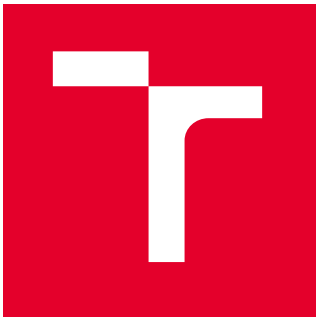


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

DEVELOPMENT OF AN AUGMENTED REALITY APPLICATION FOR VISUALISATION OF THE ENVIRONMENT AND CONTROL OF A DEMOLITION EXCAVATOR

VÝVOJ APLIKACE ROZŠÍŘENÉ REALITY PRO ÚČELY VIZUALIZACE OKOLÍ A ŘÍZENÍ DEMOLIČNÍHO BAGRU

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Ondřej Richter**

AUTOR PRÁCE

SUPERVISOR — **doc. Ing. Robert Grepl, Ph.D.**

VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Solid Mechanics, Mechatronics and Biomechanics
Student:	**Bc. Ondřej Richter**
Degree programm:	Mechatronics
Branch:	no specialisation
Supervisor:	**doc. Ing. Robert Grepl, Ph.D.**
Academic year:	2024/25

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Development of an augmented reality application for visualisation of the environment and control of a demolition excavator

Brief Description:

Advances in virtual reality are increasing the potential for its use in industrial practice, and the move to augmented reality allows the real and digital worlds to be combined. This combination makes it possible to increase safety and user orientation in space. Modern systems, such as the Oculus Quest 3 goggles, offer significant computing power, which is crucial for demanding computation and image processing.
This thesis focuses on the development of a mobile application for augmented reality goggles that allows the visualisation and simulation of industrial machine movement using camera footage and computer vision.

Master's Thesis goals:

1. Research available technologies in the field of virtual and augmented reality, with a focus on the possibilities of developing specific applications.
2. Propose ways of measuring essential variables and possible modifications of the excavator in order to obtain the necessary parameters and machine states (e.g. estimation of the centre of gravity).
3. Do the development of a mobile application for augmented reality glasses that processes and visualizes camera footage with the lowest possible latency.
4. Create a 3D model of the excavator and implement it in the app along with the parameters of the excavator.
5. Test the functionality of the application with a real system.

Recommended bibliography:

[1] Meyers, Scott. Effective Modern C++. O'Reilly Media. 2014. ISBN 9781491903995

[2] GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM,2007. ISBN 978-80-214-3530-8

[3] Romero, Marcos. Sewell, Brenden. Blueprints Visual Scripting for Unreal Engine 5. 2022. Packt Publishing.ISBN 978-1801811583

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2024/25

In Brno,

L. S.

prof. Ing. Jindřich Petruška, CSc. Director of the Institute	doc. Ing. Jiří Hlinka, Ph.D. FME dean

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací systému pro vizualizaci provozních stavů demoličního bagru s využitím rozšířené reality, jehož hlavním cílem je zvýšení bezpečnosti operátora při manipulaci se strojem. V rámci práce je řešeno stanovení limitních poloh těžiště stroje za účelem prevence jeho převrácení. Dále se zabývá přenosem obrazového záznamu s minimální latencí prostřednictvím WiFi sítě do zařízení pro rozšířenou realitu a vytvořením funkčního virtuálního dvojčete bagru pro vizuální zpětnou vazbu v reálném čase.

Summary

This thesis focuses on the design and implementation of a system for visualizing the operational states of a demolition excavator using mixed reality, with the primary objective of improving operator safety during machine handling. The work explores the determination of the excavator's center of gravity limit positions to prevent tipping. Furthermore, it focuses on the transmission of video footage with minimal latency via a WiFi network to an mixed reality device, and the creation of a functional digital twin of the excavator for real-time visual feedback.

Klíčová slova

Smíšená realita, těžiště, tenzometr, virtuální dvojče, demoliční bagr, nízká latence, ArUco, prostorové povědomí, Meta Quest 3, Meta Spatial SDK

Keywords

Mixed reality, center of mass, strain gauge, digital twin, demolition excavator, low latency, ArUco, spatial awareness, Meta Quest 3, Meta Spatial SDK

Bibliographic citation

RICHTER, O. *Development of an augmented reality application for visualisation of the environment and control of a demolition excavator.* Brno: Brno University of Technology, Faculty of Mechanical Engineering, 2025. 80 pages, Master's thesis supervisor: doc. Ing. Robert Grepl, PhD..

I hereby declare that I have completed this thesis independently and that I have not used any sources or assistance other than those explicitly referenced. All data and results presented are my own work unless otherwise stated.

Ondřej Richter

Brno . . . .

I would like to express my gratitude to my supervisor, doc. Ing. Robert Grepl, Ph.D., for his guidance and support. I also extend my thanks to all the members of the Mechatronics Laboratory. Finally, I am deeply grateful to my family and my beloved for their encouragement, patience, and emotional support throughout my studies.

Ondřej Richter

Contents

1 Introduction

As a member of the mechatronics team, I had the opportunity to work on several industrial projects. One such project was a collaboration with Mrōzek, a company specializing in demolition. The project involved the development of a mechanical platform capable of lifting a small excavator onto a tall chimney and securing it in place. Additionally, hydraulic legs were designed to enable controlled movement on the chimney's top surface. The excavator's surroundings were intended to be monitored from four different angles using industrial cameras.

