University of Tartu

Faculty of Science and Technology

Institute of Computer Science

Rauno Põlluäär

# Designing and implementing a bird's-eye view interface for a self-driving vehicle's teleoperation system

Master's Thesis (30 ECTS)

Software Engineering

Supervisor(s):

Karl Kruusamäe, PhD

Tartu 2024

# Abstract

**Designing and implementing a bird's-eye view interface for a self-driving vehicle's teleoperation system**

Autonomous vehicles and self-driving technology have big potential to transform the current transportation system by lessening accidents and optimising traffic flow. In urban environments many difficult situations may occur, which the self-driving technology cannot handle on its own. In these circumstances, a remote human operator may take control of the vehicle via a teleoperation system. To navigate the vehicle through these situations, the operator needs a good overview of the vehicle's surroundings, which can be achieved by a multi angle 360 degree view or a bird's-eye view. At the University of Tartu, the Autonomous Driving Lab is developing a self-driving vehicle, which currently lacks a multi angle or bird's-eye view interface for teleoperation.

The goal of this thesis is to develop a prototype interface that would give the teleoperator a better situational awareness of the car's surroundings. The interface is created using RViz and other ROS (Robot Operating System) capabilities and packages.

**CERCS:** T111 Image processing, T120 Systems engineering, T125 Robotics
**Keywords:** bird's-eye view, multi angle view, teleoperation, self-driving vehicles, interface

# Resümee

**Isejuhtiva sõiduki kaugjuhtimissüsteemile linnuvaate kasutajaliidese loomine**

Autonoomsetel sõidukitel ja isejuhtival tehnoloogial on suur potentsiaal muuta transpordisüsteemi, vähendades õnnetusi ja optimeerides liiklusvoogu. Linnakeskkonnas võib ette tulla palju keerulisi olukordi, millega isejuhtivad sõidukid hakkama ei saa. Sellistel juhtudel võtab kaugoperaator sõiduki juhtimise üle kasutades kaugjuhtimistehnoloogiat. Selleks, et keerulistest olukordadest sõiduk ohutult välja tuua, on operaatoril vaja head ülevaadet auto ümbrusest. Hea ülevaate saavutamiseks saab kasutada vaba vaatenurgaga 360-kraadist vaadet või kavaljeerperspektiivi. Tartu Ülikoolis isejuhtivate sõidukite laboris arendatakse isejuhtivat autot, mille kaugjuhtimissüsteemis puudub selliste vaadetega kasutajaliides.

Käesoleva lõputöö eesmärgiks on luua kasutajaliides, mis annaks teleoperaatorile parema ülevaate auto ümbrusest. Liides luuakse RVizi ja teiste ROS-i (Robot Operating System) võimaluste ja pakettide abil.

**CERCS:** T111 Pilditehnika, T120 Süsteemitehnoloogia, T125 Robootika
**Märksõnad:** kavaljeerperspektiiv, linnuvaade, kaugjuhtimine, isejuhtivad sõidukid, kasutajaliides

# Contents

# 1 Introduction

Autonomous vehicles and self-driving technology has been a popular research target over the past years as this technology has big potential to change the current transportation systems and network. These vehicles can be programmed to adhere to traffic regulations, thus reducing the likelihood of accidents and traffic violations. Unlike human drivers, self-driving cars have better reaction times and are immune to the effects of alcohol and fatigue, ensuring consistent and safe operation [1]. With advanced algorithms, these vehicles can optimise traffic flow, improve fuel efficiency, and reduce emissions [1].

As self-driving cars are becoming more widespread, the need for teleoperation systems, which are used to control machines remotely from a distance, becomes important because self-driving technology cannot handle all the unexpected situations on its own [2]. Autonomous vehicles navigate in urban environments and they may encounter incidents like avoiding and stopping at unexpected obstacles or navigating through busy traffic. In such situations the autonomous vehicle may not be able to solve the incident and the remote human operator has to take control of the car to assist the vehicle [3]. To safely get the car out of these situations, the operator needs a good overview of the environment around the car. Having multiple views of the surroundings and the ability to change the viewpoint gives the remote operator a good visual perception. To improve the ability to estimate depth and distances of objects, a 3D representation of the environment would be helpful [3]. A multi angle interface, including a bird’s-eye view, becomes a useful tool for teleoperators to get better situational awareness around the car.

The Autonomous Driving Lab at the University of Tartu has been developing and validating technologies for self-driving since 2019. Currently the vehicle used is a Lexus RX450h which is equipped with the necessary hardware and different sensors for basic self-driving. The software used on the car is mainly built with Python and Robot Operating System [4]. Currently the Autonomous Driving Lab has a basic teleoperation system that has been tested with the car, however it doesn’t have enough cameras and the software to create real-time multiple angle views and a 3D representation of the car's surroundings for teleoperation.

The objective of this thesis is to improve the situational awareness for the human remote operators by creating an interface which allows the teleoperator to look at the vehicle's surrounding environment from different viewpoints, including a bird’s-eye view.

# 2 Literature review

This chapter explains what is teleoperation and bird's-eye view and what are the main challenges and problems. Finally an overview will be given of existing solutions and software to create a bird's-eye view and improve the situational awareness for teleoperation.

## 2.1 Teleoperation

Teleoperation is a technology that allows human operators to control machines, vehicles or other kinds of systems remotely (figure 1). The operator is at a distance from the controlled object and monitors and operates the machine or vehicle using a control station. The control station typically consists of a screen and some kind of manipulator like a joystick or a steering wheel and pedals (figure 2) [5]. The main features of a teleoperation system are to provide the human operator situational awareness using cameras, radars, LiDARs or other kinds of sensors and to give commands to the vehicle using a control station [5], [6].

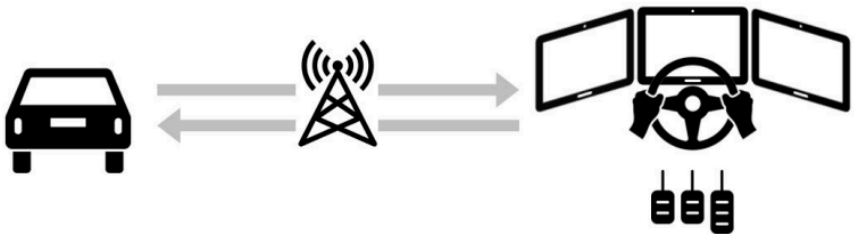


**Figure 1:** *Concept of teleoperation. A human remote operator sees data from the vehicle and gives commands to vehicle* [2].

Teleoperation interface's design and development has been explored since the 1970's and these systems have been used to control robots and vehicles in places that may be dangerous or inconvenient for a human to be present at. Most commonly teleoperation has been used in military operations, space and underwater exploration, agriculture and mining [3].

**Figure 2:** *Teleoperation control station of an autonomous vehicle* [3].

Teleoperation systems are also used for controlling autonomous vehicles and it plays an important part in the development of such vehicles. The Society of Automotive Engineers have defined the following six levels from 0-5 of driving automation [7], [8]:

- Level 0 - no automation, the human driver performs all the driving tasks.
- Level 1 - the driver assistance system does some of the steering or acceleration and breaking
- Level 2 - the driver assistance system does some of the steering and acceleration and breaking
- Level 3 - the automated driving system can execute all driving tasks, the human driver is still needed and will intervene if needed.
- Level 4 - the automated driving system performs all the driving tasks in certain circumstances and may request intervention from a remote driver. A human driver is not required to be in the car.
- Level 5 - the automated driving system performs all and no human interaction is needed.

Currently, from the defined levels, the highest available level is 4 [9], which states that the vehicle can drive autonomously and no human driver is required to be in the car. However

there has to be a human remote operator available to take control of the car if needed [3]. In that case teleoperation acts as a fallback system for autonomous vehicles.

### 2.1.1 Teleoperation methods

Teleoperation methods can be put into three categories: direct teleoperation, supervisory teleoperation and multimodal teleoperation [10].

In direct teleoperation the human operator controls the vehicle or robot by using some kind of a controller like a joystick or a steering wheel and relies on visual feedback from the vehicle like a live camera feed. For direct teleoperation common issues are with communication, data loss and latency [10].

For supervisory teleoperation, the teleoperator and the vehicle or robot share the control of the whole system. For this method to work, the vehicle must possess some level of autonomy, so it can handle lower level tasks like navigation, movement and obstacle detection, and the teleoperator provides the tasks and goals or takes direct control of the vehicle. To use this method, the robot needs a higher computation capability than direct control due to the autonomy requirement [10].

Multimodal teleoperation uses data from more than one sensor and provides the teleoperator with a multimodal view of the environment around the robot. This method is used if the robots are in a complex and dynamic environment as feedback from multiple sensors give a better situational awareness to the human operator. Multimodal teleoperation also requires a higher computation capability and bandwidth to collect and synthesise data from multiple sensors [10].

### 2.1.2 Challenges in teleoperation

Teleoperation systems have been used and tested on autonomous vehicles and multiple different categories of challenges have been identified [3].

One of the problems is the lack of physical feedback. When a driver is in the car, they can feel the acceleration, the speed of the car and the state of the road and how it inclines or declines. For a teleoperator the feedback from the car is visual and they have difficulties with estimating the aforementioned factors [3].

Another important category is video and communication quality. The biggest problem is the latency of the camera feed that is displayed to the remote operator. A study showed that if the

maximum latency is between 200-250 ms, then teleoperation is possible [3]. If the latency is higher than that, it can cause the teleoperators to over- and understeer the car and to accelerate and brake more often. Another factor that also affects the confidence of the remote operator is the video's frequency. If the frequency is between 24-30 frames per second, then the teleoperation can be done smoothly, if it is between 15-20 FPS then it causes the operator to hesitate and give bad input to the vehicle [3].

