TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

Evaluation of multiple lidar placement on a self-driving car in Autoware

Master's thesis

Mihkel Väli
163316

Supervisor
Juhan-Peep Ernits, PhD

Tallinn 2018

Declaration

I declare that this thesis is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Mihkel Väli

May 16, 2018

....

(Signature)

Abstract

The main goal of this thesis was to evaluate different lidars placements on a self-driving vehicle in Autoware software. Evaluation requires running multiple tests in the same environment to be sure of the results. Since real-life testing is time consuming and may be more inaccurate, a simulation environment called Gazebo was used throughout the thesis. Gazebo allows simulating the same test with different lidar configurations multiple times. In addition to the main goal, a lidar simulator plugin had to be integrated with Autoware and multple lidar setup had to be configured in Autoware. When evaluating lidar configuration, the goal was to improve object reprojection with a constraint that the localization process does not get worse.

At first testing different lidar configurations in Gazebo lead the author to belive that a two lidar setup with wanted configuration makes localization process worse. After analyzing and carrying out further tests it turned out that the initial results were a cause of bad map. The localization process had gotten worse because the configuration was trying to localize on parts of map that were not mapped very well. As a final result it turns out that adding a second lidar improves object reprojection while the localization process remains the same.

The thesis is in English and contains 43 pages of text, 6 chapters, 7 code examples, 14 figures, 1 table.

Annotatsioon

Mitme lidari paigutuse valik isesõitvale autole tarkvaras Autoware

Magistritöö põhieesmärk oli määrata optimaalne lidarite positsioon isesõitvale autole Autoware tarkvaras. Positsiooni määramine hõlmab endas muutumatus keskkonnas testide tegemist, et veenduda testide õigsuses. Testid viidi läbi simulatsioonikeskkonnas Gazebo, kuna see muutis võrreldes päriseluliste testidega võrreldes testimist täpsemaks ja vähem ajamahukaks. Gazebo võimaldab korduvalt sooritada sama testi erinevate lidari konfiguratsioonidega. Kõrvaleesmärgid olid Gazebos lidari simulaatorplugina ühildamine Autowarega ja mitme lidari konfigureerimine Autowares. Lidari paigutuse valikul oli oluline parandada objekti projitseerimist sedasi, et lokaliseerimine ei halveneks.

Esialgsetest tulemustest selgus, et soovitud kahe lidari paigutus muutis Autoware lokaliseerimise protsessi oluliselt halvemaks. Seetõttu analüüsiti lokaliseerimisalgoritmi tööd põhjalikumalt. Tuli välja, et esialgsed lokaliseerimise probleemid tulenesid halvast aluskaardist, kuna soovitud kahe lidari paigutus mõõtis kaardi osasid, mis olid halvasti kaardistatud. Lõpliku tulemusena selgus, et mõistlikult paigutatud teise lidari lisamine isesõitvale autole parandab objekti proijtseerimise võimet Autowares sedasi, et lokaliseerimise protsess ei halvene.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 43 leheküljel, 6 peatükki, 7 koodinäidet, 14 joonist, 1 tabelit.

List of Acronyms

ADAS Advanced driver-assistance systems

TTÜ Tallinn University of Technology

GNSS Global Navigation Satellite System

IMU Inertial Measurement Unit

ROS Robot Operating System

VLP-16 Velodyne Lidar Puck 16

NDT Normal Distribution Transform

ICP Iterative Closest Point

IP Internet Protocol

GUI Graphical User Interface

RCNN Region-based Convolutional Neural Networks

SSD Single Shot MultiBox Detector

YOLO You Only Look Once

Contents

List of Code examples

List of Figures

List of Tables

Introduction

Cars have been a part of peoples everyday lives for around a century now. Due to rapid developments in electronics and computer science over the last few decades, we have been seeing more computers being integrated with cars. At first there were electronically controlled fuel injection systems, then anti-lock breaking systems and with the development of technology, automotive electronics have got more difficult in time. Nowadays it is very common to have Advanced driver-assistance systems (ADAS) in cars, which help the person behind the wheel to drive the vehicle. The trend seems to be moving towards fully autonomous vehicles, which is proven by many companies, universities and research groups that are currently working on self-driving cars [8–10]. In addition to many other universities, Tallinn University of Technology is also developing an autonomous vehicle, which is a part of the motivation behind the thesis and gives it a real-life context and use.

The goal of automating manual labor is often to make human lives easier without many downsides. In the case of automating driving, it should also reduce the risks of accidents. To reduce the risks of autonomous driving, we need to have high quality information about the surroundings of the vehicle and then make the correct decisions based on the information we have. Self-driving cars collect information about the surroundings using sensors. These sensors often include cameras, Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU) and lidars. But sensors themselves do not make any decisions, they only provide the information. Decisions are made by specially built computer software that analyzes the information it gets from the sensors. These decisions are then used to drive autonomously. The better cars get at self-driving, the more responsibility they hold. Getting better at self-driving means understanding the more of the information from the sensors.

Lidars are often used on autonomous vehicles because of the value they provide. A lidar, which is a acronym for light detection and ranging, is basically a laser range meter. A simple lidar has a trasmitter-receiver pair that transmits a laser beam in a direction. The laser beam will hit an object and reflect from its surface. Once the transmitter receives the reflected beam, it is possible to calculate the exact position of the point that was hit. State-of-the-art lidars consist of many transmitter-receiver pairs rotating around their center axis to measure points 360° around the lidar. Combining all of the measurements, it is possible to create a cloud of measured points. This process is called mapping. As modern lidars are recording hundreds of thousands of points in a second with a accuracy of $\pm$3cm, it is possible to construct a rich and accurate 3D point cloud map of the environment. Now that the autonomous vehicle has a point cloud map, it can be used to exactly pinpoint the vehicle's position on the map. For that the lidar makes another round of measurements and then the vehicle tries to match these points to map points by minimizing the distance between all of the points. When a good enough result is found, we can say that the vehicle knows exactly where it is on the map.

In addition to mapping the and then localizing on it, lidars can be used for much more. For example when a vehicle is driving autonomously and registers some points right in front of the car, it would be best to slow down or even stop. Taking this idea one step further, the vehicle could fuse lidar data with image data from cameras. Object detection on images is a fairly common use for image data. If we detect for example a person on camera image and fuse that information with lidar data, we are able to place this person on our map. Further more, over time the vehicle may detect that the person has moved. Now we can track the person's movement and based on that make predictions to which direction the person is most probably moving. All of that data could be used to decrease different risks in traffic.

Analyzing lidar data, which means hundreds of thousands of points per second, requires a lot of resources. For mapping purposes we could for example analyze every point and then only save it if the point does not overlap with any other point to optimize computing resources. Matching points is a bit more difficult. Intuitive approach would match every recorded point to every mapped point and then try to minimize the distances between the points. This alorithm is called Iterative Closest Point (ICP) and requires more computing power than another algorithm called NDT [6, 11]. NDT is a compact surface representation algorithm that divides the space into subspaces and then computes a probability density function of points for each subspace [6, 12]. Optimizing the matching process with NDT means comparing only the normal distribution functions instead of all of the points [6, 12].

Goals and research questions

As the title of the thesis suggests, goals of the thesis are tightly tied with testing multiple lidars in a software called Autoware. Goals of the thesis are phrased as follows.

1. Object reprojection is the process of projecting an object from image detector to the point cloud map. The main goal of the thesis is to find optimal position and orientation for two lidars, so that we could improve object reprojection. The main limitation here is that the localization process cannot worsen by adding a second lidar.
2. In order to carry out tests in simulation environment Gazebo, we need to integrate a plugin called Velodyne Simulator with Autoware. This allows us to simulate the same message types in Gazebo as the `velodyne_driver` node in Autoware.
3. As the main goal of the thesis is to evaluate the reprojection process by adding another lidar to the vehicle, we need to configure both of the lidars for simulation and real-life environments.

The research questions are supporting goals we are trying to achive and are phrased as follows.

1. How to use multiple lidars in the context of Autoware?
2. How to work with multiple lidars in the simulation environment Gazebo?
3. How does lidar positioning affect localization, object detection and reprojection?

Contributions include improving Autoware's `velodyne_driver` node to support a multiple lidar setup, configuring the Velodyne Simulator plugin for Gazebo to be used in Autoware and an evaluation of different lidar positions on an autonomous vehicle.

1. Background and related work

1.1 Technical characteristics of lidars

Lidar is in its essence a laser range meter, so in other words a lidar is used to measure distances using a laser. It works by transmitting a laser beam and waiting for the reflection of the beam from an object to return [13, 14]. As we know the speed of light and we are able to measure the time it took for the beam to return, we can calculate the total distance the beam travelled and then divide it by two. A basic lidar has two basic and essential components - a transmitter and a receiver [13, 14]. Hosing these two together and rapidly rotating them around a centre axis, we are able to measure different points 360° around the lidar. Attaching some additional transmitter-receiver pairs lets us measure even more points around the lidar. These kind of measurements provide us many points. As we know the distances to the measured points and the direction of these measurements, we are able to create a 3D point cloud around the lidar.

The above-mentioned method explains how many of the modern lidars work. They have many vertically set rotating transmitter-receiver pairs that measure in the same di-

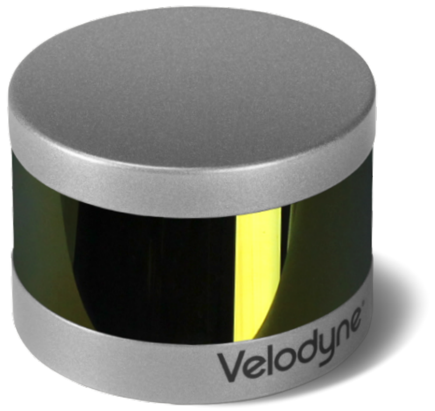


Figure 1.1: Velodyne VLP-16 [5].

Attribute	*Value*
Number of beams	16
Maximum range	100m
Accuracy	$\pm$3cm
Vertical field of view	30° (-15° to +15°)
Vertical separation between beams	2°
Horizontal field of view	360°
Rotation rate	5-20Hz

Table 1.1: Specifications of the VLP-16 [7].

rection but have a different horizontal angle. This is also how the lidar we are using throughout the thesis works. We a using a lidar by a company called Velodyne. It has 16 vertically positioned transmitter-receiver pairs and because of that is called VLP-16. An image of the Velodyne VLP-16 may be seen in Figure 1.1. The VLP-16 is able to measure points 360° around the lidar with a frequancy up to 20Hz [7]. Its vertical field of view is 30°, which means that 8 transmitter-receiver pairs measure below and the other 8 above the horizontal line of sight when the lidar is parallel to the ground [7]. At its maximum spinning rate the VLP-16 is able to measure over 300,000 points per second [7]. Using those points, it is possible to create a 3D point cloud map of the surroundings of the lidar. Other relevant specifications are shown in Table 1.1. The following sections of this chapter describe how are we able to use lidar data.

1.2 Sensor placement

Lidars generate a lot of data and since lidars play a large role in the safety of an autonomous vehicle, it is important to get the most out of the potential of lidars. Something that helps us to make the vehicle safer is analyzing the placement of a lidar on the vehicle. Lidar placement is an unexplored field, but since it is a light sensor that uses reflected light to measure the environment, many parallels could be drawn with other visual sensors, such as cameras [13, 14]. Camera placement is a widely studied field [15, 16].

Most of the studies focus on either finding out how many cameras are needed to cover all of the space in the room or how to find the optimal places of cameras to cover as much space as possible with given amount of cameras. This applies to lidars also. As it is not possibile for us to cover every single corner with lidar beams, we have to find the optimal positions for lidars. The optimal position could be found by minizing the uncovered area. On the other side, by only minimizing the uncovered area we may find ourselves in a position where we have covered for example the back of the vehicle very well, even though our vehicle is not able to drive in reverse. This leads us to the need of giving different areas around the vehicle differet relavance levels. We can draw parallels with this study [16]. This leads us to two positioning problems that we have to solve.

The first problem is the actual positioning of the lidar relative to the vehicle. This is where we have to find the optimal positions for the number of lidars we are currently using in a testing scenario. We have to consider the parameters of the lidars and then position them relative to the vehicle so, that the coverage is maximized. A thing to consider here are the constraints that the vehicle's panels, that may happen to be in the field of view of the lidar, or other sensors may set on the best possible position.

The second problem is the angle that lidar is set on. The roll-pich-yaw representation is routinely used in robotics, aerospace and navigation [17]. Solving this problem requires us to take into consideration the specifications of a lidar and the vehicle. Another thing to consider here is the reliability of algorithms and software we are using.

