

TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Software Science

Jubril Gbolahan Adigun

182486IVSM

**Ground Truth Data for Object Detection in Autonomous
Vehicle from a Driving Simulator**

Master's thesis

Supervisor: Priit Järv, PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia Teaduskond

Tarkvarateaduse Osakond

Jubril Gbolahan Adigun

182486IVSM

**Autonoomsete sõidukite objektivastuse
valideerimisandmete loomine sõidusimulaatoriga**

Magistritöö

Jubril Gbolahan Adigun

182486IVSM

Juhendaja: Priit Järv, PhD

Tallinn 2020

Declaration

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis submitted for master's degree at Tallinn University of Technology has not been presented for examination anywhere else.

Author: Jubril Gbolahan Adigun

04.08.2020

Acknowledgement

It is my greatest pleasure to offer salutations to my supervisor, Priit Järv (PhD). He was extremely helpful and supportive all through the course of my thesis. I would not have been able to complete the thesis without his guidance and mentorship.

I will also appreciate my program coordinator at Tallinn University of Technology, Associate Professor Juhan-Peep Ernits who was always the go to person for all study related concerns and showed genuine interest in seeing his students grow. I especially appreciate his help with making the simulator used in this thesis, available.

I wish to also appreciate Professor Jüri Vain (Tallinn University of Technology) for facilitating the platform for me to attend the Summer School on the Development of High-Assured Autonomous Systems at Åbo Akademi University (Åbo Akademi) in Finland.

At the same degree, I would like to thank the University of Tartu, the IT Academy and Institute of Computer Science, University of Tartu and the Department of Software Science of Tallinn University of Technology as well as the Estonian Government for the different scholarships and support rendered during the course of my studies.

At the home front, I express my profound gratitude to my family members; my parents and my siblings (The Adiguns), my wife (Dr. Amina Farayola), my relatives, and friends, particularly Tek Raj Chhetri (whose insights were very helpful) for supporting me with their kind words, encouragement and continuous checkups. Above all, I return all praises and salutations to Allah, the Most Beneficent, the Most Merciful who granted me the wherewithal to accomplish this feat.

Table of Contents

Declaration.....	iii
Acknowledgement.....	iv
Table of Contents.....	v
Abstract.....	vii
Annotatsioon.....	viii
List of Figures.....	ix
List of Tables.....	ix
List of Acronyms.....	x
1 INTRODUCTION.....	1
1.1 Overview.....	1
1.2 The development of Autonomous Vehicles.....	1
1.3 Autonomous Vehicle Architecture.....	2
1.4 Motivation.....	3
1.5 Thesis goal.....	6
1.6 Thesis Outline.....	6
2 BACKGROUND AND RELATED WORKS.....	7
2.1 Related works.....	7
2.2 Relevant theory and Concepts.....	11
2.2.1 Ground truth data.....	11
2.2.2 Object detection.....	12
2.2.3 Object detection metrics.....	14
3 METHODOLOGY.....	17
3.1 Architecture.....	17
3.2 Technical platform and tools.....	18
3.2.1 BeamNG.reseacrh.....	18
3.2.2 BeamNGPy.....	21
3.2.3 ROS.....	23
3.2.4 Autoware.....	24
3.3 System requirements.....	25
3.4 Summary.....	26
4 EXPERIMENT.....	27

4.1	Ground truth data from BeamNG.research	30
4.2	Images and LIDAR points.....	31
4.2.1	Images	31
4.2.2	Point cloud	32
4.3	Visualization of data in RViz	32
4.4	ROSBAG conversion	33
4.5	Summary	34
5	VALIDATION	35
5.1	Validating with Object Detection.....	35
5.2	Performance Measures (Precision and Recall).....	37
5.3	Summary	38
6	FUTURE WORK	39
7	CONCLUSION	41
	References	43
	Appendix	47

Abstract

Automotive technology is seeing tremendous application in the emerging field of autonomous vehicles (AV). With the prevalence of autonomous vehicles in our world today, driven by advances in computer vision algorithms on large amount of data, it is important to look for more efficient ways for development working models with high level of reliability. For us to achieve better and accurate development of algorithms, we need to get results that reflect ground truth information about pose of objects around the vehicle. Although, low-level, object detection and tracking are crucial for the buildup of many higher level functionalities of AVs. This has proven to be challenging because, it requires laborious accurate sensors' calibration, annotations of images and semantic segmentation of image pixels.

In this thesis, we adopt an open approach to explore the generation of ground truth data that would be useful for object detection using a driving simulator called beamng.research (beamng). This is important because sensors for generating training data for AVs and other autonomous systems can be expensive and many of the stack in autonomous vehicles are proprietary. We introduce a method of creating the ground truth data with a simulator. The simulator gives the true location of objects in the simulation and sensor data streams. The proposed method, can help us address the problem of annotation of ground truth data that is otherwise, difficult with real life data. We made a prototype that can generate annotated images and point cloud data. The prototype allows generation of training/validation data in an unlimited amount with the existing beamng scenarios. We generated 200 frames of images and 200 sets of point cloud data. We tested the data in practice by using it to validate YOLO v3 object detection running on ROS platform.

Keywords: Autonomous vehicle, computer vision, object detection, simulation/simulator, synthetic data, ground truth data, artificial intelligence, beamng

CERCS P170/P176: Computer science, numerical analysis, systems, control/ Artificial Intelligence

This thesis is written in English, contains seven (7) Chapters across 60 pages, including 14 figures and 9 tables.

Annotatsioon

Autonoomsete sõidukite objektituvastuse valideerimisandmete loomine sõidusimulaatoriga

Autonoomsed sõidukid (AS) on kiirelt laienev valdkond ja nende edukus sõltub tehisenägemise algoritmidest. Seetõttu on oluline leida efektiivsemaid viise kõrge usaldusväärsusega mudelite välja töötamiseks. Parema ja täpsema algoritmide arendustöö jaoks on vaja andmeid, mis kajastavad tuvastatavate objektide tegelikke asukohti antud sensorandmete korral. See on osutunud väljakutseks sensorite kalibratsiooni, piltide märgendamise ja nende semantilise segmenteerimise töömahukuse tõttu.

Lõputöös kasutame avatud platvormidel põhinevat lähenemist valideerimisandmete loomisele kasutades simulaatorit beamng.research (beamng). Avatud lähenemine on oluline, kuna sageli on AS-ide platvormid suletud ja sensorite rakendamine kulukas. Simulaator annab meile nii sensorite andmevood kui ka objektide tõelised asukohad simulatsioonis oleva sõiduki suhtes. Väljapakutud meetod võimaldab lahendada valideerimise jaoks vajaliku märgendamise probleemi, mis on päriseluliste andmete puhul keeruline. Koostasime prototüübi, mis väljastab märgendatud pildifaile ja punktipilvi. Prototüüp võimaldab piiramatus koguses märgendatud andmete genereerimist olemasolevate beamng stsenaariumite piires. Prototüübiga genereeriti 200 pildikaadrit ja 200 punktipilve. Andmete praktilise kasutuse testimiseks valideeriti nendega YOLO v3 objektituvastuse algoritmi ROS platvormil.

Märksõnad: Autonoomne sõiduk, arvuti nägemine, objektide tuvastamine, simulatsioon / simulaator, sünteetilised andmed, maapealsed tõeandmed, tehisintellekt, beamng.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 60 leheküljel, 7 peatükki, sealhulgas 14 joonist ja 9 tabelit.

List of Figures

Figure 1: Levels of Autonomy in Automobile.....	2
Figure 2: Basic autonomous vehicle model.....	3
Figure 3: Iseauto 3D view	4
Figure 4: Illustration of Ground Truth Data generation from an input image and the output of from a prediction model.	12
Figure 5: Illustration of Convolution network for object detection	13
Figure 6: Architecture.....	18
Figure 7: BeamNG environment	19
Figure 8: BeamNGpy usage	22
Figure 9: ROS basic concept.....	24
Figure 10: Minimap of west_coast_usa scenario.....	27
Figure 11: Different spawn points available in the west_coast_usa scenario. 1 L-R: Chinatown, Dirttrack; 2 L-R Dockyard, Dragstrp; 3 L-R Highway, Industrial; 4 L-R Racetrack, Redwood_forest.....	29
Figure 12: Snapshot of file folder structure used in the project.....	31
Figure 13: Sample image captured visualized in Rviz.....	32
Figure 14: Sample point cloud data visualized in RViz	33

List of Tables

Table 1: ISEAUTO Design Specification.....	5
Table 2: Darknet-53 [23]	14
Table 3: BeamNG Requirements	20
Table 4: Requirements for BeamNGpy	21
Table 5: System requirements.....	25
Table 6: Specification of sensor parameters	28
Table 7: Summary of outputs and their purpose	34
Table 8: Table showing captured color image, annotated image of ground truth and object detection output.....	36
Table 9: TP, TN, FP, FN Derived Measures.....	37

List of Acronyms

AV	Autonomous Vehicle
CG	Computer Graphics
CV	Computer Vision
DIL	Driver-in-the-Loop
Fast-RCNN	Fast Region-based Convolutional Neural Network
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
LiDAR	Light Detection and Ranging
NN	Neural Network
OS	Operating System
ROS	Robot Operating System
SSD	Single Shot Detector
VM	Virtual Machine
YOLO	You only Look Once
3D	Three dimension
COVNET	Convolutional Network
WIP	Work-in-Progress
MIME	Multipurpose Internet Mail Extensions
PNG	Portable Network Graphics
JPEG	Joint Photographic Experts Group
BMP	Bitmap (Bitmap Image File)
TIFF	Tagged Image File Format
GIF	Graphics Interchange Format
GPU	Graphics Processing Unit
RTK	Real-time kinematic
fps	Frames per second
rps	rays per second

1 INTRODUCTION

1.1 Overview

The automobile industry has grown over more than a century and the spate of development in recent times has been unprecedented. The evolution has progressed through many stages of trying to reduce human effort put into commuting from one place to another. In the world today, we have car manufacturers in a stiff competition to eliminate human involvement in the driving tasks, with the rise and popularity of autonomous vehicles (AV/AVs). Although, we are yet to attain full autonomy in all sense of autonomy, significant success has been recorded in autonomous systems.

1.2 The development of Autonomous Vehicles

More than ever before, we can already see the transformation that has happened in the automobile industry. In the wake of the sleekest environmentally friendly newer generation cars, is the need to increase human efficiency and productivity by reducing human effort put into commuting while ensuring the safest mobility of these vehicles and their passengers alike [1, 2].

Automobiles are now increasingly moving from the old age fuel-based engines to electrically powered designs with industry leaders putting aggressive effort into not just manufacturing electric cars but cars that can also drive themselves (autonomous driving). Following from the above, we shall look at the various levels of automation that we have in the automotive space. **Figure 1** below, shows an overview of the different levels of autonomy that we have. This will help to better assimilate the need for further development of autonomous vehicles in perspective [3]

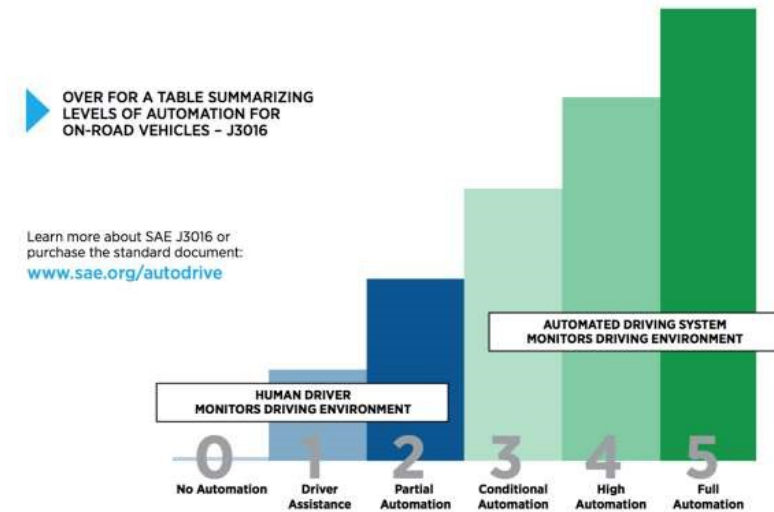


Figure 1: Levels of Autonomy in Automobile.¹

Autonomy in robotics and vehicular systems are emerging in numerous areas of endeavor so much so that during the coming years, it is postulated that cars that have achieved full automation will be driving our streets, untarred roads, highways, metropolises, cities and with no human intervention [4].