Another category of challenges lies in visual perception of the vehicle's environment. Depth perception, situational and spatial awareness are known problems for teleoperation [3] [11]. Teleoperators struggle with estimating distances and spatial relationships due to the camera feed being two dimensional and missing the depth cues. This problem mainly affects the operators when performing manoeuvres or judging the vehicle's proximity to obstacles, potentially leading to incorrect decisions about vehicle movements. Another factor that affects the situational awareness is the camera placement and the field of view. The cameras have a restricted field of view, they do not cover the entire environment of the vehicle and the remote operator does not entirely know what objects or events are surrounding the vehicle [3], [11]. One challenge also lies in changing the viewpoint. When a driver is in the car, they can change their viewing point and field of view by turning their head but for vehicles the sensors and cameras are usually mounted in a fixed position [3], [11].

## 2.2 Bird’s-eye view

Bird’s-eye view is a view of an object from a high angle as if a bird would see it flying over it. The usage of a bird’s-eye view is common in Advanced Driving Assistance Systems, as many car manufacturers like Nissan, Audi, Volkswagen and others include (figure 3) this 360° view to help drivers with parking, manoeuvring in tight spaces and help see objects in the car’s blind spots [12].


**Figure 3:** *Bird’s-eye view on the Audi system* [12].

To achieve this view the vehicle usually has multiple cameras mounted to it, ranging from 3-6 cameras [12]. If the car were to have 4 cameras the placement would be that one camera is at the front, one for each side of the car and one in the back (figure 4). To create a top down view, the perspective transformation is applied to the captured video feed and the results are projected onto a planar surface and blended or stitched together to create a smooth image of the car's surroundings [10].

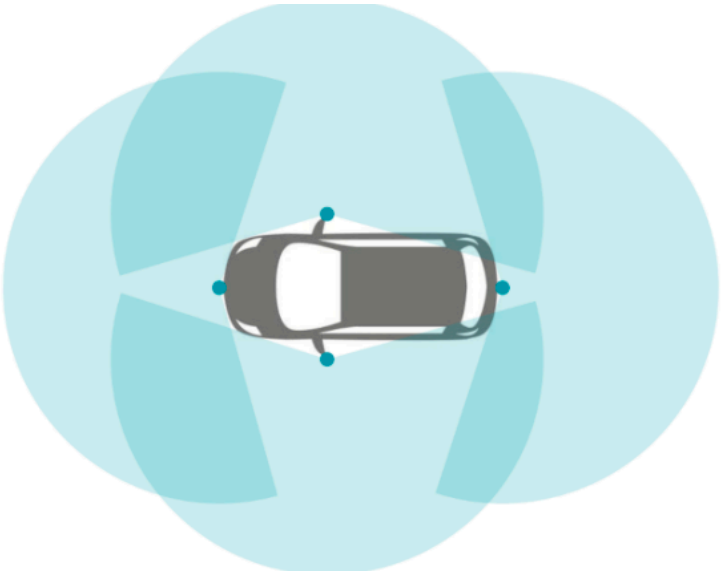


**Figure 4:** *Typical surround-view camera placement on a vehicle* [13].

### 2.2.1 Challenges in bird's-eye view

To achieve a bird's-eye view from cameras, a perspective transformation is applied to the images and it is commonly called inverse perspective mapping [14]. Inverse perspective mapping assumes that the world is flat, so all the pixels from the image are mapped to a planar surface and it causes strong visual distortion of objects that are higher than the ground [14]. Another problem using only a camera-based system is the lack of depth cues and information [15], [16]. To address this problem, LiDAR and depth cameras are one of the most popular choices. LiDARs can provide a high field of view and an accurate 3D scene to understand the surroundings of the vehicle, but the point cloud provided by this sensor does not provide colour information. Depth cameras also provide the colour information, but their range, accuracy and field of view are not as good as the LiDARs [16].

To create a bird's-eye view multiple camera images are usually used and placed so that they have a little bit of overlap [17]. When stitching these images together, they may produce ghost artefact, which are visual errors that occur at the seams of the stitched image. Artefacts may be produced due to the bending of off-centre light that causes mismatch of corresponding features, cameras are not collocated properly and when stitching together, the pixel values at the seams are averaged [17], [18].

## 2.3 Related work

In this section an overview will be given of existing work, solutions and software packages to create a bird's-eye view and give overall better situational awareness.

### 2.3.1 Fujitsu Multi Angle Vision system

Fujitsu Multi Angle Vision is a commercialised system that is installed on cars and allows the driver to see a 360-degree view of the vehicle's surroundings (figure 5) [19]. The system uses four fisheye cameras mounted on the vehicle. The cameras are installed to the front of the car and back of the car near the licence plates and on each side of the car, onto the side mirrors. The captured images are projected inside of a 3D curved mesh instead of a conventionally used 2D plane surface. The projection to a 3D curved surface prevents objects around the vehicle from being visually stretched and distorted [19]. The 3D scene also allows the driver to choose any viewing angle, including the bird's-eye view, by simply rotating the image [19].

**Figure 5:** *Fujitsu Multi Angle Vision system* [19].

In addition to Fujitsu's system, other such 360 degree camera systems can be bought and installed on vehicles [20]. These systems are meant to be used inside cars and help the driver to manoeuvre and park their vehicle, so there is not much information if the camera feeds from these systems can be accessed with a computer or broadcasted over an internet connection.

### 2.3.2 RViz textured sphere

The rviz_textured_sphere package for Robot Operating System is designed to give a better situational awareness in teleoperation by rendering panospheric vision into RViz [21], [22]. A sphere mesh is constructed with customised texture mapping where the images from two fish-eye cameras are projected onto each hemisphere (figure 6). This RViz plugin allows the user to also change different parameters like image topics and its transport mechanism, the field of view of both camera feeds and the blending angle to create a seamless image [22]. Currently the limitations with this package are that it only allows the use of two cameras at the same time and it only has a sphere mesh where the images are projected on. The package also uses image blending to create a seamless image, which could cause ghost artefacts and visual errors at the seams [22].

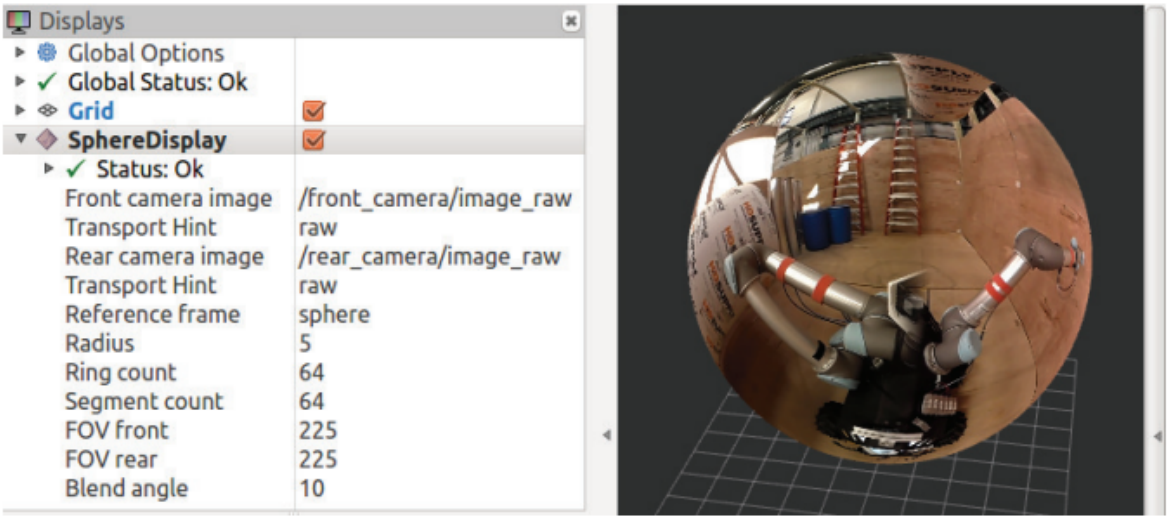



**Figure 6:** *rviz_textured_sphere graphical user interface* [22].

### 2.3.3 Point cloud painter

The pointcloud_painter is a ROS package that integrates the colour information from cameras with the depth data from a LiDAR and visualises it in RViz where the environment and objects can be viewed from multiple different perspectives (figure 7) [22], [23]. The plugin takes two fish-eye camera images and a point cloud from the LiDAR as an input and then merges them to generate a 3D XYZRGB point cloud. This process starts with the stereographic projection of the fisheye camera images to correct the spherical distortion, effectively mapping these images back to real-world coordinates. Then, the colour data from the corrected images is overlaid onto the depth data received from the LiDAR. The alignment

of these datasets is achieved using transformation algorithms that place both datasets into a unified coordinate system, centred between the two camera positions. This integration allows for the creation of a detailed 3D representation of the environment, providing a good visual context and depth perception that is essential for accurate teleoperation [22]. The process of generating and updating the colourised point cloud is computationally intensive [22], which could affect the quality of teleoperation as computationally heavy tasks can increase latency and reduce the operator's ability to respond to the changing environment. Urban environments are quite complex, so if multiple objects occlude each other, mismatches can appear between the depth and colour data when looking at the generated 3D scene from different perspectives [22].