Lidar placement on an autonomous vehicle has many different constraints to consider. From one side the main idea behind sensors is to get as much information from them as possible, which means maximizing the coverage around the vehicle. This is where we have to consider the limitations we have on our current usecase - specifications of the lidar, relevant areas around the vehicle, details of the vehicle etc.

1.3 Lidars and cameras

Lidars is often only considered for mapping the environment it is observing and localizing itself by matching measured points in the environment that it has already mapped. The two previously mentioned methods do provide us with a lot of information that we can use to understand the environment we are in better, but there is another application for lidars that helps us create a safer vehicle.

Cameras are often used on autonomous vehicles [9]. They provide us information that can be uses to create a safer vehicle. Camera's image is used for object detection by analyzing the image and finding different objects on it using implementations of algorithms. When we are able to detect objects from the camera's image, we are able to avoid them. The data we get from the cameras, has a problem - even though we are able to detect objects from it, we can never be sure how far the object is. It is possible to triangulate the position of the object using two cameras, but this approach uses a lot of computing power. Another option is to merge the image data with the points data from a lidar. This way we are able not only able to calculate the distance to the object, but also track its movement in space.

Both lidars and cameras are important objects in subsections 1.2 and 1.4. From sensor placement point of view, we have to take into account that merging image and point data requires them to have some overlapping in their fields of view. Only then it is possible to successfully first calibrate the sensors and them merge the image and point data.

Camera to lidar calibration is needed for us to calculate the exact location of a camera relative to a lidar. The calculation formula from one sensor to another is called transformation matrix. Only if we know the camera to lidar transformation matrix, we can merge their data.

1.4 Calibration

It is widely known, that calibration is a vital part of understanding data from sensors in robotics. Throughout this thesis when refer to calibration, then we refer to extrinsic calibration, which is the process of finding the orientation and location of the sensor relative to something. The other type of calibration for sensors, that we are only mentioning now, is instrinsic calibration, which this thesis does not handle.

In the self-driving vehicle context, we are finding the position and orientation of something, relative to something else, e.g. cameras position and orientation relative to a lidar. From that we can say that calibration is needed to transform information between different coordinate systems. Using the previous example, calibration is a transforming formula on how to get camera's information to lidar's coordinate system. The transforming formula is often described as a transformation matrix. There are three parts to understanding transformation.

Position In this thesis we will describe the position of a point in space with a 3-tuple position vector [18]. Vector values are described in meters.

Orientation As previously mentioned, we not only need to describe point's location in space, but also its orientation. To describe orientation, we will use a X-Y-Z fixed angles representation [18]. Finding the orientation angle can be illustrated by visualizing a 3D coordinate system. We fix each axis one at a time and rotate the coordinate system around the axis. This approach is also known as the roll-pitch-yaw convention. In this thesis, orientation angles are described in radians.

Frame Frame can be used as description of one coordinate system relative to another [18]. A frame describes both position and orientation.

1.5 Mapping and localization

Mapping and matching are a part of any sensor's localization process. Throughout this thesis, mapping and localization will be understood in the context of lidars.

Mapping is essentially the process that saves all of the points that lidar measures. The mapping process saves the point onto a map. To think of it, it is not quite as simple as it first seems. Mapping process has to take into account the movement of the lidar. This could be well illustrated by imagining a wall that the lidar is measuring. When the lidar is not moving, the wall is measured to some constant distance from the lidar. Now when the lidar is moving through the space towards the wall, the distance to the wall gets smaller and this is something that we have to consider during the mapping process. The previously described thought process shows that mapping is not quite as easy as just saving all of the points and this is why we need to use algorithms while mapping.

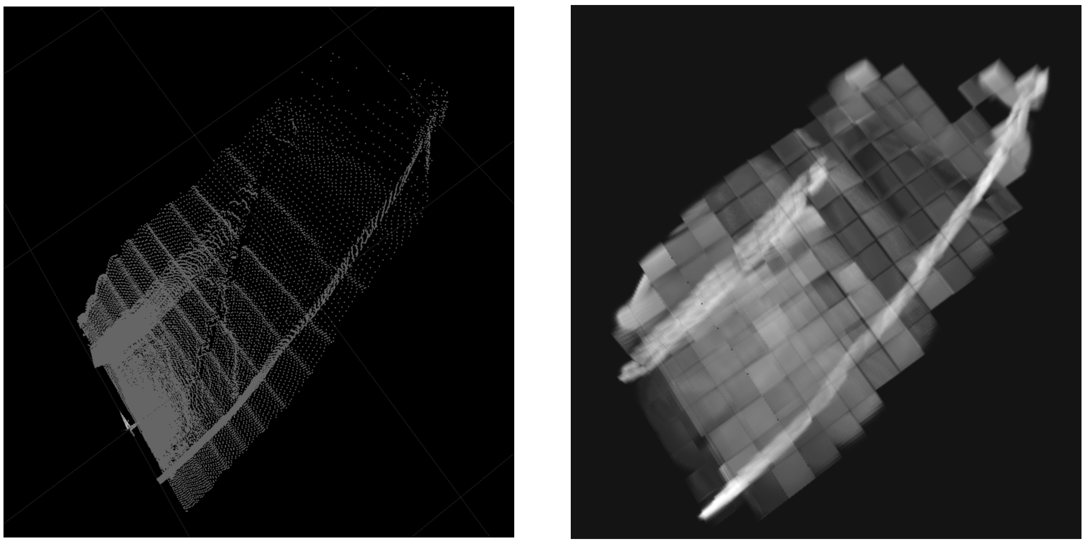
In robotics and computer vision, matching process is used to localize the robot in the environment its in. The process is using current point cloud measurements (source) to match the points to a reference point cloud (target). Matching process can be visualized as holding the target point cloud in place and then trying to tund and move the source point cloud so, that the target and source point clouds match exactly. If we have done that, we can say that we are localized and we know exactly where we are located on the map.

An algorithm often used when processing point cloud data is Iterative Closest Point (ICP). ICP uses a pose estimation to then find the closest target points to all source points. Pose estimation means giving a starting source coordinate, from which the algorithm will start the minimization process. It uses mean-square point-to-point distance to globally minimize the metric over position and orientation [11, 19]. As the matching process iteratively continues, the distance metric is decreasing. A problem with ICP is that it gets slower working on larger data sets [6].

1.6 Normal Distribution Transform (NDT)

NDT is an alternative algorithm to ICP. NDT is shown to be faster and, in most cases, more accurate and more robust to poor initial pose estimates than the ICP algorithm [6].

NDT algorithm's key is in the representation of the reference scan [6]. Opposite to the ICP algoritm's point-to-point analysis, NDT uses a linear combination of normal distributions to find the likelihood of surface points at a certain position [6]. A simplified explanation would be that NDT compares surfaces instead of points.



(a) Original point cloud. **(b)** NDT representation.

Figure 1.2: Original point cloud compared to a NDT representation. Brighter parts of 1.2b represent higher probabilities. Figure adopted from [6].

NDT algorithm works by dividing space into a grid of cells. A probability density function is computed for each cell, based on the point distribution within the cell [12]. Each probability density function can be seen as an approximation of the local surface, describing the position of the surface, as well as its orientation and smoothness [6]. The probability density function representation can be visually seen from Figure 1.2. Brighter and denser parts represent higher probabilities [6].

2. Application of lidars for autonomy in Autoware

2.1 Overview

2.1.1 Robot Operating System (ROS)

Although lidar is a powerful tool, it is only as practical as the software behind it [20]. We are using during the Tallinn University of Technology (TTÜ) autonomous car development project ROS as a software platform for our vehicle. ROS is a middleware that provides structured communications layer, which means that simply put, it is a message delivery system [21]. ROS uses *topics* to exchange *messages* between *nodes*. Communication in ROS can be thought of using a water supply network analogy. Houses in the supply network are like nodes that use the water in any way they need. Piping connecting different houses are like topics - they are used to move the water. Water flowing inside the pipes are like messages - different houses use water based on their needs as different nodes use information from messages as they need.

Leaving the analogy aside and going into how ROS actually works, we only need to add a couple of technicalities. Messages come from somewhere. That somewhere is a node, more specifically a publisher node. There is also another type of nodes - subscriber nodes. Subscriber nodes subscribe to a topic, meaning that they provide a callback function that will be called when a message is published into a topic. We can talk sepraratelY about publisher and subscriber nodes in theory, but in real life scenarios nodes are usually implemented so that a node is a subscriber and a publisher node at the same time. Ofcourse there are exceptions to that, e.g visualization nodes that only listen to a topic and then show it to the user. So to sum it all up, there are different *nodes* in ROS that transmit *messages* to each other using *topics*.

A question arises - what do these messages contain, where does the information come from? Messages usually contain some kind of data that originates from sensors and may be processed by a node. Data that originates from sensors can be reproduced at any given moment using one of two possibilities - we either simulate the data or we save it into a *rosbag*. Data simulation part is explained in Subsection 2.1.3.

Rosbags are recordings that hold the data from sensors. For visualizing purposes we can draw parallels with a video - we can save some kind of data into a rosbag and later replay it however many times. We are able to save streams of any kind of ROS data into a rosbag. That means subscribing to publisher nodes and saving their messages. By definition a rosbag is a set of tools for recording from and playing back to ROS topics [22]. Rosbags are intended to be high performance and they avoid deserialization and reserialization of the messages published to topics [22]. We can use rosbags to record real data while testing and then replaying the data to analyze and run it through different nodes and algorithms.

2.1.2 Autoware

Even though ROS is the spine of the software in the development of the autonomous vehicle, it does not provide quite enough to get the vehicle driving. The thing ROS lacks are the different implementations of various algorithms that are able to eventually as a whole make the vehicle drive autonomously. That is where a software called Autoware proves to be quite useful.

> *Autoware is open source software based on ROS. [...] Most of autonomous driving system consist of recognition, judgment, and operation. Autoware provides necessary functions, such as 3-D map generation, localization, object recognition, and vehicle control, for autonomous driving [23].*

Autoware allows the development of the autonomous vehicle to be alot faster, because it has built a practical architecture specifically for autonomous vehicles. In addition, there are many different ROS nodes implemented in Autoware, which can be used to analyze the input data and make decisions based on it.

Autoware also has a Graphical User Interface (GUI), that makes it easier learn. At first, most of the work will be done using the GUI, but later on as we need more customizability, we start using the command-line interface more. Command-line interface is also used to show explicitly which nodes are ran with which parameters.

2.1.3 Gazebo

As mentioned previously, it is advisable to test different algorithms and ROS nodes before we go field testing. One of the possibilities, as mentioned earlier, is to save the sensors data into a rosbag. Another possibility is not to go onto the field at all and simulate the sensors.

During this thesis, as well as the whole autonomous vehicle development project, a simulation software called Gazebo is used. In addition to many other things, Gazebo allows to create a simulated 3D environment that is able to replicate the work of many sensors and moving objects [24]. Gazebo and ROS interfere seamlessly. Based on that, Gazebo will be as a platform to run the initial analysis of lidar usage and positions on a vehicle. The world we are using was build by the team for the development of the autonomous car and a part of it could be seen on Figure 2.1.

In addition to simulating sensors, we are able to save waypoints of a drive that we have already driven through. This allows us to repeat the same kind of test multiple times with same parameters in the same environment with different configurations. For example a test we could run would be driving through the city and trying to localize the vehicle at the same time with different lidar configurations.

2.1.4 RViz

RViz is a 3D visualizer for ROS that we are using. It visualizes data from the sensors and nodes [25]. RViz visualizes by subscribing to a node that is publishing. RViz can be used in real-life and simulated scenarios.