1.3 Autonomous Vehicle Architecture

Autonomous systems and by extension, AVs require complex computations and decisions. More so, in AVs, to ensure safety of both passengers and surrounding traffic, advanced perception systems are employed to sense the environment, carry out scene assimilation and for detection, mapping and obstacle tracking along and near the path of the vehicle [5].

From **Figure 2** above, we see that there are 3 main modules (Sensing, Computing and Actuation) with which AVs operate. In this thesis, we are more concerned about the first section and a part of the second. The second contains way much more details which we are not able to deal with in one project task.

¹ <https://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11automakers/>

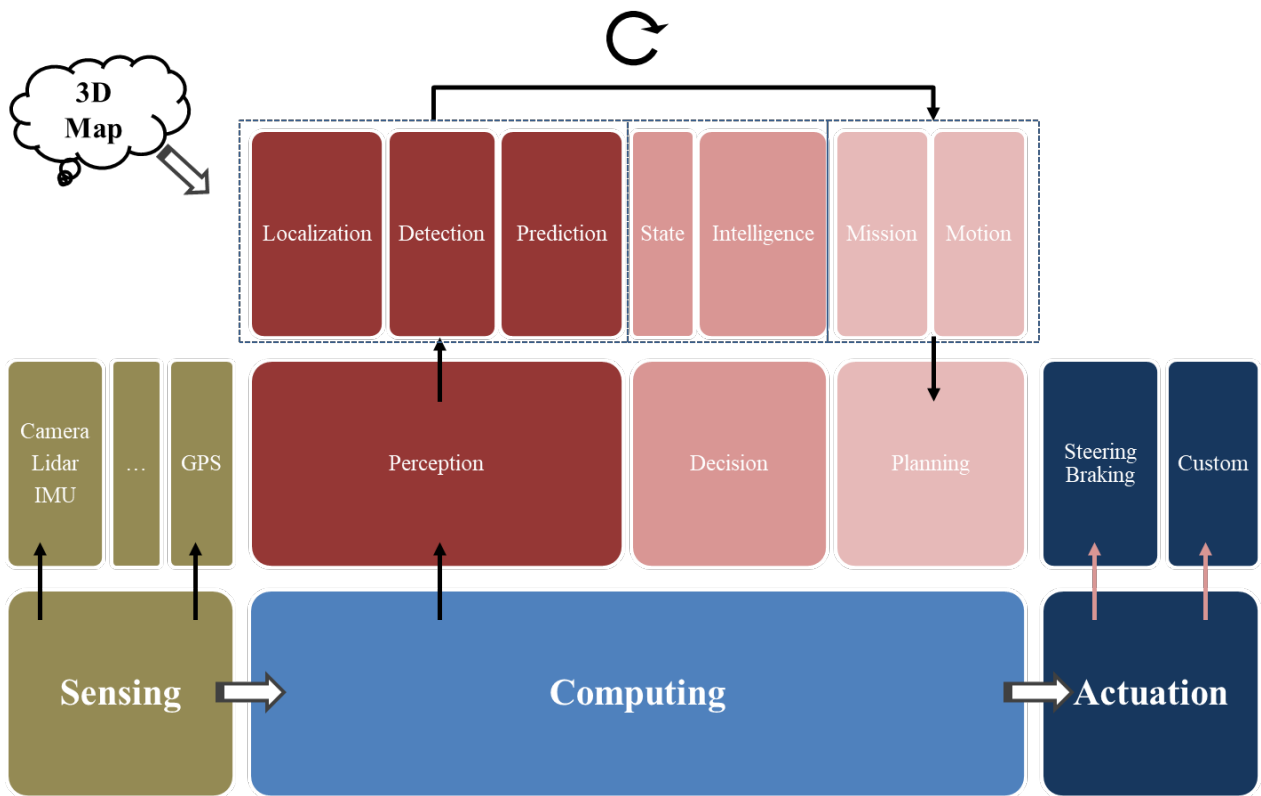


Figure 2: Basic autonomous vehicle model

1.4 Motivation

As we aim to achieve full autonomy in cars today, and object detection being a pivotal process in this technology, it is essential that we get this aspect of the autonomous systems and by extension, autonomous vehicles right. Popular autonomous vehicle manufacturing companies today, still have to contend with failures, accidents and huge fallouts every now and then associated with object/obstacle detection flaws.

Again, progress being made in the computer vision field has been steered by high-capacity models trained on huge datasets. Creation of large datasets unfortunately, are very costly due to the amount of human effort required [6]. More so, many of the existing datasets for algorithm evaluation for object detection and other computer vision tasks contain only visual imagery with no independent measurement of their locations and trajectories [7].

According to [8], it is mentioned that the key to success in design for modern automobiles is early involvement of real drivers in virtual vehicles, sub-systems and environments/scenarios via Driver-in-the-Loop (DIL) simulations. In the real sense, this is a way of assessing ground truth about performance of such automobiles against real life. If this is possible, we aim to adopt same strategies and such systems in obtaining ground truth data for autonomous purposes.

In this project, we would explore the engineering problem of obtaining ground truth data for computer vision objection detection purposes in self- driving context using a driving simulator. The ISEAUTO project is a practical example of where the result of this study can be useful. Iseauto, as the name implies in Estonian, is a pioneer self-driving car project developed by Tallinn University of Technology (TTÜ) in collaboration with Silberauto AS, in Estonia [9].



Figure 3: Iseauto 3D view²

The car has a speed limit of 10-20km/h because it is designed to operate as last-mile means of transportation, availing alternative mobility in closed and diverse traffic territories within smart and safe limits. With a vision of building interesting research projects for future innovations, the university aims to develop competence in the

² <https://iseauto.taltech.ee/en/tehniline/>

autonomous field and provide interesting projects for the engineering students. As at the time of this publication, the car runs Ubuntu version 16.04 operating system and uses the following sensors during driving for perception, localization, motion planning and navigation [10]:

Table 1: ISEAUTO Design Specification²

Technical parameters	Dimensions	Sensors	Software Stack
<ul style="list-style-type: none"> • Up to 6 passenger/cargo places • Cruising speed range - 10-20 km/h • Turning circle - 9 m • Main motor power - 47 kW • Battery 16 kWh 	<ul style="list-style-type: none"> • Height - 2400 mm • Length - 3500 mm • Width - 1500 mm 	<ul style="list-style-type: none"> • LiDAR Velodyne VLP-16 x2 • 8 Cameras • 16 Ultrasonic proximity sensors • A Near-ranged radar • An IMU sensor • An RTK-GNSS 	<ul style="list-style-type: none"> • Robot Operating System (ROS) • Autoware • Yolo

It is important to mention that driving simulators are now being used as an essential player in the development, testing and validation of AVs. There is a continuous exploration of different simulation platforms, some of which are mentioned in 2.1, in the development of AVs because they have proven to provide flexibility in terms of sensor and environmental conditions. The main area of study is the generation of ground truth data from simulated sources. There has been some work in this field but not very many when compared to instances where datasets are generated from real life environments. While perception algorithms may have their own drawbacks from data to data and from model to model. there's an obvious concern about the quality about quality of synthesized data. As such, there is a continuous need to explore different options and alternatives as we strive to improve accuracy and confidence levels in AV development.

1.5 Thesis goal

Enormous efforts have gone into determining and establishing ground truth data for autonomous vehicles. On another hand, for many reasons more than object detection and tracking, engineers have used driving simulators to get objective measurements of what real life may look like [8]. This was posited as well in [11]. In the real world, object detection proves difficult, because, objects and thence, their images are affected by many constraints such as illumination, orientation, scale and occlusion [12].

In this thesis, however, we aim to develop synthetic data (LIDAR and camera) suitable for evaluation of object detection algorithms. We introduce a method obtaining ground truth data from a vehicular software platform called beamng.research³ (which provides enough flexibility for altering the behavior of simulated vehicles thereby, resulting in fast development iteration times).

1.6 Thesis Outline

In this chapter, we had a brief introduction about autonomous vehicles, the importance of data in its development and how we can adopt synthesized data as a means for carrying out development of AVs. In **Chapter 2**, we discuss some works that have been done on generating benchmarks for different computer vision assignments. In the same vein, we will look at research works using simulated or synthesized data for similar purposes. In **Chapter 3**, we document the different tools and mention specifications of systems used for our project. Additionally, in **Chapter 4** we describe how the experiment was carried out. We share our methodology for validating our results in **Chapter 5**. We mention ways by which the work presented in this thesis can be improved in **Chapter 6**. In **Chapter 7**, we do a wrap up in the conclusion.

³ <https://beamng.gmbh/research/>

2 BACKGROUND AND RELATED WORKS

Enormous efforts have gone into determining and establishing ground truth data for autonomous vehicles. Also, for many reasons more than object detection and tracking, engineers have used driving simulators to get objective measurements of what real life may look like [8]. This is further strengthened by the comparative studies done by Nikolenko in [11]. In the real world, object detection proves difficult, because, objects and thence, their images are affected by many constraints such as illumination, orientation, scale and occlusion [12].

2.1 Related works

As opposed the simulator used in our thesis work, which is a more recently developed video game engine, [13] introduced a novel simulator called CARLA for use in urban driving simulation. The paper studied the performance of three ways of achieving autonomous driving – end-to-end reinforcement learning trained model, end-to-end imitation learning trained model, and the classical modular pipeline. On another end, Nikolenko [11] presents an underlying basis for our thesis work. He attempted to do a comprehensive analysis of the different directions that synthetic data can be developed and applied. The survey discusses synthetic datasets for basic CV problems and ways of improving and producing synthetic data, moving from synthetic-to-real domain and privacy concerns.

While object detection models have their own limitations in use for real time autonomous vehicles, the whole process of getting precisely-labeled datasets which are favored more to support these models prove a tedious task on its own. They are expensive, time-consuming and may also require both highly technical skills and physical inputs to gather the data. For instance, at the University of Waterloo, they use the Global Navigation Satellite System (GNSS) and the Inertial Navigation System (INS) post-processing method to augment different weather condition data collected over thousands of kilometers. Their goal was to use the data to create a driving dataset for a new road that accounts for omni-weather autonomy challenges (e.g. 3D object detection and tracking, precision in localization and mapping to semantic segmentation

of the driving environment using omnidirectional vision and light detection and ranging (LiDAR) data) [14].

In the same vein, there exists the widely used KITTI dataset, which is hugely used for academic and research purposes in autonomous driving. Here, the team which comprises individuals from Karlsruhe Institute of Technology (KIT), Max Planck Institute for Intelligent Systems (MPI Tübingen) and University of Toronto used the Annieway autonomous driving platform to develop new real-world computer vision benchmarks for a vast range of vision related tasks including 3D object detection and 3D tracking. To achieve this, they attached to a station wagon, cameras, LiDAR and GPS systems and drove through rural areas and highways of the mid-sized city of Karlsruhe [15].

There is a growing number of research being done every day to improve existing datasets used for autonomous driving. In a research titled “*nuScenes: A multimodal dataset for autonomous driving*”, the authors pioneered the publishing of a dataset that is obtained from the full AV sensor suite: this includes cameras (6), radars (5) and LiDAR (1), with full 360-degree field of view. The dataset consists of a thousand 20s long scenes, bounding boxes for 23 classes and 8 features and has 7 and 100 times as many annotations and images as the groundbreaking KITTI dataset respectively [16]. The experiment involved well thought out driving plans, vehicular setups, careful selection of interesting scenes, and dataset annotations. Furthermore, the driving routes were carefully chosen in a bid to capture a diverse range of settings, times and even weather conditions.

