University of Tartu
Faculty of Science and Technology
Institute of Technology

Erki Veeväli

Development of a continuous teleoperation system for urban road vehicle

Bachelor's thesis (12 ECTS)
Computer Engineering

Supervisors:
PhD Karl Kruusamäe
PhD Ulrich Norbisrath

Tartu 2023

Abstract

Development of a continuous teleoperation system for urban road vehicle

Teleoperation is a remote control technology that also provides situational awareness to the remote operator via camera feeds and other data. In the context of road vehicles, teleoperation is needed for testing the self-driving abilities as well as assisting the vehicle or implementing a fully teleoperated solution on a similar platform. At the University of Tartu, a self-driving vehicle is being developed at Autonomous Driving Lab (ADL), which currently lacks a suitable teleoperation solution.

The goal of this thesis is to develop a prototype teleoperation solution, which could be later used on the actual self-driving vehicle. During the process multiple software solutions are compared, taking into account previous teleoperation test results from ADL as well. The result is a teleoperation prototype that uses RTCBot, which is a Python library for teleoperation. Finally the prototype is tested on the Robotont platform

CERCS: T120 Systems engineering, T125 Robotics
Keywords: teleoperation, self-driving vehicles

Lühikokkuvõte

Linnasõiduki pideva kaugjuhtimissüsteemi arendus

Teleopereerimine on kaugjuhtimistehnoloogia, mille puhul tagatakse kaugjuhile pidev olukorrateadlikkus, seda nii läbi videopildi roboti ümber toimuvast kui ka muudelt sensoritelt saadava info abil. Autode kontekstis on teleopereerimist vaja peamiselt isejuhtivate autode arenduses ja testimisel või ka sarnastel sõidukiplatformidel eraldi lahedndusena kasutamiseks. Tartu Ülikoolis arendatakse Isejuhtivate sõidukite laboris autot, millel hetkel puudub sobilik kaugjuhtimise lahendus.

Antud töö eesmärgiks on luua isejuhtiva auto jaoks kasutatava kaugjuhtimissüsteemi prototüüp. Sealjuures võrreldakse erinevaid tarkvaravõimalusi selle loomiseks, võttes arvesse ka eelnevate iesjuhtival sõidukil läbi viidud kaugjuhtimistestide tulemusi. Tulemusena valmib kaugjuhtimissüsteem kasutades RTCBoti, mis on spetsiaalselt kaugjuhtimise tarbeks kasutatav pythoni teek ning kaugjuhtimist testitakse Robotont robotiplaformil.

CERCS: T120 Süsteemitehnoloogia, T125 Robootika
Märksõnad: kaugjuhtimine, isejuhtivad autod

Contents

Acronyms

ADL - Autonomous Driving Lab

ROS - Robot Operating System

OS - Operating System

CPU - Central Processing Unit

RAM - Random Access Memory

GPU - Graphics Processing Unit

API - Application Programming Interface

IEEE - Institute of Electrical and Electronics Engineers

HTML - HyperText Markup Language

LAN - Local Area Network

VPN - Virtual Private Network

Introduction

At the University of Tartu the Autonomous Driving Lab (ADL) is currently one of the research groups working on teleoperation. They have been developing a self-driving vehicle since 2019. In its current state, their self-driving vehicle, a Lexus RX450h has self-driving capabilities but lacks a working and open enough teleoperation solution for allowing a human operator to control it on the streets from a distance.[1]

The topic of vehicle teleoperation has been more widely studied since the 1970s with usages in the air, on the ground and underwater [2]. The use of teleoperation has many applications, ranging from military operations[3] to industrial automation[4] and transportation[5]. Mostly in situations where a human operator can not be present, due to dangers or inconveniences [5]. Nowadays, one of the promising areas of application is in the development of self-driving vehicles, where teleoperation systems are needed for testing and deploying self-driving features. That's because a remote human operator is still expected to be ready to intervene even with fully autonomous self-driving vehicles, with the highest level of autonomy [6]. However, open-source software solutions for implementing such teleoperation on vehicles running ROS (Robot Operating System) or Autoware are not widely available. That is because providing a reliable low-latency camera stream is not an easy task. Also on the other hand, ROS is a very versatile platform, which makes building a one-for-all teleoperation system for various vehicles practically impossible.

This thesis addresses the lack of easily implementable teleoperation solutions for ROS vehicles as well as explores the existing possibilities of implementing teleoperation and documents developing a prototype teleoperation system for the ADL self-driving vehicle. The goal of the system is to be compatible with various vehicles running ROS and be easily usable in an actual self-driving vehicle at ADL. This work on teleoperation also provides a starting point for other students getting into teleoperation on other platforms as well.

1 Literature review

1.1 Teleoperation

This chapter sets the terminological grounds and gives a brief overview of the underlying technologies for teleoperation, mainly explaining the realm of teleoperation from the perspective of usage on-road vehicles.

Teleoperation is a technology that provides an operator remote access to a machine to control it. A defining feature of a teleoperation system is that it has to provide situational awareness to the operator in addition to allowing the operator to issue commands [2]. That usually means one or more camera feeds and other data that might help the operator localise the machine.

The operator's awareness of the remote situation plays a crucial role in the success of teleoperating a vehicle [5]. Therefore the design of the environment for the remote operator and the information presented is also important. Teleoperation systems need to provide operators with just the right amount of information at the right times to be most effective [7]. Other important technological considerations include vehicle and system architecture and ensuring connectivity between the operator and vehicle [7].

In the context of road vehicles, teleoperation plays an important role in the development of autonomous (self-driving) vehicles [5]. Road-vehicle teleoperation systems usually share the same general architecture as other types of teleoperation systems - the operator needs to have a low-latency communication channel with the vehicle for a video feed and other driving commands.

A simplified schematic of an example teleoperation system is depicted in Figure 1. It shows the possible main components on the vehicle side, which include antennae and modems for network connectivity, a computer for command and video processing as well as ROS for handling vehicle movement. On the other end of the communication channel is the operator's interface in the control centre, with support for all the necessary peripherals. This communication channel needs to be low-latency enough in all network conditions to allow controlling the vehicle in real-time.