Last specification is important for situational awareness, because the working environment during the demolition can pose significant threats to the surrounding personnel. Nowadays it might be difficult to get rid of old structures in urban areas due to the high concentration of surrounding buildings. Special threats pose high buildings and chimneys as they can't be simply torn down. Company Mrōzek uses small excavators to disassemble the structures from top to bottom. Workers sitting in the cabin are essentially digging under themselves. This maneuver can be very dangerous even with significant safety precautions. By allowing the operator to control the excavator from the ground, the dangers can be reduced.

In this project, I was tasked with the implementation of four video streams into the mixed reality headset, which became one of the main goals of this thesis. Additional subtasks were added over time, each contributing to the successful fulfillment of the overall project. Together, they form the foundation of this thesis and the results are intended to improve overall safety during the demolition. In order to provide a better understanding of my thesis, **I will introduce my goals.**

- The first objective is to design methods for measuring the excavator's **center of mass**. This is necessary to gain insights into its **stability limits**, as the movement of the excavator's arm shifts the center of mass. Accurate knowledge of the center of mass will be utilized in the mixed reality application.

- The second objective is the development of a **mobile application** for a mixed reality headset. Mixed reality spatial applications can potentially provide greater situational awareness than traditional mobile interfaces. This task includes selecting a **suitable headset** and determining optimal methods for integrating four real-time camera streams with **minimal latency**. The final solution should enable the operator to monitor the excavator remotely.

- The third objective is to create and implement a **3D model** of the **excavator**, including its parameters, into the headset. This should enhance the operator's understanding of the excavator's state on the chimney.

2 General research

The aim of this chapter is to outline several conceptual and methodological directions through which the objectives of this thesis may be approached. Rather than offering definitive answers, the research will introduce several key areas and outline their positive and problematic sections.

This includes an overview of the various forms of immersive technologies, such as virtual reality, augmented reality, and mixed reality. Each of them plays a significant role in the user experience and also in overall system design. Furthermore, the chapter provides a brief survey of available software development tools that support the creation of applications within immersive environments. Lastly, it addresses the topic of parameter acquisition using direct and indirect measurement and parameter estimation methods.

Brief diagram of the whole research is visible below in the picture (2.1).

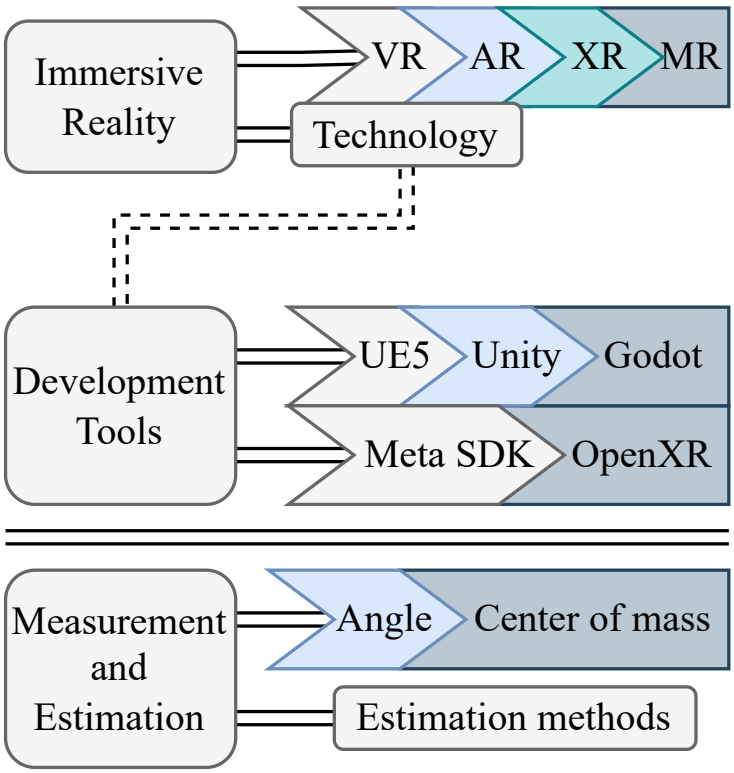


Figure 2.1: Diagram representing research flow

2.1 Immersive reality

Interactions between the virtual and the physical world have never been more immersive, thanks to the constant technological evolution. Immersive reality can take the form of a funny game or a helpful tool, but it can also manifest as an unintended or undesired experience. Immersive realities are often categorized by distinct terms which are visible in the figure 2.2.

Headsets like Bigscreen Beyond [1] or Shiftall MeganeX [2] are exclusively Virtual Reality (VR) [3] headsets, often used for racing games or aviation simulators. Their primary goal is to fully immerse the user in a virtual environment, effectively isolating them from the physical world. Any external influence could negatively impact the user's experience within this virtual realm.

On the contrary, Augmented Reality (AR) [4], Extended Reality (XR) [5], and Mixed Reality (MR) [6] are designed to be influenced by the external world. By leveraging and enhancing aspects of the real environment, the immersion is less prone to disruption by outside factors.

