



TUM SCHOOL OF COMPUTATION,  
INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Robotics, Cognition, Intelligence

# **Creation and Evaluation of a Human-Machine Interface for Teleoperating a Rover over a Portable Device**

**Michelle Tina Bettendorf**





TUM SCHOOL OF COMPUTATION,  
INFORMATION AND TECHNOLOGY

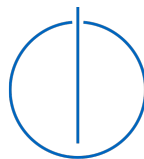
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Robotics, Cognition, Intelligence

# **Creation and Evaluation of a Human-Machine Interface for Teleoperating a Rover over a Portable Device**

## **Entwicklung und Bewertung einer Mensch-Maschine-Schnittstelle für die Teleoperation eines Rovers über ein tragbares Gerät**

Author:	Michelle Tina Bettendorf
Supervisor:	Prof. Dr.-Ing Alin Albu-Schäffer
Advisor:	Dr.-Ing Armin Wedler
Submission Date:	11.09.2023



I confirm that this master's thesis in informatics: robotics, cognition, intelligence is my own work and I have documented all sources and material used.

A handwritten signature in black ink, reading "Bettendorf. Mi".

Munich, 11.09.2023

Michelle Tina Bettendorf

# Abstract

Teleoperation for mobile robots is challenging, this thesis aims to investigate the creation of a Human-Machine Interface for a portable device, which is a smartphone extended with a six degree of freedom input control device over the USB port. This was done by analyzing three different options for transmitting the camera and mapping stream over a mobile network and evaluating them in terms of delay and robustness. Among the investigated options are Virtual Network Computing (VNC), a software for remote controlling and visualization of the rover, ROS-mobile, a Robot Operating System (ROS) App which displays ROS topics received over User Datagram Protocol (UDP), and GStreamer, an open source framework designed to compress and send media streams. Furthermore, research on how to implement the control device to control the rover and what functionalities to implement in the Graphical User Interface (GUI) was done. As a result of two user case studies, GStreamer has proven to be a robust and efficient camera streaming option that can be integrated into an application that also includes the controlling of the robot. This results in a robust and intuitive Human-Machine Interface for controlling the rover over a portable device.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 State of the art of the rover</b>	<b>2</b>
2.1 Hardware . . . . .	2
2.1.1 Rover . . . . .	2
2.1.2 Smartphone . . . . .	2
2.1.3 Spacemouse . . . . .	2
2.2 Software . . . . .	3
2.2.1 Overview of the architecture . . . . .	4
2.2.2 LN manager . . . . .	4
2.2.3 ROS . . . . .	4
2.2.4 Matlab and Simulink . . . . .	5
2.2.5 Mobile network . . . . .	6
<b>3 Related work</b>	<b>8</b>
<b>4 Concept</b>	<b>10</b>
4.1 General goal . . . . .	10
4.2 Connection spacemouse to robot . . . . .	11
4.3 VNC Viewer . . . . .	11
4.4 ROS App . . . . .	12
4.5 GStreamer with a Java App . . . . .	13
<b>5 Setup and Implementation</b>	<b>15</b>
5.1 Control App implementation . . . . .	15
5.2 VNC setup . . . . .	17
5.3 ROS-mobile setup . . . . .	19
5.4 GStreamer implementation . . . . .	19
<b>6 Testing</b>	<b>23</b>
6.1 GUI testing . . . . .	23
6.1.1 VNC GUI testing . . . . .	23
6.1.2 ROS-mobile GUI testing . . . . .	23
6.1.3 GStreamer GUI testing . . . . .	25

6.2	Camera testing . . . . .	25
6.2.1	Camera testing of all options in the lab setup . . . . .	25
6.2.2	Camera stream testing of all options in the mobile setup in the lab . . .	25
6.3	Control App testing in the lab . . . . .	26
6.4	Human-Machine Interface testing with the camera stream on Vulcano . . . . .	27
6.5	Map testing . . . . .	30
<b>7</b>	<b>User Case Studies</b>	<b>35</b>
7.1	Camera User Case Study on Vulcano . . . . .	35
7.2	Human Machine Interface User Case Study in the German Aerospace Center (DLR) lab . . . . .	37
<b>8</b>	<b>Evaluation</b>	<b>42</b>
<b>9</b>	<b>Conclusion</b>	<b>46</b>
	<b>List of Figures</b>	<b>47</b>
	<b>Acronyms</b>	<b>49</b>
	<b>Bibliography</b>	<b>50</b>

# 1 Introduction

Mobile robots have two main driving options: autonomous and teleoperation. Both play an important role in mobile robotics and especially in space robotics as many exploration rovers, such as the Mars Exploration rover [1], utilized a combination of both for their successful operations. Even though there might be the ideal image of having a fully autonomous rover, scenarios can occur, in which humans have to take over. For this, a teleoperation with a suitable Human-Machine Interface and controller comes into place. To achieve this, a suitable device with an internal or external controller and a possible GUI with functionalities for the specific use case have to be researched.

The goal of this thesis is to design, implement and evaluate a Human-Machine Interface for a portable device to teleoperate a rover. The focus will be on designing a suitable GUI, researching and implementing methods on how to transfer the control data, camera stream and a danger map, and how to implement a specific controller, called spacemouse, in order to teleoperate the rover. For this purpose, a smartphone connected to a spacemouse controller was utilized. In addition three different approaches for the transmission of the camera stream and map were implemented and evaluated. The evaluation was based on testing in different terrains like sand and stones, and also on two user case studies. Among the investigated options were VNC, a software for remote controlling and visualization of the rover, ROS-mobile, a ROS App which displays ROS topics received over UDP, and GStreamer, an open source framework designed for compressing and sending media streams.

This thesis is part of a student project on the rover called Luna Rover Mini (LRM). The project intends to have a robot that serves as a research and experimentation platform for students and scientists. Since the hardware is much less expensive than the space robots, more experimental opportunities can be achieved. The design of the LRM is based on the Exo Mars Rover, whose goal is to explore and collect data on mars [2]. The software is based on the Lightweight Rover Unit (LRU), whose goal is to explore unknown and difficult to access terrain [3].

## 2 State of the art of the rover

In this chapter, the state of the art of the rover before starting this thesis will be described. It is structured in hardware and software.

### 2.1 Hardware

The hardware consists of two rovers and a portable device, which is a smartphone extended with an input control device called spacemouse. These will be explained in the following sections.

#### 2.1.1 Rover

The design of the rovers is based on the Exo Mars rover [2] and consists of a pan/tilt camera, the Intel Realsense that provides an RGB-Depth image with an Inertial Measurement Unit (IMU), an antenna for Bluetooth connection, a mainboard and 6 wheels. Two wheels are connected through one bogie board. It rotates around the z axis (yaw). This allows the robot to move directly on the x and y axis, to drive curves, to rotate while staying in the same place and to tilt all wheels at the same angle. There are two rovers, LRM1 and LRM2. LRM2 is the improved version of LRM1 but consists of the same software. In figure 2.1 is on the left LRM1 and on the right Luna Rover Mini 2 (LRM2) is shown.

#### 2.1.2 Smartphone

For the smartphone, a Google Pixel 2XL is utilized. This is an Android based smartphone. The utilized Android version is Android 11. The connection of the smartphone to the rover is explained in section 2.2.5. A picture of the smartphone is shown in figure 2.3.

#### 2.1.3 Spacemouse

The spacemouse is a development of the DLR and the Spin-Off 3D Input GmbH. It is an external controller for the smartphone and is connected via the USB C Port. With this controller, six degrees of freedom can be reported to the smartphone. The spacemouse is shown in figure 2.2. The smartphone combined with the spacemouse, which will be the utilized portable device for this thesis, is shown in figure 2.4.





Figure 2.1: LRM1(left) and LRM2(right) on the volcano of the Aeolian Island Vulcano



Figure 2.2: Picture of the spacemouse



Figure 2.3: Picture of the smartphone



Figure 2.4: Picture of the controller

## 2.2 Software

The software consists of three different main parts, the Links and Nodes (LN) manager, a ROS part and a Matlab and Simulink part.

### 2.2.1 Overview of the architecture

In the following, an overview of the software structure of the project before starting this thesis will be given. Each part will be covered in a separate section. The user interacts with the LN-manager, which then interacts with the high level software parts (ROS), the low level software parts (Matlab/Simulink) and the robot.

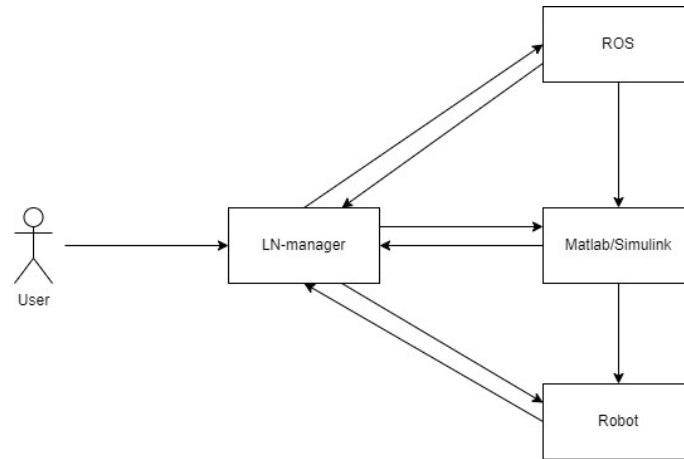


Figure 2.5: Overview of the architecture

### 2.2.2 LN manager

The LN manager is a tool of the DLR that helps visualizing, starting and stopping the processes of the rover. These processes are for different parts of the projects used. Some are dependent on others whereas others are independent. In the GUI multiple tabs are available. These are process, clients, topics, services, parameters and log. In processes are all known processes listed and can be started and stopped. In clients, topics, services and parameters are the currently running clients, topics, services and parameters of the running processes shown. The log tag is utilized for debugging as it shows all the working steps of the processes. A screenshot of the LN-manager is shown in figure 2.6.

### 2.2.3 ROS

The software is mostly realized with the help of ROS. ROS is an open source meta operation system that provides libraries and tools to create robotic applications. The architecture is composed of nodes, which publish or subscribe to a topic [4]. In ROS the autonomous driving part is realized, this consists of mapping, planning and execution of the path, which are all separate nodes. As only the finished map of the autonomous part is required for this thesis, the autonomous part will not be further explained.

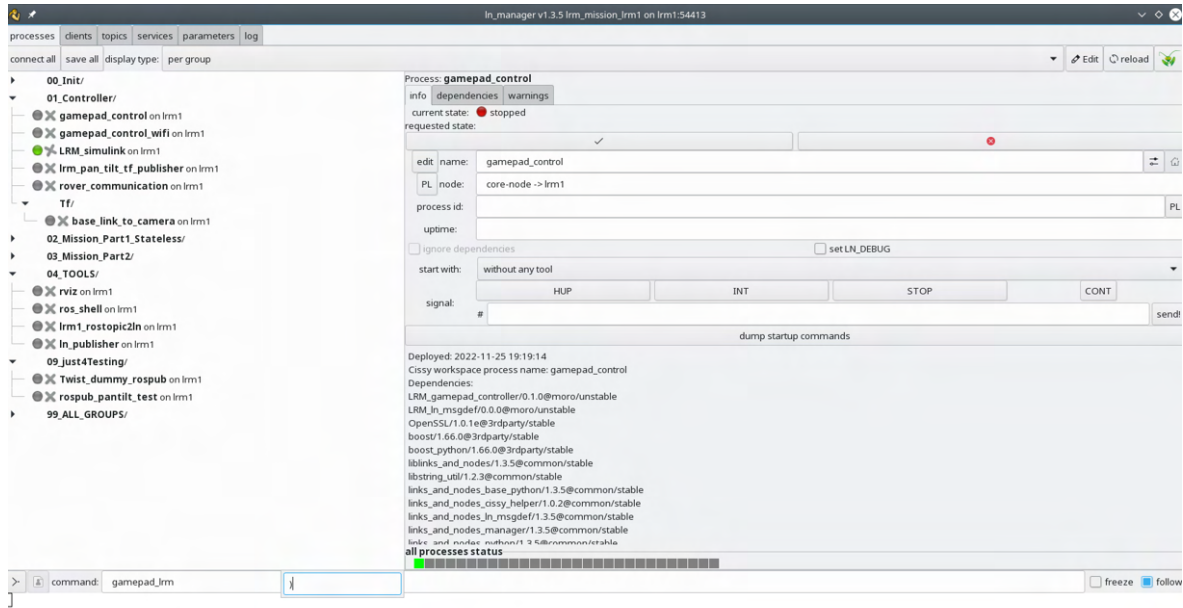


Figure 2.6: LN manger

### 2.2.4 Matlab and Simulink

The Simulink model consists of the controlling part of the rover and provides the data for the rover communication on how the rover should move next. It consists of the LN Controller (Controller In Prio1, Controller Prio2, Controller In Prio3), a Controller switch, Pantilt, Controller and Wheels. The Simulink model is illustrated in figure 2.7.

In the model, there are 3 controllers (Controller 1, Controller 2, Controller 3) implemented. These 3 controllers represent different input devices (e.g. the gamepad). Until now controller 1 is the gamepad controller and controller 2 is the autonomous navigation. The main idea is that the input values from the control devices are directly transferred to Simulink. Simulink has then a controller, that adapts the input values, so that they are transferable to the rover, e.g. how to transfer the joystick movement to the actual movement of the rover (how much is the rover moving) and the implementation of the different driving modes.

There are three different driving modes implemented: Ackermann, Rotation and Crab mode. Ackermann is the driving mode that also cars are using. In this mode the vehicle is driving around one center point. The center point is changed with every joint movement. The rotation mode is used for rotating the rover while staying in the same place. In the crab mode all the wheels are turning in the same angle. With this mode, the rover is able to move sideways. Crab and Rotation mode help the robot to reach positions faster and without the restrictions of the Ackermann mode. The different driving modes are shown in figure 2.8.

The output of the Simulink model is transferred to the rover communication over ROS topics

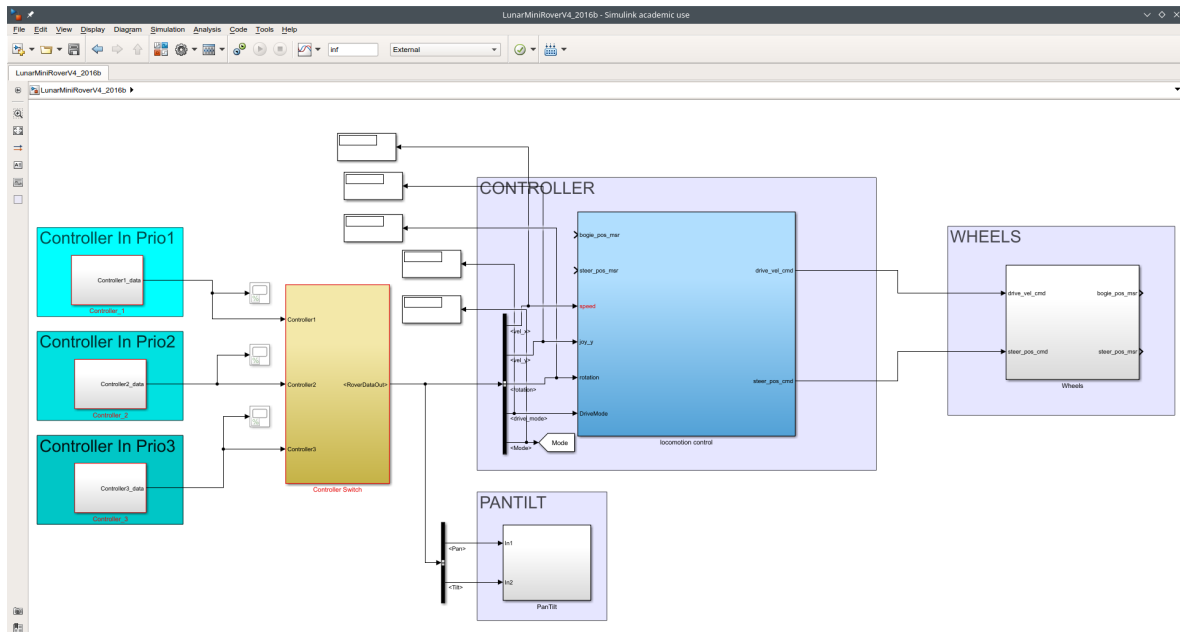


Figure 2.7: Simulink model

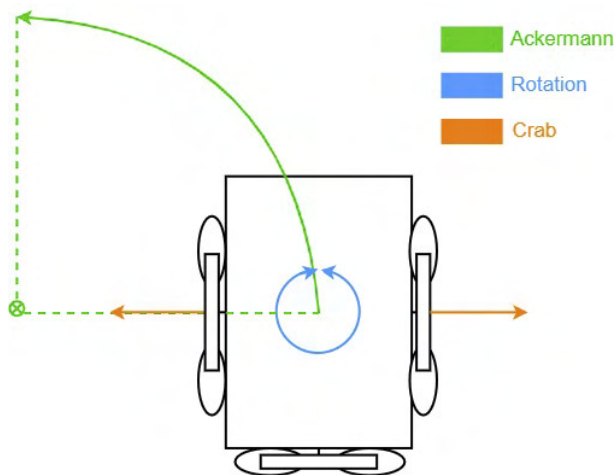


Figure 2.8: The 3 different driving modes: Ackermann, Rotation, Crab

(Data\_IN). Rover communication is then commanding each separate wheel in order to execute the desired movement.

