TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Kristjan Vilipõld 144128

COMPARISON OF OBJECT DETECTION ALGORITHMS PERFORMANCE IN DIFFERENT CONFIGURATIONS

Master’s thesis

Supervisor: Mairo Leier

PhD

TALLINNA TEHNIKAÜLIKOOL

Tallinn 2019

Infotehnoloogia teaduskond

Kristjan Vilipõld 144128

OBJEKTITUVASTUSE ALGORITMIDE TULEMUSTE VÕRDLUS ERINEVATE KONFIGURATSIOONIDEGA

Magistritöö

Juhendaja:	Mairo Leier
	Doktorikraad

Author's declaration of originality

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 has not been presented for examination anywhere else.

Author: Kristjan Vilipõld

13.05.2019

Abstract

Advances in technology have made the transportation sector look towards a newer future of driverless mobility. Tallinn University of Technology has also taken the development of a self-driving vehicle as one of its study platforms. The prerequisite for this vehicle is the ability of the robotic vehicle to perceive the surrounding environment. One of the challenges in machine engineering is object recognition.

The primary goal of the thesis is to find a suitable configuration for object recognition. This configuration consists of both hardware and software to which the camera is connected as well as the camera itself. There are two sides to everything - high accuracy of object recognition comes with the need of high computing power.

The second objective of this thesis is to launch the object detection in a real-life example, a self-driving vehicle developed by Tallinn University of Technology. A prerequisite for this task is to make sure that the proposed object recognition software works with the existing vehicle software and gives you the confidence to move on to the analysis.

The result is real-time detection of objects such as a pedestrian or a car from the camera image. It has to be noted that the prior relationship between the accuracy and the performance of object recognition comes out in the following thesis. Smooth and accurate object detection will need capable hardware, suitable video image and a well-trained object recognition algorithm.

This thesis is written in English and is 56 pages long, including 5 chapters, 31 figures and 1 table.

Annotatsioon

Tehnoloogia areng on pannud transpordisektori vaatama tuleviku suunas isejuhtivate sõidukite näol. Nõnda on ka Tallinna Tehnikaülikool oma üheks õppeplatvormiks võtnud isejuhtiva sõiduki arendamine. Vastava sõiduki eelduseks on robotsõiduki oskus tajuda ümbritsevat keskkonda. Üheks väljakutseks masinnägemises on objektituvastus.

Lõputöö esmaseks eesmärgiks on leida sobiv konfiguratsioon objektituvastuseks. Antud konfiguratsioon koosneb nii riist- ja tarkvarast, mille külge kaamera ühendatakse kui ka kaamera enda valikust. Igal asjal on kaks poolt - kõrge objektituvastuse täpsus tuleneb jõudluse arvelt.

Töö teine eesmärk on objektituvastuse arendus antud reaalelulises näites, milleks on Tallinna Tehnikaülikooli poolt arendatav isejuhtiv sõiduk. Antud ülesande eelduseks on veenduda, et pakutud objektituvastuse tarkvara toimib sõiduki tarkvaraga ning annab kindluse liikuda edasi liikuda analüüsi.

Töö tulemuseks on reaalajas toimiv objektituvastuse lahendus. Tuleb tõdeda, et eelnev objektituvastuse täpsuse ja jõudluse suhe tuleb järgnevas lõputöös välja. Sujuva ning võimalikult täpse objekti tuvastuse eelduseks on võimekas riistvara, kvaliteetne videopilt ning suure eristusvõimega objektituvastuse algoritm.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 56 leheküljel, 5 peatükki, 31 joonist, 1 tabel.

List of abbreviations and terms

ROS	Robot Operating System
GPU	Graphics Processing Unit
CPU	Central Processing Unit
YOLO	You Only Look Once
LIDAR	Light Detection and Ranging
MIPI	Mobile Industry Processor Interface
CSI	Camera Serial Interface
MP	Megapixel
POE	Power Over Ethernet
PC	Personal Computer
CUDA	Compute Unified Device Architecture
API	Application Programming Interface
USB	Universal Serial Port
GIGE	Gigabit Ethernet
SDK	Software Development Kit
HDR	High Dynamic Range
RGB	Red, Green, Blue
GPS	Global Positioning System
CV	Computer Vision
SSD	Single Shot Detector
RCNN	Region Convolutional Neural Network
VLP-16	Velodyne Lidar Puck 16
IR	Infrared Radiation
Point cloud	Set of data points in 3D space

Table of contents

List of figures

List of tables

Introduction

Transportation is irreplaceable in today’s society. Walking or running is relatively slow and does not get you very far, also the design of the modern city is perhaps inconvenient for it. Thanks to the invention of various propulsion methods such as internal combustion engine or a jet engine, we have cars, buses, planes and other forms of transport that move people and goods from one location to another in a short period of time.

The excellent engineering in the face of automobiles has led to a new problem. Statistics show alarming information - every 25 seconds someone dies in a traffic accident worldwide [1]. This is borderline unacceptable in the context of modern living standards.

While 90% of drivers think they are better than the average driver, we need to understand that we are actually pretty bad drivers overall [2]. We have evolutionary constraints - only front facing vision leaving over 150 degrees of view in the blind, relatively slow reaction rate, which is evolutionary optimized for the maximum speed of running. Emotion filled brain that sometimes works in an unpredictable irrational way and has a tendency to get distracted and fatigued [3]. The main cause of people dying in car accidents is caused by human error.

The convenience of transportation has changed how our economy works and how we live our daily life. One of the changes that are happening at the moment is automation of tasks which will cause acceleration of life [4]. The same applies also for transportation, which is part of the accelerative process of modernity. Thus, self-driving cars can help humans to cope with the new type of society and open a more convenient future. Being reliant on a human driver affects how transportation works - Removing the driver will allow more frequent schedules and smaller buses, which results in less waiting time for people. This is the ideal goal of the self-driving car, but of course work needs to be done in the field of engineering, programming and putting new technology into practice with the aim of integrating it into the society.

As stated before, humans have evident cognitive constraints. We have a slow reaction time compared to machines, this is caused by the human cognition and psychology. Having a computer manage driving decisions can increase the speed we drive since reaction times will be improved. If all the systems work in a predictable way and communicating with each other, the gap between the cars can be minimized and the flow of traffic will be more efficient and less polluting.

Having stated some of the possible effects of getting rid of the driver, we need to understand the challenges facing the task of replacing the driver. Making the car steer or accelerate is not a problem with the technology of today, with the use of actuators and servo-motors. One key issue being the decision making. Humans are very good at subjective and intuitive decision making. How much to turn the wheel and when to apply the brake, humans also evaluate a broader array of possible situations. The driving decision derive from the placement in the environment, the mission where to go and possible outcomes such as a kid running to a street. To make such decision computer needs to input data and machine learning.

The nowadays computer uses binary values and arithmetic operations to process information. The input needs to be in form of a usable digital data which can be acquired with the help of various sensors. We humans use mainly vision and other senses to operate in our daily life. The computer can be equipped with a camera for the same purpose.

This thesis is based on a Tallinn University of Technology student research project called ISEAUTO in which a self-driving vehicle is being developed. Various cameras have been evaluated for the fit of the developed system. Each camera has to be first tested and made sure it will work properly with the existing system.

1 Overview and requirements

An autonomous vehicle is a complex system with many components. To be able to make the recognition work in the domain of the ISEAUTO project, we need to understand the whole system and understand the constraints.

The ISEAUTO project got started in 2017 in Tallinn University of Technology as a learning platform for self-driving vehicles (Figure 1). The previous team has gathered an extensive amount of information and made a development decision for making of the vehicle. To understand why such decision were made and to also understanding that technology is improving at a rapid rate and what was good yesterday might not be good today.

Figure 1. A computer model of the ISEAUTO [5].

This thesis mainly sees the developed components from the object detection perceptive. The input of the object detection comes from the camera and LIDAR and the output is the classification and spatial information about detected objects. There is some basis information we need to understand for examining the possibilities for object detection and capturing, processing the sensor data. The ISEAUTO uses open-source software called Autoware, which features almost all the capabilities needed for autonomous driving.

The used software is responsible for perception, decision-making and action planning (Figure 2). A framework called ROS or Robot Operating System is used on top of which Autoware is built. ROS design consists of nodes, where each node is responsible for one task. A group of certain nodes is used for perception, another group for decision-making and lastly a third group for planning. Each node publishes data through topics which all the other nodes can listen to, irrespective of the purpose served by the node. Object detection happens in the Yolo (You Only Look Once) node. The Yolo node receives the images from the calibration node, which publishes the calibrated feed of the camera image. The object detection algorithm Yolo outputs a topic with the relative information about detected objects. This is being fused with the information from the LiDAR and presented on the 3D point cloud map, from which later decision-making information will be derived.

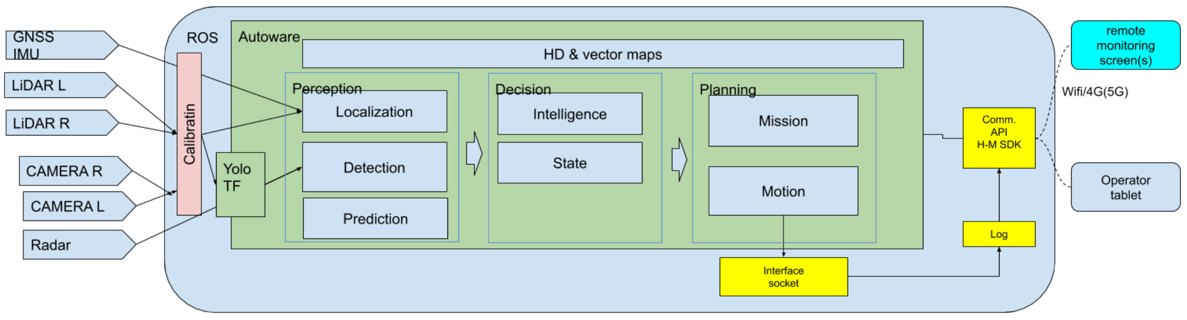


Figure 2. Software diagram of the ISEAUTO.

The ISEAUTO is driven by a master controller component (see Figure 3), which controls the various components such as electric engine or steering. The ROS will send the control input to the master controller through the interface socket. The PC running the Autoware and the master controller is connected with the Ethernet cable. At the moment all the sensors use Ethernet connection but during the development of this thesis, another approach will be tested in form a Camera Serial Interface. A joystick is also used for manual control of the vehicle and is connected through wireless USB dongle. In this thesis only camera and LIDAR sensors and the computer running the Autoware software will be covered - these are the basis for object detection.

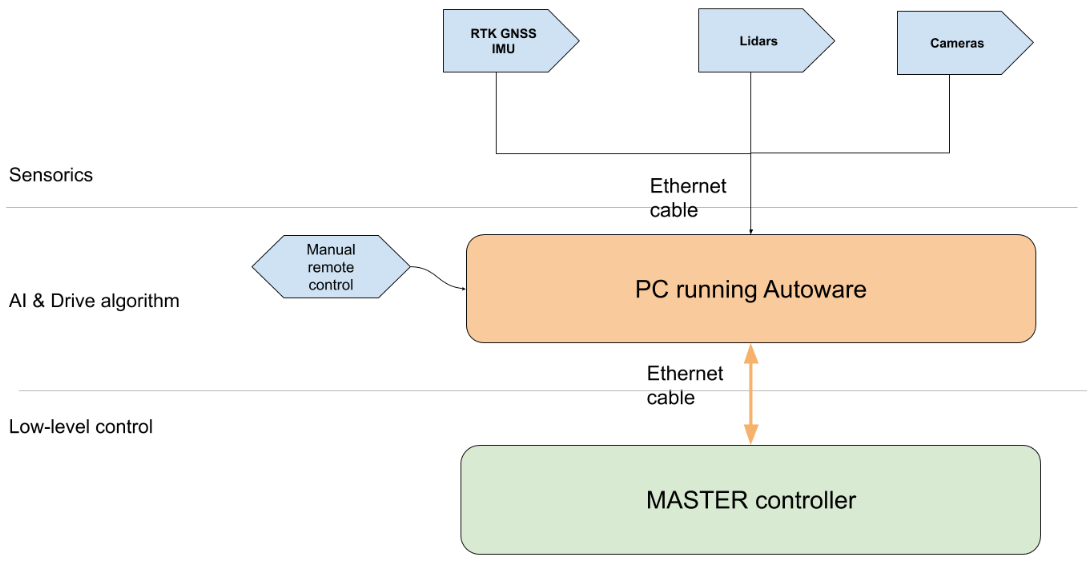


Figure 3. Overview of ISEAUTOs main hardware components [6].