We are using RViz for multiple visualization tasks. Firstly we are using RViz to visualize data from lidars and cameras. Secondly we use RViz to visualize the map we have already built. Thirdly we use RViz to see the waypoints the vehicle is driving through. Further more, we can use RViz not only to visualize, but also to publish data, e.g we are able to publish the pose estimate of the vehicle for matching algorithm. It makes giving the initial pose easier since we are able to use the GUI and pinpoint the exact location of the vehicle instead of trying to estimate the position on the coordinate system and

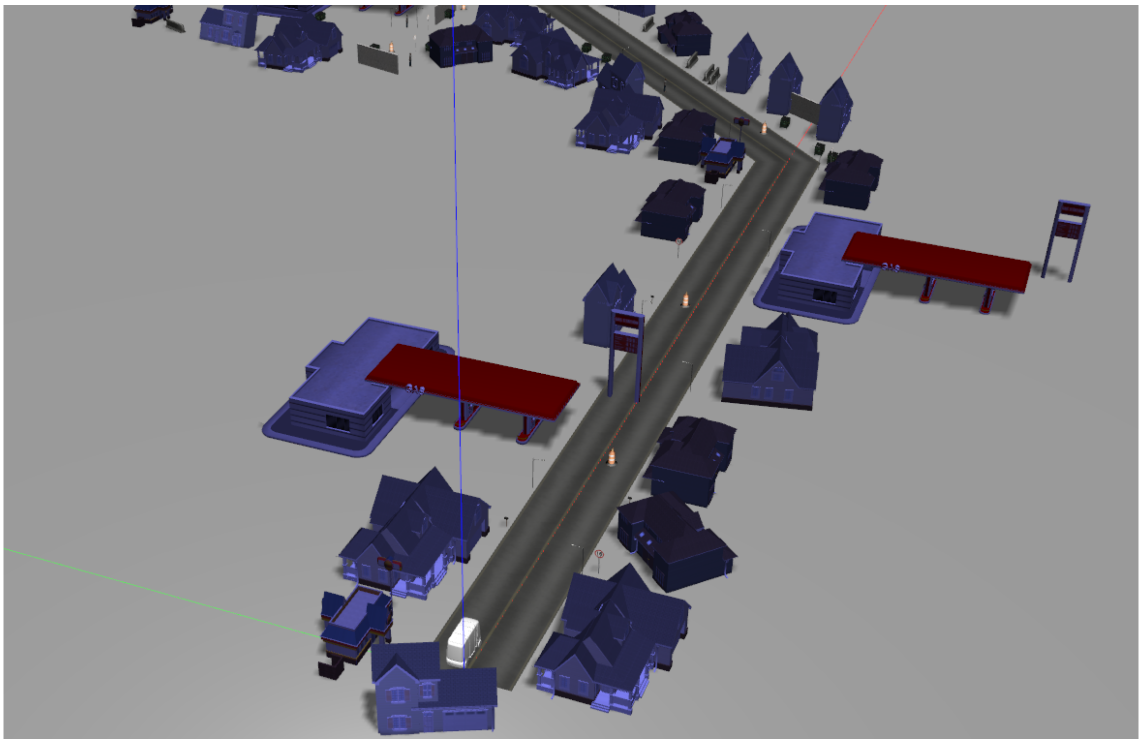


Figure 2.1: The Gazebo world we are using for simulation testing.

pass it from the command-line interface. This is extremely handy in situations where the vehicle's initial pose is not near the starting point of the coordinate system.

2.2 Lidars

2.2.1 VLP-16 and Velodyne Simulator

Working with VLP-16 lidars that we studied in Section 1.1, we have found out that testing different algorithms, usecases and scenarios is potentially a rather expensive process. Because of that we have decided to run initial tests for software in a closed simulating environment. The simulation platform is used throughout the project. The platform is called Gazebo and it comes along with installing Autoware [24, 26, 27]. It allows us to easily configure different parameters and positions of sensors to a high precision. For our usecase, Gazebo allows us to simulate data from lidars, cameras, Global Navigation Satellite System (GNSS) and Inertial Measurement Unit (IMU). Also we are able to add meshes of the actual final version of the vehicle, which is being developed. Adding meshes, which could be thought of as 3D models of the final vehicle, allows us to foresee potential problems with positioning sensors, for example a panel covering up a

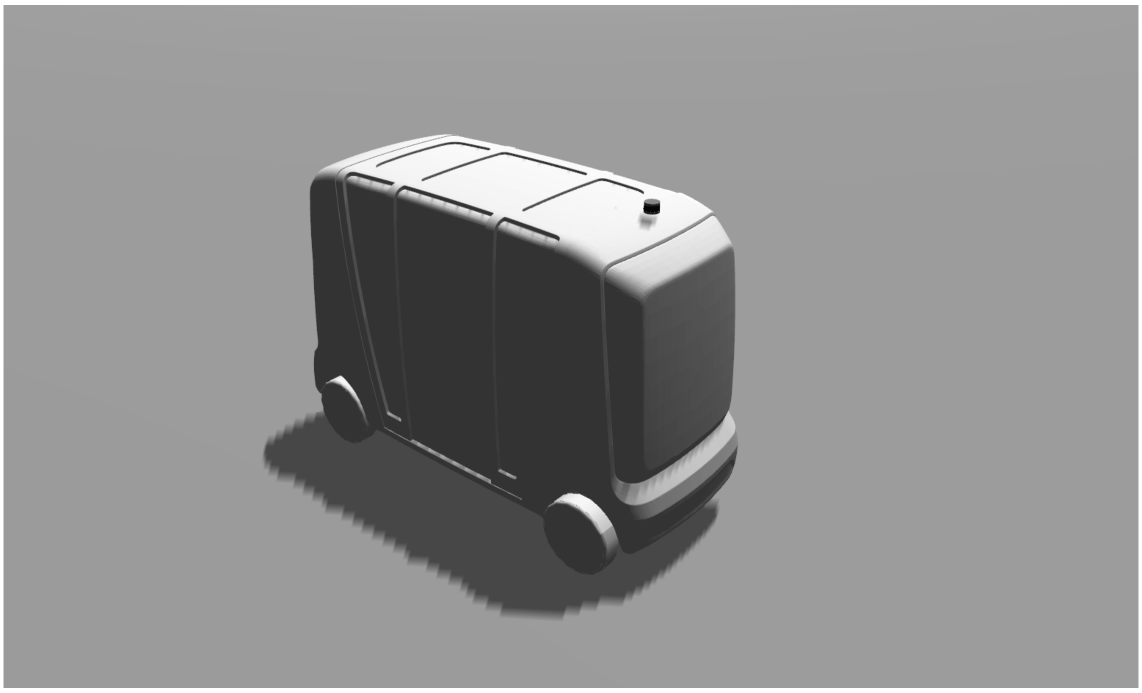


Figure 2.2: The vehicle we are using in the Gazebo with meshes of the actual vehicle and the Velodyne Simulator lidar.

part of the field of view of a lidar or a camera.

Simulating in Gazebo that comes with Autoware has two problems for our usecase. The first problem is that Gazebo has only a simulation model for a Velodyne lidar with 32 vertically aligned lasers, called HDL-32E, which means that around 2 times as many points will be measured compared to a VLP-16. The second problem is that the HDL-32E simulator is publishing `sensor_msgs/PointCloud` type of messages. The real VLP-16 is publishing `velodyne_msgs/VelodyneScan` type of messages, that are converted to `sensor_msgs/PointCloud2` type by `velodyne_driver` node that is provided in ROS [28].

To tackle both of the problems with one solution, we are using a plugin for Gazebo. The Velodyne Simulator plugin provides us the configurations equivilent to a real VLP-16 [29]. We are able to use similar data that a VLP-16 provides with `sensor_msgs/PointCloud2` message types [28, 29]. Using the simulator gives us the ability to get the same type of messages from a real and a simulated lidar, which makes the tests more accurate. Our Gazebo setup can be seen from Figure 2.2. The only thing that the Velodyne Simulator is not simulating well, are the resources that the computer needs during simulation, but on the other hand this is something that we cannot expect from a simulator.

2.2.2 Configuration

VLP-16

The initial setup of the VLP-16 is pretty straight forward, which can be expected from a product that is being sold. A couple of things to note here are that the Internet Protocol (IP) address of the VLP-16 is `192.168.1.201`. Lidar is sending its data over the broadcasting address of `255.255.255.255` on port `2368`. We can see the previously mentioned information from the VLP-16's WebServer GUI. We can also use the WebServer GUI to change the lidar's configuration, e.g. horizontal field of view, accessing IP, host address, port and many other things. The WebServer GUI configuration environment proves to be very useful when using multiple lidars.

Using multiple lidars with initial configurations means that they all try to send their data to address `255.255.255.255` port `2368`. Doing that makes the data from all of the lidars get all mixed up and it makes impossible to distinguish the lidars later on in ROS. This is why we have to reconfigure at least one of the lidars. Here we have two possibilities - either change the ports of the lidars or configure the lidars to other subnets. There could be made several reasons to use either one of the approaches. During the thesis wei have set one lidar's access IP address to `192.168.1.201`, which is the default value, and the other one to `192.168.2.201`. I did that because that way it would make accessing the lidars more convinient in the long run, since then we will be able to configure them at the same time. I also configured them to send their data to the network's broadcasting address `255.255.255.255`, ports `2368` and `2369` respectively. So one of the lidar's configuration remained the same and I incremented the access IP address subnet and the broadcasting address port of the other lidar by one.

In addition to the custom-configured lidar setup, we have a custom-configured velodyne driver for ROS. Even though there is a velodyne driver is built into Autoware, we are not using it, because it is not easily scalable to multiple lidars. We are using a modified version of the velodyne driver built into Autoware. The modifications can be seen on Code example 2.1. Making there changes allows us to group a lidar's messages into a namespace with frame's name. This way a lidar with publish its messages to a topic inside a namespace, e.g. with default values it would be `/velodyne/points_raw`.

```
@@ -20,6 +20,7 @@
   <arg name="laserscan_resolution" default="0.007" />
```

```
+  <group ns="$(arg frame_id)">
    <include file="$(find velodyne_driver)/launch/
   nodelet_manager.launch">
        <arg name="device_ip" value="$(arg device_ip)"/>
        <arg name="frame_id" value="$(arg frame_id)"/>
@@ -32,8 +33,7 @@
        <arg name="repeat_delay" value="$(arg repeat_delay)"/>
        <arg name="rpm" value="$(arg rpm)"/>
    </include>
+   <remap from="velodyne_points" to="points_raw"/>
    <include file="$(find velodyne_pointcloud)/launch/
   cloud_nodelet.launch">
        <arg name="calibration" value="$(arg calibration)"/>
        <arg name="manager" value="$(arg manager)" />
@@ -47,5 +47,6 @@
        <arg name="ring" value="$(arg laserscan_ring)"/>
        <arg name="resolution" value="$(arg laserscan_resolution
   )"/>
    </include>
+  </group>
 </launch>
```

Code example 2.1: `VLP16_points.launch` diff in velodyne driver [1].

Velodyne Simulator

Velodyne Simulator allows us to simulate the Velodyne VLP-16 in Gazebo. It is a open source software. Velodyne Simulator is a catkin workspace, which means it can be built using `catkin_make` command in the root directory of the package.

There is some work to be done to get the simulated lidars also working. After building the plugin and sourcing its variables to your environment, we have to import the lidar to the Gazebo environment. We are going to modify the catvehicle model to fit our needs, since most of the work we need is done there. We can find the file that defines which sensors and panels will be used on the catvehicle model from [30]. We can remove the links and joints that connect the built-in lidar HDL-32E to the vehicle and add the velodyne simulator VLP-16. We can also remove other sensors, since they are not relevant during our tests and only consume computer resources. We are using the Velodyne Simulator almost exactly like the example of adding the VLP-16 shows. The example can be found from the Velodyne Simulator repository, which can be found at [31]. The only thing specific to using two lidars is that we are going to write the example two times and specify unique `name` and `topic` parameters. Now we can run Gazebo and see if different sensors are working like they should. To see if sensors are working, we can echo a topic, e.g. `rostopic echo /points_raw` or visualize the sensor data in RViz.

During the configuration of the Velodyne Simulator I ran into two problems because of lack of documentation. The first thing I missed at first, was that the plugin is actually a catkin package that can be built using command `catkin_make`. The second problem I ran into was that the simulated lidar was publishing in a wrong frame. For example, I had specified it to publish to `/lidar_left`, indicating that it was the left lidar's frame, but it was publising into `/catvehicle/lidar_left`. To be able to use the lidar in the exact frame I made a fix, where I changed the `<frameName>$\textdollar${name}</frameName>` to `<frameName>/$\textdollar${name}</frameName>` in [32].

2.2.3 Calibration

Correct calibration is an important aspect of errorless localization and decision-making in order to extract realistic information and to put it in use. In our context, calibration process gives us exact positions of sensors. Using these exact positions, we can put the measurements from the sensors into perspective and use them relative to the vehicle. ROS has implemented a *tf* library that was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame they want without requiring knowledge of all the coordinate frames in the system [33]. Calibration is something that we need to do only for real-life testing, because Gazebo passes the transform matrices to different nodes in the background. There are three kind of extrinsic calibrations that are directly related to lidars.

1. Multi lidar calibration. Automatic lidar to lidar calibration works by finding similar parts in the measured point clouds and trying to estimate the position of the lidars relative to each other in the process [34]. There is a similar approach implemented in Autoware. It is called "Multi LiDAR Calibrator" and as of writing the thesis, it can be currently found under `develop` branch of Autoware [2]. The implemented calibrator works by assigning one of the lidars as a parent and the other as the child lidar [2]. The calibrator is semi-automatic, meaning that we are going to have estimate the position of child lidar relative to the parent lidar [2]. During

the development of this thesis esimates will be given by measuring the positions using measuring tape. After we have given position estimates, it will output a similar string containing the transformation matrix after analyzing each pair of lidar messages.

```
    rosrun tf static_transform_publisher 1.7096 -0.101048 -0.56108
    1.5708 0.00830573  0.843 /ParentFrame /ChildFrame 10
```

Code example 2.2: Output of Autoware's "Multi LiDAR Calibrator" [2]

This command can be entered to a sourced terminal to give the child lidar frame the transform to parent lidar frame. We can now transform one lidar's data to another lidar's position.