### 2.2.5 Mobile network

In order to control the robot, a network connection is needed. There are two options for creating that, either by a network cable or through WiFi. As DLR has some security for network connections both options are not straightforwardly realized. For the network cable

connection, the robot has a direct connection to a switch/proxy, which is called Robotics and Mechatronics Center (RMC)-Mobilenetz switch/proxy. This switch has a connection to the RMC network, which is an intern network within the RMC institute and to the internet. It is not a direct internet connection, there is a firewall in between for safety reasons. The RMC network is to the RMC-IT-Laptop and the RMC-IT-Desktop connected. These two have a connection to the rover. This option requires to have a cable connection to the robot, which is not ideal for a mobile robot as it hinders movement. To prevent this a connection over WiFi has to be created. For this, there is the bulletin AC RMC-AP0080 on the rover to create a connection to the WiFi. To have the required safety aspects, the rover is not directly connected to the DLR intern WiFi. There is a mobile network WiFi for this purpose which serves as a bridge from the mobile devices to the robot. The mobile devices are not directly connected to this network and also not to the intern DLR WiFi, they are connected to the RMC-AP0057 bulletin, which is the bridge to the rocket AC AP-0105. This rocket AC is then connected over the mobile network to the rover bulletin (RMC-AP0080). The available mobile devices are a laptop and a smartphone. Both have a static IP address to connect to the bulletin. Furthermore, there is a Bluetooth connection between a gamepad controller and the robot. As this controller is not used in the thesis it won't be further explained. The combination of the RMC-AP0057 bulletin and the rocket AC AP-0105 will be referred to as antenna. The architecture of the mobile network is illustrated in figure 2.9.

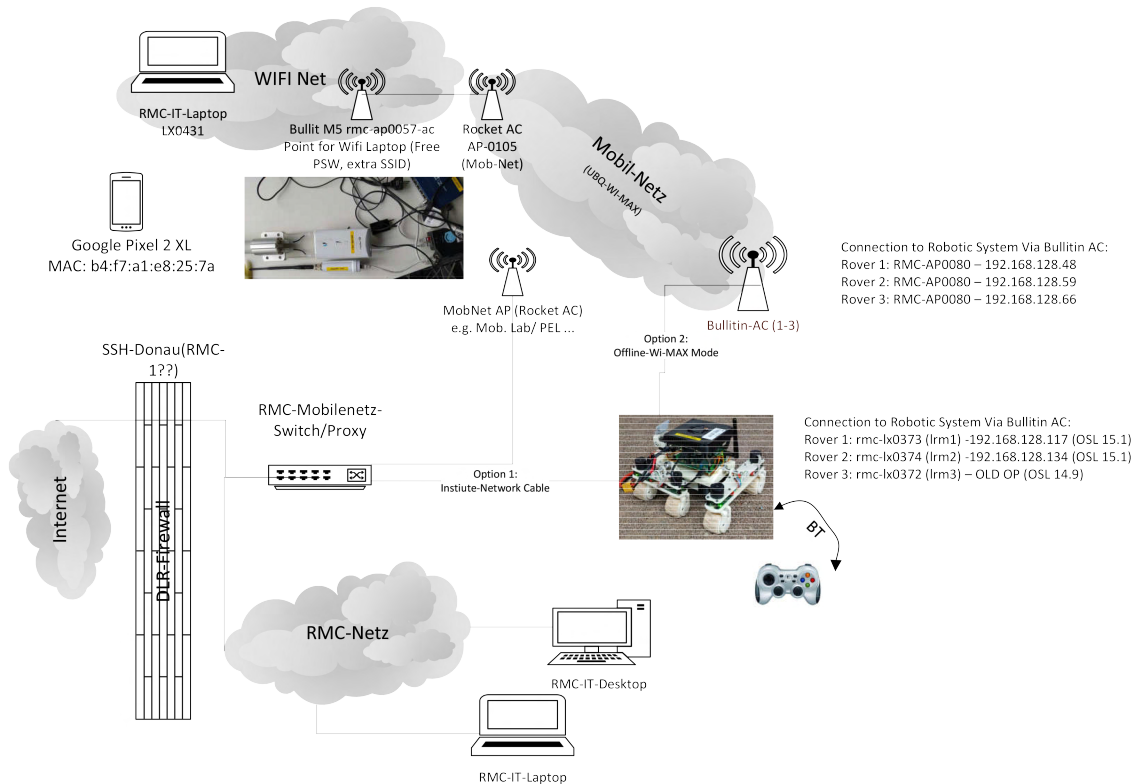


Figure 2.9: Network architecture for the smartphone connection



### 3 Related work

For this thesis, a smartphone will be utilized to control the rover. There are multiple operating systems for smartphones and it has to be decided, which one fits the best. Creating apps for an Apple smartphone is more restricted than for Android as the software stack is already fixed [5]. [5] and [6] used Android because it is an open-source operating system that provides multiple functionalities such as an application framework and a basic operating system.

The implementation of Android Apps is done in Java with a Java Software Development Kit (SDK). Java is though not sufficient for multimedia transfer [6].

[5] and [7] extended the Java application framework with the Android Native Development Kit (NDK) in order to use C/C++ functionalities in Java to implement multimedia transfer. To utilize the source code, it has to be transformed into native machine code and then included in the respective Android application package. This is done through the NDK [5]. For the purpose of multimedia transfer, the Java code has to also interfere with the C/C++. That can be done with Java Native Interface (JNI). This includes C native methods into Java application so that they can be used by the Java part [7]. The architecture is shown in figure 3.1.

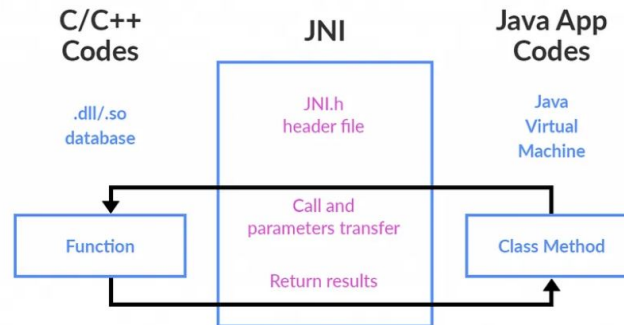


Figure 3.1: Structure of the connection from Java to C/ C++ code over JNI [8]

The goal of both papers was to play files by the FFmpeg software on Android. The FFmpeg software is not suitable for this thesis as it is mainly for recording and displaying multimedia streams and not for transferring the streams [5].

In the work of [9] the MPEG4 format is utilized for the camera stream. As in the work of [7] a Java app with the extension to C/C++ is created. The difference to this thesis is that [9] records the camera of the smartphone and transfers it to a computer. The structure of the streamer is not suitable for the rover. However, the usage of this data format might be beneficial for this use case. The architecture of how to extend Java in order to use C/C++ functionalities will be taken into consideration for the further thesis.

[6] presents an open-source multimedia framework for transferring media and handling media streams, called the GStreamer. It is realized as a plugin and pipeline architecture. GStreamer was not developed primarily for mobile devices but has an extension for Android. For this, the Android NDK has to be included [10]. In [11] are examples shown on how to create Android Apps which include a running GStreamer pipeline and display a media player. This can be utilized for the purpose of this thesis. However, it has to be considered that the examples are based on Android 9, which is from 2018 [12]. As mentioned in section 2.1.2, the utilized smartphone has the Android version 11. It has to be checked if the examples are compatible with a newer version. The GStreamer can be combined with the architecture of [5] in order to transfer multimedia streams and to have C/C++ functionalities in the Android app. This will be considered in the development of the concept (section 4).

Another approach is made by [13] and [14]. They both created a ROS based app for Android. Both apps utilize RViz for visualization and have options for the user to display multiple of the available topics. The app of [13] is called iviz and is based on the Unity Engine. It was developed to have a mobile Augmented Reality (AR) ROS app to display the robot and some virtual environment in the camera stream of the smartphone. It can also only display the robot in a virtual environment [13]. As AR is not utilized in this thesis, the iviz app is not fitting. The mapping and camera stream could be displayed over ROS nodes and therefore shown in the iviz app but because of the AR features, there is unused memory capacity which could cause delays.

The app of [14] is called ROS-mobile and is visualizing available topics. The app can be downloaded over the Google Playstore or can be built over source with Android Studio to allow modifications in the Java code. In the app, the user can add widgets which are available topics of the project, which are shown if the app is connected to the master node [14]. In comparison to [13] is ROS-mobile a better fitting option as there is no AR feature and therefore less unused memory capacity and easier adaptation for this project. It has to be analyzed if it is possible to connect the app with the DLR intern network with the necessary security aspects.

[15] utilized a different approach to transfer media to mobile devices. Instead of creating an app a remote connection through the VNC system is made. This is a server/client architecture, which enables remote control and visualization. For this, the VNC Viewer could be used. This program has an Android app for the client part, which can be downloaded over the Google Playstore and then utilized. The VNC system is not a perfect solution as mobile systems can't have a high compression ratio for videos [15]. It has to be analyzed if this approach is working with our project as the camera stream and the map over ROS have to be displayed on the mobile device without a significant delay.

In the work of [16] a smartphone was utilized for teleoperating a robot arm. The internal six degree of freedom sensors of the phone operated as the controller. It was mentioned, that the six degrees work better than a two or three degree of freedom joystick. The idea of using six degrees of freedom have to be considered in further work as the spacemouse controller has the prerequisite for it. Furthermore, [16] utilized WiFi for sending the commands to the robot, which could be applied to our system.

## 4 Concept

### 4.1 General goal

The general goal of this thesis is to have a teleoperation of the rover over a smartphone. For this, an Android based smartphone will be utilized. On the smartphone should be the camera stream of the rover and the computed map of the surroundings of the rover shown. Furthermore, an user interaction should be possible. The map illustrates where obstacles are in the environment of the robot spotted and is constantly updated as the rover moves. This has already been realized in ROS in the work of [17] and [18] and has to be now transferred to the smartphone. For this the ideas of having a VNC Viewer, utilizing a ROS app or using the GStreamer in combination with a Java app were considered. The rover should be controlled using the spacemouse connected to the smartphone. The general goal is illustrated in figure 4.1.

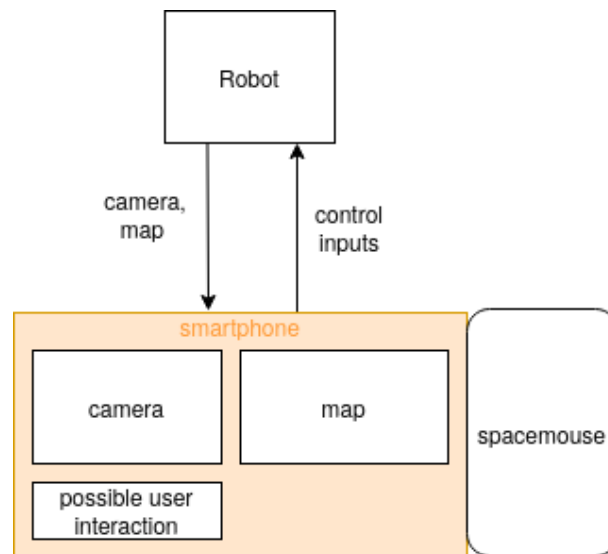


Figure 4.1: Architecture of the general goal

In order to realize this, different approaches were found during the literature review and are listed in chapter 3. In the following a concept for the connection of the spacemouse to the rover will be created. Furthermore, the transferring of the data from the rover to the smartphone is needed. For this the ideas were: having a VNC Viewer, utilizing a ROS app or using the GStreamer in combination with a Java App. In the following sections each option with its advantages and disadvantages will be discussed and an architecture of a possible



realization of each concept created. Each concept will be then implemented, tested and evaluated in order to determine the ideal option.

## 4.2 Connection spacemouse to robot

The smartphone has a spacemouse attached, which is receiving control inputs through the user. These have to be send to the robot in order to execute them. For this, a wireless connection has to be created. There are two main options: over the network or over Bluetooth. Over the network it can be done either over UDP or Transmission Control Protocol (TCP). Bluetooth is not suitable as the phone has to be near to the rover and the goal of this thesis is to control the rover also from a distance (e.g. from a different room) [19]. Furthermore, Bluetooth has a lower performance than UDP [20]. The main difference between the two network protocols is, that TCP is checking if the sent packets are correctly and complete delivered. UDP is in comparison to TCP faster as packet losses are ignored [21]. In the setup of this thesis, it is not relevant if every control input is received, it is more important that the transmission delay is minimized, so that the rover can be controlled with the least possible delay. For that reason, UDP will be utilized to send the control inputs from the spacemouse to the robot.

## 4.3 VNC Viewer

One concept option by [15] is to have a remote connection over the VNC Viewer. For this concept a VNC Server on the robot computer and a VNC Client on the smartphone has to be created. The VNC Server has already been created for remote controlling of the robot and could be utilized for this project. For the VNC Client, there exists in Google Playstore an app that could be downloaded and used. The architecture on how to integrate the VNC Viewer in our system is illustrated in figure 4.2. In this architecture the LN-manager is starting the VNC Server. The server is showing the camera stream and the map which is provided by ROS. To have an easy usage for the user, a GUI has to be created. The smartphone is then connecting to the Server and receives the remote visualization of the GUI. Over the GUI the user can remote control the rover. But this remote controlling is not including the spacemouse as it is not connected directly to the smartphone and not to the rover. For this, a communication interface has to be created. This is an app, called control app, which receives the control inputs from the spacemouse and then sends, as mentioned in section 4.2 over UDP the control inputs to a communication interface on the robot computer. This communication interface then sends the received control inputs to the LN-manager so that it will be transferred to the robot. The communication interface on the robot computer has to be also created and will run next to the VNC server.

The advantages of this concepts are that the implementation of the VNC server and client are already done. Furthermore, this architecture is not specific on Android or mobile devices and could be easily adapted for other systems e.g. Windows computer.

The disadvantages are that two parallel parts have to be created as the controlling over the

spacemouse is not compatible with the VNC system. On top of that, the speed is limited as remote controlling has some delays and also mobile system aren't able to have a high compression ratio for videos. It has to be tested if this leads into problems while performing a teleoperation.

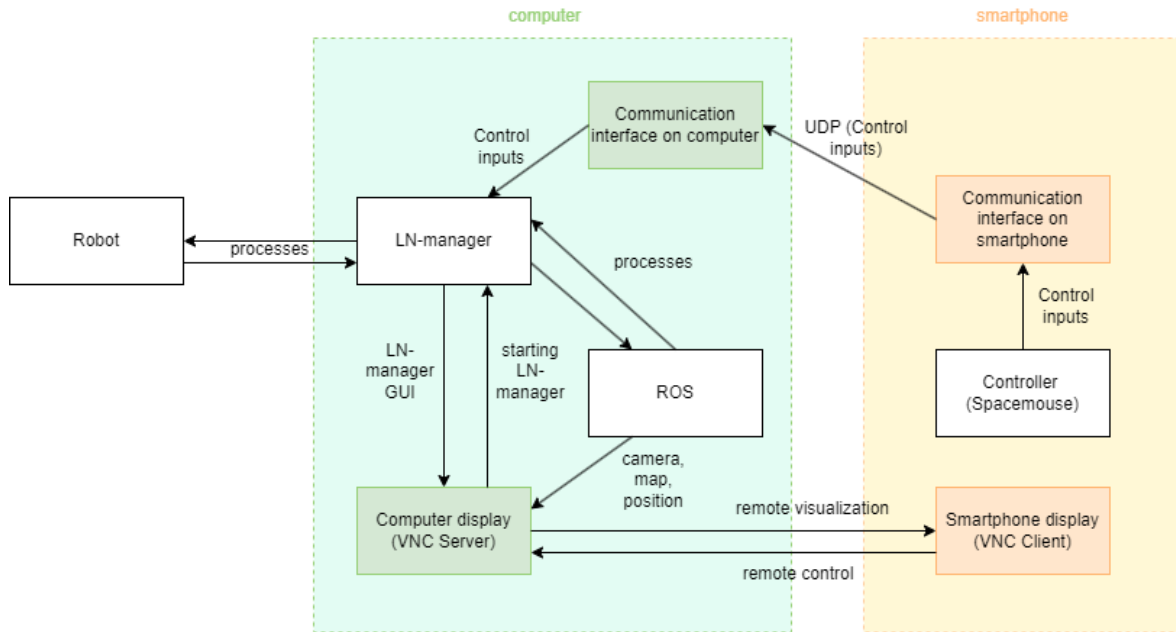


Figure 4.2: VNC Viewer architecture

## 4.4 ROS App

Another approach is to use a ROS app instead of the VNC app. There are two different possible ROS apps that could work, *iviz* [13] and *ros-mobile* [14]. As already mentioned in section 3, *iviz* is not as fitting as *ROS-mobile* as it has AR features, which is not needed for this thesis. Because of that, it will be in the following only focused on *ROS-mobile*. *ROS-mobile* is an android specific app that can be downloaded over the Google Playstore and can be modified over Android SDK. The concept behind using a ROS app is to directly being able to use the camera and map as both are already ROS nodes. *ROS-mobile* is then another ROS node that can visualize all ROS topics that are available in the project. Over *ROS-mobile* the user interaction over the app can be utilized. However, the spacemouse control inputs can't be integrated as the spacemouse is not integrated in ROS. For that, a different stream has to be created. This can be done with an UDP connection as in the architecture of VNC (figure 4.2). For this a communication interface on the smartphone and a communication interface on the robot computer have to be implemented. These run in parallel to the ROS nodes. Over the communication interfaces the control inputs from the spacemouse are transferred to the LN-manager in order to use them for the robot.

The advantages of these concept are that the camera and map information can be directly used over ROS and don't have to be transformed to send them to the smartphone. This could lead to a faster communication and less delay on the visualization. Furthermore, the implementation of the ROS app is already done and has to be only adapted to our purpose. However, it has to be checked, if the app is working with the mobile net setup. The disadvantages are that, a parallel UDP connection for the control inputs and user interaction is needed which is not as convenient for the user. Furthermore, the app works solely on Android devices.