1.1 Hardware

The vast amount of information that needs to be processed creates a need for very high computing power. Such computing resource can be implemented through extremely fast personal computers, other words PCs.

The ISEAUTO features such a PC. It has been purpose built from top of the class components (Table 1). It has the processing power many times more than the average home users' computer.

Table 1. Components of ISEAUTO PC.

Component	**Name**
Motherboard	Gigabyte X399 DESIGNARE EX - 1.0 - motherboard - ATX - Socket TR4 - AMD X399
Processor	AMD Ryzen ThreadRipper 1950X - 3.5 GHz - 16-core - 32 threads - 32 MB cache
Video card	2x ZOTAC GeForce GTX 1080 Ti ArcticStorm Mini - 11 GB GDDR5X - PCIe 3.0
Memory	4x Crucial - DDR4 - 16 GB - DIMM 288-pin - 2400 MHz / PC4-19200 - CL17 - 1.2 V
Storage	2x Samsung 960 EVO MZ-V6E1T0BW - SSD - 1 TB
PC Case	Thermaltake Suppressor F31 - Mid tower – ATX
Power supply	Corsair HX1200i - ATX12V 2.4 / EPS12V 2.92 - 80 PLUS Platinum - AC 100-240 V - 1200 Watt
Cooling system	Liquid cooling system CPU water block - EkWaterBlocks EK-Supremacy EVO Threadripper

The PC has two very powerful graphic cards. The high performance GPU or Graphical Processing Unit plays a crucial role in the ISEAUTO hardware-set. The GPU is used for neural network processing, which is used for object recognition. For these small and repetitive tasks, graphic cards outperform the CPUs. This is because GPUs are designed for parallel task.

NVidia is a major producer of GPU chips. Their chips are preferred by many home and professional users, because of one feature called CUDA or Compute Unified Device Architecture support. CUDA allows to turn each of the GPU's processor into a tiny CPU, to perform general-purpose mathematical calculations, achieving dramatic speedups in computing performance [7].

The ISEAUTO computer is extremely fast, but such speed comes at a price - meaning the actual cost of such PC, which is somewhere in the range of 5000 EUR. The lesser concerns are the size of the PC and the amount of heat it generates. One way is to not have one big processing unit but rather like few smaller brains that execute specific tasks. This is where the automotive industry is moving at the moment.

Nvidia has released a new platform for self-driving vehicle development. It is a new model of NVidia's previous line of self-driving oriented computers. It was released in 2018 and is called the Nvidia Jetson AGX Xavier.

1.1.1 Nvidia Jetson AGX Xavier Development Kit

The Xavier (Figure 4) is a compact super computer which has been developed by Nvidia and is addressed for the AI robotics domain. The heart of this module is a Tegra Xavier chip. It features a 8-core 64-bit ARM processor and 512-core GPU supplemented by 16GB of RAM. In all, it is said to be 20x faster than the predecessor Nvidia Jetson TX2 [8]. This has made it very attractive for all the biggest car manufacturers developing autonomous systems.

Figure 4. Nvidia Jetson AGX Xavier. [9]

Volvo has announced that it will use NVidia’s Xavier computer for its next generation of vehicles. It was said, that hardware-wise, it would be able to handle level 4 autonomy but it will be launched with level 2 capabilities, having the same capabilities as current Tesla models. Other manufacturers also stating using Xavier-powered systems include Uber, Volkswagen, Mercedes-Benz, and Audi. Toyota has also been collaborating with Nvidia for several years now and has announced that it would use Xavier to power the autonomous driving systems inside their future vehicles.

Seems reasonable that our student projects would also test the technology where a big part of the industry is moving on. If the Xavier would be able to replace the PC that is being used today, it will make a big difference in physical space and power usage.

The Xavier was obtained for the project in the end of last year. Autoware software is not supported for the Xavier platform. Xavier only runs a newer version of Ubuntu 18, but Autoware is supported only for earlier releases of Ubuntu. Autoware uses ROS kinetic which is mainly targeted for Ubuntu 16.04, nicknamed the Kinect [9]. Some of the software library were needed to be tailored fit for the latest version of the Ubuntu.

1.2 **Software**

There are generally two options when developing a software solution - using already available solutions or building your own. Building your own software can be time consuming but will allow more tailored fit. Using someone else’s available code usually costs and is usually not so fit for one’s specific purpose. To have the best of both worlds, one option is to use open-source software. Such software licenses allow you to download the source code and customize for your own vision. Open-source and code reuse will be the choice for making the ISEAUTO drive autonomously in a relatively short period of time.

For the developed ISEAUTO a software called Autoware is used. Autoware is an open-source software for self-driving mobility. It houses almost all the necessary capabilities of what a self-driving car should have, from perceiving the environment to an actual action command for the vehicle (Figure 5). The real world is mapped with various sensors, such as camera or LIDAR. The data from different sensors is being processed for both detection of objects and for localization. Each detected object will be also mapped to predict the future movement of each object. After that all the relevant information will be fused and Motion Planner will base decision on the basis of fused data. The outcome of Motion Planner is actual instructions for driving the vehicle.

Autoware is focused more on city driving rather than highway driving. While it is able to function on a highway, it is limited to the maximum speed of 65 km/h. Safety is not certified and must be used under owns risk. Autoware is based on software used in field of robotics called ROS.

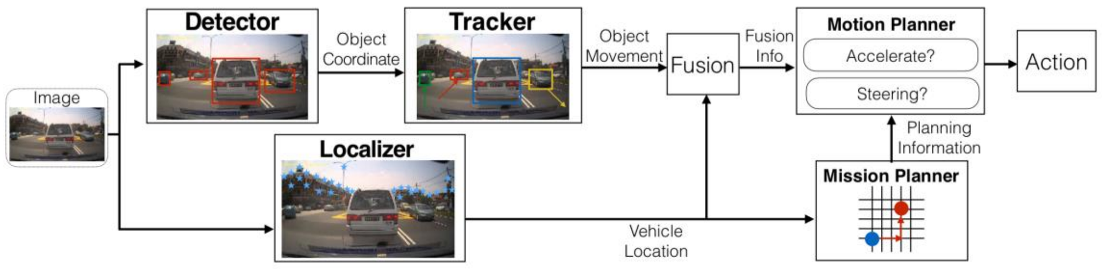


Figure 5. The conceptual flow of Autoware from perceiving to an actual action.

ROS is not an actual operating system. ROS is free and open-source middleware, meaning it houses a set of software libraries and tools that help you build robot applications. It consists of drivers, development tools and algorithms to accelerate the development time. Without ROS there would be a need to write drivers, communication framework and algorithms for perception, navigation, motion planning, logging and error handling.

The big strength of ROS is connection nodes together. Each node is a small C++ or Python code which handle small subset of tasks, for example reading a value of sensor. Nodes will share information with each other through topics. Each node can also subscribe to other topics.

1.3 **Cameras**

Cameras are used to visualize the world similarly than we, humans do. Cameras use sensors to capture the light and record the image. In order to capture an image clearly a light proof camera body is required. This ensures that only the light that is wanted will enter the camera in order to be recorded. Inside this lightproof body, there is a light-sensitive sensor. This sensor will react to light when it enters the camera, recording the information and producing a digital representation called an image.

A lens is attached to the front of the body of the camera. This allows the light entering the camera to be controlled. The light is focused by the lens to create a sharp upside down image on the sensor.

1.3.1 Requirements for the camera

There is a vast options of different sensors and interfaces of cameras. Choosing the right one is the key for receiving the best picture, at a real-time speed and staying in the frame of the budget. The main requirement is that it should be supported with Autoware software [10, 11].

The second requirement is the hardware connection. The information from the sensor needs to get somehow to the computer. Taking regard of the connection interfaces the Nvidia Jetson AGP Xavier has, three likely interfaces are selected for further choosing – USB (Universal Serial Bus), Ethernet and CSI (Camera Serial Interface). In the following Table 1, the main attributes of each interface are listed:

Table 2. Comparison of possible camera interfaces.

Attribute	*USB*	*Ethernet*	*CSI*
Maximum bandwidth	5 Gbit/s	1 Gbit/s	6 Gbit/s
Maximum cable length	up to 5m	up to 100m	up to 50cm
Low-level control	No	No	Yes
Connection limits	Up to 127 cameras	No limit	Up to 16 cameras
Hardware vision pipelining	No	No	Yes

USB is one of the most popular connection interfaces. USB cameras usually are "Plug-and-play" and are easy to integrate, but will use CPU time due to USB bus which will impact the quality if the stream during high CPU loads. USB can work up to 5 metres in distance, which will be enough in our case.

Using Ethernet cable, the signal transmission can work over long distances (802.3af standard POE or Power-over-Ethernet can usually go about 100 meters). There is no limit for how many devices can be connected as long as the network is capable of housing all the IP-addresses. This will be reflected as a minus since the Internet Protocol can suffer from network interruptions or delays. Each device needs to receive an IP or create its own DHCP network. PoE networks can supply up to 350mA at the camera end so may struggle with some needier IR and PTZ cameras.

The CSI is optimized in terms of CPU and memory usage for getting images processed and into memory. It can take full advantage of hardware vision pipeline. The CSI supports only short distances from Xavier only (maximum 50cm for the cameras we use) unless you use serialization systems (GMSL) which are immature and highly custom at the moment. Camera Serial Interface will give the access to low-level control of the sensor/camera.

The constraint of the cable length limit can be overcome with modular architecture. A computer instance could operate near the cameras and receive the data through the CSI-2 interface and publish to other nodes through ROS topics via Ethernet cable. If the computer instance is fast, it could also run the deep learning algorithm and publish the feed of the detected objects in addition to the video stream. If the bandwidth is limited, then publish only information about the objects class and location.

The environment in which the self-driving bus will be operating is outside, on the streets. The various conditions consist of temperature, humidity and over- and underexposure of light, in any case, it is highly necessary to take into account as many factors as possible.

The placement of the cameras (Figure 6) at the moment might get dirty. Forward-facing camera is placed as high as possible so the dirt thrown up by the wheels or others cars will affect the cameras as less as possible. A windscreen wiper will take care with the rain, snow and other impurities that might affect the view of the camera. It is not implemented at this time at the moment.

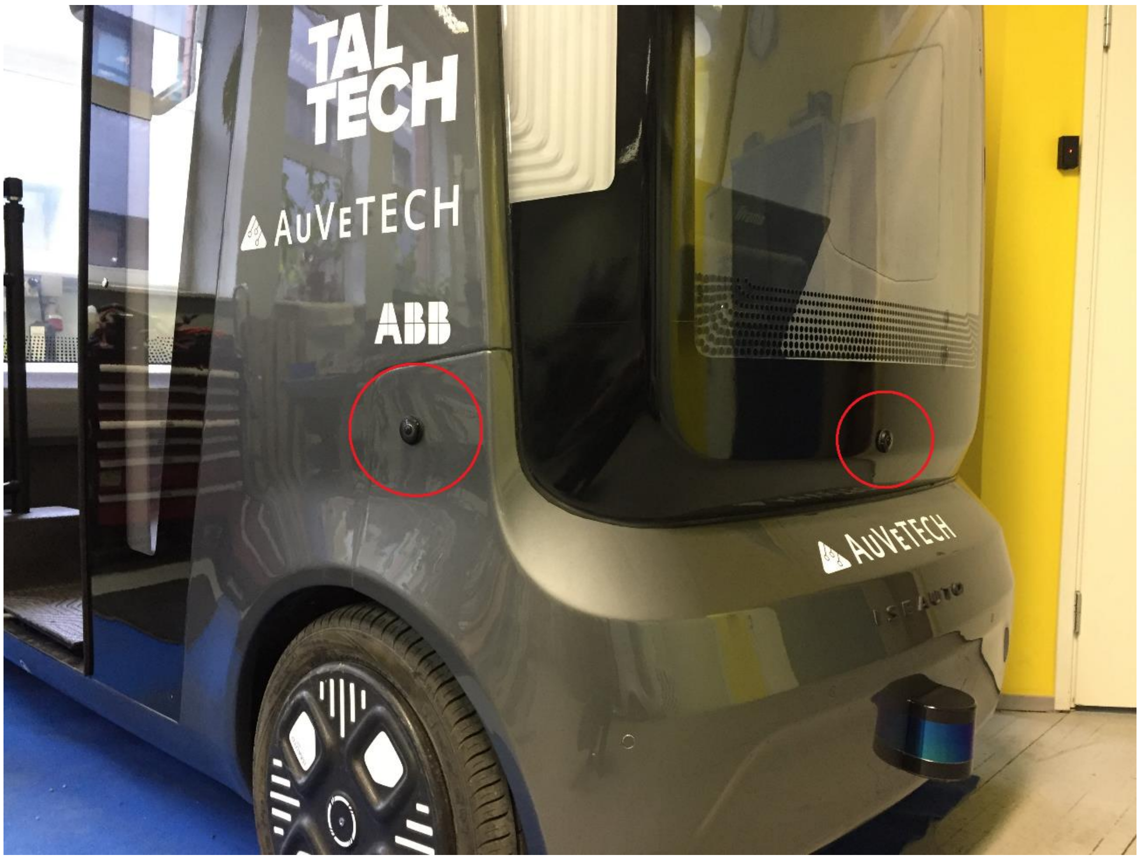


Figure 6. Placement of the cameras.