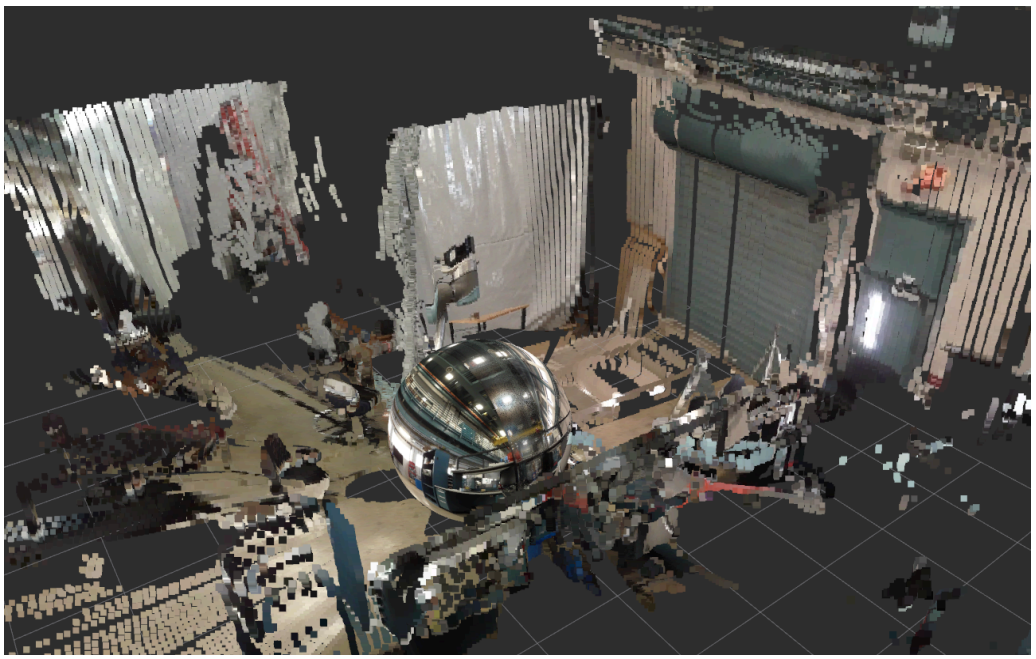


**Figure 7:** *Painted point cloud visualised in RViz* [23].

### 2.3.4 Omnidirectional vision

Omnidirectional vision is a framework containing ROS packages to give teleoperators a better situational awareness of the robots' environment [16]. By integrating images from multiple cameras mounted on a robot, the framework creates comprehensive 360° views, enhancing operators' understanding of complex environments. This system combines camera feeds with 3D LiDAR data to produce a detailed, colourized point cloud (figure 8), offering a richer visual representation that aids in object and terrain differentiation [16]. Implemented using compact omnidirectional cameras, the framework reduces complexity and hardware

demands, making it suitable for environments where space and weight are limited. Demonstrations on various robotic platforms, including the Boston Dynamics Spot robot, have shown the framework's effectiveness in enhancing situational awareness and decision-making in teleoperated robot control [16]. Currently the framework has been evaluated with low resolution cameras, so this approach needs to be tested with higher resolution cameras, as teleoperators need a good quality camera feed to focus on and distinguish different objects [3], [16].

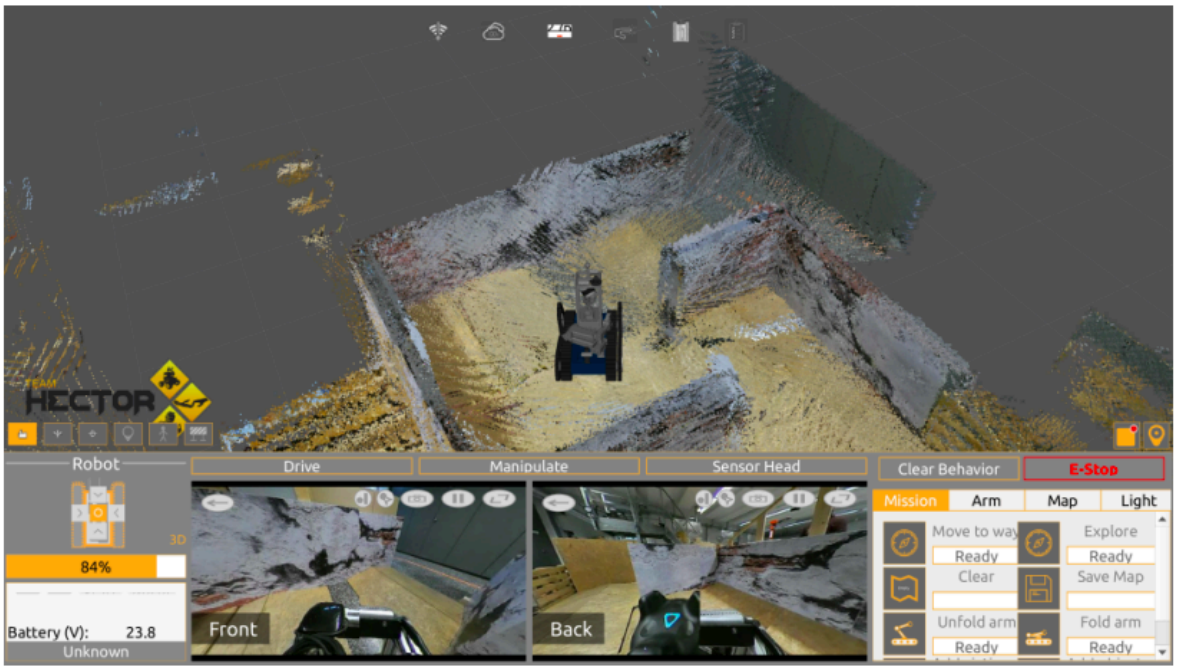



**Figure 8:** *Teleoperators' graphical user interface where the robot's surroundings are shown as a colourized point cloud* [16].

### 2.3.5 Spatio-temporal bird's-eye view

Sato et al. propose a method to create a synthesised bird's-eye view for teleoperation systems by capturing images from multiple fisheye cameras and use as spatio-temporal data [24]. To create this bird's-eye view, images are first captured from high-resolution fisheye cameras, which give extensive and high-quality coverage of the environment but are with significant distortions [24]. These images are then rectified to perspective images through spherical mapping, reducing distortion and preparing them for further transformations. Next, the rectified images undergo perspective transformation to simulate an overhead view using geometric homography matrices. The transformed images are calibrated and stitched together, ensuring precise alignment, especially in overlapping areas. This combined image is

then synthesised into spatio-temporal data, integrating timestamps and localization to allow for real-time updating and gap filling in the imagery. This process results in a continuous and accurate bird's-eye view, however the high quality images increase the computation time and the developed graphical user interface (figure 9) shows the created bird's-eye view at 4 frames per second [24]. The low video frequency and the usage of past stored images make this solution unusable for a autonomous vehicles teleoperation system as the teleoperator needs real-time high video frequency camera feed of the cars surroundings to safely drive the car.

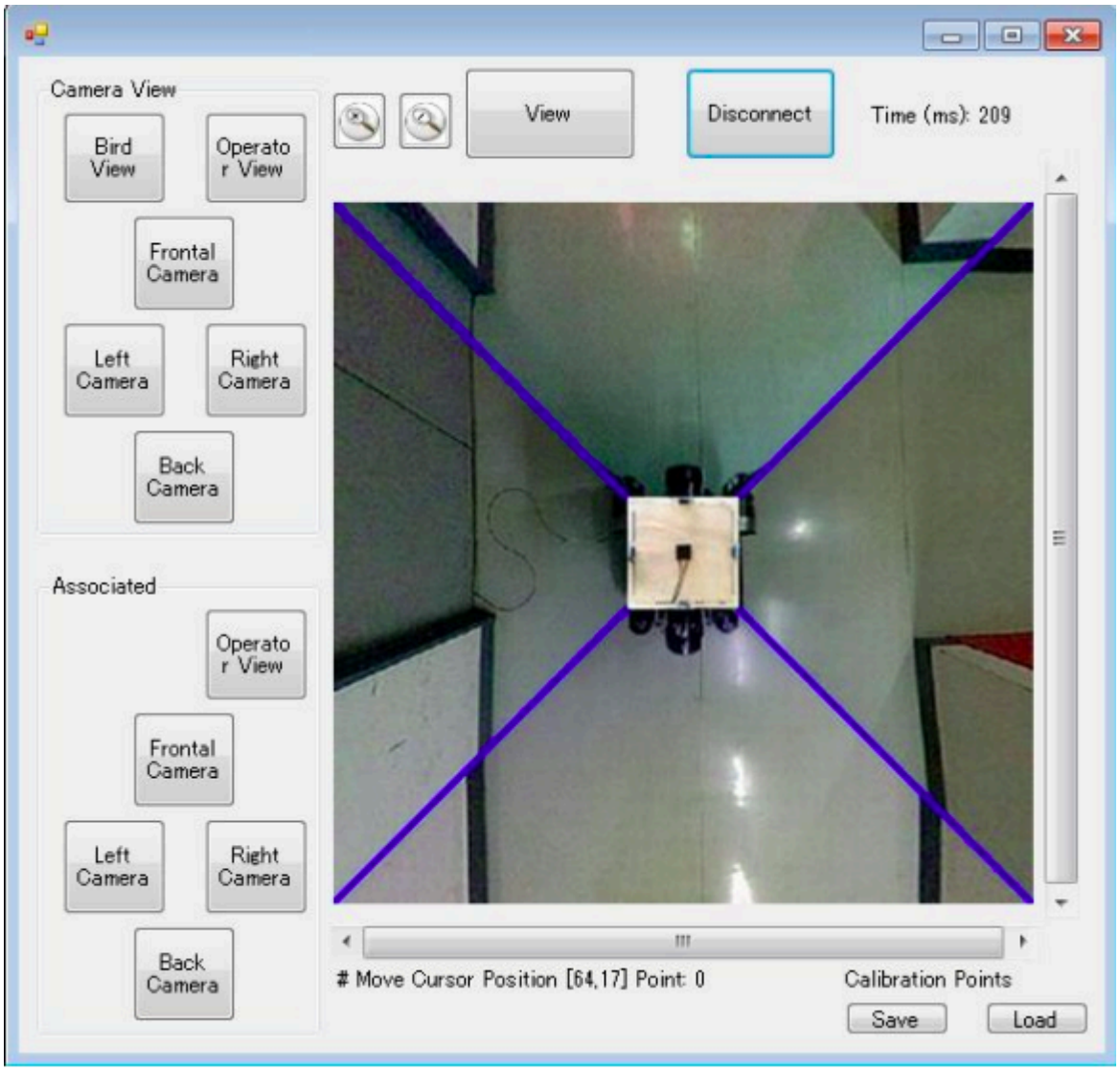



**Figure 9:** *Teleoperators' graphical user interface showing the created bird's-eye view* [24].

The reviewed related works provide valuable insights and technologies to enhance situational awareness in teleoperation systems. Each solution has its own limitations that impact their

suitability for autonomous vehicle teleoperation, however they provide a good starting point for creating a new multi angle interface for an autonomous vehicles teleoperations system.

