by objects to determine their distance. Thus the resulting point clouds are sparse and represent sampled space in a finite set of points in a 3D frame of reference. Depending on the scene the number of reflected points varies.

Even though there are many similarities to the digital images described in Section 2.4, due to their sparse nature and scene-dependent varying size, it would be highly inefficient to store point clouds in an image-like array in which the spacial location is given by the indices of the array. Therefore a typical digital representation of point clouds are vectors of spatially unorganized points and varying size. The *points* as data types can be N-dimensional and often contain other properties besides spatial location such as intensity or color.

A common file format for storing point clouds is *PCD (Point Cloud Data)* with file extension *.pcd*. It consists of a small header that describes the point type, the number of elements and encoding as well as a list of the cloud's points.

Depicted in Figure 2.16 is the visualization in ROS Visualizer (RViz) of a point cloud and the images of the left, front, right and rear cameras captured with the MIG. The point cloud's coloring represents the intensity.

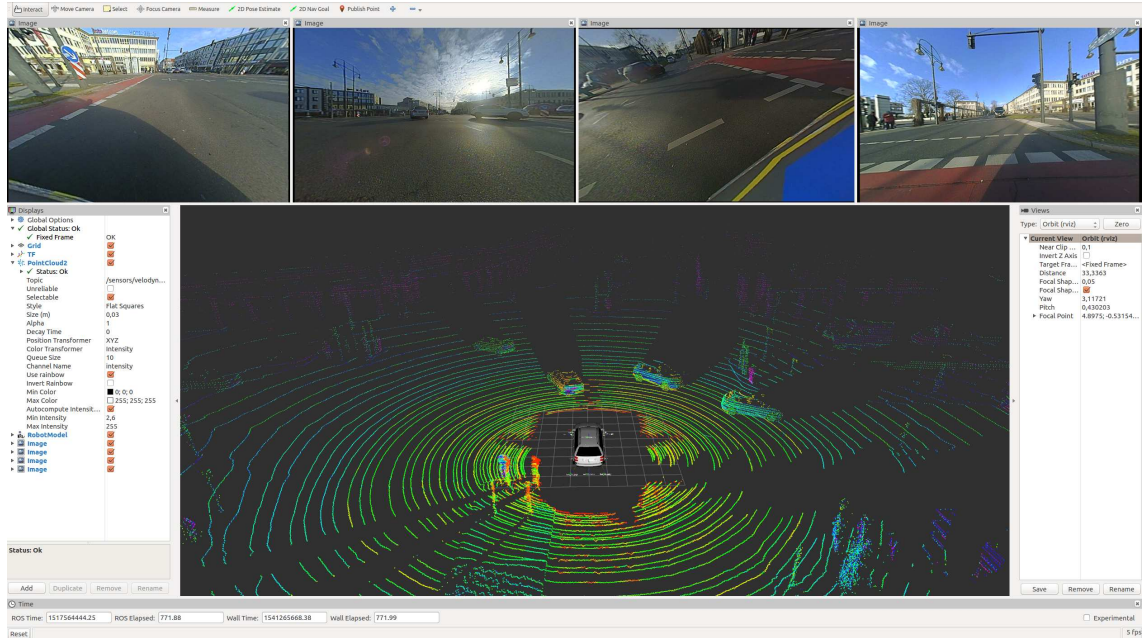


Figure 2.16: MIG Autonomous Vehicle - RViz Visualized Point Cloud And Camera Images

Further information on point clouds can be found on [pointclouds.org](http://pointclouds.org) and in [42, ch. 10].

## 2.8 Spatial Descriptions And Transformations

In order to describe three-dimensional space and the location of points as well as the orientation of point clouds in it, a Cartesian coordinate system, as introduced in Section 2.7, is defined with its origin as reference point in world space.

Such a combination of a coordinate system and a reference point is also called *frame*. This however introduces some ambiguity since a digital image is also commonly referred to as frame. The respective meaning of the term frame should nonetheless be clear from the context. A frame is defined by a set of four vectors, one position vector  $P_{ORG}$ , identifying the origin in world space, and three orientation vectors  $R_x$ ,  $R_y$ ,  $R_z$ , representing the principle axes of the Cartesian coordinate system.

Where a point itself has no spatial extension, objects represented by a set of different points do. Therefore a position as well as an orientation of objects i.e. their representing point clouds can be given. An individual frame is attached rigidly to an object to unambiguously describe position and orientation of other objects from the respective object's point of view. *Mapping* allows to change a point cloud's description from one frame to another, where the points' locations in world space stay the same but their coordinates change according to the target frame. It is also possible to change a point's location in world space by transforming it from one frame to another.

Provided homogeneous vectors are used for world and respective frame points, a general tool to represent frames in form of a 4 x 4 matrix is the *homogeneous transform*. Depending on the deployment it is used as description of a frame, as transform mapping to transform the coordinates between frames or as transformation operator to transform the points from one frame to another. The principle structure of the homogeneous transform expressed as 4 x 4 matrix  $\mathbf{T}$  is

$$\mathbf{T} = \left[ \begin{array}{ccc|c} R_{xyz} & & & P_{ORG} \\ 0 & 0 & 0 & 1 \end{array} \right]$$

whereby  $R_{xyz}$  represents the *rotation matrix* based on the coordinates systems principle axes and  $P_{ORG}$  is the *translation* vector of the coordinates system's origin. This matrix combines orientation/rotation, position/translation and scale. Even other perspective operations can be described when the last row is other than 0 0 0 1.

Frames represent spaces as well as objects within these spaces and transforms between frames i.e. spaces are used to change the representation of a space and its objects. The subsequently described spaces and transforms are used in this thesis. Table 2.5 gives an overview of the spaces.

Frame	Dimension	Origin	Axes Direction		
			x	y	z
World Space	$\mathbb{R}^3$	ground level	right	forward	up
Vehicle Space	$\mathbb{R}^3$	ground, mid.front axle	forward	left	up
Velodyne Space	$\mathbb{R}^3$	device base	forward	left	up
Camera Space	$\mathbb{R}^3$	principle point	right	down	forward
Screen Space	$\mathbb{R}^2$	canvas center	right	up	—
NDC Space	$\mathbb{R}^2 \in [0..1]$	bottom left corner	right	up	—
Raster Space	$\mathbb{N}^2$	top left corner	right	down	—

Table 2.5: Overview Coordinate Spaces



The *world space* is defined as right-handed  $\mathbb{R}^3$ -space with the right-pointing X-axis and the forward-pointing Y-axis forming the ground-level plane and the Z-axis pointing upwards. The *vehicle space*, which is also referred to as *base link*, is represented by the frame that is attached to the autonomous vehicle's reference point. In case of the MIG, the reference point is on the ground below the center of the front axle. The *velodyne space* and the *camera spaces* are defined with respect to the frames attached to the respective device.

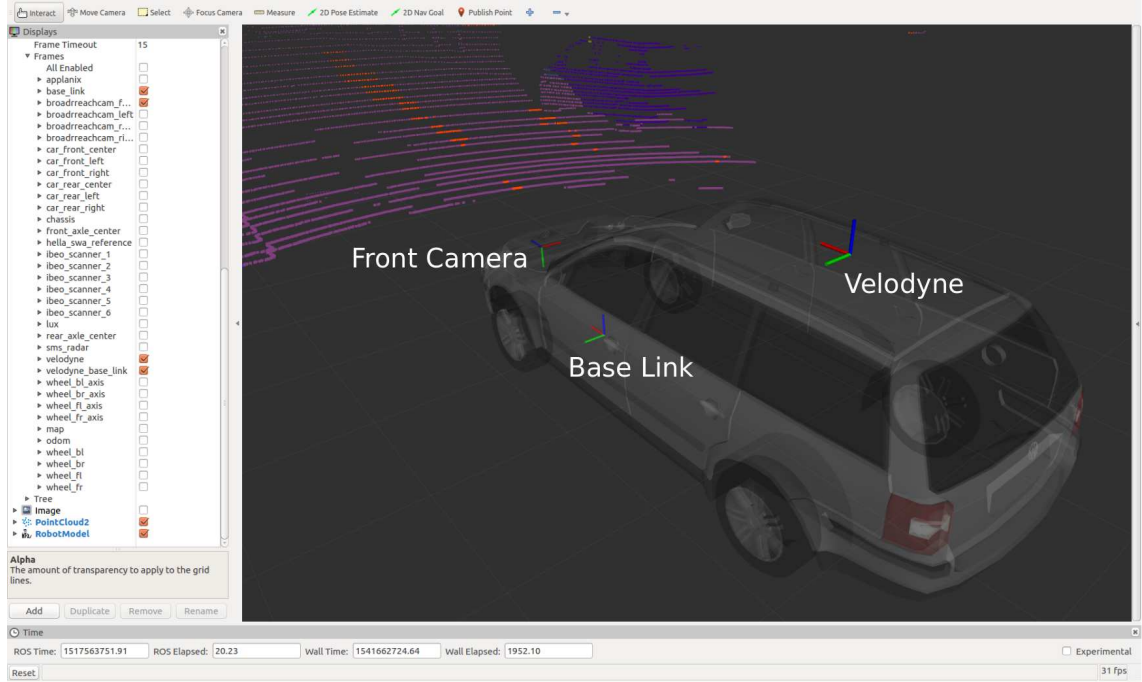


Figure 2.17: MIG Autonomous Vehicle - Frames Attached To Base Link, Velodyne and Front SatCam

Digital cameras are mapping points from  $\mathbb{R}^3$ -space to a two-dimensional image space. The camera model deployed in this work is the *pinhole camera model*. Figure 2.18 depicts its geometry with capital letters being coordinates in  $\mathbb{R}^3$ -space and lower case letters in two-dimensional space.

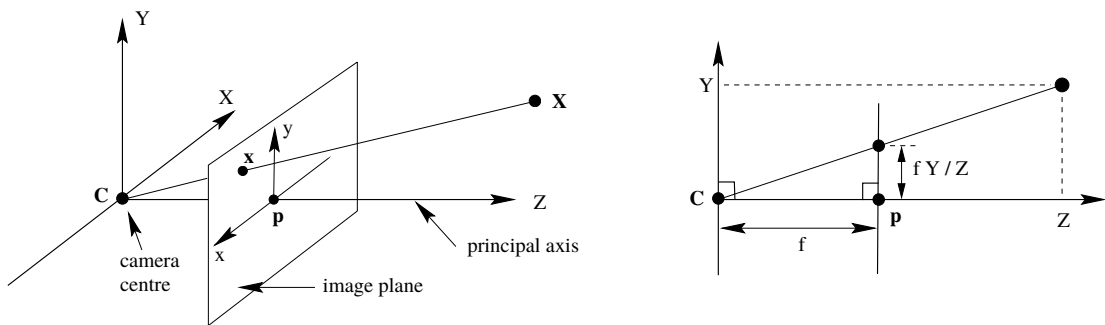


Figure 2.18: Pinhole Camera Model Geometry [64]

The *screen space* is such a two-dimensional space, representing points that lie in an *image plane* that is perpendicular to the camera's principle axis and has its center  $p(x, y)$  where the principle axis meets the image plane. The camera's *principle axis* corresponds with the camera frame's z-axis. Even though these spaces are in principle unlimited, the captured and thus represented space is eventually limited by the device's viewing frustum, described

by the horizontal and vertical field of view and the near and far ranges. Consequently the screen space is clipped and the resulting *image canvas* is limited to those areas of the infinite image plane, that are visible to the camera. This leads to *NDC (Natural Device Coordinates) space* with coordinates normalized to the image canvas, originating at its bottom left corner. In *raster space*, also called *image space*, the  $\mathbb{R}^3$ -coordinates of the projected points are assigned to the nearest  $\mathbb{N}^2$ -pixel coordinates of a rastered digital image. The frame's y-coordinate is pointing downwards, the x-coordinate points to the right and the origin is at the image's top left corner. Pixel locations in a digital image are commonly denoted as (u,v) tuples instead of using x and y.

A camera and its internal parameters are described by the *intrinsic camera matrix*  $K$ :

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.29)$$

The matrix is valid for the unrectified camera image and includes the principle point's coordinates  $p_x$  and  $p_y$  as well as the focal length's coordinates  $f_x$  and  $f_y$ . The *focal length* is the distance between camera center  $C$  and principle point  $p$ .

The *extrinsic camera matrix*  $[R|t]$ , as formulated in Equation 2.30, defines the camera's relation to another frame such as the world space frame. It is the combination of the rotation matrix  $R$ , to transform the coordinate frame's axes, and the translation vector  $t$ , to relocate the origin.

$$[\mathbf{R}|\mathbf{t}] = \begin{bmatrix} r_{x,1} & r_{x,2} & r_{x,3} & t_x \\ r_{y,1} & r_{y,2} & r_{y,3} & t_y \\ r_{z,1} & r_{z,2} & r_{z,3} & t_z \end{bmatrix}. \quad (2.30)$$

With the *generalized pinhole camera model* being  $\mathbf{P} = \mathbf{K} [\mathbf{R}|\mathbf{t}]$  the perspective projection of points  $[X, Y, Z]^T$  from  $\mathbb{R}^3$  world space to pixel  $[u, v]^T$  in  $\mathbb{R}^2$  screen space is defined as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [\mathbf{R}|\mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.31)$$

$$\Leftrightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{x,1} & r_{x,2} & r_{x,3} & t_x \\ r_{y,1} & r_{y,2} & r_{y,3} & t_y \\ r_{z,1} & r_{z,2} & r_{z,3} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \quad (2.32)$$

Further information on spatial descriptions and transformations or multiple view geometry and camera models is given in [65] and [64]. The calibration of the MIG's cameras is detailed in the master thesis [35].

# Chapter 3

## Approach

This chapter begins with presenting an overview of the complete system and identifies its main components representing the approach to reach the Goals 1 to 3 stated in Section 1.3. The section thereafter introduces SegNet, the selected deep neural network architecture, including its peculiarities and contextual suitability. The remainder of the chapter details the implementation, integration and usage of the approach’s system components. For implementation-specific insights beyond the description provided in this chapter, it is recommended to refer to the system component’s source code and its thorough code documentation.

### 3.1 System Overview And Main Components

In order to reach the goals stated in Section 1.3 the approach described hereafter is pursued. The MIG introduced in Section 2.2 represents the real world autonomous vehicle and is as such the ultimate target platform for development and implementation work. This entails constraints set by the platform’s hardware as well as software and their configuration and setup, detailed in the respective Tables 2.2, 2.3 and 2.4. For the hardware this means the TE SatCams provide the 2D images, the Velodyne HDL-64E S2 LiDAR generates corresponding 3D point clouds and the laptop runs all necessary software. Concerning the software, the ROS-based AutoNOMOS Software framework in combination with additional ROS packages, like the drivers for the SatCam camera and the Velodyne scanner, constitute the target platform for implementation work. The deep artificial neural network architecture SegNet, which is presented in Section 3.2 and further detailed in Appendix A, is chosen for pixelwise semantic segmentation of the 2D camera images. An implementation of SegNet is available in a modified version of the deep learning framework Caffe. Therefore, Caffe represents the deep network technology. It is introduced in Section 3.2 and detailed in Section 3.3.

Consequently, two ROS packages are developed and added to the framework’s catkin workspace to tackle the goals of this thesis. The ROS package *caffe\_SegNet\_cuDNN5* provides a deep neural network framework in fulfillment of goal 1. It is a support package without own nodes. Another ROS package, called *awid\_SemSeg*, contains two nodes. Firstly, a node for semantic segmentation of 2D images, targeting goal 2, therefore named *SemSegImg*. Secondly, a node for cross-modal transfer of semantic labels from 2D images to 3D points, aiming at goal 3, titled *SemSegPnt*. An according system overview is given in Figure 3.1. The components developed and implemented as part of this thesis are highlighted in blue. Due to ROS naming conventions [26], names of packages and nodes are all *lowercase\_underscored* in the implementation. For better readability however, *mixedCase* is used throughout this work.

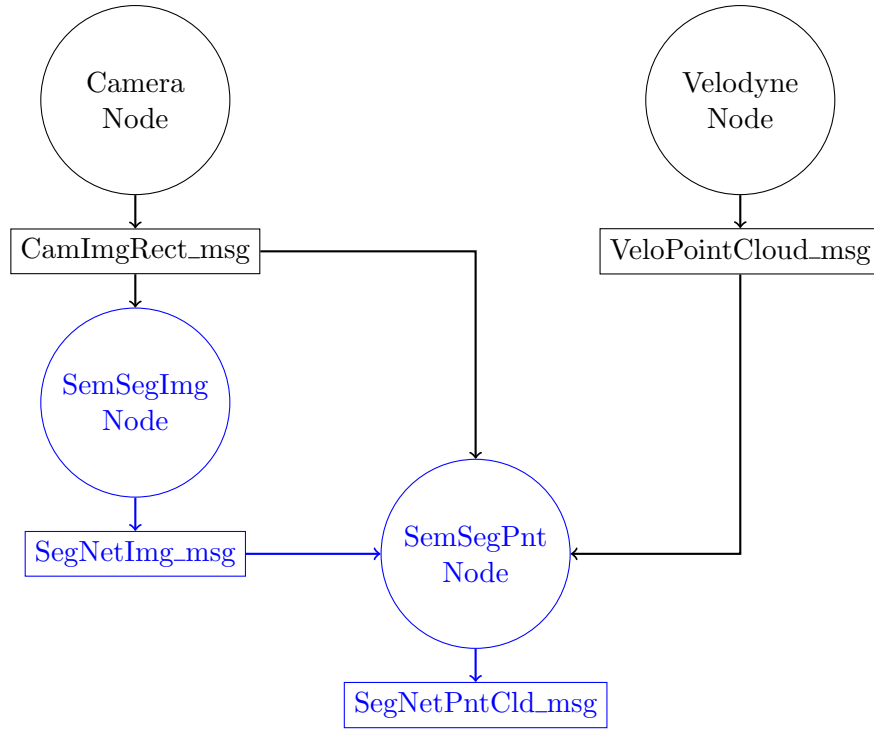


Figure 3.1: Implementation Approach - Overview System And Components

With the basic components selected, the thesis' goals from Section 1.3 can be concretised and rephrased to:

1. Integration of a catkin CAFFE package into MIG's ROS workspace.
2. Development and implementation of a MIG-ROS package using SegNet for pixelwise semantic segmentation of 2D SatCam images.
3. Development and implementation of a MIG-ROS package for cross-modal transfer of pixelwise semantic labels from 2D images to 3D Velodyne LiDAR point clouds.

### 3.2 SegNet - CNN For Semantic Segmentation Of Images

**SegNet** [28], [19] is a deep fully convolutional neural network architecture for semantic multiclass pixelwise image segmentation. It was designed by members of the Computer Vision and Robotics Group at the University of Cambridge, UK as comparatively smaller and faster architecture for memory-constrained real-time applications such as autonomous driving.

A highlevel overview of SegNet’s principle architecture is illustrated in Figure 3.2. Key elements of the network’s architecture are visualized as graphs in Figure A.1 in Appendix A. The appendix also contains a detailed listing of the arrangement of all layers and their number of respective parameters and memory requirements in Tables A.3 and A.4. All visualizations of the SegNet architecture follow the color scheme in Table A.1. The network’s classes are colored in accordance with the scheme listed in Table A.2.

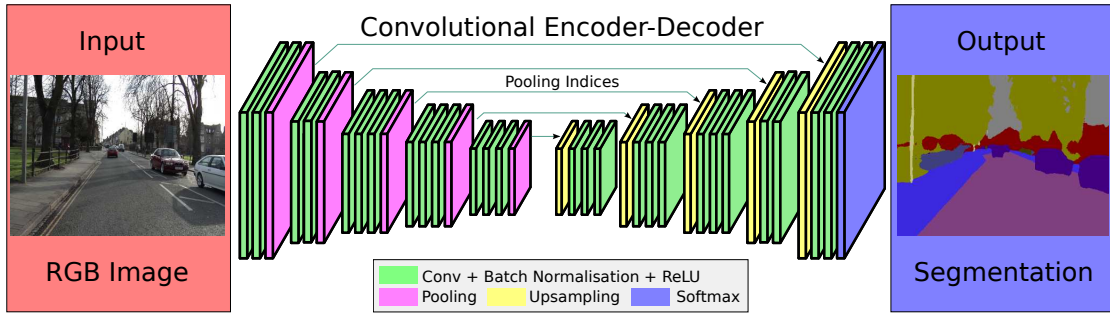


Figure 3.2: SegNet Architecture Illustration, adapted from [28]

SegNet is composed of a data input layer, a stack of 13 encoders followed by a stack of 13 corresponding decoders and a final softmax layer for pixelwise multiclass image labeling. The network’s encoder stack is topologically identical to the 13 convolutional layers of the VGG16 network described as VGG ConvNet Configuration D in [66].

Each *encoder* performs dense convolutions with a 3 x 3 kernel, a zero-padding of 1 and a kernel stride of 1 for a set of 64 up to 512 filters, followed by feature map batch normalization, element-wise  $\max(0, x)$  ReLU non-linearity and a non-overlapping 2 x 2 max-pooling with a stride of 2, thus resulting in a set of 64 to 512 feature maps, each horizontally and vertically downsampled by factor 2 compared to the spatial input dimensions, and a set of encoder max-pooling indices.

Each *decoder* reuses the indices stored during the max-pooling step of the corresponding encoder to perform non-linear upsampling, convolution with a trainable filter bank and batch normalization. The decoders use the same values as the encoders for kernel size, padding and strides. Reversing the downsampling by horizontally and vertically upsampling the feature maps with factor 2 creates sparse feature maps. Convolution of these sparse maps with the trainable filter bank results in according dense feature maps. Storing the encoder max-pooling indices and reusing them during decoder upsampling preserves spatial information and improves accuracy when mapping the lower resolution encoder feature maps to higher resolution decoder feature maps. The last decoder’s output has the spatial dimensions of the network’s initial input and a depth of 12. Each of the depth slices represents one class and contains the class probabilities for each individual pixel.

The last decoder’s output is fed into the networks final layer, a soft-max layer, representing the networks *classifier* which assigns the class with the maximum probability to the pixel, creating the full input resolution pixelwise segmented image as the network’s output. Coloring each of the output image’s pixels according to its class affiliation, creates a *SegNet-colored image*.

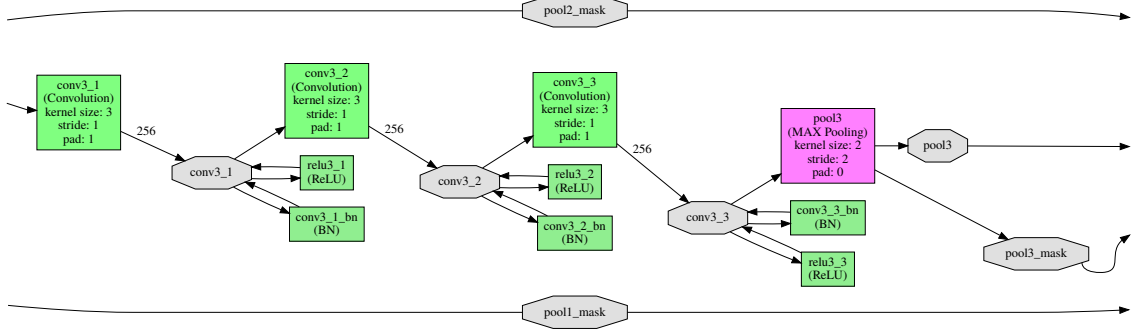


Figure 3.3: SegNet Encoder - Consecutive Convolution Layers

Stacking directly consecutive convolution layers before applying pooling, as depicted in Figure 3.3, allows to include spatial context while still significantly reducing the number of network parameters. As explained in Section 2.6, this architectural design allows growing the receptive field and thus spatial context with network depth while significantly reducing the number of network parameters. A stack of three consecutive convolution layers with a  $3 \times 3$  convolution kernel and a convolution stride of 1 represents the same effective receptive field as one convolution layer with a  $7 \times 7$  kernel. However, with  $C$  denoting the number of channels i.e. depth slices, where  $C_{input} = C_{output}$ , the  $7 \times 7$ -filter kernel design requires  $7^2 * C^2 = 49C^2$  hence 81% more parameters compared to the  $3 \times (3 \times 3)$  version with just  $3 * (3^2 * C^2) = 27C^2$  parameters [66]. Besides application of sparse connectivity and parameter sharing, further discarding the 3 fully connected layers of the VGG16 architecture reduces the encoder's parameters from 134M to 14.7M. Adding decoder stack and classifier with almost the same amount of approximately 14.8M parameters, totals in 29.5M trainable network parameters. With a size of 4 bytes per parameter and an input of dimensions  $360 \times 480 \times 3$ , Equation 2.28 calculates to 1067 MB of required memory for the complete network.

This massive reduction of trainable parameters and memory requirements enables end-to-end training with SGD on current hardware in reasonable time. Stage-wise successive training of layers is not necessary and SegNet's architecture allows to jointly train encoder stack, corresponding decoder stack and the classifier whereby all weights in the network can be optimized with the same update technique. Making use of the VGG16 architecture for the encoder stack, brings along the convenience to choose from a broad range of previously trained networks for initializing the SegNet network parameters. An additional benefit is that the trained network requires only forward evaluation during inference.

As reported in [28] and [19] SegNet performs competitively, achieving high scores for road scene understanding on large, well known datasets such as CamVid [67] and KITTI [29] while significantly increasing efficiency in terms of computational time and memory requirements during training as well as inference. SegNet shows off it's strength especially in preserving even small segmentation classes and in accurate boundary delineation. This is important for robust segmentation of classes like pedestrians, bicyclists or road signs, which are majorly important for securely navigating autonomous vehicles in public environments, but represented by far less pixels in usual road scenes compared to classes such as sky, road or buildings.

The SegNet project offers a software implementation of SegNet, based on a modified version of Caffe. *CAFFE (Convolutional Architecture for Fast Feature Embedding)* [68] is an open source deep learning framework, developed by Berkely Vision and Learning Center (BVLC), now renamed to Berkeley AI Research (BAIR), and community contributors.

Apparently SegNet seems to be an appropriate architecture to further inspect deep learning technologies in autonomous driving applications.

### 3.3 ROS Caffe Package - ROS CNN Support

The *ROS Caffe package* is a support package without any own nodes. It provides the ROS workspace and thus the *awid\_SemSeg* package with a deep CNN framework. Consequently, it is a prerequisite for applying SegNet in the *awid\_SemSeg* package's *awid\_SegSemImg* node. While the ROS Caffe package by itself targets to meet goal 1, the combination with the *awid\_SemSegImg* node represents the approach to satisfy goal 2. Due to flexibility considerations the Caffe integration is not realized as dependency in the `/include` subfolder of the *awid\_SemSeg* package but as separate ROS package. This allows to switch between different Caffe versions or even use other frameworks with the *awid\_SemSeg* package. Furthermore it makes it possible for other ROS packages to use Caffe free of any dependency to the *awid\_SemSeg* package.

As mentioned in Section 3.1, the SegNet project provides a Caffe-based implementation [27]. However, support for current hardware and software necessitates to employ a modified version [69] called *Caffe SegNet cuDNN5*. Usage of the Caffe-based SegNet implementation adds further requirements, or rather dependencies, to the system. The range of components introduces a set of dependencies, that make it a challenging task to set up a working system. Not precluding that other configurations might work as well, Table 3.1 represents the one used throughout this thesis.