There is no doubt that AVs have attracted considerable amount of attention in the past few years from the academic, commercial and general public entities. The reason for this can be linked to the expectations of their societal benefits. The expectations are that safety, mobility, and the environment would be tremendously affected and these have captivated the interests of people all over the world. The key word here, is safety particularly in light of recent accidents attributed to AVs. It has become evident that we have a long way to go in order to meet the high standards and expectations associated with AV [2].

Now, there’s the argument that an AV has to be test-driven many millions of miles in difficult conditions to be able to demonstrate that fatalities and injuries are reduction to a high degree of statistical reliability. Even with this, it could takes take several years of

road tests under the most intensive evaluation strategies to achieve the desired reliability. A possible solution that has been proposed, is to use of simulation systems, which are already common in other sectors like manufacturing, medical training, law enforcement and military. Simulations would help us to test and validate the capability of AVs in the areas of environmental perception, control and navigation and also being able to generate large amount of training data to train machine learning/computer vision algorithms, such as a deep neural network (NN). The latter has more recently been adopted in computer vision.

Following from the above, a common practice to generate simulated data is to come up with combinations of computer graphics (CG) models and robot motion planning methodologies to create synthesized environments wherein vehicles and other players can be rendered and animated. Based on this technique, a number of simulators have been developed by tech giants and a number of key players. Examples include as Intel's CARLA, Microsoft's AirSim, NVIDIA's Drive Constellation, Google/Waymo's CarCraft. They have all been proven to achieve qualitative rendering results. In an article published on WARDSAUTO by Danny Atsmon, the author presents several arguments to support the use of simulated technology for AVs. The author's line of thought follows the underlying demerits to favor simulated technology: software stack for telling the location of a car with respect to the world alone is a serious bone of contention [17].

Furthermore, Atsmon iterates that the ability of the car to make decisions on the go within the testing environment is questionable. Also, accuracy in real world tests is undermined because they do not include ground truth while the controllability of data more consistently and closely, will inevitably be of higher quality. On the angle of economies of scale, AV manufacturers usually have to build test models manually and single at a time, which keeps production costs on the high side. This also requires that during a test run, a real driver, engineers, supervisors and safety coordinators may have to be involved including securing permission from every region involved. As laws may vary from one municipality to another, local regulations need to be engulfed and terrain learned. Insurance is also expensive in this regard because of the high risk nature of autonomous systems. As the use of simulators is a relatively new endeavor in AVs, below, we shall see a snapshot of where simulation technologies have been employed to support AV dataset.

Based on an Advanced Driver Assistance Systems (AADS) manufacturer's demand, CVEDIA's SynCity simulator was able to enable an environment that included custom objects, such as city infrastructure and humans, and extreme weather conditions that could be manipulated via its interface and API. The said manufacturer was having difficulties with identification and classification of close-up objects also with creating semantic segmentation of LiDAR. With the simulator, it was possible to generate synthetic segmented LiDAR data within SynCity [18]. To scale up these kinds of synthetic data, [19] designed an approach called the AADS. They use augmented real-world images with a simulated traffic flow to create photo-realistic synthesized images and renderings using Lidar and cameras as sensors. These renderings are then used to generate some realistic traffic flows for cars and pedestrians in readiness for end-to-end autonomous driving training and testing.

While it is clear that it will be impossible to capture every environmental and weather condition in which real life use of autonomous driving may happen, it is glaring from the above, the amount of physical and intellectual efforts that need to be put into establishing reliable ground truth dataset or benchmarks. Interestingly, as driving simulators for games and experimental purposes are becoming widely popular, they may offer some opportunities for discarding this laborious and somewhat inefficient and slightly unreliable way of gathering ground truth data for autonomous vehicles. This is important for us because, we need to drive efficiency just as much as we aim to achieve the goal of safe autonomous driving. Yet, challenges that current state-of-the-art methodologies pose are that the simulation engines are either not open-source or simply lack features that help us model closely real life environment and scenarios as we want and are reliant on game engines or high-fidelity computer graphics (CG) models.

In our work, we shall make use of a combination of sensors, particularly LiDAR and cameras to generate high-fidelity images and annotations within our simulator just as [16] obtained better dataset generating a multimodal dataset using a combination of sensors but in a real world driving context. Our work may set a good precedence to explore multimodal dataset generation and eventually place simulated technology at the core for AV development and testing.

To evaluate our results, we deployed an object detection model to validate our results. In the context of TTÜ Iseauto project, [20] developed an evaluation model to analyze

the performance of three (3) object detection models - Single-Shot Detector (SSD), Fast Region-based Convolutional Neural Network (Fast-RCNN) and You Only Look Once v2 (YOLO v2). Expectedly, a self-generated dataset containing relevant data was used. He mentioned that, “annotating ground truth data for object detection is relatively expensive task”. The outcome of his work is that, although YOLO v2 seemed to have performed best, object detection reliability required for safety-critical applications was not attained by any of the object detectors. The author postulated the use of YOLO v3 for better performance.

2.2 Relevant theory and Concepts

2.2.1 Ground truth data

Ground truth data helps us with data that has relevant and related features and objects on the ground. With ground truth data, we can perform evaluation of different computer vision algorithms. Just as ground truth data collection using data from calibrated state-of-the-art sensors in real-world environments has evolved for many years, synthetic, simulated or imitated data has been used for several years as point of reference for the performance of algorithms for computer vision [7]. Previous works have established that driving simulators especially those that rely on game engines and high-fidelity computer graphics can be used to model real world driving scenarios.

The main goal of a ground-truth generation process is that at any instant, we are able to accurately give details of the position, orientation and kinematics of an observed object or obstacle as the case may be within the view reference of the ego-vehicle. Ground truth data includes images, image labels (which may be added manually or automatically by analysis) and a defined model for object detection [21].



Figure 4: Illustration of Ground Truth Data generation from an input image and the output of from a prediction model. [22]

2.2.2 Object detection

We have established earlier that one of the computer vision (CV) tasks in AVs is Object detection (OD). This follows simple logic because, human-driving requires perception and control and similar assignments would be transferred to computers. Being able to accurately identify objects in images and in real-time as in the case of continuous image/video streams such as cars, traffic and road signs, pedestrians, dogs, forests etc., may speed up the development of autonomous cars with high level of safety borne as human drivers.

While it is a relatively easy task for humans to detect and identify objects present in an image, it proves a rather a more non-trivial activity for computers. There is a massive biological computing power that the visual faculty of humans offers, making it fast and accurate and can perform complex tasks like identifying multiple objects and even spot obstacles with little consciousness. Owing to the availability of massive amount of data, more refined machine vision algorithms and GPUs with faster computing speed today, we are now able to train devices to detect and classify different objects from images with higher accuracies.

In the past, 2D methods of object detection were popular. These approaches involve classifying objects based on some feature extract paradigm to predict the probabilities of an object in an image. The correctness of these systems is poor because 2D image signals are sensitive to altering imaging conditions such as brightness, darkness etc. 3D information is more consistent because it holds shadow, as a consequence, more detailed

results are obtained. Figure 5 shows an example of such a model, where a model is trained on a dataset of closely cropped images of a car and the model predicts the probability of an image being a car [22].

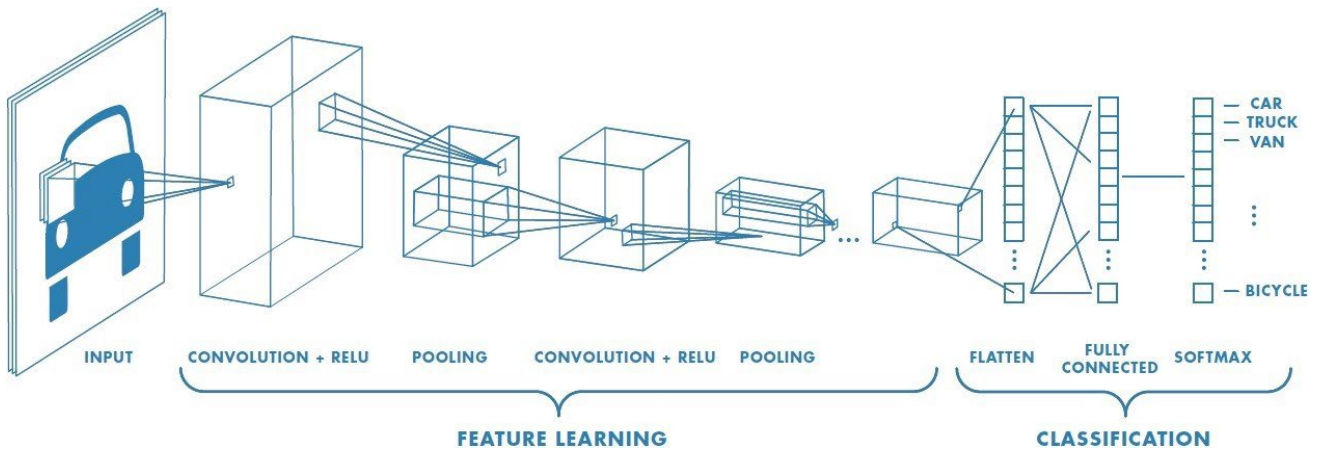


Figure 5: Illustration of Convolution network for object detection⁴

YOLO (latest stable version YOLO v3 and with new release YOLO v4) is one of the fastest object detection models. It uses the Darknet-53 (i.e. 53 (3 X 3) and (1 X 1) convolutional layers) feature extractor. Following from the 19 layered darknet extractor in v1, darknet-53 with an extended network of 53, draws inspiration from the ResNet idea of skipping connections within the convolution network without gradient diminishment as the propagation to deeper layers ensue. Prediction of bounding boxes is done in at 3 different scales based on the concept of feature pyramid networks.

⁴ <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

Table 2: Darknet-53 [23]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

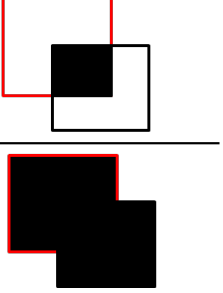
Object detection is not limited to 2D methods on images alone. We now also have 3D object detection models on images and methods that leverage LiDAR (Light Detection and Ranging) point cloud information. Although, Lidar produces data in the form of point cloud, it generally gives higher accuracy than camera-based approaches. It is a faster, effective and highly accurate way of object detection and creates high-resolution. Examples of 3D object detection methods are: PerspectiveNet, HGNet (Geo only), SVGA-Net, IPOD, PV-RCNN, 3D-MPA, CenterPoint, ImVoteNet etc.

2.2.3 Object detection metrics

Performance measurement is key in many tasks in computer vision. When you build an object detector, you concern yourself about how well your model has performed. And sure, while your model can identify objects within an image, there's a need to be able to quantify the performance. The objective of an object detection assignment is to;

- (i) Spot if or not an object is present and determine the class of that object in an image
- (ii) Estimate the coordinates of bounding boxes placed around the objects within images.

When we use bounding boxes in images, these two outcomes are used for further evaluation of performance. We use the concept of Intersection over Union (IoU) which is a computation of the ratio of intersection and union of ground truth the bounding box and the prediction bounding box.

$$Intersection\ over\ Union\ (IoU) = \frac{Overlap\ area}{Union\ area}$$


For instance, assuming the box with red boundary is the ground truth and the other, prediction, an IoU of ‘1’ will mean that both boxes coincide perfectly. We set a suitable value for the IoU threshold to determine if the OD is valid or not. [24]

Based on this threshold, we can classify object detection into 2 categories and an additional 2 that relate to undetected objects:

- True Positive (TP) – $IoU \geq \text{threshold}$ or correctly classified
- False Positive (FP) - $IoU < \text{threshold}$ or wrong detection
- False Negative (FN) – ground truth not detected.
- True Negative (TN) – every part Not useful in OD task

Note that the threshold can be set to any arbitrary value as there is no justification for the value.

Based on this, it means metrics gotten with these parameters are only for one setting. As metrics to evaluate the performance of the model, we sometimes use precision and recall. Precision and recall are quite familiar concepts.