On slow speeds and distances the camera can be replaced with cameras since there is no need for seeing very far. It is important to use Global Shutter cameras, which captures true images and not blurred images. With global shutter the image is read in a full exposure and not from line to line. This comes visible when objects or the sensor is moving at a fast phase as seen on Figure 7. Comparison of rolling and global shutter [12].

A rolling shutter captures an image by using a process in which it repeatedly scans either vertically or horizontally – not all of the parts of the image are recorded at the same time, but when you review the footage it is displayed as one entire image. Rolling shutter can cause images to look skewed, blurry or warped.

Given similar specifications, the global shutter equipped image sensor is much more expensive than a rolling shutter sensor.

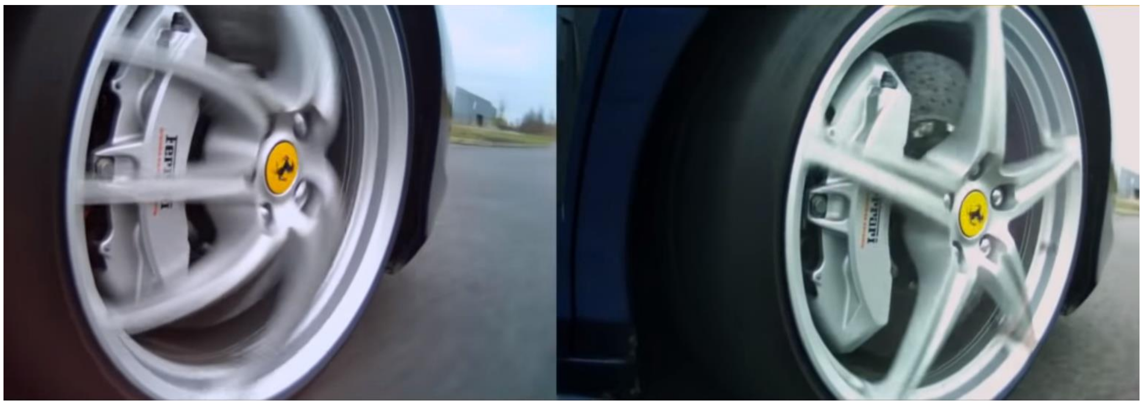


Figure 7. Comparison of rolling and global shutter [12].

On the left rolling shutter and on the right global shutter. If you try to compare the location of each spoke of the wheel you can see the effect of the distortion.

1.3.2 FLIR Blackfly S Colour 3.2 MP GigE Vision (Sony IMX265 sensor)

FLIR names all the models in a name of an insect. FLIR's most popular camera line, The Blackfly S (Figure 8), supports wide range of sensors. For our purpose the cost-optimized Sony IMX265 sensor was chosen. The FLIR camera supports PoE or Power-over-Ethernet, which would remove the need for an external power source - the Ethernet cable that is used for the data connection can also be used as a power source. The switch or network adapter that the camera connects must be PoE enabled.

Figure 8. FLIR Camera [14].

All the Blackfly S models share the same size 29x29mm design makes upgrading existing systems to faster or higher-resolution cameras easy. It has a durable casing and is IP67 certified, in other words waterproof to some extent.

The connection interface is either USB3 or GigE, which in our case is the GigE example. The camera is supported by Spinnaker SDK software which allows easy parameter manipulations on a PC.

Image sequencer allows you pre-configure image parameter sets, such as gain, exposure and resolution and automate the transition from one set to the next. The colour transformation allows for more true-to-life imaging.

Table 3. Technical data comparing the cameras.

	FLIR BFS-PGE-31S4C-S	Basler acA2040-35gc	Leopard IMX274CS
Sensor	IMX265	IMX265	IMX274
Sensor type	CMOS	CMOS	CMOS
Resolution	3.2 MP	3.2 MP	8 MP
Dimensions	2048x1536	2048x1536	3864x2196
Frame rate @ max res	35 FPS	36 FPS	60FPS
Global shutter	Yes	Yes	Yes
Interface	GigE with PoE	GigE	MIPI CSI-2
Lens mount	C-mount	C-mount	CS

1.3.3 Leopard IMX274CS MIPI CSI-2 Camera (Sony IMX274)

The Leopard Imaging cameras (Figure 9) were the only option with MIPI CSI-2 support during the time when Xavier was released. There are several other options on the market during writing this thesis, but when the cameras were bought, these were the only option. The IMX274CS which I will reference in this thesis as the CSI camera uses Sony IMX247 sensor, which features a 4K resolution image.

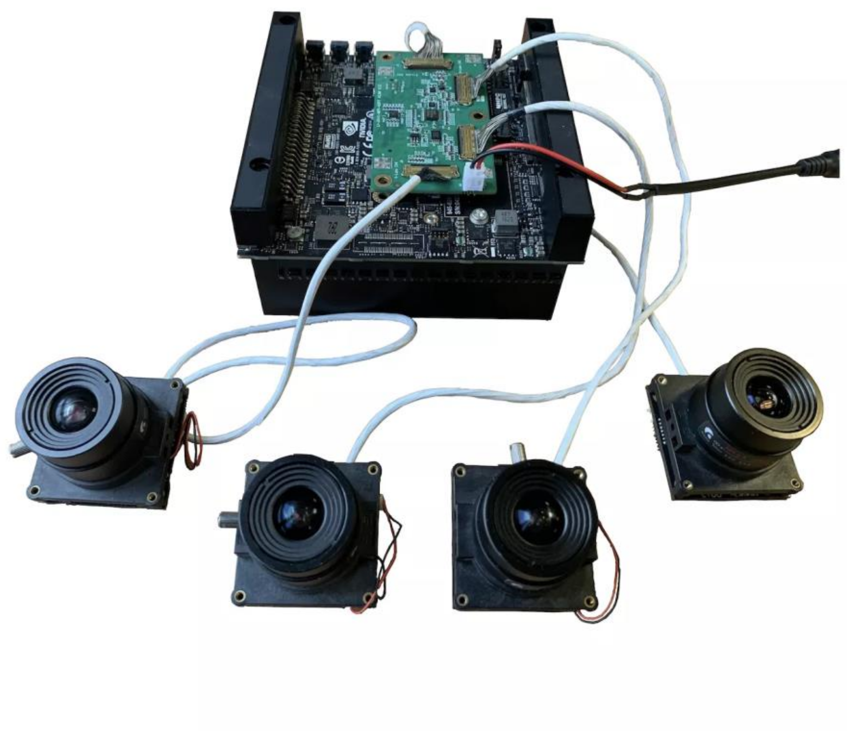


Figure 9. Leopard cameras with Nvidia Jetson Xavier [15].

The MIPI CSI-2 has a constraint concerning the cable length. While the low power consumption at 3.3V is excellent for things with a battery, e.g. smartphone - it just has not enough power to send signals through a long cable (resistance is too great).

Jetson AGX Xavier can capture images from RAW Bayer sensors. Multiple sensors can be connected via the CSI interface. However, the current software version is validated to capture images from one sensor at a time.

1.3.4 Basler acA2040-35gc GigE (Sony IMX265)

The Basler was used on ISEAUTO before Author started writing this thesis. The camera worked well indoors and outdoors, the quality of the picture suitable and all the necessary drivers were installed. The camera was chosen previously for the ISEAUTO because it was supported by Autoware and already had a driver set [17].

Figure 10. Basler Pylon camera [16].

The Basler Pylon camera is very similar to the previously mentioned FLIR camera and features the same sensor. The dimensions of the camera are also very similar and it seems that the only downside of this camera is missing PoE compared to FLIR model.

1.4 **Testing of the cameras**

To confirm that all the cameras work in different environments the cameras were tested with the supported software.

The first step was to confirm that all the cameras work. Manufacturer software and drivers were used in Ubuntu 16.04 operating system. All three cameras had no issues when installing the software and video feed streaming. After that the cameras were tested on Xavier kit which featured Ubuntu 18.04. The results stayed the same with manufacturer's software.

The Author has contacted the manufacturer and received a reply concerning the over-exposure. The manufacturer said that they will try to make a fix and send a new firmware update. The fix has yet not been released, while this thesis was written.

The over-exposure means that the sensor is over exposed with light due to the shutter remaining open too long. This leads to the image being almost fully white and very much unusable.

In 2016, a Tesla Model S was involved in a fatal accident while the self-driving system was engaged. The cause of accident was a result of system not recognizing the white side of a trailer against a brightly lit sky. The vehicle drove right into the trailer never even applying the brakes. Such errors can be tackled with the use of LiDAR.

1.5 ROS Drivers

It is no surprise that the cameras work with the software that the manufacturers provide but if we want to use the cameras they need to work with our choice of software. ROS needs drivers to make use of the O/S video drivers and its own driver to publish a node out of the camera.

The Basler camera had a driver in Autoware and terms of Autoware was plug and play. The ROS driver for the Basler camera was already installed and the camera was listed in the Graphical User Interface or GUI of Autoware.

The FLIR camera was not supported by Autoware officially but there was an open-source driver available for ROS [18]. The driver needed some modifications because the source image was in black-and-white on the first try. The camera.launch file needed additional parameter specified for the correct colour format. The default value was mono8, which produced the grayscale image. After changing the image colour format value to BayerRG8, the camera stream worked.

The Leopard IMX274CS CSI camera did not have a driver and a driver was needed to be developed to work in ROS/Autoware. The camera driver works thanks to ROS Gstreamer node called the GSCAM.

1.5.1 Building a ROS driver with GStreamer

The feed from the camera sensor can be in various formats and sizes. The Analog-digital-converter inside the camera will convert the infinite number of colours to finite information about the colour. There are numerous ways to encode the value of each pixels colour. The video feed from the CSI camera features NV12 colour encoding for reduced bandwidth and not the straightforward RGB expression of pixel colours, which ROS understands. Thus NV12 to RGB conversion needs to be done.

This is done with the help of Gstreamer. GStreamer is a framework for creating streaming media applications. Its objective is to manipulate and pipeline the video to from the input source to the output called the sink. The Gstreamer is CUDA enabled and will allow to take the full benefit of the GPU cores the Xavier and Xavier Nano.

There are also other similar software solutions for this task but gstreamer is the mostly widely used and, like stated before, has the benefit of faster processing through the GPU with CUDA. The Gstreamer is very well documented and has an library of hundreds of filters, converters, resizes and so on. The gstreamer also supports almost every media format known.

The middleware, ROS, also shows how choosing an widely used option has it benefits. ROS has a prebuilt package which features gstreamer compatibility with ROS. It is able to publish the feed both as ROS camera or ROS image transport. The ROS camera feed only consists of the raw video stream, but the image transport creates topics for compressed video size for better performance.

The pipeline can be constructed of available components. Each components is like a lego brick and the pipeline is created with connecting many of these blocks. Each block is used for specific task, for example format conversions, filtering or streaming. Note that every block has an input called the sink and the output called the source. This has to be defined on every block except the first block, which only has the output in terms of the pipeline and the last block, which only inputs the feed. Each block has rules defined which kind of formats it accepts.

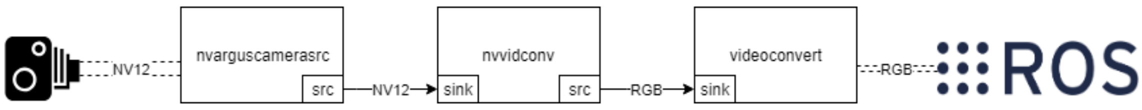


Figure 11. The constructed gstreamer pipeline for the driver.