Characteristic	Value
Operating System	Ubuntu 16.04.3 LTS (Xenial)
Linux Kernel	4.4.0-130-generic(x86_64)
Compiler Version	GCC 5.4.0
Graphics Card (GPU)	Nvidia GeForce GTX 1050
GPU Memory (VRAM)	4GB GDDR5
Graphics Card Architecture	Pascal
Graphics Card Driver Version	384.130
Nvidia CUDA Version	8.0.61
Nvidia cuDNN Version	5.1.10
Robot Operating System (ROS) Version	Kinetic Kame
Python Version	2.7.12
Caffe Version	Caffe SegNet cuDNN5

Table 3.1: Target System Environment Characteristics

Caffe uses Nvidia CUDA for GPU computation and supports GPU-based acceleration computational kernel libraries such as Nvidia cuDNN. *CUDA (Compute Unified Device Architecture)* [70] is a parallel computing platform and application programming interface (API) that allows to use compatible GPUs for general purpose processing. GPUs based on Nvidia's pascal architecture require CUDA SDK version 8 or higher which is one of the reasons for using Caffe SegNet cuDNN5. *cuDNN (CUDA Deep Neural Network Library)* is a GPU-accelerated library of primitives for deep neural networks that provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers [71].

The following paragraph broadly outlines the integration of Caffe SegNet cuDNN5 into the MIG software framework. Details can be found in the executable install script `awid_caffe_segnet_into_catkin_installer.py` which has been composed in the course of this thesis. An overview of the script's principal actions is depicted in Figure 3.4.



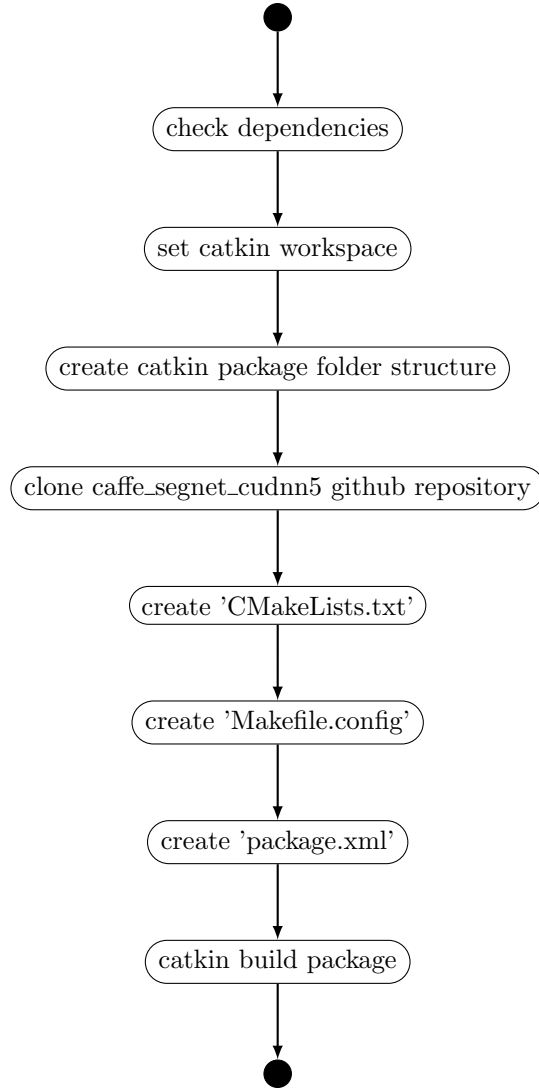


Figure 3.4: ROS CAFFE Package Installer - UML Activity Diagram

The MIG software framework is a modified and extended version of the AutoNOMOS Software framework. Its catkin workspace is organized according to REP-128 [43], as described in Section 2.3. The install script creates the ROS CAFFE package within this workspace's `/src` folder and names it `caffe_segnet_cudnn5`. To accomplish this, it clones the CAFFE SegNet cuDNN5 framework from the github repository [69] into the `caffe_segnet_cudnn5` package's folder structure and creates the package files `CMakeLists.txt`, `Makefile.config` and `package.xml` in compliance with [43], [26], [44] and [72]. Once the `caffe_segnet_cudnn5` package has been created, it is catkin build and afterwards available for usage within ROS.

With the ROS CAFFE package, alias `caffe_segnet_cudnn5`, being integrated into the autonomous driving system MIG as artificial neural network framework suitable for semantic segmentation, the requirements formulated in goal 1 are successfully accomplished.



### 3.4 ROS Node *awid\_SemSegImg* - Semantic Segmentation of 2D Images

The purpose of the *awid\_SemSeg* ROS package's *awid\_SemSegImg* node is to provide deep neural network-based pixelwise semantic segmentation of 2D camera images in the MIG autonomous vehicle platform as demanded in goal 2. Broadly, it has to subscribe to a ROS image message topic published by the SatCam driver, process received images with the SegNet architecture, classify and finally ROS publish accordingly SegNet-labeled images. This core functionality of the *awid\_SemSegImg* node is shown in Figure 3.5, illustrated in form of a UML activity diagram. The *awid\_SemSegImg* node also has a command line argument parser which is described in subsequent paragraphs within this section. For reasons of visual clarity Figure 3.5 does not include the UML conditional nodes related to command line options.

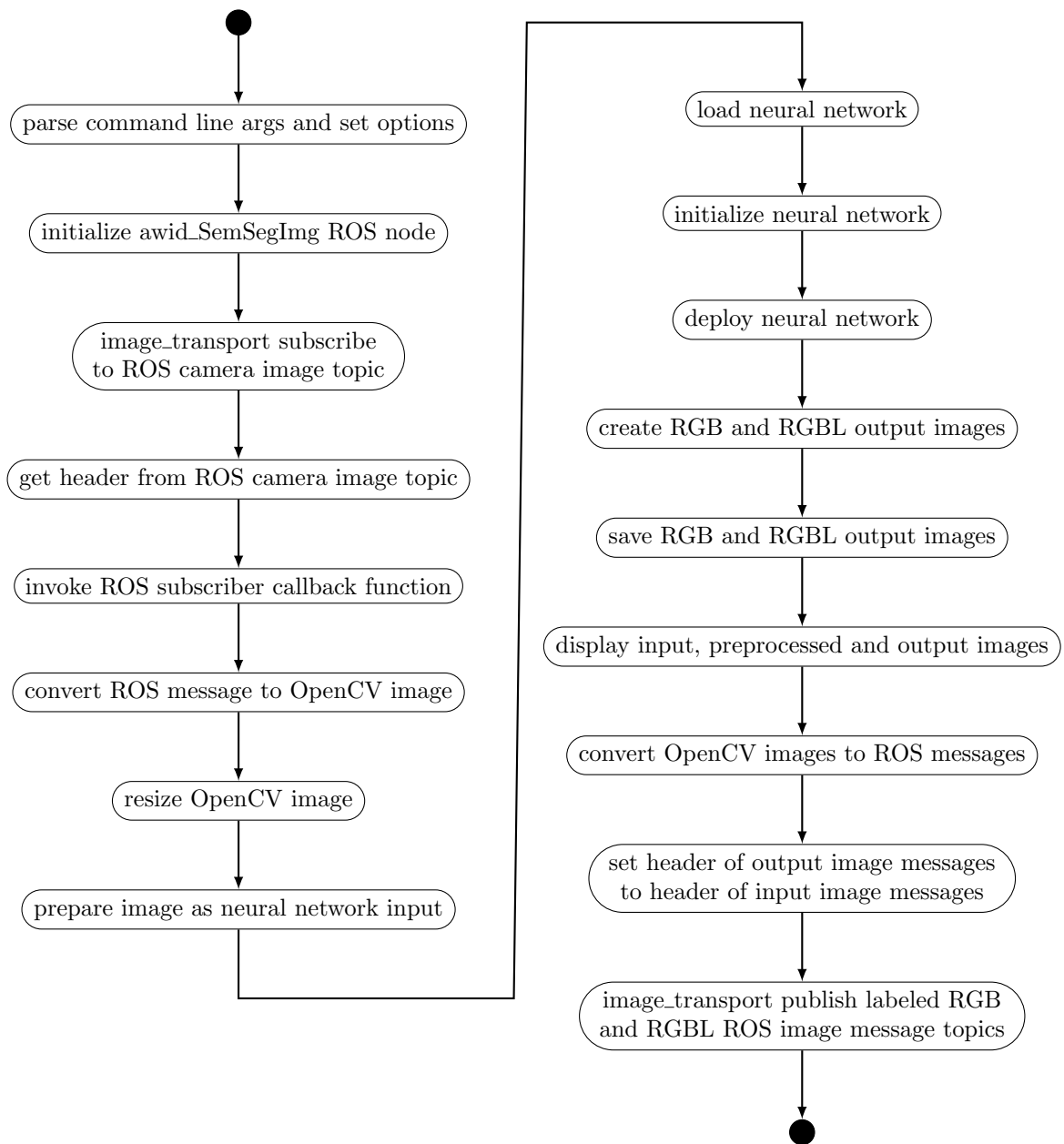


Figure 3.5: SemSegImg Node Implementation - UML Activity Diagram

Some of the technologies and functions used to implement the *awid\_SemSegImg* node like *image\_transport* subscriber and publisher, which are described in the following paragraphs, are only available in C++ and not in Python. Consequently the *awid\_SemSegImg* node is implemented in C++ and the *roscpp* client library is used. This enables to easily switch from node to *nodelet* once a stable version has been implemented. A *nodelet* is a special type of ROS node, designed to run multiple algorithms or nodes in a single process. This allows to use zero copy pointer passing thus avoiding costs of intraprocess copies which is especially useful in high throughput data flows such as camera image feeds. However, *nodelets* also require an implementation in C++ making the use of the *roscpp* client library inevitable. The C++ source code implementation of the *awid\_SemSegImg* node mainly consists of a command line argument parser and the class *NeuralNetApplication*. Figure 3.6 depicts the UML class diagram of the node's *NeuralNetApplication* class. The command line argument parser *CmdArgParser(int argc, char \*\*argv)* is not a member function of the *NeuralNetApplication* class. It parses the command line arguments, configures the switches accordingly and passes them to the class as struct *cmlOptions*.

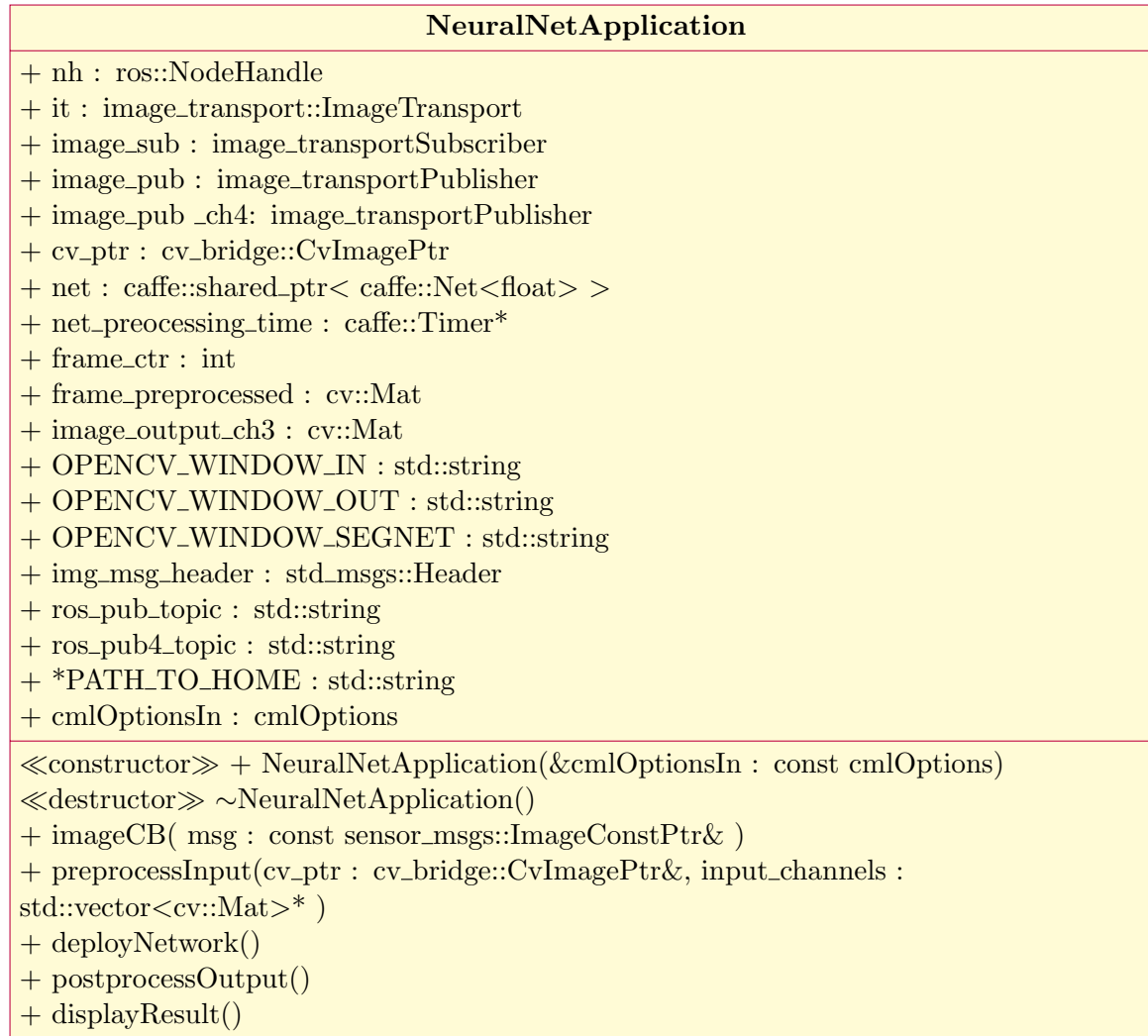


Figure 3.6: SemSegImg Node Implementation - UML Class Diagram

The executable script *awid\_semsegimg\_ros\_pkg\_installer.py* installs and catkin builds the *awid\_SemSeg* package including the *awid\_SemSegImg* node. The deployed version of ROS and its corresponding catkin build system use Python version 2.7 as target platform [73]. Consequently the installer is a Python 2.7 script to be able to use the Catkin

Python API. Except for cloning the `awid_SemSeg` git repository [20] and not creating a `Makefile.config` it performs in principle the same actions as the script described in Section 3.3 and depicted in Figure 3.4 called `awid_caffe_segnet_into_catkin_installer.py`. A renewed description and a separate diagram is therefore omitted. A folder and file structure as described in Section 2.3 is created. The node's source code file `awid_semsegimg.cpp` is located in the `awid_SemSeg` package's `/src` subfolder. The `/include` subfolder contains a `sources.txt` identifying original file download sources and several SegNet files: the learned model parameters `segnet_weights_driving_webdemo.caffemodel`, different `.prototxt` network models and the `camvid12.png` label colors. Files with extension `.caffemodel` contain learned models in form of serialized binary protocol buffer files. The files with extension `.prototxt` are plaintext Google protocol buffer schema files defining neural network models. The `segnet_model_driving_webdemo.prototxt` model from the SegNet project is used as default model by the `awid_SemSegImg` node. The file's input section is modified, so that an input layer is replacing the deprecated input fields, which however does not change the model itself. The other provided model files have adapted spatial scale parameters to enable processing of KITTI or SatCam images in their native spatial resolution. As explained in Sections 2.6 and 3.2 the SegNet network architecture can in principle process input images of arbitrary resolution. The `.png` file includes the color patches used to create images in which the pixel's color represents the assigned class label.

The `awid_SemSegImg` node is designed to allow a very flexible and broader application than just SegNet processing of SatCam images. It is a general mechanism to subscribe to an arbitrary ROS image topic, load, initialize and deploy any neural network architecture and ROS publish or save the resulting output images. This requires only minimal or no code modifications at all. For this purpose the node includes a command line argument parser. An overview and a short description of possible command line argument options is given in Table 3.2. A detailed description is provided in the source code file's Doxygen documentation.

Option	Short Description
<code>-usage</code>	Prints usage information to the console.
<code>-ros_sub_topic</code>	ROS image transport topic to subscribe to.
<code>-model</code>	CAFFE-compliant neural network <code>.prototxt</code> model file.
<code>-weights</code>	Neural network initialization <code>.caffemodel</code> weights file.
<code>-label</code>	Neural network output class label colors <code>.png</code> file.
<code>-cpu</code>	Use CPU for neural network processing instead of GPU.
<code>-no_resize</code>	Omit spatial rescaling of input to neural network.
<code>-save_output</code>	Save neural network's output to storage.
<code>-display</code>	Display input, preprocessed and output images.
<code>-kitti</code>	To be used with the <code>awid_KITTL_SemSeg</code> dataset.

Table 3.2: SemSegImg Node - Command Line Argument Options Overview

In case the `awid_SemSegImg` node is started without any arguments, the default values listed in Table 3.3 are used.

The `awid_SemSegImg` node subscribes to a single ROS `image_transport` topic and per default to the rectified images of the SatCam front camera. For subscription and publishing, the node uses `image_transport` instead of usual ROS subscribers and publishers. Besides specialized transport strategies for images like automatic JPEG compression, `image_transport` classes provide further resource-saving features such as conditional publisher behavior with which objects are only published if an active subscriber for the particular

Default Option	Short Description
<code>-ros_sub_topic</code>	<code>/sensors/broadreachcam_front/image_rect</code>
<code>-model</code>	<code>segnet_model_driving_webdemo.prototxt</code>
<code>-weights</code>	<code>segnet_weights_driving_webdemo.caffemodel</code>
<code>-label</code>	<code>camvid12.png</code>
<code>-cpu</code>	Disabled. Neural network is processed on GPU.
<code>-no_resize</code>	Disabled. Input is spatially resized according to dimensions given in network's <code>.prototxt</code> model file.
<code>-save_output</code>	Disabled. Neural network's output is not saved to storage.
<code>-display</code>	Disabled. Input, preprocessed and output images are not displayed.
<code>-kitti</code>	Disabled. Not set for usage with the <code>awid_KITTL_SemSeg</code> dataset.

Table 3.3: SemSegImg Node Command - Line Argument Options Defaults

topic is present. The `-ros_sub_topic` switch enables to subscribe and thus process other camera image transports, also from other cameras, like for example the raw images from the rear SatCam camera. It is also possible to process image transports from multiple cameras by starting multiple instances of the `awid_SemSegImg` node. This necessitates usage of the node's `-ros_sub_topic` switch in combination with ROS remapping arguments for node name and publisher topic name.

The header information of each subscribed image is read and reused for the corresponding published output image so that the timestamp of the published image represents the time of acquisition. This is a requirement for matching the processed and published output images to other sensor modalities like point clouds.

The SatCam ROS camera driver publishes unrectified JPEG-compressed 8bit unsigned 3-channel RGB images with a spatial resolution of 800 x 1280 pixels. According SatCam camera driver parameters are set for rectified images and usage of the driver's faster turbojpeg decompression [35]. The `awid_SemSegImg` node's input images from the SatCams are received as ROS messages and converted into OpenCV images for resizing and further preprocessing prior to neural network deployment. *OpenCV (Open Source Computer Vision Library)* [74] is an open source software library of optimized algorithms for computer vision and machine learning. Spatial resizing of the input images can be omitted by using the `-no_resize` option at node launch. Otherwise the input images are resized to the spatial dimensions provided in the input layer of the `.prototxt` network model file.

During preprocessing the input image is transformed from 8-bit unsigned integer OpenCV Mat with structure  $[Y[Z * X]]$  to 32-bit float CAFFE Blob  $[B[Z[Y[X]]]]$  representation, where  $X$ ,  $Y$  and  $Z$  are the digital image dimensions presented in Section 2.4 or the respective network layer dimensions described in Section 2.6 and  $B$  is the batch size explained in Section 2.6. A *CAFFE Blob* is a wrapper over the data being stored and communicated in CAFFE networks. To exemplify this, a native resolution SatCam image in form of a OpenCV Mat data structure with `<uint8> [800[3*1280]]` becomes a `<float32> [1[3[800[1280]]]]` CAFFE Blob. To avoid memory copies during this process, the network's input layer is wrapped in separate OpenCV Mat objects, one for each channel. Unlike many similar network's input images, the images prepared for the SegNet Webdemo model are not normalized before being fed into the network.

Prior to deploying the neural network, the Caffe mode is explicitly set to GPU, unless the `awid_SemSegImg` node's `-cpu` option has been set. Especially in combination with higher resolution images and the `-no_resize` switch, processing on the CPU might be the only possible option. With Equation 2.28 and 4 bytes per parameter, the memory requirements for a single SatCam image in native spatial resolution calculate to 6.3 GB, which exceeds the available memory of most commonly available GPUs.

The output generated by the neural network is postprocessed to create `png` images as visual representations. In case of the SegNet model used, the output Caffe Blob `<float32> [1[1[Y[X]]]]` is converted into a 3-channel RGB OpenCV Mat `<uint8> [Y[3*X]]` and a 4-channel RGB + class label OpenCV Mat `<uint8> [Y[4*X]]`. The loaded label file is used to apply an RGB *LUT* (*Lookup Table*) to color each pixel according to its class label, thus creating a *SegNetRGB* image. With the default label file `camvid12.png` the coloring follows the SegNet class color scheme defined in Table A.2. Adding the class label integer value as fourth channel creates a *SegNetRGBL* image. Thus, the *L* channel of an image carries machine readable classification information whereby the pixel value directly represents the class ID. The `awid_SemSegImg` node's `-label` option allows to apply other color schemes to generate differently colored output images. Using the node's `-save_output` option creates the directory `<home>/Temp/awid_SemSegImg_Output/` and saves the output images as `awid_SemSegImg_Output_RGB-<frameNumber>.png` files in it. With the addition `rgbl` the option `-save_output rgbl` causes to also save the RGBL files, named `awid_SemSegImg_Output_RGBL-<frameNumber>.png`, in the same directory.

Regardless of the optional saving, the RGB as well as RGBL output images are converted to ROS messages, timestamped with the time of acquisition and finally `image_transport` published under the topics `<subscribedToTopic>_awid_semsegimg_rgb` and `<subscribedToTopic>_awid_semsegimg_rgbl`.

When the `-display` option is used the subscribed to image as well as the corresponding preprocessed and resulting output RGB images are displayed on the screen, each in a separate window.

The option `-kitti` sets the `awid_SemSegImg` node in a mode dedicated for usage with the `awid_KITTLSemSeg` dataset and the `awid_kitti_20180_07_15_drive_0001_synced.bag` file. Both, the dataset and the bag file, are described in Chapter 4. The `-kitti` option enables the `awid_SemSegImg` node to SegNet-classify the color camera images of the `awid_KITTLSemSeg` dataset in native spatial resolution and thus creating SegNet-colored output images for evaluation of the node's as well as the underlying neural network's technology. According qualitative and quantitative assessment methods and results are presented in the respective subsections of Chapter 4. To avoid manual command line-based remapping of the `awid_SemSegImg` node's subscriber, it is set to match the `awid_KITTLSemSeg` dataset ROS bag's camera image publisher. Because resizing of the input images is omitted, the appropriately modified SegNet model `awid_segnet_model_driving_webdemo_KITTI.prototxt` is used. For the initial weights and class label colors the defaults listed in Table 3.3 are deployed. Since the purpose of the `-kitti` mode is to generate images for evaluation rather than low latency realtime performance, the buffer size of the node's callback function is increased to the amount of images contained in the dataset, ensuring that all images are processed and no images are skipped. The SegNet-colored KITTI RGBL images are saved as described in the paragraph about the `-save_output` option above, then renamed to match the KITTI naming conventions and added to the `awid_KITTLSemSeg` dataset as depicted in Figure 4.1 and described in Section 4.2.2. Publishing occurs as just mentioned above.

### 3.5 ROS Node *awid\_SemSegPnt* - Label Transfer To 3D Point Clouds

The purpose of the *awid\_SemSeg* package's *awid\_SemSegPnt* node is the cross-modal transfer of semantic labels from pixels in 2D images to points of LiDAR-generated 3D point clouds in the MIG autonomous vehicle platform, thus satisfying goal 3. Therefore the node has to subscribe to and synchronize ROS messages containing camera info and image messages published by the TE SatCam camera's driver, semantically segmented images published by the *awid\_SemSegImg* node and point clouds published by the Velodyne driver. The general approach is to perform a perspective projection of the 3D points to the camera's image plane to transfer the semantic label of the synchronized semantically segmented image's pixels to the spatially corresponding projected 3D points. The accordingly semantically labeled point cloud is finally ROS published. The UML activity diagram in Figure 3.7 shows a high-level representation of the *awid\_SemSegPnt* node's core functionality. The term *SemSeg* is used synonymously for semantically segmented. To focus on the node's main purpose, optional features like display of images and point clouds are omitted from this diagram.

To install the node, its source code file is copied to the *awid\_SemSeg* package's `/src` sub-folder and the `CMakeLists.txt` and `package.xml` are modified accordingly to catkin build the package including the new *awid\_SemSegPnt* node. All these steps are performed by the script `awid_semsegpnt_ros_node_installer.py`. Analogous to the *awid\_SemSegImg* node described in Section 3.4, the *awid\_SemSegPnt* node is implemented in C++ and has a similar structure, consisting of a command line argument parser and a class named *CrossModalLabelTransfer*. Command line arguments i.e. options are used to configure the node and an according struct is passed to the class. Available command line options are listed in Table 3.4, the defaults applied when starting the node without any arguments in Table 3.5.

Option	Short Description
<code>-usage</code>	Prints usage information to the console.
<code>-ros_sub_cam_info_topic</code>	ROS camera info topic to subscribe to.
<code>-ros_sub_cam_topic</code>	ROS camera image transport topic to subscribe to.
<code>-ros_sub_sem_topic</code>	ROS semantic image transport topic to subscribe to.
<code>-ros_sub_pnt_topic</code>	ROS point cloud topic to subscribe to.
<code>-extensive</code>	Alternative processing, using less library functions.
<code>-save_output</code>	Save node's output point clouds to storage.
<code>-display</code>	Display camera input and semantic input images.
<code>-overlay</code>	Display camera input and semantic input images with projected point cloud overlays.
<code>-pcl_visualizer</code>	Display input point cloud and class-label-colored output point cloud in PCL Visualizer.
<code>-kitti</code>	To be used with the <i>awid_KITTILSemSeg</i> dataset.

Table 3.4: SemSegPnt Node - Command Line Argument Options Overview

The `-usage` option and the `-ros_sub_XXX_topic` options provide analogous functionality to the one described for the *awid\_SemSegImg* node in Section 3.4. Besides subscribing to image messages with image-transport subscribers, the *awid\_SemSegPnt* node also subscribes to *CameraInfo* messages that contain meta information for the camera and *PointCloud2* messages that include the point cloud data from the LiDAR scanner.