Recall (sensitivity) is a probabilistic measure that a test will yield true positive for objects classify an object to its correct class and is given by the formula:

$$Recall\ (Sensitivity) = \frac{TP}{TP + FN} \times 100\%$$

Precision on the other hand, measure the degree of accuracy of positively determined classification, given by the formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\%$$

Depending on the developer of a model, there are several other metrics that can be used to evaluate the performance of a model within a specific area of application. Examples of these other metrics include: Average Precision (AP), Interpolated AP, AP (Area under curve AUC), mAP (mean Average Precision), Localization Recall Precision (LRP) etc.

3 METHODOLOGY

In this section, we describe briefly the architecture showing how processes flow from setting up the systems, data collection, processing and evaluation. We also describe the tools and techniques employed with some explanation on why they are chosen. We used a vehicular software platform that supports autonomous driving and provides functionalities for sensors. Our principal computer runs on the Windows 10 Enterprise operating system while we setup a virtual machine (VM) (Oracle Virtual Box) which runs the Ubuntu Xenial (16.04 LTS). The latter is the same as for the Iseauto project to maintain consistency. On the virtual machine, we installed and configured the ROS and Autoware environment.

3.1 Architecture

Figure 6 shows the architecture of the proposed method. We began by setting up the BeamNG.research video game and test run for integrity. Note that the forked version available on the project site⁵ does not provide as much functionality. So we got a more upgraded version from the development team.

Although we considered two setup arrangement for the BeamNGpy interface – one is to have it on the local machine and then connect it directly with the video game; the other is to set it up within the VM together with the ROS+Autoware setup using a pipeline and then another developing another interface to communicate with environment located on the local computer, the former was used because of the challenges faced in trying to work with the latter.

Once this setup is established, we can then extract relevant data for the purpose of our work. Geometric resources in the gaming platform are communicated to the Graphics Processing Unit (GPU) [7]. More specifically, we will use LiDAR and camera together with the vehicle agent to scan street scenes. Simulated data provides for pixel, photo-realistic and annotated data. Pixel-level annotations provide more accurate details about ground truth objects. The color images can serve as training dataset while annotated images as validation dataset. Later, we validate Yolo v3 with color images obtained from the simulator to see how well the model performs.

⁵ <https://projects.beamng.com/projects/research/repository>

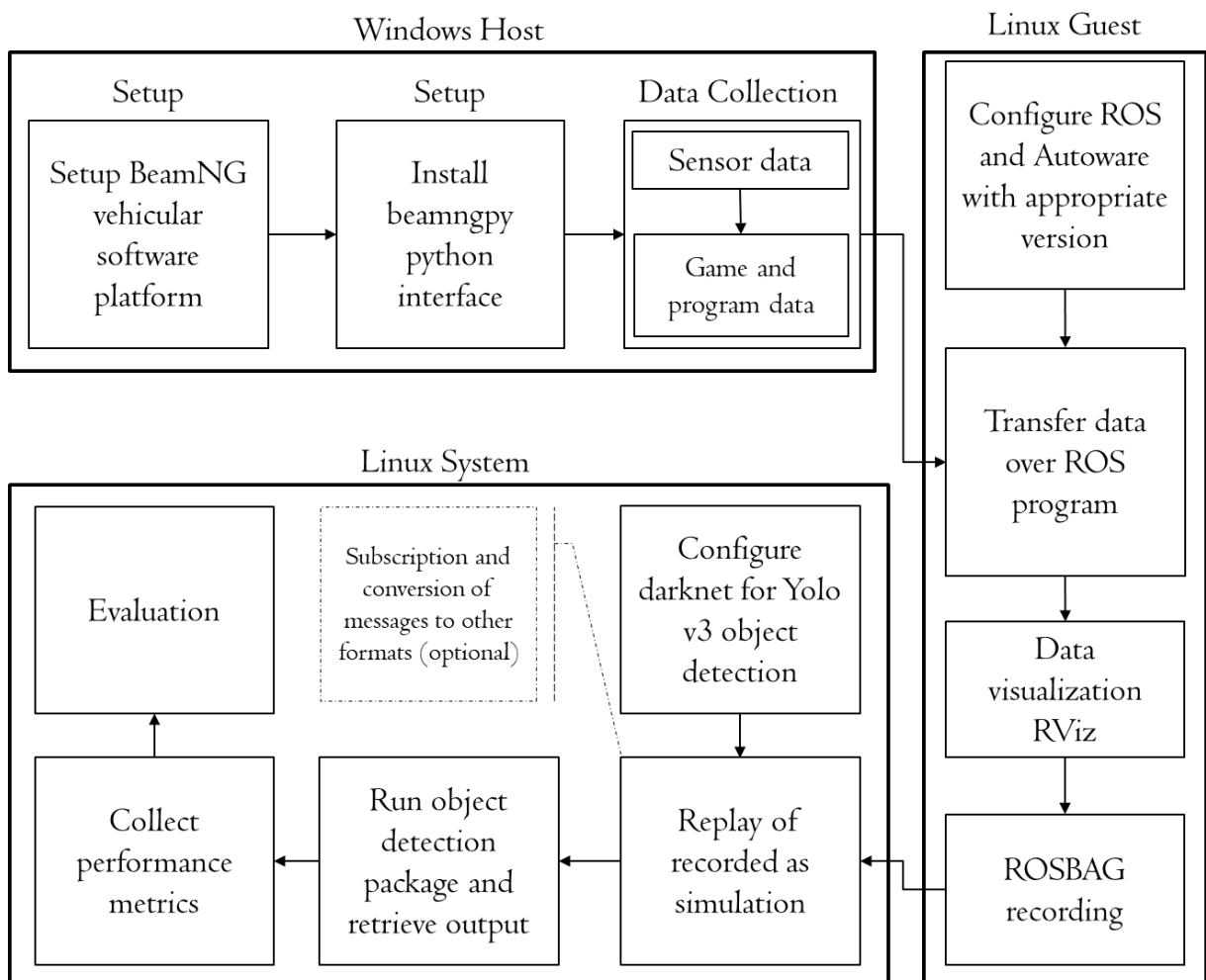


Figure 6: Architecture

3.2 Technical platform and tools

3.2.1 *BeamNG.research*

BeamNG.research [25] is BeamNG GmbH's non-commercial, versatile and open source platform for academic studies and research projects in development related to vehicles. Its versatility in automotive industry is hinged on the several features it has - state-of-the-art soft body physics, collision detection, interoperability with other standards, image annotation, simulation, aerodynamics and hydrodynamics, sensor inclusiveness and autonomous vehicle testing.

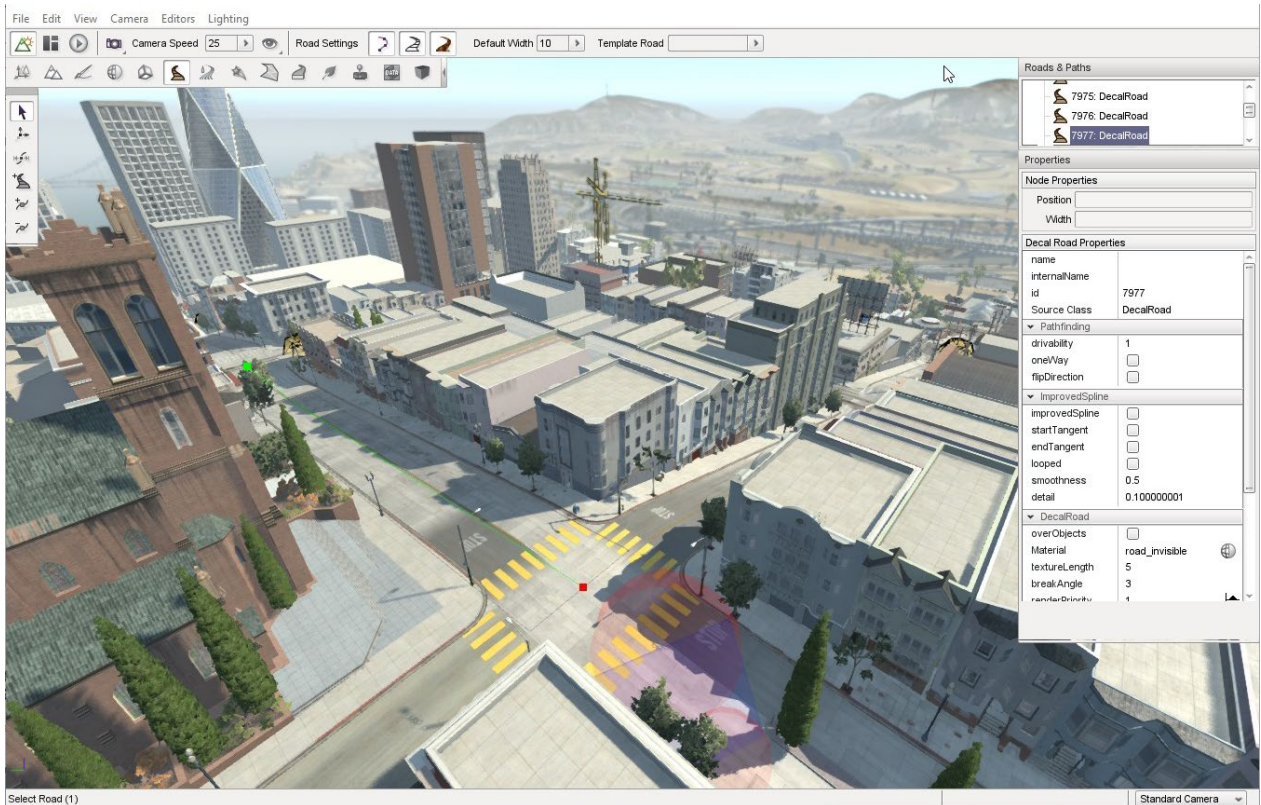


Figure 7: BeamNG environment

Beamng.research and beamng are used interchangeably throughout this thesis. The beamng simulator provides good physics support for crash and gaming activities. It also avails some of the sensors. As of the time of working on this project, there was already availability of LiDAR and camera sensors (two which were used in this thesis) although not with sensor fusion capabilities. Other sensors, like GPS and Radar are a work in progress.

Sensors

- Forces/Accelerations in high resolution on 2k points in the car
- LiDAR laser scanner currently equipped to capture 2.2million points at a rate of 30 FPS
- GPS positioning - WIP
- Ultrasound / Distance - WIP
- Radar - WIP
- Multiple cameras - WIP
- Sensor setup changeability and data re-extraction (Sensor fusion) – WIP

Requirements

Table 3: BeamNG Requirements⁶

Recommended	Minimum
<i>Normal setting at 1080p</i>	<i>Lowest setting at 720p</i>
<ul style="list-style-type: none">▪ OS: Windows 10 64-Bit▪ CPU: AMD Ryzen 7 1700 3.0Ghz / Intel Core i7-6700 3.4Ghz (or better)▪ RAM: 16 GB RAM▪ GPU: AMD R9 290 / NVidia GeForce GTX 970▪ DirectX: Version 11▪ Storage: 20 GB available space	<ul style="list-style-type: none">▪ OS: Windows 7 Service Pack 1▪ CPU: AMD FX 6300 3.5Ghz / Intel Core i3-6300 3.8Ghz▪ RAM: 8 GB RAM▪ GPU: Radeon HD 7750 / NVidia GeForce GTX 550 Ti▪ DirectX: Version 11▪ Storage: 15 GB available space

The camera sensor provides various types of image data depending on the perspective of the user. It can provide the following types of data:

- Colour images that can be converted to different formats within the MIME standard e.g. PNG, JPEG, BMP, GIFF etc.
- Pixel-wise depth
- Pixel-wise object annotation

A single camera sensor can be configured to provide any or all of these data at once, ensuring they all align to the same perspective. The camera sensor is set up with a fixed offset position and directional vector to face relative to the vehicle. This means as the vehicle moves and rotates, the camera is moved and rotated accordingly. Apart from the position and orientation, the image can further be customized with the FoV angle the camera should have, the resolution of the image(s) it outputs, and the near/far plane at which objects get clipped from view. The type of camera sensor data to provide can be indicated using boolean bit for the corresponding type. By default, they are all set to “False” and one needs to specify what type when polling the sensors while driving.