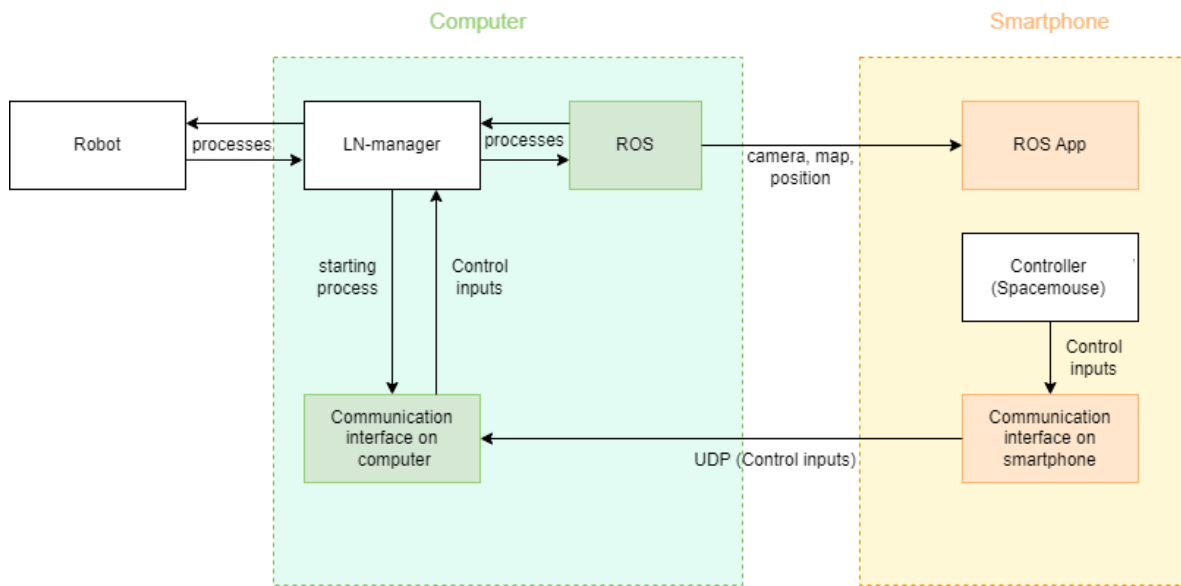


Figure 4.3: ROS-mobile architecture

## 4.5 GStreamer with a Java App

A different concept is to use the combined ideas of [6], [7] and [5]. This results in using a GStreamer to transfer the media streams (camera, map etc.) to the smartphone. In order to have an Android App with GStreamer compatible the Android NDK and the JNI have to be utilized. To transfer the control data, another transfer method has to be added as GStreamer does not work bidirectionally. For this, as mentioned in section 4.2, a UDP connection will be utilized. This concept architecture is shown in figure 4.4. In this figure the interfering of the different parts is illustrated. The LN-manager is communicating with the robot and with ROS over processes. For having the two transfer streams a communication interface on the computer and on the smartphone is needed. The communication interface on the computer receives the camera and map data from ROS and sends them over a GStreamer pipeline to the communication interface on the smartphone. The communication interface on the smartphone receives from the controller (in this case the spacemouse) and from the

user interaction with the app control inputs, which are sent to the computer via an UDP connection. These control inputs are sent from the communication interface on the computer to the LN-manager, so that other processes are able to access it. The smartphone app is created from the communication interface on the smartphone with the help of the Android SDK and NDK.

The biggest advantage of this concept is that the transferring delay is probably low. This results in fast controlling, which is important for teleoperations. Furthermore, the app is specifically designed for this purpose and therefore, it has an intuitive and easy user interaction. It can be also easily modified and extended.

One disadvantage of this architecture is that it is more complex to implement as two different transfer streams, a creation of a Java App with the extension with C/C++ and the transferring of the camera and the map from ROS to GStreamer are needed. Furthermore, it has to be tested if GStreamer is compatible with ROS. Another disadvantage is that this architecture is only working on Android devices.

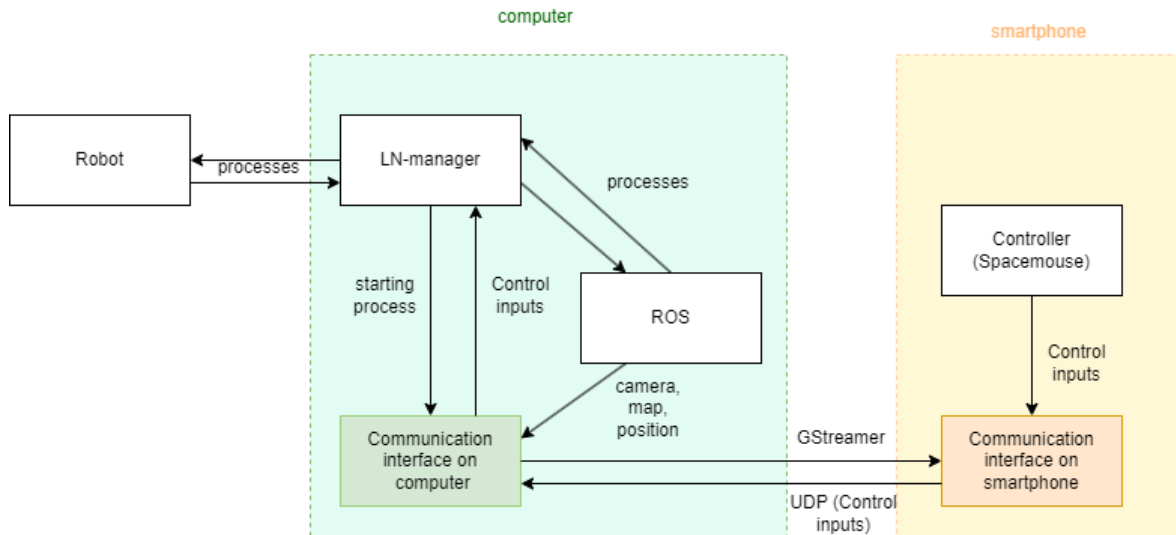


Figure 4.4: Architecture option with GStreamer and a Java App

## 5 Setup and Implementation

### 5.1 Control App implementation

The main purpose of the control app is to read and send the spacemouse data to the rover. On top of that, settings like changing between driving and pan/tilt, driving mode and which control axis from the spacemouse is responsible for which movement has to be implemented. Also this app should be able to work with both rovers. For this, the GUI illustrated in figure 5.1 was designed.

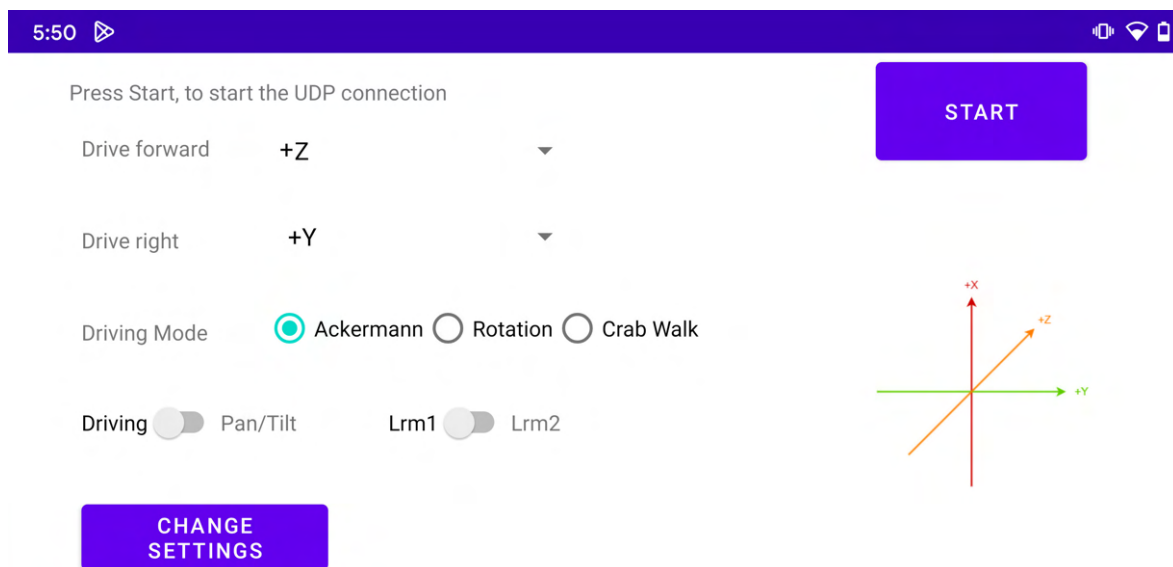


Figure 5.1: Control App GUI

As the spacemouse controller has six degrees of freedom, it has to be figured out which to utilize and how. Through testing, it was noticed that the values are highly connected so that a movement on one degree of freedom will also change other values. Due to this, the decision was made to not fully use the potential of the spacemouse and only utilize three degrees of freedom. Out of these three, only two will be used at the same time as it was noticed that the x and z axis are highly influencing each other. As seen in the coordinate system of figure 5.1 the x axis is moving the spacemouse to the front and back whereas the z axis consists of the movement pressing down and up. In theory, these movements are decoupled but in reality when moving front or back, the spacemouse will also move due to the pressure caused by

the user in the z axis. The same for the z axis as it has to be a very precise movement to not interfere the x axis. For the x axis, the x movement of the spacemouse is utilized. Unlike x the y and z axis are from the rotation around the y and x axes. Through testing, it has been noticed that these are working better. In general, the idea is to have one degree of freedom for driving forward/backward and tilting the camera, and one for driving to the sides and executing the pan movement of the camera. The user should be able to choose which of the three degrees of freedom is for which movement responsible. Furthermore, the direction of the axes should be also adaptable, e.g. if pressing down resembles driving forward or if pressing down resembles driving backward. This is realized by having signed axes.

In the GUI, there is a button to start the UDP connection to the rover, which then sends in a thread the read spacemouse data. This data is read over the USB connection and converted into an integer array of length six as this stores every value of the six degrees of freedom spacemouse. There is a switch that declares to which rover the UDP connection gets built. This is because the rovers have different Internet Protocol (IP) addresses. For the connection, the smartphone is the client and the rover is the server. When pressing the change settings button, the settings of drive forward/tilt, drive right/pan axis, the driving mode and if it is the drive or the pan/tilt mode get read and then send to the rover. For this, there is a break in the spacemouse data sending as otherwise the data would overlay. The logic of this is shown in figure 5.2.

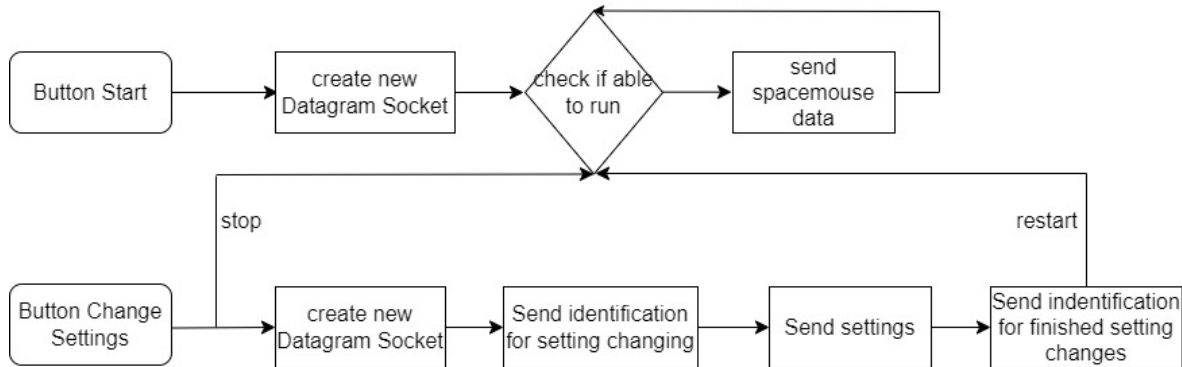


Figure 5.2: Control App Code Logic

For the start, there are some default settings given. For Drive forward/ Tilt is the + Z axis, Drive right / Pan is + Y, Driving Mode is Ackermann, Driving and LRM1 is chosen. These settings have to be tested during a user case study if they are the most used or preferred ones. On the rover, the server part of the UDP connection is implemented. As it is UDP and data can get lost, there are checks if the data is correct and if data is missing in order to avoid mistakes like having a control input mapped to the wrong axis. It receives the sent data from the smartphone. First, there is a check if the obtained data are the spacemouse control inputs or the changed settings. If they are the changed settings, they are saved in the relative variables. The control inputs of the spacemouse need to be first converted to integer. Depending if driving or pan/tilt mode is chosen, the calculation differs.

For driving, the velocities for forward and right have to be calculated. For that, there has to be a check on which axis is chosen in order to utilize the respective control inputs. These two values will be scaled by the factor 10 and noisy peaks (over 1500 or under -1500) will be cut off. Furthermore, there is an observed noise around -10 and 10. To avoid the wrong values, these values are set to 0. Afterwards, the angle has to be computed. For that, the control input data for the axis for right are scaled between 0 and 360. The positive values are scaled between 0 and 180 and the negative between 180 and 360 as this refers to a turn to the left.

For camera movement, there is a different approach as the camera movement should be slower and steadier. For this, there is a +1 increment in pan or tilt every time there is a positive value in the respective axis. Similar is the handling with negative values.

Camera movement and driving are not possible at the same time, so when driving mode is activated the camera moves back to the start position. In the other mode, the velocity gets set to 0 so that no driving is possible.

After the user case study in section 7.1, the following implementation changes have been applied. The pan/tilt update is solely happening after every 4th control input on the spacemouse to ensure an even slower movement. In between sending control inputs from the spacemouse to the rover a 10 ms rest is applied in order to avoid crashes due to parallel sending of UDP packets. Also, the text of drive forward was changed to drive forward / tilt and drive right to drive right / pan to display the full functionality of this value. The feature of automatically having the camera set back to its start position was shifted to a button in order to have the user decide when to use it. The code logic is changed so that there are now two buttons interrupting the sending loop of the spacemouse data. The change camera settings are similarly implemented as the change button settings (logic illustrated in figure 5.2). Furthermore, there is a check added so that the two buttons are not sending at the same time in order to avoid package losses. The GUI with the newly added button is shown in figure 5.3.

## **5.2 VNC setup**

There is already a variety of existing and working VNC mobile apps available in the Google Playstore. Due to this, one of them will be utilized. The app, called VNC Viewer, was selected because it is from the same developer as the utilized desktop app. The connection to the rovers is made over the mobile net, shown in figure 2.9. To create the VNC connection the IP address (192.168.128.117 (LRM1) or 192.168.128.134 (LRM2)) and the port (5900) of the robot is needed. Furthermore, it can be selected between remote controlling or only observing and if a desktop preview is wanted. Both options could work as only a display of the camera and map is needed. With only observing, there is no option in changing settings, as this could be useful, the remote controlling option was chosen. The desktop preview option is not relevant for the setup but is turned on, as it helps with debugging. The setup for connecting to LRM1 is shown in figure 5.4. The VNC App can run in parallel to the Control App. For that, both apps have to be started one by one. Both processes can also work in the background and foreground without disconnecting, it can be dynamically chosen depending on the specific

needs which GUI is displayed.

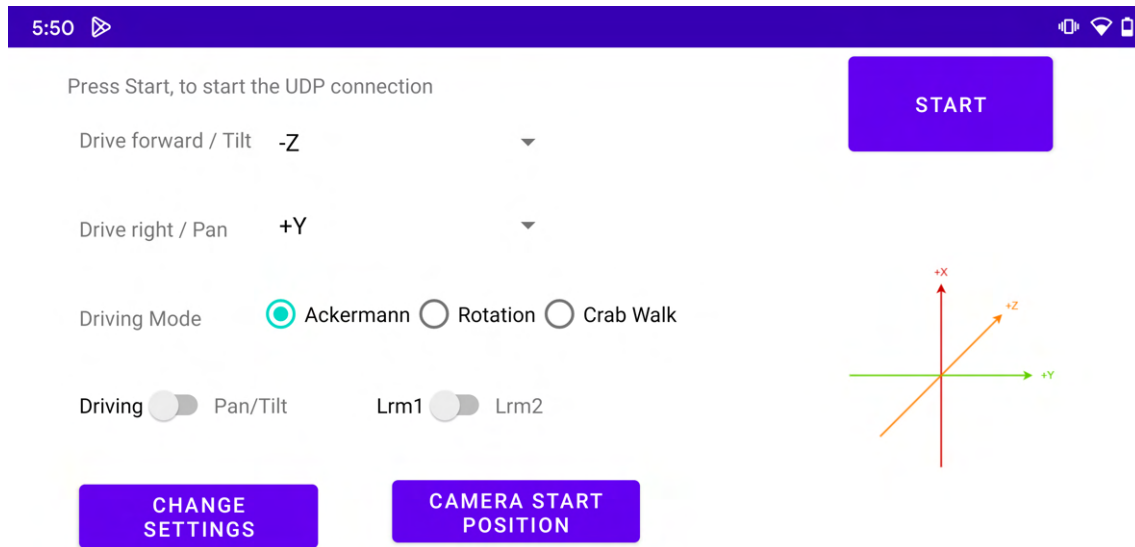


Figure 5.3: Final Control App GUI

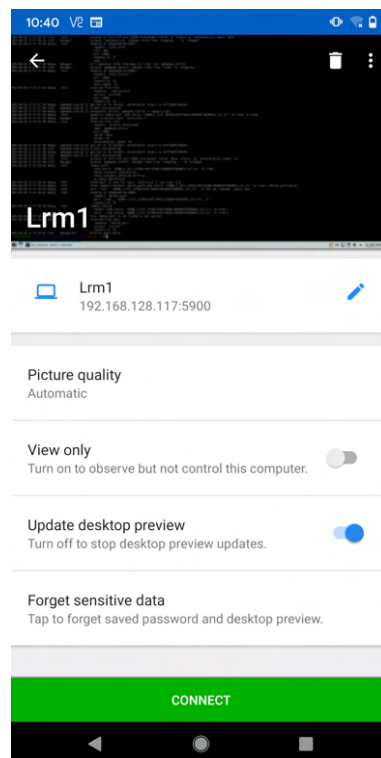


Figure 5.4: Connection setup for VNC for LRM1



### 5.3 ROS-mobile setup

For setting up ROS-mobile, the IP address of the rover and port is needed, the setup for LRM1 is shown in figure 5.5. The setup for LRM2 is similar, only the IP address is different. The IP address for LRM2 can be seen in figure 2.9. After connecting, the displayed widgets can be adapted. For this, some parts in the code have to be changed as the topic names are fixed in the app and don't match with this project. This can be changed in the GridMapEntity and in the CameraEntity class. A comparison between raw and compressed images was made in order to figure out which to utilize. To show the difference, timestamps were printed after drawing the images, which means the displaying of a new camera image on the screen. It can be seen that drawing one raw image takes approximately 1 second whereas the time of one compressed image is around 4 milliseconds (Listing 5.1, 5.2). The complete delay in the lab setup of sending the raw images via UDP from the rover to the smartphone, converting and drawing is around 5 seconds. In comparison, the delay of the compressed images is under 1 second.