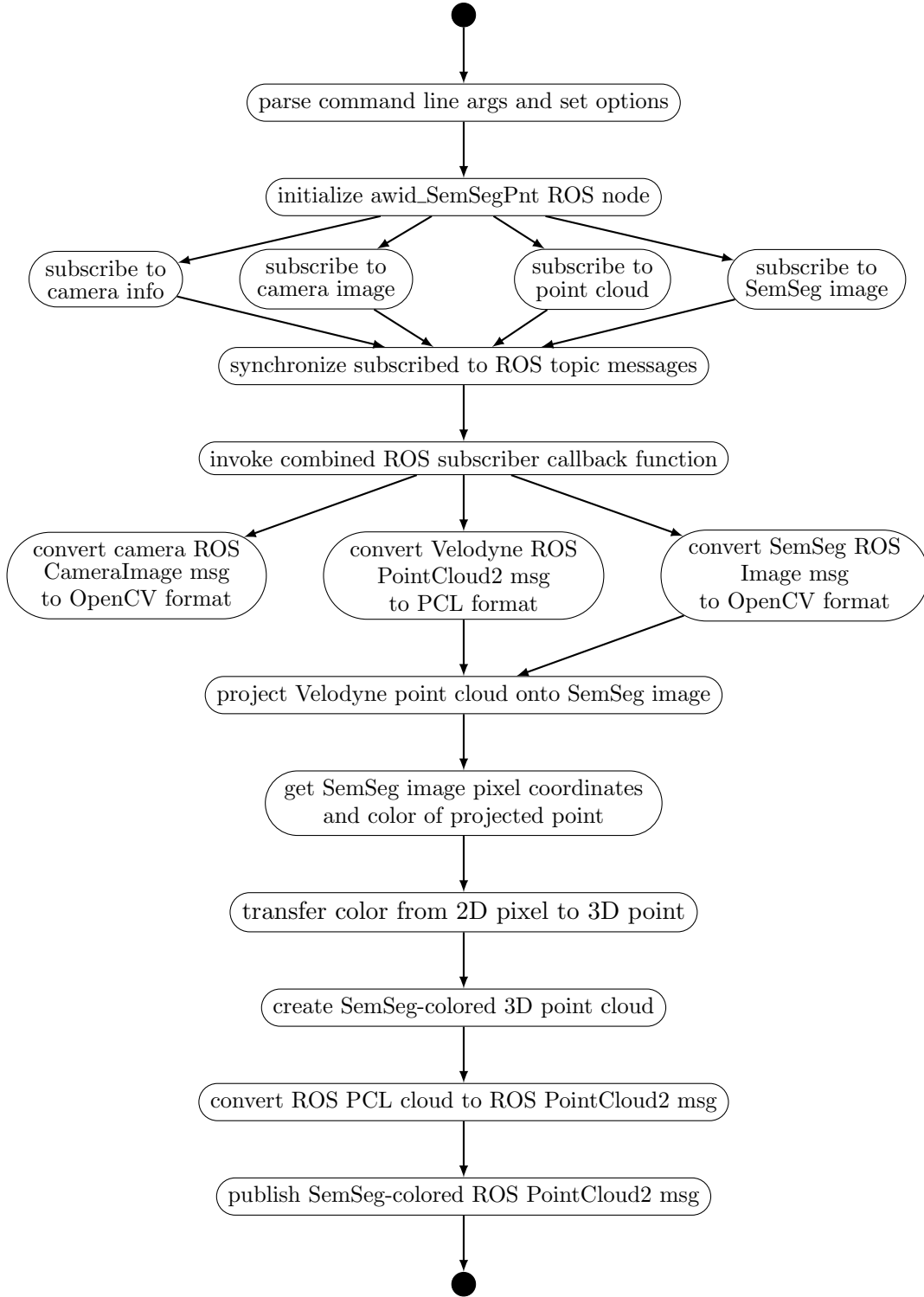


Figure 3.7: SemSegPnt Node Implementation - UML Activity Diagram

The node employs its own internal data structure for working with points. The according point type *PointXYZIABCUVL* is defined as struct and extends the commonly used ROS or PCL point type *PointXYZI*. It combines all related task-relevant point information in one common data structure. The *XYZ* fields contain the XYZ values of the current processing step. The *IABC* values preserve the *XYZ* values of the originally captured and unprocessed 3D point in Velodyne space. Keeping these values allows to avoid backprojection for transferring the class assigned to a pixel back to the corresponding point. The

Default Option	Short Description
-ros_sub_cam_info_topic	'/sensors/broadreachcam_front/camera_info'
-ros_sub_cam_topic	'/sensors/broadreachcam_front/image_rect'
-ros_sub_sem_topic	'/sensors/broadreachcam_front/image_rect_awid_semsegimg_rgb'
-ros_sub_pnt_topic	'/sensors/velodyne_points'
-extensive	Disabled. Processing is based on library functions.
-save_output	Disabled. Output is not saved to storage.
-display	Disabled. Camera and semantic input images are not displayed.
-overlay	Disabled. Point cloud overlays in images are not displayed.
-pcl_visualizer	Disabled. Point clouds are not shown in PCL Visualizer.
-kitti	Disabled. Not set for usage with the awid_KITTI_SemSeg dataset.

Table 3.5: SemSegPnt Node - Command Line Argument Options Defaults

fields  $U$  and  $V$  contain the image space-related X and Y pixel coordinates of the point projected to the camera's image plane. And the last field  $L$  holds the class label assigned to the point.

Throughout the node, smart pointers are the preferred mechanism to pass on point cloud and image data, hence avoiding copying and saving resources.

In order to follow the approach to transfer the semantic labels from the pixels of the images published by the *awid\_SemSegImg* node to the points published by the Velodyne LiDAR scanner's driver it is inevitable to temporally synchronize and spatially align these source feeds. Synchronization is based on the ROS *Transfer Frame tf* package and alignment on the ROS *image\_geometry* package.

*Temporal synchronization* is based on the ROS message's timestamps and accomplished by implementation of a ROS message filter deploying an approximate time policy to find the closest temporal match between camera-originated image transport messages and LiDAR-based point cloud messages. As listed in Table 2.2 the MIG's SatCam cameras output images with a rate of 30 Hz whereas Table 2.3 reports the Velodyne LiDAR scanner to output point clouds at a rate of 10 HZ. The actual publishing rates differ slightly. The SatCams and the Velodyne scanner operate independently of each other, not being clocked or synchronized in any manner. Furthermore the *awid\_SemSegImg* node requires some time to generate and publish a semantically segmented output image as described in Section 4.2.3. In addition the scenery captured by the individual SatCam camera and the Velodyne scanner overlap only partially and only from different perspectives. Consequently the approximate time synchronization policy's queue size has to be set appropriately large or otherwise the ROS message filter's synchronizer won't output any messages to invoke the joint callback function. The joint callback function, which is being invoked only when all of the multiple subscriber messages of different type are available and synchronized within a specific time window, is realized by using *boost::bind* to create a function object from a shared pointer to the callback function and its expected parameters, one for each subscriber message. This function object is then registered to the shared pointer that is used to create the ROS message filter's synchronizer according to the chosen synchronization policy.



*Spatial alignment*, or multi-modality registration, of camera-originated images and corresponding point clouds acquired by a LiDAR scanner is realized through perspective projection of the LiDAR point cloud into the camera's image plane. Required camera parameters are obtained from the subscribed to camera info message to parametrize the used pinhole camera model and all other camera-related variables such as the image plane's spatial dimensions, the camera's intrinsic or projection matrices. In case the semantically segmented input image's spatial dimensions are different to those received through the camera info message, the semantic image is resized accordingly to allow projection of the points into the camera image plane. A simple pixel replication or elimination is used for rescaling the semantic image to avoid the introduction of undefined colors and thus classes caused by interpolating pixels.

Depending on the usage of the *-extensive* option the subsequent processing differs. The *-extensive* option switches the node's processing from using mostly pre-defined library functions provided by ROS, OpenCV and PCL to processing performed with own functions, offering more flexibility and a higher degree of intervention in the processing pipeline. ROS and OpenCV have already been introduced in previous sections and *PCL* [75] stands for *Point Cloud Library* which is a C++ open-source library of algorithms for 2D and 3D point cloud and geometry processing. The option *-extensive* is disabled by default and the node uses the pre-defined functions as detailed in the following paragraph.

The point cloud is clipped to the points visible by the camera with a PCL frustum culling filter which is parametrized with the camera model and set to the camera's vertical and horizontal field of view as well as the LiDAR scanner's near and far clipping ranges. Depending on the car configuration deployed, the values from Tables 2.2 and 2.3 for the MIG or the values from [29], given in Tables B.1 and B.2, for the KITTI car are used. For correct alignment it is however important to differentiate between the rectified and the non-corrected raw camera image. In case of the MIG the resulting horizontal field of view of the rectified image reduces to approximately 126° and the vertical field of view to 78°. The clipped point cloud is transformed from the Velodyne's coordinate frame to the camera's coordinate frame. To do this, a ROS tf listener waits for both coordinate frames to be available, to then lookup the transform and utilize it for a `ros_pcl` transform of the point cloud. Subsequently a ROS camera model's function is used to project the clipped and transformed 3D points to 2D pixels. The class of the semantic image's pixel at the location calculated for each projected point is now transferred to the corresponding point. A point cloud consisting of `PointXYZRGB`-type points that are colored according to their class assignment is created, then converted to a ROS message and finally ROS published.

In case the *-extensive* option is activated, a tf listener is used to wait for the Velodyne's and the camera's coordinate frames to become available but the transform is assembled by calculating and combining the rotation matrix and the translation vector. The resulting matrix is then used to transform the point cloud from Velodyne space into camera space. By performing a perspective divide the points are projected to the camera plane and thus transformed from camera space to screen space. Then the point cloud is clipped to the points visible to the camera by applying logically combined conditional statements based on the image dimensions received from the camera info message's parameters. The points are then transformed to the camera's NDC (Natural Device Coordinate) space by normalizing the point's coordinates. Afterwards the point's coordinates are rounded off to the nearest following integers and the direction of the y-axis is inverted which represents the transformation from NDC space to raster i.e. image space. The just described processing is an adapted and extended implementation of [76] which describes the mathematics of

computing the 2D coordinates of a 3D point. The subsequent processing from label transfer to ROS publishing of the colored point cloud is independent of the *-extensive* option and identical to the one explained above. Figure 3.8 depicts the UML activity diagram of the just described callback function's processing with enabled *-extensive* option.

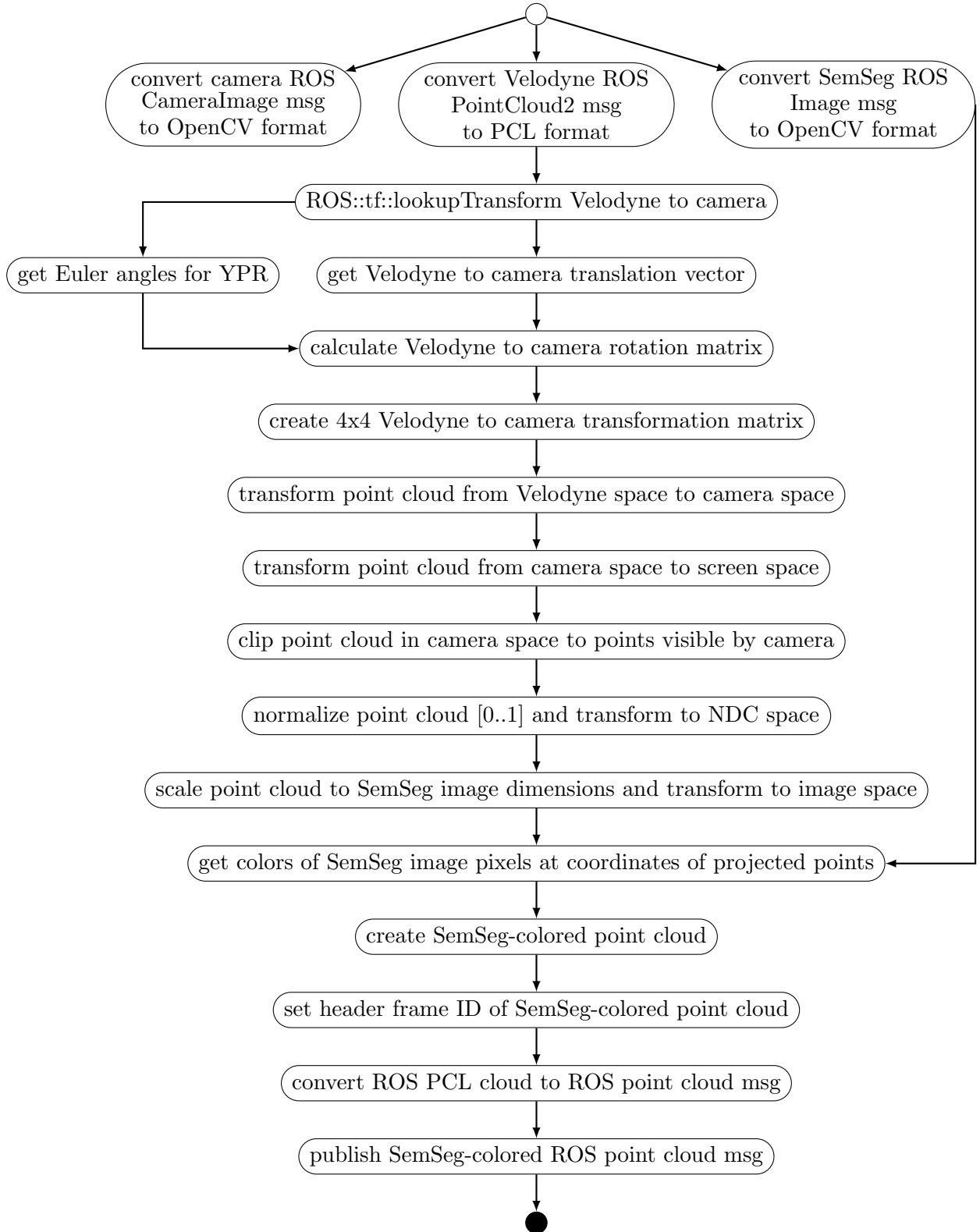


Figure 3.8: `SemSegPnt` Node - Callback Function Extensive Option Implementation - UML Activity Diagram

Using the node's `-save_output` option creates a directory to save the node's labeled and accordingly colored RGB and RGBL output point clouds. The files that are named `awid_SemSegImg_Output_RGB_<frameNumber>.pcd` and the corresponding files including the label ID `awid_SemSegImg_Output_RGBL_<frameNumber>.pcd` are saved inside the directory `<home>/Temp/awid_SemSegPnt_Output/`.

Apart from using RViz, the node also provides several options to visualize images and point clouds. When the `-display` option is used the subscribed to camera image and the semantically segmented image are displayed on the screen, each in a separate window. The ROS timestamp information is superimposed at the bottom of the displayed images whereby the input camera image's timestamp is shown in white type and the semantic image's one in green.

With the option `-overlay` the camera and the semantically segmented images are displayed with projected point cloud overlays. The point cloud message's timestamp information is shown in red type.

Option `-pcl_visualizer` opens a PCL Visualizer with two viewports whereby the left side shows the input point cloud and the right side the label-transfer-colored semantically segmented output point cloud. Both viewports also show the coordinate frames of the camera and the Velodyne scanner. In contrast to RViz the PCL Visualizer even displays the node's RGBL output point cloud by just considering the known XYZRGB fields of the `PointXYZRGBL`-type points and ignoring the L field. Rviz stays blank and does not visualize any portion of the RGBL point cloud. Thus for visualization in RViz, solely the node's RGB output cloud can be selected.

The `-kitti` option sets the `awid_SemSegImg` node in a mode dedicated for usage with the `awid_KITTLemSeg` dataset and an especially for this purpose created ROS bag file named `awid_kitti_20180_07_15_drive_0001_synced_GeneratedTimestamps.bag`. The dataset is described in Section 4.1 and details on the bag file are given in Section 4.3.1. The main purpose of this option is to generate sets of point clouds that are added to the `awid_KITTLemSeg` dataset as `pcd` files for evaluating the segmentation accuracy of the `awid_SemSegPnt` node.

# Chapter 4

## Evaluation

This chapter discusses the evaluation of the implemented ROS nodes described in the previous chapters. Besides determining resource consumption such as execution time and actual memory usage, the focus is mainly on the assessment of the segmentation accuracy of 2D images and that of the cross-modal transfer to 3D point clouds. The assessment is meant to be an indication of usefulness of the implementation and the used technologies with respect to the goals of this thesis rather than an in-depth analysis.

### 4.1 Dataset For Semantic Segmentation Evaluation

Quantitative evaluation of the `awid_SemSeg` ROS node's segmentation accuracy and the cross-modal transfer of class labels from 2D images to 3D point clouds requires an appropriate *dataset* providing the following:

- calibrated, rectified, synchronized and timestamped data
- continuously temporal progressing timestamp information
- sensor setup and calibration information
- captured diverse real-world traffic situations containing a significant number of objects of interest
- ground truth data with a SegNet-compatible classification scheme
- RGB camera images
- XYZ LiDAR point clouds
- pixelwise labeled semantic segmentation ground truth 2D images
- pointwise labeled ground truth point clouds with 3D object information

Such a dataset or other appropriately annotated ground truth data is not available for the MIG autonomous vehicle platform. Therefore it is inevitably necessary to seek alternatives to enable a quantitative evaluation of the implemented system. Typical road scene datasets like CamVid [67], Cityscapes [30] or KITTI [29] do make available pixelwise semantically labeled RGB camera images, but not in combination with corresponding annotated point clouds.

However, the KITTI Vision Benchmark Suite [77], [29] appears to provide all components to assemble a dataset satisfying the requirements formulated above. Furthermore the

KITTI autonomous vehicle capturing platform, that is outlined in Appendix B, is similar to the MIG, which is preferable, to achieve higher comparability of the evaluation’s results. The KITTI Semantic Segmentation Benchmark Dataset [78] is comprised of RGB camera images and corresponding semantic segmentation ground truth images. The ground truth images are for instance available as single channel 8-bit `png` images in which each pixel’s value represents directly the semantic label ID. Besides that, ground truth images are offered as semantic RGB images in which pixels are colored according to their class label. Throughout the KITTI dataset the label IDs, names, instance classes and label colors of the Cityscapes dataset are used. Corresponding definitions are given in the Cityscapes `labels.py` [79] file.

Ground truth information for 3D objects is provided for some of the KITTI Vision Benchmark Suite’s raw data in form of *tracklet labels* that are stored in `tracklet_labels.xml` files. All tracklet labels for the complete image sequence i.e. drive are stored in one single file. A tracklet provides per image information on height, width, length of an object’s 3D bounding box as well as its 3D location and orientation, all with respect to the Velodyne coordinate frame.

Selecting only those images of the KITTI Semantic Segmentation Dataset that are also available in the KITTI Vision Benchmark Suite’s raw data [77] allows to add the missing corresponding Velodyne point clouds, timestamps, odometry as well as 3D object data. Consequently only a subset of the 200 training images can be used. Identification of matching images in the raw data is done manually. Only images with available tracklets, i.e. 3D object data, are included resulting in a total of 120 images. Of those, 77 images belong to the KITTI category *City*, 10 to *Residential* and 33 to *Road*.

Several Python scripts `awid_timestamps-*.py` are written to extract and assemble the timestamp data to serve as temporal reference for each image, point cloud and IMU file. Tracklet label information is extracted for some objects of the selected frames. Sensor calibration information is provided in the calibration files: `calib_cam_to_cam.txt`, `calib_imu_to_velo.txt`, `calib_cam_to_velo.txt`, `calib_velo_to_cam.txt`. The image files are renamed and arranged in the folder structure depicted in Figure 4.1, based on the description given in the KITTI Raw Data Development Kit available for download under the KITTI Vision Benchmark Suite website’s raw data section.

Mimicking the just described folder and file structure, allows to use the tool *kitti2bag* [80] to create ROS `bag` files from the newly assembled dataset files. Two bag files are created and added to the dataset:

The file `awid_kitti_2018_07_15_drive_0001_synced.bag` contains the originally captured timestamps and is used with the `awid_SemSegImg` node in conjunction with the `-kitti` option to generate `png` files for evaluation that are saved in the datasets sub folders `/SegNetRGB` and `/SegNetRGBL`.

The file `awid_kitti_2018_07_15_drive_0001_syncedGeneratedTimestamps.bag` is a version with the same essence data but modified timestamp information. In case continuously progressing timestamps are required, like the synchronized fusion of data from cameras and the LiDAR scanner dealt with in Section 3.5, the `timestamp.txt` files are replaced with versions containing generated timestamps. For generating timestamp information, the timestamps of the dataset’s first set of captured sensor readings i.e. images, point clouds, etc. serve as references. A continuous temporal proceeding with a step size of 10 ms is used to increase the camera’s timestamp between sequential images of the dataset. To properly simulate real world conditions, the timestamp’s temporal offsets between the different sensors are preserved, thus providing the system under challenge with realistic data, even though the timestamps are modified. The file

`awid_kitti_20180_07_15_drive_0001_synced_GeneratedTimestamps.bag` in conjunction with the `-kitti` and the `-save_output` options allow to generate and save a set of point cloud files as basis for qualitative and quantitative evaluation of the `awid_SemSegPnt` node's cross-modal label transfer. The `awid_KITTI_SemSeg` dataset is extended and the `pcd` files are stored in the dataset's subfolders `/SegNetPntRGB` and `/SegNetPntRGBL`.

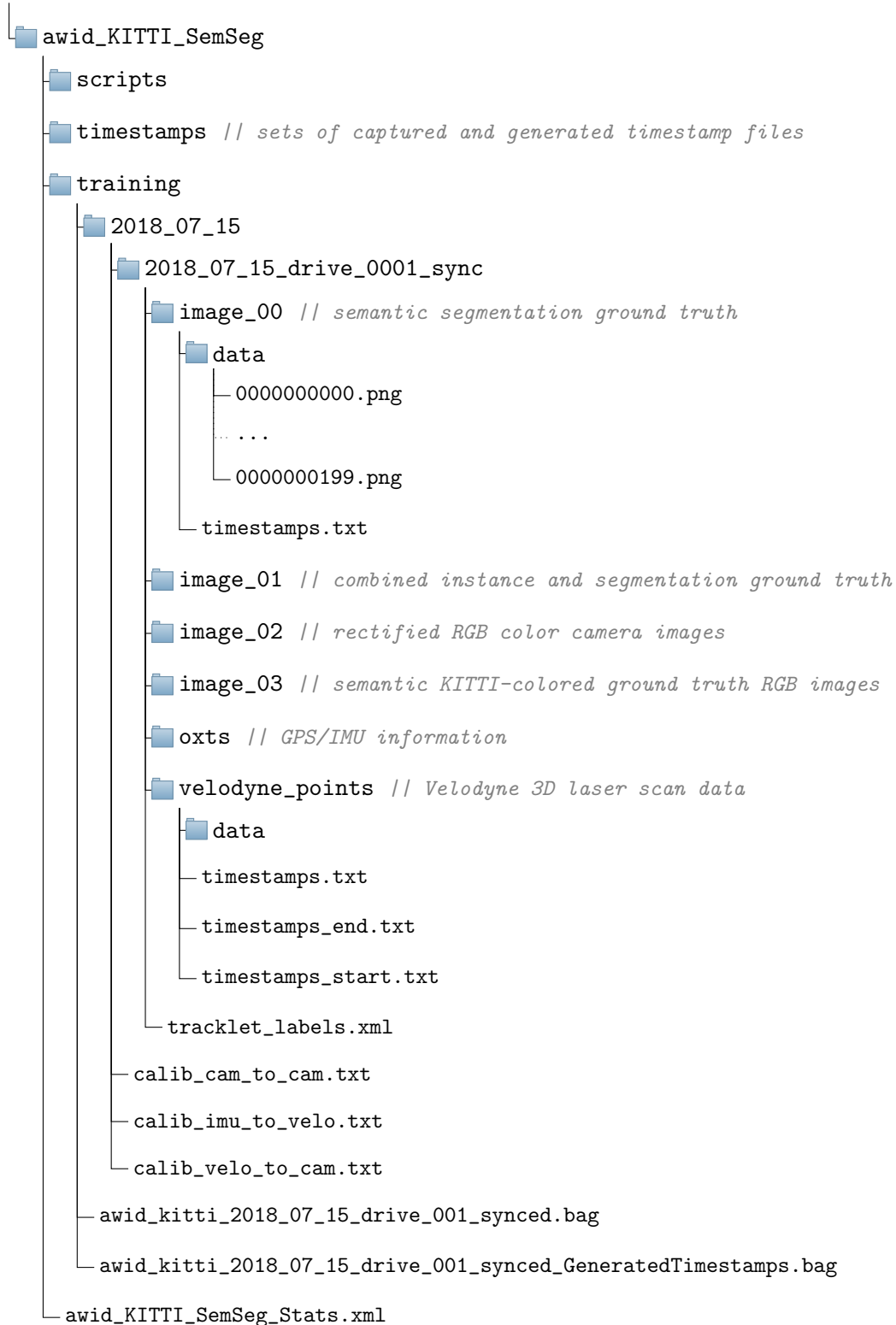


Figure 4.1: `awid_KITTI_SemSeg` Dataset - Folder Structure And Content

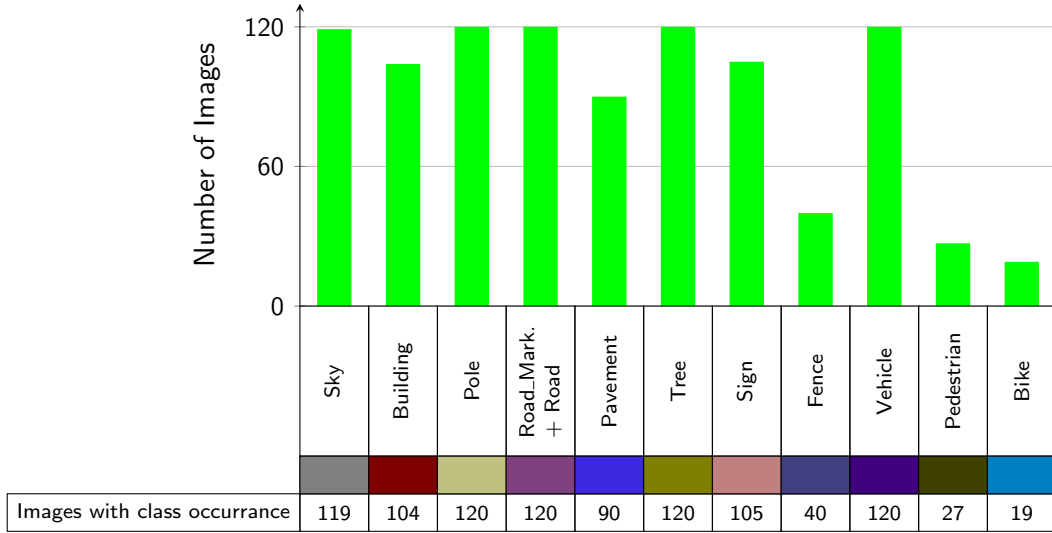


Figure 4.2: awid\_KITTL\_SemSeg Dataset - SegNet Class Occurrence Summary

The resulting *awid\_KITTL\_SemSeg dataset* is a rearranged subset of the KITTI dataset. The number of images of the dataset that contain pixels of a SegNet class is given in Figure 4.2. The chart in Figure 4.3 depicts the distribution of accumulated pixels associated with each SegNet class over all images of the complete dataset. Pixels of ignored classes are neglected in the calculation of the percentage values. Therefore, the pixels of not ignored classes sum up to 100%.