2. Lidar and camera calibration is a widely studied area [35–37]. Camera to lidar calibration is essential on an autonomous vehicle, because when we have calibrated our camera to a lidar, we know exactly to which direction the camera is facing. Once we know that and object detectors detect an obstacle from camera image, we are able to place the detected object into the environment we are measuring with lidar. It is vital for safety and decision making that we are able to place the object in the environment. As of writing this thesis, Autoware has two separate implementations of camera to lidar calibration [38, 39]. Since one of them is still in the `develop` branch of Autoware and rather new, we will not be using it in the current project [39]. The one we are be using during the development of the project has one downside - it is not fully automatic and needs manual labor. With the camera and lidar fixed to their positions, a chessboard has to be moved in front of them [38]. At each position suggested by the documentation of the calibrator, you are needed to mark the exact position of the chessboard on the point cloud. Based on my experience testing the calibrator, it could be said that all of the process takes around a quarter of an hour if you know what you are doing. Comparing it to the previously analyzed multi lidar calibrator, which calibrates the sensors in a matter of seconds, it is a quite long time. Analyzing the length of the time needed for calibrating the camera and lidar in the context of an autonomous vehicle, it is essential that the cameras and lidars wont be moved after they have been fixed to the vehicle. If they have moved even slightly, the sensors have to be recalibrated. If the sensors are not recalibrated, then the autonomous vehicle may end up making wrong decision that may have unthinkable consequences. An example of calibrating a camera to a lidar can be seen on Figure 2.3. Top sections are live data from camera and lidar. Bottom sections are freeze-frames of the current

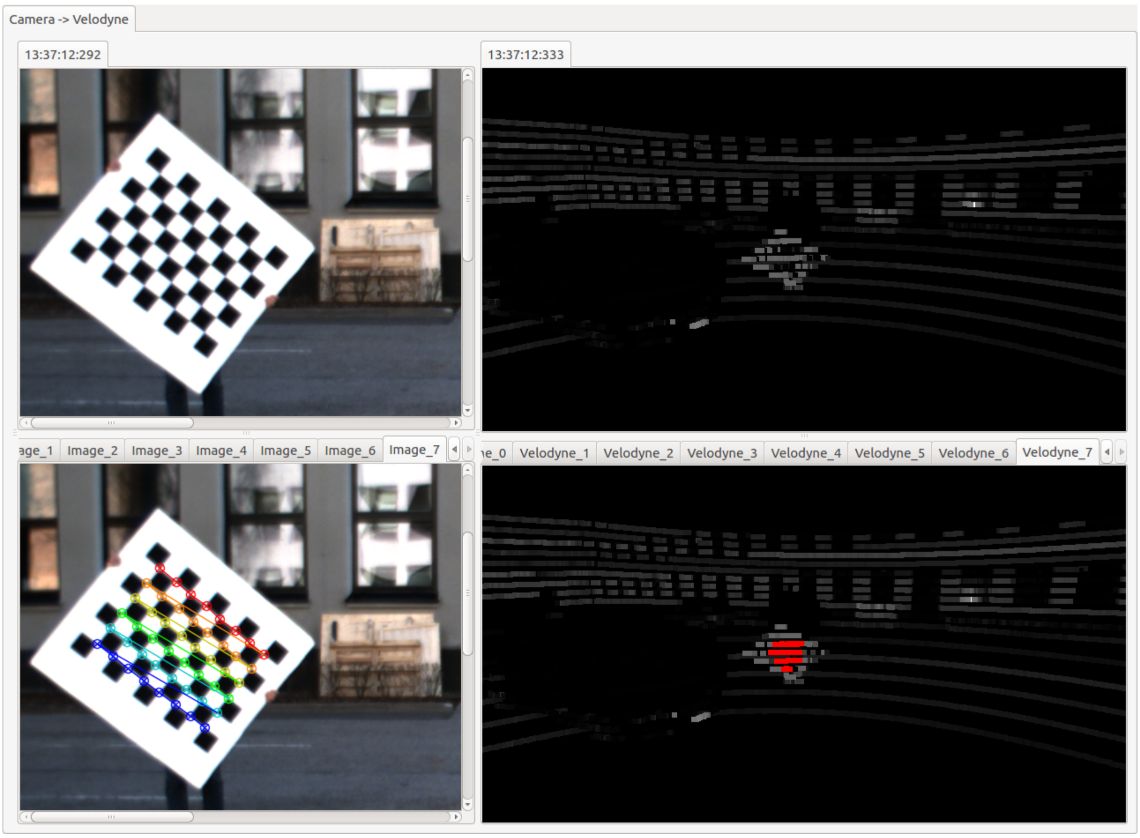


Figure 2.3: A screenshot of the GUI of camera to lidar calibration.

view. Algorithm searches the chess board pattern from the image and the user has to select the center of the chess board from the point cloud. The selection is marked with red.

3. Lidar to vehicle calibration seems to be a little studied area [10]. Since at the time of writing the thesis, there is no implementation of a fully automatic extrinsic calibration, we will be using measuring tape to estimate the location of lidars. Lidar to vehicle calibration is a realistic thing to implement and can be seen as a future work in the field. We are launching the tranformation from a lidar to the base link of the vehicle using a the `static_transform_publisher` node. An example of doing that can be seen on the Code example 2.3. The first three constants of the `args` argument are the `x`, `y`, `z` coordinates and the next three are roll, pitch and yaw in radians.

```
<launch>
    <node pkg="tf" type="static_transform_publisher" name="
   velodyne_to_base_link_tf" args="1.4 -0.5 1.75 0 0 0 1
   base_link velodyne 100" />
```

```
</launch>
```

Code example 2.3: Launch file to create a tranformation from `velodyne` frame to the `base_link` frame.

2.2.4 Merging data from lidars

When we have the transformation matrix from the lidar to lidar calibrator, we are able to merge two lidars point into a single frame. Even though we can see them in the same frame, they are still publishing into two entirely different topics. This is where a node called `points_concat_filter` comes in play. The `points_concat_filter` is a rather straight forward node - it subscribes to two topics and publishes into one topic. These two topics are lidar topics and the published topic is the one where the node has added the points together. During the thesis, concatenated points will be published to `/points_concat` topic. Also, we will be running the `points_concat_filter` node from the Autoware's GUI, since it is rather straight forward and allows us easily to be sure whether the node is running currently or not.

Two lidar setup is seen by ROS as follows. One of the lidars is the parent lidar, the other one the child lidar and we get their location relative to each other by using the `multi_lidar_calibrator` node. After doing that, we add another layer of abstraction using `static_transform_publisher` node by creating a `velodyne` frame to the same location as the parent lidar. We are using a custom launch file for that, Code example 2.4.

```
<launch>
    <arg name="lidar_frame" default="velodyne_first" />
    <node pkg="tf" type="static_transform_publisher" name="
   lidar_to_velodyne_tf" args="0 0 0 0 0 0 1 velodyne $(arg
   lidar_frame) 100" />
</launch>
```

Code example 2.4: Launch file to create a abstract `velodyne` frame to the position of the parent lidar.

The point in doing that is that child lidar's points will be published to child lidar's frame, parent lidar's points are published to parent lidar's frame and concatenated points are published to `velodyne` frame. That means that every separate points publisher has its own frame.

2.3 Using lidar data

2.3.1 Localization

Localization walks hand-in-hand with mapping. Localization is also called matching, since it is the process of matching mapped points to currently measured points. Mapping is often something we do once in a new location, when you are not simultaniously mapping and matching, like we are not, so and there is not much to talk about it. Matching on the other hand is a bit more difficult process, especially with two lidars. We touched the theory behind matching in Section 1.5. Through out the thesis, when I am not specifying the matching algorithm, I am talking about NDT matching. NDT matching requires us to run two nodes.

The first node is called `voxel_grid_filter`. What this node does, is downsamples the point cloud. This means that it reduces the number of points in the dataset [40]. This is done for two intuitive reasons. Firstly, since the VLP-16 lidar is measuring up to 300,000 points per second, there will be many points that are measured to be very close one to another. This leads us to the second reason - since there are many points close together, we will win more in computing resources and lose less in accuracy when we remove some of the points.

The second node is the actual `ndt_matching` node [41]. The `ndt_matching` node is a very large component - over 1600 lines of code. It subscribes to six topics and publishes to thirteen topics. The for our purposes, the most important topics the `ndt_matching` node is publishing to are `/ndt_pose` and `/ndt_stat`. The first topic contains messages, as the name suggests, of the pose of the vehicle in the environment. The `/ndt_stat` topic contains messages on the statistics of the calculations - timestamp of the measurement, execution time, iterations it took to find the result, score, velocity of the vehicle and acceleration [41]. In Chapter 3.2, we will be mainly analyzing the score value when comparing a single lidar setup to a double lidar setup.

2.3.2 Reprojection

Object reprojection is the process where we take the image data from the camera, analyze the image and find objects on it and then merge the image data with lidar data. After doing that, we are able to reproject recognized objects onto a point cloud map. After we have the camera to lidar transformation matrix, we need to run five ROS nodes from Autoware.

Image detector

The first thing to do is to run an image detector. Autoware has three well performing image detectors built in - Region-based Convolutional Neural Networks (RCNN), Single Shot MultiBox Detector (SSD) and You Only Look Once (YOLO)2 [42]. According to [42] YOLO v2 gives the best results on the data we have collected, so YOLO v2 will be used throughout reprojection tests in this thesis.

Points image

The `points2image` node uses the calibration to merge image data with point cloud data. It publishes into `/points_image` topic. This is another place, where we cannot use the launch file provided by the Autoware and have to modify it. In Code example 2.5 we have changed the launch file of `points2image` in Autoware to allow specifying the lidar data topic [43]. This lets us run `points2image` node on `/points_concat` topic. This way we can merge the data from two lidars with the image data.

```
@@ -1,12 +1,12 @@
<launch>
    <arg name="camera_id" default="/"/>
    <arg name="camera_info_src" default="/camera_info"/>
+   <arg name="points_topic" default="/points_raw"/>
    <arg name="projection_matrix_src" default="/
   projection_matrix"/>
-   <arg name="sync" default="false" />

    <node pkg="points2image" type="points2image" name="
   points2image" output="screen">
        <param name="camera_info_topic" value="$(arg camera_id)$
   (arg camera_info_src)"/>
+       <param name="points_node" value="$(arg points_topic)"/>
        <param name="projection_matrix_topic" value="$(arg
   camera_id)$(arg projection_matrix_src)"/>
```

```
-        <remap from="/points_raw" to="/sync_drivers/points_raw"
   if="$(arg sync)" />
    </node>
</launch>
```

Code example 2.5: Diff of the changes we need to make to Autowares `points2image` launch file to specify lidar data topic.

In addition to the launch file, we need to change the node's source code. The changes made to the original source code of the node can be seen in Code example 2.6. The change was needed, because the node kept crashing when running it on real data. It appears that the Flea3 camera may publish empty messages and when they do, the node crashes. The fix shown in Code example 2.6 handles the empty messages by just ignoring them.

```
@@ -64,6 +64,7 @@ static void intrinsic_callback(const
   sensor_msgs::CameraInfo& msg)
    imageSize.height = msg.height;
    imageSize.width = msg.width;

+   if (msg.D.size() == 0) return;
    cameraMat = cv::Mat(3,3, CV_64F);
    for (int row=0; row<3; row++) {
            for (int col=0; col<3; col++) {
```

Code example 2.6: Diff of the changes we need to make to Autoware's `points2image` node to get the node working with real-life cameras [3].

Range fusion

The `points2image` node merges image data with lidar data and the `range_fusion` node uses the merged data. In addition to the merged data, the `range_fusion` node uses detected objects from an image. That means that the `range_fusion` node uses modified image data differently two times - firstly `points2image` has merged image with lidar points and secondly an image detection algorithm has detected objects on the image. The `range_fusion` node takes these both as input and merges their info - `range_fusion` node calculates distance to the detected object. The `range_fusion` node is publishing to `/[image_detector_namespace]/image_obj_ranged` topic. Now we know the distance to each object we are able to detect from camera image.