⁶ <https://wiki.beamng.com/Requirements>

Parameters: pos (tuple): (x,y,z) this tuple represents the camera's position offset relative to the vehicle it's attached to. direction (tuple): (x,y,z) tuple expressing the direction vector the camera is facing. fov (float): The Field of View of the camera. resolution (tuple): (width, height) tuple encoding the camera's output resolution. near far (tuple): tuple of the distance beyond which and after which geometry gets truncated. colour, depth and annotation are all (bool) flags and determine whether to output colour, depth or annotation information respectively.

The Lidar sensor provides 3D point clouds representing the environment as detected by an emitting laser pulse from the simulator vehicle. The lidar parameters provided in this simulator are similar to those of the Velodyne HDL-64E lidar⁷. The position, range, and refresh rate of this sensor can be modified as deemed suitable.

3.2.2 *BeamNGPy*

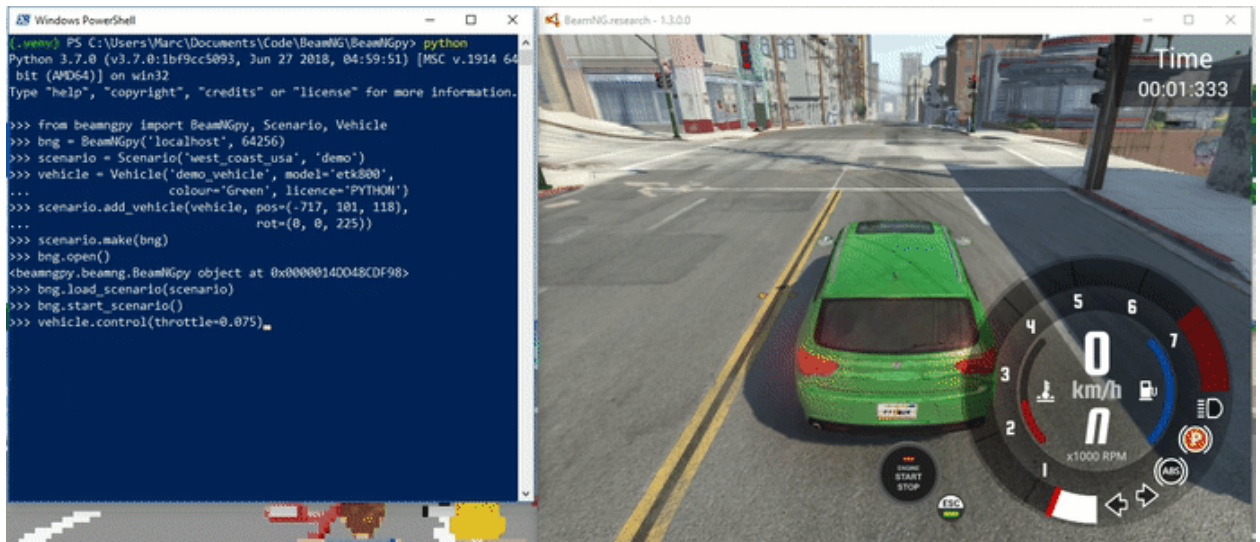
Although beamng.research offers a GUI-based method for starting and configuring the simulator, it still offers an interface to programmatically interact with the simulator. The forked video game uses a Python interface called BeamNGpy, which is an official library that allows make remote controlling of the simulation including the vehicles possible. Vehicles (known as agents) and environment may be configured with different sensors that facilitate simulated sensor data such as a feed from camera, that provide options for pixel-perfect semantic annotation and depth values or a simulated LiDAR sensor [26].

Table 4: Requirements for BeamNGpy

Module	Version
Click	7.0
Jinja2	2.10.1
msgpack-python	0.5.6
numpy	1.16.3
Pillow	6.0.0
PyOpenGL	3.1.0
scipy	1.2.1

⁷ <https://velodynelidar.com/products/hdl-64e/>

The package can be added to the python libraries existing in the computer by simply installing the beamngpy package. After installation, the package can be simply imported into the python code. This interface provides a pipeline for extracting data from sensors – images, speed, direction, point cloud etc. and also a means for imputing loopback closure for steering and control during simulation in real time.



Python interface

Simulator

Figure 8: BeamNGpy usage⁸

The game uses two virtual machines, one for the game engine and the other for each vehicle (regarded as an agent each having its own operating system) in the simulator and is written in the open-source software, Lua.

“Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.”⁹

We can change the behavior of the game by making a number of key presses with the GUI and subtle alterations in the code using the python interface, making it possible to run iterations during development faster [25]. We tried this all through our work adjusting behavioral parameters until suitable. One can better appreciate the Lua

⁸ <https://github.com/BeamNG/BeamNGpy>

⁹ <http://www.lua.org/about.html>

programming language as used for the development of beamng.research going through the beamng.drive documentation¹⁰.

3.2.3 ROS

The Robot Operating System (ROS) [27, 28] is a multi-platform framework which provides a rich variety of packages, nodes, libraries and tools used for build robot software and applications with the aim of simplifying the tasks related to build robust and complex robot behaviors. This is also a good framework that support collaboration of people working on the same project.

ROS packages created, reside in a **workspace** directory. The command for creating a ROS package is:

```
catkin_create_pkg package-name
```

Package name in our case, is pointcloud. The building blocks of implementing a ROS program includes nodes, messages, topics, and services. All of these resources are collectively known as **graph resources** and individually, as **graph resource name** [29]. The main mechanism for interacting with the ROS system is with **nodes**. They are the process objects that are able to perform computation.

From the programmer's / user's prospective, "ROS Computation Graph" is kind of "front-end". It is a peer-to-peer network of "nodes" and "services" processing the data and exchanging it in the form of "messages". A ROS node is an executable software module that perform computation. Nodes help to maintain the modularity of ROS in the sense that each node is, ideally, designed to perform single task. Nodes in ROS communicate via passing "messages". Each message has a type that is specified by the type of data it carries. So simply put, a message is a data structure consisting of type fields of integer, float and Boolean, etc.

Each message is shared among the nodes under a specific "topic". The producer node of the message "publishes" the topic and the user of that message "subscribes" to that topic. Although the data sharing among the nodes via the publishers and subscribers provide a flexible mode of communication, it is not very suitable for request / reply

¹⁰ <https://documentation.beamng.com/index.html>

mode of communication. Because it is one-way data communication. This deficiency is overcome by “services”. A service is a request/reply mode of communication. A service node has both the “request message” and “reply message” structures. A client which needs to “use” a service sends a request message and then waits for the “response message” from the provider of the service [28].

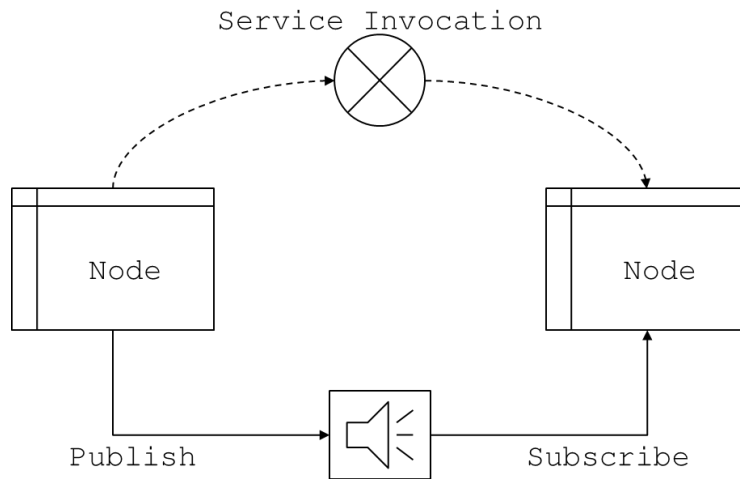


Figure 9: ROS basic concept

Our choice for the ROS framework is because the framework is mature and large community base. It is widely used across board from academics, to research and to robotic development projects. ROS also provides other packages that facilitate autonomous systems development e.g. RViz for 2D and 3D visualization e.g. images and laser scan data and ROSBAG for summarizing topic messages into a bag that can be replayed within a ROS computation graph.

3.2.4 *Autoware*

Autoware is an open source research and development platform used for autonomous driving technology by researchers, enthusiasts, developers, and students. It is built upon ROS 1 and has evolved over the years to support future projects. In the Autoware portfolio are, Autoware.AI, Autoware.IO and Autoware.Auto which is based on ROS 2 and has a redesigned architecture [30]. It offers a ROSBAG enabled simulation environment for testing without using real time autonomous vehicles. This is useful for

our thesis because both the driving setup and AV development stack reside in different computers.

In a ROS-based computer, autoware can provide functions for object detection, localization, path following, ego vehicle motion controls and 3D mapping. It also provides interfaces for controlling autoware (Runtime manager) and for 3D visualization of functions (RViz).

3.3 System requirements

Table 5 shows different system requirements for the tools we used in this thesis. Note that some of the tools normally have frequent updates which may have happened during the course of our thesis.

Table 5: System requirements

Machine/Tool	Specification
Host PC	Memory: 256GB SSD Processor: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80GHz Installed RAM: 16.0 GB (15.9 GB usable) Type: 64-bit Operating System, x64-based processor DirectX: 12
Virtual Machine (VM)	VirtualBox Graphical User Interface Version 6.1.10 r138449 (Qt5.6.2) Installed OS: Ubuntu 16.04 (Xenial) Base Memory: 8 GB Processors: 4 Acceleration: VT-x/AMD-V, Nested Pegging, KVM Paravirtualization Storage: 48.36 GB
ROS	Distribution: kinetic Version: 1.12.14
Autoware	V 1.8/V1.12
Python	Python 3.8.3 64-bit

3.4 Summary

In this chapter, we highlighted the different tools and the setup of the study. We demonstrated the system requirements to ensure proper functioning of setup. At the same time, we identified some key installation processes. In the next chapter, we look at how we collected experimental data and the processes involved in data manipulation.

4 EXPERIMENT

In this chapter, we demonstrate how we collected our data, the tools involved and how we ordered our folder. We also attempted to identify key actions that aided our data collection and data processing.

As we have identified in Chapter 3 earlier, the setup involved the use of two operating systems (Windows 10 Enter on the host PC where the simulator is run and Linux Ubuntu 16.04 on the VM where our robot operating system and the autoware are housed). Although, with the host being slightly below the specified system requirements, we were able to use it to achieve the minimum results from data generation to object detection. A video demonstration of the latter can be found in the link in the footnote section¹¹. The whole process is described in fuller details in the paragraphs below.

A python code that leverages the beamNGpy interface was used to interact with the game as mentioned previously. Code can be found in link in the Appendix item 1. The vehicular software platform offers a number of scenarios that correspond to typical settlement types in the real world. E.g. east_coast_usa, port, island, canyons, west_coast_usa etc.

We opted for the west_coast_usa scenario because it is the default used by the software providers and on inspection, appears to be quite sophisticated. The west_coast_usa map offers a good variety of environmental scenery similar to where the ISEAUTO vehicle is expected to operate – a metropolitan urban settlement with many elements that highlight modern realities.



Figure 10: Minimap of west_coast_usa scenario

¹¹ <https://shorturl.at/desS2>

We selected 2 sensors that will be attached to the ego vehicle in the game for getting data from the game – a camera and a Lidar. We have presented in Table 6, some parameters specified for each sensor used in the experiment.

Table 6: Specification of sensor parameters

Parameters	Camera	Lidar
*Position (tuple)	[1,0,1.6]	-
Direction (tuple)	[0,1,0]	[0,1,0]
*Offset (tuple)/(m)	-	[0,0,1.6]
fov (°)	60	-
Resolution (tuple)/pixel	[1392, 512]	-
Near_far (tuple)/m	[0.01, 120]	-
Vres	-	64
Vangle (°)	-	26.9
rps (real number)	-	2200000
fps (hz)/refresh rate	-	20
Angle (°)	-	360
max_dist (m)	-	120

* parameters are with respect to the position of the vehicle

Guide to some parameters used in the table 5:

- **offset** - refers to coordinates specifying the position of the Lidar sensor relative to the ego vehicle's.
- **fov** – (field of view) refers to the range of view in the observable world around the sensor.
- **Vres** (vertical resolution) indicates the number of vertical lines sampled by the Lidar
- **Rps** (rays per second) is similar to pulse per second for common lidar sensors available.
- **Fps** (frames per second) – how many frames the sensor can shoot in one second, also called the refresh rate
- **Max_dist** – (maximum distance) beyond which any ray of the sensor that falls within with not be displayed in the sample data.