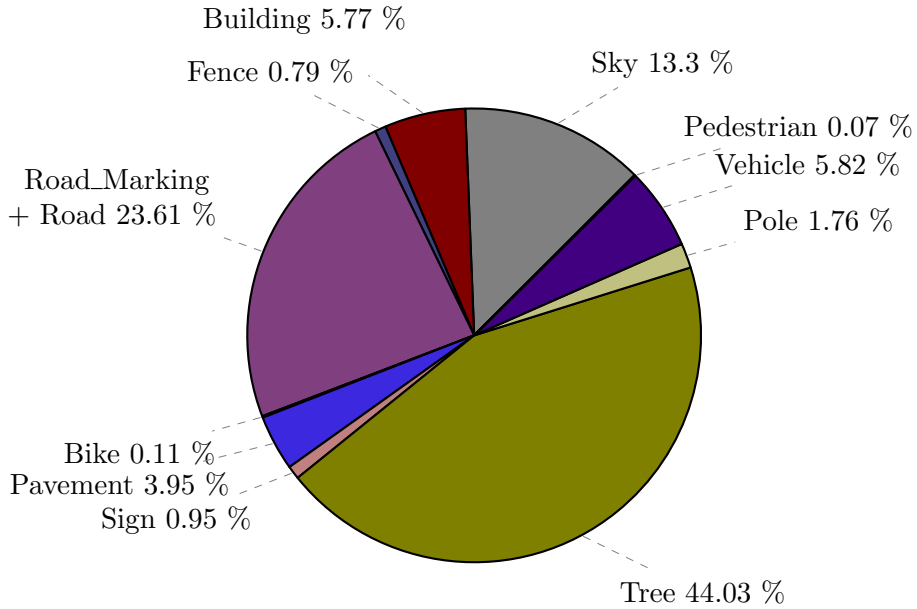


Figure 4.3: awid\_KITTL\_SemSeg Dataset - SegNet Class Pixel Distribution

Performing a further in-depth analysis of the dataset itself would be outside the focus of this thesis. However, the dataset appears to adequately satisfy the requirements formulated in the beginning of this section to assess the segmentation performance of the implemented *awid\_SemSeg* nodes and their underlying technologies for providing an indication of their usefulness with regard to the goals of this thesis. Consequently, the *awid\_KITTL\_SemSeg* dataset is used and further extend in the course of this thesis.

## 4.2 SemSegImg Node Evaluation

Besides determining execution time and actual memory consumption, the assessment of semantic segmentation of 2D images is primarily based on comparing the class of each pixel of the predicted image to the class of the corresponding pixel of the associated ground truth image.

A prerequisite for such a comparison are matching classes of predicted and ground truth images. Therefore, it is necessary to map the 35 KITTI classes, that are identical to the Cityscapes classes, to the 12 SegNet classes. It is unavoidable to map multiple classes to one class. Cityscapes does not provide a *Road\_Marking* class. Consequently the SegNet classes *Road\_Marking* and *Road* are merged into class *Road*. Among the 35 KITTI classes are 16 classes that are per definition [79] ignored during evaluation. Overall just 3.14% of the pixels of all images in the complete awid\_KITTI\_SemSeg dataset are assigned to these ignored classes. The mapping of Cityscapes classes to SegNet classes is listed in Table 4.1, whereby only the relevant 19 Cityscapes classes included during evaluation are shown. KITTI semantic ground truth images are recolored to match the SegNet color scheme with the Python script `awid_KITTI_SemSeg_KITTI2SegNetColor.py` and saved in the subfolder `/semantic_gt_SegNet` of the awid\_KITTI\_SemSeg dataset. The recolored images are subsequently referred to as *KITTI2SegNetColored* or *SegNet\_gt\_rgb* images.

KITTI i.e. Cityscapes			SegNet		
Color	Class ID	Class Name	Class ID	Class Name	Color
	23	Sky	0	Sky	
	11	Building	1	Building	
	12	Wall	1	Building	
	17	Pole	2	Pole	
	7	Road	3	Road_Marking	
	7	Road	4	Road	
	8	Sidewalk	5	Pavement	
	21	Vegetation	6	Tree	
	22	Terrain	6	Tree	
	19	Traffic Light	7	Sign	
	20	Traffic Sign	7	Sign	
	13	Fence	8	Fence	
	26	Car	9	Vehicle	
	27	Truck	9	Vehicle	
	28	Bus	9	Vehicle	
	31	Train	9	Vehicle	
	31	Train	9	Vehicle	
	24	Person	10	Pedestrian	
	25	Rider	11	Bike	
	32	Motorcycle	11	Bike	
	33	Bicycle	11	Bike	

Table 4.1: Class Mapping - KITTI i.e. Cityscapes ↔ SegNet



### 4.2.1 Image 2D Quantitative Evaluation Methodology

Apart from measuring execution time and actual memory consumption, *quantitative evaluation* of the awid\_SemSegImg ROS node's 2D image segmentation accuracy is done by calculating the *IoU*, the intersection over union, per class. All measurements are just meant to be an indication of principle usefulness rather than an in-depth analysis.

The execution time of the node's core component i.e. the neural network's inference time is calculated by implementing the *caffe::Timer* class. The system's overall in-to-out execution duration is obtained through the time passed between two consecutive output messages published by the node. This is observed with the ROS command `rostopic hz <topicName>`. The values determined on the system specified in Table 2.4 with some of the options listed in Tables 3.3 and 3.2 are shown in Table 4.2.

Actual memory consumption is reported in Table 4.2 as measured with the command line utility `nvidia-smi` (*Nvidia System Management Interface*) for the GPU and the Ubuntu's `gnome-system-monitor` for the process' computer RAM. Shared, virtual or other RAM usage is not observed. To get an indication of suitability of the implemented nodes this approach is considered sufficient. For a deeper analysis profiling tools like `gprof`, `valgrind`, `callgrind` or such alike are more appropriate. Information on how to deploy these with ROS can be found at [81]. Neither code or system have been optimized for performance.

*Recall*, also known as *True Positive Rate TPR* or *sensitivity*, is a measure for how many of the existing pixels of a class are correctly labeled and thus *true positives TP*. As Equation 4.1 shows, the TRP penalizes *false negatives FN* and therefore pixels belonging to an object of a class which however are not correctly classified as such. *Precision*, stated in Equation 4.2 and also referred to as *Positive Predicted Value PPV* or *specificity*, penalizes *false positives FP* and therefore pixels that are incorrectly assigned to a class even though they belong to an other class.

$$TPR = \frac{TP}{TP + FN} \quad (4.1)$$

$$PPV = \frac{TP}{TP + FP} \quad (4.2)$$

In contrast to recall and precision the IoU penalizes both FPs and FNs. The *intersection over union IoU* is also known as *Jaccard Index* and based on the definition given in the PASCAL VOC Challenge [82]. As Equation 4.3 shows, the IoU is the intersection of the inferred i.e. predicted segmentation and the ground truth, divided by their union. The calculation of the segmentation accuracy is a per image, per class comparison of the class of each pixel of the predicted image to the class of the corresponding ground truth image's pixel. Equation 4.3 can also be expressed as fraction of true positives (TP) and the sum of true positives, false positives (FP) and false negatives (FN) which is formulated in Equation 4.4. The pixels belonging to classes that are ignored during evaluation have to be excluded. Even though the IoU is calculated per class and either per single image or all images of the complete dataset, subscripts for both are omitted in the equations below for better readability. According subscripts are used subsequently if considered helpful to distinguish between IoUs calculated for an individual image and IoUs averaged over all images of the dataset.

$$IoU = \frac{\mathbf{I}_{pred} \cap \mathbf{I}_{gt}}{\mathbf{I}_{pred} \cup \mathbf{I}_{gt}} \quad (4.3)$$

$$\Leftrightarrow \quad seg.acc. = \frac{TP}{TP + FP + FN} \quad (4.4)$$

The SegNetRGBL image's L-channel represents the predicted semantic image. The ground truth is the dataset's corresponding KITTI semantic image. As explained previously, the KITTI semantic ground truth images contain pixels with class assignments of classes that are ignored during evaluation. These pixels have to be excluded from IoU calculations.

Due to the class mapping described above, it is not possible to use the tools provided in the KITTI Vision Benchmark Suite. Hence the Python script `awid_KITTI_SemSeg_IoU.py` has been written to calculate the IoU i.e. the segmentation accuracy for each class of each image in the dataset. The script also creates the report file `awid_KITTI_SemSegStats.xls`. For every image of the `awid_KITTI_SemSeg` dataset this report lists the number of pixels per Cityscapes class and the number of ignored as well as included pixels during evaluation. Besides the IoUs, it also contains the amount of pixels, TPs, FPs as well as FNs per SegNet class, per image. It constitutes the data basis of the `awid_SemSegImg` node's quantitative evaluation and is extended for further analysis purposes. As explained in Section 4.2.2, it is possible to generate TPFNIG images with the script, that visualize correct and incorrect classification as well as areas that are ignored during evaluation. Figure 4.4 shows a simplified UML diagram of the script.

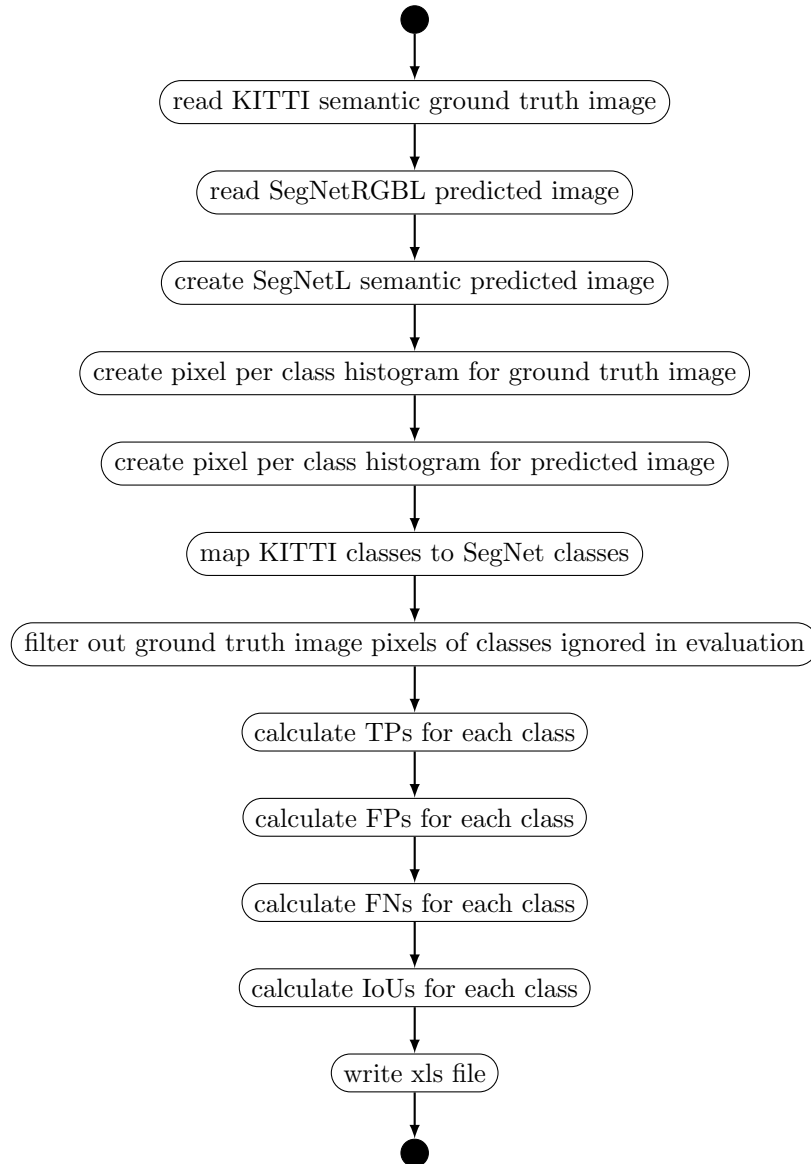


Figure 4.4: IoU Python Script Implementation - UML Activity Diagram

### 4.2.2 Image 2D Qualitative Evaluation Methodology

*Qualitative evaluation* of the `awid_SemSegImg` ROS node's 2D image segmentation is done by visually comparing pixelwise segmented RGB images in which each pixel is colored according to its class affiliation. Either the `KITTI2SegNetColored` images or the *semantic\_rgb* images from subfolder `/image_03` of the `awid_KITTL_SemSeg` dataset described in Section 4.1 serve as visual ground truth. The predicted images are created by processing the dataset's rectified KITTI RGB camera images from folder `/image_02` through the `awid_SemSegImg` node.

Own and other attempts [83] to train the SegNet architecture did not show any superior segmentation accuracy compared to available pretrained models. For that reason the pretrained SegNet Webdemo is used, which consists of the `prototxt` model and the `caffemodel` parameters listed in Table 3.3 and described in Section 3.4.

This model has been trained with the CamVid dataset and further undisclosed images to classify road scenes into the 12 SegNet classes listed in Table A.2. The CamVid RGB images have a spatial resolution of 360 x 480 pixels and the SegNet Webdemo obviously rescales input images to this resolution before processing them through the network. As explained in Sections 2.6 and 3.2, the SegNet network architecture can in principle process input images of arbitrary resolution. Convolution, small-sized kernels, sparse connectivity, small step-sizes make feature extraction spatially invariant. However, the also described constraints have to be considered. As long as pooling, scaling and padding are handled properly, the limiting factor is the available memory size. Applying Equation 2.28 with a parameter size of 4 bytes calculates the SegNet memory requirements for one KITTI image of dimensions 375 x 1242 x 3 to be approximately 2.88 GB, which is within the limits of the deployed system that is specified in Table 2.4. The amount of network parameters stays unchanged. Thus, to avoid artifacts introduced by scaling, the `awid_KITTL_SemSeg` dataset's KITTI RGB camera images will be ingested into SegNet in its native spatial resolution of 375 x 1242 pixels. Furthermore, this allows to evaluate SegNet's superior performance for smaller/thinner classes and higher boundary delineation as stated in [28]. The SegNet Webdemo `prototxt` network model file is modified appropriately as described in Section 3.4. Besides setting the new input dimensions, it is important to round up in case the spatial dimensions of the downscaled image are not dividable by 2 without remainder. The spatial dimensions of the upscaled image in the decoder have to match the ones in the corresponding encoder. Usage of the `-kitti` command line option at launch instructs the `awid_SemSegImg` node to deploy the adapted SegNet Webdemo model file `awid_segnet_model_driving_webdemo_KITTI.prototxt`.

Playback of the ROS bag file `awid_kitti_2018_07_15_drive_0001_synced.bag` contained in the `awid_Kitti_SemSeg` dataset allows to classify the dataset's camera RGB images with the `awid_SemSegImg` node within the MIG's AutoNOMOS software framework, a real world system. However, besides the `awid_SemSegImg` node's `-kitti` option, that is described in Section 3.4, some general rosbag command-line options need to be set. Due to the dataset's noncontinuous timestamp information, it is necessary to use the `-skip-empty` option. In some cases usage of the `-rate` option might be considered.

Since the SegNet architecture is used, the `awid_SemSegImg` node's output images are referred to as SegNet-labeled or SegNet-colored images. The `awid_SemSegImg` node's output is an RGBL image as described in Section 3.4. For visual display, only its RGB portion is of relevance. Thus, the label channel is stripped off to create the SegNet-colored RGB image, being the predicted image. Class affiliation of each pixel in this SegNet-colored image is represented by the pixel's displayed color, which follows the class color scheme defined in Table A.2 in Appendix A. Both types of the SegNet-classified images,

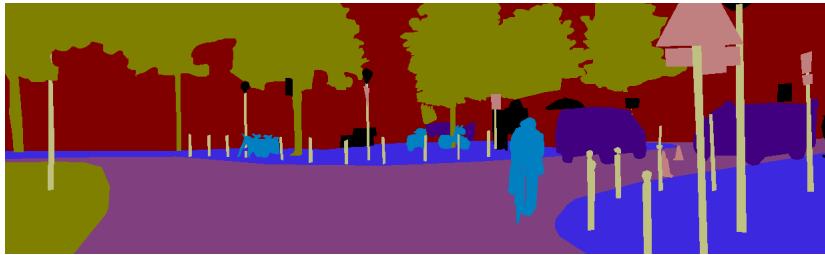
the SegNetRGBL and the SegNetRGB images, are saved in according subfolders of the awid\_KITTL\_SemSeg dataset. The SegNetRGB images serve as the predicted images to be visually compared to the KITTI semantic\_rgb ground truth images. The L-channel of the SegNetRGBL images represents the predicted semantic images to be compared to the KITTI semantic ground truth images that are located in the dataset's /image\_01 subfolder.



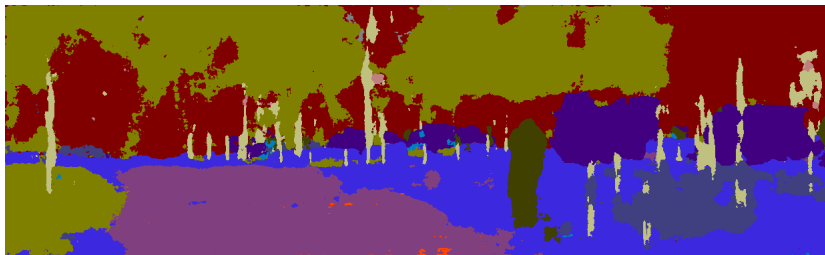
(a) Rectified KITTI RGB Color Camera Image



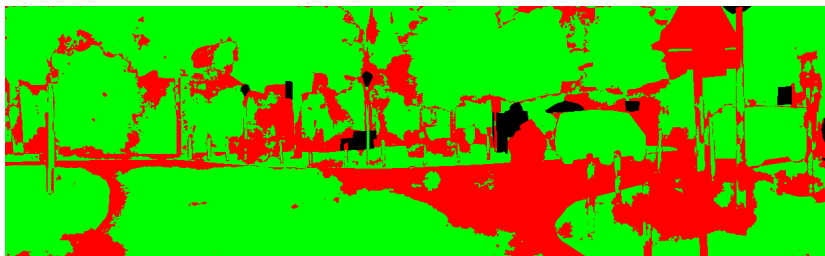
(b) Semantic KITTI-Colored Ground Truth Image



(c) Semantic KITTI2SegNet-Colored Ground Truth Image



(d) SegNet-Colored Predicted Image



(e) TPFPFNIG Image

Figure 4.5: SemSegImg Node - Qualitative Evaluation - Images For Visual Comparison

The Python script `awid_KITTI_SemSeg_IoU.py`, used and presented in the Section 4.2.1 on quantitative evaluation, offers to create *TPFPFNIG* images in the `TPFPFNIGN` subfolder of the `awid_KITTI_SemSeg` dataset. These images contain green colored pixels, in case of *true positives (TP)* i.e. ground truth and prediction are associated with the same class and red pixels wherever the class of ground truth and prediction are different. The latter, combines *false positives (FP)* and *false negatives (FN)*. SegNet class assignments are exclusive. Consequently, in an image showing all classes simultaneously, each pixel with a false positive class assignment for one class is a false negative for another class. Pixels assigned to classes that are *ignored during evaluation* are colored black. Thus *TPFPFNIG* images are a visually intuitively accessible presentation of areas of correct as well as inaccurate pixelwise semantic segmentation.

Figure 4.5 shows a rectified KITTI RGB color camera image (a) and the corresponding KITTI-colored semantic RGB ground truth image (b) as provided in the KITTI Semantic Segmentation Evaluation Dataset. The next image (c) is a KITTI2SegNet-recolored ground truth image in which the KITTI classes have been mapped to the SegNet classes. This recoloring makes visual comparisons to SegNet-colored images more intuitive. The SegNet-colored predicted image in (d) represents the RGB output of the `awid_SemSegImg` node. Finally Figure 4.5 (e) depicts the *TPFPFNIG* image. The combination of those images allows qualitative evaluation based on profound interpretation of visually accessible evidence.

### 4.2.3 SemSegImg Resource Consumption Assessment

Performing an assessment with the previously described methods for 2D images results in the data presented in this section. Table 4.2 summarizes performance indicators concerning execution time and actual memory consumption for some selected configurations. All underlying measurements have been performed on the system specified in Table 2.4 with some of the options listed in Tables 3.3 and 3.2. The values are averaged over 120 frames and reported together with the according standard deviation. The `awid_KITTI_SemSeg` dataset's `awid_kitti_2018_07_15_drive_0001_synced.bag` file is used to provide KITTI RGB images and the `20180202-092743-3_laps-Reinickendorf.bag` file is the source in SatCam image configurations.

As to be expected the `awid_SemSeg` node's network processing i.e. the network inference time is with at least 90% causing the major portion of the execution duration. The remaining time spent on all other processes such as preprocessing, postprocessing and overall node management is relatively small.

Resizing prior to network deployment, which is the only difference between configurations 1 and 2, only adds 8% to the ratio between network inference and node in-to-out duration. The 2.7 times higher pixelcount of the not resized KITTI input images and the according higher number of network calculations cause a 2.5 times longer network inference duration. Comparing the network inference time between configurations 3 and 4 shows that processing the network on the GPU instead of the CPU increases the output rate by more than factor 82.

The network's memory requirements for data reported by CAFFE in the console during network loading match those calculated with Equation 2.28. However, this number excludes the memory requirements for the network parameters, diffs and intermediate blobs. Using Equation 2.27 and a memory size of 4 bytes per parameter, the parameters of the

deployed SegNet model require an additional 118MB of memory. Depending on the implementation of the node itself and the used frameworks, actual memory utilization can differ. Concerning the configurations examined, the actual GPU memory consumption is 16% lower for configuration 1 and 5% higher for the other configurations. Calculated memory consumption is a good indicator if available resources are sufficient. In case of SatCam images it reveals that the resource demands for SegNet processing images in native spatial resolution exceed the available GPU memory on the target system. A cropping of the SatCam images according to available VRAM together with an according modification of the prototxt network model would be a possible solution in case resizing shall be avoided. The RAM consumption in Table 4.2 relates to the memory consumed by the process, the reported resident memory consumption is about 470MB higher.

Performance Indicator	Value	Options	Short Description
Node In-to-Out [ms]	$390.531 \pm 1.512$	-model -ros_sub_topic -no_resize	Configuration 1: KITTI RGB images 375x1242@10fps
Network Inference [ms]	$386.695 \pm 2.100$		
Network Inference [%]	$\geq 98.10$		
GPU VRAM [MB]	2516		
RAM [MB]	462		
Node In-to-Out [ms]	$161.577 \pm 5.661$	-ros_sub_topic	Configuration 2: KITTI RGB images 375x1242@10fps resized to 360x480
Network Inference [ms]	$156.255 \pm 5.445$		
Network Inference [%]	$\geq 90.18$		
GPU VRAM [MB]	1250		
RAM [MB]	456		
Node In-to-Out [ms]	$164.015 \pm 5.465$	default	Configuration 3: SatCam images 800x1280@25fps resized to 360x480
Network Inference [ms]	$155.367 \pm 10.055$		
Network Inference [%]	$\geq 93.34$		
GPU VRAM [MB]	1250		
RAM [MB]	466		
Node In-to-Out [ms]	$13437.058 \pm 74.421$	-cpu	Configuration 4: SatCam images 800x1280@25fps resized to 360x480
Network Inference [ms]	$13368.947 \pm 285.027$		
Network Inference [%]	$\geq 96.84$		
GPU VRAM [MB]	—		
RAM [MB]	$\sim 3900$		

Table 4.2: SemSegImg Node - Execution Duration And Memory Consumption

With network inference being by far the determining factor straining the node’s execution duration, spatial resizing appears to be an adequate method to increase the frequency of output images and to reduce the consumption of system resources. Of course possible segmentation quality thresholds have to be considered. This is however not the focus of this work and a further analysis deploying more adequate tools might be of interest in other theses.

Overall it can be stated, that from a resource consumption perspective the awid\_SemSegImg node’s implementation and the underlying technologies being mainly the CAFFE implementation of SegNet in combination with ROS, CUDA, cuDNN and OpenCV appear to provide a useful, even though further optimizable, solution to the Goals 1 and 2.



#### 4.2.4 SemSegImg Segmentation Assessment

Due to nonexistent ground truth data for the MIG, the quantitative assessment is based on the `awid_KITTLSemSeg` dataset and thus on data captured with the KITTI autonomous vehicle platform which in principle is similar to that of the MIG. Assessment of segmentation accuracy is based on data in `awid_KITTI_SemSeg_Stats.xls`, which has been generated as described in Section 4.2.1. An extended version of the file is saved as `awid_KITTI_SemSeg_Stats_Analysis.xls` in the same folder of the dataset `awid_KITTLSemSeg`. It presents an  $\overline{IoU}_{Weighted\_Dataset}$  of 73.14%. This averaged weighted dataset IoU is calculated by averaging the sum of the per class dataset average IoUs weighted by the number of pixels of the corresponding class. The averaged per class dataset IoUs are listed in the table in Figure 4.6 together with according averaged per class dataset recall (TPR) and precision (PPV) values. Per dataset, indicated by the according subscript, means that the calculation includes all images of the dataset. In contrast a per image calculation is based on a single image. In the following interpretation of the data, scores are categorized in high, medium and low, whereby high describes scores above 0.75, medium scores are between 0.75 and 0.25 and low refers to scores below 0.25. In some cases these categories refer to values presented as percentages.

	Sky	Building	Pole	Road_Mark. + Road	Pavement	Tree	Sign	Fence	Vehicle	Pedestrian	Bike
avg $\overline{TPR}_{Dataset}$ [%]	97.55	45.15	60.77	90.58	55.98	88.12	22.35	31.65	94.23	33.46	13.34
avg $\overline{PPV}_{Dataset}$ [%]	90.49	47.67	46.37	93.40	47.49	95.18	41.58	18.85	81.14	39.85	23.61
avg $\overline{IoU}_{Dataset}$ [%]	88.56	33.44	35.82	85.25	34.72	84.45	15.58	11.96	77.24	23.89	7.70
$\frac{avg \overline{TPR}_{Dataset}}{avg \overline{PPV}_{Dataset}}$	1.08	0.95	1.31	0.97	1.18	0.93	0.54	1.68	1.16	0.84	0.56

Figure 4.6: `awid_KITTLSemSeg` Dataset - SegNet Class Precision Recall IoU Summary

SegNet classes *Sky*, *RoadMarking + Road*, *Tree* and *Vehicle* achieve high  $\overline{IoU}_{Dataset}$  scores above 0.75. Medium scores with an  $\overline{IoU}_{Dataset}$  between 0.75 and 0.25 are reported for SegNet classes *Building*, *Pole* and *Pavement*. Classes *Sign*, *Fence*, *Predestrian* and *Bike* show a segmentation accuracy below 0.25.

Likewise high scores for  $\overline{TPR}$ ,  $\overline{PPV}$  and  $\overline{IoU}$  together, indicate high values for TPs and vice versa. A ratio of  $\overline{TPR}$  and  $\overline{PPV}$  close to 1 shows a balanced proportion of FPs and FNs. The higher the ratio above 1, the higher the amount of FPs compared to FNs. The further the ratio below 1, the higher the amount of FNs compared to FPs. The low IoUs for SegNet classes *Pole* and *Fence* appear to be caused by a relatively high amount of FPs, which means that objects of other classes are incorrectly recognized as objects belonging to these classes. The SegNet classes *Sign* and *Bike* have low  $\overline{IoU}$  scores caused by relatively high amounts of FNs, which shows that these classes are not detected well.