# 3 Implementation

Currently the Autonomous Driving Labs vehicle has a basic teleoperation system installed by Clevon (figure 10), where the teleoperator can see the video from the cars front facing cameras and does not have a 360 degree overview of the vehicles environment.

**Figure 10:** *Current teleoperation system used by ADL.*

The objective of this thesis is to create an interface for the teleoperator to have a better situational awareness by displaying multiple viewpoint images including a bird's-eye view. During the development, various available software is evaluated and validated. It also includes modifying or creating new software and features.

## 3.1 Requirements

The Autonomous Driving Lab's vehicle is equipped with necessary sensors and hardware that are required for basic autonomy. The car has 2 LiDAR sensors, cameras, radar and a powerful onboard computer with the following specifications [4]:

- CPU: Intel Xeon
- Memory: 32GB
- Storage: 1TB SSD
- GPU: NVIDIA RTX 2080 Ti
- Networking: 6x Gigabit Ethernet

- OS: Ubuntu 20.04
- ROS distribution: Noetic

The software used is Autoware Mini, which is built using Python and ROS (Robot Operating System) and is based on an open-source software stack for self-driving called Autoware [25]. Robot Operating System is an open source software framework that includes various tools and libraries to help build applications for robots. The purpose of ROS is to standardise and to support code reuse in robotics software development [26].

Based on the software and the architecture of the ADL-s autonomous vehicle the following requirements are:

### 3.1.1 Functional requirements

- The interface displays a camera-based bird's-eye view
- The interface displays a 3D scene from LiDAR and camera fusion
- The teleoperator can freely change viewing angles
- The teleoperator has a control panel to snap into different views
- The interface shows the trajectory of the car based on the steering angle

### 3.1.2 Nonfunctional requirements

- The interface is compatible with ROS Noetic and Ubuntu 20.04

As tasks in image processing can be computationally heavy, a computer with the following specifications is used for development:

- CPU: AMD Ryzen 5 5600
- RAM: 32 GB
- GPU: NVIDIA RTX 3080
- OS: Ubuntu 20.04
- ROS distribution: Noetic

## 3.2 Interface overview

As a part of this thesis a prototype version of the teleoperation interface was developed (figure 11). The solution uses RViz as the base due to its capability to easily display different sensor data like camera feeds, point clouds, depth clouds and robot models in a three dimensional space and create custom control panels [27]. The remote human operator can view the vehicle's environment from a camera-based view or as a point cloud representation. Also a panel was created, so with a click of a button the viewing angle would snap in place. To give the operator a better understanding of where the car is headed, the car's trajectory is also visualised in the interface.

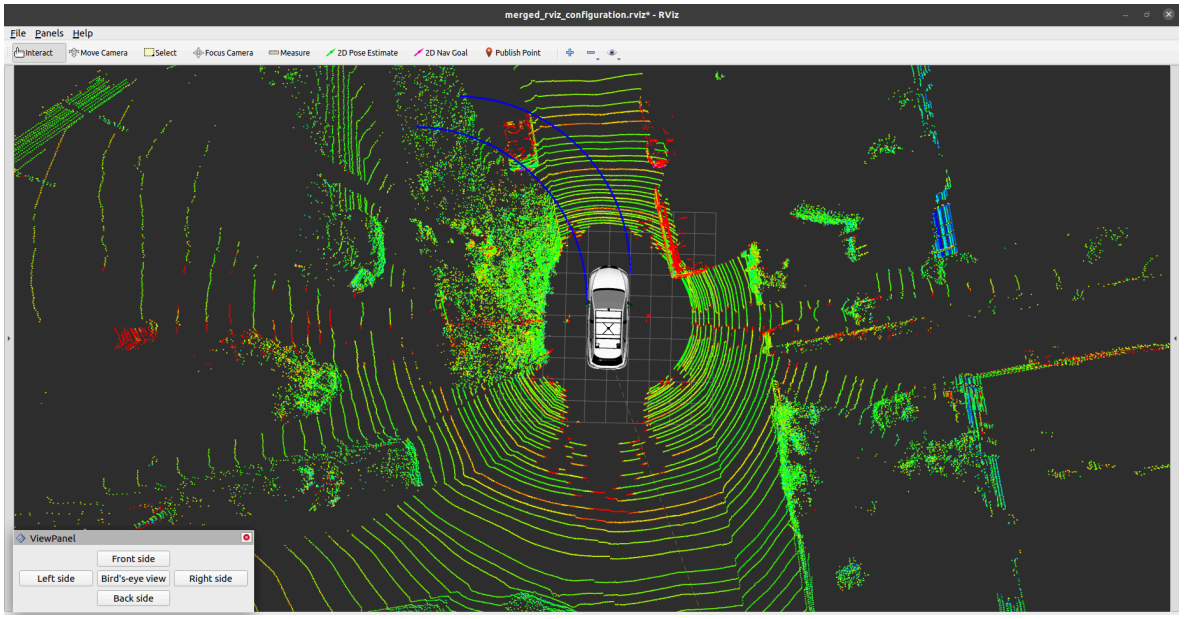



**Figure 11:** *Created interface displaying the car model and surroundings in a bird's-eye view with future trajectory of the vehicle and control panel.*

The developed solution can be divided into two main parts: the image processing and getting data from the sensors, which is done on the vehicle's computer and the multi angle interface displayed on the teleoperator's computer (figure 12). On the vehicles side, two fisheye camera feeds are taken and merged together to create a one image that contains spherical images. The merged image is further processed and transformed into a equirectangular projection that should be sent to the remote operator's interface and projected onto a 3D mesh. From the vehicle side, the point cloud is also created from the data gathered from LiDAR and should be sent to the teleoperator's interface where it is displayed as a 3D scene.

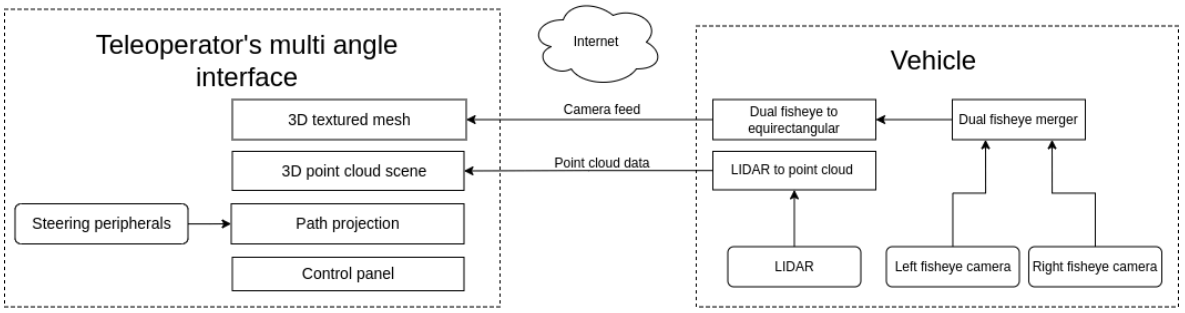



**Figure 12:** *Diagram of the proposed solution displaying the interface and image processing.*

The whole solution is designed to use ROS, which is designed to be modular and uses nodes which are individual processes. Nodes can communicate with each other through topics. A node can publish to a topic to send data and also subscribe to a topic to receive data. There can be multiple publishers and subscribers for one topic and one node can publish and subscribe to multiple topics [28]. The created interface is formatted as a ROS package called multi_angle_interface [29]. To easily run the software, the package has two launch files. A launch file called *start_interface.launch* needs to be started on the teleoperators machine as it starts up the RViz interface and a node for creating the vehicle trajectory, a launch file called *start_visual_data.launch* needs to be started on the vehicle to get the image feed from the fisheye cameras and process it to a equirectangular projection. On figure 13 a ROS node graph describes how each node subscribes and publishes to a topic and what message types are used.