AR overlays digital content onto the real world without necessarily integrating or responding to real-world elements. The product is generally referred to as AR goggles as they use transparent waveguide displays. An example of this technology is Microsoft HoloLens [7] or Meta Ray-Ban [8]. In contrast, XR utilizes the physical environment to enhance the experience. For instance, spatial anchors [9] can be used to position virtual screens or models in fixed locations relative to the real world. A practical example would be a 3D bounding box representing a real-world table acting as an obstacle to a falling virtual ball.

MR serves as a general term encompassing all realities that interact with the real world, blending physical and digital elements to create new environments where real and virtual objects coexist and interact in real time [6]. It often refers to the combination of image data. However, it is possible to argue that any virtual reality that can enhance the experience through the usage of the real world could be considered mixed reality. For instance, some racing games allow the utilization of a feedback controller or even feedback seats. In combination with these technologies, a virtual headset could be considered Mixed. For the purposes of this thesis, MR will be considered as a combination of visual data.

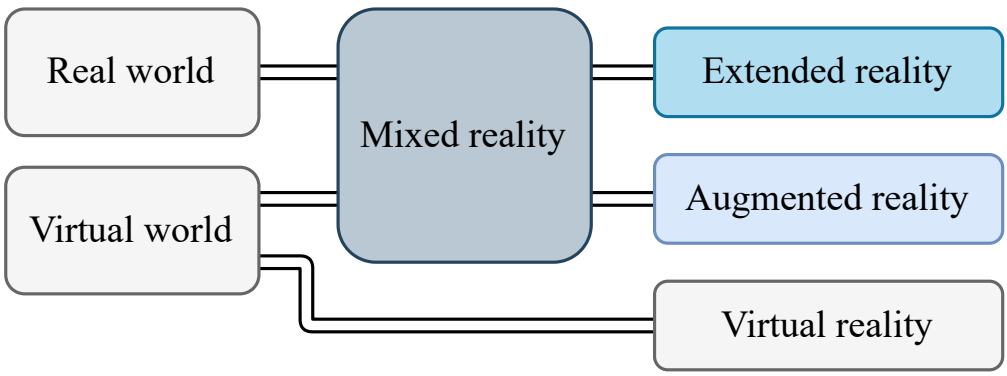


Figure 2.2: Diagram representing state of realities

2.1.1 Optics and display technologies

Every type of digital reality has to be implemented on a piece of hardware (HW), that has the features allowing for good immersion. The essential capabilities are clear optics, high-resolution displays, and good sensor fusion for spatial positioning.

There are a few different types of lenses, that are often used for VR and MR applications. The types can be either categorized by the materials [10] or by the technology used. Glass lenses have good optical properties, but they are heavy and difficult to manufacture compared to the other lens materials. Plastic lenses are more common due to their light weight and ease of manufacturing with worse but comparable optical quality. In this context, "plastic" refers to a plethora of different materials like resin, polycarbonate, and specialized high-index materials as they are usually made from plastic-like substances [10].

Technological differences in the lenses can be made to optimize certain aspects of the lenses. The most basic type of lens is a spherical lens. This type, commonly made from glass, is often too heavy and has numerous optical defects like spherical aberrations. Both issues can be resolved by using the following types of lenses [11].

The Fresnel lens was originally made as a light guide in lighthouses, but its construction was developed to optimize the thickness and weight of the lens [12]. Modern manufacturing capabilities allow us to make lenses as thin as a piece of paper with good optical clarity, but the other optical properties are usually much worse. The discretizations lead to unwanted scattering and loss of light [13] which can be visible in the picture 2.3 created using an online simulator [14]. This type of lens is found in most head-mounted displays.

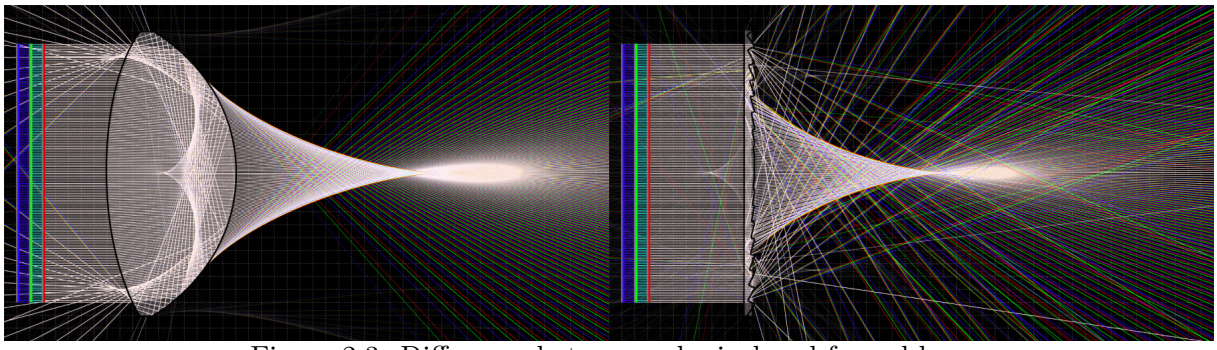


Figure 2.3: Difference between spherical and fresnel lens

Aspherical lenses [15] address the issues of chromatic aberration, providing clear images in a wide field of view, but do not solve the issue of weight. Aspherical lenses are known to be used in Varjo [16] headsets, but their overall usage is quite rare.