---

```
I/System.out: Drawing at: 2023-04-25T12:23:10.150Z
I/System.out: Drawing at: 2023-04-25T12:23:11.090Z
I/System.out: Drawing at: 2023-04-25T12:23:12.100Z
```

---

Listing 5.1: Timestamps Drawing with raw images

---

```
I/System.out: Drawing at: 2023-05-11T10:50:13.061Z
I/System.out: Drawing at: 2023-05-11T10:50:13.112Z
I/System.out: Drawing at: 2023-05-11T10:50:13.146Z
```

---

Listing 5.2: Timestamps Drawing with compressed images

The resulting conclusion is to use the compressed images for transferring the camera stream. In figure 5.6 an example output of the camera stream and in figure 5.7 combined with the map is illustrated.

### 5.4 GStreamer implementation

The implementation of GStreamer consists of multiple steps. First a pipeline has to be created as these are used for acquiring, compressing and sending the data. The data is sent in packages. When receiving the packages, they have to be assembled and decompressed [22]. The utilized pipeline for the camera stream for LRM2 is the following:

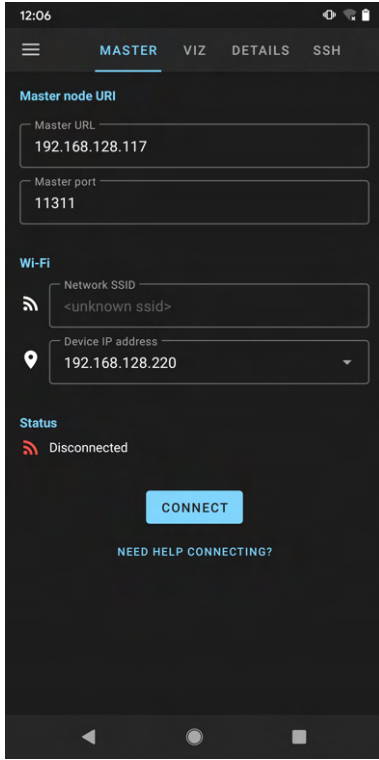


Figure 5.5: Connection setup for ROS-mobile

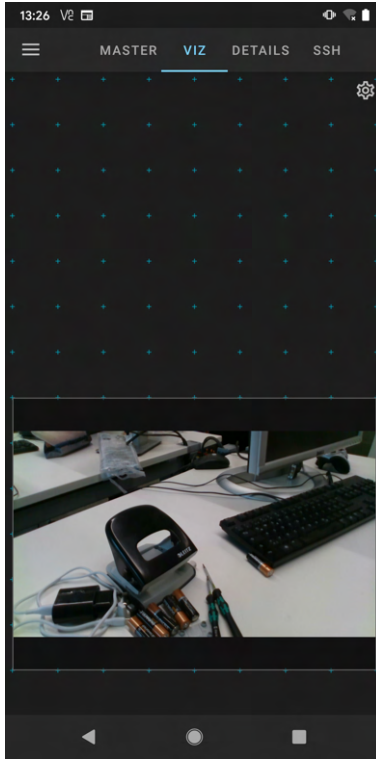


Figure 5.6: Camera stream of ROS-mobile

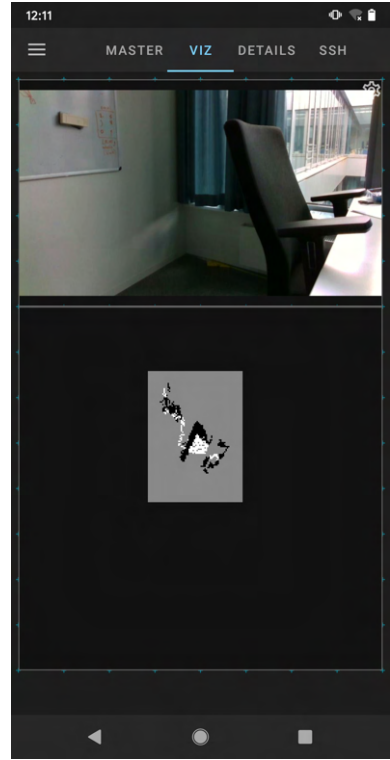


Figure 5.7: Camera and map of ROS-mobile

---

```
rosimagesrc ros-topic=/lrm2/color/image ! videoconvert ! x264enc
  speed-preset=ultrafast tune=zerolatency intra-refresh=true bitrate=100 aud=false
  vbv-buf-capacity=100 option-string="slices=4" byte-stream=true ! rtp264pay
  config-interval=1 timestamp-offset=0 seqnum-offset=0 ssrc=1 ! udpsink port=50042
  sync=false host=192.168.128.40
```

---

For the map the pipeline is:

---

```
rosoccgribsrc ros-topic=/lrm2/grid_map ! videoconvert ! x264enc
  speed-preset=ultrafast tune=zerolatency intra-refresh=true bitrate=1000
  aud=false vbv-buf-capacity=100 option-string="slices=4" byte-stream=true !
  rtp264pay ! udpsink port=50043 sync=false host=192.168.128.40
```

---

Both pipelines are sending ROS topics. The pipeline consists of multiple specifications. X264enc is the chosen encoding specification, as it is one of the most commonly utilized encoders due to its fast speed [23]. The speed-preset is set to ultrafast, which produces the smallest possible images to ensure the fastest possible connection. Tune is set as zero latency as no delay is desired. Intra-refresh is a setting where especially large frames get avoided, this ensures fast communication. The bit rate indicates the ratio of the amount of data that is displayed in a certain amount of time. Access Unit Delimiter (AUD) is per default set to false

in the X264 encoding [24]. Over experiments, it is shown that the vbv-buf-capacity of 100, bit rate of 1000 and option-string=slices=4" works well for this system. For transferring the data Real-Time Transport Protocol (RTP) packets are needed, which is done by rtph264pay [25]. These packets can then be sent over UDP, for this a port is needed. As both pipelines are sending at the same time different ports have to be utilized.

The next step is to decode the pipeline and display the video stream in the app. As mentioned in section 4.5, the GStreamer methods for decoding the pipeline are C based and can't be converted to Java. To solve this problem, native methods can be utilized. With this, a link between Java and C was made. Over the GUI in Java, the pipeline decoding can be started in C. The Surface View is a graphical element that can be utilized for initialization in C and visualization in Java, which makes it suitable for displaying the GStreamer media stream.

As it is a Java app and only requires a button for starting the stream and a Surface View for displaying the stream, it can be integrated into the control app. For this, two different designs were made. One for only displaying the camera stream, which was created to have a user case study purely focusing on the camera stream. The GUI is shown in figure 5.8. It can be seen the integration with the control app (figure 5.1). The only difference is the camera view and two buttons for starting and stopping the camera.

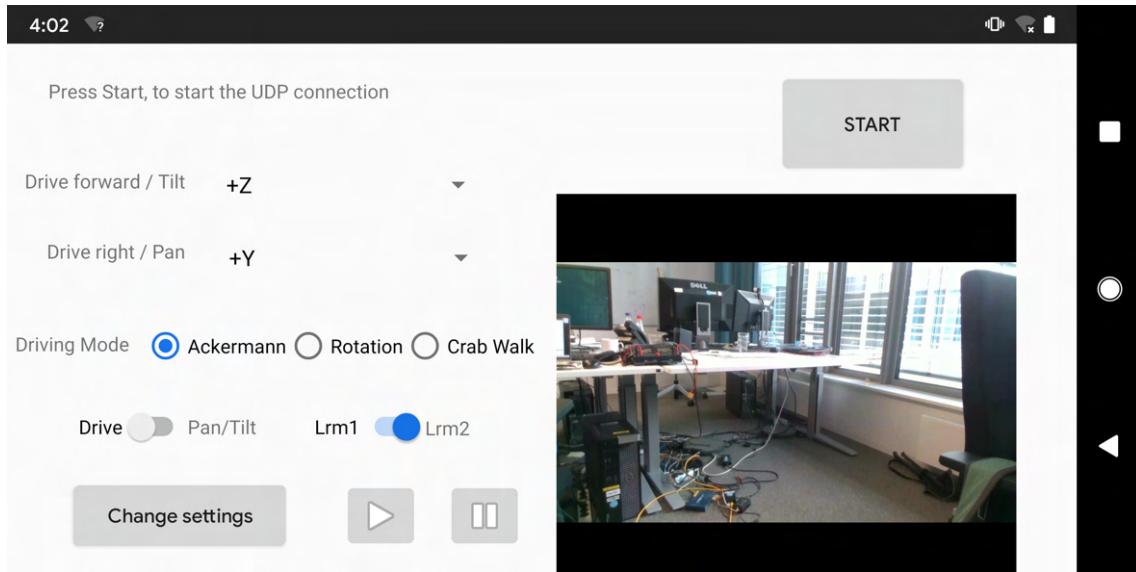


Figure 5.8: Screenshot of the GStreamer GUI before adding the map

The second design option is without the start and stop buttons. There are two different buttons, which are for starting the camera stream or the map. When clicking on the respective button either the camera or the map will be displayed on the surface view. As there is limited space on the smartphone, there was the decision between having the camera and map next to each other or having the settings displayed at all time. For a teleoperation, the driving modes and the change between driving and camera movement are important and were often utilized during the user case study. Due to this, the settings were instead of the map chosen to be

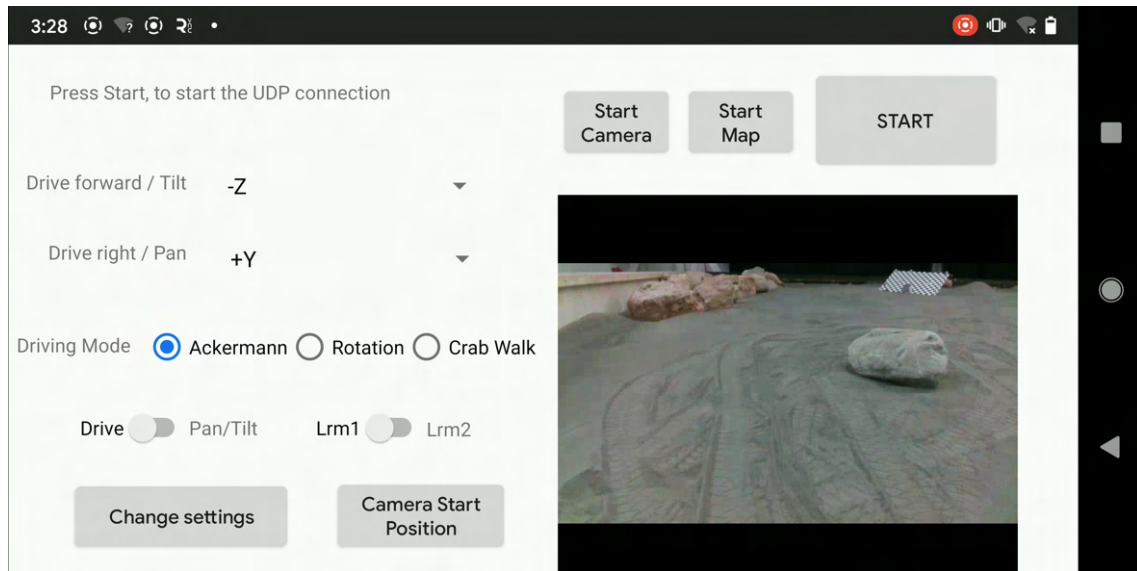


Figure 5.9: Screenshot of the final GStreamer GUI

permanently shown next to the camera. As already mentioned in section 5.1 a camera start button was added after the first user case study was added. The GUI is shown in figure 5.9

## 6 Testing

As the Human-Machine Interface consists of different components, each was tested on its own before having tests of the whole Human-Machine Interface.

### 6.1 GUI testing

#### 6.1.1 VNC GUI testing

With the VNC app, the robot can be fully remote controlled and no other device is needed. When using another device to start all processes, the app only has to be connected to the rover and the camera stream and map will run automatically. If no other device is utilized, the following steps have to be done. First, the login screen will appear. It can be seen that the resolution is not made for the smartphone, due to the black stripes on the side. To use the app properly, the user has to zoom in as the text is too small to be read and clicking the button is barely possible. To type in the password either the copy paste function or the virtual keyboard can be used. After logging in, the LN manager can be started over the terminal. For teleoperating the rover, the created map and the camera stream are needed. In previous works of [17] and [18] a grid map for the obstacles was created. This can be visualized next to the camera stream in RViz. To start the rover and the mapping the following processes have to be started: `base_link_to_camera` and every process in `03_Mission_Part2` and in `02_Mission_Part1_Stateless`. After starting these processes RViz will already open with the right setup, shown in fig. 6.1. The user can choose which features are visible. The camera stream is due to the screen resolution small and can barely be used for the teleoperation. There is the option to zoom in, but then the map is either cut off or not shown anymore. A zoomed camera view is displayed in 6.2. Furthermore, it can be seen, that there is a taskbar, which can't be moved or minimized. This might be disturbing for the users as it is partly covering the screen. The experience has shown, to use VNC properly a pre prepared GUI display on the rover side shall be created accordingly to the mobile device screen details, which shall be started after connecting the VNC to the rovers. Due to the lack of specific video compression features in VNC, this has not been realized in this thesis.

#### 6.1.2 ROS-mobile GUI testing

The ROS-mobile app can't run on its own. All the processes mentioned in 6.1.1 have to be started as well. This can be done via an external computer over VNC or the VNC app. Afterward, the app can be connected over the connect button. If the camera stream and map are already added as widgets, they will automatically appear after connecting. In combination

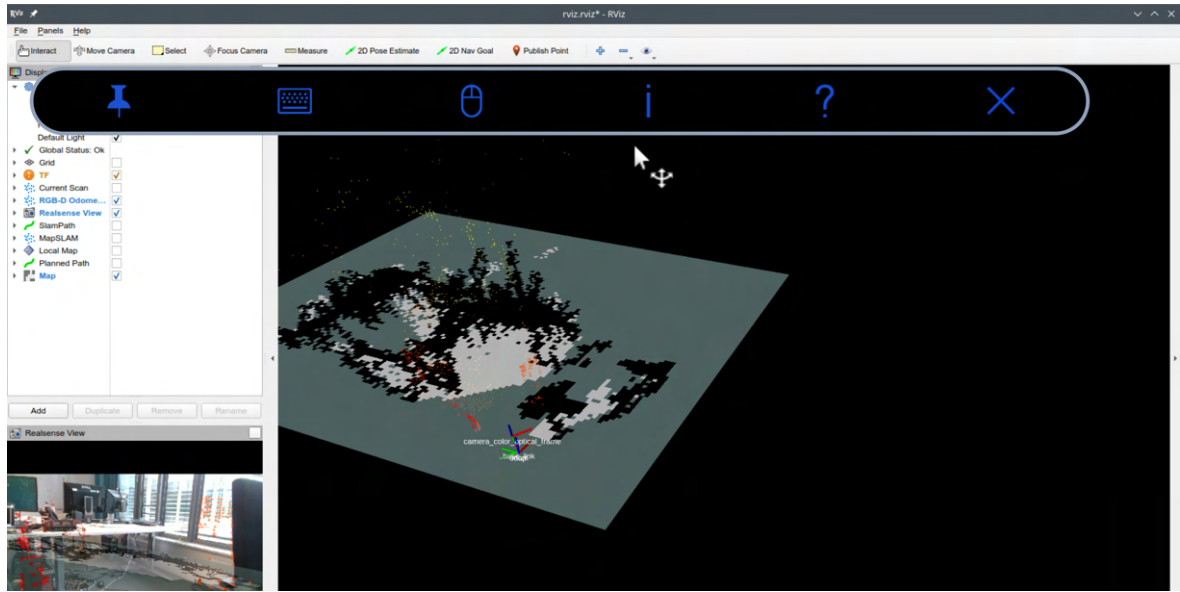


Figure 6.1: RViz on the smartphone over VNC

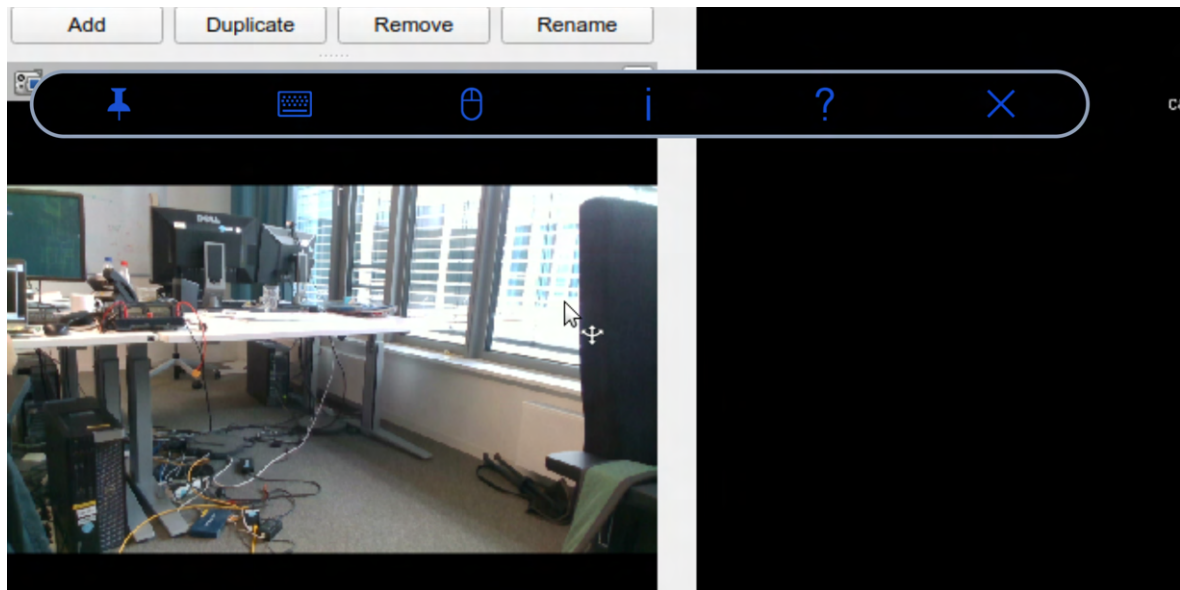


Figure 6.2: Increased camera view in VNC

with the spacemouse, it is more intuitive to use the landscape format. For this, the setup has to be adapted as the resolution and positioning of the widgets are only fitting either for portrait or landscape mode. While changing from portrait to landscape mode, the app disconnects from the rover, which makes it more cumbersome to use. ROS-mobile can run in the background but is not reliable in this. So when exiting the ROS-mobile, it sometimes disconnects from the rover. It has to be tested if this is suitable for a Human-Machine Interface

as changing settings in the control app during a teleoperation is necessary.