To investigate possible causes for the so far presented  $\overline{IoU}_{Dataset}$  per class performance, the  $\overline{IoU}_{Image}$  scores, based on per image calculations, are examined. Being the foundation of the averaged  $\overline{IoU}_{Dataset}$  scores reported in Figure 4.6, the  $\overline{IoU}_{Image}$  scores present the same classes in the categories of high, medium and low IoU scores. Figure 4.7 shows the number of images with according  $\overline{IoU}_{Image}$  score categories and their distribution as tabularized absolute and percentage values as well as visual representations in a stacked

bar chart. The percentage values are based solely on the number of images in which the respective class occurs. Consequently the sum of images in which pixels of the class occur represents 100%.

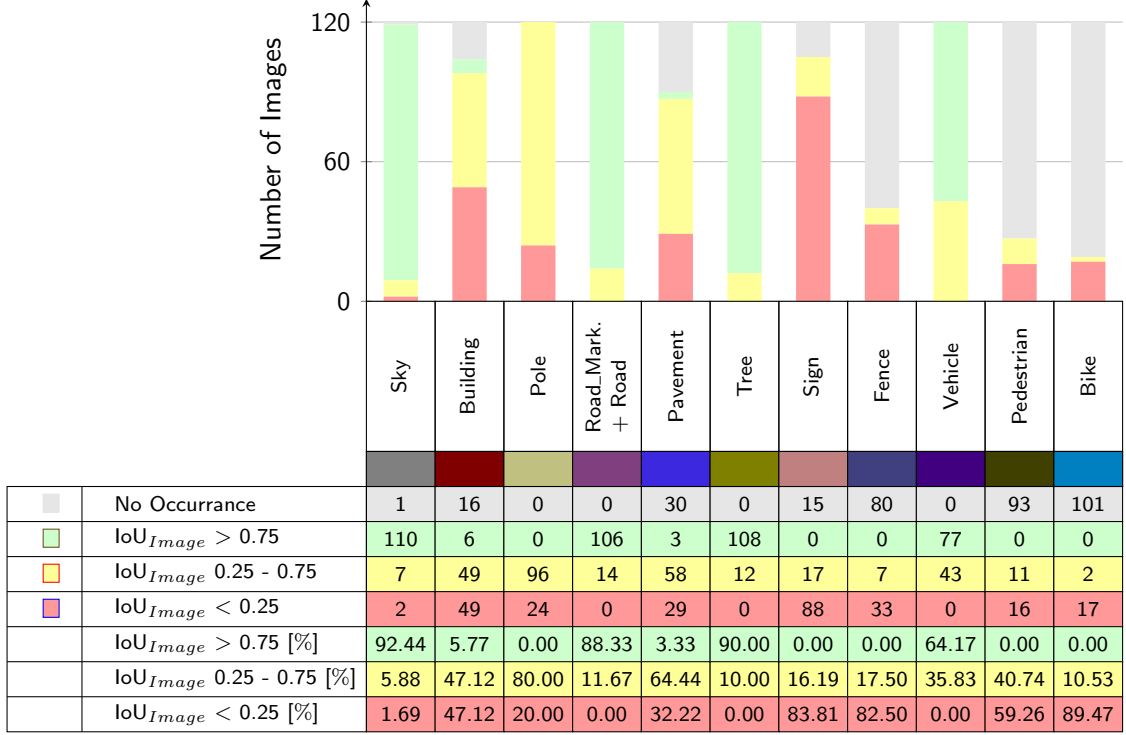


Figure 4.7: awid\_KITTLSemSeg Dataset - SegNet Class  $\text{IoU}_{Image}$  Score Summary

The SegNet classes in which the majority of images have  $\overline{\text{IoU}}_{Image}$  scores above 0.75 are *Sky*, *RoadMarking + Road*, *Tree* and *Vehicle*. Classes with the majority of images with  $\overline{\text{IoU}}_{Image}$  scores between 0.75 and 0.25 are *Pole* and *Pavement*. The class *Building* has exactly as many images with scores between 0.75 and 0.25 as below 0.25. Since the number of images with an IoU above 0.25 is higher than the number those below, the class *Building* is assigned to the category between 0.75 and 0.25. *Sign*, *Fence*, *Pedestrian* and *Bike* belong to the category of classes in which the majority of images have per image IoU scores below 0.25.

The classes *Pedestrian* and *Fence* exist in Cityscapes as well as SegNet and are directly mapped one-to-one which excludes any side effects due to mapping. *Bike* as well as *Sign* are SegNet classes merging multiple Cityscapes classes which is a potential source of error. As listed in Table 4.1 the Cityscapes class *Traffic Light* is mapped to the SegNet class *Sign* within this thesis. The SegNet classes provided by the SegNet project do not include the class *Traffic Lights* even though it exists in the CamVid dataset. Therefore it is presumed that merging traffic lights into the class *Sign* is a good match.

A visual examination of the dataset’s SegNetRGB images (subfolder /SegNetRGB), the TPFPFNIG images (subfolder /TPFPFNIGn) and a comparison to the semantic SegNet-recolored KITTI ground truth RGB images (subfolder /image\_03) and the KITTI color camera RGB images (subfolder /image\_02) helps to further investigate the segmentation accuracy’s assessment results. Figures 4.5 and 4.8 to 4.10 are sets of images for some exemplary frames that provide visual references. Overall the deployed SegNet model detects objects of the different classes and partitions images into coherent semantically meaningful parts. Detection of objects, clustering corresponding pixels into continuous regions and



delineation works better for objects covering larger image areas and apparently best for the SegNet classes *Vehicle*, *Sky*, *Tree*, *Road*, *Road Marking* and *Building*. Difficulties especially with delineation are apparent for classes *Pavement* and *Pole* even though the images were processed in their native spatial resolution. Classes *Fence*, *Pedestrian* and *Bike* show issues in detection as well as proper delineation. Visual inspection generally confirms the results from the data-based analysis above and allows further interpretation of those.

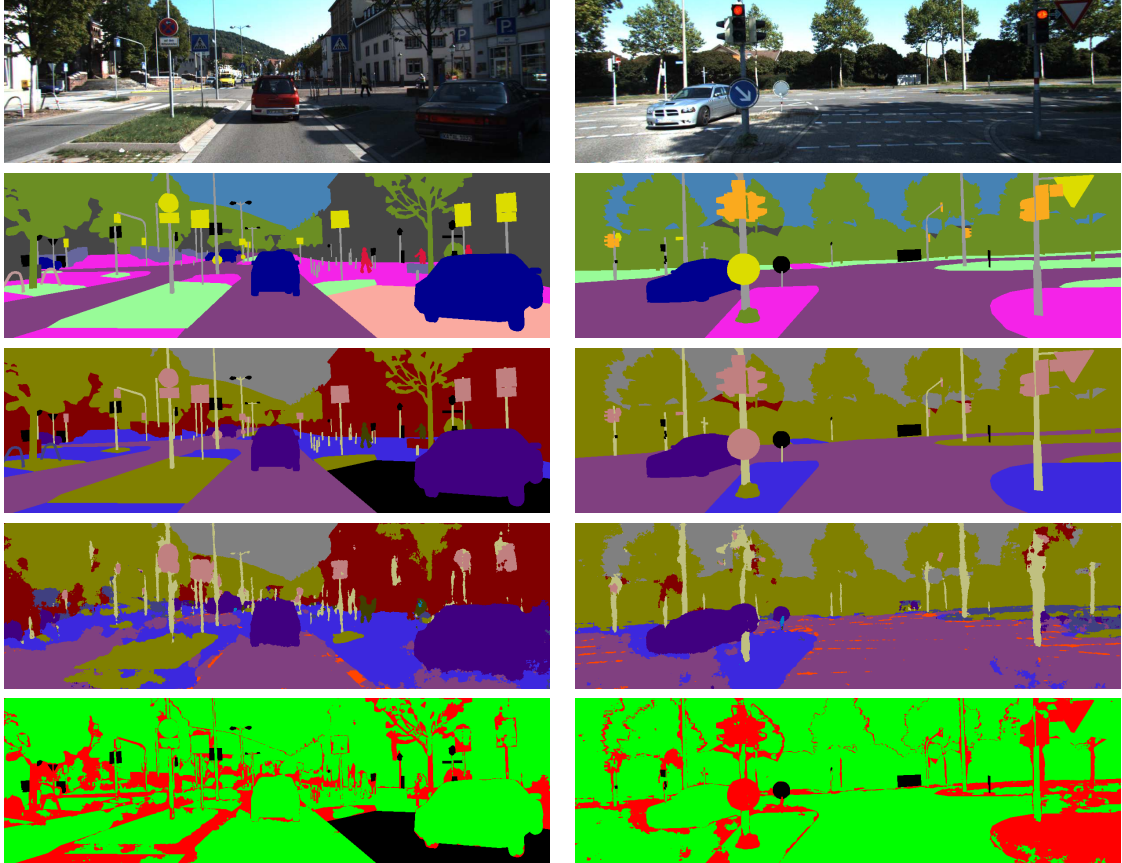


Figure 4.8: SegNetImg Node - Evaluation Image Set - awid\_KITTTLSemSeg Frames 105 (left) And 7 (right) - Highest  $IoU_{Image}$  And Highest FN In SegNet Class *Sign*

As visible in the images of frame 7, presented in Figure 4.8, traffic lights show clusters of correctly labeled pixels but a rather great portion of each traffic light is recognized as pole, contributing to the high FP count examined for the class *Pole*. Thinner structures of vertical extension tend to be generally assigned to class *Pole*. The German traffic sign 'Leitplatte' with ID 626 (StVO, Vz-Kat [84]) is a red and white striped delineator and occurs in 46 of the 103 frames of the dataset containing traffic signs, often with multiple instances. Apparently SegNet has difficulties to correctly label this particular sign. When it is mounted to a pole and especially in cases with rather high distance to the object, like in frame 32 that is displayed in Figure 4.9, the sign is falsely labeled to be a pole, again adding FPs to class *Pole*. In frame 81 shown in Figure 4.9, the sign is often assigned to the class *Pedestrian*. The traffic sign 'Leitplatte' is commonly used as delineator at road construction sites, thus in ground-level areas with pixelwise rather noisy, non-flat regions which SegNet appears to generally categorizes to be *Pavement*. This is particularly noticeable when driving on cobblestone roads or on leaves covering the road and SegNet assigns the label *Pavement* instead of *Road*. The latter is noticeable in

Figure 4.5, showing images of frame 1. This set of images also reveals that the bicyclist is labeled as *Pedestrian* which is admittedly difficult to differentiate from a single image. Nevertheless, it contributes to the low IoU scoring of both classes, which furthermore have a relatively low number of pixel appearances in the dataset.

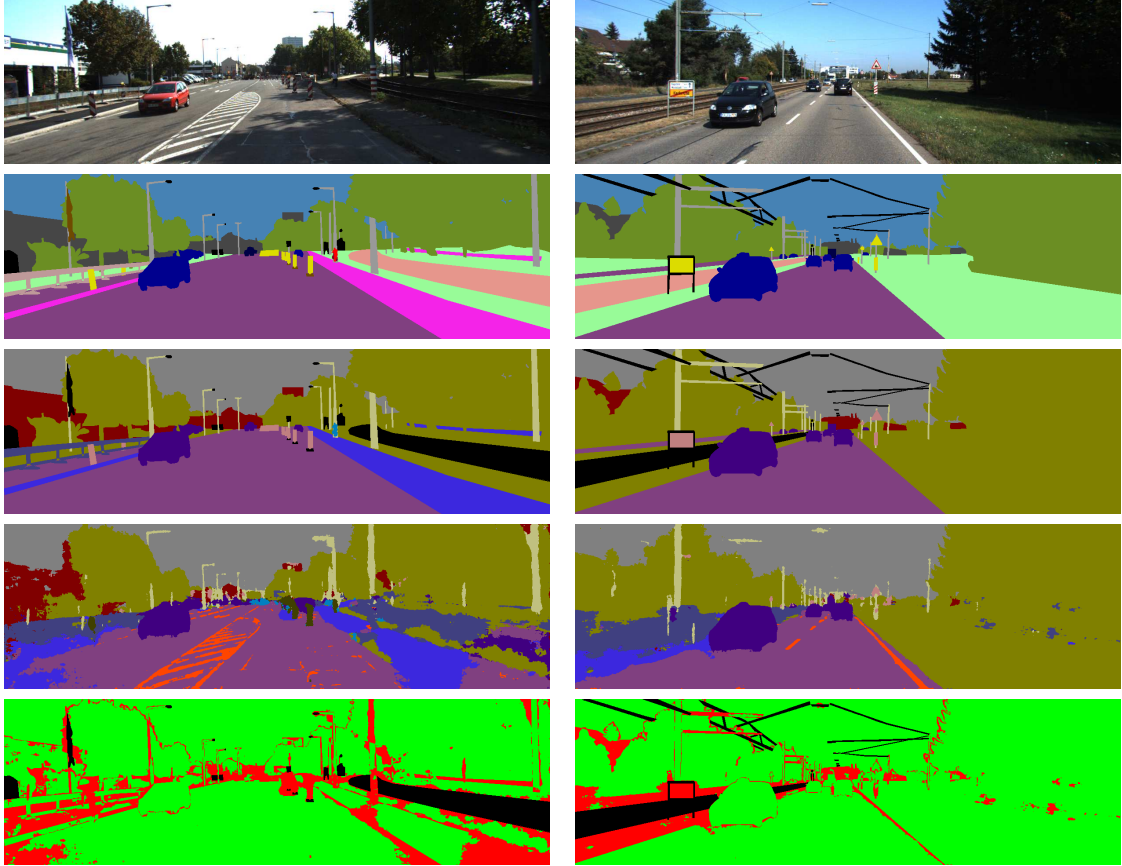


Figure 4.9: SemSegImg Node - Evaluation Image Set - awid\_KITTLSemSeg  
Frames 81 (left) And 32 (right)

Rail tracks, occurring in frame 81 in Figure 4.9 or frame 92 shown in Figure 4.10, are recognized as *Pavement*. Even though the pixels of rail tracks are omitted in the calculations of the quantitative assessment, the rather deficient delineation causes a high amount of FPs in neighboring areas. With 42 out of the 120 frames of the dataset containing rail tracks, this adds a substantial amount of FPs to the class *Pavement*. The images of frame 115 depicted in Figure 4.10 show a false labeling due to mapping the Cityscapes to SegNet classes. The wall on the left is assigned to class *Wall* in the KITTI ground truth image which is mapped to SegNet class *Building*. In the predicted SegNet images, the wall is labeled as *Fence* which again adds FPs to class *Fence* as well as FNs to *Building*. In accordance with the data reported in the quantitative assessment, detection and segmentation of objects of classes *Vehicle*, *Road* and *Road Marking* seems to be a strength of SegNet.

The category assignment described above correlates in general with the distribution of pixels per class in the dataset which is depicted in Figure 4.3. Classes with many pixel occurrences appear to achieve high IoU ratings, whereas classes with lower amounts of pixel occurrences get low IoUs. For classes with lower pixel occurrences the probability of misclassification is higher and at the same time it has a greater negative impact on the

resulting IoU score. The awid\_KITTL\_SemSeg dataset reflects real world driving scenarios but is obviously missing some balance towards content representing classes with lower IoU scores. Image sequences of the category 'Person' of the KITTI Vision Benchmark Suite's raw data might be helpful to balance the dataset. Those however are missing labeled ground truth imagery.

Furthermore, it seems that the training of the network model occurred with a similar unbalance in pixel occurrences. Regarding the CamVid dataset's class distribution this is obvious from the published information: sky (18.04%), building (20.79%), road (25.98%), tree (10.76%), sign (0.17%), pedestrian (0.56%), bicyclist (0.30%). Unfortunately it is unknown which other images were also used to train the deployed SegNet Webdemo model. Training the network with a more balanced dataset should improve evaluation results, especially for the classes with low IoU scores.

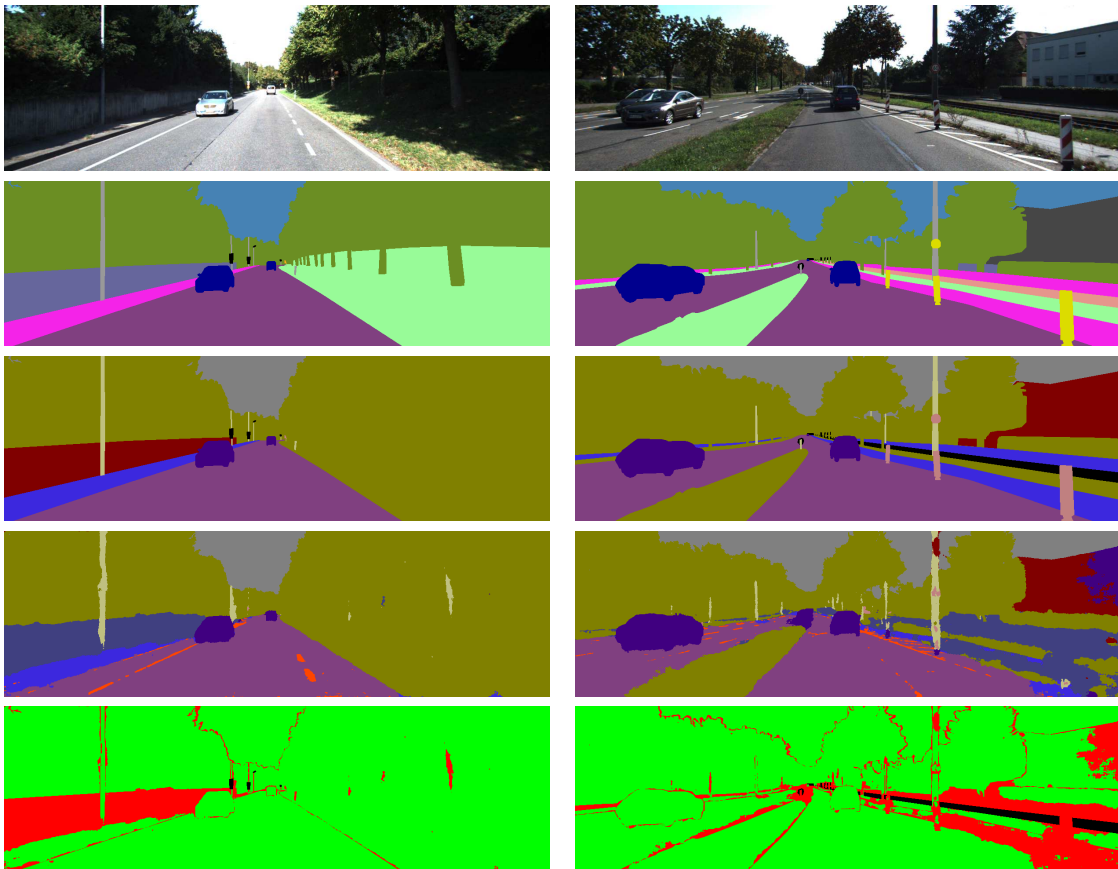


Figure 4.10: SemSegImg Node - Evaluation Image Set - awid\_KITTL\_SemSeg Frames 115 (left) And 92 (right) - Highest And 2nd Highest FP In SegNet Class *Fence*

The reasoning in this chapter shows that the IoU is a good indicator, but by itself not enough to interpret segmentation accuracy and to make an educated decision regarding an implementation and its underlying technologies. Despite some limitations and potential for improvement, especially regarding the available dataset, it can clearly be stated, that the awid\_SemSegImg node's implementation as well as the used technologies such as SegNet represent a useful solution to satisfy Goals 1 and 2.

### 4.3 SemSegPnt Node Evaluation

Evaluation of the `awid_SemSegPnt` node is done by determining execution time, actual memory consumption, spatio-temporal point cloud to camera image alignment and the node’s point cloud segmentation accuracy. The node’s point cloud segmentation accuracy is a result of, and as such represents, the cross-modal transfer of class labels from pixels of semantically segmented 2D images to points of a 3D point cloud. Assessment of segmentation accuracy is based on comparing the classes assigned to points by the `awid_SemSegPnt` node to the corresponding ground truth.

Ground truth class labels are provided in form of tracklet labels. To enable the just described comparison, classes represented by tracklet labels need to be mapped to appropriate SegNet classes. Tracklet labels are available for the following eight classes: *Car*, *Van*, *Truck*, *Pedestrian*, *Person (sitting)*, *Cyclist*, *Tram*, *Misc*. These labels however, do neither match those of Cityscapes [79] or those of SegNet [19]. Consequently a new mapping is required. As listed in Table 4.3 tracklet labels *Car*, *Van*, *Truck*, *Tram* are combined in the SegNet class *Vehicle*, tracklet labels *Pedestrian* and *Person* are mapped to SegNet class *Pedestrian* and *Cyclist* to *Bike*. The remaining tracklet label *Misc* is ignored.

KITTI Tracklet Label	SegNet		
Class Name	Class ID	Class Name	Color
Car	9	Vehicle	
Van	9	Vehicle	
Truck	9	Vehicle	
Tram	9	Vehicle	
Pedestrian	10	Pedestrian	
Person	10	Pedestrian	
Cyclist	11	Bike	

Table 4.3: Class Mapping - KITTI Tracklet Label  $\rightarrow$  SegNet

#### 4.3.1 Point Cloud 3D Evaluation Methodology

All *measurements* to evaluate the `awid_SemSegPnt` node and consequently the implemented overall system are performed on the `awid_KITTI_SemSeg` dataset and where possible on the *target platform MIG* that is specified in Section 2.2. Since no ground truth data for the MIG is available, solely the *awid\_KITTI\_SemSeg* dataset is used for quantitative assessments. The dataset consists of synchronized, rectified, spatio-temporally aligned data and thus serves as ground truth. The data is composed into the generated rosbag file `awid_kitti_20180_07_15_drive_0001_syncedGeneratedTimestamps.bag` that satisfies the requirement of continuous temporal proceeding time information, necessary for the node’s synchronization of multiple subscribers. The characteristics of the KITTI autonomous driving platform are provided in Appendix B. The cameras are hardware-triggered by the LiDAR scanner when it is facing forward. As listed in Tables B.1 and B.2 the system operates at 10 Hz.

Evaluation on the MIG autonomous car platform, introduced in Section 2.2, is done in live mode or with the bag file `20180202-092743-3_laps-Reinickendorf.bag`. The MIG’s sensors are free-running and synchronized based solely on ROS timestamp information as described in Section 3.5.

The *quantitative assessment methodology* for determining the node’s *execution time* and *actual memory consumption* corresponds to the one in Section 4.2. The `awid_SemSegPnt` node is not using the Caffe framework, therefore the execution duration is timed through implementation of the `ros::Time` class instead.

*Quantitative assessment* of the node’s *segmentation accuracy* is performed by comparing the ground truth to the cross-labeled class labels of the 3D cloud’s points. However, the available ground truth for 3D points provided in the KITTI Vision Benchmark Suite is comparatively limited and does not allow a pointwise equivalent to the pixelwise assessment described in Section 4.2.1. As already described in Section 4.1, 3D ground truth data is provisioned as tracklet labels accumulated in `tracklet_labels.xml` files. An object’s volumetric extension is defined once by assigning values to an encasing 3D bounding box. Only objects with a minimum height of 25 pixels are considered for tracklet labeling. The object is then tracked and the box’s position and orientation accordingly adjusted over the number of frames in which it appears. The KITTI 3D object detection benchmark’s evaluation criteria is defined as overlap between ground truth and detected 3D bounding boxes. However, a 3D box is expected to be a too rough estimate of an object’s shape for a meaningful pointwise assessment and the `awid_SemSegPnt` node is not intended to be a 3D box-generating object detector. Consequently there is no box overlap to quantify. Also not all points inside the ground truth’s 3D bounding box can be determined to be TPs or FNs. All points outside the ground truth boxes, which are by far the majority of the cloud’s points, are unlabeled and can not be determined for certain to be TNs or FPs. For example correctly SemSeg-labeled objects represented by less than 25 pixels height could be appearing outside any of the tracklet 3D ground truth boxes and falsely determined to be FPs. Furthermore, the amount of diverse tracklet labels as well as the number of objects within a scene that belong to one of those tracklet labels is rather limited. Considering the `awid_KITTLSemSeg` dataset’s class distribution, this eventually leaves only the class *Vehicle* for potentially meaningful 3D assessment.

With the available point cloud ground truth data the quantitative evaluation is eventually limited to determine the amount or ratio of TPs of the SemSeg-colored points inside the volumes encased by the ground truth 3D bounding box. To get an indication if the implemented approach satisfies Goal 3, it is considered sufficient to do this for some selected scenes of the `awid_KITTLSemSeg` dataset. After all, the `awid_SemSegPnt` node’s segmentation accuracy is based on that of the `awid_SemSegImg` node and the latter has been thoroughly assessed in Section 4.2 with quantitative as well as qualitative results.

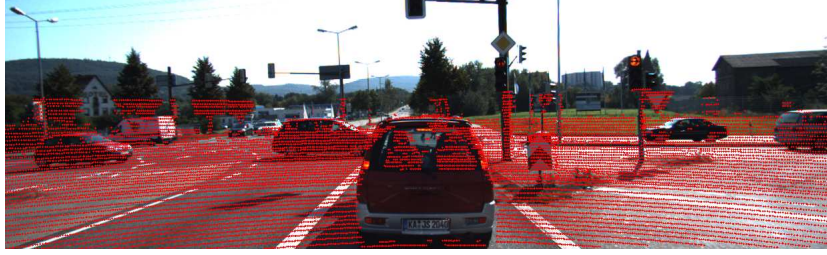
Moreover the system’s *point cloud semantic segmentation accuracy* and thus the implemented approach to cross-label through perspective projection, is *qualitatively assessed* by visually examining the `awid_SemSegPnt` node-generated SemSeg-colored point clouds and by comparing the SemSeg cloud’s point labels to those of the corresponding KITTI ground truth tracklet labels. *Qualitative evaluation* is done by visually determining the `awid_SemSegPnt` node’s spatio-temporal point cloud alignment as well as its semantic segmentation accuracy.