For our case, the pipeline (Figure 11) has only one main task in hand which is to change the colour format. The first block called *nvarguscamerasr*c gets the video from the camera. The block features many more parameters for controlling the camera but at this point the main interest is in receiving the video stream. The *nvvidconv* block is the engine for video format conversion. *Videoconvert* is needed element to ensure the compatibility between *nvvidconv* source and the ROS camera stream. This element will just pass frames through for ROS, without performing any job. An example of a constructed pipeline variables is shown in Figure 12.

```
GSCAM_CONFIG="nvarguscamerasrc sensor-id=$(arg sensor_id) ! video/x-
raw(memory:NVMM), width=(int)$(arg width), height=(int)$(arg
height),format=(string)NV12,framerate=$(arg fps)/1 ! nvvidconv !
videoconvert"
```

Figure 12: Code example of the pipeline

This pipeline will allow us to use the CSI camera in ROS. The created driver can be used in future for more in-depth manipulation of the camera feed.

1.6 **Experimental Result**

After testing the cameras with the software provided by the manufacturer and making the cameras work in Autoware, it is time to pick a winner. While it is not a surprise that all the cameras do function in Autoware software, the surprise was how much effort was needed for getting the CSI camera working. And had it been that the effort was worth it - the CSI camera also performed the worst in real life testing. It was over-exposed during daylight and underexposed in dark conditions.

The CSI cable itself also houses a terrible constraint of only up to half a meter in length. Taking into account all the characteristics of the camera, the Leopard IMX274CS fails to meet the set of requirements. A future proposition can be e-Con Systems camera called the e-CAM130_CUXVR, but only if the half a meter cable length can be acceptable.

The Basler camera and the FLIR camera carry the same sensor. The Power-over-Ethernet will give a second iteration of the ISEAUTO a more clean wiring since the camera will be powered by the ethernet cable.

While the Basler camera was the easiest to setup and previously tested. FLIR camera provides a clearer image at higher resolution. It also features PoE and it will be an upgrade, not a big one though, over the Basler camera.

Taking into account that this is an educational project and Basler camera has already been used as the choice of camera, in this thesis the FLIR camera will be used. The FLIR camera will provide the video feed for object detection.

2 Calibration of camera and LIDAR

Whenever using a sensor in autonomous driving systems, we need to make sure that the sensor installed is actually giving out the right and usable information. In our case, the camera should have high precision and resolution, low or no distortion. Most of the cameras do have some sort of distortion and to make it valid and comparable with real world parameters we need to calibrate and compensate for the suffered distortions.

The calibration is an important part for receiving accurate data from the sensors. The self-driving systems needs to have valid information to make correct decisions. Calibration will give the exact location of the sensors. Using the location of each sensors and positioning it relative to other sensors.

Since light enters the camera through a lens instead of a pinhole, most cameras suffer from some kind of distortion. This kind of distortion can be modelled and compensated with the help of calibration. Calibration will set the parameters for the so-called "compensation funnel" that the information will be processed through. This will recalculate the original data to an output of data which is comparable with other sensors and the real world representation.

There are two types of calibrations - extrinsic and intrinsic.

Intrinsic calibration is used for correcting the lens distortion. Intrinsic calibration is related only with the camera and does not depend on various parameters of the outside world. The calibration will result with parameters which will be used for removing the distortion. These intrinsic parameters are the attributes that are related only with the image sensor or with the lens of the camera, like focal length.

Extrinsic calibration is the process of finding the orientation and position of the sensor relative to some other factor or object. It is also called the global localization of the the sensors and it can be solved with supported algorithms.

In robotics field, to be able to fuse measurements from different sensors (in our case camera and LiDAR), all the sensors measurements must be expressed with respect to a common frame of reference, which requires knowing the relative pose of sensors.

2.1 **Lidar overview**

Lidar of Light Detection and Ranging or LIDAR is a sensing method that uses light pulses to measure the surrounding environment. In very understandable terms, LIDAR shoots out light beams and measures how long it takes for the light energy to bounce back. By having a constant flood of millions of light beams shooting out in each direction, the measurement of each beam allows to visualize the surrounding world in 3D. This makes the LiDAR an active sensor and can work regardless of the natural illumination. Bats and dolphins use something similar to a LIDAR - they create soundwaves and wait for the echo to bounce back.

The LIDAR provides quite accurate depth information. The precision of each points distance can be determined within around +- 3 cm accuracy and not only is it good for getting a precise distance, it also allows to see far of over 100m going with the top range models.

The LIDAR's have quite a narrow vertical field of view, around 15-30 degrees on the most popular models, meaning it can see all around 360 degrees but not so much up or down. To cover all the possible blind spots, there is a need to use several sensors. Most car manufacturers experimenting with self-driving vehicles have usually more than one LIDAR placed in various locations of the car. The most typical being on the top of the vehicles roof.

The ISEAUTO uses two Velodyne VLP-16 (Figure 13) LIDAR's and calibration between the two LIDAR's using previously mentioned method was already done by a member of ISEAUTO's team.

Figure 13. The Velodyne Puck VLP-16. [20]

The Velodyne VLP-16 has 16 vertically position transmitter-receiver pairs and there comes the name VLP-16. The “Puck” as it is called, is able to measure points 360 degrees around the LIDAR with a frequency up to 20Hz. The vertical field of view is 30 degrees. 100 meter range and up to +- 3 cm accuracy [19, 20]. Compared to the previous LiDAR models this has a small footprint, is lightweight and has a low power consumption of around 8W. The Puck is rated IP67.

These impressive stats all come for the price, one of such sensor is around 7000 euros making it the most expensive sensor on the bus - and the ISEAUTO has two of those. Both are placed on top front-end of the ISEAUTO.

These points create the point cloud map of the surrounding environment. These points play a crucial role for the computer to understand and in end base its decision from.

2.2 **Camera calibration**

This is the raw image from the camera:

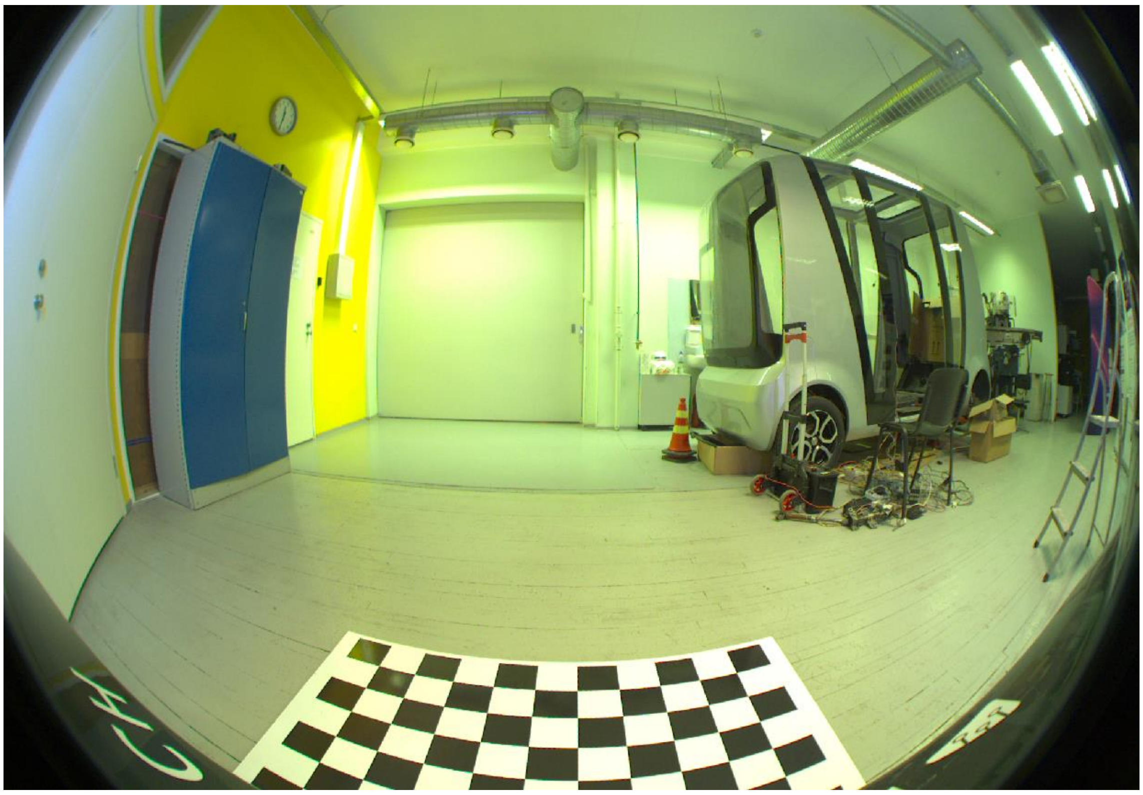


Figure 14. Distorted image from the camera. Notice the curved lines that should be straight.

The intrinsic calibration is done by using the autoware_camera-calibration package and is a spin-off from the official ROS calibration tool [21].

Radial distortion causes straight lines to appear curved and becomes more noticable at the farther points from the centre of the image. As you can see we got a bad case of distortion from our fisheye camera.

Radial distortion can be represented as follows:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

As we can see we need to get five parameters, known as distortion coefficients.

Camera calibration is needed for two reasons:

1. Finding the relation between pixels and the real world.

2. Fixing distortion.

Calibration you may also determine the relation between the camera's natural units (pixels) and the real world units

The camera Author is using has a negative distortion that is produced by the “fisheye” lens. The image has to be corrected to get an image that is the same in the middle as on the sides.

ROS uses openCV calibration tool (Figure 15). The tool allows you to calibrate your camera with the use of a black-and-white 10x7 chessboard. The main idea is that you move the chessboard around the frame and the calibration tool gathers information to calculate the needed distortion matrix. This matrix can be used to rectify the misplaced pixels to the correct position.

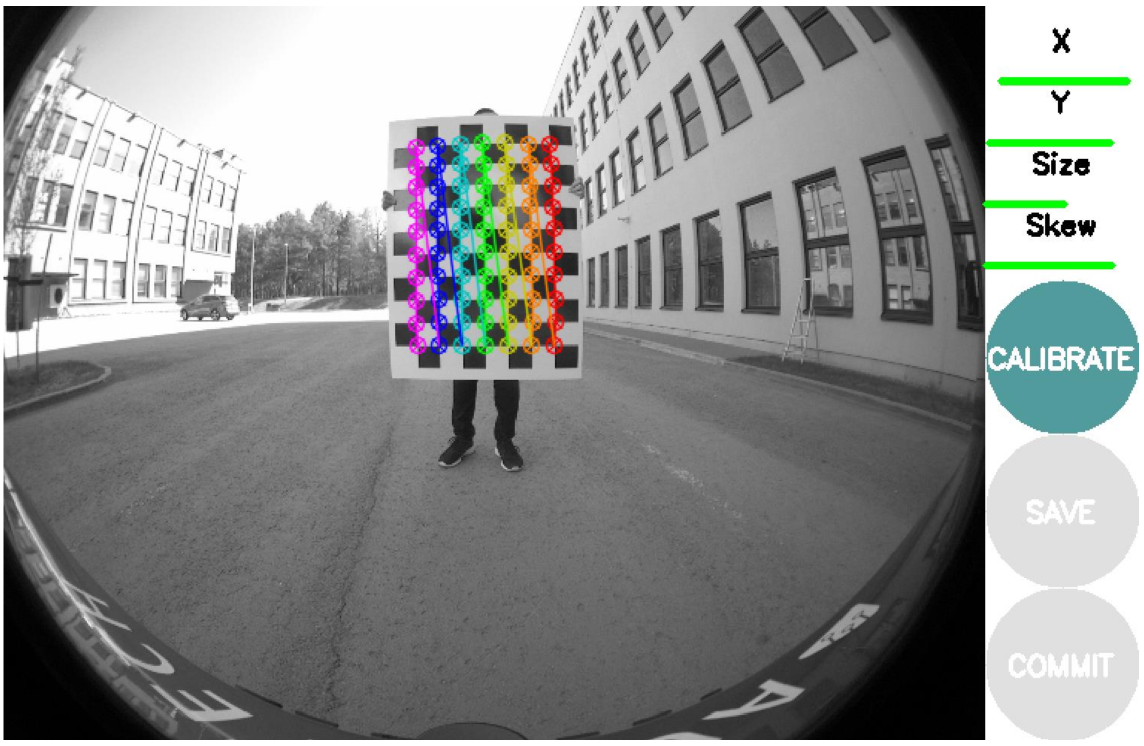


Figure 15. Screenshot of the calibration tool.