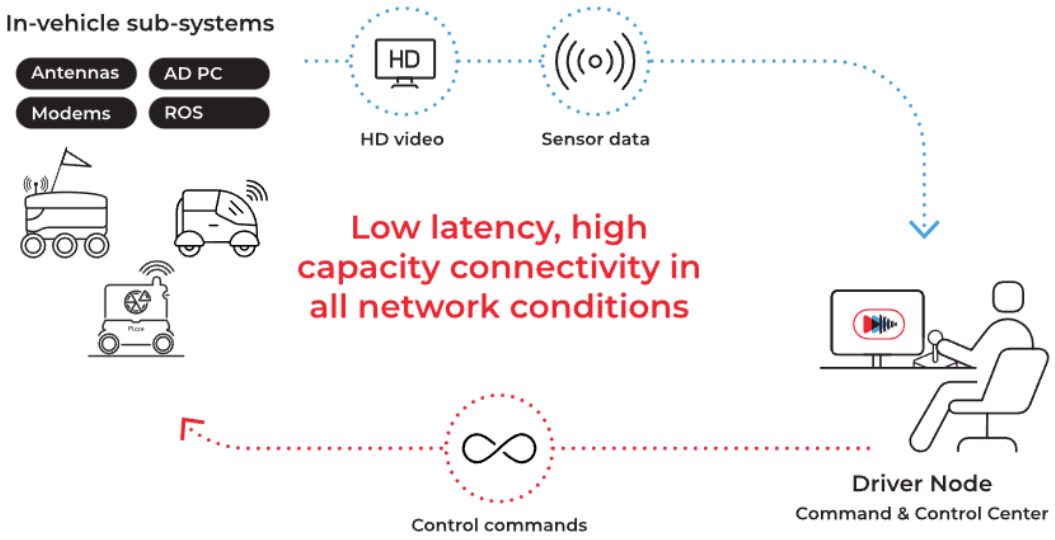

Figure 1. Architectural diagram of a vehicle teleoperation system. [8]

For maximal situational awareness, the operator's interface should be designed to resemble an actual vehicle [5]. That means controls - a steering wheel and pedals as well as buttons placed similarly to an actual car and monitors placed around the operator to see the vehicle's surroundings at a glance (Figure 2).

Figure 2. Example of a teleoperator's interface. [9]

Problems with vehicle teleoperation also mostly lay in the same categories as in other fields [5], for example, latency, image quality, and reactiveness. Creating an environment for the operator which feels actually like being in a vehicle is one of the main challenges ahead.

Lack of physical feedback and other perception issues have been reported as the biggest challenges along with communication quality with the vehicle [5]. Figure 3 depicts data from a study which analysed the challenges of remote driving by operators' feedback. That shows the major challenges users have reported are related to a lack of physical sensing and perception issues followed by communication quality. This means the two most important parts of a teleoperation system are having a good underlying solution for high-quality streaming and then focusing on the situational awareness part. Other functions like audio are not considered as important.

For this thesis, the results about situational awareness are not as important, but rather we need to focus on the communication quality block in Figure 3, which clearly shows that keeping the latency as low as possible needs to be a priority for our solution.

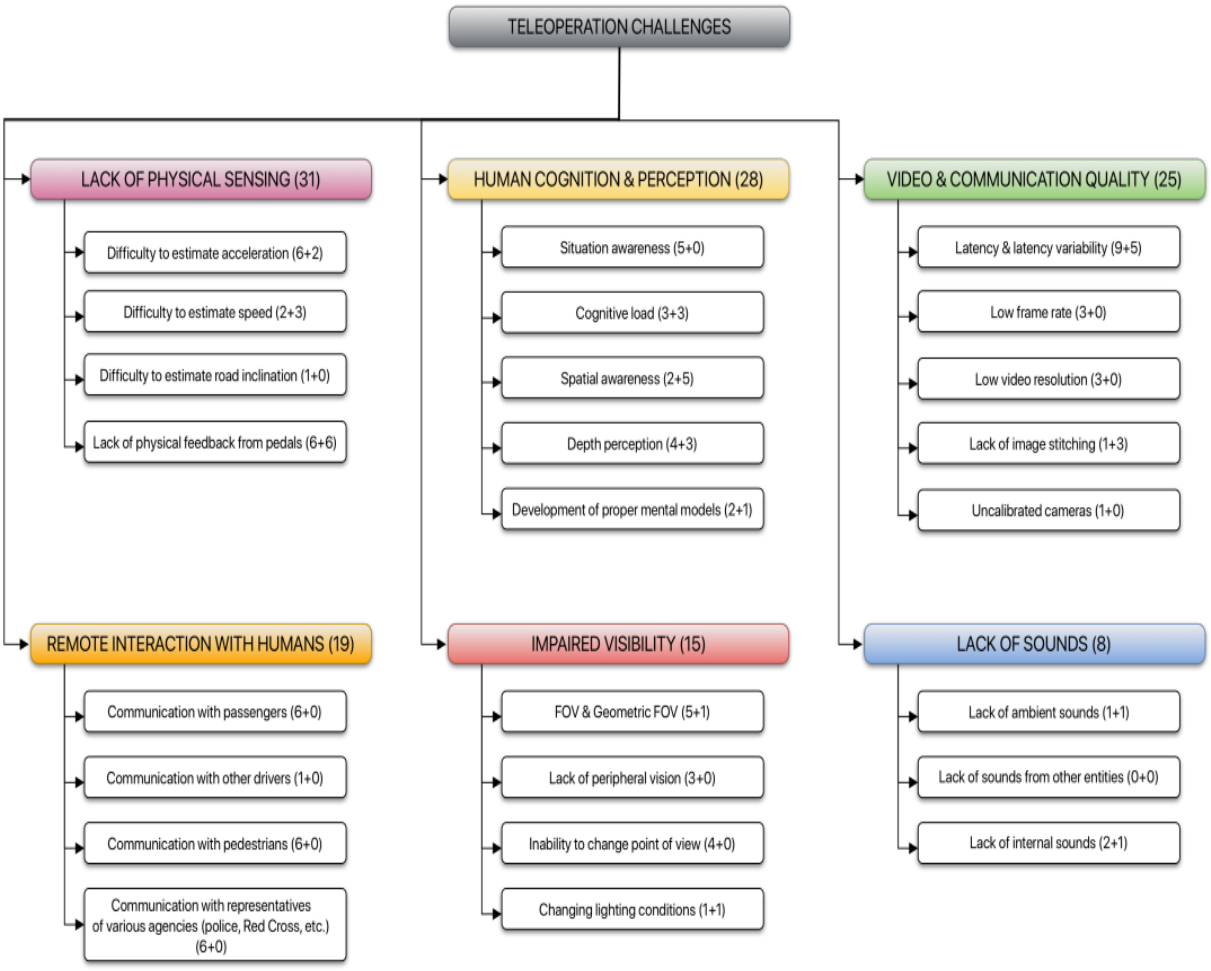

Figure 3. Categories of teleoperation challenges. [5]