We had the liberty to choose from a number of spawn points as shown in the **Figure 11** below. Also, we modified the scenario parameters in the game to allow for scanning through every area of the map automatically. This is done by setting the `ai_set_mode()` method which is an inbuilt simulator's AI for the vehicle, to span.

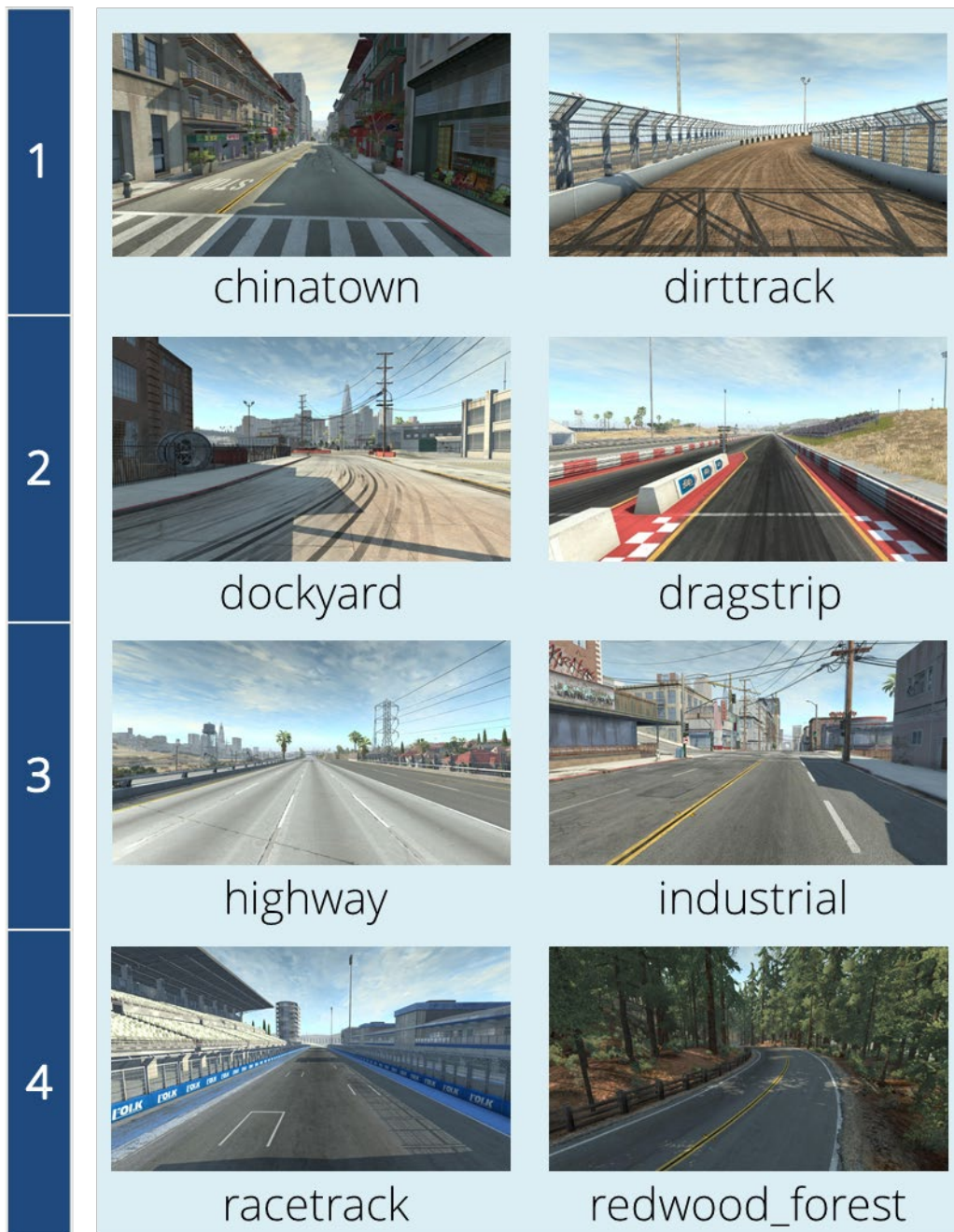


Figure 11: Different spawn points available in the west_coast_usa scenario. 1 L-R: Chinatown, Dirttrack; 2 L-R Dockyard, Dragstrp; 3 L-R Highway, Industrial; 4 L-R Racetrack, Redwood_forest

Once the beamng simulator is started, we can see the different communication messages on the game console window. Furthermore, there is a logging library that enables functions and variables be seen on the python compiler used in running the simulator program.

4.1 Ground truth data from BeamNG.research

The path to the beamng simulator executable is set in the system variable. The BeamNGpy instance runs the simulator from the given path and communicates over the localhost server on port 64526 (localhost:64256).

Using the `find_static_objects()` method provided in the documentation, we were able to determine that there are at least, 1567 static objects (members of the TSSStatic class) with their positions in the west_coast_usa scenario. In this thesis, we focus only on stationary objects which the method provides for. Hence, there are no instances of pedestrian and/or moving cars. The reason this is so, static objects within the world readily provide ground truth information as they would not change no matter the number of iterations we make in the simulator. We would consider pedestrians and vehicles in future works. With the aforementioned, we can obtain ground truth information about objects along the path of the ego vehicle.

As we drive around the road, we adopt a file system structure similar to that used in the KITTI dataset for both the images and point cloud data gathered from our simulations. We use the KITTI point cloud dataset (KITTI-PC) as a point of reference. The KITTI-PC consists of 7481 and 7518 training and testing images respectively [31]. This dataset provides benchmarks evaluation for 2D object detection and orientation estimation, 3D object detection, and bird's eye view analyses.

We polled all the sensors in the game with the `poll_sensors()` method on every iteration to capture data from each sensor attached to the vehicle in the shared memory. We captured a total of **200 frames**. The idea is to ensure that we get as many varied scenes as possible, to diversify the difficulty level for the object detection. Each frame instance, collects images from camera and point cloud data from the LIDAR. We also stored information about the vehicle position at the instance frames are captured, timestamps and sensor data.

Host PC + Simulator	Guest + ROS Program
<pre> C:\USERS\ASK4JUBAD\DOCUMENTS\RESEARCH\BEAMNGRESEARCH beam.py Copy of object_lists.txt DxDiag.txt object_lists.txt README.md +---.vscode launch.json settings.json +---annotation timestamp.txt \---annotation 0.png 1.png 2.png +---colour timestamp.txt \---colour 0.png 1.png 2.png \---data timestamp.txt vehicle_pos.txt \---data pointcloud0.csv pointcloud1.csv pointcloud2.csv </pre>	<pre> D:\POINTCLOUD CMakeLists.txt package.xml +---bagfiles .listener.py.swp 2020-07-31-02-52-14.bag +---include \---pointcloud +---launch launcher.launch \---src listener.py talker.py </pre>

Figure 12: Snapshot of file folder structure used in the project

4.2 Images and LIDAR points

As mentioned above, our ego vehicle is equipped with camera and LiDAR sensor. The camera is the visual sensor providing the of images of the environment, from which we convert to RGB format. The LiDAR provides the depth data in the form of point clouds. In our ROS package pointcloud, we gather all the runtime processes in the launch file called launcher.launch. This helped us to run multiple nodes at the same time like talker (publisher node), listener (subscriber node), rviz and rqt_graph nodes.

4.2.1 Images

The images are gotten with the invocation of the camera sensor object in the simulator. The raw images are converted to RGB color format and stored as PNG files (we attempted to store in the irreversible lossy compression format JPEG, in a bid to aid data storage, handling and transmission bearing in mind trade-off for image quality but encountered some challenges with the output being somewhat distorted). PNG graphics support lossless compression of data.

We converted the same raw images to their **annotated format** and stored them likewise as PNG files. The annotated images represent ground truth data about object boundaries within image frames. In the ROS program, the publisher node (talker) images are being

published under the `/image_view` topic. Images published on this node are first converted to

4.2.2 *Point cloud*

The Lidar points were collected in the form of xyz coordinates into an array variable. This was then appended in a file as comma separated values (CSV). This is a practical way of storing such data as csv files can be converted to other point cloud formats including .xyz and .bin. In the publisher node in the ROS program, the point cloud data is being published under the topic `/pointcloud`.

4.3 Visualization of data in RViz

In our project, we used the in-built ROS visualization package called RViz to visualize both the images produced by the camera sensors and the point cloud data generated by the LiDAR. RViz comes as one of the packages accompanying ROS installed in the Ubuntu OS on our virtual machine. It is a 3D visualization tool for ROS. RViz lets us see what our robot is seeing, thinking or doing. Rviz understands sensors and state information like Laser Scan, cameras, point clouds and coordinate frames. Rviz is extensively used for debugging robot applications.

A sample image captured is provided in Figure 13 below. The corresponding point cloud data of the image frame is also shown in Figure 14.



Figure 13: Sample image captured visualized in Rviz

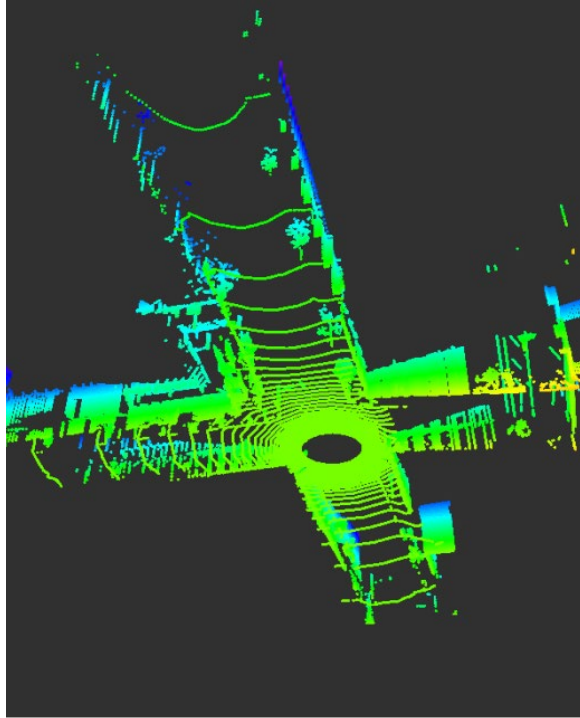


Figure 14: Sample point cloud data visualized in RViz

4.4 ROSBAG conversion

To summarise the content of our package and ensure that we can easily reproduce the collected data, we recorded the messages from the ROS topics `/image_view` for images and `/pointcloud` for point cloud into a rosbag. This is used for further processing while testing our object detection model in chapter 5. This was done by running the following command in a the `~pointcloud/bagfiles` folder within the pointcloud ROS package.

```
rosviz record -b 0 -a
```

The `-b` flag sets the buffer size (0 as infinite) and `-a` (all) flag to record all incoming topics. The rosbag file was played back and messages from both the `/image_view` and `/pointcloud` topics were read to test object detection algorithms below.

4.5 Summary

In this chapter, we demonstrated how we collected our data, the tools involved, the data formats and how we ordered our folder. We also attempted to identify key commands that aided our data collection and processing such that same can be reproduced. In the next chapter, we are going to show how we validated the object detection model for our images and some performance metrics. Table below summarizes the outputs of our method and the purpose of each.