Object tracking

We are also using an object tracking algorithm from Autoware called KLT tracking algorithm. Autoware has implemented a node for that `klt_track`. The `klt_track` node subscribes to the original image topic, to the range fusion topic and published into `/[image_detector_namespace]/image_obj_tracked` topic objects that the reprojection node can use.

Object reprojection

The `obj_reproj` node of Autoware inputs the results of the `klt_track` node, `tf`, the camera's `projection_matrix`, the camera's `camera_info` and reprojects detected images onto the point cloud using bounding boxes. The `obj_reproj` publishes into `/[image_detector_namespace]/obj_label_bounding_box` topic.

We had the same problem with running the `obj_reproj` node that we had with running the `points2image` node. With the Flea3 camera we got some empty messages at times and that kept crashing the node. The fix is similar to the one we made in Code example /refpoints2image-node and can be seen in Code example 2.7.

```
@@ -230,6 +230,7 @@ static void projection_callback(const
   autoware_msgs::projection_matrix& msg)

static void camera_info_callback(const sensor_msgs::CameraInfo&
   msg)
{
+   if (msg.D.size() == 0) return;
    double fkx = msg.K[0 * 3 + 0]; // get K[0][0]
    double fky = msg.K[1 * 3 + 1]; // get K[1][1]
    double Ox  = msg.K[0 * 3 + 2]; // get K[0][2]
```

Code example 2.7: Diff of the changes we need to make to Autoware's `obj_reproj` node to get the node working with real-life cameras [4].

3. Improving lidar coverage near the vehicle

3.1 Overview

Many autonomous vehicles have placed their main lidar on top of the vehicle's roof [8, 9]. This is a logical and intuitive move, since it provides the lidar a 360° field of view. Also it allows the lidar to measure over some objects and improve its object detection that way. The autonomous vehicle developed in TTÜ is using Velodyne VLP-16 lidars. The problem with VLP-16 compared to other popular lidars used in the industry is that when the VLP-16 has only 16 vertically aligned laser beams, others use lidars with 32 or 64 vertically aligned beams. Because of that the compared coverage of it is rather low - there is too much space between the vertical lasers, especially at further distances.

The vehicle currently developed will only be moving forward. Because of that it is vital for us to get best coverage from lidars just in front of the vehicle. Also it is not practical and visually acceptable to put the lidar on a pole so that the lidar would be able to measure in all directions around the vehicle. This means that when using a single lidar on a vehicle it is important to put it to the front part of the roof. It is also important to be sure that the roof is not covering the most bottom lidar beams in front of the vehicle. If we set the lidar just so that the front looking bottom beams nearly miss the roof of the vehicle, we get the best possible location for a single lidar with rotation parallel to the ground on the vehicle. The configuration we are using with a single lidar in Gazebo can measure a point cloud that can be seen on Figure 3.1.

A possibility to see closer to the vehicle with a single lidar positioned in the front of the vehicle would be to rotate the lidar so that it would see closer to the vehicle, over the y-axis, or in other words to change the pitch of the lidar relative to the vehicle such that its bottom beams move closer to the vehicle. The problem with rotating when using a single lidar is that the coverage of the lidar beams gets significally worse on the sides, since rotating the lidar in that position causes the beams to hit the vehicle.

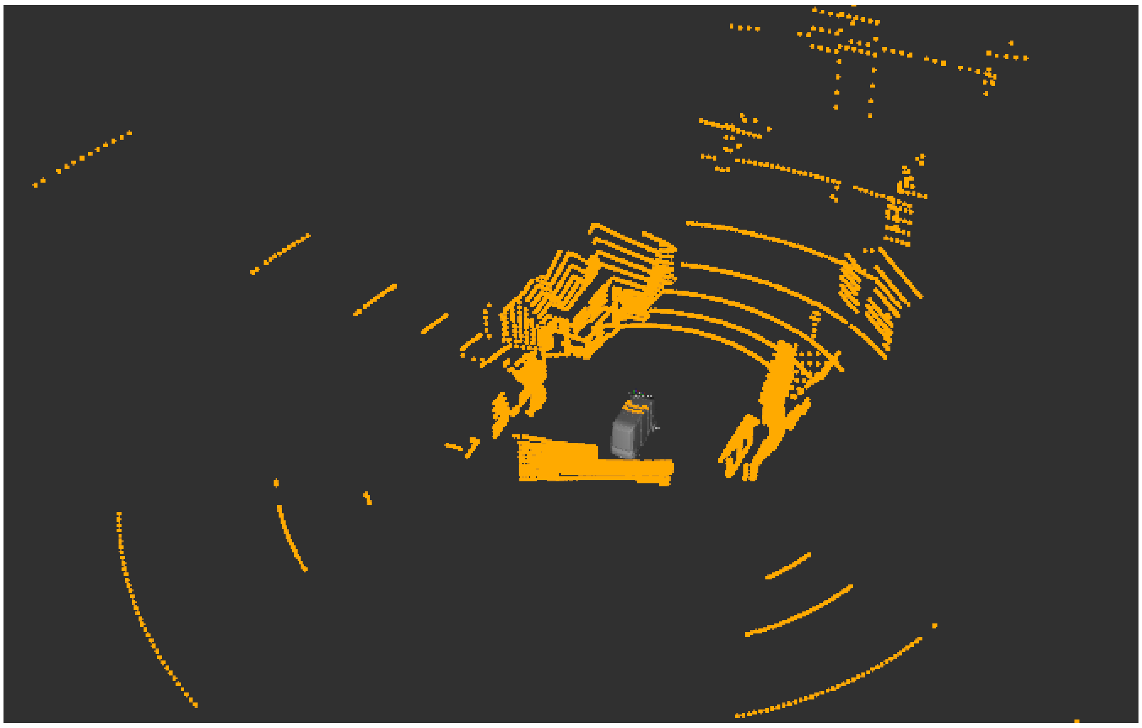


Figure 3.1: Point cloud coverage around the vehicle with one lidar in Gazebo.

3.2 Improvements

3.2.1 Reprojection and object tracking

We want to improve the object reprojection process, especially in front of the vehicle. If we manage to improve object reprojection from the point cloud's perspective, then object tracking is also improved in the process. This applies only to lidar coverage and point cloud data. Image part of the reprojection and tracking can be improved by the quality of image from cameras, object detection algorithms and other image data specific variables that are not in the scope of this thesis. So to improve reprojection in the context on point clouds we have two possibilities - either we cover a larger area with lidars or we improve the density of measured points. By adding another lidar and using two lidars instead of one we can improve reprojection in both of the above-mentioned areas.

Even though the most imporant area to cover with point cloud is in the front of the vehicle, we still need to have coverage on the sides of the vehicle as well since otherwise we would not be able to track and reproject objects on the sides at all and if we are not able to track objects on the sides, we cannot predict their movement relative to our path. This is why the lidar positions shown in Figure 3.2 are optimal. Left lidar is able to cover

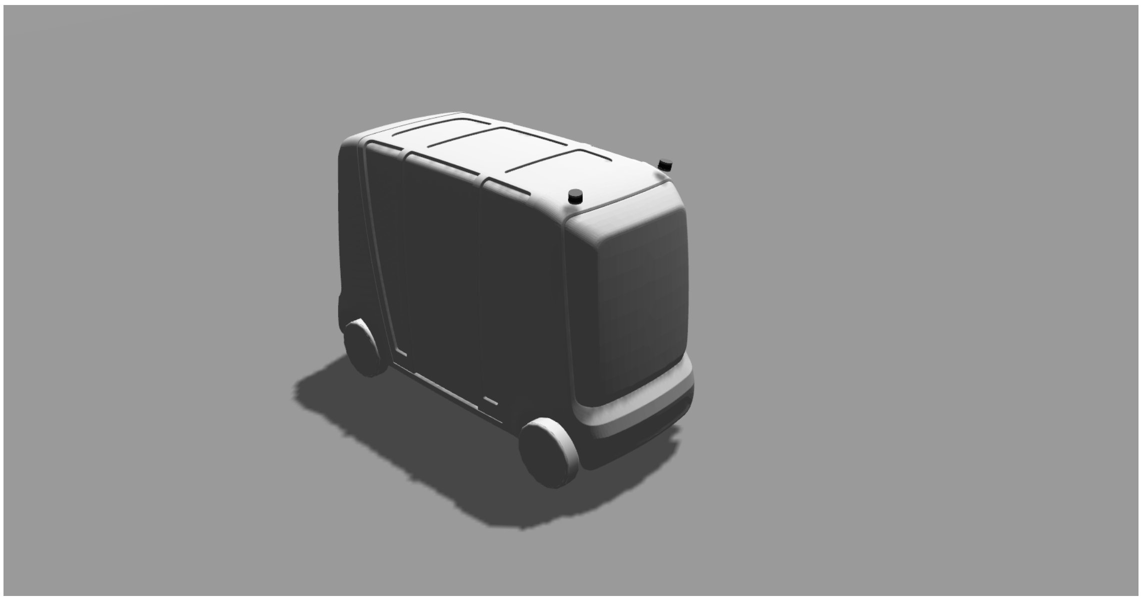


Figure 3.2: Vehicle with two lidars in Gazebo.

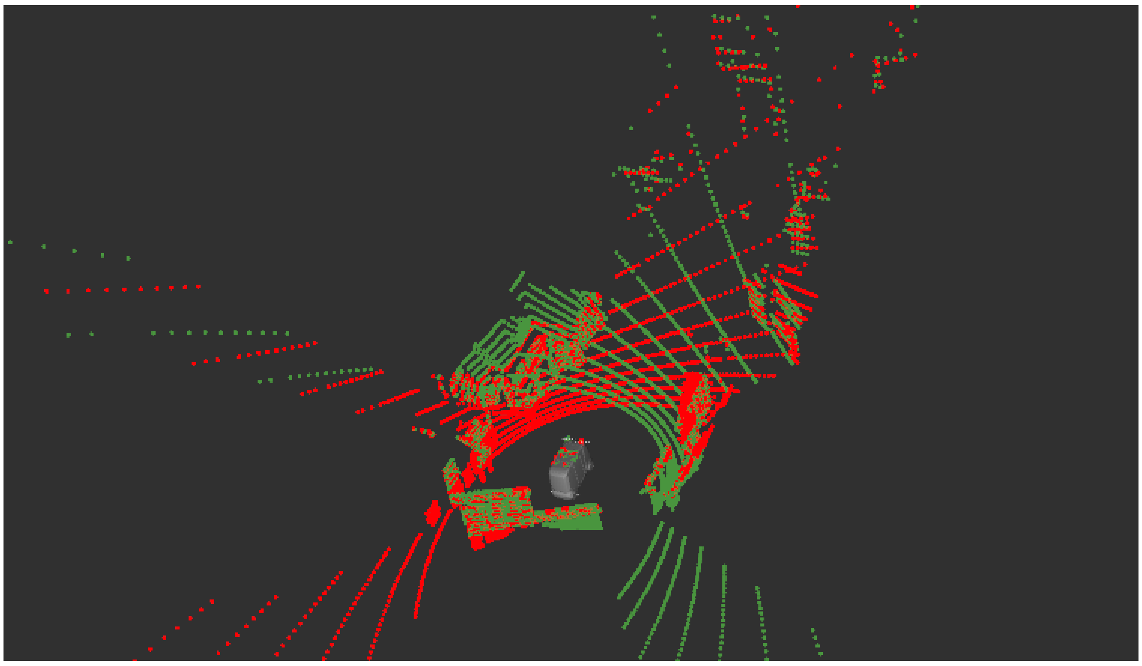


Figure 3.3: Point cloud coverage around the vehicle with one lidar in Gazebo.

the area on the left and in the front of the vehicle and right lidar is algo to cover the area on the right and in the front of the vehicle. From the previous sentance it also appears that the most important part of the environment, the part in the front of the vehicle, is covered by both lidars. To get better coverage, we could rotate the lidars. That would improve reprojection, since by rotating the lidars around their x-axis and y-axis towards outside the vehicle reduces the area not covered by the lidars near the vehicle. After doing that we can achive coverage shown in Figure 3.3 around the vehicle. Red points are measured by the left lidars, green points by the right lidars. On the Figure 3.3 lidars have been rotated 0.15 rad around x-axis and y-axis, which is about 8.6°. In addition to this rotation, we will also be using rotation of 0 rad, 0.05 rad, 0.10 rad, which are about 0°, 2.8°, 5.7° respectively in different tests carried out with two lidars.

As it can be seen from Figure 3.3, area in front of the vehicle is covered by a grid that is created by two lidars. This improves greatly the coverage in front of the vehicle compared to a single lidar, which can be seen on Figure 3.1. In addition to the grid, lidar measurements also get closer to the vehicle, which means that we are able to reproject objects closer to the vehicle. For example solving a simple trigionometric exercise of adding 0.15rad to a VLP-16's pitch and roll on a vehicle, where the lidar would be 2.45m above the ground, brings the bottom beam closer to the vehicle from about 9.14m to 5.61m, which is a difference of about 3.53m. This difference is important in object reprojection and tracking.