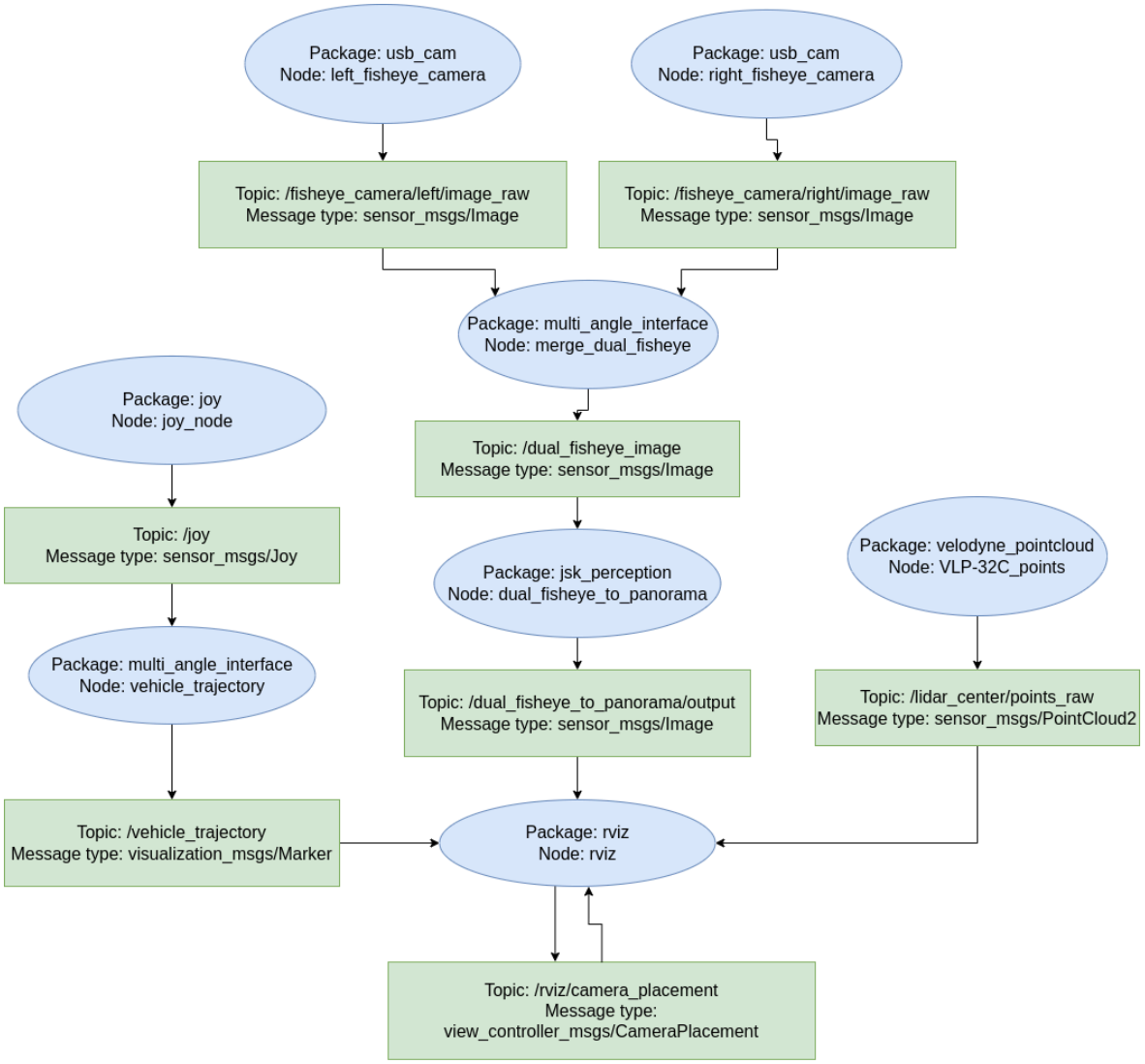



**Figure 13:** *ROS node graph of the interface.*

### 3.2.1 Visual perception

To improve situational awareness, the teleoperator can view the car's surroundings from the camera feed or as a point cloud representation. To visualise the point cloud, RViz's built-in capabilities were used. The display type must be set to PointCloud2 and then as the input the correct ROS topic must be given. The point cloud view allows the teleoperator to inspect any object from multiple perspectives (figure 11). To display a bird's-eye view, the RViz's camera viewing angle has to be set so that it views the point cloud from the top. It can be done through the control panel or with a computer mouse.

The teleoperator also has a choice to view the surroundings from a camera-based view (figure 14). For this view, the rviz_textured_sphere [22] package was used as a base and modified to

use a hemisphere mesh similar to the Fujitsu Multi Angle system. The hemisphere was created by using the parametric sphere equations:

$$x = r\,sin(\phi)\,cos(\theta)$$
$$y = r\,sin(\phi)\,sin(\theta)$$
$$z = r\,cos(\phi)$$

where $\phi$ ranges from 0 to $\pi/2$ to limit the mesh vertically from the equator to the bottom pole and $\theta$ ranges from 0 to $2\pi$ for a complete circular coverage horizontally. The created mesh also has a flattened bottom, so that a 3D model of the vehicle can be placed on the mesh and mimic the view of what would be seen from above.

**Figure 14:** *Dual fisheye camera images projected on a hemisphere mesh at a bird's-eye view angle.*

Before the image is projected to the hemisphere, it is first blended together to create an equirectangular projection. Equirectangular projection linearly maps the longitude and latitude of a sphere image to a rectangular image and produces one 2D 360 degree image. This method is typically used for 360-degree videos, where the created image is projected onto a viewing sphere and the viewer can freely choose a viewing direction [30]. This projection is needed as the circular fisheye image is designed to cover a hemisphere, which in

a 2D plane is a circle. If the fisheye image is to directly be mapped to one half of a hemisphere, it would create black borders around the edges (figure 15a) that come from the circular fisheye image. Equirectangular projection maps the entire panorama onto a rectangular image and removes the black borders (figure 15b).

(a)

(b)

**Figure 15:** *Dual fisheye camera images (a) projected as an equirectangular presentation (b).*

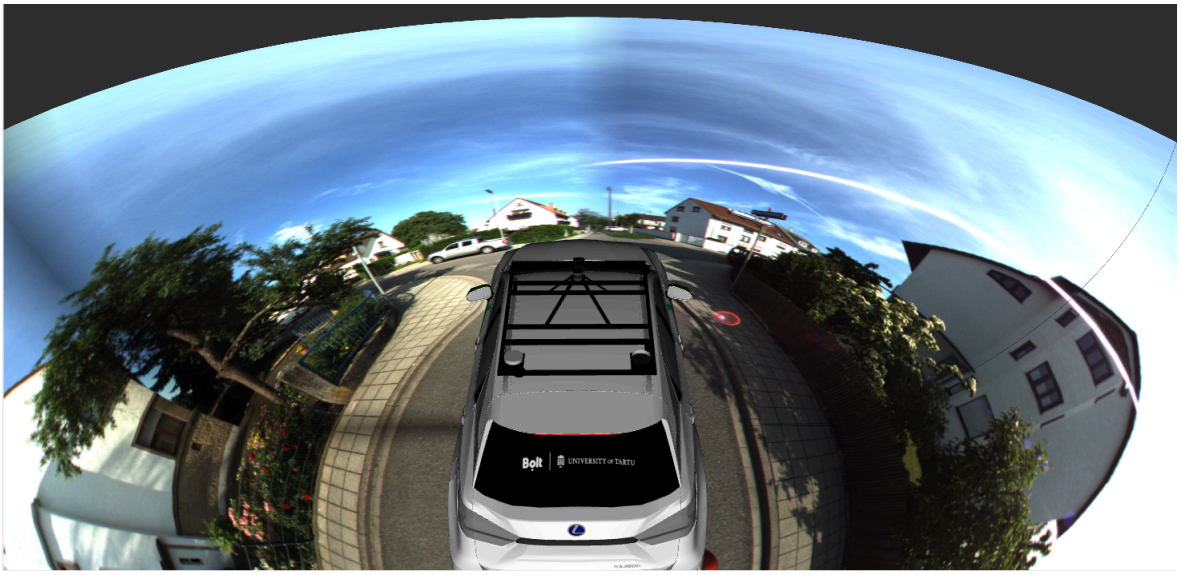
The equirectangular projection is done using a ROS package called jsk_perception, which contains scripts for different kinds of 2D image perception tasks [31]. The created image is then mapped inside the created hemisphere mesh, where the viewing angle of the hemisphere can be changed through the control panel or with a mouse.

### 3.2.2 View control panel

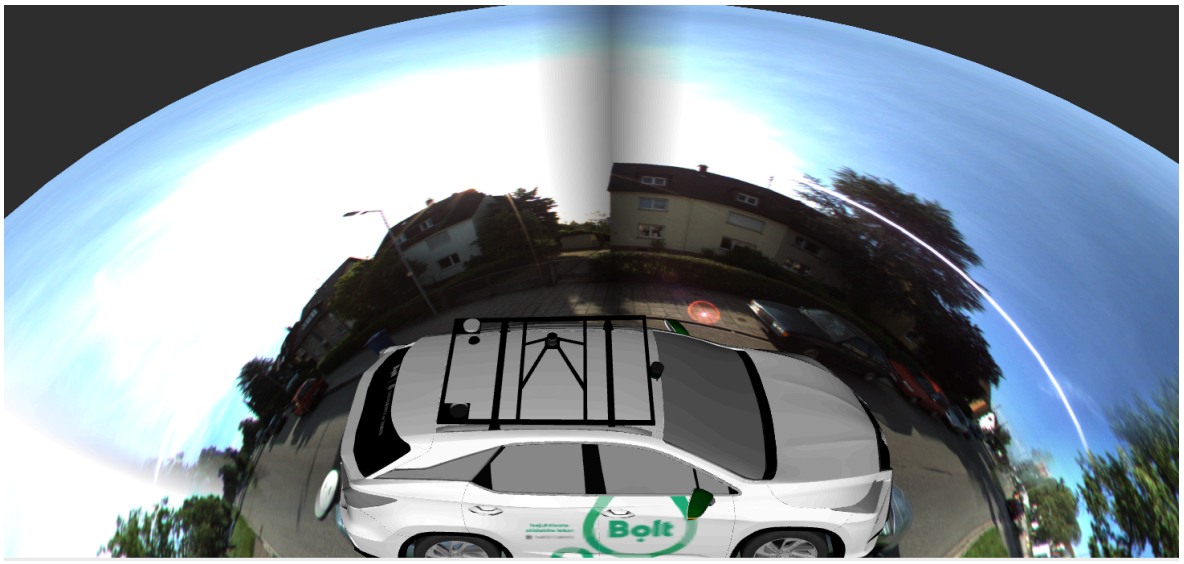
To quickly snap into a different viewing angle of the environment, a small panel was created. The panel consists of five buttons (figure 16), which allows the operator to pan the view to the left, right, front and back side of the car or to a bird's-eye view (figure 17). The panel was created using C++ and graphical user interface elements that exist in RViz. The panel can be docked in the main window of the interface or it can be dragged out of the window and placed anywhere on the screen.


**Figure 16:** *RViz control panel to quickly change between viewing angles.*

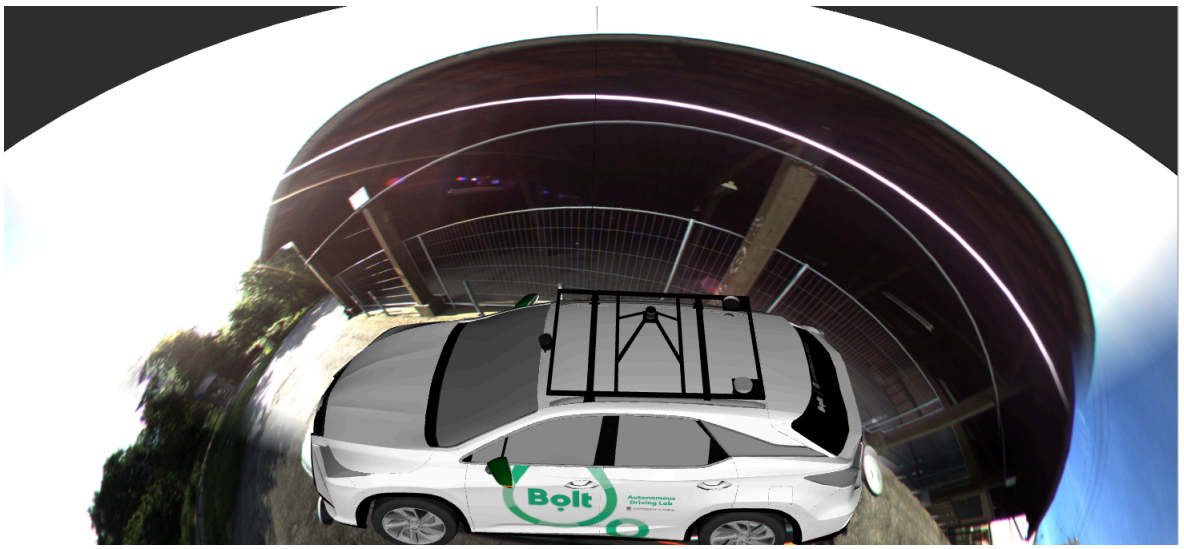
To change the viewing angle via the buttons, a package called rviz_animated_view_controller [32] was used. This package allows users to change the RViz's virtual camera angle by sending a message to the /rviz/camera_placement topic [32]. The buttons on the panel send a message to that topic with different predefined parameters for each view.