Table 7: Summary of outputs and their purpose

Source	Output	Purpose
Camera	color image (.png)	To reproduce camera stream on ROS/Autoware
Camera	Pixel-wise annotated image (.png)	validation object detection on ROS/Autoware
Lidar	Pointcloud (.csv)	Reproduce pointcloud data in ROS/Autoware and for 3D visualization in RViz
Time (Python method)	Timestamps	For us to inspect the time gap between successive frames of data collection from all the sensors.
Both image and pointcloud data	Bag file (.bag)	Full recording of the data streams for playback in ROS/Autoware
Darknet (Yolo v3)	Bounding box images (.jpg)	For us to compare the bounded objects and the annotated images

5 VALIDATION

To validate our prototype of the proposed method, we ensured that we are able to read data by writing commands through the Python interface provided by the game engine and that we are able to read collected via ROS (publish and subscribe) nodes. We were able to get ground truth annotated data has been provided by the converted annotated we did not have to manually mark the boundaries of target objects in the image.

5.1 Validating with Object Detection



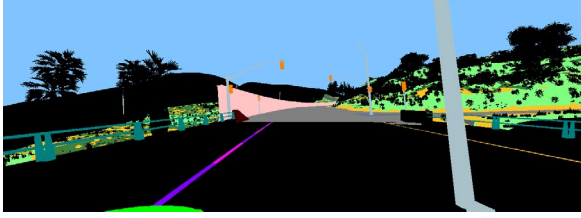
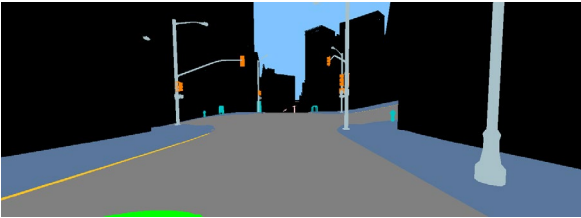


To test our method against image detection baseline, we examined image-based 2D object detection algorithm, Yolo v3 (available in ros darknet package)¹² on the images being published. We have opted for Yolo v3 detector based on the outcome and recommendation of the work from [20] on the ISEAUTO project and also the popularity of the detector as a fast and more accurate object detection algorithm. We installed the ROS darknet [32] folder into our workspace and built the project. We ran the yolo launch file from inside the launch folder to read images from the `/image_view` topic by modifying the yolo_v3 launch file suitably. We then played our previously recorded bag file to publish saved image files. The darknet package offers a GUI for viewing the outcomes of object detection. We saw that the outputs matched the input images without any errors.

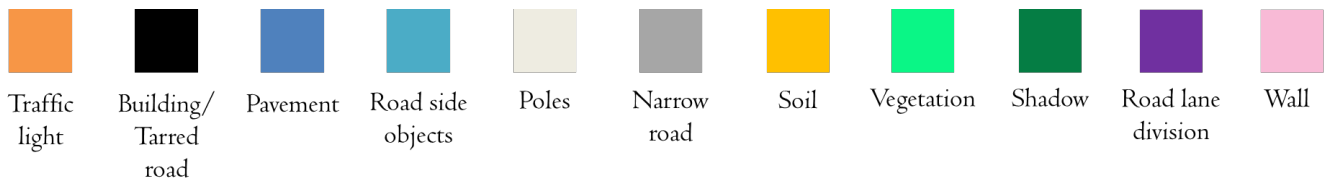
Annotation of objects within images helps us to accurately identify objects using a self-driving car software. Examples of objects of concern are pedestrians, traffic lights, stop signs, cars, animals, fire hydrants etc. This is the underlying concept for computer vision technologies [33]. From our beamng simulation, we were able to generate pixel-wise annotated images that provided ground truth information without the need for the usage of bounding boxes that require much effort of tagging an image with a class. In **Table 8**, we identified some of the annotation objects. A qualitative comparison of the annotated objects and the output of the Yolo v3 object shows that this proposed method is suitable for generating ground truth data.

A sample of the results obtained is shown in **Table 8** below.

¹² https://github.com/leggedrobotics/darknet_ros

Table 8: Table showing captured color image, annotated image of ground truth and object detection output.

	Sample 1	Sample 2
Color image from simulator		
Annotated image obtained from simulator		
Yolo v3 detection output with bounding boxes		



From qualitative check, we observed some slight shift in the frames between the color images and annotated images. This was due to the split second difference in the time when the data get stored in the computer. Although, the OD model was able to identify some objects that coincide with the ground truth, not every object was identified in processed frames.

5.2 Performance Measures (Precision and Recall)

Recall the formulas for precision and recall highlighted in section 2.2.3., we calculated our recall and precision values based on the total number of images that were processed by Yolo v3. These two measures are important because, recall helps us determine the completeness of our results and precision tells us about the relevance of our results.

Assumptions

- IoU threshold is relatively small and any observable correctly determined object is considered TP.
- Ground truth objects are those objects within all images that are also available in object classes in the Yolo v3 Common Objects in Context (COCO) names e.g. traffic light, stop sign, fire hydrant and potted plant

Since our ground truth is provided as pixel-wise annotations, it means detections that closely conform to ground truth are treated as TP, and detections that do not conform to observable ground truth (wrong classifications) are treated as FP. We also consider bounding boxes with no observable object as FP. Similarly, FN relate to objects within the processed images that should have been identified but were not.

The `darknet_ros` node topic `/darknet_ros/bounding_boxes`, publishes messages about the performance of the OD like fps and confidence score. By writing a subscriber node program to read messages from the topic, we were able to collect those pieces of information. Out of 200 read frames, on an NVIDIA GeForce RTX 2070, with 93% GPU load and an average speed of 26.74 fps, the Yolo v3 algorithm processed 41 images. The two measures values were calculated based on the identifiable objects in the images processed. We are able to do this manually because the number of frames involved are minimal.

Table 9: TP, TN, FP, FN Derived Measures

Definition	Total Number	Recall	Precision
TP	66		
FP	37		
FN	17	0.6408	0.7952
TN	NA		

5.3 Summary

In this chapter, we validated Yolo v3 by our method. We determined the precision and recall values of the object detection model as 0.7952 and 0.6408 respectively. These results show that our method is suitable for the generation of ground truth data.

6 FUTURE WORK

In our work, we used a monolithic weather condition all through. We did not explore multiple environmental conditions in which may further strengthen the adequacy of the use of purely synthetic ground truth data. Thankfully, the simulator offers features to modify environment conditions like time of day, weather or even the spawn location. Further work can be done in this regard, to determine what impact they would have on the outcome of object detection algorithm(s) for autonomous driving. More work should be done to see how synthesized image parameters play a part in the usefulness of the data.

Owing to limitations due time constraint, and the platform documentation, which made referencing a bit challenging some aspects of the study can be further improved. For instance, while we were able to get location of objects within map, we are unable to determine what type of objects are located in those positions. This study also lacked the computational requirement needed to get a smooth continuous stream of data from the simulation. This would help us to process in our ROS system, data at a more accurate frame rate thereby giving us a stream that can reflect real life driving. Also, since our ROS program was written in Python, the program can be re-written in C language to provide a robust and faster system.

As we have been able to establish the use of synthetic data from this new software platform, we can do a comparative study of different simulators and game engines available (e.g. LGSVL simulator^{13,14} (2020) available in Autoware, CARLA¹⁵ (2017) etc.) against some set benchmarks in the future. In this study, our focus was ground truth data for object detection and although this study was validated with object detection, we can adopt the prototype method introduced to explore more complex autonomous vehicle tasks like localization, path planning and even open area driving. The platform already has in built functions for controls and waypoints following which can be explored in future studies.

In principle, we can extend the validation done, by running lidar based object detection. Although it is more challenging task to deal with, they have proven to provide more accurate result. We can also employ strategies for fusion of vision and point cloud data as demonstrated in [34]. Furthermore, we validation using a bounding box annotation model;

¹³ <https://content.lgsvlsimulator.com/>

¹⁴ <https://github.com/lgsvl/simulator>

¹⁵ <https://carla.org/>

in future work, we can explore algorithms (MASK R-CNN and PANOPTIC FPN) that identify objects at pixel-level for better comparison with the annotated ground truth. As a result, we can improve iterations during development of autonomous vehicle by reducing human-effort in setting up systems for and generation of ground truth data.

7 CONCLUSION

The aim for this thesis is to generate ground truth from a simulator for object detection algorithm in autonomous vehicles. This was done by exploring some commonly used sensors in-built in the simulator. Furthermore, the implementation needs to be easy to produce and be improvable. We did check out the camera and LiDAR agents within the simulator and exploited the access we have to the game through the BeamNGPy python interface.

We created a prototype method that gets data from beamng.research by controlling and accessing the game via beamngpy. We generated 200 colored images and annotated images in PNG format from the camera sensor and also generated 200 captures of pointcloud data. We went further to visualize the images in RViz by converting them to OpenCV “bgr8” format. For the point cloud, we generated 200 comma separated value files corresponding to each capture of point cloud data. In a similar way, we transformed this point cloud data in our ROS program to Pointcloud2 type for visualization in Rviz.

Data streams were recorded in a rosbag file for further simulations and validation tasks. We used different metrics to specify system and detection performance. Our work shows that this method is feasible for creating ground truth data. Although, we faced some challenges with working with the simulator, with further exploration and support within the developer community we will be able to achieve better and more practical results. The following limitations were identified with use of the proposed method:

- The simulation has wavery driving compared to real-life which means that real time driving may not be of desired quality.
- The game is currently only supported on the Microsoft Windows. Other Operating System and Platforms, Virtual Machines, Emulators and compatibility layers are not supported.⁶
- Multiple agents are not configurable at the moment.
- Documentation for this forked version does not provide enough detailed information.
- Ground truth bounding boxes of frames in the surrounding of the ego vehicle are not yet included in the camera sensor functionalities.

In conclusion, we demonstrated a prototype method to achieve ground truth data using the beamng.research simulator by having sample data generated and showing the plausibility

with visualizations in the ROS environment using different ROS functionalities. We also validated the Yolo v3 object detection model with our data and evaluated our results. With the aforementioned, more studies need to be carried out to explore the use of simulated environments data for the development of autonomous vehicles.

There are well founded prospects in the field of autonomous vehicles, ranging from sensing, perception, decision making, sensor fusion, path planning to control and steering. We have provided in the Appendix section, some very recent development in the autonomous and computer vision space that may be of interest to the reader.

References

- [1] M. Jaafarnia and A. Bass, "Tracing the Evolution of Automobile design: Factors influencing the development of aesthetics in automobiles from 1885 to the present.," in *Proceedings of the IMProVe International conference on Innovative Methods in Product Design*, Venice, Italy, 2011.
- [2] Dreamstime, "Automobile history: how the shape has evolved in one century," n.d. n.d. n.d.. [Online]. Available: <https://www.dreamstime.com/royalty-free-stock-photo-automobile-history-how-shape-has-evolved-onecentury-image29995115>. [Accessed May 2019].
- [3] "The Self-Driving Car Timeline – Predictions from the Top 11 Global Automakers. Retrieved in May 2019 from," [Online]. Available: <https://emerj.com/aiadoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/>. [Accessed May 2019].
- [4] T. Litman, *Autonomous vehicle implementation predictions*, Victoria Transport Policy Institute, 2017.
- [5] M. F. Kragh, "Lidar-Based Obstacle Detection and Recognition for Autonomous Agricultural Vehicles," AU Library Scholarly Publishing Services, Aarhus, 2018.
- [6] S. R. Richter, V. Vineet, S. Roth and V. Koltun, "Playing for Data: Ground Truth from Computer Games," in *European Conference on Computer Vision (ECCV)*, Amsterdam, the Netherlands, 2016.
- [7] Michigan Tech Research Institute, "Michigan Tech Research Institute (2019). Ground-Truth Data Collection for Autonomous Vehicle Development," 2019. [Online]. Available: <https://mtri.org/multisensorevaluation.html> . [Accessed 20 October 2019].
- [8] P. Morse, "Use Cases for Automotive Driving Simulators," [Online]. Available: <https://www.ansiblemotion.com/automotive-driver-in-the-loop-simulation-articles/use-cases-for-automotive-driving-simulators>. [Accessed 16 October 2019].
- [9] Taltech Autonomous Vehicles Lab, "Self-driving shuttle – ISEAUTO," [Online]. Available: <https://autolab.taltech.ee/portfolio/iseauto/>. [Accessed 12 July 2020].