Calibration is relatively easy, a chessboard the size 100x100mm 7x10 squares is used to calibrate the camera. Holding the chessboard towards the camera and moving about until all the bars turn green.

To match the X value you need to move the board from left to right in the field of the camera. The same goes with the Y value but only in its respective y-axis or up-to-down.

Size is calibrated through moving closer and further away from the camera so that at the nearest point the chessboard almost fills the whole view. Skew is when the image bends diagonally in one direction or another as a minor manifestation of the wobble phenomenon.

Pressing the save button will save the output file containing the intrinsic parameters to rectify the image.

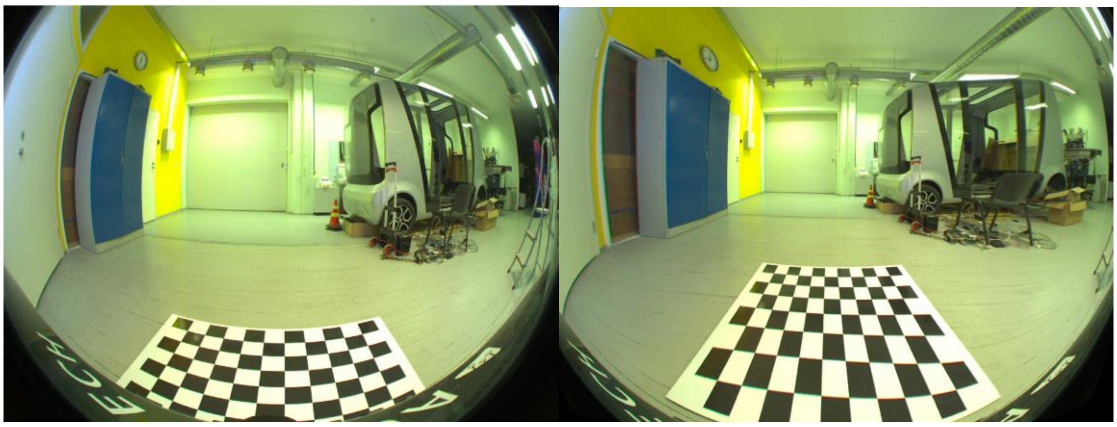


Figure 16. Before and after calibration. Lines that should be straight are straight now.

As you notice now all the lines on the chessboard are straight as it is in true and the proposed multiplier to stretch each pixel for alignment is true.

2.3 Calibration of camera and LIDAR

The extrinsic calibration of the camera and the LIDAR's is done by finding corresponding points in both the image and the previously fused point cloud. The Camera and calibration is done with the help of Autoware package named *autoware_camera_lidar_calibrator* and needs to have the following ROS nodes running: *clicked_point, screenpoint* from *rviz* and *image_view2* package [22].

The input for the image_view2 is the camera calibration file *calibration.yaml* that was done previously, camera image topic */pylon_camera_front/image_raw* and camera info source */pylon_camera_front/camera_info*

The Input for the rviz is the calibrated point cloud topic *lidar_front*

The calibration is actually done on two different screens - the *image_view2* screen and the *rviz* screen (Figure 17) One screen is for the camera image and the other one is for the LIDAR point cloud.

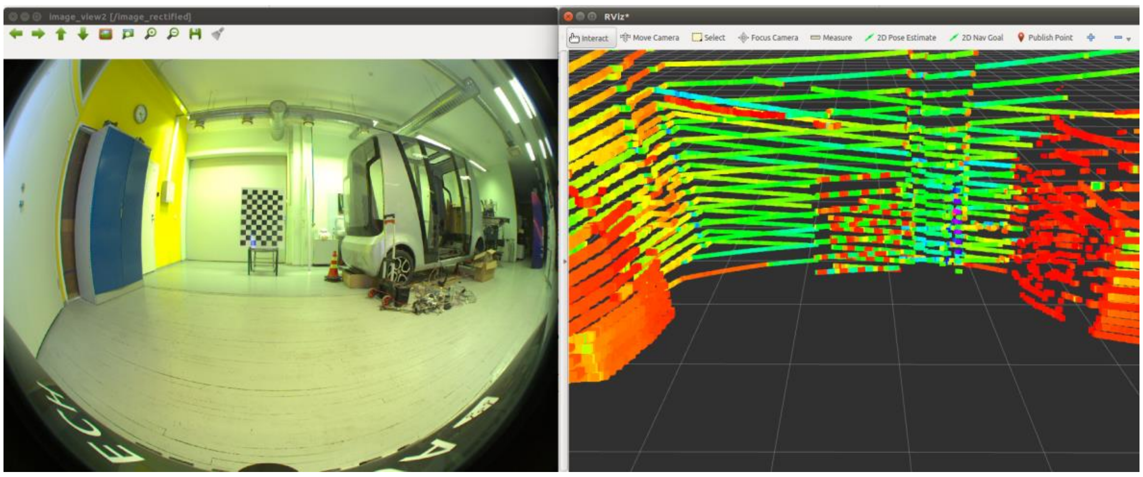
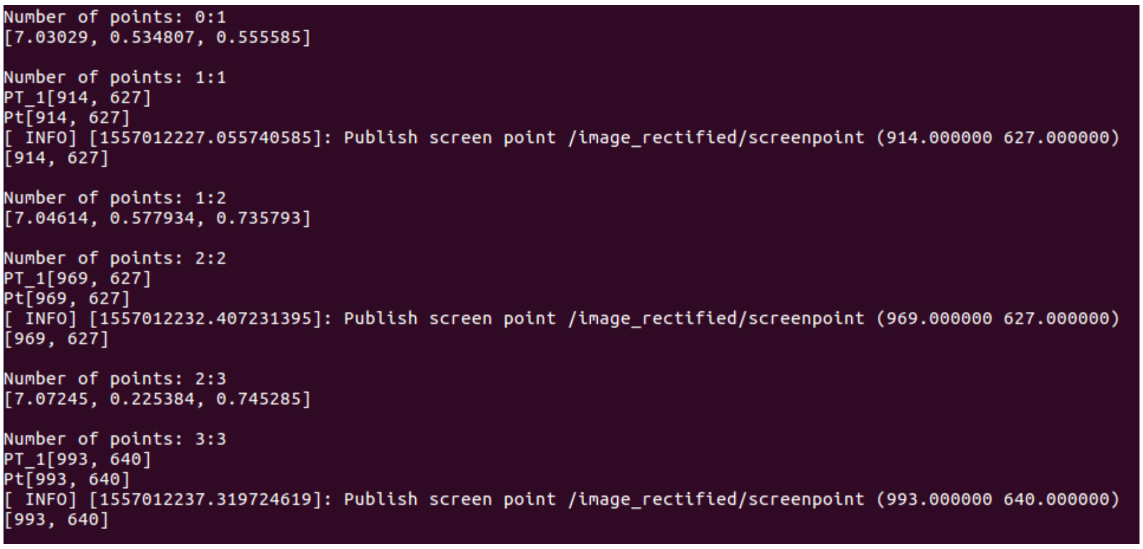


Figure 17. The view from both image_view2 and rviz point cloud. The image from rviz shows how the chessboard is seen by the camera and LIDAR. Green colour is for white and the red is for black squares on the board.

The needed parameters are the calibration file from the camera intrinsic calibration and the image source *image_src:=/basler_front_camera/image_raw*.

The rest is quite simple. You compare the two feeds and find a point that is visible both on LIDAR point cloud and the camera image. You use the computers cursor to click on that point in the *image_view2* screen and after that you move to the *rviz* screen, click “Publish Point” in the upper corner and select the same spot on the point cloud.

This same procedure has to be done eight times more, so nine times in total - both on the image and on LIDAR projection. The count of the points can be seen from the terminal screen, also this view can be used to verify the selection of each click. (Figure 18)



```
Number of points: 0:1
[7.03029, 0.534807, 0.555585]

Number of points: 1:1
PT_1[914, 627]
Pt[914, 627]
[ INFO] [1557012227.055740585]: Publish screen point /image_rectified/screenpoint (914.000000 627.000000)
[914, 627]

Number of points: 1:2
[7.04614, 0.577934, 0.735793]

Number of points: 2:2
PT_1[969, 627]
Pt[969, 627]
[ INFO] [1557012232.407231395]: Publish screen point /image_rectified/screenpoint (969.000000 627.000000)
[969, 627]

Number of points: 2:3
[7.07245, 0.225384, 0.745285]

Number of points: 3:3
PT_1[993, 640]
Pt[993, 640]
[ INFO] [1557012237.319724619]: Publish screen point /image_rectified/screenpoint (993.000000 640.000000)
[993, 640]
```

Figure 18. Terminal screen registering each chosen calibration point.

After choosing all the suitable points, an output file will be saved which can be used with Autoware's Calibration Publisher which publishes the transformation between the LiDAR and camera. This file contains both the camera calibration values and the camera-LIDAR calibration outcome.

2.4 **Sensor fusion and object projection**

Sensor fusion is a process of combining data from various sensors into one signal. The purpose is to make the signal more robust or to increase the quality of information. In ISEAUTO case the sensor fusion is needed to combine the data from the camera and object detection with LiDAR point cloud representation. Since the Autoware software supports LiDAR and camera fusion, the main focus will be the fusion between the LiDAR and camera.

LiDAR uses reflections to localize objects quite accurately, but it cannot differentiate objects based on their colour. The problem is also with objects in the distance which might be recognized with only few points, making object detection by the shape inaccurate. Recognition of the objects can be done with the help of the camera. An image is processed through deep neural network that detects objects in the image. The detected objects are then mapped on the 3D LIDAR's view.

Fusion of camera and LiDAR can be done in two ways — fusion of data or fusion of the results. Fusion of data is the overlapping of the camera image and LiDAR point cloud so that we get depth information for the pixels in the camera image. Fusion of results is where, say we do object detection in the camera image and in the LiDAR point cloud separately, and fuse the results to increase our confidence. In this case we will be conducting the fusion of the results case.

The computer understand the world in binary values. Perception of the world simplifies down to distance and colour. Using our two eyes we receive a "vision" from distance and colour. The colour is perceived by each reflecting light wave and distance is calculated by our brain from comparison of these two images. The human eyes deficit is having external conditions for working - needing a source of light to receive data of the environment.

Thus we need to solve 2 problems - how to receive distance and placement of each point in computer representation of the environment.

The human neural net derives distance through vision and the process of triangulation. We use the same implementation as stereo cameras with two fixed light sensor side-by-side. Possible alternative options are sending out light-, sound- or radio waves and timing the return of the pulses

The following nodes are being used to publish to the data to the point cloud map. Some of the details can be quite detailed but in the hopes that someone actually will find my thesis helpful in future, will be specific around the topics that each node uses.

ROS packaged called the i*oints image* uses the calibration to merge the data (Figure 19) from the point cloud with the image data. Topic being published is called the /*points_image* topic.

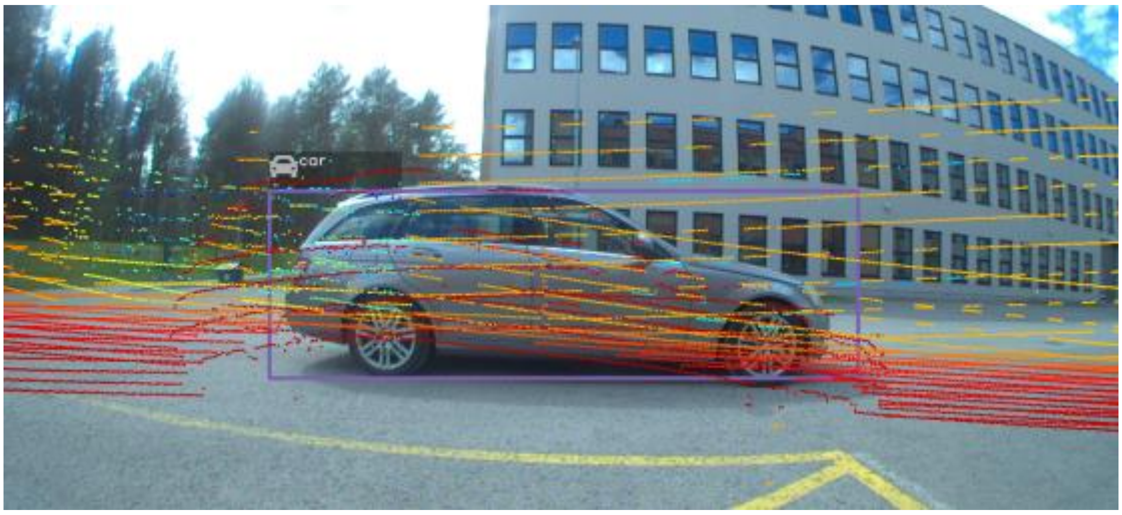


Figure 19. The camera image is being overlayed with the LiDAR markings.