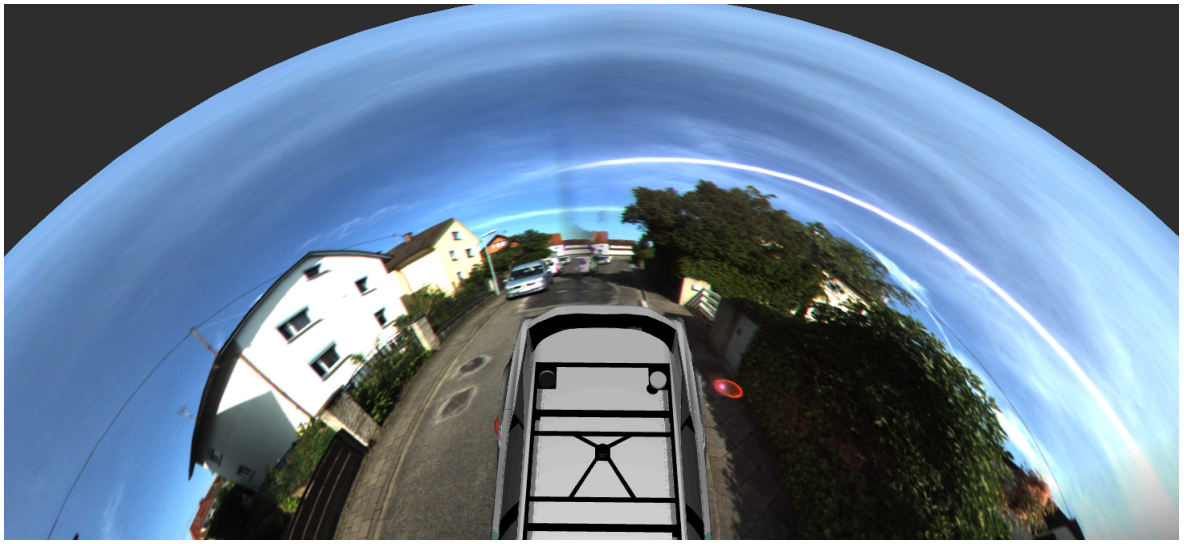

(a)



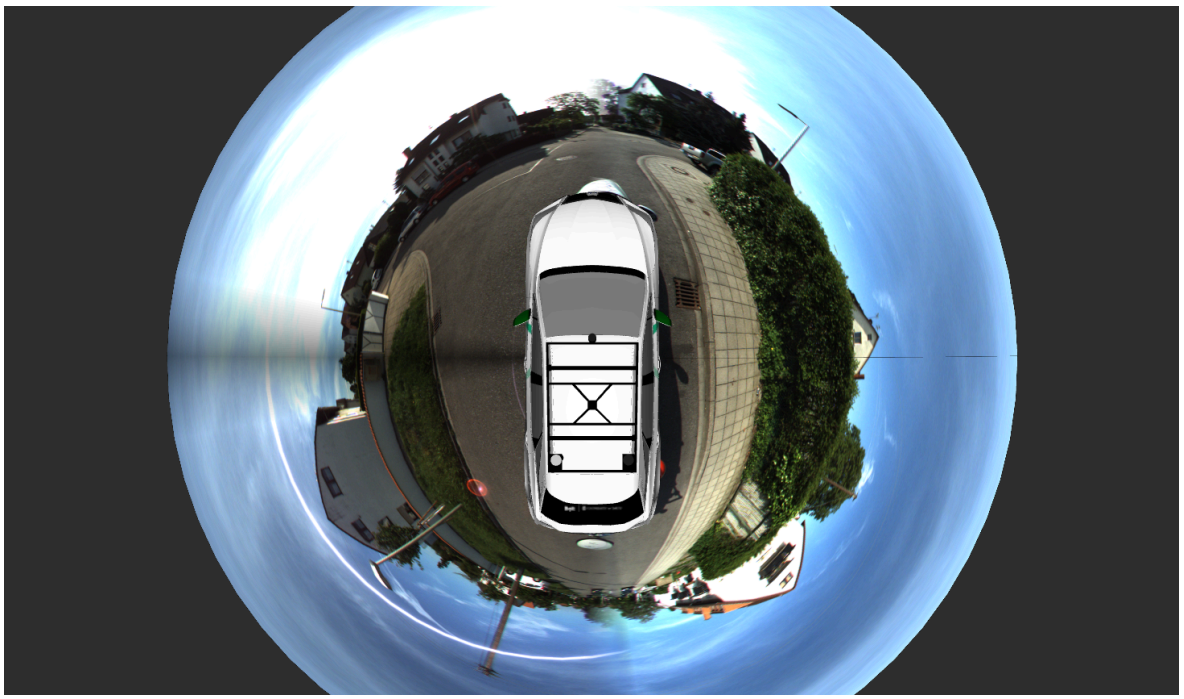

(b)




(c)



(d)



(e)

**Figure 17:** *Different angles that can be snapped to with the control panel: front view (a), left view (b), right view (c), back view (d) and bird's-eye view (e).*

### 3.2.3 Vehicle trajectory

To give feedback about the vehicle's path, a future trajectory calculation was implemented (figure 18). The trajectory of the vehicle was modelled based on the Ackermann steering geometry, which is designed to trace out circles of different radii for the wheels on the inside and outside of a turn [33]. The future path of the vehicle is calculated based on the steering angle, wheelbase and track width of the vehicle. Steering angle is taken from a Logitech G29 steering wheel by using a ROS joy package, which publishes a message that contains the current state of a controller's joysticks and buttons to a topic (figure 13) [34]. The calculated points are visualised by using Markers, which allow to display different shapes in RViz by publishing the marker messages to a topic.

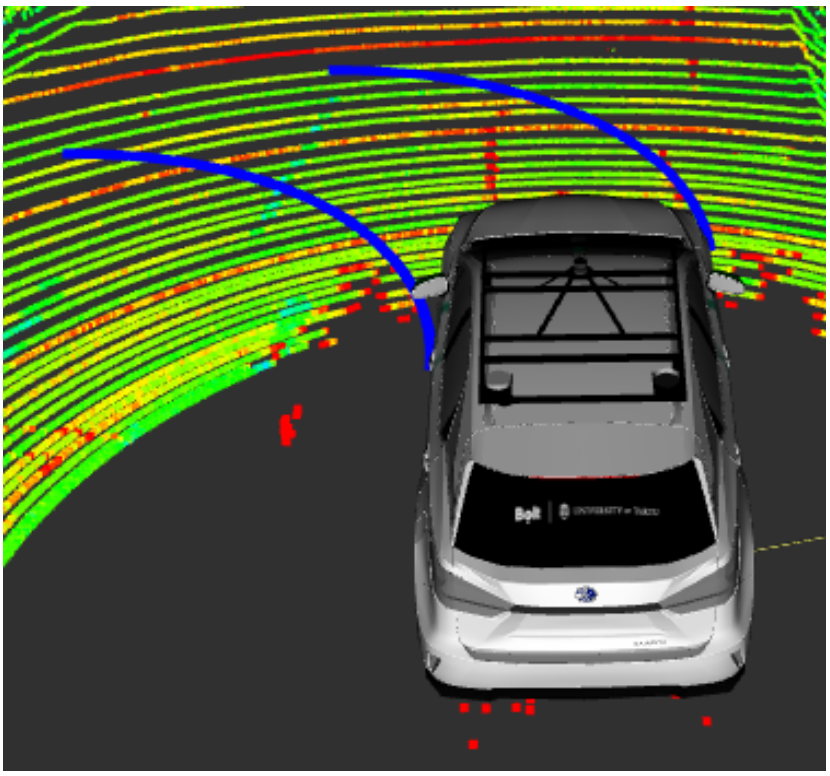


**Figure 18:** *Vehicle's future trajectory visualised with blue lines.*

## 3.3 Challenges

During the development of the interface, some issues and challenges were encountered. In this subsection an overview of the main problems is given and what methods were tried to solve them.

### 3.3.1 Image alignment and stitching

The rviz_textured_sphere package's one main problem is that it assumes that the two fisheye images are ideal and that the camera centres match. Due to this assumption the combined spherical image has artefacts (figure 19). The package was also tested in real life with two KODAK PIXPRO SP360 4K fisheye cameras to test if the artefacts could be lessened by mounting the cameras back-to-back or putting some distance between the units. The different positionings did not reduce the artefacts, so to create a better and more accurate seam between two images, image stitching was tested.