Technology that solves both spherical aberration and the weight problem is known as polarization-based folded optics, commonly referred to as pancake lens [17]. By utilizing half mirrors and polarizing filters, pancake lens offers the best image of all available optics. Downsides are poor optical efficiency and difficult manufacturing procedures. These lenses can be found in Meta Quest 3 and pro headsets [18].

For most AR headsets that resemble regular glasses, a so-called waveguide (2.4) can be used. These waveguides are made from a piece of glass or resin that has a specific internal structure. This structure can guide photons through a narrow, flat space similar to the optics of dioptric glasses.

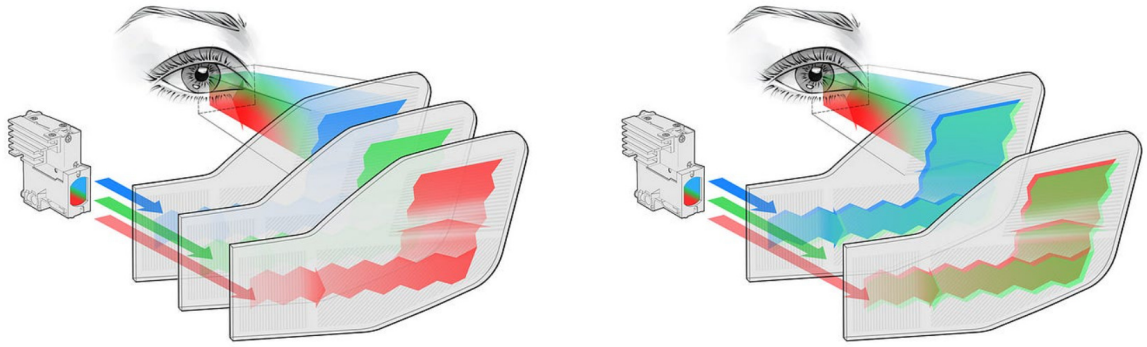


Figure 2.4: WaveGuide lens [19]

Optics can directly affect a field of view (FOV) [20]. It can be defined in both horizontal and vertical axes or by the diagonal FOV. Its value directly specifies the area of visibility from the eye's point of view. Most modern headsets have a horizontal FOV above a hundred degrees [21].

Same as the optics, the displays are a very important part of the whole assembly. Most of the affordable headsets use either one wide LCD panel or two square panels, each for one individual eye.

The benefits of the LCD are an affordable price, high resolution and wide viewing angle. The main downside of the panel with static backlight is a very low contrast ratio compared to the other technologies [22]. When using a dynamic backlight, the contrast is much better. However, these displays suffer from so-called blooming, which appears as a light halo around the bright objects, which is especially noticeable around text.

OLED's and micro LED's are a newer technology [23], making them the more expensive option with superior specifications. Headsets with these panels benefit from better efficiency, deeper dynamic range, faster response times, and much higher brightness and vibrance.

Another important aspect of modern displays is high resolution, refresh rate, and latency. Resolution is defined by pixels per inch (PPI) or in the case of VR and MR headsets pixels per degree (PPD). Usual resolution is around 20 PPD, but for instance, Meta Quest 3 supposedly has a PPD of twenty-five and Varjo XR-4 [16] has around fifty. Resolution without speed is not enough these days, a typical refresh rate of the displays is sixty at minimum and 144 Hz in the better alternatives. High enough resolution and fast refresh rate are key for the elimination of motion sickness that is often connected with VR and MR [24].

All important metrics are shown in the table below (2.1). Instead of the advertised PPD, a calculated one was used to better compare the individual headsets.

Brand Product	Optics [−]	FOV $[deg]$	Display type [−]	PPD $\left[\frac{px}{deg}\right]$	Refresh rate $[Hz]$
Varjo XR-4	Aspheric	120	Mini LED	∼33	90
Apple Vision pro	Pancake	∼105	Micro OLED	∼35	100
Meta Quest 3	Pancake	110	LCD	∼20	120
Pico 4 Ultra	Pancake	104	LCD	∼21	90
Valve Index	Fresnel	108	LCD	∼15	144
Microsoft HoloLens 2	Waveguide	43	N/A	∼30	60

Table 2.1: Comparison of different headsets

2.1.2 Processing capabilities

Another way to categorize immersive technologies is by their dependency on processing units. Some headsets function primarily as head-mounted displays (HMDs), as they are fully dependent on an external PC. They are typically connected via a DisplayPort or a USB Type-C interface that supports high-speed video data transfer. Head tracking in these systems is performed using external infrared trackers or by specialized laser stations. Although this tracking solution offers high accuracy, it requires external HW.

An alternative approach uses in-built inertial sensors combined with camera data to track the headset's position in real time. While this method is less accurate, it eliminates the need for external HW. The majority of MR headsets use this approach, as they are designed to operate independently of an external computer.

A significant leap in the processing power of ARM-based [25] devices has enabled the elimination of external computing units, allowing full independent battery use. The most commonly used systems on chip (SoCs) are produced by Qualcomm, particularly the Snapdragon series. Meta Quest 3 uses Snapdragon XR2 Gen 2 [26], and even though it is not the most powerful chip, it is optimized for low power consumption and minimal heat dissipation. These SoCs also include graphical accelerators and specialized circuits for AI computing.