Point cloud *overlay images*, generated when the node’s `-overlay` option is activated, serve as visually accessible representation of the *spatio-temporal alignment* of a camera image and its corresponding point cloud. When deploying the KITTI `bag` file with the SemSeg package’s nodes, the `-kitti` option has to be used. In the overlay images the perspective projected LiDAR points are visualized as red dots, superimposed on the camera’s image. For qualitative assessment of the system’s point cloud segmentation accuracy, the SemSeg-colored point clouds that are published by the `awid_SemSegPnt` node are visually inspected. A point’s color represents the class label assigned through the perspective projection-based cross-labeling. Thus the visualized point clouds should show coherently

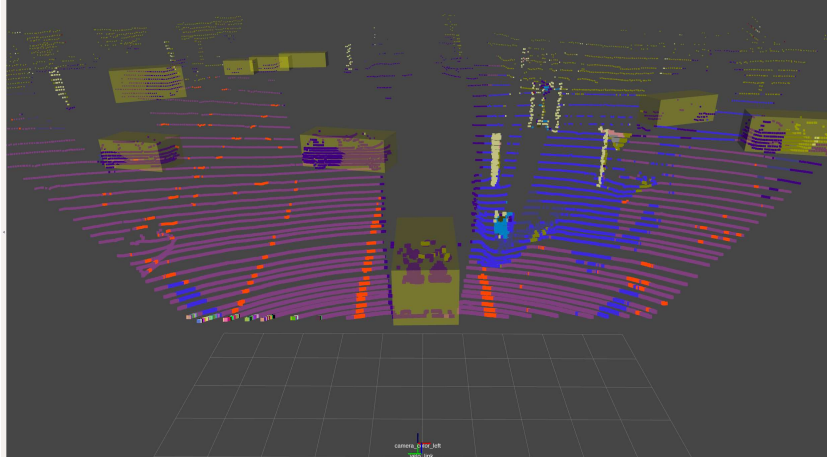


colored volumetric clusters according to the objects within the scene. The SegNet-colored point clouds used in this thesis follow the color scheme defined in Table A.2.

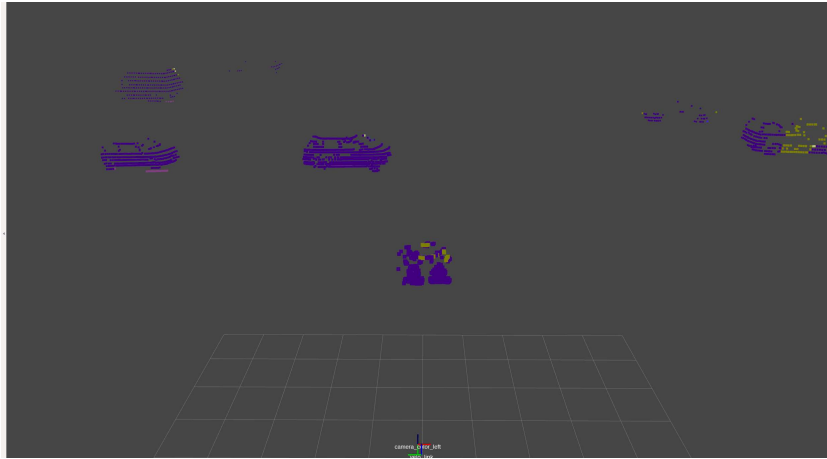
The *point cloud's ground truth* is represented by 3D bounding boxes that are extracted from the KITTI tracklet labels file and visualized as RViz markers in form of semi-transparent yellow boxes. This should allow to easily identify correctly SemSeg-colored points inside the boxes. The 3D bounding boxes are used to box filter the point cloud and only those points of the SemSeg cloud remain, for which ground truth data is available. Figure 4.11 shows a set of images for visual qualitative evaluation: An overlay image (a) and RViz visualizations of the SemSeg-colored point cloud with semi-transparent yellow ground truth bounding boxes (b) as well as SemSeg-colored points remaining after ground truth box-filtering (c).



(a) KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 99



(b) Cross-Labeled SegNet-Colored KITTI Point Cloud With Ground Truth Tracklet Label Boxes



(c) Cross-Labeled SegNet-Colored Points Of Tracklet Labeled Objects

Figure 4.11: SemSegPnt Node - Qualitative Evaluation - Images For Visual Comparison

### 4.3.2 SemSegPnt Resource Consumption Assessment

This section presents the results from performing the assessment methodologies described above. Performance indicators, reflecting the `awid_SemSegPnt` node's resource consumption, are listed in Table 4.4. All underlying measurements have been performed on the system specified in Tables 2.4 and 3.1.

The configurations and thus the options used for the `awid_SemSegImg` node, which provides the segmented images containing the to be transferred pixel labels, correspond to those detailed in Section 4.2.3. An underscore and a second index are postfixed in the naming of the configurations to indicate the options used for the `awid_SemSegPnt` node which are listed in column *Options* of Table 4.4. The `awid_KITTI_SemSeg` dataset's `awid_kitti_20180_07_15_drive_0001_synced_GeneratedTimestamps.bag` file is used to provide KITTI images and the `20180202-092743-3_laps-Reinickendorf.bag` file is the source in SatCam image configurations.

Performance Indicator	Value	Options	Short Description
Execution Time [ms]	$6.552 \pm 0.790$	-kitti	Configuration 1_1: KITTI RGB images 375x1242@10fps
RAM Average [MB]	$42.0 \pm 2.8$		
RAM Min - Max [MB]	37.7 - 51.0		
Execution Time [ms]	$10.240 \pm 1.126$	-kitti -extensive	Configuration 1_2: KITTI RGB images 375x1242@10fps
RAM Average [MB]	$49.4 \pm 5.6$		
RAM Min - Max [MB]	37.4 - 55.6		
Execution Time [ms]	$6.167 \pm 0.650$	-kitti	Configuration 2_1: KITTI RGB images resized to 360x480@10fps
RAM Average [MB]	$40.3 \pm 2.2$		
RAM Min - Max [MB]	36.5 - 48.2		
Execution Time [ms]	$10.263 \pm 0.904$	-kitti -extensive	Configuration 2_2: KITTI RGB images resized to 360x480@10fps
RAM Average [MB]	$47.4 \pm 5.2$		
RAM Min - Max [MB]	37.7 - 53.3		
Execution Time [ms]	$16.769 \pm 6.746$	-extensive	Configuration 3_3: SatCam images resized to 360x480@25fps
RAM Average [MB]	$53.6 \pm 2.3$		
RAM Min - Max [MB]	61.4 - 108.9		

Table 4.4: SemSegPnt Node - Execution Time And Memory Consumption

The difference between configurations `x_1` and `x_2` is the spatial dimension of the semantically labeled SemSeg input images generated and published by the `awid_SemSegImg` node. Whereas configurations `1_1` and `1_2` processes and publishes images in native resolution, the corresponding configurations `2_1` and `2_2` downscale the images to 360 x 480 pixels. The `awid_SemSegPnt` node rescales the semantic input images as described in 3.5. The rescaling from 360 x 480 to 375 x 1242 pixels in configurations `2_1` and `2_2` does not cause the node's execution duration to increase significantly. The point clouds output by the Velodyne LiDAR scanner are sparse compared to the native spatial dimensions of the camera-generated images. Thus considering the savings in `awid_SemSegImg` node's network inference duration achieved through resizing, which account to about 230 ms according to the measurements reported in Section 4.2.3, resizing the input images appears to be an adequate option to increase overall system throughput.

Usage of the `awid_SemSegPnt` node's *-extensive* option is the difference in configurations `1_x` and `2_x`. Comparing configurations `1_1` and `1_2` as well as `2_1` and `2_2` it increases the execution duration by factor 1.6. However, the *-extensive* option's purpose is to offer a higher flexibility in processing on source code side, not necessarily providing a higher performance.

The actual memory consumption is dependent on the scene and the number of reflected laser beams resulting in a LiDAR point reading, which is reflected by the *RAM Min - Max* values reported in according rows of Table 4.4. The resident RAM consumption is about 160 MB higher than the process' consumption listed in the table.

As a result from this assessment the `awid_SemSegPnt` node appears to be suitable for usage in the MIG and such satisfies Goal 3 as formulated in Section 1.3 from a resource consumption perspective.

### 4.3.3 SemSegPnt Alignment And Segmentation Assessment

As demonstrated in Figures 4.12 to 4.14 and 4.11 the implementation detailed in Section 3.5 shows an overall accurate spatio-temporal alignment of camera images and corresponding point clouds for the KITTI autonomous vehicle platform. Resolution and frequency of the LiDAR scanning are sufficient to generate point clouds with a density high enough to even reliably detect small poles as shown in Figure 4.12. Resulting from visual judgment in overlay images the alignment is best around the camera image's center and slightly increasing misalignment is apparent towards the sides as demonstrated in Figure 4.13. The main causes for this are probably optics, rectification, physical displacement of the capturing devices relative positions and the scanning process.

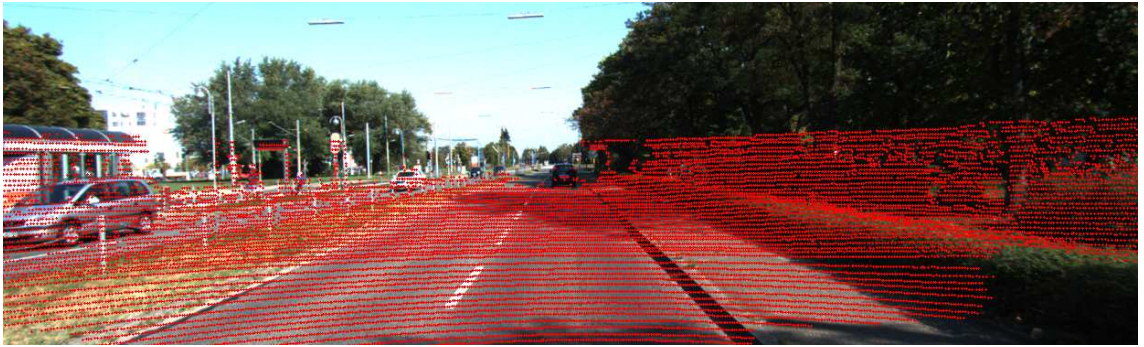


Figure 4.12: KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 27

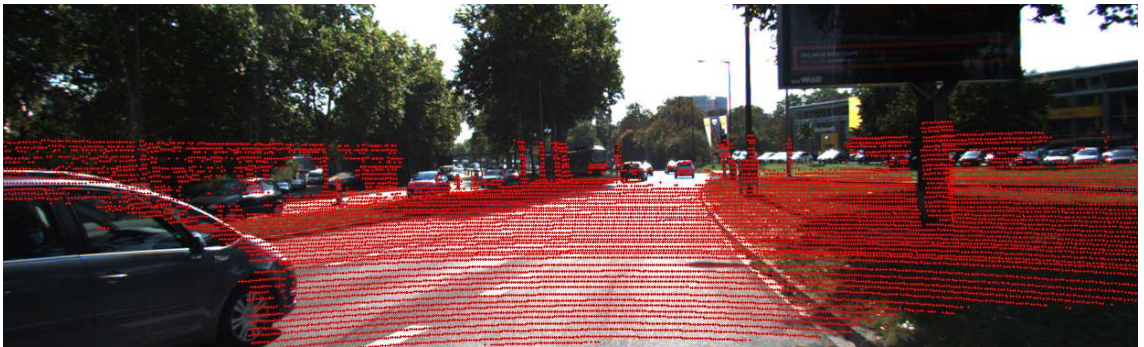


Figure 4.13: KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 84

The LiDAR scanner spins counter clockwise and the camera is triggered when the LiDAR scanner faces forwards. Consequently points around the image's center are perfectly aligned, whereas on the sides it is noticeable that the point cloud was captured at a slightly different point in time than the camera image. The car passing on the left side in Figure 4.13 is close enough to be inside the Velodyne's near clipping range applied in the



implementation, hence the car's shape is clearly showing. The passing car is moving fast enough to see that the point cloud was captured before the camera image. Whereas the KITTI car was not moving in Figure 4.13, it is driving in Figure 4.14, showing the same misalignment increasing towards the image's sides. Removal of the lens distortion caused by the camera's optics strengthens this effect. Overall it appears as if the point cloud is offset slightly to the right.

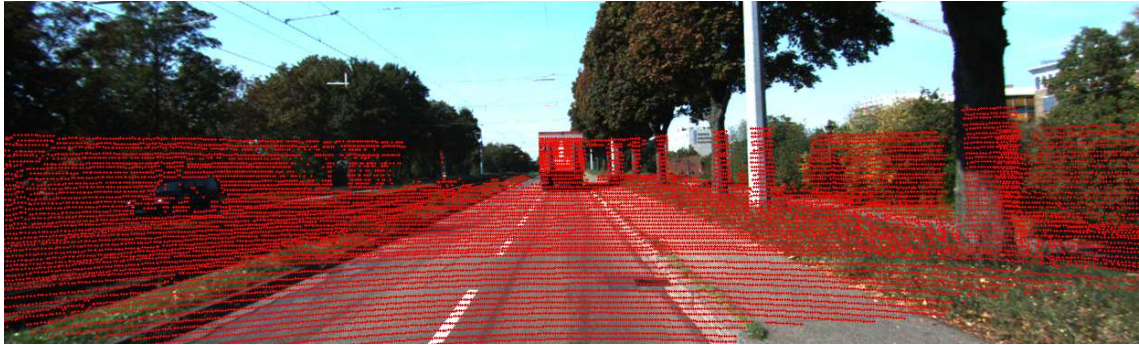


Figure 4.14: KITTI RGB Color Camera Image With Point Cloud Overlay - Frame 78

The overlay images created with the MIG autonomous vehicle platform reveal a comparatively strong spatial as well as temporal misalignment between the front camera's images and the corresponding LiDAR point clouds. Figure 4.15 shows a MIG SatCam overlay image in which the car is not moving.



Figure 4.15: MIG SatCam RGB Color Image With Point Cloud Overlay

The misalignment has several reasons, discussed in the following. Intrinsic as well as extrinsic calibration of the used sensors needs to be redone. For the MIG's cameras this is addressed in [35]. The spatial displacement between the cameras and the LiDAR scanner in combination with the SatCam camera's optical characteristics, especially the very wide fisheye lenses, are disadvantageous and additionally constraining the according

data fusion. A major misalignment is caused by the ROS timestamp-based synchronization of the otherwise independently free-running sensors. As reported in Tables 2.2 and 2.3 the SatCams operate at 30 fps whereas the Velodyne scans at 10 fps. Figure 4.16 demonstrates the resulting misalignment due to different capturing times of the camera and the LiDAR scanner. The MIG was moving forward and image (a) shows that the points representing the tree stump on the left as well as the those reflected by the van on the right side in the image are obviously captured before the camera image. Image (b) shows the directly subsequent overlay image with a temporal relatively correct alignment. Evident through the burned-in timestamp information, the difference in capture time between the camera image and the corresponding point cloud is 622 milliseconds for the first image (a) and only 288 for the subsequent frame (b). With the deployed configuration of the MIG a point cloud is often captured obviously before or after the camera image.

Whereas it was possible to mitigate calibration issues to some extent, it was not possible to compensate the temporal misalignment caused by the independently free-running sensors. For sure a modification of the hardware configuration allowing a signaling-based synchronization would show significant improvement. Also other software-based approaches to synchronize and temporally match the sensor data like Kalman-Filtering or Recurrent Neural Networks (RNN) with LSTM (Long Short-Term Memory) units are options worth investigating in future works.

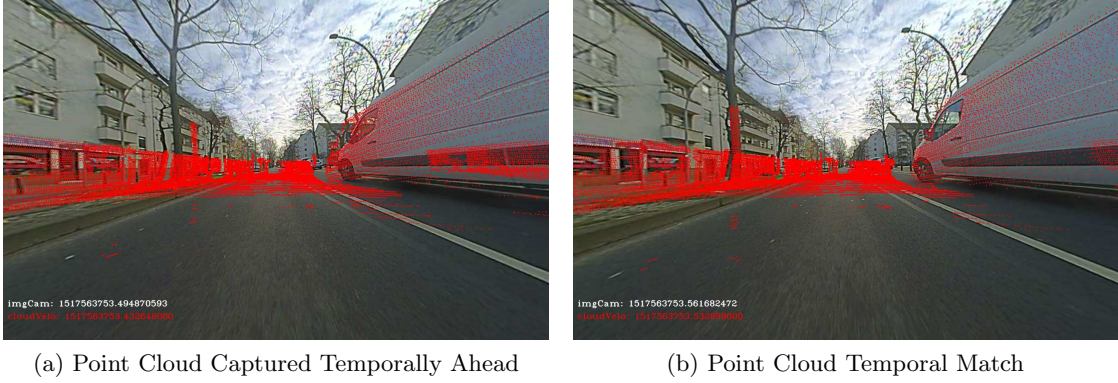


Figure 4.16: MIG SatCam Overlay Images - Point Cloud Alignment

To summarize the alignment assessment of the SemSegPnt node’s implementation, the dataset assembled and described in Section 4.1 provides the information necessary to evaluate the spatio-temporal alignment within the scope and intend of this work. The implemented synchronization and projection of 3D point clouds to corresponding 2D images provides spatially as well as temporally aligned and thus appropriately fused data in case the vehicle platform delivers captured data of sufficient quality. This satisfies Goal 3 with regard to the implemented spatio-temporal alignment as prerequisite for the approach’s cross-modal label transfer.

Frame	Tracklet Labels	$TP_{Frame}$	$FP_{Frame}$	$TPR_{Frame}$	$\overline{IoU}_{Image}$
005	3	815	35	0.96	0.66
066	2	865	107	0.89	0.77
082	7	715	15	0.98	0.73
099	5	1767	144	0.93	0.83
107	13	1828	142	0.93	0.96

Table 4.5: SemSegPnt Node - Quantitative Evaluation - Sample Frames Class *Vehicle*



Examination of the SemSeg-colored point clouds, that are published by the awid\_SemSegPnt node, shows partitioning of the point cloud into coherent volumetric parts, representing objects occurring in the scene.

Figures 4.11 and 4.17 to 4.18 represent according visualization sets of exemplary frames from the awid\_KITTL\_SemSeg dataset. Table 4.5 lists quantitative quality indicators which are discussed on the following example. To distinguish between indicators for 2D images and 3D points, former are subscripted with *Image* and latter with *Frame*.

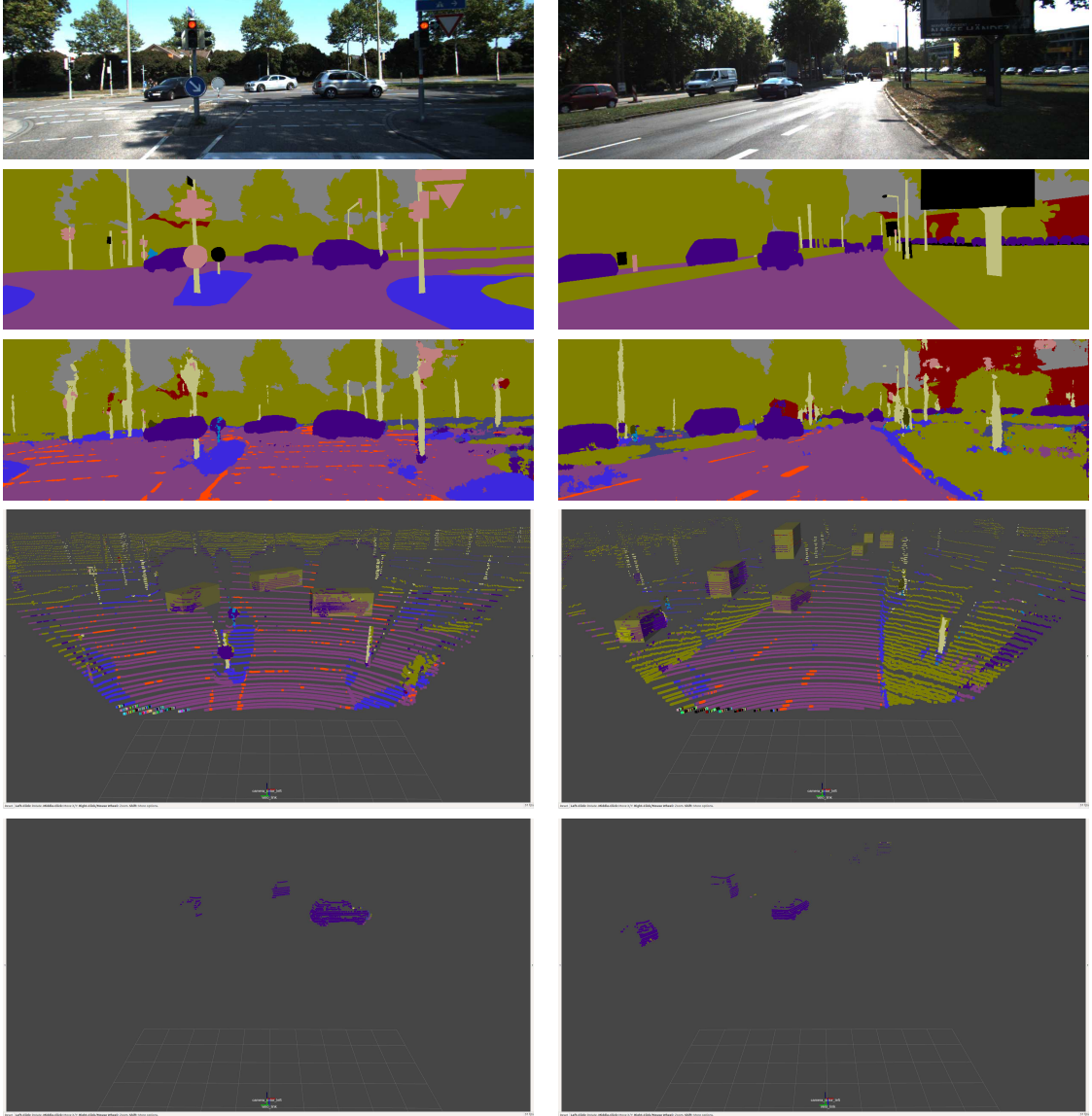


Figure 4.17: SemSegPnt Node - SemSeg-Colored Point Clouds - awid\_KITTL\_SemSeg Dataset Frames 5 (left) and 82 (right)

The example in Figure 4.18 shows evaluation visualizations for the awid\_KITTL\_SemSeg dataset frames 66 on the left and 107 on the right. As explained above, 3D point cloud ground truth data in those frames is only available for objects of the SegNet class *Vehicle*. In the 2D image evaluation in Section 4.2.4, Frame 66 has scored an average per Image  $\overline{IoU}_{Image}$  for class *Vehicle* of 0.77 which conveniently equals the  $\overline{IoU}_{Dataset}$  for this class. Frame 107 has the highest reported  $\overline{IoU}_{Image}$  score of 0.96. With a  $\overline{TPR}_{Dataset}$  of 0.94

and a  $\overline{PPV}_{Dataset}$  of 0.81, the class *Vehicle* does not contain many frames with low scores. As obvious from the ratio of these values, the problem are rather FPs than FNs. However, without any ground truth information outside the tracklet bounding boxes, determining the FPs is not possible. Thus, the selected frames 66 and 107 should serve well as representatives for the class *Vehicle*. For Frame 66 ground truth tracklet labels for 2 objects are available, for frame 107 there are 13, all represented as semi-transparent yellow boxes in Figure 4.18. As also shown, the points remaining after box-filtering are mostly labeled correctly as belonging to class *Vehicle*. In numbers, frame 66 has a recall for class *Vehicle* of  $TPR_{Frame} = 0.89$  with 865 correctly labeled points i.e. TPs and 107 FNs inside the 3D ground truth bounding box. Frame 107 achieves a  $TPR_{Frame}$  of 0.93 with 1828 TPs and 142 FNs in the encasing box. The point readings of the two nearest vehicles parked on

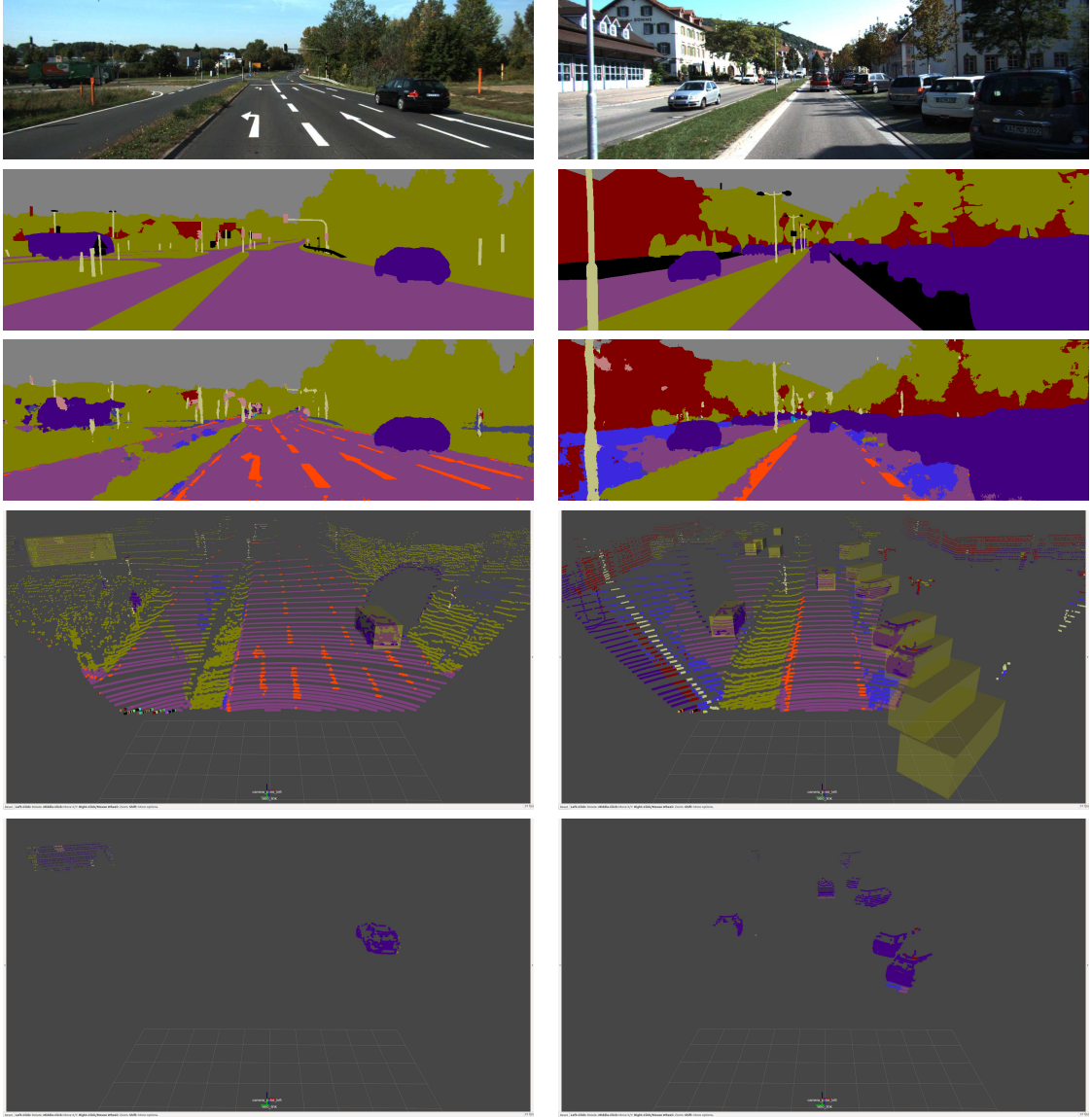


Figure 4.18: SemSegPnt Node - SemSeg-Colored Clouds - awid\_KITTLSemSeg Frames 66 (left) and 107 (right) - Average And Highest  $\overline{ToU}_{Image}$  For Class *Vehicle*

the right side in frame 107 are clipped due to near range settings for the Velodyne scanner in the implemented camera frustum cropping and thus excluded from above  $TPR$  calculations. However, the also displayed SegNet-colored SemSegImg node's output image

shows correct labeling of the according region, therefore resulting in potentially even higher  $TPR$  scores. As already discussed in Section 4.3.1, no ground truth is available outside the tracklet label boxes, making quantitative statements such as calculation of FPs and TNs impossible. Also due to the box-shaped and not pointwise ground truth, calculations can only serve as indication and are to be interpreted accordingly.