**Figure 19:** *Body of a car is not aligned correctly on the seam of the combined images.*

CS205-ImageStitching is a software library that uses C++ and OpenCV to stitch photos or live video streams together to create a seamless high-resolution image [35]. The process to create a stitched image contains several key steps. The first step is to identify the keypoints on the two images. Usually the keypoints are more distinguished features like corners and edges of an object. To detect these features the library uses the Speed-Up Robust Features (SURF) algorithm. Next, the keypoints of the two images are matched using Euclidean distance to get the best result. Once the matching keypoints have been established, a transformation matrix is derived by using the Random Sample Consensus algorithm. If the matrix is created then the image is projected to the right side of the plane where the image on the left is at and then the image is stitched together by placing pixels from both images to a blank canvas [35]. Due to the process being computationally heavy, shared memory parallelization and GPU acceleration methods were also used to speed up the process. The image stitching libraries were tested with a lower 720p resolution Logitech C270 web cameras. The results by using shared memory parallelization OpenMP produced an image with lots of artefacts and a frame rate of 1 - 5 frames per second (figure 20). The results were a bit better with the OpenACC GPU acceleration as the frame rate ranged from 11 - 15 frames per second, but the stitched image produced a lot of artefacts, especially when there was movement in the video feed (figure 21). From these results, it can be concluded that

stitching is not a good choice for creating a real-time seamless image as it is a computationally heavy task and affects the video frequency and image quality and can impact the remote driving negatively.


**Figure 20:** *Stitched camera feed using SURF and OpenMP*

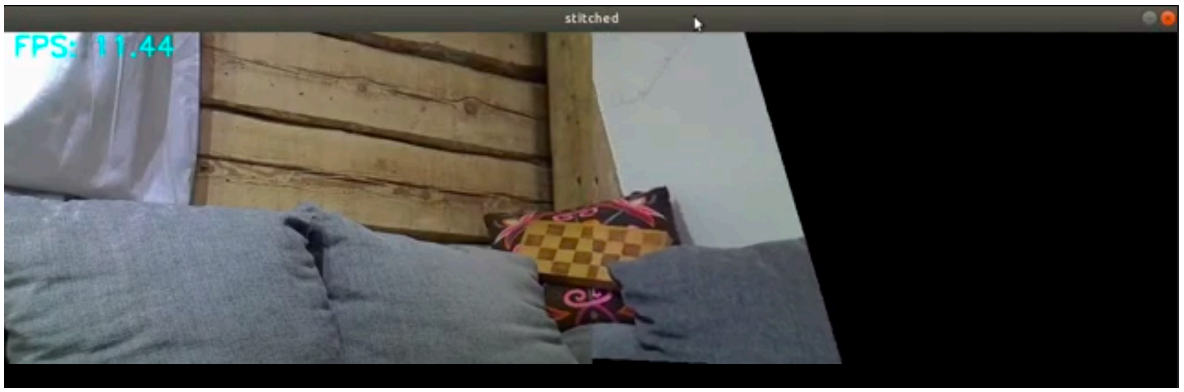



**Figure 21:** *Stitched camera feed using SURF and OpenACC*

The stitching was also tested by switching out the SURF algorithm with Scale-Invariant Feature Transform (SIFT) algorithm for keypoint detection. The result was similar compared to using the SURF algorithm and GPU acceleration with a video frequency of 11 - 15 frames per second and unstable picture quality when there was movement (figure 22).

**Figure 22:** *Unstable stitched image using SIFT and OpenACC.*

### 3.3.2 Image and point cloud alignment

Issues also came up while creating the bird's-eye view by trying to use sensor fusion by painting the LiDAR point cloud with the colour info from the fisheye cameras. The pointcloud_painter [22] package was explored. The first problem was that the package only took input and data from a rosbag. After modifying the code to work in real-time and setting up the parameters for our setup, the package failed to project colour onto most of the depth points. For example, out of 92415 depth points, it only assigned colour to 4537 and the colour of the objects was not correct. Another problem was that the package's performance was quite slow. After modifying the parameters and enabling image compression the best result was one painted point cloud was created in 4 - 5 seconds.

Also another point cloud painting package was looked into. The package color_cloud_from_image is a package part of the aforementioned Omnidirectional Vision framework and allows to colourize LiDAR point clouds from a 360 degree camera [36]. The main issue with this package was that there was no documentation on how to use it and the package would fail at the building stage. From the error messages it seemed that the software was not compatible with ROS Noetic and Ubuntu 20.04.

Although these problems remain unsolved currently, working to solve these issues gives a good understanding of what future work needs to be done.

## 3.4 Testing

During the implementation, the KITTI-360 dataset was used to test the created interface as it has a similar set of sensors to the ADL self-driving car. KITTI-360 is a suburban driving dataset, the successor of the popular KITTI dataset, that offers a rich 360° data. The data is

obtained using fisheye cameras and LiDARs and other sensors that are mounted on top of a vehicle (figure 23) [37]. The dataset contains over 300 000 images and over 100 000 laser scans from a driving distance of 73.7 km [38]. To use the dataset with ROS, the kitti360_ros_player is used to publish all sensors to ROS topics. This package allows to pause, speed up or go through the simulation frame by frame [39].

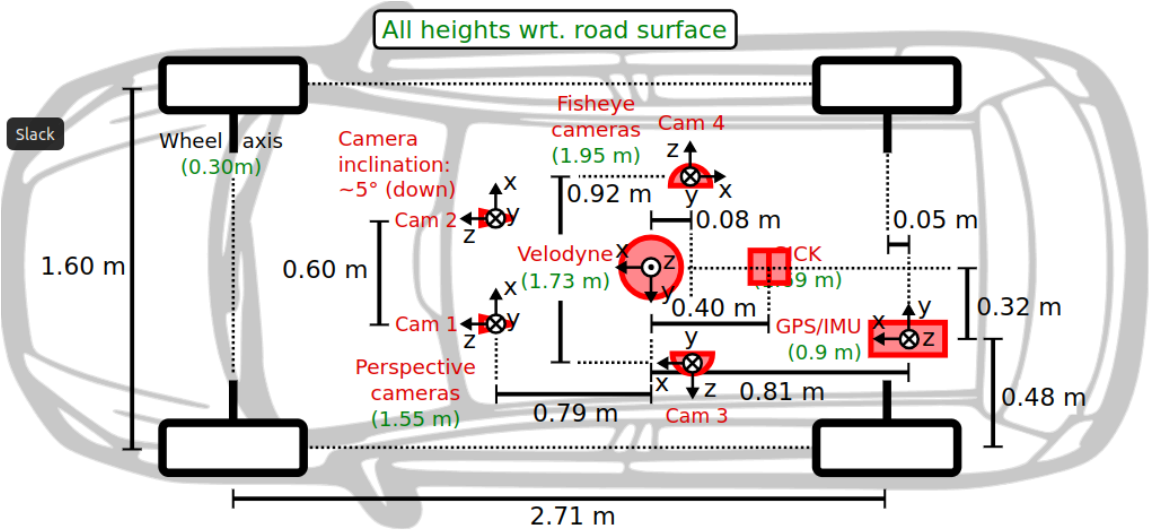



**Figure 23:** *Camera and sensor placement on the KITTI-360 vehicle* [37].

# 4 Discussion

In this section the results and limitations of the created interface are discussed. To improve the interface and reduce limitations, some ideas are proposed for further development.

During this thesis an interface was developed for a self-driving vehicles teleoperations system to enhance the situational awareness for the human remote operator. The software is formatted as a ROS package and can be found on GitHub [29]. Almost all the set requirements were met, as the operator can freely change the viewing angle and use the created control panel for it, the vehicle's future path is displayed. The teleoperator can also change between a camera-based and point cloud view. The requirement that was not met was that the point cloud and camera feed are fused together to create a colourized point cloud. Although most of the requirements were met there are still limitations and points of improvement.

## 4.1 Limitations and future work

Currently the solution is built on using two fisheye cameras that are mounted on top of the car. If more cameras are to be added, then one solution would be to modify the equirectangular projection code so that instead of two fisheye cameras, it takes more than two cameras as input and creates a one 360 degree image. Also if needed the images could be projected straight to the 3D mesh, but then the texture mapping should be edited, so that each camera image would get a section of the hemisphere.

Another problem that exists in the current solution is that the camera-based view has quite a lot of barrel distortion. Barrel distortion is common with wide-angle fisheye lenses and causes the image's edges to look curved [40]. One solution to remove barrel distortion is to replace the fisheye cameras with rectilinear wide angle cameras. This approach might need more than two cameras as rectilinear lenses have a smaller field of view than fisheye lenses. Another way to reduce barrel distortion from fisheye images is to use rectilinear projection, which is a common method used to map a curved surface to a flat plane, where straight lines in a 3D space are also preserved on the 2D image [41]. This projection limitation is that the edges of the projection may have extreme distortion, so the field of view of the created image is limited. To overcome this issue panorama viewers apply the rectilinear transformation in proportions [41]. So to apply the rectilinear projection in, we would have to get the position

of the RViz's virtual camera and apply the transformation to the viewed proportion of the whole image.

Currently the interface displays a point cloud, which gives the teleoperator a better depth sense, but it lacks colour, so it may be hard to identify the surrounding objects. Multiple packages to add colour information to the point cloud were explored, but did not give a usable result. One approach may be to take the aforementioned packages as a base and create an improved version that could handle large point clouds in real-time.

Another limitation lies in displaying the vehicle's future path, as currently the path is only projected to the front of the vehicle. A solution to this problem may be by subscribing to a Gear report ROS topic from Autoware, which gives information of what gear the car is currently in [42]. If the gear is known, then the path of the car can be projected to the front or back, according to the gear.

For further testing and development, the created interface should be migrated to the ADL's vehicle. First, two fisheye cameras need to be mounted on top of the vehicle and as the vehicle already has a frame where other sensors are mounted, the cameras may be also mounted onto it. Secondly, the car currently also has a teleoperation system that was developed and installed by Clevon, so it should be explored how the interface could be integrated to the existing solution [43].

# 5 Conclusion

Teleoperation systems are needed to help self-driving vehicles come out of difficult situations. To give a better overview of complex environments to the human remote operator, 360 degree multi angle view and bird's-eye view are used. To get the a better understanding of the surroundings, it is better to use data from multiple sensors and combine them together, as only camera-based view gives a good overall awareness, but lacks depth cues and perception and point cloud generated by LiDARs give a better 3D scene and depth information, but lack colour information.