Comparing classical computers equipped with dedicated graphics cards to systems utilizing Snapdragon SoCs reveals notable differences in computational performance. In terms of total computational power, PCs with x86 architecture generally outperform ARM-based counterparts. However, when evaluating performance relative to power consumption, ARM SoCs demonstrate superior single-core capability per watt. For instance, Qualcomm's Snapdragon X Elite SoCs achieve approximately 6 to 8 points per watt in Cinebench 2024, whereas comparable x86 processors like the AMD Ryzen 7 8845HS and Intel Core Ultra 7 155H score around 3 points per watt. This indicates that ARM-based systems can deliver more than double the performance per watt compared to their x86 equivalents [27].

2.2 Software development tools

As established in the previous section, many different platforms support some form of reality enhancement. To utilize these platforms meaningfully, appropriate development tools are required. The choice of tool depends on the specific platform.

For platforms that rely on an external computer, development tools must be compatible with the operating system of that computer. For instance, if the external computer runs Windows, a development tool that supports Windows development is necessary.

In contrast, for platforms that perform their own computing and operate on an internal operating system, different tools are needed. The most commonly used operating systems on such platforms are Android or various Linux distributions.

For general development, most platforms support the creation of virtual worlds using complete pre-made solutions such as game engines, which often eliminate the need for additional functionality. However, for applications requiring specialized features, it may be preferable to use a dedicated VR/MR API or a software development kit (SDK) provided by the specific manufacturer.

2.2.1 Game engines

Game engines are powerful tools that simplify the development of complex applications. Originally developed by gaming studios for internal use, these tools combined assets and applied logic. Over time, the companies streamlined the process that led to the creation of high-quality games. While many engines remain proprietary—such as Frostbite [28] and Snowdrop [29], some companies have chosen to make their engines publicly available under specific conditions.

One of the most popular game engines to date is Unreal Engine. Originally developed for a game called Unreal, it has since become an industry standard for creating visually stunning and highly playable games. Unreal Engine is also widely used in the entertainment industry as a real-time rendering engine for virtual production and mixed reality applications [30]. At the time of writing this thesis, the latest version is Unreal Engine 5 [31].

Another widely used engine is Unity. It was specifically designed to help aspiring developers enter the game development industry. Unity supports both 2D and 3D game development and uses the C# programming language [32].

Unity has recently faced backlash over changes to its licensing model, which initially proposed fees based on the number of game installs. Although the company later reversed these changes, the controversy led many developers to consider switching engines for their future projects .

One alternative engine gaining popularity is Godot [33]. This lightweight, open-source engine is particularly well-suited for 2D games but also supports 3D development. Godot uses its own scripting language, GDScript, and offers support for VR and MR development.

Although not a game engine in its current form, older versions of Blender included a built-in game engine. Today, Blender [34] functions as an all-in-one open-source application for 3D modeling, texturing, video and sound editing, sculpting, drawing, and rendering. It can also serve as a development tool for MR applications running on external devices.

2.2.2 APIs and SDKs

Software development kits (SDKs) and application programming interfaces (APIs) are essential components in the development of any MR platform. An SDK is a collection of tools, libraries, debuggers, and comprehensive documentation. SDKs support both rapid prototyping and the development of robust final applications.

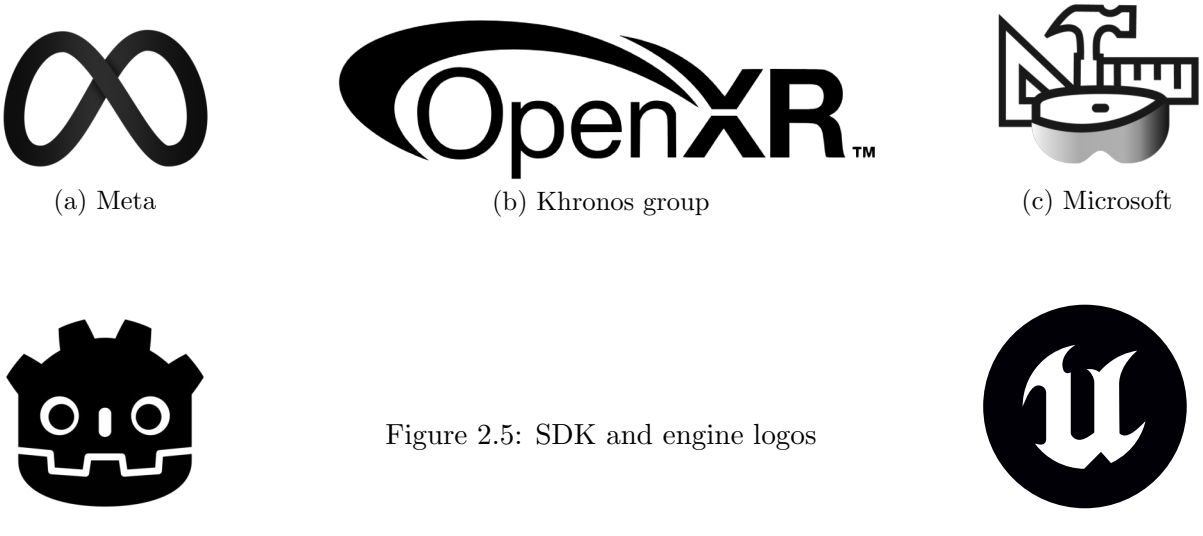
Vendor-specific SDKs provide access to proprietary HW features while maintaining a high-level programming interface to accelerate development. Examples of such SDKs include Microsoft Mixed Reality Toolkit (MRTK3) [35], Valve SteamVR [36], Google VR SDK, ARCore [37], ARKit, PICO XR SDK [38], Qualcomm's Vuforia [39], and Varjo SDK [40] (2.5).