In addition to reprojection, that uses image data with lidar data, it is also possible to use lidar data to track objects. Autoware has a lidar tracker nodes implemented [44], which lack proper documentation. It can be assumed that since these trackers use lidar data, they also benefit from improvements in object reprojection that increase lidar coverage in front of the vehicle, but this is not further analyzed in this thesis.

3.2.2 Localization

From the analysis above, in addition to intuition, it could be concluded that adding another lidar improves the safety of the vehicle, passengers and pedestrians. Even though we are able to improve the reprojection process, we have to also be sure that the localization process either stays the same or gets better, so this is something we have to analyze further. To analyze the goodness of the localization algorithm, we can use `score` attribute from the NDT matching node's output in Autoware. The `score` is calculated during the work of the algorithm and gives us the mean distance between closest matched points.

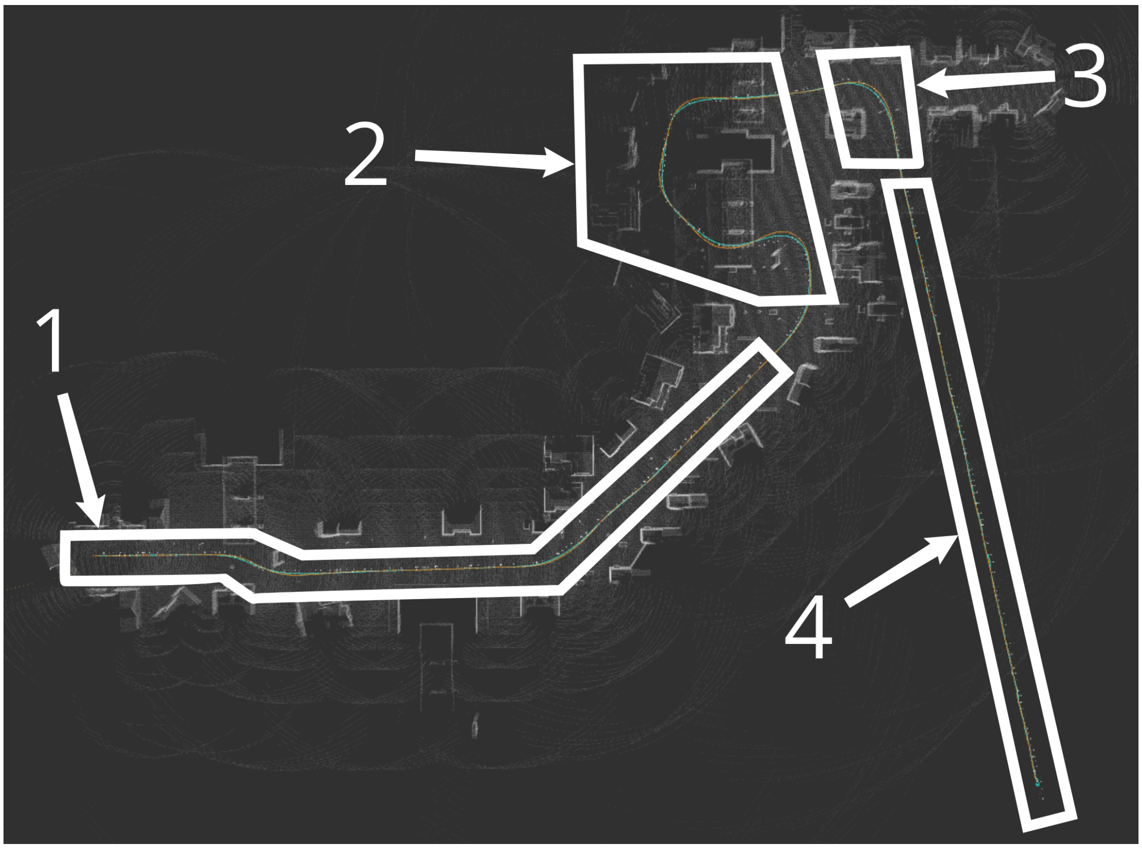


Figure 3.4: The route that was used during testing in Gazebo.

This means that for every point NDT algorithm has matched with the map, it finds the distance to the closest map point, adds all of the distances and divides it with the sum of the points.

It turns out that the localization process seems to get worse when a second lidars is added. I carried out tests with a single lidar and with two lidars with different rotations. To test the scenario multiple times with the same variables, I used Gazebo. Firstly I drove the vehicle through the map and created waypoints for the tests. Waypoints can be made using Autoware's `waypoint_saver` node. It is essential to be localizing while creating waypoints, because the waypoints are created to the current location. Once we have our waypoints, we can run the same test multiple times in a row. The route can be seen on Figure 3.4. The route is marked as follows.

1. The waypoints start off from the left side of the first part. The first part of the route should be rather easy for the localization NDT algorithm, since it is a rather straight line driving in a well mapped area. On Figure 3.6 this part of the route is from 0 seconds until around 130 seconds.

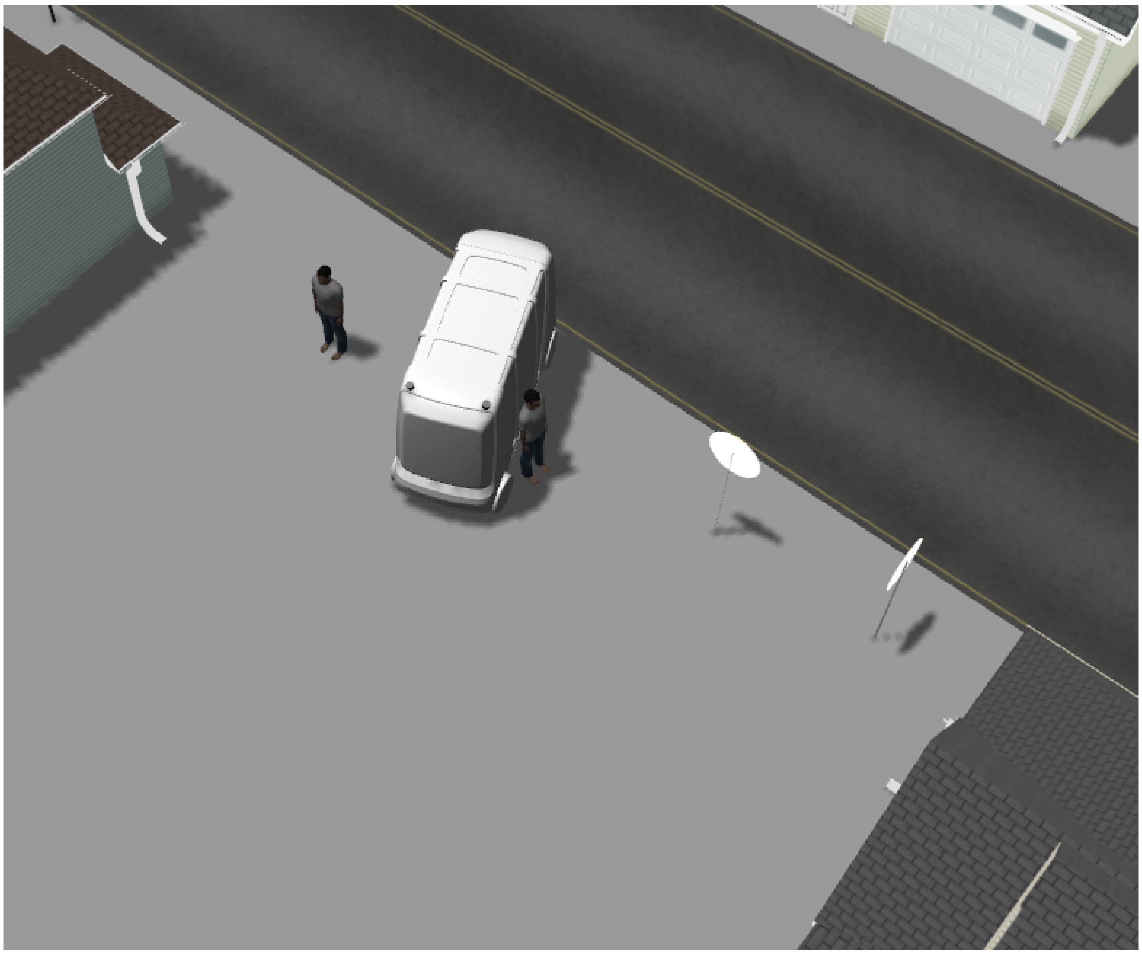


Figure 3.5: The third part of the route in Gazebo, where the vehicle has to drive right between two simulated persons. Image shows consequences when the localization process is not working as well as it should.

2. The second part of the route is designed to test two things at once - localizing in a area that is not specially mapped and to see how the algorithm handles tight turns. The first turn leads into a petrol station in Gazebo, that the vehicle has to navigate through. After that it reaches the area that is not specially mapped. The area is only mapped by points that have reached it from the main road, which means that it is more difficult to localize there. Also some of the buildings are mapped from the other side, which makes the job of NDT localization algorithm even more difficult. After that the waypoints go right underneath the second petrol station onto the main road. This stage is the most difficult for the algorithm since it has a tight turn, a turn in a badly mapped area and has to drive underneath objects. The last two difficulties should increase the score. On Figure 3.6 this part of the route is from around 130 seconds until around 180 seconds.

3. The third part of the path is designed to see how well the vehicle can stay on its track. Two simulated persons are on the side of the road and the vehicle has to drive right between them. Figure 3.5 shows what happens when the localization algorithm is not working as well as it should. Real-life consequences could be fatal. This is why we need to test a close manouvering scenario. On Figure 3.6 this part of the route is from around 180 seconds until around 210 seconds.

4. The fourth part of the route is driving to the distance. The waypoints were created with one lidar and the route ends at the location where the vehicle lost its localization with one lidar. This is the absolute minimum for the two lidar setup also. If the vehicle cannot make it there while localizing with two lidars then we could conculde that the localization process got worse by adding another lidar. On Figure 3.6 this part of the route is from 210 seconds until the end.

3.3 Consequences

The tests described in Subsection 3.2.2 were carried out with different lidar configurations. Firstly the path was driven and localized on by a single lidar. Driving with one lidar set us a ground truth that we can compare the other results to. The results can be seen on Figure 3.6, the top graph. We can see from the graph that the localization algorithm feels rather comfortable thoughtout the route and start to get worse while driving off the map in section 4 of Figure 3.4. We can see some points of time where the score was not near-zero. These spikes are localized in the area that was not mapped in section 3 of Figure 3.4.

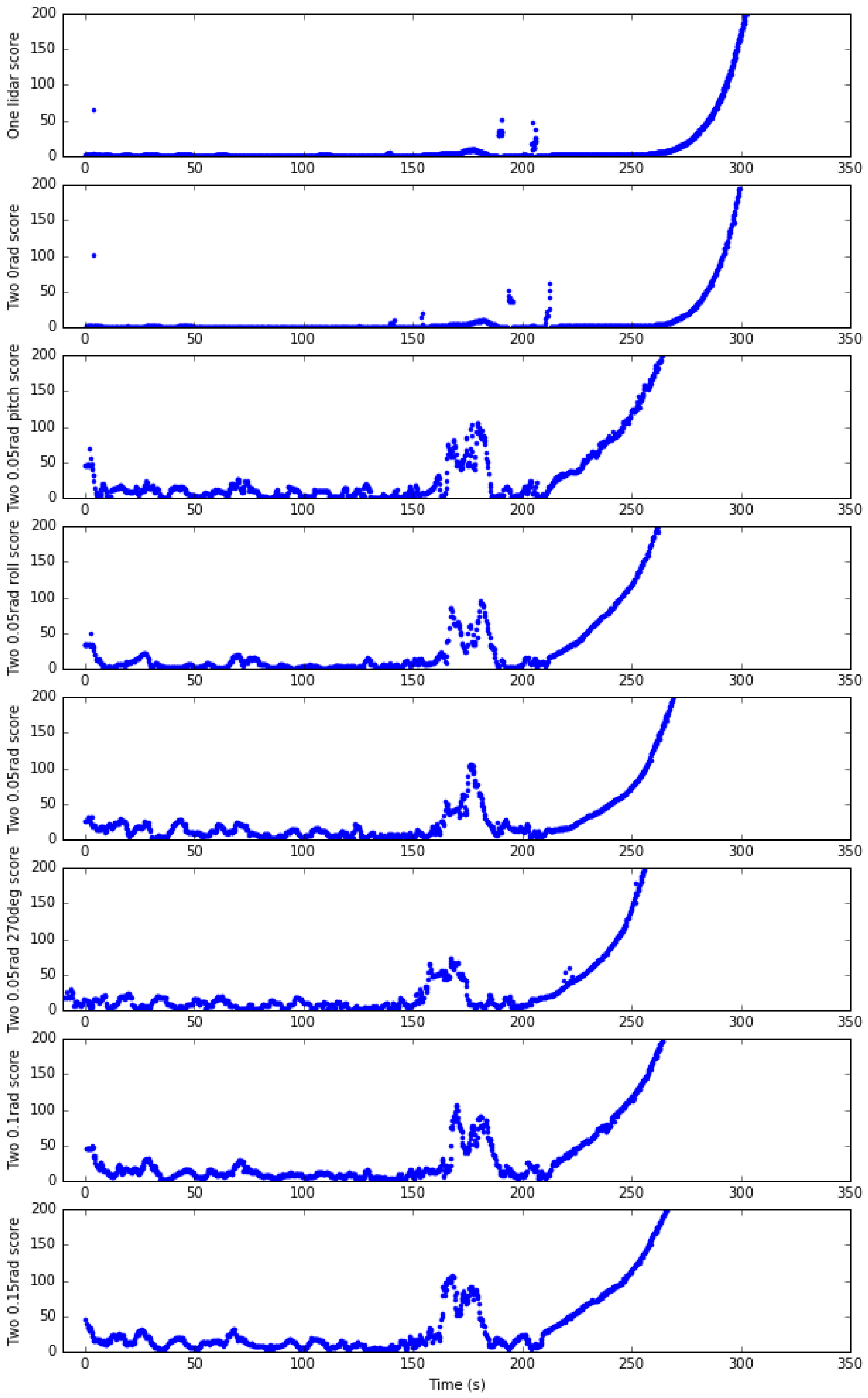


Figure 3.6: The score from NDT matching algorithm from driving test route with one lidar in Gazebo.