1.2 Existing teleoperation solutions

There are already a few providers/developers of teleoperation systems in the world. Examples of those companies include driveU [10] and getUgo[11] but also Estonian companies ELMO[12] and Clevon[13], which are using teleoperation on their own vehicles.

DriveU offers teleoperation software solutions for various vehicles from delivery robots to heavy machinery, promising up to 4K streaming and API for transmitting various control data [10].

GetUgo is a Latvian company, which offers a complete teleoperation software and hardware platform [11]. Their implementation promises to work on various vehicles as well, ranging from autonomous vehicles to heavy machinery, with claimed latency under 100ms with encrypted communications and scalability built in [11].

1.2.1 Remote-controlled road vehicles in Estonia

There are multiple successfully implemented teleoperation systems in Estonia, that are already being used on the streets. One example is ELMO, which is offering car sharing and renting services with their vehicles[12]. Teleoperation is used for delivering rental cars to customers, exactly where they need them. Since teleoperation is only used for delivering the car and every car isn’t being teleoperated all the time, the teleoperators can switch between different vehicles. A typical scenario of teleoperating an Elmo vehicle can be seen in Figure 4. Their teleoperation systems have been approved by Estonia's Transport Administration and they are serving customers daily with teleoperated electric vehicles [12].

Figure 4. Elmo's teleoperation setup [14]

Another example of teleoperated vehicles in Estonia is Clevon, whose delivery vehicle CLEVON 1 uses teleoperation for parcel delivery as well as the development of autonomous driving systems [13]. Unlike Elmo they don't focus on teleoperating existing vehicles but have developed their own new vehicle platform (Figure 5), specially for parcel delivery. Creating a fully teleoperated vehicle means there is no need for a driver's seat and therefore the vehicle can be smaller and fit more cargo instead.

Figure 5. Clevon1 - Clevon's delivery vehicle platform [13]

1.3 Previous testing at ADL

Multiple teleoperation implementations have been already tested on the actual vehicle before this thesis. The author has also participated in testing two different solutions from different providers - GetUgo and RcSnail. Participating in that work gave the author actual experience of remotely driving a vehicle as well as allowed to see the strengths and weaknesses of these competing solutions.

1.3.1 GetUgo

First of the tested solutions was from GetUgo[11]. A successful streaming quality result of that solution can be seen in Figure 6. Most problems experienced during testing were related to excessive latency and streaming quality issues or problems with network connectivity. An example of low streaming quality can be seen in Figure 7. Ultimately these network issues were resolved.

Figure 6. Example of achieved streaming quality.

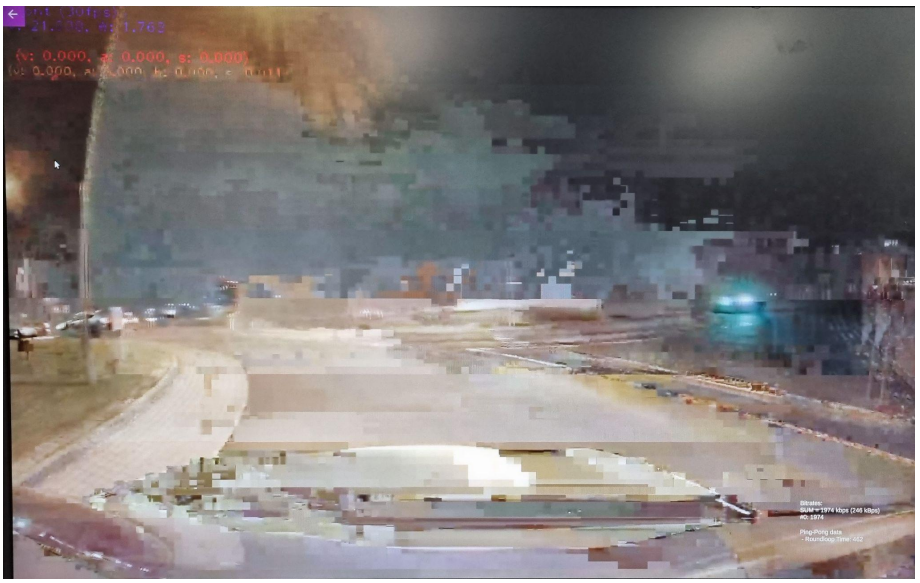

Figure 7. Example of worse streaming quality.

The main problem which guided ADL from not using that solution in the end was the use of GetUgo's servers, which caused excessive latency, connection problems and unexpected downtime and errors.

1.3.2 RcSnail

Testing the second solution revealed similar results for the image quality. The teleoperation software there was from RcSnail [15] and was initially meant for controlling RC cars but now modified to run on an actual road vehicle.

Deciding against this solution was due to the authentication between the car and the operator happening through RcSnail's server, which meant they effectively had control over access to the car remotely. In addition to that some of the source code was closed-source, which meant there was no way to verify the security aspects there. That was also the reason why open-source software was added as a requirement for the new solution and a reminder to think about the authentication process between the car and teleoperator for establishing a secure connection.

2 Requirements

The given requirements are based on the findings of previous teleoperation tests at ADL.

2.1 Objective

The objective of this thesis is to develop an open-source solution to teleoperate a vehicle over the network in an urban setting, e.g. the ADL self-driving vehicle. That includes validating other available software and/or modifying it in addition to writing new software. Finally, the proposed system has to meet the specified requirements to be able to solve the problem at hand.