Hardware-agnostic SDKs and APIs are not tied to any specific hardware. They achieve portability by abstracting certain functionalities. Examples include WebXR [41] and OpenXR [42].

OpenXR is an open-source API standard developed by the Khronos Group [42]. Its goal is to unify development across hardware platforms. By incorporating vendor extensions—such as foveated rendering, hand tracking, eye tracking, and depth masking—into optional modules, OpenXR is ideal for early-stage development when the hardware is not finalized and also for production scenarios requiring advanced features (2.5b).

A typical OpenXR workflow consists of several basic steps:

1. Instance creation − Initializes the OpenXR runtime.
2. System call − Selects the appropriate hardware and platform.
3. Session creation − Binds the chosen graphics API and initiates a session.
4. Space and action set definition − Defines interaction contexts and input mappings.
5. Swapchain creation − Enables rendering by looping and updating frames.
6. Polling for state changes − Continuously checks for events before each frame.
7. Cleanup on exit − Destroys swapchains, sessions, and OpenXR instances.



(a) Meta (b) Khronos group (c) Microsoft

(d) Godot (e) Unreal Engine

Figure 2.5: SDK and engine logos

2.3 Measurement and estimation methods

2.3.1 Angle measurement

Direct methods require no additional computation to determine the desired angle. Many devices, such as mechanical protractors, can be used for this purpose. However, measuring joint coordinates of an excavator in real-time requires more advanced devices.

One solution involves rotary encoders mounted on rotating shafts. These encoders can be either absolute or incremental. Absolute encoders output a specific n-bit value corresponding to a particular angle. Incremental encoders, on the other hand, count pulses to measure angular displacement. These devices can achieve high precision and are a suitable choice for angle measurement.

If it is not possible to implement an encoder in the system, a digital inclinometer can be used instead. These sensors often work on MEMS technology and typically combine accelerometers and gyroscopes. This sensor might not retain its accuracy or work properly when exposed to vibration noise.

when direct measurement is not possible, an indirect technique has to be used. For instance, Encoder measurement on a linear actuator would classify as an indirect measurement with fairly high precision. Installation would require tampering with existing hydraulic systems which might not be permitted.

Alternative methods include using LiDAR and point cloud analysis. By generating extensive 3D point data, it is possible to approximate the spatial position of an object. This approach can be very computationally expensive.

Another approach is using computer vision with standard RGB cameras. By placing markers on the system, it is possible to track joint coordinates in real time. This method generally provides lower accuracy and robustness compared to other solutions, but its implementation is easy and requires close to no modifications to the existing system.

2.3.2 Center of mass measurement

To measure the center of mass (COM) of a rigid object, two primary methods can be used.

The first method works by suspending the object from multiple points, tracing the vertical planes defined by the suspension rope, and finding their intersection. Steady state is achieved when the net force and torque are equal to zero (eq.: 2.1). This method works well for small simple static objects, but it becomes rather impractical for large changing objects like excavators.

$$\begin{aligned} \sum F &= 0 \\ \sum M &= 0 \end{aligned} \tag{2.1}$$

This equation can also be utilized when using the second method, which involves the measurement of static forces acting on the object. By placing the object on a few force gauges, we can calculate the exact position of the resulting force from the reaction forces of the gauges.

If the masses of the individual parts of the object are known, it is possible to approximate the center of mass from coordinate position using the analytical solution. This approach is done by weighted averaging of individual mass centers (eq.: 2.2).

$$x_{com} = \frac{\sum m_i x_i}{\sum m_i} \tag{2.2}$$

The COM can also be approximated through visual data. By analyzing the volume of a model or the area in an image, an approximate COM position can be estimated. However, this method can be inaccurate due to challenges in estimating material densities and internal mass distributions. In cases where the weight or density of individual components is unknown, estimation methods must be employed to approximate these values.

2.3.3 Estimation methods

Parameter estimation is a mathematical procedure that aims to approximate unknown variable values using real measured data. Classical techniques such as the method of moments, least squares, and maximum likelihood estimation provide estimates based on sampled data, typically without explicitly considering statistical distributions.

The least squares method minimizes the sum of squared errors, known as residuals, between the model predictions and the observed data. A common extension is nonlinear least squares, where the model output depends nonlinearly on the parameters. The efficiency of this algorithm can be improved using the Gauss-Newton method. The Levenberg-Marquardt algorithm further stabilizes convergence by incorporating both gradient descent and damping strategies [43],[44],[45].

The Bayesian approach treats parameters as random variables and updates statistical beliefs based on observed data. It combines a prior distribution, representing prior knowledge, with the likelihood of observed data to form the posterior distribution. A single parameter estimate can be obtained by taking the maximum value of this distribution estimate or its mean. Algorithms such as Metropolis-Hastings, Sequential Monte Carlo, Particle Filters, and Variational Inference are commonly used for Bayesian estimation [46].