### 6.1.3 GStreamer GUI testing

Similar to the ROS-mobile app, GStreamer can't run on its own. The processes have to be started via an external device or over the VNC app. Afterward, GStreamer can be started with one button. As it is combined with the control app, no other app is needed and the control settings can be easily changed while having the camera stream. In comparison to VNC and ROS-mobile, the size of the camera view can't be adapted by the user. For further evaluation of the GUI, a user case study is necessary. This will be mentioned in section 7.

## 6.2 Camera testing

### 6.2.1 Camera testing of all options in the lab setup

After implementing and individually testing all options, a lab test for comparing the different camera options was performed. The first tests consisted of solely camera transferring as this can be done in the lab setup which uses the institute network. For this setup, no WiFi between the rover and antenna is created. However, the antenna is still needed for creating a connection for the smartphone. The rover and antenna are plugged via an ethernet cable to the institute network. This ensures the fastest possible internet connection. The results after completing the experiments were that VNC had a noticeably bigger latency than the other two options. Furthermore, it fluctuated between 2 seconds and 5 seconds. ROS-mobile and GStreamer had in general no recognizable difference. One has to mention that delay peaks appeared with ROS-mobile. These consisted of a delay of around 5 seconds. However, they appeared irregularly. During some tests, no peak appeared, whereas in other experiments they appeared in a significant amount. With neglecting these irregularities, the general delay of GStreamer and ROS-mobile was in milliseconds and for the human eye barely recognizable, whereas in VNC was on average around 3-4 seconds. The most stable option was VNC as it had no app crashes during the experiments. ROS-mobile didn't have a complete app failure but had some of the previously mentioned peaks and starting problems in terms of connecting to the rover and connecting to the ROS camera topic. GStreamer had some startup problems as the app irregularly didn't start. Furthermore, the launch of the camera stream took between 1 to 5 seconds. After restarting the app, the starting time reduces which indicates that the pipeline gets cached. The next step is to test the camera options with the mobile setup as the results could vary from the lab setup.

### 6.2.2 Camera stream testing of all options in the mobile setup in the lab

As the application areas of the rover don't have access to the institute network, the mobile setup has to be utilized. For this, a WiFi between the rover and antenna is created. As expected, the latency increased in general around 1-2 seconds as WiFi connections are not as fast as ethernet connections. Except for the higher delay, the experiments showed similar



results as gathered during the lab setup tests. Summarized VNC had the highest delay around 3-7 seconds and no app failures. Whereas ROS-mobile and GStreamer had a similar delay, which was in this setup noticeable between 1 and 2 seconds. Failures happened sporadically during the start of the GStreamer app. Furthermore, irregularities appeared during the camera stream of ROS-mobile and during the initialization of the GStreamer camera stream. In the next tests, the Control App will be tested in order to have a full test of the Human-Machine Interface.

### 6.3 Control App testing in the lab

Before combining the control app with the camera stream to have a Human-Machine Interface, the control app has to be tested. For this, driving experiments around the lab in the mobile setup were performed. The mobile setup was chosen as driving with a cable connected to the floor is barely possible. On the lab floor, the tests were barely possible to execute as the wheels were not designed for smooth surfaces. As a result, the rotating wheels could barely move the rover as they were slipping. The only possible testing consisted of the camera movement. The rover could execute the given control inputs and could also adapt to different settings. There was a low amount of crashes in the app, which has to be further investigated. The delay between user input and rover moving was not significant. Overall, the latency did not exceed 1 second. However, there was an irregular latency appearing from time to time of 1-2 seconds. Through observations, it could be identified that the spacemouse reacted sensibly to the user input. This means, that small inputs reacted to bigger camera movements. If this is beneficial for the user, has to be further researched. Another feature that has to be evaluated is, whether releasing the controller should result in the camera moving back to its initial position or not. To answer these research questions, a user case study has to be performed and evaluated (7.1 User Case Study on Vulcano).

In order to fully test the control app, the rover was moved to a test area of DLR which consists of volcanic sand, which resembles moon surfaces (see figure 6.3).

In this area, the rover was able to drive. Through inputs via the spacemouse, the robot was able to navigate through the area. However, there were occasions when the robot was struggling with driving. This impacted the evaluation of the driving as it was not always clear if hardware or controlling problems caused the issue. Further evaluations on different terrains have to be done in order to identify the exact problem (6.4 Human-machine interface testing on Vulcano). Changing between driving and camera mode and between driving modes worked accurately without noticeable delay. The spacemouse is not as intuitive as it differs from common controllers like a joystick. During testing a parkour, which consisted of turns, driving in a straight line and slalom was completed. However, during driving the parkour the rover couldn't move as precise as expected. As a comparison, a different controller with a joystick completed the same parkour more precisely and faster, which leads to the result that the spacemouse is harder to control and not appropriate for precise driving. In order to have an evaluation on preciseness multiple users have to test it, as the driving and perception varies. With switching between different driving axes, the driving control technique had to be



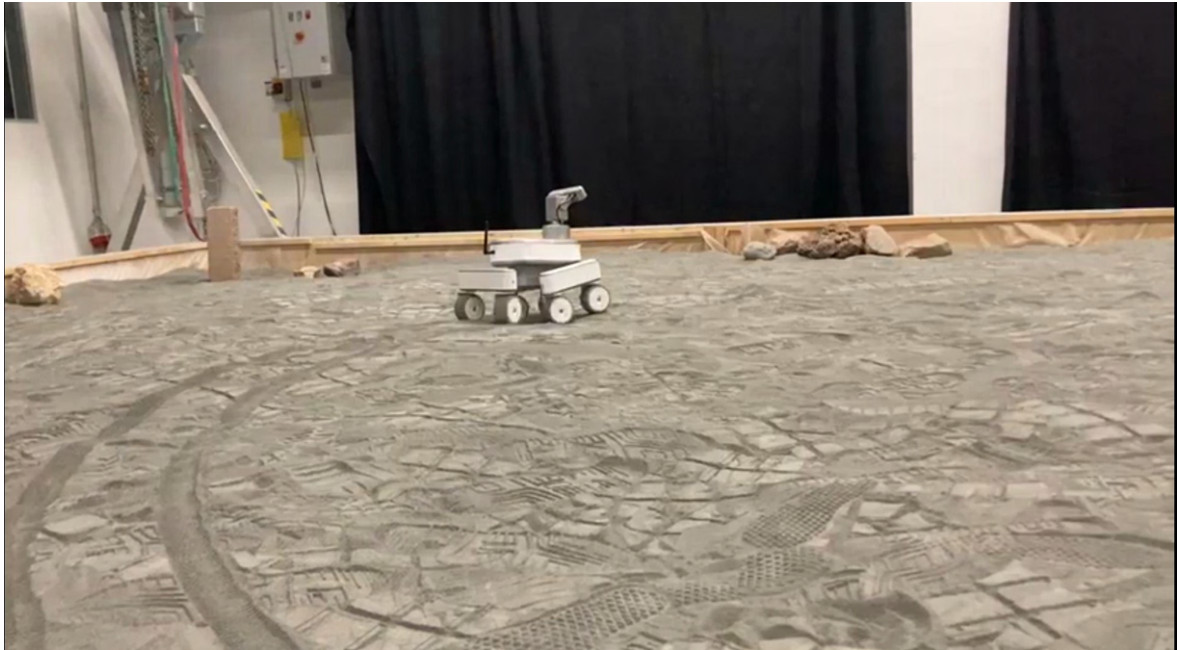


Figure 6.3: DLR test area

adapted. Which axis for driving and camera movement is the user-friendliest option has to be evaluated through a user case study (7.1 User Case Study on Vulcano). In conclusion, the rover can be commanded with the control app. The controlling over the spacemouse requires some learning but after some testing, the controlling is getting more intuitive and a parkour can be completed. Aspects like the favored controlling axes, sensibility of the controller and preciseness have to be further researched on (7.1 User Case Study on Vulcano).

## 6.4 Human-Machine Interface testing with the camera stream on Vulcano

The first Human-Machine Interface tests will purely focus on combining the spacemouse with the camera stream. That is why, the GStreamer GUI with only the camera stream is utilized. The experiments were done on the Aeolian island Vulcano. Vulcano has analogies to Mars [26], which makes it an ideal environment for testing the rover. The island consists of three main regions called la Fossa, il Piano and Vulcanello [27]. Through explorations, three different testing spots were chosen, two on Vulcanello and one on la Fossa. Each location consisted of a different terrain. On la Fossa there are small stones almost like a gravel consistency, on Vulcanello is a sandy and rocky surroundings. In figure 2.1 the two rovers can be seen on la Fossa.

During the previous sections, the camera options and control app were separately evaluated and have to be now evaluated in combination. Each got tested in every testing spot. The different terrains worked all for the rover and the results didn't significantly differ. In the

following, a summary of all tests will be given.

The first option to elaborate on is VNC. As already mentioned in 6.2.2 there is a significant latency. This complicates the control and can cause accidents as the delay plus the reaction time of the user can be enough for crashing the rover to an obstacle. During the testing, the rover had to be manually saved as the reaction was not fast enough to save the rover from an obstacle. However, it could be easily zoomed into the camera which helps to recognize the surroundings. This can be seen in figure 6.4. There weren't any app crashes, which makes the app reliable in this aspect. During long-range testing, the delay increased to the point of almost impossible control. Furthermore, VNC could be used to start all processes so that no additional laptop was needed. This saved time during the experiments.



Figure 6.4: VNC on Vulcano

The second option to analyze is ROS-mobile. The delay was between 1-2 seconds which made it suitable for a teleoperation. A disadvantage was that for applying different settings switching between ROS-mobile and the control app is needed. Not only did this take extra time, but also ROS-mobile disconnected almost every time from the rover. This is not ideal for a real-time operation, especially with dynamic obstacles accidents could happen. On top of that, irregular peaks of significantly increased delays happened. The control app didn't have these delays which resulted in a difference in controlling and camera view. During these irregularities, the teleoperation had to be paused as the risk of an accident was too high. Excluding these irregular delays, the Human-Machine Interface worked well if no changes in the control app were needed. In figure 6.5 a teleoperation at Vulcanello is shown.

The last experiments consisted of using GStreamer. As already experienced in the lab, the delay was around 1-2 seconds with no irregularities like ROS-mobile. The control app is integrated into the GStreamer GUI which resulted in a convenient use. No time was lost as no switching between apps was needed and fast changes could be applied directly. Furthermore,



Figure 6.5: ROS-mobile at Vulcanello

there was no noticeable difference between using the spacemouse and the resulting changes in the camera stream. In long-range experiments, the quality of the image reduced, whereas the latency stayed the same. This is shown in figure 6.6. Although the quality reduced a teleoperation was still possible. The same start errors as already mentioned in section 6.2.1 happened. These appeared irregularly but after restarting the app, the error got solved.

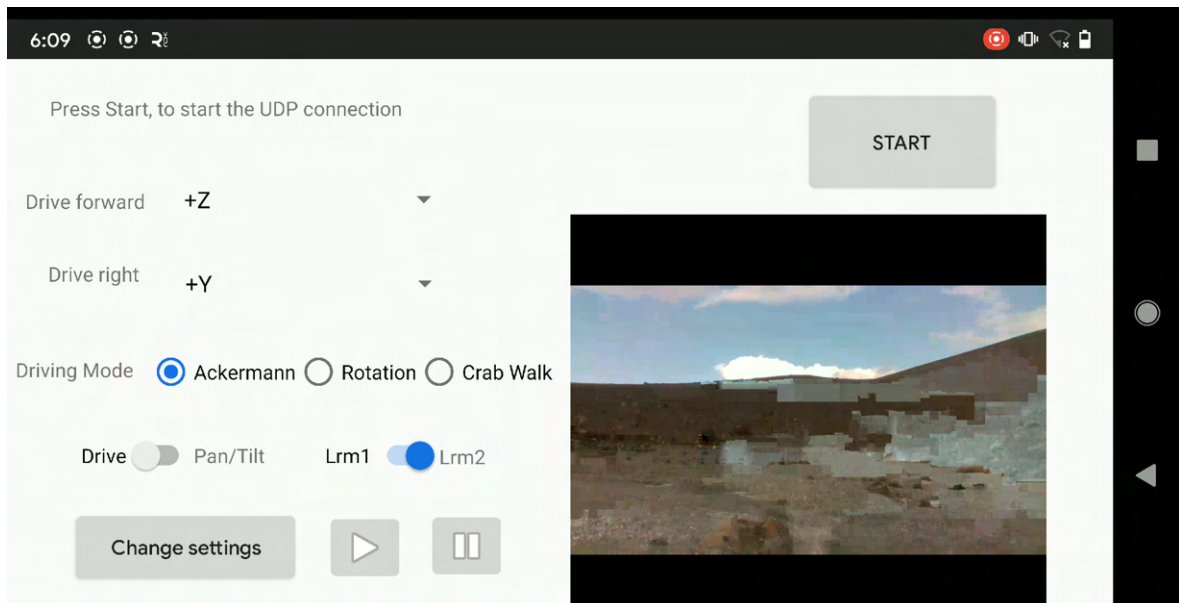


Figure 6.6: GStreamer reducing the quality of the image when the WiFi connection is slow

An interesting finding that has emerged is that when another device like a laptop is via VNC connected to the rover, the delay is increasing. Unexpectedly in this scenario, a difference between ROS-mobile and GStreamer is noticeable. The latency of ROS-mobile raise to around 3-5 seconds. In comparison, GStreamer had a delay of 2-3 seconds. This is a significant difference which has to be considered. Another finding was, that the VNC app has less delay than the VNC running on the computer. The reason for that could be the smaller resolution on the phone.

In summary, GStreamer outperformed ROS-mobile and VNC as the observed latency was the lowest. During the experiments, the most important factor was the delay as this is significant for a teleoperation. The VNC app was used for setting up the whole rover as it is faster than with the laptop and only the smartphone as a device is needed.

## 6.5 Map testing

In order to expand the Human-Machine Interface a created danger map should be shown. Each app is able to show the created map from ROS. In the following it will be tested if the map is displayed correctly, if it is suitable integrated and if it is robust. The first option to test is VNC. The map occupies the majority of the screen and can also be increased. This enables a better view of the obstacles and ensures that details won't be missed. On the other hand, the camera view is too small for a teleoperation. When zooming into the camera view, the majority of the map is not in the range of view anymore. In order for the user to utilize the map a constant swiping between camera and map is needed. Furthermore, different functionalities of the map can be made visible, which are also then illustrated in the camera stream. The map is also shown in the camera stream. The different options are displayed the figure 6.7 and 6.8. This feature is not possible in ROS-mobile and in GStreamer. Furthermore, the position of the rover is shown in the map, which makes it easier to read, whereas the standard setting of a tilted map complicates it. The delay is significant as it is over 5 seconds, which makes it less reliable.

The second option to elaborate on is ROS-mobile. When adding the map to the app, it can be noticed that the resolution is fixed and differs each time. During experiments, multiple maps were created, as shown in figure 6.9 and 6.10. It can be seen that not all maps had a fitting resolution for the smartphone. At some, the map was barely recognizable. Furthermore, it has to be analyzed if the map provides advantages for the user. The amount of observed map failures was significantly high to the point of the map being not a reliable and usable part of the ROS App. Approximately only a third of the time was the map displayed at all. Restarting the app and re-adding the widget of the map helped to solve the error the majority of the time but didn't prevent the error from happening again.

The last analyzed method was GStreamer. In the GStreamer GUI, it can be changed between the camera stream and the map. An example map is shown in figure 6.11.