2.2 Functional requirements

Functional requirements define how a system should behave or what functions it needs to have. These requirements are derived from the objective, by specifying the functions it needs to have to meet the objective. In our case the provided solution has to allow full teleoperation of the testing vehicle, which must include:

1. The operator can send driving commands to steer, accelerate and brake the vehicle
2. The operator can see at least one live video feed of the vehicle's surroundings
3. Dual-way audio connection between the operator and the safety driver.

2.3 Research questions

In addition the following research questions were posed, to find out the capabilities of the proposed solution:

1. Is it possible to send multiple video feeds via a single connection? If so, how many and with which quality?
2. What is the latency of the communication between the vehicle and the teleoperator?
3. How can additional data be sent via the same connection? For example, controlling various extra equipment on the vehicle or getting data about the vehicle’s current state

2.4 System requirements

System requirements describe the functions and features of the system as well as set describe the requirements of software and hardware components that need to be considered in development.

2.4.1 Software

On the operator's side:

- Client application running without any additional software installed, for example in a web browser. That is necessary for minimising the need to install software on various computers as well as ensuring compatibility with various operating systems.
- Support for additional input devices (steering wheel, pedals, game controllers). As this solution is meant to be eventually implemented on an actual vehicle, then using a steering wheel and pedals is almost mandatory. Firstly to create a comfortable environment and secondly for creating situational awareness for the operator, which importance was discussed before.

On the vehicle:

- Computer running Ubuntu operating system along with ROS
- Camera feed available as video device in the OS

2.4.2 Hardware

The software needs to be able to run on the chosen testing platform - Robotont[16], which hardware specifications are [16]:

- CPU: Intel Core i5 (7th Gen) 7260U (2 cores, up to 3.4 GHz)
- RAM: DDR4 2133 MHz 4 GB
- GPU: Intel Iris Plus Graphics 640
- Network: Intel Dual Band Wireless-AC 8265, IEEE 802.11a/b/g/n/ac

The ADL self-driving vehicles on-board computer's hardware specifications are [17]:

- CPU: Intel Xeon
- RAM: 32GB
- GPU: NVIDIA RTX2080Ti
- Network: 6x Gigabit Ethernet

The hardware specifications indicate that we have plenty of processing power available on the ADL self-driving vehicle, a lot more than on the Robotont. That means if our software is able to run on Robotont, it should not easily run on the ADL vehicle as well.

3 Implementation

Implementation of the teleoperation system, which takes into account the previously laid out requirements, started with choosing the software packages that could be used. Two main options there were RTCBot [18] and aiortc [19]. Although some initial tests were conducted with both, ultimately RTCBot was chosen for supporting video and audio streaming as well as sending control commands via a single connection, which promises to fully satisfy functional requirements 1 and 2 as well as partly solve requirement 3. Aiortc on the other hand isn't explicitly made for teleoperation, but general streaming applications, which would have necessitated developing a custom solution for driving commands and other signalling.

3.1 Architecture of the solution

Now considering the set requirements and capabilities of the chosen software, a prototype solution [20] was developed for further testing and development. After that, everything was adapted to work on the Robotont [16] and tested on that platform as well. The architecture of the solution is described in Figure 8.

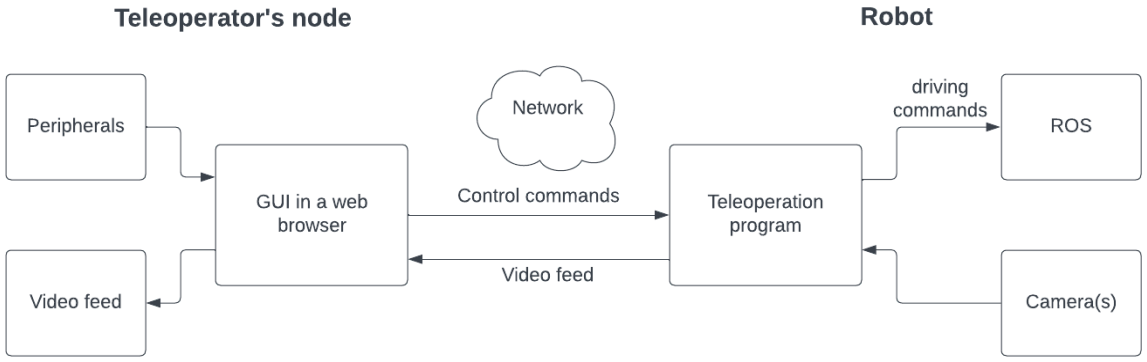

Figure 8. Block diagram of my implementation.

The centre point of the RTCBot implementation is the Python teleoperation program, which runs on the robot's onboard computer. That code handles serving the user interface to a web browser as well as all the communication with ROS and the onboard cameras.

Audio was also a requirement but considering that it is a less important part of the solution, the priority was on the actual video streaming and control - not on audio. As RTCBot had built-in support for one-way audio that was added to the camera stream and development of dual-way audio solution was left as future work.

For testing on the Robotont[16] platform the driving commands consist of x- and y-axis linear speed vectors moving around and a z-axis angular vector for turning, which is the standard command structure for the Robotont. For a vehicle that steers by turning its (front) wheels (e.g. the ADL self-driving car) a different set of commands needs to be implemented that uses the acceleration, steering and braking commands that are available there.

The user interface consists of a simple HTML page, which is shown in Figure 9 with a video element and some text to guide the user. All the commands are handled by capturing keyboard key presses. Serving this website is handled by aiohttp[21], which is integrated with RTCBot to serve the necessary javascript code for streaming and control commands to work on the operator's side. Those control commands are then converted to the robot's driving commands in the Python code and used to control the robot.