The visualizations of the SemSeg-colored point clouds illustrate the gaps in the point clouds due to occlusion. Also clearly visible is the misclassification of points around the objects edges which is perspectively extended into the scene. The faulty labeling is caused by the already wrong label assignment provided by the SemSegImg node's 2D image segmentation and not by the perspective projection which however admittedly spreads and thus intensifies the misclassification.

Deploying the implemented system including the `awid_SemSegPnt` node on the MIG also produces cross-labeled semantically segmented point clouds with clearly recognizable scenery and objects as depicted in Figure 4.19. The corona-building delineation deficiencies in the semantic segmentation of the 2D images cause the labels assigned to points close to the camera to be shadow-casted onto all more distant points. Apart from the calibration and synchronization that need refinement, the low positioning of the SatCam front camera and the ultra wide fisheye optics cause this effect to be even stronger in the MIG autonomous vehicle platform. The sequence of visualizations on the left side of Figure 4.20 demonstrates how the labels of the passing car are casted onto the building behind it. The pictures on the right half of Figure 4.20 show the effect resulting from the front SatCam camera's low position and fisheye lens in conjunction with the physical distance between camera and Velodyne scanner. The closer the MIG comes to the car in front of it, the more severe is the faulty labeling due to the just described label-casting. The camera's low position results in according occlusion and the surroundings sensed by the camera and the LiDAR scanner do not match. The cross-labeling approach through perspective projection inevitably fails in such a scenario. This however does not exclude a meaningful sensor data fusion. Occlusion effecting one sensor system could be compensated by the another one, providing a more comprehensive environmental perception.

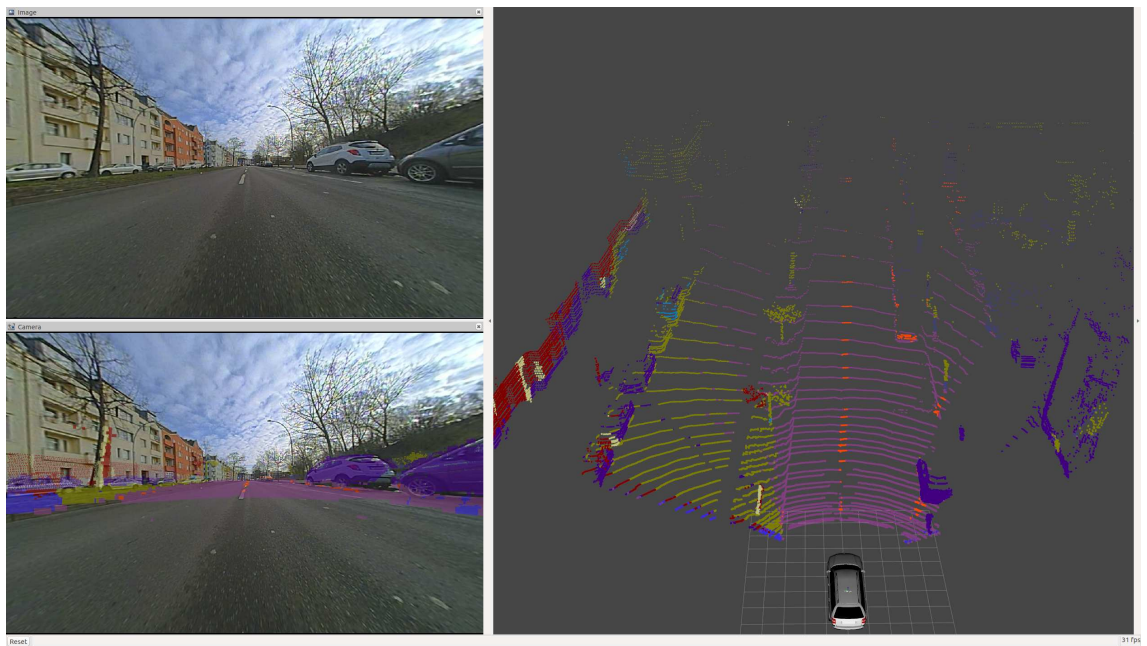


Figure 4.19: SemSegPnt Node - MIG Overlay Image And SegNet-Colored Point Cloud



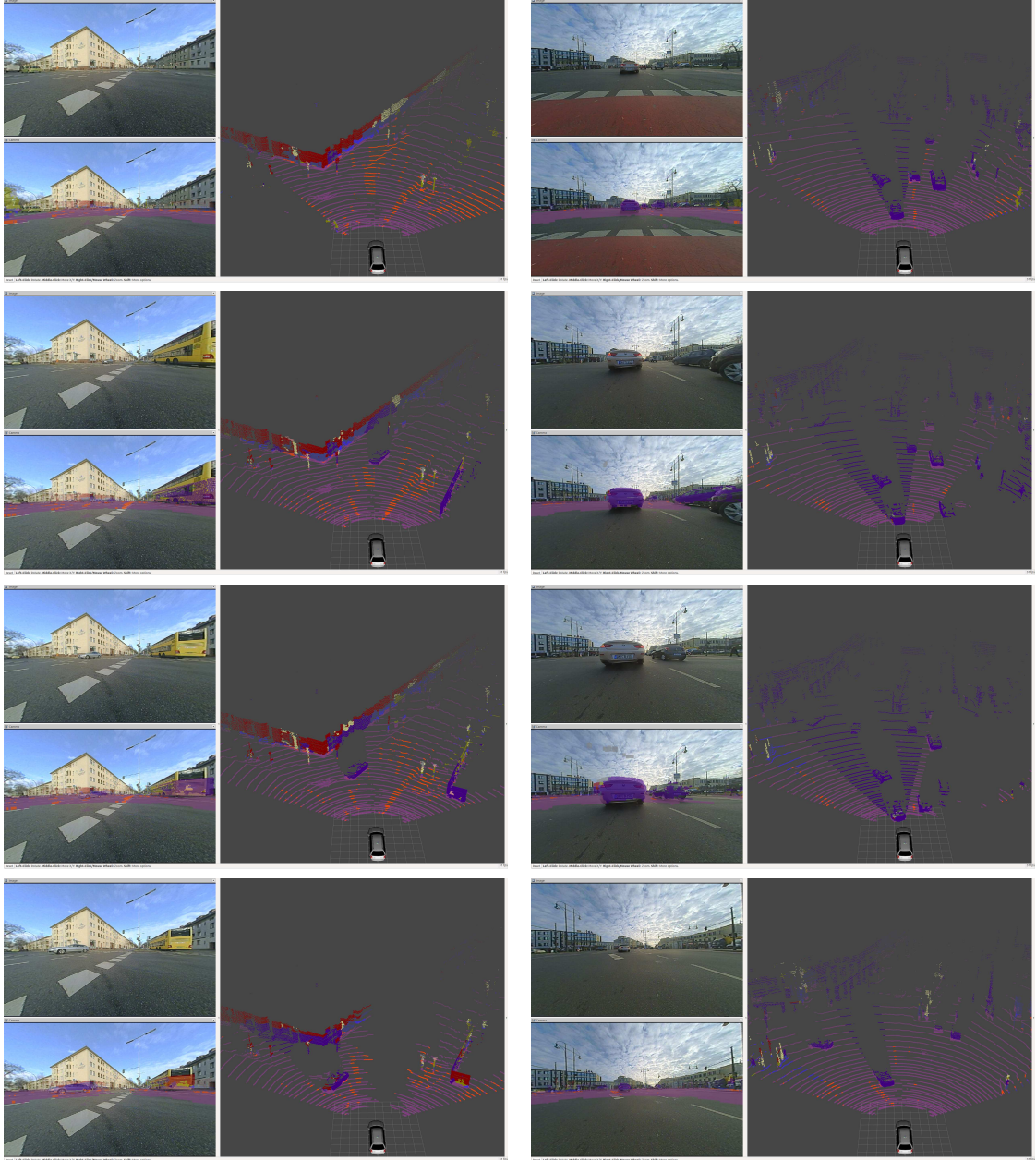


Figure 4.20: SemSegPnt Node - MIG SatCam Images And SegNet-Colored Point Clouds  
- Label-Casting And Occlusion

Provided the underlying semantic segmentation of the camera's 2D images is accurate, the implemented cross-modal label transfer to corresponding 3D point clouds satisfies Goal 3. Overall the system mainly consisting of the `awid_SemSegImg` and `awid_SemSegPnt` nodes as well as the CAFFE support package and the approach to transfer labels across modalities through perspective projection appears to produce respectable results for semantic segmentation of LiDAR-originated point clouds. Consequently, it can be stated, that the approach and its implementation, satisfy Goals 1, 2 and 3, as described in this work. Thus providing an applicable solution for pixelwise semantic segmentation of camera images with a deep neural network architecture and a cross-modal transfer of semantic labels from pixels of a 2D image to points of a 3D LiDAR-based point cloud in a real world autonomous driving vehicle.

## Chapter 5

# Conclusion

This chapter gives a conclusive summary of the accomplished work and the key findings gathered. Finally it presents potentially interesting directions for future work taking up on this master thesis.

### 5.1 Conclusive Summary

The aim of this thesis was to extend a real world autonomous vehicle’s capabilities towards a more comprehensive environmental perception by designing and implementing a system that deploys a deep neural network architecture and sensor data fusion for multi-modal semantic segmentation of camera images and LiDAR point clouds. For this purpose, three goals were formulated and pursued during the course of this master thesis:

1. A Caffe-based deep neural network framework was integrated into the existing ROS software framework of the MIG autonomous vehicle.
2. A software package was developed and implemented as general mechanism to load and deploy Caffe-based neural network architectures on the MIG autonomous vehicle platform, demonstrated on the example of SegNet and its application for pixelwise semantic segmentation of the car’s camera images.
3. A software package was developed and implemented to semantically segment the LiDAR-generated 3D point cloud of the MIG autonomous vehicle through fusion of multi-modal sensor data and cross-modal transfer of semantic labels from corresponding pixels of camera-originated 2D images.

These software packages are components of an overall system and build upon each other, such that the Caffe framework from goal 1 is a prerequisite for the deployment of SegNet in goal 2, which again generates the semantically segmented images being the necessary input for the cross-modal label transfer addressed in goal 3. The system’s architectural design, its implementation and application are described in Chapter 3 and further insights are provided through the source code and its code documentation. A git repository [20] containing the successfully implemented system’s components and according installation scripts has been created.

The contextual suitability of the implemented system and its underlying technologies were assessed in Chapter 4. The requirements of an appropriate assessment dataset were defined and an according dataset was composed to the best abilities within the scope of this work. Due to nonexistent ground truth data from the MIG autonomous vehicle, the dataset is based on publicly available assets that were captured with the KITTI driving



platform. A class mapping scheme from KITTI to SegNet was developed and the ground truth data was transformed accordingly. Deployment of the dataset during evaluation demonstrated general adequateness but also revealed limitations, being mainly the unbalanced class distribution and underrepresentation for important classes like *Pedestrian* or *Bike*. The KITTI 3D ground truth data available in the assembled dataset, represented by 3D bounding box information in form of tracklet labels for some classes, did not enable a meaningful quantitative evaluation of pointwise semantic segmentation accuracy. This would have required a comprehensive pointwise labeling of the point cloud, for more classes, as well as also including points outside the tracklet label boxes. Despite some limitations, the dataset served as adequate foundation for the intended assessment of the contextual usefulness of the implemented system approach.

In further preparation of the assessment, according evaluation methodologies were defined and described. The system’s accuracy for the pixelwise segmentation of 2D camera images was quantitatively and qualitatively assessed. Precision (TPR), Recall (PPV) and IoU (Jaccard Index) for each of the SegNet classes were calculated per image as well as for the complete dataset. Results of the quantitative, guided the qualitative evaluation performed through visual inspection and comparison of prediction and ground truth as well as visualizations of correct and incorrect labeled pixels. The performed evaluation showed that the IoU is a good indicator, but only the combination with visual inspection allowed to properly interpret segmentation accuracy to make an educated decision regarding the implementation and its underlying technologies.

The cross-modal label transfer from 2D pixels to points defined in 3D space, realized through perspective projection of the LiDAR scanner’s point cloud to the camera’s image plane, necessitates according spatio-temporal alignment. Assessment of the implementations spatio-temporal alignment of camera images and corresponding LiDAR point clouds was done with overlay images containing the perspectively projected point cloud superimposed on the camera’s image.

The implemented system’s segmentation accuracy for 3D point clouds was primarily evaluated based on visual inspection of visualizations of semantically cross-labeled and thus accordingly colored point clouds. Visual representations of ground truth tracklet labels in form of semi-transparent boxes were included in case of KITTI data. Quantitative evaluation regarding semantic segmentation of point clouds was limited as described in the above paragraph on the dataset. True Positives (TPs) and True Positive Rate (TPR) were calculated for some selected frames of the dataset.

Measured values reported of the overall system’s and its components’ memory consumption and execution duration were acceptable for the thesis’ target platform, whereby deep neural network architectures remain a challenging component to be fitted into an autonomous vehicle. The memory consumption as well as the execution duration were almost exclusively determined by the deep neural network architecture deployed for the pixelwise semantic segmentation of the camera images. For the selected example network architecture SegNet, the memory consumption was in the range of 1.7GB to 3.0GB for the tested KITTI configurations and 3.9GB for the MIG. The memory usage was driven by the spatial resolution of the images fed into the network’s input layer. Naturally for deep networks, spatial downsampling of the input images significantly reduced the resource consumption with respect to memory usage as well as execution duration. With more than 90% of the overall system’s execution duration accounted to the neural network’s inference time, resizing of input images appeared to be an adequate method to reduce resource requirements and increase the system’s throughput rate. Reducing the pixel count by factor 2.7 increased the output rate of segmented point clouds from below 3 fps to about 6 fps. Processing the neural network on the GPU instead of the CPU was 82 times faster.

The implementation scored an  $\overline{IoU}_{Weighted\_Dataset}$  of 73.14% which is the overall IoU for all images of the dataset weighted by the number of pixel occurrences per class.

The best image segmentation results were achieved for SegNet classes *RoadMarking + Road*, *Vehicle*, *Tree* and *Sky*, whereas classes *Pedestrian*, *Bike*, *Sign* and *Fence* did not perform well. As described above, this was partially due to the limited available evaluation dataset. Investigation of the classes *Building* and *Fence* revealed inadequacies in the class mapping. The network showed difficulties to generalize in the class *Sign*, especially misclassifying a particular traffic sign, which is expected to be a matter of training rather than network architecture. Even without applying spatial scaling outside the neural network, thus using the camera’s images in native resolution, boundary delineation, especially for less frequent classes, representing thinner and smaller objects, was identified as area worthy of improvement. Deficiencies in delineation that occurred already in the semantic segmentation of 2D images, caused faulty classifications around objects. These were perspective spread into the scene and resulted in accordingly high amounts of incorrectly labeled points of the 3D point cloud. The effect of shadow-casting labels to points around and behind an object was strengthened by the camera’s ultra wide fisheye optics and thus had an even stronger impact on the MIG compared to the KITTI vehicle platform.

The absolute as well as relative positioning of the different sensors affected the sensor data fusion. The low position of the MIG’s front camera caused according occlusion and in combination with the distance to the LiDAR scanner and the thus stronger divergence in perspective, the implemented approach of correctly cross-labeling through perspective projection inevitably failed when objects occurred in close distance in front of the car.

The MIG showed general and rather strong spatial as well as temporal alignment issues. The spatial misalignment was mainly due to necessary intrinsic and extrinsic recalibration of the deployed sensors and was mitigated. However, it was favorable to use the rectified images. The MIG’s sensors were independently free-running and hence unsynchronized which resulted in temporal misalignment of the camera’s image and the corresponding point cloud. Usage of the hardware-synchronized KITTI-based data from the composed dataset demonstrated accurate spatio-temporal alignment which evidenced the usefulness of the implemented approach, provided that the captured data is of sufficient quality.

In the implemented cross-labeling approach, the accuracy of the semantic segmentation of a point cloud depended on that of the preceding semantic segmentation of the corresponding 2D image. Assessment of the implemented system’s semantic segmentation accuracy for point clouds reported a  $TPR_{Frame}$  of 0.89 and higher for class *Vehicle* of the selected frames. Examination of visualized semantically labeled and accordingly colored point clouds revealed that False Positives (FPs) are more of an issue than False Negatives (FNs), especially for a class like *Vehicle* with high scores in the underlying 2D  $IoU_{Image}$  ratings. Apart from the already described deficiencies, the visualized point clouds were in general appropriately semantically segmented.

The evaluation of the system evidenced the implementation’s contextual suitability. The system demonstrated decent results for pixelwise semantic partitioning of 2D camera images as well as sensor data fusion-based cross-modal label transfer and resulting pointwise semantic segmentation of corresponding LiDAR-generated 3D point clouds. Conclusively this thesis presents a modularized, extensible system that operates on a real world autonomous vehicle and expands its capabilities towards a more comprehensive environmental perception and road scene understanding, which eventually allows gathering of insights in real world traffic and thus facilitates further research in the area of autonomous vehicles.

## 5.2 Future Work

The implemented modular system and its components were primarily designed as applicable proof of concept with flexibility and accessibility in mind to allow uncomplicated modification depending on future application and research interests. Some potentially interesting areas for future work taking up on this master thesis are presented in the following.

Obvious and rather easy to achieve are some gains in system performance with regard to memory consumption and execution duration. The system was purposely implemented in C++ to allow a simple switch from ROS nodes to nodelets.

SegNet, being the performance-determining component of the current implementation, is solely deployed in inference mode and can therefore be optimized by merging batch normalization and convolutional layers. An according script [85] is available and reported to speed up SegNet by around 30%. It appears to be worth investigating technologies like Nvidia's TensorRT, a high-performance deep learning inference acceleration library, to automatically optimize trained neural networks for run-time performance. Application in automotive deep learning, using the Cityscapes dataset and deploying a modified VGG16-based network architecture were reported [86] to achieve an about 5x higher inference throughput and lower latency compared to the original network running in Caffe. The implemented system is designed to experiment with other network architectures and deploying other supposingly much faster architectures like ENet [87] should be a straightforward process. Possibilities to reduce the neural network's parameters as well as finding alternative network architectures are areas of active research. The MIG in conjunction with the implemented system represent a useful platform for according research in the context of autonomous driving. The assembled dataset, assessment methodologies and tools can serve as testbed to generate comparable and reproduceable data for evaluation of different technologies and setups.

As demonstrated in the thesis, the camera's images ingested into the deep neural network architecture drive the resource consumption. If not the segmented image, but the semantically segmented point cloud is of primary interest, cropping the camera's image to the area covered by the projected LiDAR-generated point cloud is a simple approach to improve system throughput significantly. Investigation of means to control the pixelcount through spatial or temporal resizing while meeting acceptable quality thresholds is obviously an area of greater interest. Spatial resizing will inevitably deteriorate delineation accuracy of semantic segmentation at some point. Depending on the camera system and the imaging pipeline, a higher pixelcount might not necessarily contribute to better segmentation accuracy while certainly burdening the system's performance. Less for the KITTI vehicle than for the MIG and its cameras with their even wider fisheye optics, it seemed that the high spatial image resolution, especially in off-center image areas, did not contribute to achieve a better delineation accuracy. In the implemented system, deficiencies in delineation of the camera image's semantic segments have been identified to have an extraordinary negative impact on the semantic segmentation accuracy of the cross-labeled point clouds. SegNet already included architectural measures to improve boundary delineation, but obviously further research should be considered.

Not surprisingly, usage of faster hardware immediately increases the system's throughput rate. This alone already showed an output rate of 10 fps, thus realtime performance related to the Velodyne LiDAR scanner, on a laptop equipped with an Nvidia GeForce GTX 1080 GPU. Deployment of the system on dedicated specialized hardware like Nvidia's Drive PX or Drive AGX would allow to broaden the knowledge about the system's resource requirements and make it more applicable for usage in future production cars.

The inspection of sequences of semantically segmented images revealed that consideration

of inter-frame relations and therefore according technologies such as Kalman-Filtering or Recursive Neural Networks (RNNs) with LSTM (Long-Short-Term-Memory) units, are almost certain to improve semantic segmentation accuracy. An object incorrectly labeled in one frame, appeared correctly labeled in successive frames. Consideration of inter-frame relations can also be used to correct faulty spatio-temporal alignment. With respect to the MIG, a signaling-based synchronization of the sensors would result in better spatio-temporal alignment. With a refinement of the intrinsic and extrinsic calibration of the MIG autonomous vehicle's cameras and LiDAR sensor, including accurate data on the lens distortion, it will be possible to operate on unrectified raw camera images, which eliminates resource consumption by according processes. Furthermore it would allow to reduce the current implementation's message buffer size used to synchronize and fuse the sensor data, freeing up additional resources.

With the sheer endless capabilities of deep neural networks the question arises why the semantic segmentation of the point cloud is not also performed by the network that segments the images. In case of SegNet, the authors believe that "using depth information for segmentation merits a separate body of work" [28]. SqueezeSeg [88] is an approach to segment road-objects from 3D LiDAR point clouds. The approach involves projection of the point cloud to a sphere and a recurrent CRF (Conditional Random Field) as final layer to refine the label map for better boundary delineation.

The presented cross-modal label-transfer is an attempt to generate data and transfer its semantic. The data generated by one component of the system could be used to train another component. It would be interesting to investigate the possibilities to build a neural network that generates data to train another neural network. Hence extending a neural networks semantic segmentation capabilities by adding another modality.

The expanded perceptual capabilities of the autonomous vehicle provided by the implemented system can also be combined with other components of the vehicle enabling higher-level functions. Information from semantic segmentation could for example be used to generate map information or be fed into the trajectory planning for accident mitigation. In case of an inevitable impact, advanced safety measures could be initiated, depending on the scenery and the parties involved. If for example a person has been identified to be hit by the car, the hood is deformed or external airbags are triggered to reduce injuries.

The potentially useful applications of the implemented system are manifold such as the possibilities of its extension and optimization. The presented system, the described technologies and methodologies as well as the set of data and tools provide a good starting point for future work.

## Appendix A

# SegNet - Deep CNN for Semantic Segmentation

The SegNet architecture is described in Section 3.2 and in [19] and [28]. SegNet network visualizations in this thesis follow the color scheme listed in Table A.1. Affiliation to one of the 12 segmentation classes is done in accordance with the color scheme in Table A.2.

Color	Type	Color Name	HEX	R	G	B
	Input	Light_Coral	#ff8080	255	128	128
	Convolution, ReLU, Batch Normal.	Light_Green	#90ee90	144	238	144
	Pooling	Fuchsia_Pink	#ff80ff	255	128	255
	Upsampling	Witch_Haze	#ffff80	255	255	128
	Softmax, Argmax, Output	Light_Slate_Blue	#8080ff	128	128	255
	Blob	Gainsboro	#e0e0e0	224	224	224

Table A.1: SegNet Network Visualization Color Scheme













Color	Class	Color Name	HEX	R	G	B	PCL Packing
	Sky	Gray	#808080	128	128	128	8421504
	Building	Maroon	#800000	128	0	0	8388608
	Pole	Pine_Glade	#c0c080	192	192	128	12632192
	Road_Marking	OrangeRed	#ff4500	255	69	0	16729344
	Road	Red_Robin	#804080	128	64	128	8405120
	Pavement	Han_Purple	#3c28de	60	40	222	3942622
	Tree	Olive	#808000	128	128	0	8421376
	Sign	Oriental_Pink Symbol	#c08080	192	128	128	12615808
	Fence	Jacksons_Purple	#404080	64	64	128	4210816
	Vehicle	Indigo	#400080	64	0	128	4194432
	Pedestrian	Trutle_Green	#404000	64	64	0	4210688
	Bike	Cerulean	#0080c0	0	128	192	32960

Table A.2: SegNet Class Color Scheme

Calculations in Tables A.3 and A.4 are based on Equations 2.27 and 2.28 and the SegNet Webdemo prototxt network model [89]. Memory requirements are based on 4 byte per parameter, hence as reported to the command line during network loading. Actual memory usage depends on implementation and usually exceeds the calculated value. The network's input dimensions originate from training with the CamVid dataset [67]. Inputs to pooling

layers are padded in case they are not divisible by 2, resulting in accordingly rounded up spatial dimensions of affected layers.

layer count	layer type	layer name	height [px]	width [px]	depth	spatial scale factor	depth scale factor	kernel size	memory requirements [Byte]	number of parameters
	input	input	360	480	3				2073600	
1	conv3-64	conv1_1			64			3	46310400	1728
	bn	conv1_1_bn							90547200	
	relu	relu1_1							134784000	
2	conv3-64	conv1_2			64			3	179020800	36864
	bn	conv1_2_bn							223257600	
	relu	relu1_2							267494400	
	maxpool	pool1	180	240		2	2		289612800	
3	conv3-128	conv2_1			128			3	311731200	73728
	bn	conv2_1_bn							333849600	
	relu	relu2_1							355968000	
4	conv3-128	conv2_2			128			3	378086400	147456
	bn	conv2_2_bn							400204800	
	relu	relu2_2							422323200	
	maxpool	pool2	90	120		2	2		433382400	
5	conv3-256	conv3_1			256			3	444441600	294912
	bn	conv3_1_bn							455500800	
	relu	relu3_1							466560000	
6	conv3-256	conv3_2			256			3	477619200	589824
	bn	conv3_2_bn							488678400	
	relu	relu3_2							499737600	
7	conv3-256	conv3_3			256			3	510796800	589824
	bn	conv3_3_bn							521856000	
	relu	relu3_3							532915200	
	maxpool	pool3	45	60		2	2		538444800	
8	conv3-512	conv4_1			512			3	543974400	1179648
	bn	conv4_1_bn							549504000	
	relu	relu4_1							555033600	
9	conv3-512	conv4_2			512			3	560563200	2359296
	bn	conv4_2_bn							566092800	
	relu	relu4_2							571622400	
10	conv3-512	conv4_3			512			3	577152000	2359296
	bn	conv4_3_bn							582681600	
	relu	relu4_3							588211200	
	maxpool	pool4	23	30		2	2		591037440	
11	conv3-512	conv5_1			512			3	592450560	2359296
	bn	conv5_1_bn							593863680	
	relu	relu5_1							595276800	
12	conv3-512	conv5_2			512			3	596689920	2359296
	bn	conv5_2_bn							598103040	
	relu	relu5_2							599516160	
13	conv3-512	conv5_3			512			3	600929280	2359296
	bn	conv5_3_bn							602342400	
	relu	relu5_3							603755520	
	maxpool	pool5	12	15		2	2		604492800	

Table A.3: SegNet Layers Part 1 - Input And Encoders

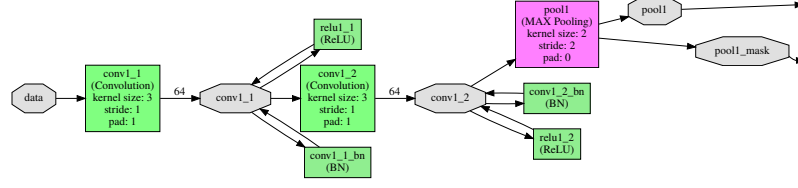
layer count	layer type	layer name	height [px]	width [px]	depth	spatial scale factor	depth scale factor	kernel size	memory requirements [Byte]	number of parameters
	upsample	upsample5	23	30		2	2		605905920	
14	conv3-512	conv5_3_D			512			3	607319040	2359296
	bn	conv5_3_D_bn							608732160	
	relu	relu5_3_D							610145280	
15	conv3-512	conv5_2_D			512			3	611558400	2359296
	bn	conv5_2_D_bn							612971520	
	relu	relu5_2_D							614384640	
16	conv3-512	conv5_1_D			512			3	615797760	2359296
	bn	conv5_1_D_bn							617210880	
	relu	relu5_1_D							618624000	
	upsample	upsample4	45	60		2	2		624153600	
17	conv3-512	conv4_3_D			512			3	629683200	2359296
	bn								635212800	
	relu								640742400	
18	conv3-512	conv4_2_D			512			3	646272000	2359296
	bn								651801600	
	relu								657331200	
19	conv3-256	conv4_1_D			256			3	660096000	1179648
	bn								662860800	
	relu								665625600	
	upsample	upsample3	90	120		2	2		676684800	
20	conv3-256	conv3_3_D			256			3	687744000	589824
	bn								698803200	
	relu								709862400	
21	conv3-256	conv3_2_D			256			3	720921600	589824
	bn								731980800	
	relu								743040000	
22	conv3-128	conv3_1_D			128			3	748569600	294912
	bn								754099200	
	relu								759628800	
	upsample	upsample2	180	240		2	2		781747200	
23	conv3-128	conv2_2_D			128			3	803865600	147456
	bn								825984000	
	relu								848102400	
24	conv3-64	conv2_1_D			64			3	859161600	73728
	bn								870220800	
	relu								881280000	
25	upsample	upsample1	360	480		2	2		925516800	
26	conv3-64	conv1_2_D			64			3	969753600	73728
	bn								1013990400	
	relu								1058227200	
27	conv3-64	conv1_1_D			12			3	1066521600	6912
28	argmax	argmax	360	480	1				<b>1067212800</b>	

Total Encoder Parameters: **14710464**  
 Total Decoder Parameters: **14752512**  
 Total Network Parameters: **29462976**  
 Total Memory Required for Data [Byte]: **1067212800**

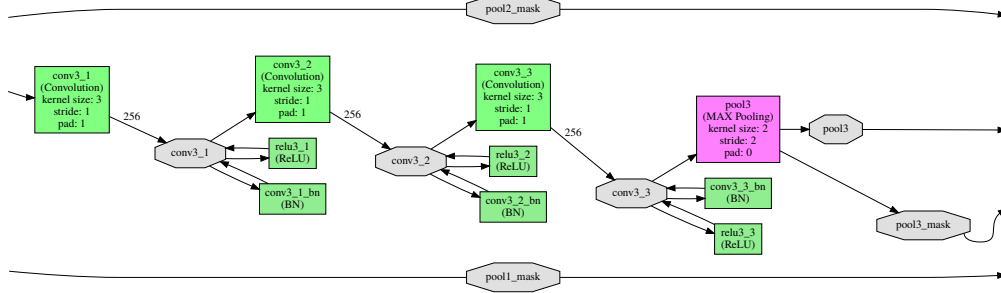
Table A.4: SegNet Layers Part 2 - Decoders And Classifier

Key elements of SegNet's architecture are shown as graph drawings in Figure A.1. Constrained by the page dimensions of this thesis, the size of SegNet's deep architecture prevents a useful depiction of the complete network. <PathToCaffe>/Python/draw\_net.py generates a png or svg file of the complete network. Drawings adapted to the color scheme in Table A.1, require according modification of <PathToCaffe>/Python/cafe/draw.py. Blobs are shown as gray octagons, layers as rectangles, colored depending on their type. A layer's depth or number of feature maps is depicted as number on the outbound edge.