Range fusion node called the *range_fusion* inputs the data from the points2image node. The points2image merges the LIDAR and camera data. The *range_fusion* also uses the detected objects from the image. This gives us the distance of the detected objects.

Object tracking algorithm called KLT tracking (meaning Kanade-Lucas-Tomasi tracking) is named as a ROS package called the *klt_track* node. The node inputs the original image topic called *image_raw*, the *range_fusion* output topic and publishes it for object reprojection.

Object reprojection is the process of projecting an image from the object detector to the point cloud map. On Autoware it uses the *obj_reproj* node and inputs data from the *klt_track* node, *tf*, the camera's *projection_matrix*, the camera's *camera_info* and outputs the detected objects on to point cloud as bounding boxes. To topic that is being published is called the *obj_label_bounding*.

This leaves us with the result - data of *obj_label_bounding* and gives out information about other objects. The data is used by other Autoware nodes that do the prediction and decision making on how to react and what to do. Now that the objects are projected on the point cloud (Figure 20) and the information and location of the objects is usable by other nodes we move on to analysing and trying to understand how to make the object detection as best as possible.

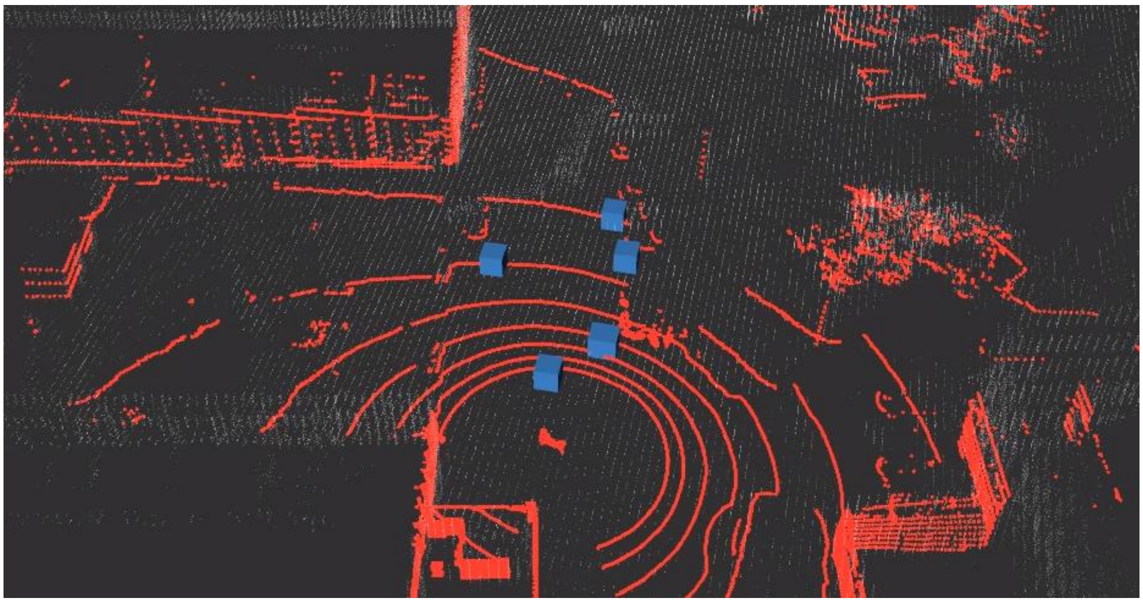


Figure 20. Detected objects projected in the point cloud map.

The result of calibrating the camera with the LIDAR an information usable for planning the motion. Without the calibration, the detected objects may be mapped in a different location as in the real world. The calibration overall was a success and first impressions seemed accurate. This is also something which could be explored further how much deviation there actually is between the objects in real life, but for a proof on concept, subjective validation was used. The objects on the map were in around the same place as in real world. As object detection is the main part of this thesis, the main concern is detection of correct objects in real time.

3 Object detection

Detection of objects is a natural ability for humans and other animals. We move about in the world effortlessly - avoiding collisions, avoiding threats, finding something to eat, and so on. The task of object detection is used in almost every step in person’s day. This comes so easily from the human brain and with the years of training, the knowledge of different objects is vast - it takes only a quick glimpse to recognize and name a certain object.

For the computer this is definitely not a natural ability. The computer just sees a set of pixels with certain colour. This is just a mess-up of colours, just like a baby’s painting on the kitchen refrigerator. Understanding each image has been a challenging task up until now thanks to deep learning and neural networks.

3.1 Basics of neural networks and deep learning

Object detection is easy - an input image goes through a deep learned neural network and objects are detected. Just kidding - understanding how object detection work is not an easy task. Let's start with the basics.

A neural network is basically a mathematical model, inspired by the human brain, that can be used to discover patterns in data. From the moment we are born, we start to learn from patterns of what the probable outcome of certain action might be, e.g. putting your hand on a stove. Neural networks have a hidden layer in between the input and output. The hidden layer node is a matrix that simply transform the input data into probabilities of outcomes. These matrices have been created with training data [23].

Training is the process of teaching the neural network. An example of training can be showing 500 images with and without certain object. The algorithm checks for certain features and by understanding if the set of features are present, the object is present in the image. Each images is searched for edges. The computer can detect variance in colours and create a new matrix out of it is called the tensor.

Once we have multiple levels of hidden layers we have a deep learned neural network (Figure 21). The word “deep” in deep learning simply means many layers. You can think of it this way - to learn something and then to do the recognition, such networks like the one below have to do a lot of computations in multiple layers.

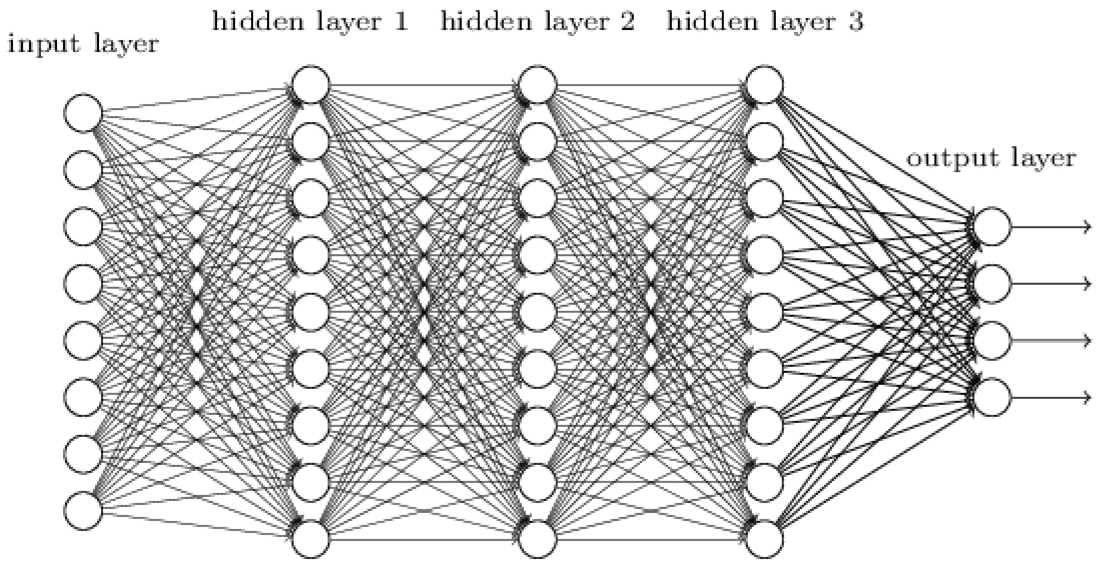


Figure 21. A “deep learned” neural network consists of multiple hidden layers [24].

The amount of computation needed to train a deep neural network is so high that we require powerful devices, like the GPU. These models are made by consuming large data sets consisting of "training" data points. In our case the data points are images and labels - for example (228.png, 'car'). Image datasets can consist of millions of images. This is necessary in order to achieve accurate models that will work on images outside the "training wheels”.

Convolutional add addition dimension to the matrices. Convolutional neural networks task is to classify objects on images, e.g. name what they see [25]. Typical usage is to detect and identify cars, persons, faces, street signs and many more. The meaning of convolutional comes from a Latin word *convolvere* meaning “to roll together”. Convolutional neural networks ingest and process images as tensors. Tensors are matrices of numbers with additional dimensions.

Convolutional networks perceive images as volumes; i.e. three-dimensional objects, rather than flat canvases to be measured only by width and height. That’s because digital colour images have a red-blue-green (RGB) encoding, mixing those three colours to produce the colour spectrum humans perceive. A convolutional network ingests such images as three separate strata of colour stacked one on top of the other.

The Autoware software has three well performing object detectors built in:

- Region-Convolutional Neural Network (RCNN)

- Single Shot Detector (SSD)
- You Only Look Once (YOLO)

So far YOLO or more specifically YOLOv3 presents the best results by several studies in means of speed and accuracy [26, 27]. This is mainly because SSD and RCNN divides the picture into regions while YOLO sees the whole picture.

YOLO has also been previously used on ISEAUTO and we will continue to use it on the vehicle’s software.

3.2 You Only Look Once

YOLO or short for You Only Look Once is an open-source software for object detection. The input for YOLO is an image and the output is the vector of bounding boxes and class prediction.

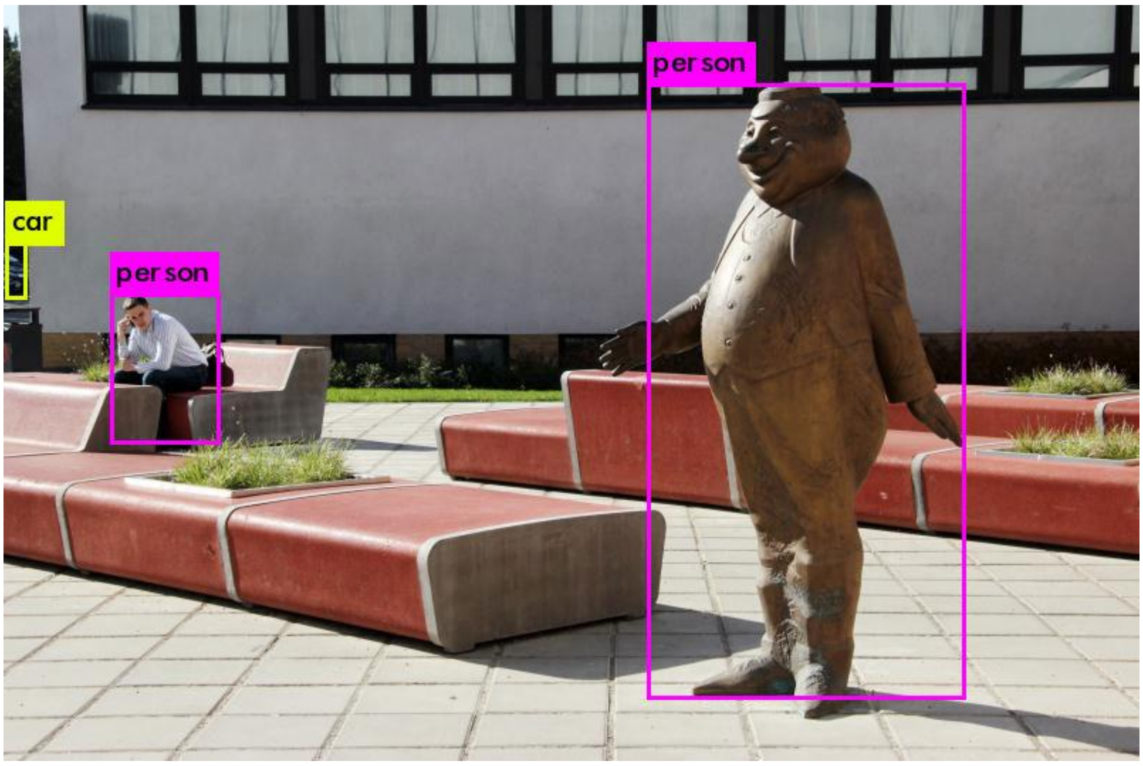


Figure 22. Objects detected with YOLO of an image.