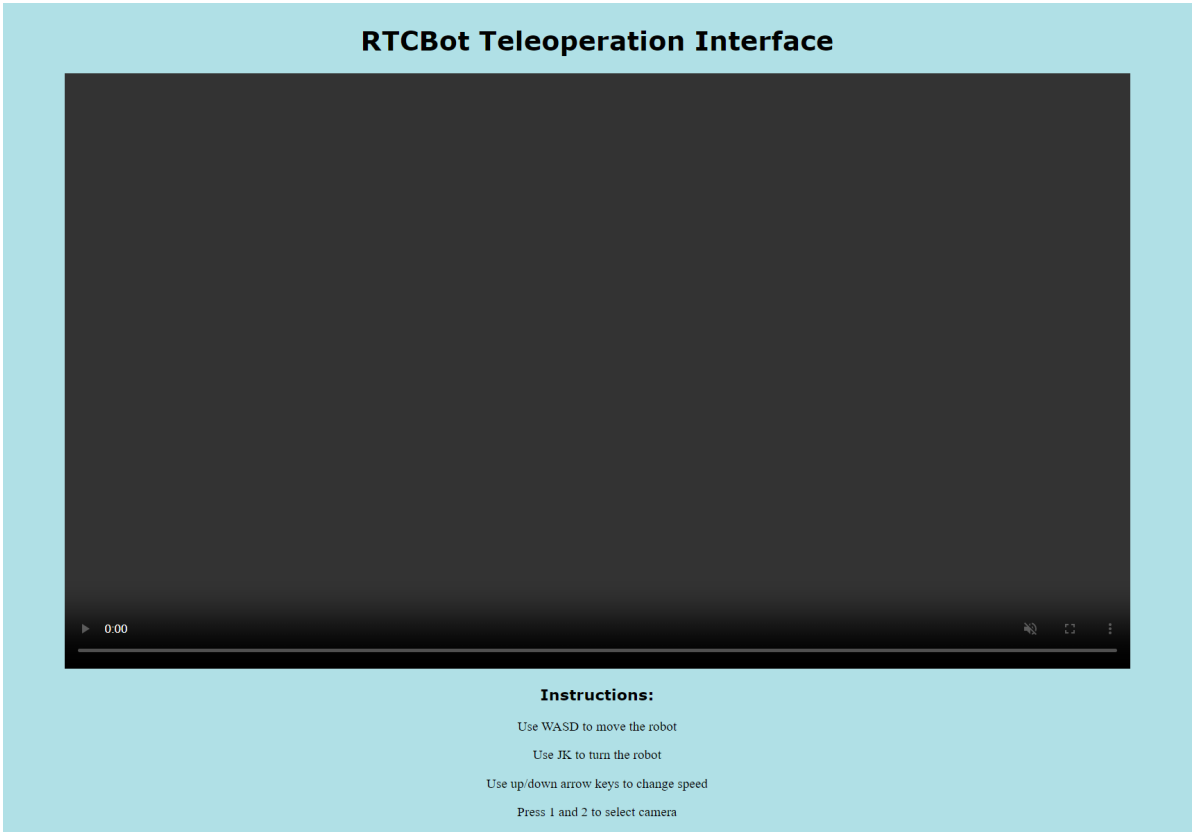

Figure 9. Screenshot of the teleoperator's interface.

The program logic on the robot's side is depicted in Figure 10. Getting the feed from cameras is solved by using RTCBot's inbuilt OpenCV [22] implementation called CVCamera. That gets processed by RTCBot and served to the teleoperator. Processed driving commands from the operator are passed on to ROS using rospy and a publisher on the necessary values, which runs in a separate thread in Python.

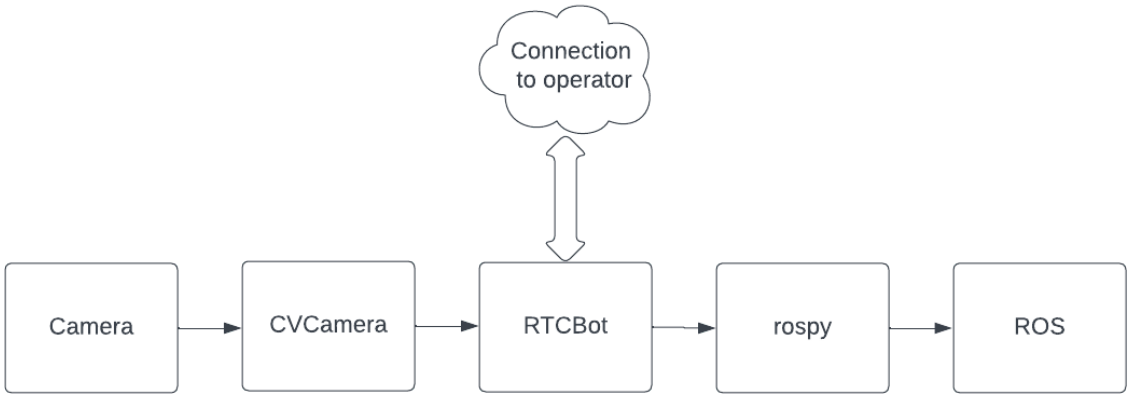

Figure 10. Program flow of the teleoperation software on the robot.

3.1.1 RTCBot

RTCBot is a Python library, purpose-built for building remote-control applications for various robots [18]. Internally, RTCBot uses WebRTC via aiortc for its real-time streaming [18]. In the documentation, we can find various tutorials with examples ranging from basic setup[23] to using different controllers [24] and supporting multiple simultaneous connections [25]. Different operating systems are also supported - RTCBot claims to run on Ubuntu, Raspbian, macOS and Windows, although on some platforms setup may be more complicated[26].

Being a small project, little active development goes on nowadays and besides the official documentation, few resources are available to provide support. According to the GitHub repository [27], RTCBot received its last minor update in August of 2022 with major updates ranging back to over 2 years ago, which turned out to cause some compatibility issues later on.

3.1.2 Networking

Since the teleoperator is expected to be at a remote location, creating a reliable and fast network connection between the operator and vehicle is also important. The architecture necessitates connecting two hosts from different LANs over the internet. For this thesis no specific requirements on the configuration of said LANs were posed and implementing a proprietary solution was not in the scope of work. Therefore existing software for handling the network connection was used.

Three options the author explored were OpenVPN [28], WireGuard [29] and ZeroTier [30]. OpenVPN is a general-purpose VPN software, through which multiple hosts can be added to the same virtual network and allow connecting that way [31]. Wireguard works on the same principle but promises to be more performant [29], which is also desirable for our application. The third option - ZeroTier was the easiest to configure, while not resulting in any noticeable drop in performance.