The goal of this thesis is to create an interface for the teleoperator to have a better situational awareness by displaying multiple viewpoint images including a bird's-eye view. As a result, a prototype interface was created, which allows the teleoperator to view the vehicles surroundings from a camera-based multi angle view or as a point cloud, the operator can also quickly change viewing angles by using a control panel.

Despite the created systems limitations and issues, it provides a good starting point for further development and testing of the current solution.

# Bibliography


[1] D. J. Fagnant and K. Kockelman, ‘Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations’, *Transp. Res. Part Policy Pract.*, vol. 77, pp. 167–181, 2015.

[2] D. Majstorović, S. Hoffmann, F. Pfab, A. Schimpe, M.-M. Wolf, and F. Diermeyer, ‘Survey on teleoperation concepts for automated vehicles’, in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2022, pp. 1290–1296.

[3] F. Tener and J. Lanir, ‘Driving from a distance: challenges and guidelines for autonomous vehicle teleoperation interfaces’, in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–13.

[4] ‘Autonomous Driving Lab - Vehicle’, Autonomous Driving Lab. Accessed: Oct. 12, 2023. [Online]. Available: https://adl.cs.ut.ee/lab/vehicle

[5] T. Fong and C. Thorpe, ‘Vehicle teleoperation interfaces’, *Auton. Robots*, vol. 11, pp. 9–18, 2001.

[6] P. F. Hokayem and M. W. Spong, ‘Bilateral teleoperation: An historical survey’, *Automatica*, vol. 42, no. 12, pp. 2035–2057, 2006.

[7] M. A. Khan, H. El Sayed, S. Malik, M. T. Zia, N. Alkaabi, and J. Khan, ‘A journey towards fully autonomous driving-fueled by a smart communication system’, *Veh. Commun.*, vol. 36, p. 100476, 2022.

[8] I. Barabas, A. Todoruţ, N. Cordoş, and A. Molea, ‘Current challenges in autonomous driving’, in *IOP conference series: materials science and engineering*, IOP Publishing, 2017, p. 012096.

[9] V. S. R. Kosuru and A. K. Venkitaraman, ‘Advancements and challenges in achieving fully autonomous self-driving vehicles’, *World J. Adv. Res. Rev.*, vol. 18, no. 1, pp. 161–167, 2023.

[10] M. Moniruzzaman, A. Rassau, D. Chai, and S. M. S. Islam, ‘Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey’, *Robot. Auton. Syst.*, vol. 150, p. 103973, 2022.

[11] J. Y. Chen, E. C. Haas, and M. J. Barnes, ‘Human performance issues and user interface design for teleoperated robots’, *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 37, no. 6, pp. 1231–1245, 2007.

[12] A. Pujol Miró, ‘Real-time image stitching for automotive 360º vision systems’, B.S. thesis, Universitat Politècnica de Catalunya, 2014.

[13] C. Eising, L.-F. Pereira, J. Horgan, A. Selvaraju, J. McDonald, and P. Moran, ‘2.5 D vehicle odometry estimation’, *IET Intell. Transp. Syst.*, vol. 16, no. 3, pp. 292–308, 2022.

[14] L. Reiher, B. Lampe, and L. Eckstein, ‘A Sim2Real Deep Learning Approach for the Transformation of Images from Multiple Vehicle-Mounted Cameras to a Semantically Segmented Image in Bird’s Eye View’. 2020.

[15] H. Li *et al.*, ‘Delving into the devils of bird’s-eye-view perception: A review, evaluation and recipe’, *IEEE Trans. Pattern Anal. Mach. Intell.*, 2023.

[16] M. Oehler and O. von Stryk, ‘A flexible framework for virtual omnidirectional vision to improve operator situation awareness’, in *2021 European Conference on Mobile Robots (ECMR)*, IEEE, 2021, pp. 1–6.

[17] Y.-C. Liu, K.-Y. Lin, and Y.-S. Chen, ‘Bird’s-eye view vision system for vehicle surrounding monitoring’, in *International Workshop on Robot Vision*, Springer, 2008, pp. 207–218.

[18] I.-C. Lo, K.-T. Shih, and H. H. Chen, ‘Image Stitching for Dual Fisheye Cameras’, *2018 25th IEEE Int. Conf. Image Process. ICIP*, pp. 3164–3168, 2018.

[19] S. Shimizu, S. Taniguchi, T. Kidena, H. Yamada, and M. Yamada, ‘Multi angle vision system to supplement driver’s visual field’, *Fujitsu Ten Tech J*, vol. 36, pp. 12–18, 2011.

[20] ‘3D Bird Eye Panoramic Rearview Mirror Camera For Lexus RX Car Dashboard System Reverse Parking Assistance 360 Degree Surround Camera 4K - AliExpress 34’. Accessed: May 14, 2024. [Online]. Available: https://fr.aliexpress.com/item/1005005816664327.html?gatewayAdapt=glo2fra

[21] ‘RViz textured sphere plugin’. UTexas Nuclear and Applied Robotics Group (Public), Nov. 30, 2023. Accessed: Dec. 29, 2023. [Online]. Available: https://github.com/UTNuclearRoboticsPublic/rviz_textured_sphere

[22] V. Vunder, R. Valner, C. McMahon, K. Kruusamäe, and M. Pryor, ‘Improved Situational Awareness in ROS Using Panospheric Vision and Virtual Reality’, in *2018 11th International Conference on Human System Interaction (HSI)*, 2018, pp. 471–477. doi: 10.1109/HSI.2018.8431062.

[23] ‘UTNuclearRoboticsPublic/pointcloud_painter: Projects a spherical RGB image onto a colorless pointcloud to create an RGBD pointcloud of a scene.’ Accessed: Apr. 23, 2024. [Online]. Available: https://github.com/UTNuclearRoboticsPublic/pointcloud_painter?tab=readme-ov-file

[24] T. Sato, A. Moro, A. Sugahara, T. Tasaki, A. Yamashita, and H. Asama, ‘Spatio-temporal bird’s-eye view images using multiple fish-eye cameras’, in *Proceedings of the 2013 IEEE/SICE international symposium on system integration*, IEEE, 2013, pp. 753–758.

[25] ‘autowarefoundation/autoware: Autoware - the world’s leading open-source software project for autonomous driving’. Accessed: May 03, 2024. [Online]. Available: https://github.com/autowarefoundation/autoware

[26] ‘ROS/Introduction - ROS Wiki’. Accessed: Apr. 23, 2024. [Online]. Available: https://wiki.ros.org/ROS/Introduction

[27] ‘rviz - ROS Wiki’. Accessed: Apr. 24, 2024. [Online]. Available: http://wiki.ros.org/rviz

[28] M. Quigley *et al.*, ‘ROS: an open-source Robot Operating System’, in *ICRA workshop on open source software*, Kobe, Japan, 2009, p. 5.

[29] ‘rauno99/multi_angle_interface’. Accessed: May 14, 2024. [Online]. Available: https://github.com/rauno99/multi_angle_interface

[30] W. Yang, Y. Qian, J.-K. Kämäräinen, F. Cricri, and L. Fan, ‘Object detection in equirectangular panorama’, in *2018 24th international conference on pattern recognition (icpr)*, IEEE, 2018, pp. 2190–2195.

[31] ‘jsk-ros-pkg/jsk_recognition’. jsk-ros-pkg, May 05, 2024. Accessed: May 05, 2024. [Online]. Available: https://github.com/jsk-ros-pkg/jsk_recognition

[32] ‘rviz_animated_view_controller - ROS Wiki’. Accessed: May 01, 2024. [Online]. Available: http://wiki.ros.org/rviz_animated_view_controller

[33] ‘Ackermann steering geometry - Wikipedia’. Accessed: May 03, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Ackermann_steering_geometry
[34] ‘joy - ROS Wiki’. Accessed: May 03, 2024. [Online]. Available: http://wiki.ros.org/joy
[35] ‘CS205 Real Time Image Stitching’. Accessed: Apr. 25, 2024. [Online]. Available: https://cs205-stitching.github.io/
[36] ‘tu-darmstadt-ros-pkg/color_cloud_from_image: Color point cloud data from camera images’. Accessed: May 03, 2024. [Online]. Available: https://github.com/tu-darmstadt-ros-pkg/color_cloud_from_image
[37] Y. Liao, J. Xie, and A. Geiger, ‘Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d’, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3292–3310, 2022.
[38] ‘autonomousvision/kitti360Scripts: This repository contains utility scripts for the KITTI-360 dataset.’ Accessed: Apr. 25, 2024. [Online]. Available: https://github.com/autonomousvision/kitti360Scripts
[39] ‘dcmlr/kitti360_ros_player: ROS package to publish the KITTI-360 dataset.’ Accessed: May 02, 2024. [Online]. Available: https://github.com/dcmlr/kitti360_ros_player
[40] R. G. de A. Azevedo, N. Birkbeck, F. De Simone, I. Janatra, B. Adsumilli, and P. Frossard, ‘Visual distortions in 360° videos’, *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 8, pp. 2524–2537, 2019.
[41] ‘Rectilinear Projection - PanoTools.org Wiki’. Accessed: May 08, 2024. [Online]. Available: https://wiki.panotools.org/Rectilinear_Projection
[42] ‘Vehicle Interface - Autoware Documentation’. Accessed: May 09, 2024. [Online]. Available: https://autowarefoundation.github.io/autoware-documentation/main/design/autoware-interfaces/components/vehicle-interface/
[43] ‘Clevon andis Tartu ülikooli isejuhtivale sõidukile kaugjuhtimise võimekuse’. Accessed: May 09, 2024. [Online]. Available: https://sakala.postimees.ee/7995710/clevon-andis-tartu-ulikooli-isejuhtivale-soidukile-kaugjuhtimise-voimekuse

# Licence

**Non-exclusive licence to reproduce the thesis and make the thesis public**

I, Rauno Põlluäär,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

   **Designing and implementing a bird's-eye view interface for a self-driving vehicle's teleoperation system**

   supervised by Karl Kruusamäe.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in points 1 and 2.
4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Rauno Põlluäär
**15.05.2024**