After driving with a single lidar and setting the ground truth, we can drive the route using different configurations of two lidars. Firstly I ran our double lidar test on "flat" lidars where the rotation is 0rad. The score graph can be also seen on Figure 3.6, the second graph. The results seem to be pretty much the same as with one lidar, which means that adding a second lidar does not affect localization process much.

Using single and double lidar configuration with 0rad rotation seems to be more or less the same. The score measurements change quite a bit as we rotate the lidars even just a little bit - 0.05rad. I ran tests with only 0.05rad pitch, only 0.05rad roll and with both pitch and roll rotation being 0.05rad, which are respectively third, fourth and fifth graphs of Figure 3.6. As we can see, the score measurements are much more unstable from measurement with flat lidars, especially in the are that is not that well mapped and when driving off the map to the distance. This means that the localization process has gotten worse by rotating a lidar even a little bit. The results remained the same using configurations where 90°of the lidars measurements towards the sky were turned off. This test was ran to be sure that the score would not be effected by some wild points in a area that does not matter from our use case. As we can see from the sixth graph of Figure 3.6, that also did not effect our score results. The score results remained the same with 0.1rad and 0.15rad rotations as can be seen from two last graphs of Figure 3.6.

From the results of the measurements made in this section, we can see, that we could improve the reprojection and object tracking process on an autonomous vehicle by adding another lidar to the vehicle and rotating it a little bit in justified directions. Even though adding another lidar to the vehicle seems to not have any effect on the localization process, rotating the lidars is a different story. Although we rotated the lidars only 0.05rad, which is less than 3°, we saw a worse score when localizing. That shows us that the localization process gets worse when using two rotated lidars. Localization is a vital part of an autonomous vehicle and it is unacceptable to make it worse, even when we are improving other parts of self-driving and decision making.

4. Analysis of results

From tests made in Section 3.2 and conclusions made in Section 3.3 it turns out that even though adding a second lidar and rotating both of the lidars a little bit improves object reprojection, localization based on NDT matching algorithm in Autoware gets worse. To be exact, localization remains more or less the same by adding another lidar but gets significally worse when we rotate the lidars even a little bit. Analyzing the work behind NDT algorithm and how it should work, it seems odd that rotating the lidars makes the localization worse [6, 12].

To further analyze and understand the results we got in Section 3.3, I tried to understand what is going on in Autoware's implementation of NDT matching [41]. My goal was to find out which points are making the localization score worse. NDT matching score describes the mean distance to the closest map point of all matched points. This means that if the score is larger, then the mean distance to matched points is larger. If the score is larger than with "flat" lidars, but the vehicle is still able to navigate, it means that there are some matched points with larger distances to the closest map points. This is the part I changed in NDT matching node. The idea behind changing the node is to find out which measurements make the score worse. We can do that by defining some value from which larger distances between closest points will be accounted as a matching error that significally worsens the score. For the tests I carried out I used a value of 0.9, which seemed to work best for our usecase. Changing the value does not change the results of our tests - if we increase it, we classify less points as matching errors and if we decrease it, we classify more points as matching errors. The points shown in Figures 4.1 and 4.2 are a downsampled version of the original point cloud measurements. The `voxel_grid_filter` node in Autoware has done that [40]. In Figures 4.1 and 4.2 blue dots are points used in the localization process and red dots are points used in the localization process that have larger distance to the closest map point than we previously defined. Red dots are bad for us, because they symbolize points that have too large of a matching error.

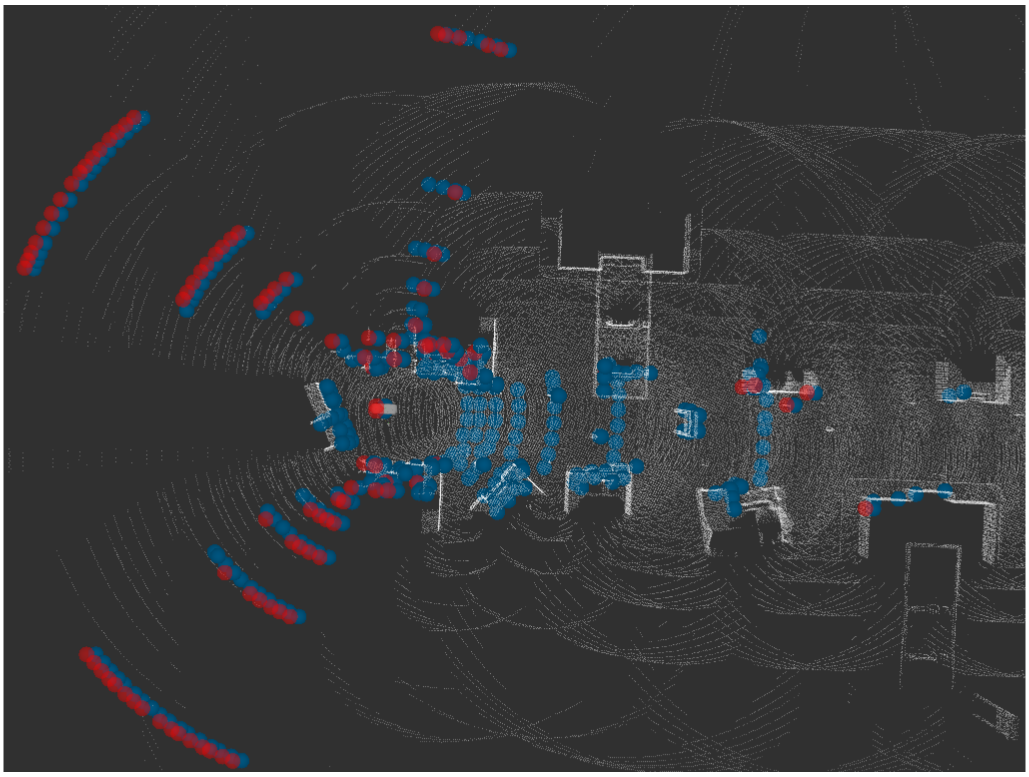


Figure 4.1: Matching errors are colored red measured with one lidar at the beginning of the test route in Gazebo.

From Figure 4.1 we can see that even localizing with one lidar can have quite many bad points, which are marked with red. These points have a larger distance than 0.9 to the closest map points. This why we see a higher score calculation on the first graph of Figure 3.6. Other than that, the one lidar setup did not have many matching errors.

The two lidar setup on the other hand had much more matching errors throughout localizing on the test route. Figure 4.2 shows a especially bad scenario where the vehicle is at the third section of the route from Figure 3.4, where the vehicle is in a unmapped area and turning at the same time.

Both of the Figures 4.1 and 4.2 seem to have a common nominator for the matching errors - red dots are appearing in places that farther away from the white points, which is the map. This is a logical explanation for our graph statistics in Figure 3.6. Because red points are farther away from white points, we can say that the matching errors that increase the score are in areas that are not mapped that well. This is magnified by rotating the lidars. As we can see from Figure 3.3, rotating the lidars makes beams that are facing backwards measure points at farther distances. Those farther distances measured are often areas that are not mapped and this is why our analysis classifies them as matcing errors. Because of that we can not take those matching errors into account. Our results also show

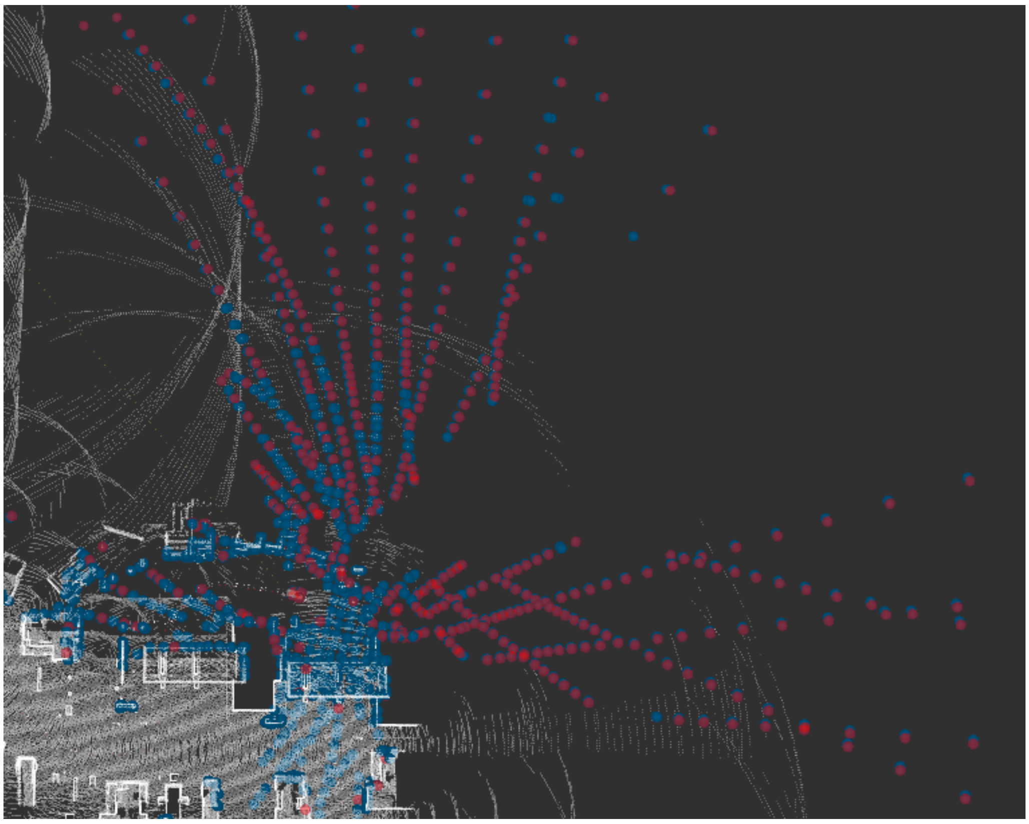


Figure 4.2: Matching errors are colored red measured with two lidars with 0.15rad pitch and roll rotation at the third section of the test route in Gazebo.