Eventually, both ZeroTier and Wireguard were tested for teleoperation but ZeroTier was used throughout testing due to easier setup on robot and operator sides. Although it is worth keeping in mind that ZeroTier, is not free for an unlimited number of hosts. Wireguard on the other hand is completely free to use and does not rely on service providers, which could be desirable features for an actual deployment.

3.1.3 Technical challenges

Implementing the previously described solution did not come without any issues. Firstly, constraining the OS version to Ubuntu 20.04 was needed, due to incompatibilities with RTCBot. Versions 18.04 and 22.04 were also tested but without success. That set the requirement of using Ubuntu 20.04 along with the compatible ROS version - ROS Noetic, on the robot’s onboard computer.

Secondly, during testing, problems with pyOpenSSL[32] versioning arose. RTCBot was found to require version 22.0 or lower. Version 22.0 was released in January of 2022, and by the time of writing, 4 newer versions are already available [33]. Since pyOpenSSL is dependent on the cryptography[34] library, a compatible version of that had to be installed as well. Using incompatible versions of either software RTCBot failed to establish a connection to the teleoperator and malfunction of other programs. Using pyOpenSSL v22.0 and cryptography v35.0 was found to be a working configuration.

Since one of the optional requirements was support for multiple simultaneous camera streams, implementing that was also explored. Due to a lack of support in the RTCBot, that feature could not be added. Although it was possible to get streams from multiple sources, there was no support in the RTCBot's javascript side to get those streams through the same connection. To at least partially meet that requirement, a feature to switch between different camera sources was added, but only one of them could be visible to the user at a time.

Overall the core problem of RTCBot was dependency issues, the official documentation lists 16 dependencies to get the software running and with version updates to each of them, problems are easy to come. A solution would be to containerize the software in some way, to rely less on different software incompatibilities, but that was out of the scope of this thesis.

4 Testing

The software was tested using the Robotont [16] robot platform (Figure 11). That platform was chosen for its availability to the author and for Ubuntu 20.04 as well as ROS running on its onboard computer.

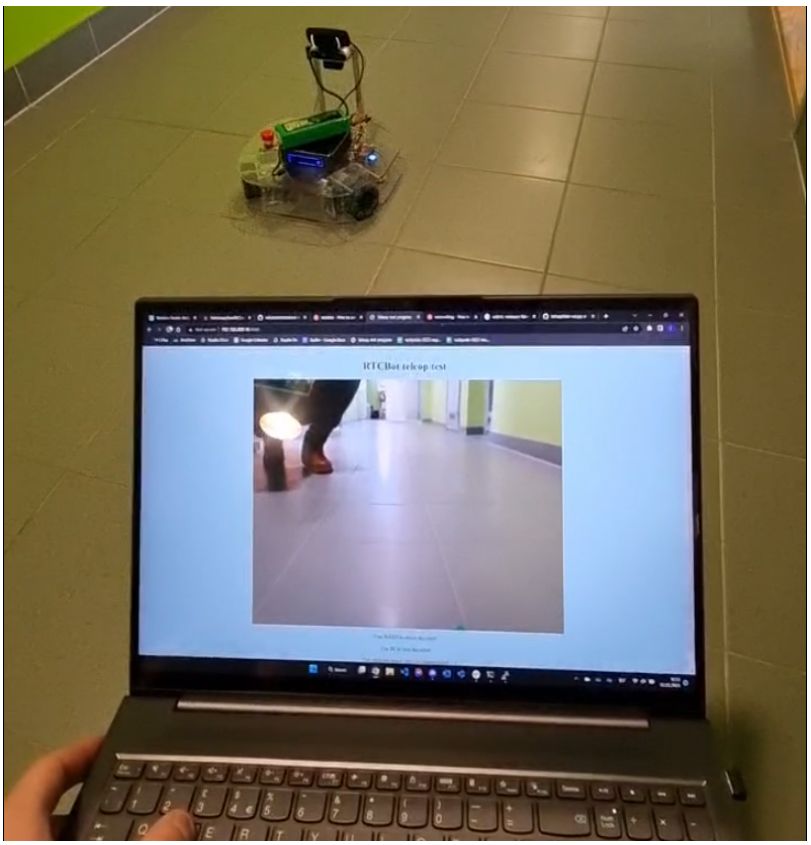

Figure 11. Teleoperation test with Robotont.

After solving installation and software compatibility problems, testing was successful on Robotont. The author was able to successfully navigate the Robotont using keyboard input to drive the robot according to the camera feed, with latencies comfortably allowing teleoperation over a Wi-Fi network.

4.1 Migrating to Autoware

Since the goal is to ultimately run the developed software on the ADL self-driving vehicle, there are things that need to be modified in the software to ensure compatibility. List of changes includes:

- Changing the driving command structure - Robotont uses a system where it takes one or more velocity vectors and moves accordingly. Autoware needs inputs for acceleration, brake and wheel angle separately. For Robotont, there was a single command (cmd_vel) that handled the movement in every direction and there was no extra command for braking. In Autoware 3 different commands (accel_cmd, steer_cmd and brake_cmd) need to be used to handle accelerating, steering and braking respectively.
- Ensure compatibility with OS and other software versioning - since the compatible Ubuntu version was found to be 20.04, this version needs to be running on the vehicle's onboard computer.
- Implementing any feedback data from the vehicle

Other changes can arise during testing, but the mentioned command structure seems to be the main difference between the platforms.

5 Conclusions

The objective of this thesis was fulfilled - an open-source solution for teleoperating a vehicle over the network was developed and tested on a small robot. Functional requirements 1 and 2 were completely met. Functional requirement 3 was only partially met since the audio is working only one way - the remote operator can hear the car but not vice versa.

Due to previously discussed issues with RTCBot and multiple video streams, the author did not manage to successfully stream more than a single camera feed at a time, which means at this point the maximum number of simultaneous video streams is 1. However running multiple instances might allow more, depending on available network bandwidth.

Since RTCBot and its dependencies proved to be incompatible with some OS and software versions. This issue does not affect the ability to run software on the self-driving vehicle but might do so in the future.

To deploy this solution on an actual road vehicle, such as the self-driving car at ADL, the program needs to be modified to support the command structure of that vehicle as described before as well as thoroughly tested to ensure safety during operation.