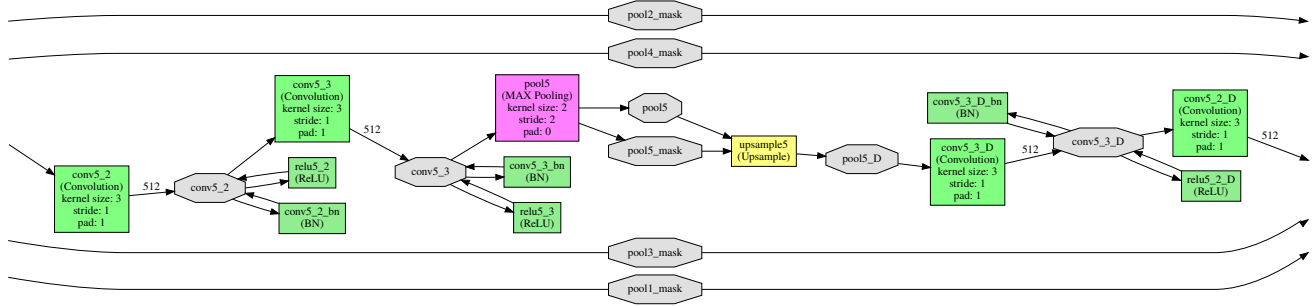




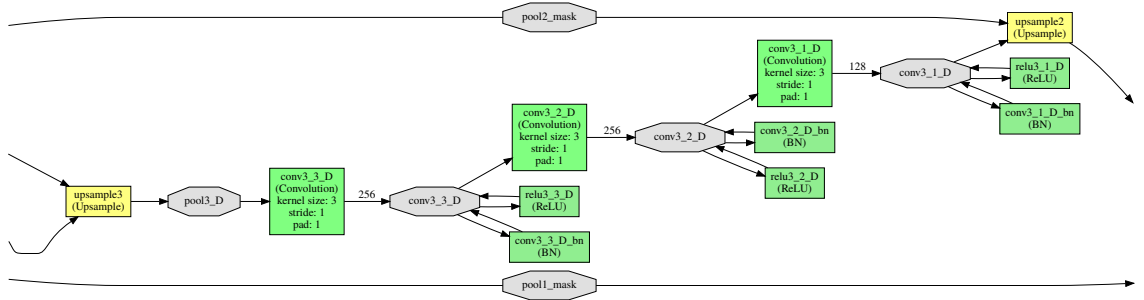
(a) SegNet Graph Drawing - Network's Input Layer And First Encoders



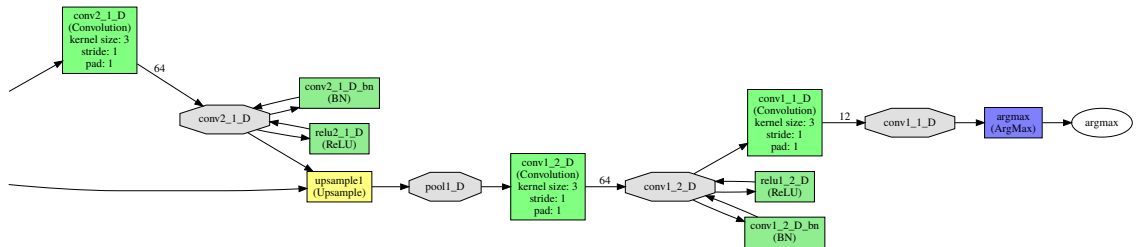
(b) SegNet Graph Drawing - Group Of Successive Encoders



(c) SegNet Graph Drawing - Encoder Stack To Decoder Stack



(d) SegNet Graph Drawing - Group Of Successive Decoders Corresponding To Group Of Encoders In (b)



(e) SegNet Graph Drawing - Last Decoders And Network's Output

Figure A.1: SegNet Graph Drawing - Key Elements Of The Network Architecture

## Appendix B

# KITTI Autonomous Driving Platform

The autonomous driving platform used to generate the KITTI Vision Benchmark Suite [29] is similar to this thesis' target platform MIG that is introduced in Section 2.2. It's also a Volkswagen Passat with a set of color cameras, a rooftop-mounted Velodyne HDL-64E LiDAR scanner and a GPS/IMU. The sensor configuration is depicted in Figure B.1 and the corresponding ROS TF frame tree in Figure B.2.

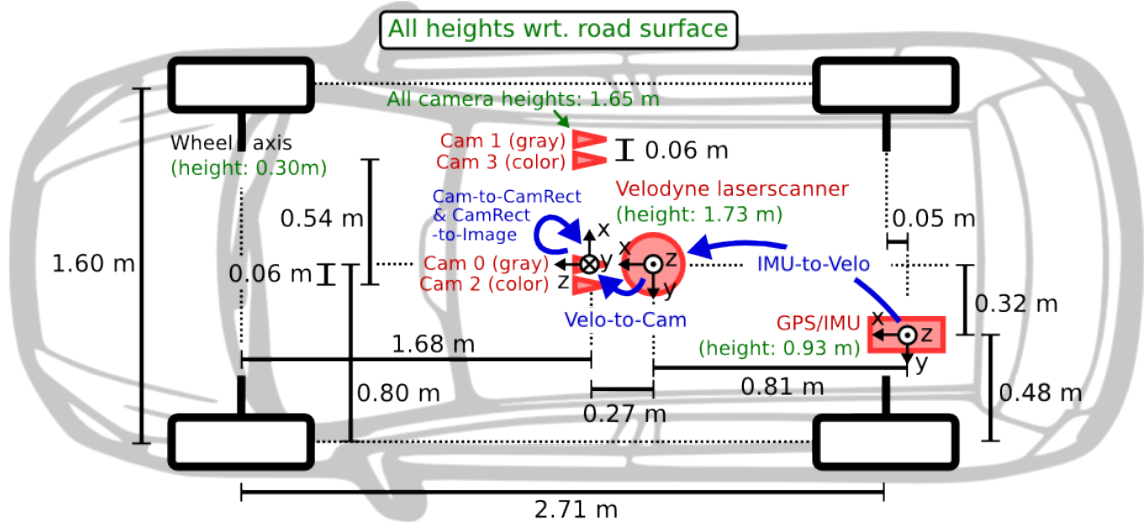


Figure B.1: KITTI Autonomous Vehicle - Sensor Setup [90]

The sensors directly addressed by the implemented nodes detailed in Chapter 3 are the left color camera *Cam 2* and the *Velodyne laserscanner* with ROS TF frame ids *camera\_color\_left* and *velo\_link*.

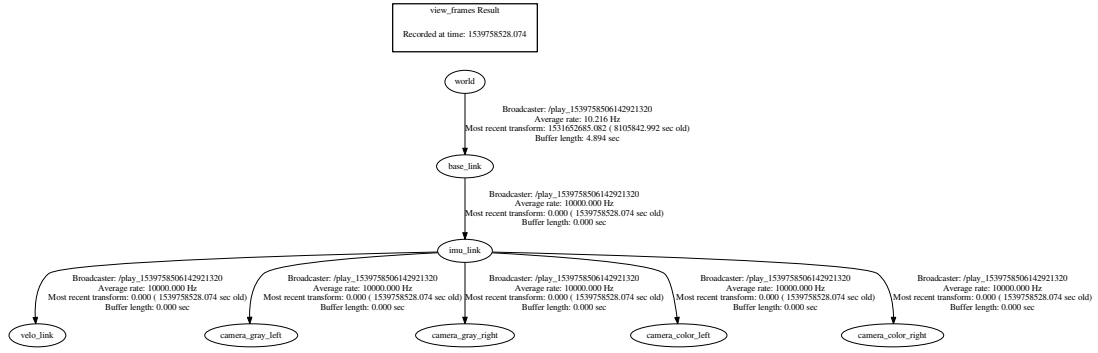


Figure B.2: KITTI Autonomous Vehicle - ROS TF Frame Tree

The Velodyne scanner triggers the cameras at 10 Hz when front facing. The color camera's characteristics are given in Table B.1, those for the Velodyne scanner in Table B.2.

Camera Feature	Characteristic
Imaging Model	FL2-14S3C-C [29]
Horizontal Field Of View (hFOV) [deg]	90 [29]
Vertical Field Of View (vFOV) [deg]	35 [29]
Spatial Output Resolution [px]	1240 x 376 [91]
Output Frame Rate [fps]	10
Output File Format	8-bit RGB JPG

Table B.1: KITTI Autonomous Vehicle - Left Color Camera Characteristics

LiDAR Scanner Feature	Characteristic
Scanner Model	Velodyne HDL-64E
Number of Lasers	64
Number of Detectors	64
Rotations per Minute [rpm]	600
Rotations per Second [hz]	10
Vertical Field of View (vFOV) [deg]	+2 to -24.8
Horizontal Field of View (hFOV) [deg]	360
Total Points per Revolution	133,333
Range [m]	up to 120

Table B.2: KITTI Autonomous Vehicle - Velodyne LiDAR Scanner Characteristics

Additional information on the KITTI autonomous vehicles configuration can be found at the KITTI Vision Benchmark Suite's website [cvlibs.net](http://cvlibs.net) under the setup tab or in the conference papers [29], [91].

# Bibliography

- [1] Google, “Google Trends Search Engine Report For Search Interest ‘Autonomous Driving’.” <https://trends.google.com/trends/explore?date=all&geo=US&q=autonomous%20driving>, Oct. 2018.
- [2] W. Bernhart, I. Olschewski, C. Burkard, and M. Yoon, “Automated Vehicles Index Q4/2017,” study, Roland Berger GmbH – Automotive Competence Center & fka Forschungsgesellschaft Kraftfahrwesen mbH, Aachen, Germany, 2017.
- [3] OECD International Transport Forum, “Automated and Autonomous Driving - Regulation Under Uncertainty,” Report, OECD International Transport Forum, Paris, France, May 2015.
- [4] SAE - Society of Automotive Engineers, “SURFACE VEHICLE RECOMMENDED PRACTICE - (R) Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - J3016-201609,” Sept. 2016.
- [5] N. Jaynes, “Timeline: The future of driverless cars, from Audi to Volvo.” <https://mashable.com/2016/08/26/autonomous-car-timeline-and-tech/>.
- [6] G. Rudolph and U. Voelzke, “Three Sensor Types Drive Autonomous Vehicles | Sensors Magazine,” *Sensors Online*, Nov. 2017.
- [7] Y. Horwitz, O. Zimmer, J. Yang, K. Yoshi, and M. Kekeisen, “Google Automotive Trends Report 2018,” tech. rep., Google, US, Germany, Japan, 2018.
- [8] Euro NCAP, “EUROPEAN NEW CAR ASSESSMENT PROGRAMME (Euro NCAP) - TEST PROTOCOL – Lane Support Systems Version 2.0.1 November 2017,” Nov. 2017.
- [9] Euro NCAP, “EUROPEAN NEW CAR ASSESSMENT PROGRAMME (Euro NCAP) - ASSESSMENT PROTOCOL – OVERALL RATING Version 7.0.1 March 2016,” Mar. 2016.
- [10] US DOT - NHTSA, “U.S. DEPARTMENT OF TRANSPORTATION - NATIONAL HIGHWAY TRAFFIC SAFETY ADMINISTRATION - LABORATORY TEST PROCEDURE FOR FMVSS 111 Rear Visibility,” Feb. 2018.
- [11] K. Heineke, P. Kampshoff, A. Mkrtchyan, and E. Shao, “Self-driving car technology: When will the robots hit the road? | McKinsey,” May 2017.
- [12] R. Marçal, F. Antonialli, B. Habib, and A. D. M. Neto, “AUTONOMOUS VEHICLES: Scientometric and bibliometric studies,” p. 21, Nov. 2017.
- [13] Google, “Engineering & Computer Science - Google Scholar Metrics.” [https://scholar.google.de/citations?view\\_op=top\\_venues&hl=en&vq=eng](https://scholar.google.de/citations?view_op=top_venues&hl=en&vq=eng).

- [14] Google, “IEEE Conference on Computer Vision and Pattern Recognition, CVPR - Google Scholar Metrics.” [https://scholar.google.de/citations?hl=en&vq=eng&view\\_op=list\\_hcore&venue=w44irn7CFc0J.2018](https://scholar.google.de/citations?hl=en&vq=eng&view_op=list_hcore&venue=w44irn7CFc0J.2018).
- [15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.
- [16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” *arXiv:1409.4842 [cs]*, Sept. 2014. arXiv: 1409.4842.
- [17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *arXiv:1311.2524 [cs]*, Nov. 2013. arXiv: 1311.2524.
- [18] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *arXiv:1411.4038 [cs]*, Nov. 2014. arXiv: 1411.4038.
- [19] V. Badrinarayanan, A. Handa, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling,” *arXiv:1505.07293 [cs]*, May 2015. arXiv: 1505.07293.
- [20] A. Widera, “awid\_semseg Git repository.” <https://git.imp.fu-berlin.de/awid/SemSeg>, Dec. 2018.
- [21] P. Bouchier, “ROS CppStyleGuide.” <http://wiki.ros.org/CppStyleGuide>, Mar. 2018.
- [22] G. van Rossum, B. Warsaw, and N. Coghlan, “PEP 8 – Style Guide for Python Code,” Aug. 2013.
- [23] D. van Heesch, “Doxygen Documentation.” <http://www.doxygen.org/>, May 2018.
- [24] Object Modeling Group - OMG, “The Unified Modeling Language Specification Version 2.5.1,” Dec. 2017.
- [25] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open-source Robot Operating System.” ICRA Workshop on Open Source Software, Volume 3, 2009.
- [26] V. Rabaud, “REP 144 – ROS Package Naming (ROS.org).” ROS Enhancement Proposal, Jan. 2015.
- [27] A. Kendall, “Implementation of SegNet: A Deep Convolutional Encoder-Decoder Architecture for Semantic Pixel-Wise Labelling.” <https://github.com/alexgkendall/caffe-segnet>, Nov. 2018. original-date: 2015-08-07T15:21:09Z.
- [28] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *arXiv:1511.00561 [cs]*, Nov. 2015. arXiv: 1511.00561.
- [29] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research (IJRR)*, vol. 32, no. 2013, pp. 1229–1235, 2013.
- [30] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” *arXiv:1604.01685 [cs]*, Apr. 2016. arXiv: 1604.01685.

- [31] IEEE - Computer Society, “IEEE Computer Society Style Guide,” Oct. 2014.
- [32] The University of Chicago, “The Chicago Manual of Style, 17th Edition,” 2017.
- [33] G. Autonomos, “Autonomos GmbH.” [https://www.autonomos-systems.de/wp-content/uploads/2015/08/pic\\_self-driving-car\\_neu.png](https://www.autonomos-systems.de/wp-content/uploads/2015/08/pic_self-driving-car_neu.png), 2015.
- [34] IEEE, “IEEE 802.3bw-2015 - IEEE Standard for Ethernet Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100base-T1),” 2015.
- [35] C. Kühling, “Fisheye camera system calibration for automotive applications,” Master’s thesis, FU Berlin, Berlin, May 2017.
- [36] OmniVision, “OmniVision OV10635 HD HDR Product Brief,” Dec. 2013.
- [37] Technica Engineering, “BroadR-Reach SatCAM User Manual,” Jan. 2015.
- [38] IEEE, “IEEE 1722-2016 - IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks,” 2016.
- [39] Velodyne LiDAR Inc., “HDL-64e S2 manual\_rev C\_2011,” Dec. 2012.
- [40] Velodyne LiDAR Inc., “86-0105 REV A OUTLINE DRAWING HDL-64e S2,” Apr. 2015.
- [41] M. Wang, *A Cognitive Navigation Approach for Autonomous Vehicles*. Berlin: Mensch & Buch, 2012.
- [42] W. Newman, *A Systematic Approach to Learning Robot Programming with ROS*. Boca Raton: Chapman and Hall/CRC, 1 edition ed., Sept. 2017.
- [43] T. Foote and D. Thomas, “REP 128 – Naming Conventions for Catkin Based Workspaces.” ROS Enhancement Proposal, Mar. 2014.
- [44] D. Thomas and J. O’Quin, “REP 140 – Package Manifest Format Two Specification.” ROS Enhancement Proposal, May 2017.
- [45] T. Foote, “tf: The transform library,” in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pp. 1–6, April 2013.
- [46] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. Sebastopol: O’Reilly Media, 1 edition ed., Dec. 2015.
- [47] J. M. O’Kane, *A Gentle Introduction to ROS*. Leipzig: CreateSpace Independent Publishing Platform, Oct. 2013.
- [48] A. Mahtani, L. Sanchez, E. Fernandez, and A. Martinez, *Effective Robotics Programming with ROS - Third Edition*. Packt Publishing - ebooks Account, 3rd revised edition edition ed., Dec. 2016.
- [49] R. C. Gonzalez and R. E. Woods, *Digital Image Processing: International Edition*. Upper Saddle River, NJ: Pearson, international edition edition ed., Jan. 2009.
- [50] A. Butterfield, G. E. Ngondi, and A. Kerr, eds., *A Dictionary of Computer Science*. Oxford, United Kingdom ; New York, NY, United States of America: Oxford University Press, 7 edition ed., Apr. 2016.

- [51] R. Rojas and J. Feldman, *Neural Networks: A Systematic Introduction*. Berlin ; New York: Springer, 1 edition ed., Jan. 1996.
- [52] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River: Pearson, 3 edition ed., Dec. 2009.
- [53] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, Massachusetts: The MIT Press, Nov. 2016.
- [54] A. Karpathy, “Stanford University - CS231n Convolutional Neural Networks for Visual Recognition - Lecture 5 - ConvNet Notes.” <http://cs231n.github.io/convolutional-networks/>, Apr. 2017.
- [55] N. Buduma and N. Locascio, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. Beijing Boston Farnham Sebastopol Tokyo: O’Reilly Media, 1 edition ed., June 2017.
- [56] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, May 2015.
- [57] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec. 1989.
- [58] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [59] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, Oct. 1986.
- [60] F.-F. Li, A. Karpathy, and J. Johnson, “Stanford University - CS231n: Convolutional Neural Networks for Visual Recognition.” <http://cs231n.stanford.edu/>, June 2017.
- [61] A. Karpathy, “Stanford University - CS231n Convolutional Neural Networks for Visual Recognition - Lecture 6 - NeuralNets\_notes3.” <http://cs231n.github.io/neural-networks-3/>, Apr. 2017.
- [62] A. Karpathy, “Stanford University - CS231n Convolutional Neural Networks for Visual Recognition - Lecture 2 - Image Classification Notes.” <http://cs231n.github.io/classification/>, Apr. 2017.
- [63] A. Karpathy, “Stanford University - CS231n Convolutional Neural Networks for Visual Recognition - Lecture 5 - Convolution Demo.” <http://cs231n.github.io/assets/conv-demo/index.html>, Apr. 2017.
- [64] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, UK ; New York: Cambridge University Press, 2 edition ed., Apr. 2004.
- [65] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Upper Saddle River, N.J: Pearson, 3 edition ed., Aug. 2004.
- [66] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Sept. 2014. arXiv: 1409.1556.
- [67] G. J. Brostow, J. Fauqueur, and R. Cipolla, “Semantic object classes in video: A high-definition ground truth database,” *Pattern Recognition Letters*, vol. 30, pp. 88–97, Jan. 2009.



- [68] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature Embedding,” pp. 675–678, ACM Press, 2014.
- [69] T. Sämann, “Fork of Alex Kendall’s caffe-segnet running with cuDNN5.” <https://github.com/{TimoSaemann}/caffe-segnet-cudnn5>, Oct. 2018. original-date: 2016-12-14T10:33:20Z.
- [70] Nvidia, “CUDA Zone.” <https://developer.nvidia.com/cuda-zone>, July 2017.
- [71] Nvidia, “NVIDIA cuDNN.” <https://developer.nvidia.com/cudnn>, Sept. 2014.
- [72] W. Woodall, “REP 136 – Releasing Third Party, Non catkin Packages.” ROS Enhancement Proposal, Mar. 2013.
- [73] T. Foote and K. Conley, “REP 3 – Target Platforms.” ROS Enhancement Proposal, Sept. 2010.
- [74] G. Bradski, “Open Source Computer Vision Library,” *Dr. Dobb’s Journal of Software Tools*, Jan. 2008. original-date: 2012-07-19T09:40:17Z.
- [75] R. B. Rusu and S. Cousins, “3d is here: Point Cloud Library (PCL),” in *2011 IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 1–4, IEEE, May 2011.
- [76] Scratchapixel, “Computing the Pixel Coordinates of a 3d Point (Mathematics of Computing the 2d Coordinates of a 3d Point).” <https://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point/mathematics-computing-2d-coordinates-of-3d-points>, Nov. 2014.
- [77] A. Geiger, P. Lenz, and R. Urtasun, “The KITTI Vision Benchmark Suite.” [http://www.cvlibs.net/datasets/kitti/raw\\_data.php](http://www.cvlibs.net/datasets/kitti/raw_data.php).
- [78] H. Abu Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, “Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes,” *International Journal of Computer Vision*, vol. 126, pp. 961–972, Sept. 2018.
- [79] M. Cordts, “Cityscapes Dataset - Label and Metadata Information - labels.py.” <https://github.com/mcordts/cityscapesScripts>, Oct. 2018. original-date: 2016-02-20T13:00:11Z.
- [80] T. Krejci, “Convert KITTI dataset to ROS bag file the easy way.” <https://github.com/tomas789/kitti2bag>, Oct. 2018. original-date: 2016-11-28T11:20:56Z.
- [81] K. Okada, “Profiling ROS Nodes.” <http://wiki.ros.org/roslaunch/Tutorials/Profiling%20roslaunch%20nodes>, Aug. 2018.
- [82] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes Challenge: A Retrospective,” *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.
- [83] A. Mkhitarian, “Pixel-wise Semantic Segmentation for Low-power Devices.” Bachelor thesis, FU Berlin, Berlin, 2017, June 2017.
- [84] Bundesanstalt für Straßenwesen - BASt, “Liste der amtlichen Bezeichnungen Gefahrzeichen nach Anlage 1 (zu § 40 Absatz 6 und 7 StVO),” Apr. 2017.

- [85] T. Samann, “Script to merge SegNet layers.” <https://github.com/TimoSaemann/caffe-segnet-cudnn5/blob/master/scripts/BN-absorber.py>, Nov. 2018. original-date: 2016-12-14T10:33:20Z.
- [86] Nvidia Developer Blog, “Fast INT8 Inference for Autonomous Vehicles with TensorRT 3.” <https://devblogs.nvidia.com/int8-inference-autonomous-vehicles-tensorrt/>, Dec. 2017.
- [87] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, “ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation,” *arXiv:1606.02147 [cs]*, June 2016. arXiv: 1606.02147.
- [88] B. Wu, A. Wan, X. Yue, and K. Keutzer, “SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3d LiDAR Point Cloud,” *arXiv:1710.07368 [cs]*, Oct. 2017. arXiv: 1710.07368.
- [89] A. Kendall, “Prototxt model driving SegNet webdemo.” [https://github.com/alexgkendall/SegNet-Tutorial/blob/master/Example\\_Models/segnet\\_model\\_driving\\_webdemo.prototxt](https://github.com/alexgkendall/SegNet-Tutorial/blob/master/Example_Models/segnet_model_driving_webdemo.prototxt), Jan. 2016.
- [90] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “The KITTI Vision Benchmark Suite - Sensor Setup.” <http://www.cvlibs.net/datasets/kitti/setup.php>, 2018.
- [91] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, (Providence, RI), pp. 3354–3361, IEEE, June 2012.