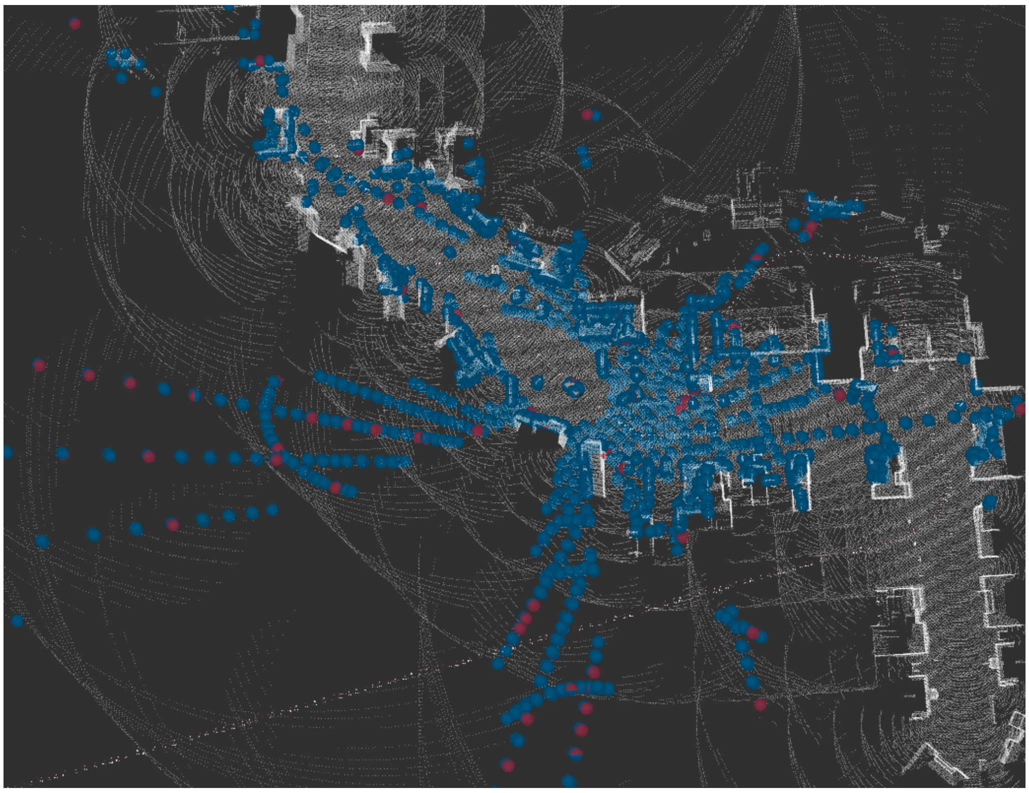


Figure 4.3: Matching errors are colored red measured with two lidars with 0.15rad pitch and roll rotation at the beginning of the third section of the test route in Gazebo.

that mapping with one lidar may leave some areas unmapped that may be useful for a two lidar setup. It is very important that the 3D point cloud map is high quality in areas that lidars use for localization. As we can observe from Figure 4.3 compared to Figure 4.1, when the vehicle is using two rotated lidars to localized in an environment that has a high map quality, it can use much more points to localize itself. As we analyzed earlier, the coverage has significally increased in front of the vehicle.

Even though it may be intuitive to map the environment by just driving through it and saving all of the points, it is advisable to put more effort into it and make the maps with as high quality as possible. In our tests in Gazebo we observed that NDT localization gets worse on matching points at larger distances, because the map did not have any buildings, obstacles or landmarks beyond the first line of buildings, that was surrounding the route. This is inherent for testing in simulation environment and is not affecting the localization process in the real world. Higher buildings could potentially also enhance localization performance with two rotated lidars, because the rotation gives better measurements for higher objects, specially behind the vehicle.

Conclusion

The purpose of this thesis was to evaluate a multi-lidar placement on an autonomous vehicle in Autoware. There were several steps we had to take to analyze the placement of lidars. Firstly the simulation environment Gazebo had to be configured in order to test the setup of lidars. In order to be more accurate, a Velodyne Simulator plugin for Gazebo was used. The Velodyne Simulator allowed publishing the same type of messages as the `velodyne_driver` lidar driver in Autoware, that is used to publish real-life lidar data, is using. Secondly, Autoware had to be configured to be used with multiple lidars. This step included improving the `velodyne_driver` to support a multi-lidar setup. Thirdly, contributions to Autoware's nodes were made to make the reprojection node work with real lidar data. After the initial setup and configurations were made, tests were carried out.

As a result of driving through a route with different configurations in Gazebo, it seemed that even though object reprojection is improved by using a two lidar setup instead of a single lidar, localization process had gotten worse. In order to have a deeper understanding of why the localization process got worse, the Autoware's implementation of the localization algorithm was analyzed. It turned out that localization got worse by using a justified two lidar setup because it was important to rotate the lidars to improve the reprojection process. That lead to measurements of areas that were not that well mapped, which in turn lead to worse localization scores. It was discovered that in order to localize with different lidar configurations it is improtant to have a very high quality map.

Author contributed into improving Autoware's `velodyne_driver` node by implementing a multi-lidar setup support, configuring the Velodyne Simulator plugin for Gazebo to be used in Autoware to simulate a real-life lidar and an evaluation of different lidar position and configurations on an autonomous vehicle was carried out.

References

[1] J. O'Quin, P. Beeson, M. Quinlan, and Y. Liu. ROS Velodyne Driver VLP-16 launch, 2016. URL `https://github.com/ros-drivers/velodyne/blob/master/velodyne_pointcloud/launch/VLP16_points.launch`. Accessed 2018-04-22.

[2] Tier IV. Autoware lidar to lidar extrinsic calibrator, 2018. URL `https://github.com/CPFL/Autoware/tree/develop/ros/src/sensing/fusion/packages/multi_lidar_calibrator`. Accessed 2018-04-01.

[3] Tier IV. Autoware points2image node, 2018. URL `https://github.com/CPFL/Autoware/blob/master/ros/src/sensing/fusion/packages/points2image/nodes/points2image/points2image.cpp`. Accessed 2018-04-01.

[4] Tier IV. Autoware object reprojection node, 2018. URL `https://github.com/CPFL/Autoware/blob/master/ros/src/computing/perception/detection/packages/cv_tracker/nodes/obj_reproj/obj_reproj.cpp`. Accessed 2018-04-01.

[5] Inc. Velodyne LiDAR. Velodyne VLP-16, 2018. URL `http://velodynelidar.com/vlp-16.html`. Accessed 2018-04-22.

[6] Magnusson, M. *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. PhD thesis, Örebro University, 2009.

[7] Velodyne LiDAR, Inc. VLP-16 user's manual, 2016.

[8] Alphabet Inc. Waymo, 2016. URL `https://waymo.com/`. Accessed 2018-04-22.

[9] M. Harris. Uber Could Be First to Test Completely Driverless Cars in Public, 2015. URL `https://spectrum.ieee.org/cars-that-think/transportation/self-driving/uber-could-be-first-to-test-completely-driverless-cars-in-public`. Accessed 2018-04-22.

[10] C. Gillberg and K. Larsson. Targetless extrinsic calibration of vehicle mounted lidars. Master's thesis, Chalmers University of Technology, 2016.

[11] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001. doi: 10.1109/IM.2001.924423.

[12] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2743–2748 vol.3, Oct 2003. doi: 10.1109/IROS.2003.1249285.

[13] J. Ring. The Laser in Astronomy. In *New Scientist*, pages 672–673, June 1963.

[14] R. T. H. Collis. Meteorological lidar system with an improved information display, 1965.

[15] E. Hörster and R. Lienhart. Approximating Optimal Visual Sensor Placement. In *2006 IEEE International Conference on Multimedia and Expo*, pages 1257–1260, July 2006. doi: 10.1109/ICME.2006.262766.

[16] E. Hörster and R. Lienhart. On the Optimal Placement of Multiple Visual Sensors. In *Proceedings of the 4th ACM International Workshop on Video Surveillance and Sensor Networks*, VSSN '06, pages 111–120, New York, NY, USA, 2006. ACM. ISBN 1-59593-496-0. doi: 10.1145/1178782.1178800. URL `http://doi.acm.org/10.1145/1178782.1178800`.

[17] M. H. Ang and V. D. Tourassis. Singularities of Euler and Roll-Pitch-Yaw Representations. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(3): 317–324, May 1987. ISSN 0018-9251. doi: 10.1109/TAES.1987.310828.

[18] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson-/Prentice Hall Upper Saddle River, NJ, USA, 2005.

[19] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992. ISSN 0162-8828. doi: 10.1109/34.121791.

[20] M. Varela-Gonzalez, H. Gonzalez-Jorge, B. Riveiro, and P. Arias. Performance testing of LiDAR exploitation software. *Computers & Geosciences*, 54:122–129, 2013. ISSN 0098-3004. doi: 10.1016/j.cageo.2012.12.001.

[21] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System.

[22] T. Field, J. Leibs, and J. Bowman. Rosbag, 2015. URL `http://wiki.ros.org/rosbag`. Accessed 2018-04-22.

[23] Tier IV. Autoware manuals, 2018. URL `https://github.com/CPFL/Autoware-Manuals`. Accessed 2018-03-23.

[24] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sept 2004. doi: 10.1109/IROS.2004.1389727.

[25] D. Hershberger, D. Gossow, and J. Faust. RViz. URL `https://github.com/ros-visualization/rviz`. Accessed 2018-04-01.

[26] Open Source Robotics Foundation. Gazebo, 2014. URL `http://gazebosim.org/`. Accessed 2018-04-22.

[27] Tier IV. Autoware, 2018. URL `https://github.com/CPFL/Autoware`. Accessed 2018-03-23.

[28] J. O'Quin, P. Beeson, M. Quinlan, and Y. Liu. ROS Velodyne Driver, 2016. URL `http://wiki.ros.org/velodyne_driver`. Accessed 2018-04-22.

[29] Dataspeed Inc. Velodyne Simulator, 2015. URL `https://bitbucket.org/DataspeedInc/velodyne_simulator`. Accessed 2018-04-22.

[30] Tier IV. Autoware catvehicle configuration file, 2018. URL `https://github.com/CPFL/Autoware/blob/master/ros/src/system/gazebo/catvehicle/urdf/catvehicle.xacro`. Accessed 2018-03-23.

[31] Dataspeed Inc. Velodyne Simulator usage example, 2015. URL `https://bitbucket.org/DataspeedInc/velodyne_simulator/src/master/velodyne_description/urdf/example.urdf.xacro`. Accessed 2018-04-22.

[32] Dataspeed Inc. Velodyne Simulator VLP-16 configuration file, 2015. URL https://bitbucket.org/DataspeedInc/velodyne_simulator/src/master/velodyne_description/urdf/VLP-16.urdf.xacro. Accessed 2018-04-22.

[33] T. Foote. tf: The transform library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, April 2013. doi: 10.1109/TePRA.2013.6556373.

[34] C. Gao and J. R. Spletzer. On-line calibration of multiple LIDARs on a mobile vehicle platform. In *2010 IEEE International Conference on Robotics and Automation*, pages 279–284, May 2010. doi: 10.1109/ROBOT.2010.5509880.

[35] S. A. Rodriguez F., V. Fremont, and P. Bonnifait. Extrinsic calibration between a multi-layer lidar and a camera. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 214–219, Aug 2008. doi: 10.1109/MFI.2008.4648067.

[36] Mirzaei, Faraz M. *Extrinsic and Intrinsic Sensor Calibration.* PhD thesis, The University of Minnesota, 2013.

[37] F. M. Mirzaei, D. G. Kottas, and S. I. Roumeliotis. 3d lidar–camera intrinsic and extrinsic calibration: Identifiability and analytical least-squares-based initialization. *The International Journal of Robotics Research*, 31(4):452–467, 2012. doi: 10.1177/0278364911435689. URL https://doi.org/10.1177/0278364911435689.

[38] Tier IV. Autoware camera to lidar extrinsic calibrator, 2018. URL https://github.com/CPFL/Autoware/tree/master/ros/src/sensing/fusion/packages/calibration_camera_lidar. Accessed 2018-04-01.

[39] Tier IV. Autoware camera to lidar extrinsic calibrator in develop stage, 2018. URL https://github.com/CPFL/Autoware/tree/develop/ros/src/sensing/fusion/packages/autoware_camera_lidar_calibrator. Accessed 2018-04-01.

[40] Tier IV. Autoware voxel grid filter node, 2018. URL https://github.com/CPFL/Autoware/blob/develop/ros/src/sensing/filters/packages/points_downsampler/nodes/voxel_grid_filter/voxel_grid_filter.cpp. Accessed 2018-04-01.

[41] Tier IV. Autoware NDT matching node, 2018. URL `https://github.com/CPFL/Autoware/blob/develop/ros/src/computing/perception/localization/packages/ndt_localizer/nodes/ndt_matching/ndt_matching.cpp`. Accessed 2018-04-01.

[42] A. Vainola. Estimating object detection reliability for TTÜ "Iseauto". Master's thesis, Tallinn University of Technology, 2018.

[43] Tier IV. Autoware points2image node launch, 2018. URL `https://github.com/CPFL/Autoware/blob/master/ros/src/util/packages/runtime_manager/scripts/points2image.launch`. Accessed 2018-04-01.

[44] Tier IV. Autoware lidar tracker nodes, 2018. URL `https://github.com/CPFL/Autoware/tree/develop/ros/src/computing/perception/detection/lidar_tracker/packages`. Accessed 2018-04-01.