This is more like the game “Where is Waldo”, where you need to pinpoint a certain character. The YOLO has been trained with several “Waldo’s” and the program is looking for certain features that seem similar to the features of pre-trained models. This makes classification faster.

In YOLO's case, YOLO divides the image into many sub-regions, and then outputs the probability for each object using the trained model they created via deep learning. Each sub region gets a probability for each object. The images of YOLO that you see with the overlay is just showing the bounding boxes of areas with high probability for those classes.

We humans functions similar. If we try to see something for example far distance. We may take a guess "the object in sight is a car" but we can also make a statement of confidence "this is definitely a car" or "I'm fairly sure it is a car". This is how Yolo also works, by giving out confidence of prediction. It is possible to set the minimal confidence that yolo creates an actual prediction on the outcome.

The problem is classification, where the output is discrete - there is an object in the image or there is not. Extreme simplification is that regular yolo is looking with a magnifying glass, but it takes more time.

On the implementation of ISEAUTO, only objects as pedestrians and cars are used. Future use can be traffic light detection which is supported by Autoware but since the potential routes that are planned for testing do not cover any intersection with traffic light - this is not a priority.

3.2.1 Tiny YOLO

Tiny Yolo or officially *YOLOv3-tiny* is a lighter version of the lastly explained YOLO. The main objective is faster detection, thus increasing the number of frames being processed in a given time. It does this by looking for smaller amount of objects in a larger given area. The image in the Tiny YOLO case will be divided in bigger regions than in case of regular YOLO. Note that in figure #, where Tiny YOLO has been used for object detection, the car on the left side has not been detected, while with YOLO it was detected.

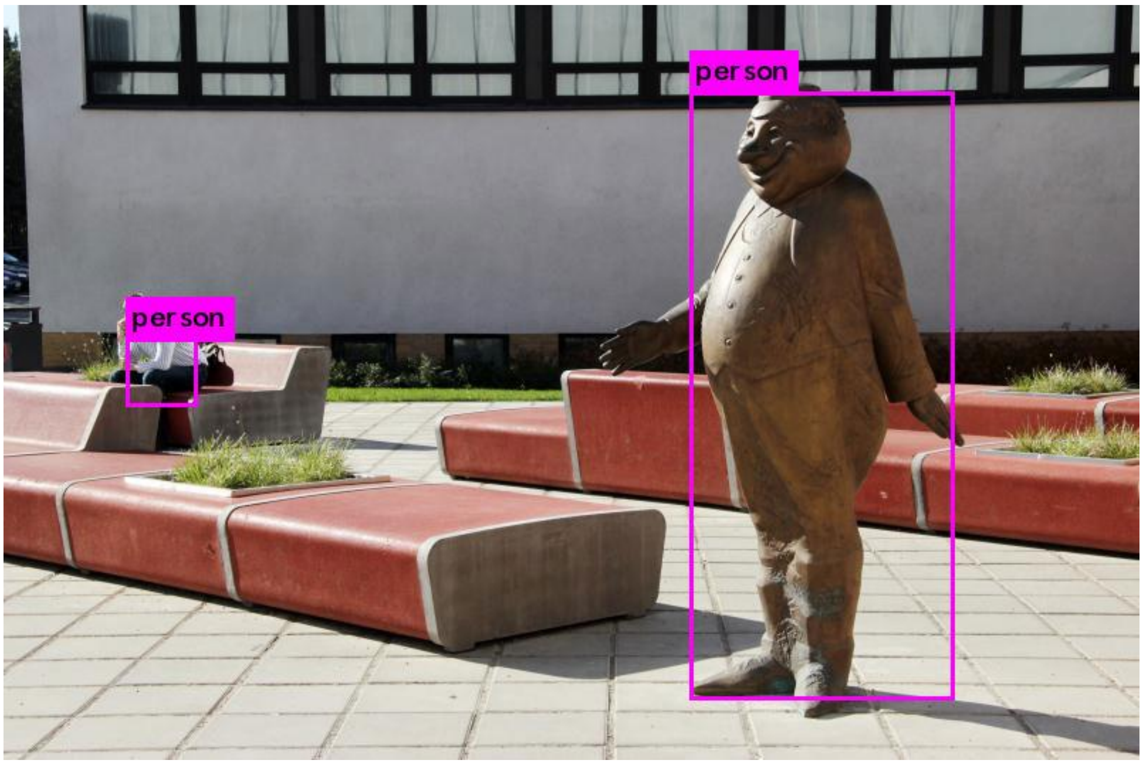


Figure 23. Object detection with tiny yolo.

All this increase in processed frames with the cost of accuracy and confidence of each object. It will depend comparing the Tiny Yolo with the regular Yolo to see which one performs better.

3.3 Testing object detection accuracy

For understanding if the object detection algorithm actually solves a real world problem of detecting moving objects, a test vehicle (Figure 24) was equipped with the Xavier. A FLIR camera and both YOLOv3 and YOLOv3-tiny was used to process the image feed and to compare the results. The objective of this test is to see validate if either Yolov3 or Yolov3-tiny actually detect objects and to compare the results.

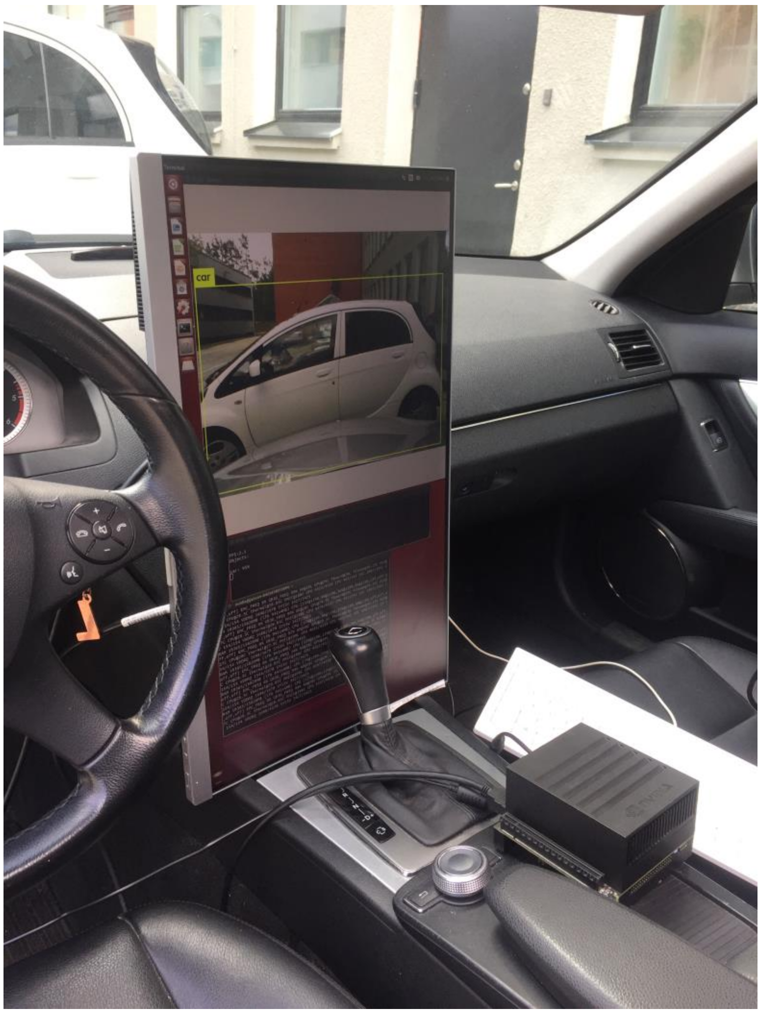


Figure 24. Test-vehicle setup.

The second purpose of this test is to the speed which YOLO and Tiny YOLO run on Xavier and how does the various amount of objects affect the object detection speed.

The test runs are done over a period of few days to test various weather, traffic and light conditions.

On a clear day the YOLO has no problem in detecting both cars and traffic lights. Tiny Yolo seems to only detect bigger objects. From the first test it can be seen that the YOLO is averaging around 2 FPS and Tiny Yolo 15 FPS. Yolo is very good at detecting objects but the speed has to be definitely improved when running on Xavier.

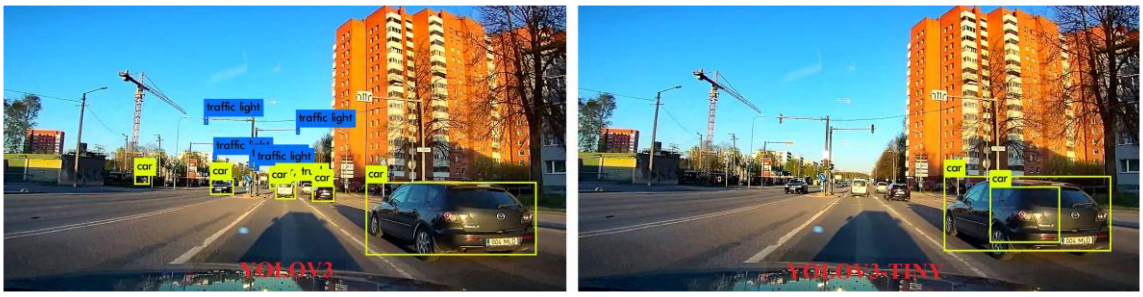


Figure 25. Object detection during a clear sunny day.

In a rainy and cloudy day, there is not much of a difference and almost all the objects are detected. Tiny YOLO still has the tendency to see two cars, where there is actually only one car.

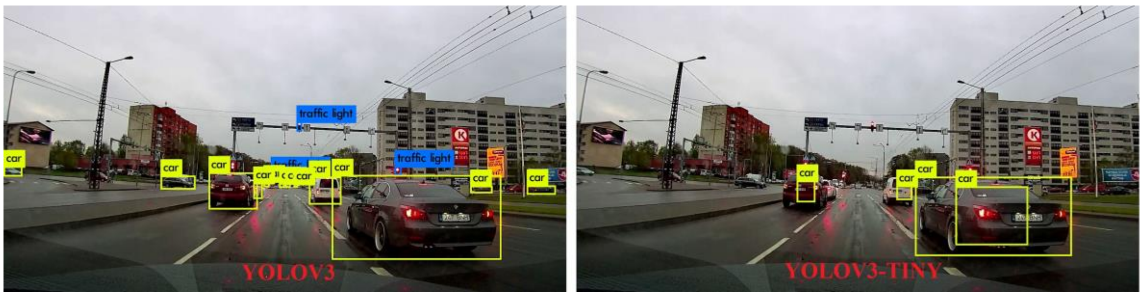


Figure 26. Object detection during a cloud and rainy day.

The accuracy of YOLO is surprisingly good. It can detect parked cars with ease.

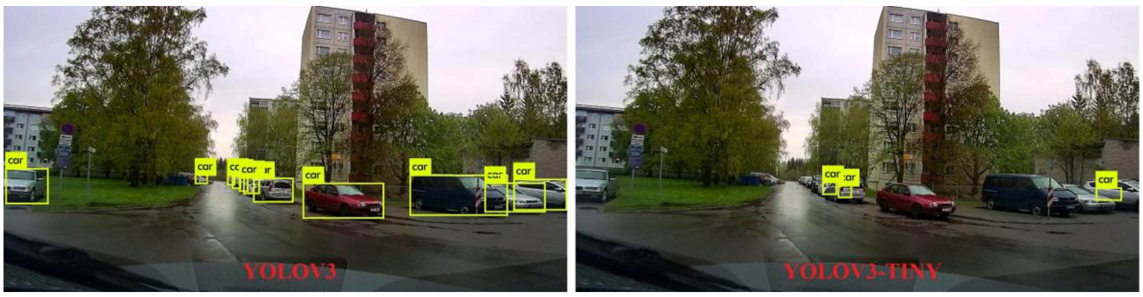


Figure 27. Object detection on a rainy day.

It was interesting to see that YOLO could actually pick up a vehicle with the windscreen covered with rain (Figure 28). Dirt and rain are a common problem for sensor suits that consist of cameras.