3 Center of mass measurement and estimation

This chapter presents the development of a methodology for estimating the masses of excavator arm segments. Knowledge of these masses enables subsequent simulations and visualizations of the machine's center of mass in an external device. This provides the operator with real-time feedback, especially valuable when the system approaches tipping conditions.

To support this goal, a set of system specifications is defined to guide the process. Several measurement strategies are considered and evaluated based on their feasibility and practical constraints. Emphasis is placed on non-intrusive techniques suitable for deployment on existing machinery.

The methodology combines mechanical measurement with computer vision, enabling both force and angle acquisition without requiring structural modifications. The chapter also discusses the reasoning for the selected approach and outlines the steps taken to prepare for real-world implementation, including validation and data processing.

Figure 3.1: Brokk 520 [47]

3.1 Specification and general concept

- Development of measurement methods and procedures
- Minimal modifications of the measured excavator
- Acquisition of joint coordinates and center of mass for multiple static positions
- Repeatability of the measurement

3.1.1 Concept

Based on a review of relevant literature, force transducers were selected as the primary measurement method. The excavator is equipped with four actuated legs capable of supporting its weight under various conditions (3.1). The goal is to place force transducers beneath each leg to measure the forces acting at these points across different arm positions. The angles of the arm will be determined using computer vision techniques, specifically by detecting the positions of ArUco markers. Subsequently, the center of mass will be determined using one of two methods, which will be detailed in the following sections.

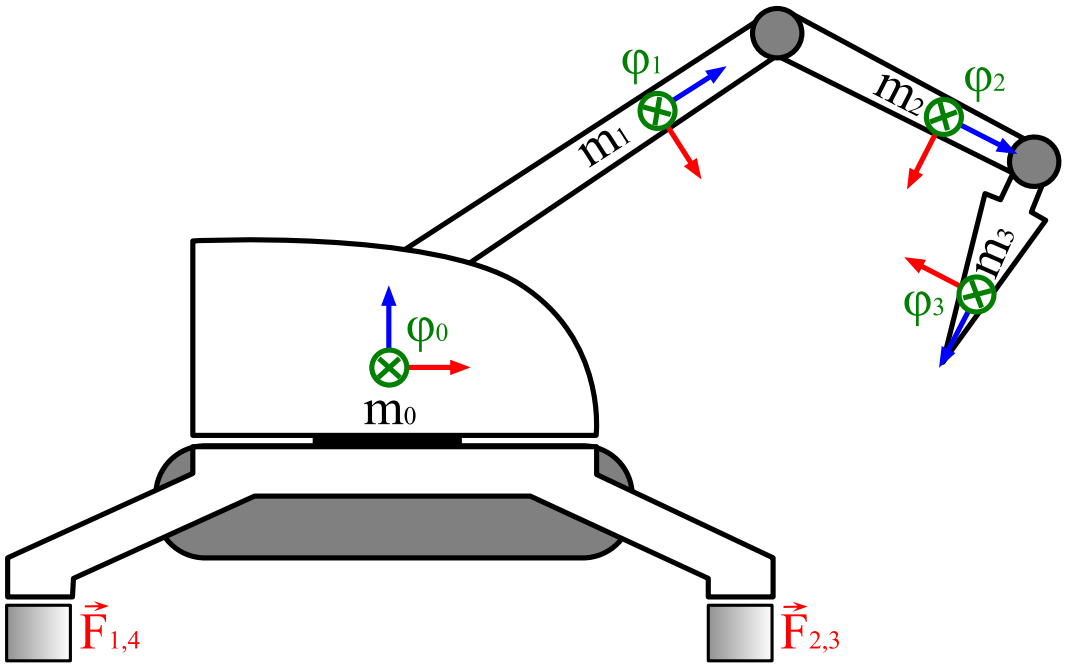


Figure 3.2: Concept of the measurement

Exact location of the force gauges and the ArUco markers is visible in the figure 3.2. Blocks $\vec{F}_{1,2,3,4}$ represent force transducers and $\varphi_{0,1,2,3}$ represent ArUco markers.

3.2 Proof of concept

3.2.1 Simulation

As mentioned above, the center of mass of the excavator can be determined using several methods. The first involves creating a dependency table, where the center of mass depends on up to three joint coordinates. Since this step occurs early in the study, a simplified Simscape Multibody simulation of an approximate model was made to accelerate the process of choosing the right method.