- [10] [Online]. Available: <https://iseauto.taltech.ee/en/tehniline/>. [Accessed 12 July 2020].
- [11] S. I. Nikolenko, "Synthetic Data for Deep Learning," *ArXiv*, vol. abs/1909.11512, 2019.
- [12] G. Liu, "Real-Time Object Detection for Autonomous Driving Based On Deep Learning," Master's Thesis. Texas A&M University-Corpus Christi, Corpus Christi, 2017.
- [13] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *Conference on Robot Learning (CoRL)*, 2017.
- [14] GPS WORLD, "Accurate ground truth for autonomous vehicles," North Coast Media and NovAtel, [Online]. Available: <https://www.gpsworld.com/sponsoredcontent/accurate-ground-truth-for-autonomous-vehicles/>. [Accessed 16 October 2019].
- [15] A. Geiger, P. Lenz, C. Stiller and R. Urstasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231-1237, 23 August 2013.
- [16] H. Ceasar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom, "nuScenes: A Multimodal Dataset for Autonomous Driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [17] D. Atsmon, "Autonomous-Driving Simulation Tackles Kangaroo Issue," *Informatel*, 03 April 2018. [Online]. Available: <https://www.wardsauto.com/technology/autonomous-driving-simulation-tackles-kangaroo-issue>. [Accessed December 2019].
- [18] CVEDIA, "CVEDIA. SYNTHETIC DATA FOR AUTONOMOUS VEHICLES. Simulated environments, synthetic datasets, and machine learning solutions from CVEDIA.," CVEDIA, [Online]. Available: <https://www.cvedia.com/industry/autonomous-vehicle/>. [Accessed 2019 October 17].
- [19] W. Li, C. Pan, R. Zhang, J. Ren, Y. Ma, J. Fang, F. Yan, Q. Geng, X. Huang, H. Gong, W. Xu, G. Wang, D. Manocha and R. Yang, "AADS: Augmented Autonomous Driving Simulation using Data-driven Algorithms," *Science Robotics*, vol. 4, no. 28, 27 March 2019.
- [20] A. Vainola, "Estimating object detection reliability for TTU "Iseauto" self-driving car," Master's thesis. Tallinn University of Technology, Tallinn, Estonia, 2018.

- [21] S. Krig, "Ground Truth Data, Content, Metrics, and Analysis," in *Computer Vision Metrics*, Apress Berkeley, CA, 2014, pp. 283 - 312.
- [22] S. Gupta, "Introduction to Object Detection," 7 August 2018. [Online]. Available: <https://www.hackerearth.com/blog/developers/introduction-to-object-detection/>. [Accessed 11 July 2020].
- [23] E. Y. Li, "Dive Really Deep into YOLO v3: A Beginner's Guide," Medium.com, 31 December 2019. [Online]. Available: <https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e>. [Accessed 02 August 2020].
- [24] R. Khandelwal, "Evaluating performance of an object detection model," Medium.com, 06 January 2020. [Online]. Available: <https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b>. [Accessed 04 August 2020].
- [25] BeamNG, "BeamNG.Research," BeamNG GmbH, Bremen, Germany, 2018. [Online]. Available: URL: <https://www.beamng.gmbh/research>. [Accessed 16 October 2019].
- [26] Palculator, "BeamNG / BeamNGpy," 03 June 2020. [Online]. Available: <https://github.com/BeamNG/BeamNGpy/blob/master/README.md>. [Accessed 19 July 2020].
- [27] Open Source Robotics Foundation, "About ROS," 2019. [Online]. Available: <https://www.ros.org/about-ros/>. [Accessed 16 October 2019].
- [28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, *ROS: an open-source Robot Operating System*, vol. 3, ICRA Workshop on Open-Source Software, 2009, p. 5.
- [29] J. M. O’Kane, *A Gentle Introduction to ROS*, 2.1.6 ed., Columbia, South Carolina: Independently Published, 2018.
- [30] J. Becker, "Announcing the Autoware Foundation—Open Source for Autonomous Driving," 9 December 2018. [Online]. Available: https://medium.com/@jan_26255/announcing-the-autoware-foundation-open-source-for-autonomous-driving-f340e9960eda. [Accessed May 2019].

- [31] G. Andreas, P. Lenz and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *IEEE Conference on Computer Vision and Pattern Recognition*, Providence, Rhode Island, 2012.
- [32] M. Bjelonic, YOLO ROS: Real-Time Object Detection for ROS, 2018.
- [33] V. Petrosyan, "Why pixel precision is the future of the Image Annotation," Medium.com.
- [34] H. Wang, X. Lou, Y. Cai, Y. Li and L. Chen, "Real-Time Vehicle Detection Algorithm Based on Vision and Lidar Point Cloud Fusion," *Journal of Sensors*, vol. 2019, pp. 1 - 9, 2019.
- [35] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda and T. Hamada, "An Open Approach to Autonomous Vehicles in IEEE Micro," *IEEE Micro*, vol. 35, no. 6, pp. 60-68, 29 December 2015.
- [36] Microsoft, "Visual Studio Code FAQ," Microsoft, 09 July 2020. [Online]. Available: <https://code.visualstudio.com/docs/supporting/faq>. [Accessed 16 July 2020].
- [37] H. Hajri, E. Doucet, M. Revilloud, L. Halit, B. Lusetti and M.-C. Rahal, "Automatic generation of ground truth for the evaluation of obstacle detection and tracking techniques," arXiv preprint arXiv:1807.05722, 2018.
- [38] S. Jégou, M. Drozdal, D. Vazquez, A. Romero and Y. Bengio, "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation," in *IEEE Computer Vision and Pattern Recognition Workshops*, 2017.
- [39] J. Hui, "Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3," [Online]. Available: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088. [Accessed 03 August 2020].

Appendix

- 1. Link to repository containing code used in beamng python interface for interacting with the simulator, ros-based code for reading collected images and LIDAR point cloud data for visualization**

<https://gitlab.cs.ttu.ee/juadiq/beamngresearch>

or on <https://bitbucket.org/ask4jubad/beamngresearch/>

Access to repository granted upon request to ask4jubad@yahoo.com. The repository contains a read me file for project description on how to use some parts of the code base. Please note that the paths used in the code may not exist, so you would have to figure it out as it suits appropriately for your use.

- 2. Useful links used during the thesis**

- a. BeamNGpy documentation*

<https://beamngpy.readthedocs.io/en/latest/api/beamngpy.html>

- b. BeamNGpy github repository*

<https://github.com/BeamNG/BeamNGpy>

- c. ROS Documentation*

<http://wiki.ros.org/Documentation>

- d. Autoware-AI repository*

<https://github.com/Autoware-AI/autoware.ai>

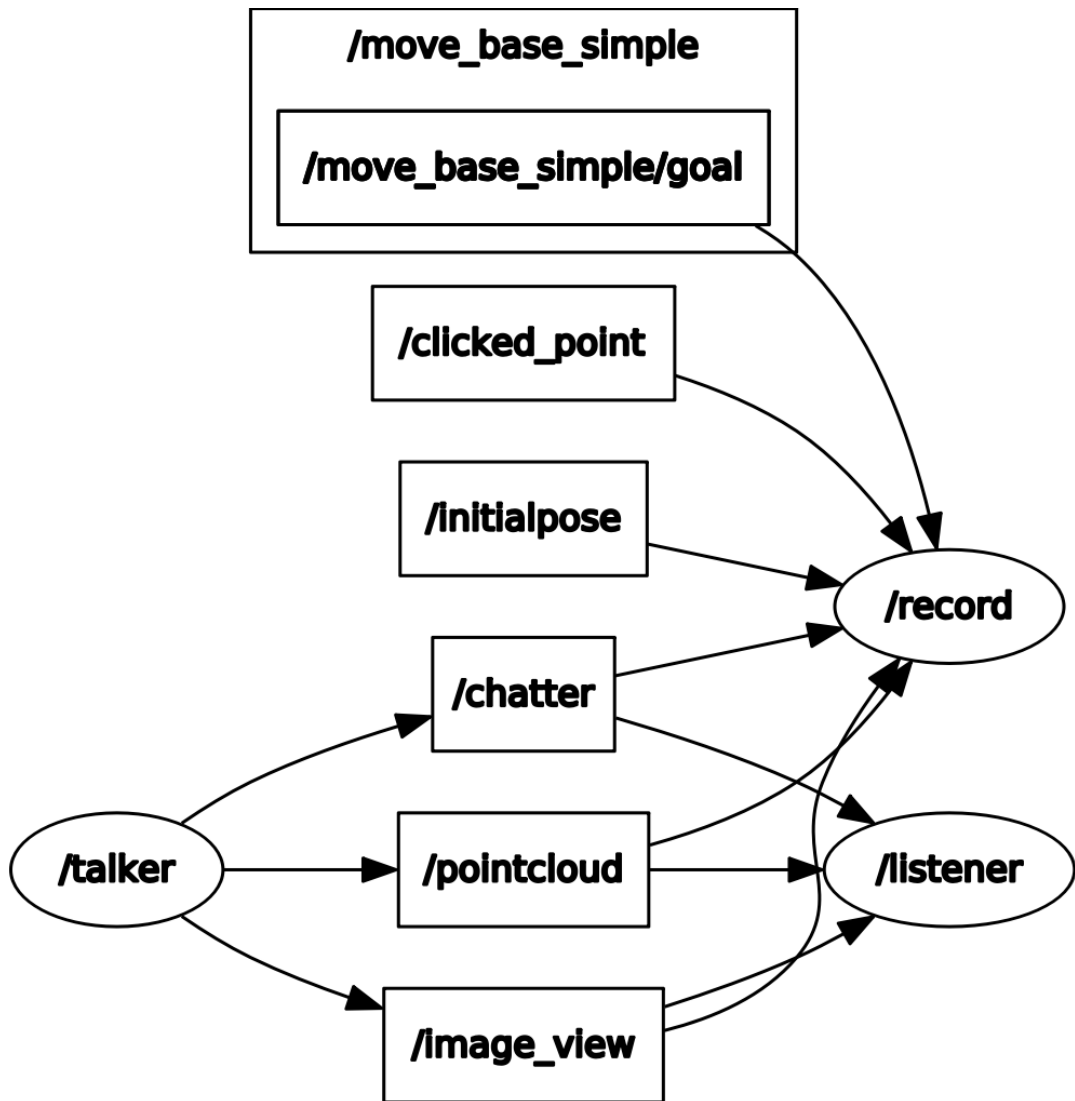
- 3. Further reading and viewing**

- <https://towardsdatascience.com/3d-object-detection-using-lidar-data-for-self-driving-cars-ee0eb0e6389e>
- <https://towardsdatascience.com/lidar-3d-object-detection-methods-f34cf3227aea>
- <https://www.youtube.com/watch?v=b5TZmefWNVM>

d. <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>

4. Computation graph of ROS nodes and topics

a. *Without debug*



6. NVIDIA System Information used for validation

```

+-----+
| NVIDIA-SMI 440.100      Driver Version: 440.100      CUDA Version: 10.2      |
+-----+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+
|    0  GeForce RTX 207...  Off | 00000000:01:00.0  On |          N/A         |
| N/A   83C    P0     79W /  N/A | 2673MiB / 7982MiB |    93%      Default  |
+-----+-----+

```

```

+-----+
| Processes:                                GPU Memory |
| GPU      PID    Type   Process name                               Usage      |
+-----+-----+
|    0     2405    G     /usr/lib/xorg/Xorg                          499MiB    |
|    0     2552    C     ...space/devel/lib/darknet_ros/darknet_ros 1698MiB    |
|    0     2582    G     /usr/bin/gnome-shell                        206MiB    |
|    0     4142    G     ...AAAAAAAAAAACAAAAAAAAAA= --shared-files  112MiB    |
|    0     13219   C     /usr/lib/libreoffice/program/soffice.bin   97MiB     |
|    0     15278   G     ...uest-channel-token=16070837118886536314 42MiB     |
|    0     31213   G     /usr/bin/nvidia-settings                    3MiB      |
+-----+-----+

```