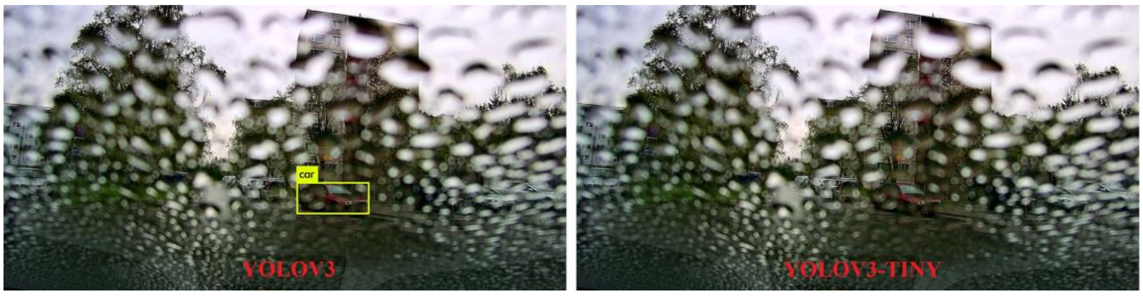


Figure 28. Object detection with water drops.

The direct sunlight is a hard test (Figure 29). This is the condition that caused the fatal Tesla accident which was mentioned before. As it can be seen, the cars are not being detected near the path of the direct sunlight. Cars in the distance in the other hand are being detected.

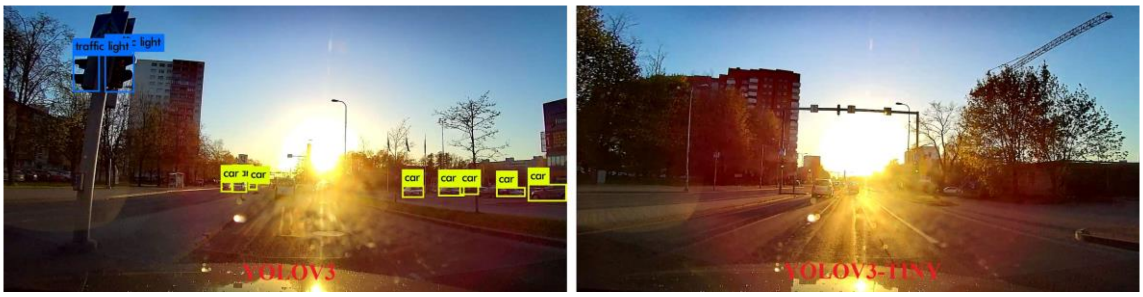


Figure 29. Direct sunlight.

The camera is a passive sensor and needs a source of light to work. In the dark, only objects illuminated by headlamps are detected (Figure 30).

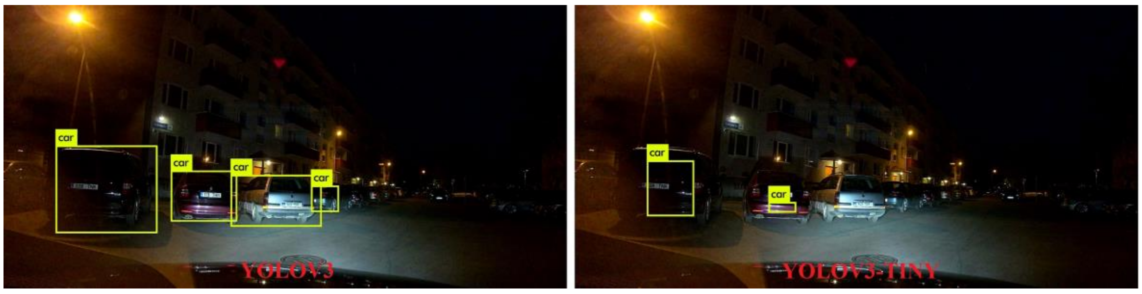


Figure 30. In the night.

The Figure 31 is an excellent comparison for YOLO and Tiny YOLO. If Tiny YOLO would be used, this sequence would have the potential to end with a bad outcome.

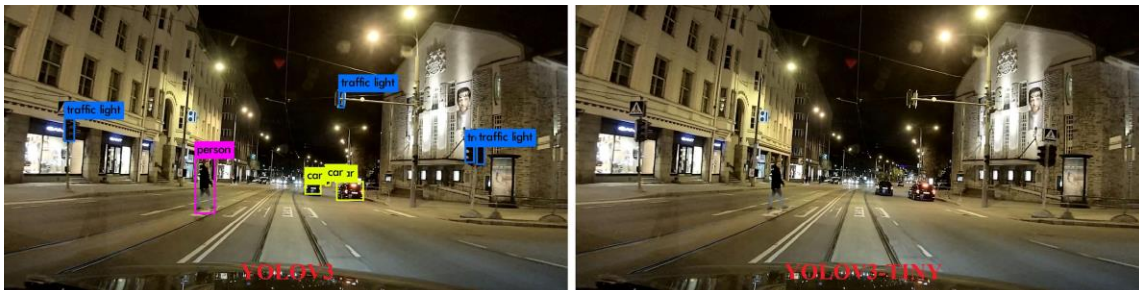


Figure 31. Person on a crosswalk.

This test was an example where real life testing brings out other result than seemingly good results from the lab. The certain conclusion can be made that Tiny Yolo is not suitable for object detection in a safety-critical system. While it allowed for faster detection of objects, the accuracy of the objects was worse than expected. Many times objects, even in plain view of the camera, were not detected.

Xavier was able to run both Yolo and Tiny Yolo, but the frame rate was slow for Yolo, around 2-3 FPS. The more objects appeared in the view, e.g. an intersection, the framerate dropped. After eliminating Tiny Yolo as a possible option, the test results that with the same setup, the Xavier will not be suitable for the purpose with examined configuration.

A new custom configuration of yolo was developed. It consisted only 3 objects – cars, bicycles and pedestrians. The stated configration allowed to increase the maximum FPS to cameras actual maximum limit – 36 FPS on ISEAUTO PC. On Xavier, the improvement was around 10%.

Experimenting with the input resolution showed the biggest increase in frames per seconds (Table 4. Comparison of frames per seconds with different resolutions.. The lower resolution images for object detection comes with a trade off in accuracy. Since confidence is decreased, the smaller objects which have lower level of confidence, will be left undetected. The bigger the resolution, the farther objects can be detected. Even with the smallest 224x224 resolution, most of the objects were still being detected on the video. A further development possibility would be developing ground truth data to compare exact loss of accuracy of detected objects.

Table 4. Comparison of frames per seconds with different resolutions.

	ISEAUTO PC			Xavier		
Resolution	yolov3	yolov3-tiny	custom	yolov3	yolov3-tiny	custom
608x608	14.9 fps	24.9 fps	15.7 fps	2.0 fps	6.0 fps	2.3 fps
448x448	24.0 fps	25.1 fps	29.3 fps	3.4 fps	7.1 fps	4.4 fps
416x416	24.1 fps	25.2 fps	35.8 fps	4.2 fps	7.3 fps	4.5 fps
352x352	24.5 fps	25.0 fps	36.0 fps	6.1 fps	8.2 fps	6.3 fps
224x224	25.3 fps	25.3 fps	36.1 fps	8.2 fps	9.2 fps	8.6 fps

The further development suggestion can be training new weights for the object detection. Using yolo, which is a multi-purpose multi-object, detection is only capable of executing with high computing cost, which in our case was not be sufficient enough to run on Nvidia Xavier. This brings us to the conclusion that the proposed configuration will need a powerful computer to run, and frankly the Xavier seems to lack such power in our tested configuration. The best choice at the moment is to use YOLOv3 algorithm for object detection, FLIR Blackfly S camera and stick to the fast PC which is housed in the ISEAUTO during writing this thesis.

Summary

The purpose of this thesis was to find a suitable configuration for object detection. The configuration had to take into account the existing system. A potential new computer Nvidia Jetson AGX Xavier was also tested for suitability. Based on the requirements, the most suitable camera at the moment was the FLIR camera.

As a result of field testing the object detection, it seemed that a faster algorithm called Tiny YOLO was not fit for the purpose. It missed quite a few objects in critical situations and was even worse in cases such as direct sunlight or night time. The main algorithm YOLO performed quite well even in edge case circumstances. The minus side is the need for computing power - while Tiny YOLO run relatively smooth at around 16 FPS on the Xavier, YOLO was slow averaging only 2-3 FPS.

There are several areas for future research and improvement. The algorithm used for object detection can be deep learned with needed dataset and optimized only for needed purpose. Even without relearning of the neural network, the configuration of objects and detection can be evaluated for better configuration. Another question for future research is what refresh rate will be enough for the safety-critical system - this could affect the requirements on both the hardware and the algorithm. New cameras could also potentially perform better for the accuracy of object detection.

References

[1] World Health Organization, „Global Status Report on Road Safety 2018," 2018.

[2] D. Kahneman, Thinking, Fast and Slow, 2011.

[3] R. Tihane, „Mootorsõidukijuhtide vigade seosed impulsiivsuse, elamustejanu ja agressiivsusega," 2017.

[4] H. Rosa, A New Theory of Modernity, Columbia University Press, 2015.

[5] "http://iseauto.ttu.ee/wp-content/uploads/2019/02/ISEAUTO_REND_20_02_19.png," [Online].

[6] R. Sell, „Vehicle hardware diagram," 2018.

[7] „What is CUDA?," [Võrgumaterjal]. Available: https://www.nvidia.com/object/io_69526.html.

[8] „NVIDIA Jetson AGX Xavier Developer Kit Now Available," 2018. [Võrgumaterjal]. Available: https://news.developer.nvidia.com/nvidia-jetson-agx_xavier-developer-kit-now-available/.

[9] „ROS Kinetic Kame," [Võrgumaterjal]. Available: http://wiki.ros.org/kinetic.

[10] „Cameras," [Võrgumaterjal]. Available: http://wiki.ros.org/Sensors/Cameras.

[11] M. Leier, „Analüüs - Sensoorikalahenduste valik," 2017.

[12] D. Osman, „Global Shutter vs. Rolling Shutter: What’s Better for Your Production Line?,“ [Võrgumaterjal]. Available: https://www.2020hindsight.com/data/Global Shutter vs. Rolling Shutter2.pdf.

[13] J. Thomas, „Rolling shutter versus Global shutter,“ [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=alXQWGyX3xk.

[14] „Blackfly S GigE,“ [Võrgumaterjal]. Available: https://www.flir.com/globalassets/imported-assets/image/blackflys-cmount-gige.png.

[15] „Leopard LI-XAVIER-KIT-IMX274-X,“ [Võrgumaterjal]. Available: https://i1.wp.com/leopardimaging.com/wp-content/uploads/Xavier-Kit.png?fit=1282%2C1250&ssl=1.

[16] „Basler acA2040-35gc - Basler ace,“ [Võrgumaterjal]. Available: https://www.baslerweb.com/fp-1478427532/media/editorial/product_images/camera_series/ace_1/ace_GigE_Sensor-big_Filter_f_l_670x500px__x250.jpg.

[17] „Pylon Camera,“ [Võrgumaterjal]. Available: http://wiki.ros.org/pylon_camera.

[18] „spinnaker_camera_driver,“ [Võrgumaterjal]. Available: https://github.com/ros-drivers/flir_camera_driver/tree/kinetic-devel/spinnaker_camera_driver.

[19] "Velodyne Lidar PUCK," [Online]. Available: https://velodynelidar.com/vlp-16.html.

[20] „Velodyne LiDAR PUCK™,“ [Võrgumaterjal]. Available: https://www.cadden.fr/wp-content/uploads/2017/02/Velodyne_VLP-16-Puck.pdf.

[21] „Calibration,“ 2018. [Võrgumaterjal]. Available: https://github.com/autowarefoundation/autoware/wiki/Calibration.

[22] „autoware_camera_lidar_calibrator,“ [Võrgumaterjal]. Available: https://github.com/autowarefoundation/autoware/tree/master/ros/src/sensing/fusion/packages/autoware_camera_lidar_calibrator.

[23] „What is a neural network?,“ 2018. [Võrgumaterjal]. Available: https://mukulrathi.com/demystifying-deep-learning/neural-network-terminology-explained/ .

[24] M. Nielsen, „Why are deep neural networks hard to train?,“ 2018. [Võrgumaterjal]. Available: http://neuralnetworksanddeeplearning.com/chap5.html.

[25] J. Huang, „Speed/accuracy trade-offs for modern convolutional object detectors“.

[26] A. F. Joseph Redmon, „YOLOv3: An Incremental Improvement,“ University of Washington, 2018.

[27] A. Vainola, „Estimating object detection reliability for TTU "Iseauto" self-driving car,“ 2018.