Regarding possible deprecation issues with RTCBot, it also needs to be evaluated if it is worth developing this existing base or if alternatives should be explored at this point. A more robust solution using WebRTC directly for streaming and a new implementation for sending control data would result in a more suitable and customizable solution overall if we already need to start heavily modifying RTCBot to achieve all the necessary functionality.

However, this base definitely provides a valuable starting point for other students starting with teleoperation and RTCBot remains a relatively easy starting point to get a basic demo running on a robot.

Bibliography

[1] "About us," *Discover - Autonomous Driving Lab*. https://adl.cs.ut.ee/discover/about (accessed Mar. 21, 2023).

[2] T. Fong and C. Thorpe, "Vehicle Teleoperation Interfaces," *Autonomous Robots*, vol. 11, no. 1, pp. 9–18, 2001, doi: 10.1023/A:1011295826834.

[3] "Driving the Future," *www.army.mil*. https://www.army.mil/article/220565/driving_the_future (accessed Apr. 04, 2023).

[4] D. Aschenbrenner, M. Fritscher, F. Sittner, M. Krauß, and K. Schilling, "Teleoperation of an Industrial Robot in an Active Production Line," *IFAC-PapersOnLine*, vol. 48, no. 10, pp. 159–164, 2015, doi: 10.1016/j.ifacol.2015.08.125.

[5] F. Tener and J. Lanir, "Driving from a Distance: Challenges and Guidelines for Autonomous Vehicle Teleoperation Interfaces," in *CHI Conference on Human Factors in Computing Systems*, New Orleans LA USA: ACM, Apr. 2022, pp. 1–13. doi: 10.1145/3491102.3501827.

[6] K. Kuru, "Conceptualisation of Human-on-the-Loop Haptic Teleoperation With Fully Autonomous Self-Driving Vehicles in the Urban Environment," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 448–469, 2021, doi: 10.1109/OJITS.2021.3132725.

[7] A. Podhurst, "The complete guide to autonomous vehicle teleoperation," *DriveU*, Apr. 06, 2021. https://driveu.auto/blog/the-complete-guide-to-av-teleoperation/ (accessed Mar. 28, 2023).

[8] "Robot teleoperation connectivity platform - DriveU.auto," *DriveU*. https://driveu.auto/product/driveu-100/ (accessed Apr. 17, 2023).

[9] Editorial, "Vehicle teleoperation - Three basic models and their limitations," *RoboticsBiz*, Feb. 02, 2021. https://roboticsbiz.com/vehicle-teleoperation-three-basic-models-and-their-limitations/ (accessed Apr. 17, 2023).

[10] "DriveU.auto - Superior connectivity platform for autonomous vehicle teleoperation," *DriveU*. https://driveu.auto/ (accessed Apr. 17, 2023).

[11] "getUgo." https://www.getugo.com/ (accessed Apr. 17, 2023).

[12] "Esileht," *Elmo*. https://www.elmoremote.com/et/ (accessed Mar. 28, 2023).

[13] "CLEVON 1," *Clevon*. https://clevon.com/et/clevon1/ (accessed Apr. 04, 2023).

[14] K. K. | ERR, "Elmo rendiautod hakkavad kliendini sõitma kaugjuhtimise teel," *ERR*, Nov. 25, 2022. https://www.err.ee/1608799978/elmo-rendiautod-hakkavad-kliendini-soitma-kaugjuhtimise-teel (accessed Apr. 28, 2023).

[15] "RCSnail." https://rcsnail.ee/ (accessed May 01, 2023).

[16] "Overview — Robotont 0.0.1 documentation." https://robotont.github.io/html/files/overview.html (accessed Mar. 21, 2023).

[17] "Vehicle," *Autonomous Driving Lab*. https://adl.cs.ut.ee/lab/vehicle (accessed May 02, 2023).

[18] "Welcome to RTCBot's documentation! — RTCBot 0.2.4 documentation." https://rtcbot.readthedocs.io/en/latest/ (accessed Apr. 13, 2023).

[19] "aiortc — aiortc documentation." https://aiortc.readthedocs.io/en/latest/ (accessed Apr. 13, 2023).

[20] erkiveevali, "erkiveevali/teleop." Jan. 17, 2023. Accessed: May 13, 2023. [Online]. Available: https://github.com/erkiveevali/teleop

[21] "Welcome to AIOHTTP — aiohttp 3.8.4 documentation." https://docs.aiohttp.org/en/stable/ (accessed Apr. 23, 2023).

[22] “Home,” *OpenCV*. https://opencv.org/ (accessed Apr. 23, 2023).
[23] “RTCBot Basics — RTCBot 0.2.4 documentation.” https://rtcbot.readthedocs.io/en/latest/examples/basics/README.html (accessed Apr. 28, 2023).
[24] “Keyboard & Xbox Controller — RTCBot 0.2.4 documentation.” https://rtcbot.readthedocs.io/en/latest/examples/remotecontrol/README.html (accessed Apr. 28, 2023).
[25] “Multiple Connections & Reconnecting — RTCBot 0.2.4 documentation.” https://rtcbot.readthedocs.io/en/latest/examples/multiconnect/README.html (accessed Apr. 28, 2023).
[26] “Installing RTCBot — RTCBot 0.2.4 documentation.” https://rtcbot.readthedocs.io/en/latest/installing.html (accessed Apr. 28, 2023).
[27] “rtcbot/index.rst at master · dkumor/rtcbot,” *GitHub*. https://github.com/dkumor/rtcbot (accessed Apr. 14, 2023).
[28] “Business VPN | Next-Gen VPN,” *OpenVPN*. https://openvpn.net/ (accessed May 01, 2023).
[29] J. A. Donenfeld, “WireGuard: fast, modern, secure VPN tunnel.” https://www.wireguard.com/ (accessed May 01, 2023).
[30] “ZeroTier | Global Area Networking.” https://www.zerotier.com/ (accessed May 01, 2023).
[31] “What Is A VPN? | VPN Definition,” *OpenVPN*. https://openvpn.net/what-is-a-vpn/ (accessed May 01, 2023).
[32] “Welcome to pyOpenSSL’s documentation! — pyOpenSSL 23.2.0.dev documentation.” https://www.pyopenssl.org/en/latest/ (accessed Apr. 23, 2023).
[33] “Changelog — pyOpenSSL 23.2.0.dev documentation.” https://www.pyopenssl.org/en/latest/changelog.html (accessed Apr. 23, 2023).
[34] “Welcome to pyca/cryptography — Cryptography 41.0.0.dev1 documentation.” https://cryptography.io/en/latest/ (accessed Apr. 23, 2023).