In the example, it can be seen that even though the obstacles are visibly colored in black, the distances from the rover to the obstacles have to be estimated. For this, prior knowledge about the map in terms of distances and position of the rover is needed. Through a series of tests, the

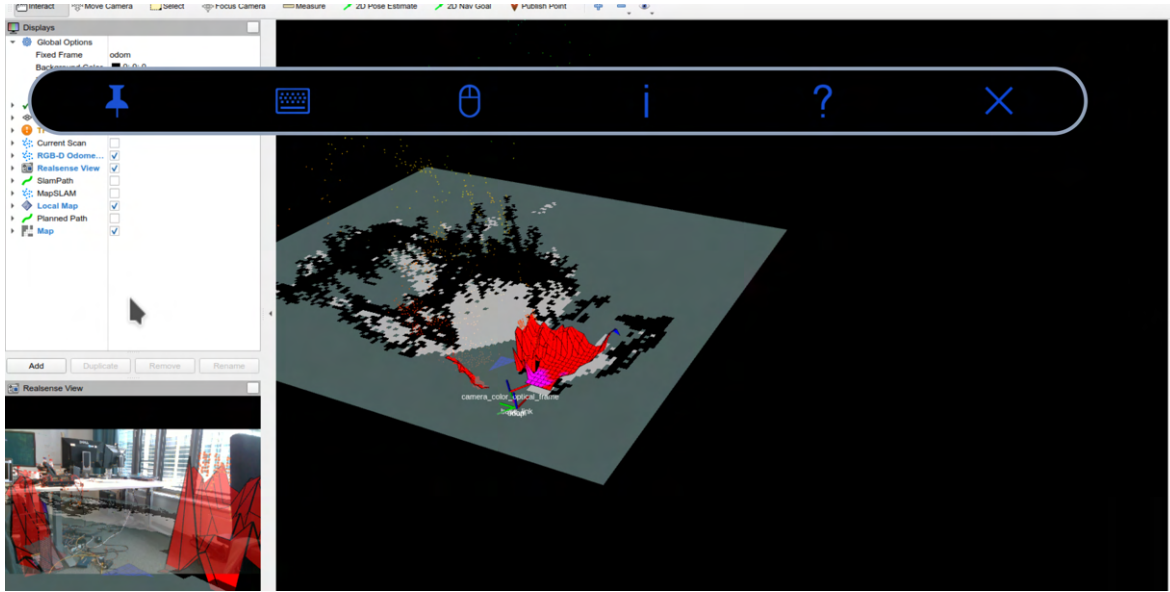


Figure 6.7: VNC map with local map on

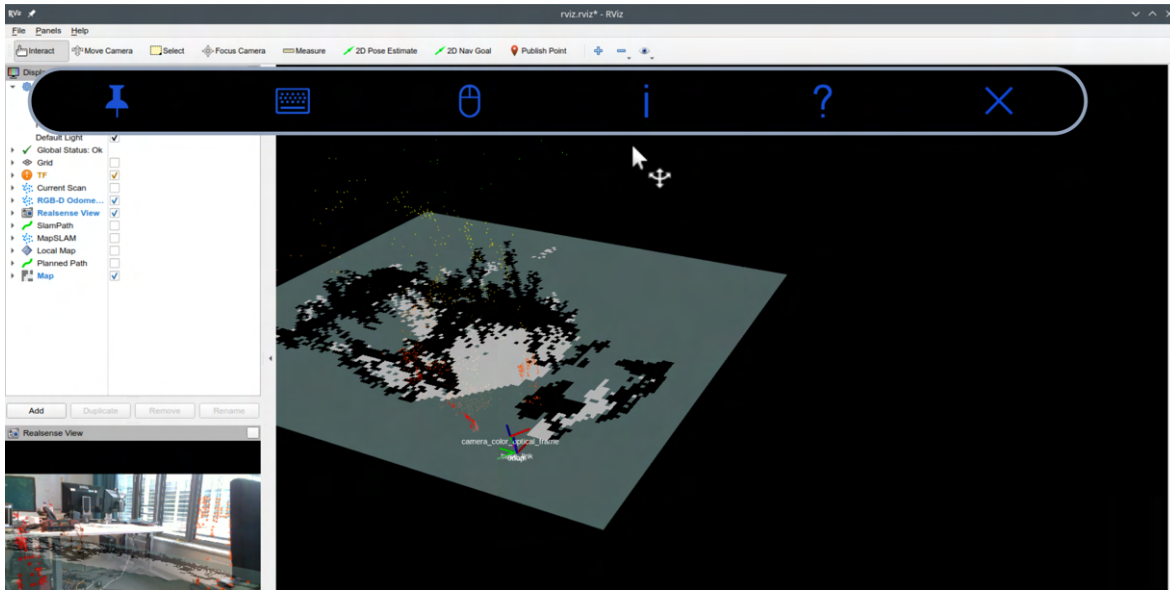


Figure 6.8: VNC map with no extra features

findings were that the map topic does not have a regular publish rate, which means that the sending of the GStreamer pipeline has the same irregularity. There is solely a topic message if there is an update in the map. This happens if the environment alters so that unknown parts get discovered or if known parts have changed. In experiments, it was discovered that this only occurs if the rover is moving slowly. Dynamic obstacles and fast movements of the robot are not always taken into consideration. GStreamer is sending pixel by pixel and is



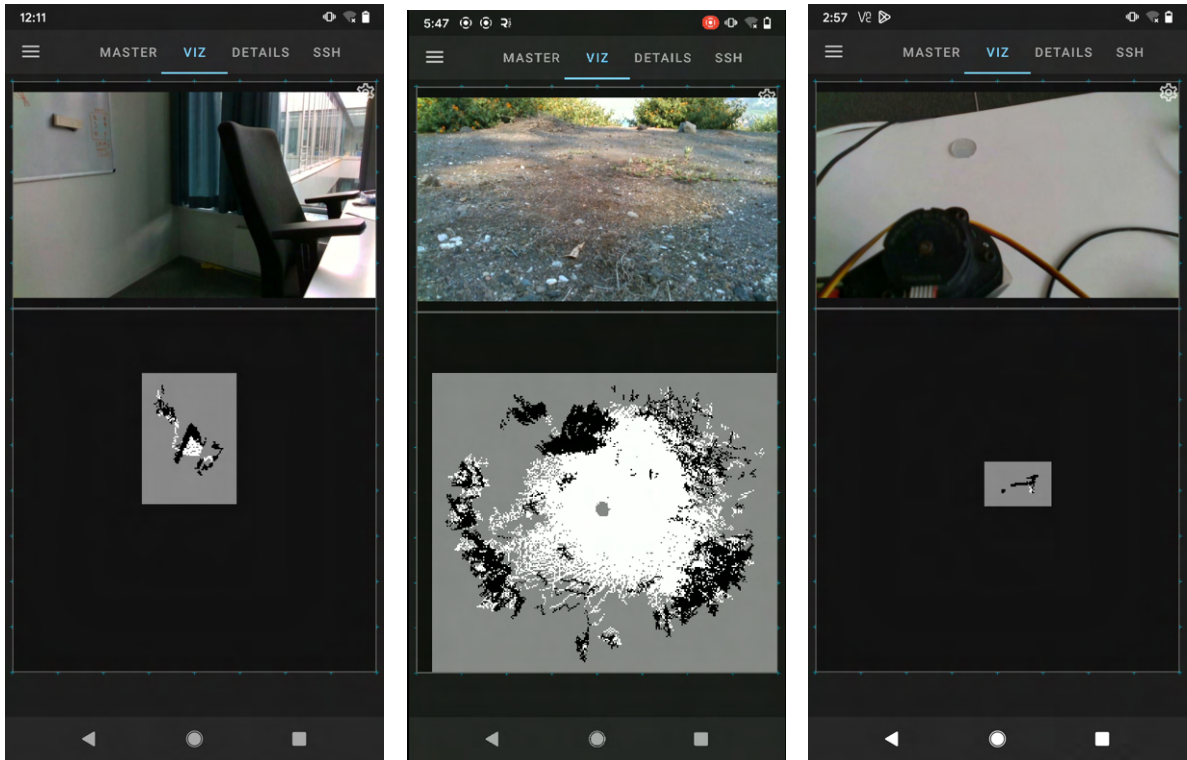


Figure 6.9: Pictures of different map scenarios with ROS-mobile

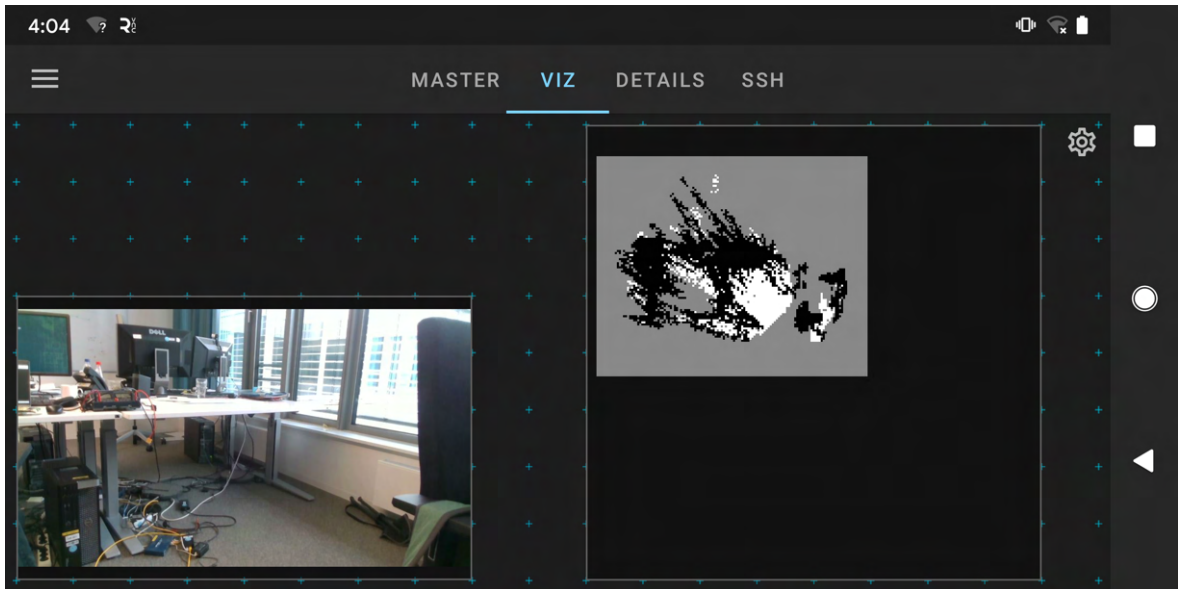


Figure 6.10: ROS map in the lab

not checking if a picture is complete. In combination with the updating irregularities, the

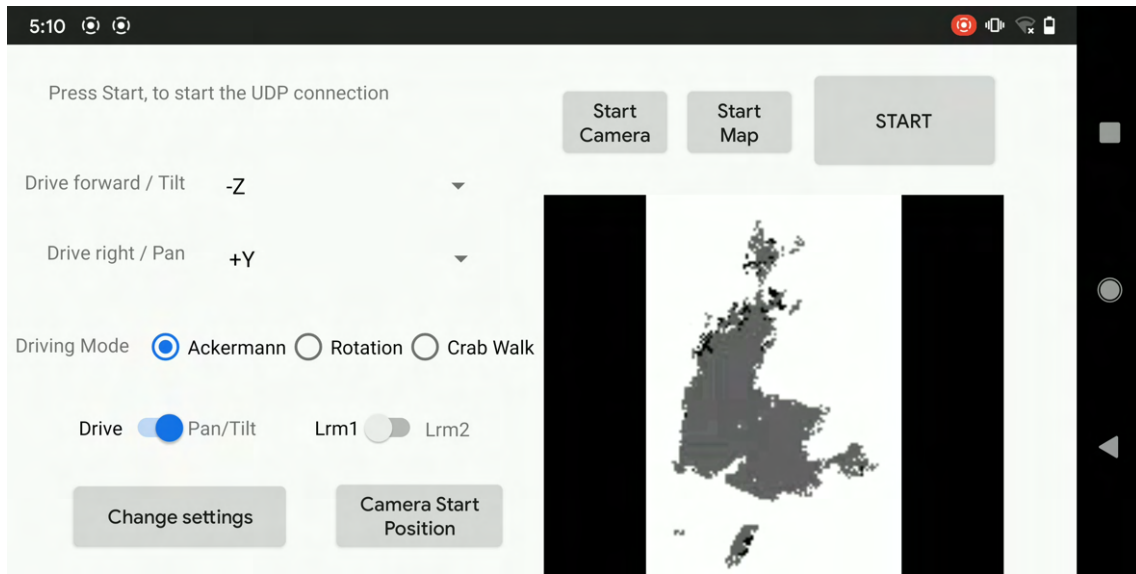


Figure 6.11: Working GStreamer map

transferring to the smartphone can lead to faulty maps. Two erroneous maps are illustrated in figure 6.12 and 6.13. After triggering some updates the errors disappeared.

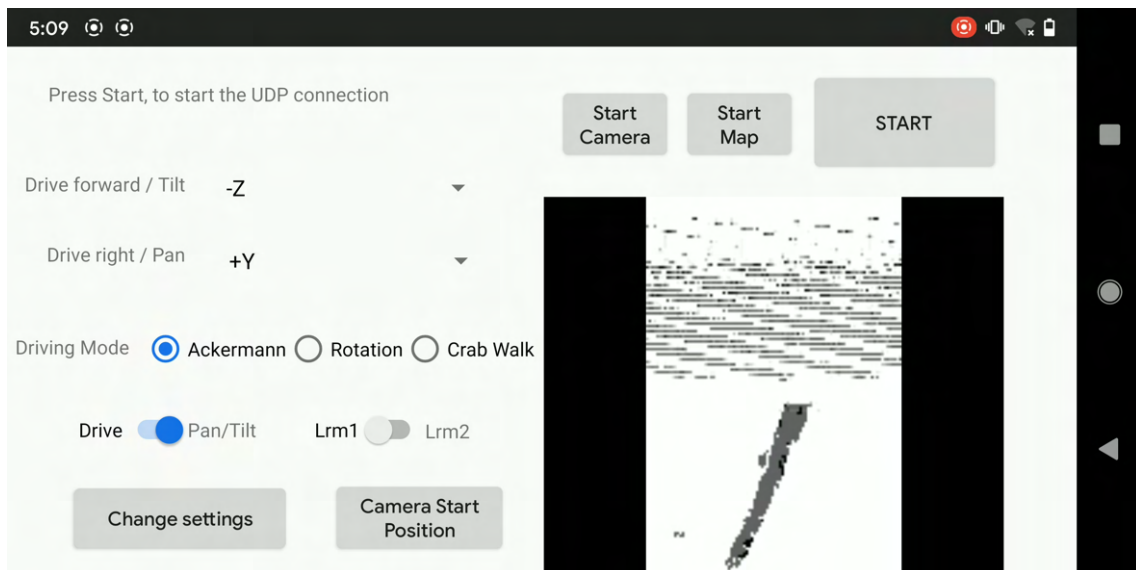


Figure 6.12: Error in the GStreamer map 1

Summarized all options had problems with displaying the already existing maps. VNC is the only one able to display the robot's position in the map, whereas the top-down view of ROS-mobile and GStreamer ensures better readability. The downside of ROS-mobile is its unreliability in displaying the map and illustrating it at a suitable size. With the irregular

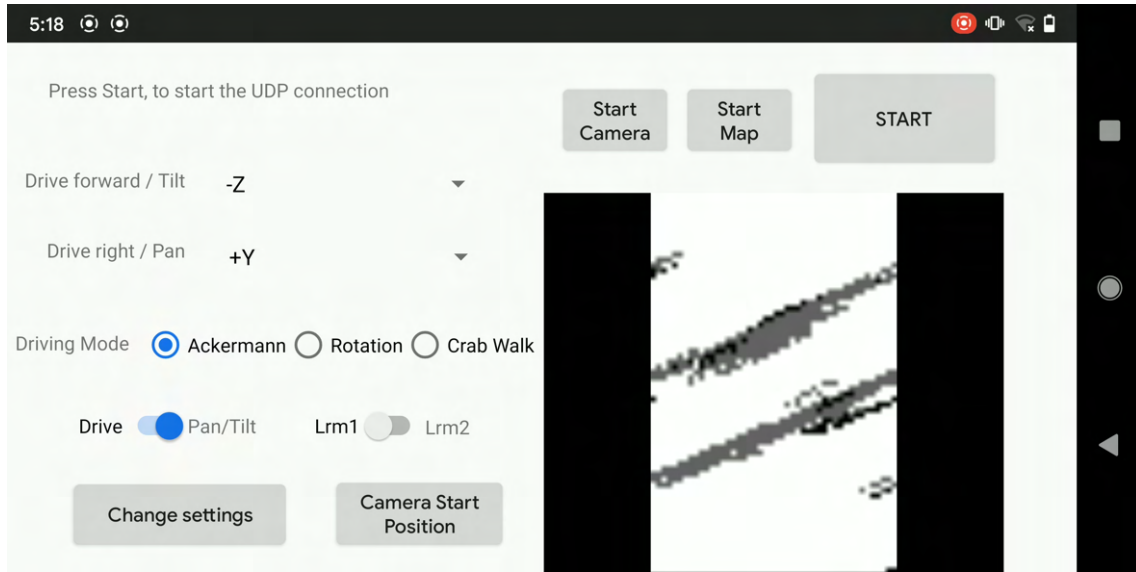


Figure 6.13: Error in the GStreamer map 2

updates, GStreamer is running into transferring mistakes. Whereas VNC showed a significant delay in displaying the already irregular updates. Overall, the maps of all three options can, despite these problems, be utilized for a teleoperation.



## 7 User Case Studies

### 7.1 Camera User Case Study on Vulcano

Within the scope of the trip to Vulcano a user case study to test the Human-Machine Interface and each implemented concept was performed. During testing on Vulcano, a wheel problem occurred that couldn't be repaired on the field. This led to a changed user case study that does not evaluate the whole Human-Machine Interface but to one which purely tests controlling the pan/tilt movement of the camera with the spacemouse. Within this framework, six participants completed the user case study.