Acknowledgements

I would like to thank my supervisors Karl Kruusamäe and Ulrich Norbisrath for guiding me throughout this thesis as well as Tambet Matiisen and the team at ADL for helping along the way.

Appendices

Appendix 1. Source code

```
#teleop.py


#!/usr/bin/env python3
from aiohttp import web
from rtcbot import RTCConnection, CVCamera, Microphone, getRTCBotJS


import rospy
from geometry_msgs.msg import Twist

import threading

#rospy init
rospy.init_node("velocity_publisher")
velocity_pub = rospy.Publisher("cmd_vel", Twist, queue_size=0)
rospy.sleep(2)

#Route table definition
routes = web.RouteTableDef()

# Create cameras
cam = CVCamera(cameranumber=0)
#cam2 = CVCamera(cameranumber=2)
mic = Microphone()

# 1 global connection
conn = RTCConnection()
# Initial subscriptions
conn.video.putSubscription(cam)
conn.audio.putSubscription(mic)

keystates = {"w": False, "a": False, "s": False, "d": False, "j": False,
"k": False}
robot_speed=0.1

#Publisher for speed commands
def publisher():
    global keystates
    global robot_speed
    loop_rate = rospy.Rate(10)
    robot_vel = Twist()
```

```
    #Slow infinite loop to set robot_vel
    while True:
        #Forward/back
        if keystates["w"]:
            robot_vel.linear.x=robot_speed
        elif keystates["s"]:
            robot_vel.linear.x=-robot_speed
        else:
            robot_vel.linear.x=0
        #Left/right
        if keystates["a"]:
            robot_vel.linear.y=robot_speed
        elif keystates["d"]:
            robot_vel.linear.y=-robot_speed
        else:
            robot_vel.linear.y=0
        #Turn
        if keystates["j"]:
            robot_vel.angular.z=robot_speed
        elif keystates["k"]:
            robot_vel.angular.z=-robot_speed
        else:
            robot_vel.angular.z=0

        velocity_pub.publish(robot_vel)
        loop_rate.sleep()

@conn.subscribe
def onMessage(m):
    global keystates
    global robot_speed
    #Reading keycodes
    if m["keyCode"] == 87:  # W
        keystates["w"] = m["type"] == "keydown"
    elif m["keyCode"] == 83:  # S
        keystates["s"] = m["type"] == "keydown"
    elif m["keyCode"] == 65:  # A
        keystates["a"] = m["type"] == "keydown"
    elif m["keyCode"] == 68:  # D
        keystates["d"] = m["type"] == "keydown"
    elif m["keyCode"] == 74:  # J
        keystates["j"] = m["type"] == "keydown"
    elif m["keyCode"] == 75:  # K
        keystates["k"] = m["type"] == "keydown"
```

```
    #Change speed
    elif m["keyCode"] == 38 and m["type"] == "keydown":  # Up arrow
        robot_speed+=0.1
    elif m["keyCode"] == 40 and m["type"] == "keydown" and
robot_speed>0.1:  # Down arrow
        robot_speed-=0.1

    #Switch cameras
    #elif m["keyCode"] == 49 and m["type"] == "keydown":  # 1
        #conn.video.putSubscription(cam)
    #elif m["keyCode"] == 50 and m["type"] == "keydown":  # 2
        #conn.video.putSubscription(cam2)


# Serve the RTCBot javascript library at /rtcbot.js
@routes.get("/rtcbot.js")
async def rtcbotjs(request):
    return web.Response(content_type="application/javascript",
text=getRTCBotJS())

# This sets up the connection
@routes.post("/connect")
async def connect(request):
    clientOffer = await request.json()
    serverResponse = await conn.getLocalDescription(clientOffer)
    return web.json_response(serverResponse)

@routes.get("/")
async def index(request):
    return web.Response(
        content_type="text/html",
        text=open("index.html", "r").read()
    )

async def cleanup(app=None):
    await conn.close()

#separate thread for publisher function
pub_thread = threading.Thread(target=publisher)
pub_thread.start()

#webapp
app = web.Application()
app.add_routes(routes)
app.on_shutdown.append(cleanup)
web.run_app(app)
```

```
<!--index.html-->
<html>
        <head>
            <title>Teleop</title>
            <script src="/rtcbot.js"></script>
        </head>
        <body style="background-color: powderblue; padding-top: 30px;
text-align: center;">
            <h1 style="font-family:verdana;">RTCBot Teleoperation
Interface</h1>
            <video width= "1280" height="720" autoplay muted playsinline
controls></video> <audio autoplay></audio>
            <h3 style="font-family:verdana;">Instructions:</h3>
            <p>Use WASD to move the robot</p>
            <p>Use JK to turn the robot</p>
            <p>Use up/down arrow keys to change speed</p>
            <p>Press 1 and 2 to select camera</p>
            <script>
                var conn = new rtcbot.RTCConnection();
                conn.video.subscribe(function(stream) {
                    document.querySelector("video").srcObject = stream;
                    });
                    conn.audio.subscribe(function (stream) {
                    document.querySelector("audio").srcObject = stream;
                    });

                var kb = new rtcbot.Keyboard();
                async function connect() {
                    let offer = await conn.getLocalDescription();

                    // POST the information to /connect
                    let response = await fetch("/connect", {
                        method: "POST",
                        cache: "no-cache",
                        body: JSON.stringify(offer)
                    });
                    await conn.setRemoteDescription(await
response.json());

                    kb.subscribe(conn.put_nowait);
                    console.log("Ready!");
                    }
                connect();</script>
        </body>
    </html>
```

Licence

I, Erki Veeväli

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

   Development of a continuous teleoperation system for urban road vehicle

   supervised by Karl Kruusamäe and Ulrich Norbisrath

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.

Erki Veeväli
13.05.2023