The first part of the study was to control the camera movement over the spacemouse in the controlling app to get the user comfortable with the spacemouse and to assess how intuitive the controlling part is. Furthermore, the first feedback on the GUI was given. The participants chose their own time on how long to test as the task was to get to know the system, to figure out the control and to get enough impressions to give a first feedback. A short introduction guidance and help with questions were provided. On average the users took 5-10 minutes for this task. The feedback consisted of a rating on how easy controlling the spacemouse after the first tests felt and on the setup and use of the control app GUI. On a range from 0 to 10, where 0 reflects unusable and 10 resembles easy usage. The spacemouse controlling got an average of 3.8, which indicates that the users had problems with this controller. In order to have an assessment if the spacemouse is in general rather difficult to use or if it is solely the adjustment period, the same question will be asked after the complete user case study. Furthermore, the intuitiveness and setup of the app in regards of layout and provided functions were rated. In terms of intuitiveness, the criteria were the layout, default settings and functionalities. How easy and quick it is to connect to the rover and set the desired functions was evaluated in the setup. The measurement scale covered 0 to 10, where 0 reflects a counter-intuitive / incomprehensible setup and 10 resembles a very intuitive / easy to use setup. The average rating was 6.6 for intuitiveness and 6 for setup. One has to mention that the app crashed during these first tests an average of three times, which could have influenced the results as the setup had to be done multiple times per person. On top of that, the chosen axis settings were tracked. The observations showed that the default options for driving weren't as fitting to the camera movement and none of the participants decided to use them. Around 1/3 of the participants chose to use the X axis, whereas the majority the Z axis with a changed direction favored (-Z instead of +Z). In conclusion, the evaluation indicates that the control app is user-friendly but there is room for improvement. Suggested feedback was to adjust the text labels for more comprehensibility, changing the default options and reducing the sensibility of the spacemouse. Furthermore, the function of the camera moving to the middle position when releasing the spacemouse wasn't evaluated as practical. As an

improvement, a graphical button for this feature was proposed.

In the second part of the study, the three different options for teleoperating with the camera stream were tested. For this, the task for the participants was to control the camera movements with the spacemouse and to evaluate with each option the setup and delay. Furthermore, they were asked how comfortable they felt controlling the robot with the apps. After each option, feedback was given. Later adaptations in the answers were possible. The order of the testing was randomized with each participant. In figure 7.2 the gathered feedback is illustrated. Displayed is the calculated average of the received answers. In the case of intuitiveness, setup and comfortableness a higher number is the better option as 10 relates to very intuitive, easy setup and very comfortable use. The opposite is with the delay, the higher the number the more lag got noticed by the user. The number of complete app failures was tracked. In contrast to the lab testing, ROS-mobile did not fail a single time during the user case study. Whereas VNC failed for the first time during the user case study. Solely GStreamer behaved as previously observed. In general, it can be seen that in 4 out of 5 evaluated criteria GStreamer performed the best. It has to be considered, that the difference in intuitiveness and setup is not significant and purely shows a small tendency. In terms of setting up the whole system there was already some prior help given as not everything is designed to be done by the users. A variance in the answers is in delay and comfortableness noticeable. In these GStreamer is performing better than the other options. In terms of delay VNC and ROS-mobile are not significantly different which is surprising. In previous tests ROS-mobile was mostly working with no significant increased delay compared to GStreamer but there were some irregular times when the delay was unusually high. As the participants tested each option for an average of 10 minutes, there is the possibility that most of them experienced at least one of these irregularities and evaluated based on this the general delay high. Surprisingly, the least comfortable option was ROS-mobile.

After testing all options, the participants were asked their preferred option. The results are that 4 chose GStreamer, 1 ROS-mobile and 1 VNC (figure 7.1). Combined with the other part of the evaluation, there is an overall preference for GStreamer even though it had the most app failures. Unexpected VNC got in the combination of figure 7.1 and 7.2 a slightly better feedback than ROS-mobile. This could have been because of the unexpected amount of high delays in ROS-mobile. The last question was to evaluate the controlling of the spacemouse again to check if the controlling difficulties are reduced to the adjustment period or a general problem. After the control app testing, the spacemouse got an average of 3.8 on a scale from 0 to 10, where 10 resembles an easy usage. In the end, the spacemouse got rated a 4.2 which is slightly better but not significantly higher to prove that the controlling difficulties are only occurring with adjusting to the new system. As feedback, most participants suggested a less sensitive joystick or having less possible degrees of freedom, so that the joystick can't physically move in some directions. Nevertheless, all participants agreed that they would prefer a normal joystick over the spacemouse.

In conclusion, the user case study delivered the results that there is a clear preference for the camera option. GStreamer outweighs significantly ROS-mobile and VNC in regard of noticeable delay and comfortable delay. A negative aspect of GStreamer was the failure rate

which has to be investigated. Furthermore, the spacemouse controlling was not ideal for the users. It has to be further tested if adaptations change the output. On top of that new research can be done on improving the control app. For that, a new user case study with the implemented feedback could lead to new findings.



Figure 7.1: Evaluation of the preferred option

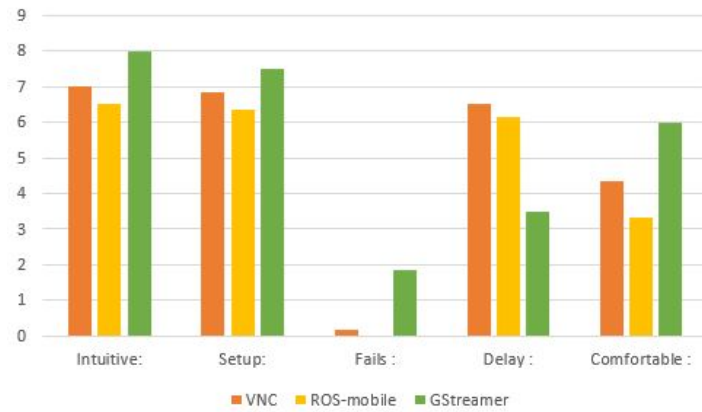


Figure 7.2: Evaluation of the three options

## 7.2 Human Machine Interface User Case Study in the DLR lab

After repairing the rover, the Human Machine Interface User Case Study with eight participants took place in the DLR lab. The user case study on Vulcano (section 7.1) evaluated the camera and led to three main changes. These are that a new button for resetting the camera position is added so that it won't automatically reset, the camera movement got slowed down in order to have a more steady and understandable control and -Z got set as the default drive forward / tilt axis. With this user case study, it will be seen if these modifications improved the overall feedback.

The goal of the Human Machine Interface User Case Study was to teleoperate the rover through a small parkour. This parkour consisted of driving an 8, which is shown in figure 7.3. The first task was to navigate the rover purely with the control app while seeing the rover move. This was a test round in order to get to know the control of the spacemouse and to get a first evaluation. The questions were the same as in section 7.1. Out of the range from 0 (hard to control) to 10 (easy to control) the spacemouse scored on average 6.125, which indicates an intermediate controlling. One has to mention that the spacemouse differs from known controllers and might take some adjustment time. However, the same question was answered after finishing the user case study, which approximately took one hour and the value did not significantly change. This may give rise to the assumption that the spacemouse is not as intuitive as known controllers or might need more training time to get fully comfortable with the usage. One feedback was that the hardware has for turning right less resistance as for



Figure 7.3: Teleoperation test route

turning left. This could be further improved by either changing the hardware accordingly or changing the sensitivity for turning left in the code so that fewer control inputs are needed for the same movement. In terms of the GUI, intuitiveness scored on average rated 8.125 and the setup 8.375. The range and meaning of intuitiveness and setup are explained in section 7.1. These values indicate that the layout and features were suitable and comprehensible. For the setup, there was a variation in the feedback ranging from 6 to 10. The reason for this is the preference of the control axes. The default axes were - Z for driving forward/tilt and + Y for turning right/pan. All of the participants preferred the + Y axis for turning right and pan. For driving forward 5 out of 8 chose + Z and 3 stayed with the default setting of - Z. Whereas in the tilt axis, it was equally distributed between + Z and - Z. A graphic illustration of these distributions is shown in figure 7.4. The participants who favored the default settings rated the setup better than those who chose a different one. In general, the average score of 8.375 for the setup indicates that the default settings and how the connection to the rover was established were well received. In section 8 a comparison between the two user case studies will be made in order to evaluate if the changed layout and default settings improved the results.

After getting to know the control of the spacemouse and completing the first test round of the parkour, the teleoperation started. For this, the users weren't allowed to see the rover anymore and had the task to complete the parkour solely with the camera stream. For this, the three different options were tested in a random order as with more teleoperations the user got more confident and better in controlling and it shouldn't affect the results. As the result of the first teleoperation, independently of the option, was in general rated the worst, a second round was done with each option. The first round can be seen as a test round as it was the first time teleoperating with the spacemouse controller and is not as expressive as the second

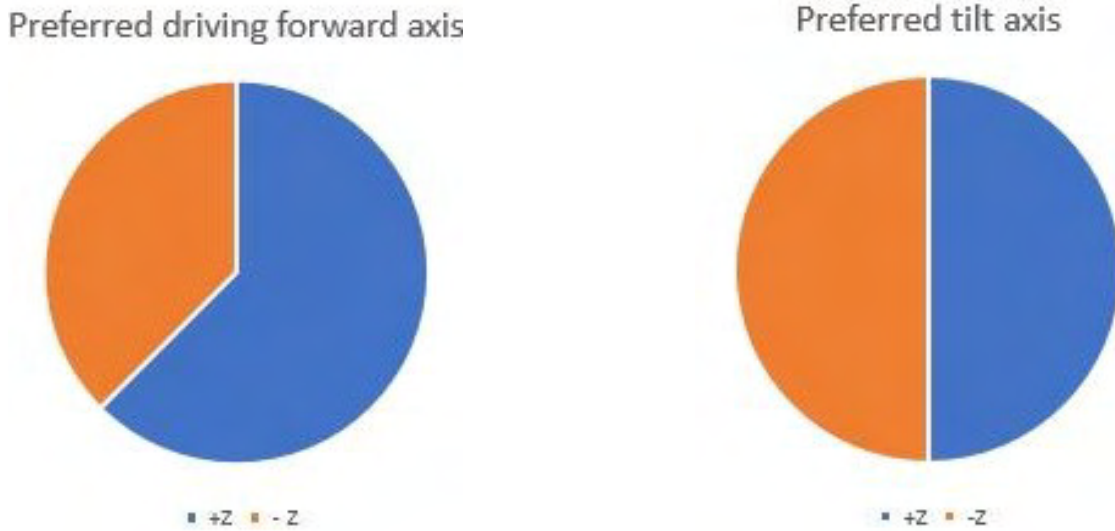


Figure 7.4: Evaluation of the preferred axes settings

round. Furthermore, the camera delay can differ due to the UDP connection. As there is the possibility of a bad connection due to some data losses one round is not as meaningful as two. With two rounds the likelihood of having connection problems in the same option can be reduced and the overall results are more accurate for the general case. For the teleoperation, the same criteria as in the user case study on Vulcano (section 7.1) were evaluated. In figure 7.5 the results are displayed. The score of the intuitiveness and setup did not significantly differ between all options. One has to mention, that GStreamer got a slightly better ranking than the other two options. In general, the distribution didn't significantly differ between round one and two except for the camera delay. In round one ROS-mobile and GStreamer had the same average delay and in round two users experienced slightly less latency with GStreamer. The values of the two rounds however didn't differ significantly. The results are: VNC with 7.125 and 7, ROS-mobile with 4.875 and 4.5 and GStreamer with 4.875 and 3.75. As the difference between ROS-mobile and GStreamer is less than 1, the delay can be seen as approximately equal between both options. However, there is a significant difference to VNC. The values are showing an experienced high delay. Furthermore, in the total of 16 test rounds, 7 were not completed. The participants weren't able to finish the parkour due to multiple complete failures of the app. In all the failures the camera image stopped and couldn't be fixed through restarting. However, some users didn't experience any failures even though no changes were made and some user case studies were done after each other. This is on one hand showing the unreliability of VNC and on the other hand the significant delay when it is running. With these results, it has to be discussed if VNC is suitable for a teleoperation. In figure 7.5, it can be seen that in round 2 there is also GStreamer with one failure. In 16 rounds one unexpected fail of GStreamer happened. As it is only one and was solved with restarting, it is not significant and won't be considered in further evaluation. ROS-mobile

performed without any failures. Furthermore, after each option test the question on how comfortable the user is with using this method on a range from 0 (not comfortable at all) to 10 (very comfortable) was asked. It can be seen that in both rounds but especially in round 2 GStreamer outperformed the other options. VNC had both rounds a score under 5, which raises the assumption that it is not the ideal option for a teleoperation. This assumption will be further discussed in section 8. In general, GStreamer and ROS-mobile performed similarly with a slight tendency for GStreamer being the preferred option. In order to prove which method is overall the best approach for the Human-Machine Interface, the users had to choose their preferred option after the finish of all the tests. Out of the 8 participants 2 selected ROS-mobile. The remaining users agreed on GStreamer being the preferred option, which is a clear majority. This is illustrated in figure 7.6.

Furthermore, there were some suggestions for improving the control app which included having graphical feedback on how the camera is located. This could be realized either in the form of a text or visual in the camera stream. Also, the start position of the camera has to vary between the two different rovers as they differ in the height position of the camera. In further tests, a more suitable start position of the camera could be researched. In these tests, both rovers were utilized and especially with LRM2, every user changed the camera position. Both rovers could be utilized for this user case study as the software is identical in both. One has to mention that with both rovers hardware problems occurred which made it impossible to only focus on one rover for the scope of this user case study.

In the next section (8) the results of both user case studies in combination with the testing will be evaluated.

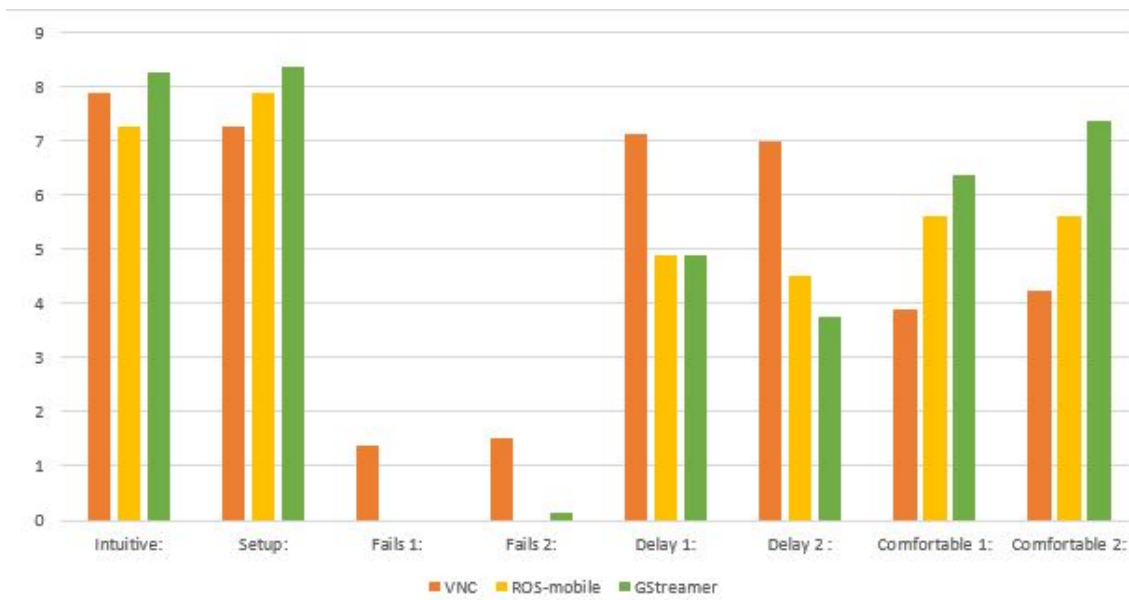


Figure 7.5: Evaluation of the three options

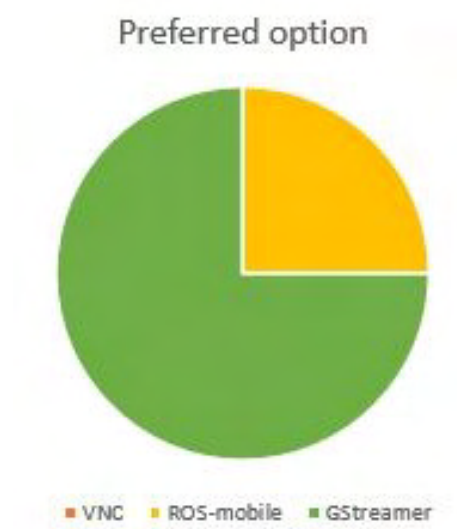


Figure 7.6: Preferred option after the Human-Machine Interface

## 8 Evaluation

In the scope of this thesis, a Human-Machine Interface for teleoperating a rover over a mobile device should be created and researched. This includes controlling the robot over the spacemouse and having the camera stream and map displayed on the smartphone. For this, three different methods have been implemented, tested and rated over two user case studies. In this section, it has to be evaluated which option is the overall most suitable one, which control app GUI features and control input translation from the spacemouse worked the best. First, the controlling part consists of the spacemouse and the control app GUI. The spacemouse is not a common controller. Through the user case studies, it has been discovered that a training time is required before the users fully understand the control. In the first user case study, only the camera was moved and in the second a full teleoperation was done. After the first one, the sensitivity was changed so that more control inputs were needed for the same movement. In both the question on how easy the controlling of the rover with the spacemouse on a scale from 0 (unusable) to 10 (easy usage) got asked. The more sensitive controller got an average of 3.8 rating whereas the less sensitive one scored an average of 6.128. This shows that the users preferred less sensitive controllers. In order to further improve the spacemouse, the sensitivity of the left and right turns have to be equal. Until now the hardware of the spacemouse has a different hardness degree which causes a bigger required force on the left than on the right turn.

In the two user case studies the GUIs of the control apps differed slightly. The second one consisted with an extra button for setting the camera back to its start position instead of having it automatically done after releasing the spacemouse. Furthermore, the default driving forward / tilt axis was changed from + Z to - Z. With these two changes the intuitiveness and setup scores improved from 6.6 and 6 to 8.125 and 8.375. This could be the result of the modifications. During the first user case study there was feedback on the automatically reset and the default settings, whereas there wasn't any on the second. However, not every user decided on using the default control axes during the second user case study. This shows that the changed default settings are not impacting the results. One has to mention that during the first user case study the app crashed a significant amount with each participant whereas during the second time, there weren't any. This shows, that the added waiting time between sending control inputs solved the error. It could be that the improved scores result from a combination of the added button for changing the camera back to its start position and not having any app failures.

Furthermore, there was research on the preferred control axes in order to improve the default settings. In the second user case study for driving forward there was a slight tendency for choosing +Z could be seen. However, the amount of -Z users is not to be neglected (figure 7.4). As the first user case study only consisted of the camera movement, there couldn't be



any data gathered on the driving forward axis. For the tilt axis, the gathered data of the combined user case studies can be utilized. A graphical visualization of the results is shown in figure 8.1. It can be seen that there are three different chosen axes for this movement. Out of 16 participants 9 chose -Z, 4 preferred +Z and 3 selected +X. As there is only a slight majority with around 56% for -Z, there are no clear preferred axes. After the first user case study the default axis was changed to -Z as no participant chose +Z. The second user case study showed that the selected axis is based on individual preferences as also +Z was chosen even though it was not the default setting. However, all participants of both user case studies agreed on having the drive right/pan axis set to +Y. Overall, it is shown that there are no clear preferred axes for driving forward and tilt. This leads to the reason why the GUI is required to have the option for adapting the axes to personal preferences.

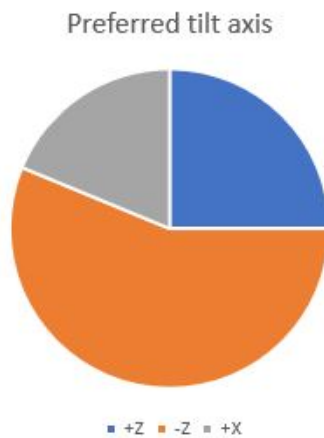


Figure 8.1: Evaluation of the preferred tilt axis of both user case studies

Furthermore, the user case studies gave the insides that most participants change the settings on a regular basis. This indicates, that the layout requires the change settings option easily accessible and shows that the layout of the designed GUI is suitable for the user.

The Human-Machine Interface should also display the camera stream of the rover and an already created danger map of the surroundings. For this, three different options were implemented and tested. Each was assessed on the GUI, camera delay, map and combination with the spacemouse.

In terms of the GUI VNC has the largest camera window and multiple features to add like the robot's position in the map, which is not possible with the other options. This could be on the one hand useful but on the other hand, lead to confusion for non-experienced users. For combining VNC with the spacemouse a separate app is needed. In ROS-mobile there was only the camera stream and map shown, the size of the map was fixed. Furthermore, the map had a significant amount of failures as it was not displayed the majority of the time. On the other hand, there was not much to change in the GUI so there wasn't the possibility of misunderstandings from the users. As well as for VNC a separate app for the spacemouse controlling is required. GStreamer is the only option which has the controlling integrated.

The layout is fixed and can't be changed by the user. In both user case studies, GStreamer received slightly better scores than the other two options.

The experienced delay differed only for ROS-mobile in both studies. Also during testing some irregular peaks were experienced which could have also happened during some user case studies. This could have led to an overall higher experienced delay in the first user case study. Furthermore, the comfortable score was lower for ROS-mobile in the first user case study which could have been also the reason for the increased delay. ROS-mobile and GStreamer have different approaches for handling a bad connection. ROS-mobile increases the delay but keeps a high-quality image whereas GStreamer reduces the quality of the image in order to keep the delay as low as possible. The two user case studies didn't provide an insight on which option they preferred as both options received approximately an equal score on delay. The preferences on this can be researched in future work.

Another noticeable difference between the results of the user case studies is the fail rate of VNC and GStreamer. While with GStreamer the fails reduced to almost 0, VNC had a significant increase of crashes. The reason for this is, that GStreamer was coupled with the control app and by solving the issue with the control app the GStreamer failures also disappeared. So in conclusion, there was no failure with GStreamer in the first user case study. The app crashes were caused by the control app. Whereas in the VNC app, there was no change in the app in between the user case studies. The significantly increased amount of failures could have happened due to a higher usage time of the app. A lower usage time has a lower possibility of failures. This could be the reason for the significant difference in the user case studies. The other values behaved similarly in both user case studies.

For analyzing the preferred option of the user, the results of the two user case studies were combined and illustrated in figure 8.2. In this figure, it can be seen that there is a clear majority (around 71.4%) favoring GStreamer. Around 21.4 % preferred ROS-mobile and about 7.2% chose VNC as their favorite option. One has to mention that the 7.2% is in absolute numbers 1 of the 14 participants.



Figure 8.2: Evaluation of the preferred option of the both user case studies

In summary, VNC didn't perform as well as the other two options as it has overall the highest delay and failure rate. The fact that the control app is not integrated into the VNC app is a disadvantage. Only one out of 16 test person preferred this option. One has to mention that during the second user case study 43.7% of the test rounds couldn't be completed due to high delay and reoccurring app crashes. This proves the unreliability of VNC and that it is not as suitable for a teleoperation as the other options. Nevertheless, it has to be mentioned that the app showed different benefits as it can be used for starting all processes on the rover before having the teleoperation so that no other device is needed. As for this, the delay is playing a less important role as for direct controlling the rover, it is suitable for this use case. ROS-mobile had an overall good performance. There were some delay irregularities which are not ideal for a teleoperation. The average delay was approximately the same with GStreamer and all participants could successfully control the rover through a small parkour despite the delay. About 21.4% of the users preferred this option which leads to the result that ROS-mobile can be a suitable Human-Machine Interface for teleoperating the rover. However, the fact that the control app is not integrated into the same app makes it less convenient. Another disadvantage are the issues with the map. This leads to the conclusion that it is not ideal for this use case. Overall, GStreamer outperformed ROS-mobile and VNC for this use case. The experienced delay of GStreamer was still suitable for the teleoperation and was significantly less than for VNC. In combination with the integration of GStreamer into the control app led to being the favored option of the users. This proves that GStreamer in combination with the control app is a suitable Human-Machine Interface for teleoperating a rover.

## 9 Conclusion

The goal of this thesis was to realize a Human-Machine Interface for teleoperating a rover over a smartphone with a spacemouse controller. This consisted of translating the control inputs in order to utilize all functionalities of the rover, which include three different driving modes and camera movement. Furthermore, the camera stream and an already existing map are displayed on the smartphone screen. Through related work, three different suitable concepts for transferring the media were found and implemented. After combining the controlling with the three different camera options, there has been research on the most suitable GUI for the user, the delay, which axes are preferred by the users and the controlling over the spacemouse.

The evaluated options were VNC, ROS-mobile and GStreamer. Through tests and user case studies, it was shown that GStreamer and ROS-mobile are suitable for teleoperating a rover in combination with the spacemouse by solely using the camera stream. In the case of the map, there were in both options issues occurring which have to be solved in future work. Due to high delays and failures, VNC is not considered as a suitable option. Overall, GStreamer was the preferred option by the majority of the participants of the user case study due to its low delay and integration into the control app.

In further work, the ideal start camera position, how to have the same sensitivity in the left and right turn and what kind of map is suitable for a teleoperation could be researched on. In order to have a suitable map, the transmission errors have to be solved. As extensions for this Human-Machine Interface, the camera position could be displayed either in text format or directly in the camera stream for a better understanding on how the image is related to the robot. Also, the position of the robot could be added in the map in order to improve readability. Furthermore, a mix with autonomy could be implemented, this could consist of a button to change directly to autonomy or to have an overlay of teleoperation with autonomy. This could result in having haptic feedback in the controller so that certain movements are not possible in order to avoid crashes.

As a conclusion of this thesis, the Human-Machine Interface with GStreamer combined in a control app is outperforming the other options. Controlling over the spacemouse requires some learning but teleoperations are afterwards possible. Overall, the combination of GStreamer with the controlling over the spacemouse is a suitable solution for teleoperating a rover over a portable device.

## List of Figures

2.1	LRM1(left) and LRM2(right) on the volcano of the Aeolian Island Vulcano . .	3
2.2	Picture of the spacemouse . . . . .	3
2.3	Picture of the smartphone . . . . .	3
2.4	Picture of the controller . . . . .	3
2.5	Overview of the architecture . . . . .	4
2.6	LN manger . . . . .	5
2.7	Simulink model . . . . .	6
2.8	The 3 different driving modes: Ackermann, Rotation, Crab . . . . .	6
2.9	Network architecture for the smartphone connection . . . . .	7
3.1	Structure of the connection from Java to C/ C++ code over JNI [8] . . . . .	8
4.1	Architecture of the general goal . . . . .	10
4.2	VNC Viewer architecture . . . . .	12
4.3	ROS-mobile architecture . . . . .	13
4.4	Architecture option with GStreamer and a Java App . . . . .	14
5.1	Control App GUI . . . . .	15
5.2	Control App Code Logic . . . . .	16
5.3	Final Control App GUI . . . . .	18
5.4	Connection setup for VNC for LRM1 . . . . .	18
5.5	Connection setup for ROS-mobile . . . . .	20
5.6	Camera stream of ROS-mobile . . . . .	20
5.7	Camera and map of ROS-mobile . . . . .	20
5.8	Screenshot of the GStreamer GUI before adding the map . . . . .	21
5.9	Screenshot of the final GStreamer GUI . . . . .	22
6.1	RViz on the smartphone over VNC . . . . .	24
6.2	Increased camera view in VNC . . . . .	24
6.3	DLR test area . . . . .	27
6.4	VNC on Vulcano . . . . .	28
6.5	ROS-mobile at Vulcanello . . . . .	29
6.6	GStreamer reducing the quality of the image when the WiFi connection is slow	29
6.7	VNC map with local map on . . . . .	31
6.8	VNC map with no extra features . . . . .	31
6.9	Pictures of different map scenarios with ROS-mobile . . . . .	32
6.10	ROS map in the lab . . . . .	32

6.11	Working GStreamer map . . . . .	33
6.12	Error in the GStreamer map 1 . . . . .	33
6.13	Error in the GStreamer map 2 . . . . .	34
7.1	Evaluation of the preferred option . . . . .	37
7.2	Evaluation of the three options . . . . .	37
7.3	Teleoperation test route . . . . .	38
7.4	Evaluation of the preferred axes settings . . . . .	39
7.5	Evaluation of the three options . . . . .	40
7.6	Preferred option after the Human-Machine Interface . . . . .	41
8.1	Evaluation of the preferred tilt axis of both user case studies . . . . .	43
8.2	Evaluation of the preferred option of the both user case studies . . . . .	44

# Acronyms

**AR** Augmented Reality. 9, 12

**AUD** Access Unit Delimiter. 20

**DLR** German Aerospace Center. v, 2, 4, 6, 7, 9, 26, 27, 37, 47

**GUI** Graphical User Interface. iii, iv, 1, 4, 11, 15–18, 21–23, 25, 27, 28, 30, 35, 42, 43, 46, 47

**IMU** Inertial Measurement Unit. 2

**IP** Internet Protocol. 16, 17, 19

**JNI** Java Native Interface. 8, 13, 47

**LN** Links and Nodes. 3–5, 11–13, 23

**LRM** Luna Rover Mini. 1, 2, 16–19, 40, 47

**LRM2** Luna Rover Mini 2. 2, 19

**LRU** Lightweight Rover Unit. 1

**NDK** Native Development Kit. 8, 9, 13, 14

**RMC** Robotics and Mechatronics Center. 7

**ROS** Robot Operating System. iii, iv, 1, 3, 4, 9–14, 19, 20, 23, 25, 26, 28, 30, 32, 33, 39, 40, 43–47

**RTP** Real-Time Transport Protocol. 21

**SDK** Software Development Kit. 8, 12, 14

**TCP** Transmission Control Protocol. 11

**UDP** User Datagram Protocol. iii, 1, 11–14, 16, 17, 19, 21, 39

**VNC** Virtual Network Computing. iii, iv, 1, 9–12, 17, 18, 23–26, 28, 30, 31, 33, 34, 36, 39, 40, 43–47

# Bibliography

- [1] J. J. Biesiadecki, P. C. Leger, and M. W. Maimone. "Tradeoffs between directed and autonomous driving on the mars exploration rovers". In: *The International Journal of Robotics Research* 26.1 (2007), pp. 91–104.
- [2] E. S. A. (ESA). EXOMARS MISSION (2022). Mar. 2020. URL: <https://exploration.esa.int/web/mars/-/48088-mission-overview>.
- [3] G. A. C. (DLR). *Lightweight Rover Unit (LRU)*. 2023. URL: [https://www.dlr.de/rm/desktopdefault.aspx/tabid-11431/20129\\_read-47344/](https://www.dlr.de/rm/desktopdefault.aspx/tabid-11431/20129_read-47344/).
- [4] O. S. R. Foundation. "ROS Introduction". In: online, state of 29.07.2020. <https://wiki.ros.org/de/ROS/Introduction>, 2013.
- [5] M. Song, W. Xiong, and X. Fu. "Research on architecture of multimedia and its design based on android". In: *2010 International Conference on Internet Technology and Applications*. IEEE. 2010, pp. 1–4.
- [6] H. Wang, F. Hao, C. Zhu, J. J. Rodrigues, and L. T. Yang. "An android multimedia framework based on gstreamer". In: *International Conference on Green Communications and Networking*. Springer. 2011, pp. 51–62.
- [7] X. Fu, X. Wu, M. Song, and M. Chen. "Research on audio/video codec based on Android". In: *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*. IEEE. 2010, pp. 1–4.
- [8] A. Saraf. *JNI-101 — Introduction to Java Native Interface*. Nov. 2022. URL: <https://medium.com/@sarafanshul/jni-101-introduction-to-java-native-interface-8a1256ca4d8e>.
- [9] N. Vun and Y. Ooi. "Implementation of an Android Phone Based Video Streamer". In: *2010 IEEE/ACM Int'l Conference on Green Computing and Communications Int'l Conference on Cyber, Physical and Social Computing*. 2010, pp. 912–915. DOI: 10.1109/GreenCom-CPSCom.2010.76.
- [10] GStreamer. *GStreamer open source multimedia framework*. Dec. 2022. URL: <https://gstreamer.freedesktop.org/features/index.html>.
- [11] GStreamer. *GStreamer open source multimedia framework*. Dec. 2022. URL: <https://gstreamer.freedesktop.org/documentation/tutorials/android/index.html?gi-language=c>.
- [12] Android Developers. *All Android releases*. Apr. 2023. URL: <https://developer.android.com/about/versions>.



- [13] A. Zea and U. D. Hanebeck. "iviz: A ROS visualization app for mobile devices". In: *Software Impacts* 8 (2021), p. 100057.
- [14] N. Rottmann, N. Studt, F. Ernst, and E. Rueckert. "ROS-Mobile: An Android application for the Robot Operating System". In: *arXiv preprint arXiv:2011.02781* (2020).
- [15] R. A. Kalje and S. Kosbatwar. "Android Based Remote Control of Mobile Devices Using VNC System". In: *International Journal of Computer Science & Information Technolo* (2014).
- [16] C. Parga, X. Li, and W. Yu. "Smartphone-Based Human Machine Interface with Application to Remote Control of Robot Arm". In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. 2013, pp. 2316–2321. doi: 10.1109/SMC.2013.396.
- [17] F. Wieser. "Implementation of a Navigation Solution for an Experimental Rover Platform, Extendable to Multiple Agents". In: 2019.
- [18] A. R. Ortigosa. "Systems Integration with Autonomous Navigation of the Lunar Rover Mini for a Space Demo Mission". In: 2023.
- [19] R. Shorey and B. A. Miller. "The Bluetooth technology: merits and limitations". In: *2000 IEEE International Conference on Personal Wireless Communications. Conference Proceedings (Cat. No. 00TH8488)*. IEEE. 2000, pp. 80–84.
- [20] M. Connolly and C. J. Sreenan. "Analysis of UDP performance over bluetooth". In: *IT&T2003: Proc. of the IT & Telecommunications Conference*. 2003.
- [21] S. Kumar and S. Rai. "Survey on transport layer protocols: TCP & UDP". In: *International Journal of Computer Applications* 46.7 (2012), pp. 20–25.
- [22] A. Loonstra. *Videostreaming with Gstreamer*. 2014.
- [23] L. Merritt and R. Vanam. "x264: A high performance H. 264/AVC encoder". In: *online* [http://neuron2.net/library/avc/overview\\_x264\\_v8\\_5.pdf](http://neuron2.net/library/avc/overview_x264_v8_5.pdf) (2006).
- [24] Chaneru. *X264 Settings*. Feb. 2013. URL: [http://www.chaneru.com/Roku/HLS/X264\\_Settings.htm#intra-refresh](http://www.chaneru.com/Roku/HLS/X264_Settings.htm#intra-refresh).
- [25] A. Schimpe, S. Hoffmann, and F. Diermeyer. "Adaptive Video Configuration and Bitrate Allocation for Teleoperated Vehicles". In: *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*. 2021, pp. 148–153. doi: 10.1109/IVWorkshops54471.2021.9669258.
- [26] M. Baqué, G. Alemanno, S. Adeli, E. Bonato, M. D'Amore, S. P. Garland, K. Gwinner, C. Herrmann, J. Helbert, P. Irmisch, et al. "Vulcano (Italy) lava fields as Mars and Venus analogs: field and laboratory characterization". In: (2022).
- [27] G. De Astis et al. *Geology, volcanic history and petrology of vulcano (central aeolian archipelago)*. *Geol Soc Mem* 37 (1): 281–349. issn: 04354052. 2013.