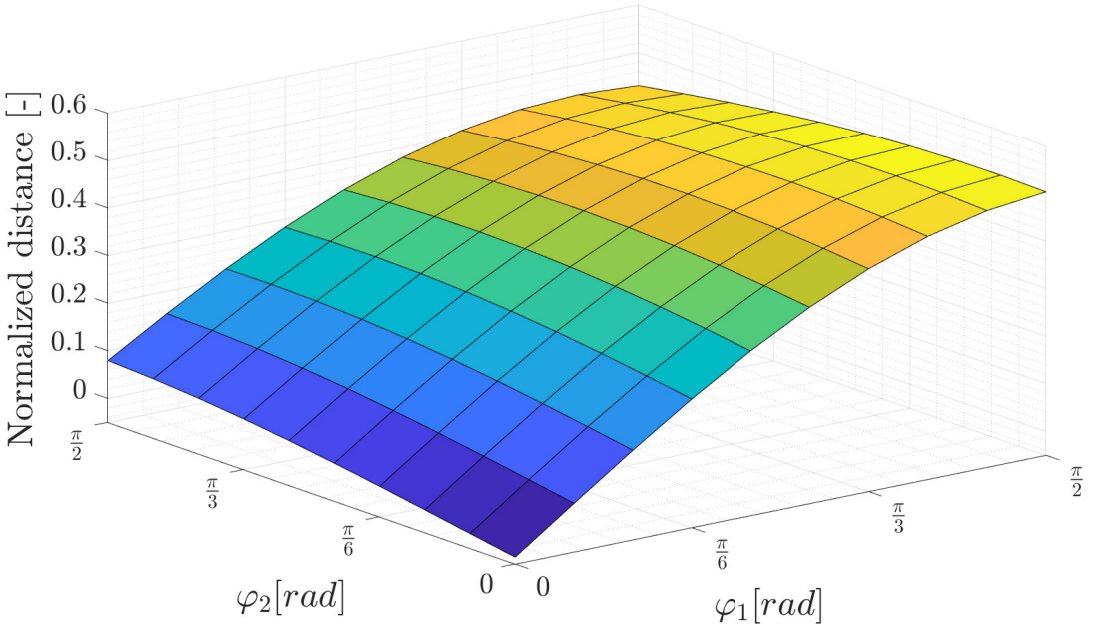


Figure 3.3: Dependency table for two angles

In Figure 3.3, the behavior of the center of mass as a function of joint coordinate changes is clearly apparent. That means that the creation of the dependency table is possible. However, the number of joint coordinates on the real excavator is three, which would create a three-dimensional lookup table. That implies cubic growth of the number of values inside this lookup table.

The total number of required measurements is directly tied to the desired resolution. For example, achieving a resolution of ten steps per joint (approximately every 20 degrees) would require 1,000 measurements. Since the measurement process is not easily automated, this approach becomes unmanageable in practice.

An alternative is to calculate the center of gravity directly using an analytical equation. The equation $T_x = f\left(\varphi_{1,2,3}, L_{1,2,3}, m_{0,1,2,3}\right)$ depends on the joint angles $\varphi_{1,2,3}$, segment lengths $L_{1,2,3}$, and segment masses $m_{0,1,2,3}$. While the kinematic parameters are known, the individual masses are usually unavailable and must be approximated or estimated.

Given that the first method is very impractical, the second method is used. A detailed explanation of this approach can be found in Section 3.6 **Analysis**, under the subsection Excavator mass distribution estimation.

3.2.2 Physical model

To further prepare for the actual measurement, a simple physical model of the excavator's base was constructed (3.4). This model was designed so that the placement of the strain gauges corresponds to the placement of the force transducers on the actual excavator. This configuration allows for testing the measurement system in a controlled environment.The physical model aids in identifying unforeseen issues that may arise during actual measurements. Determining the exact mass of an object necessitates a calibration procedure.

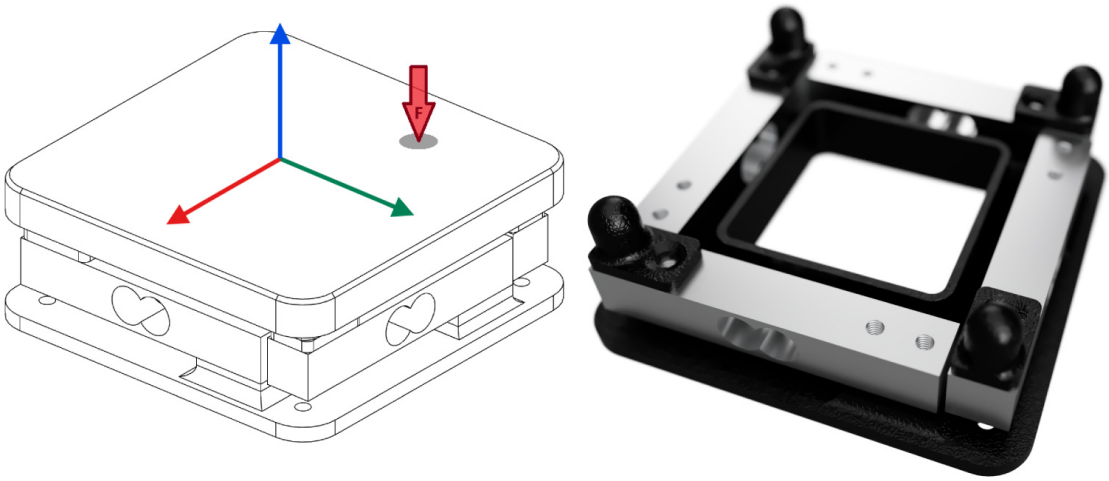


Figure 3.4: Sketch and render of physical model

Strain gauges measure strain by detecting changes in electrical resistance, which are then converted into voltage signals. These voltage signals must be converted back to strain and subsequently to force and mass. When the measurement system is unloaded, the average offset of each strain gauge can be determined using the following equation 3.1.

$$x_{off} = \sum_{i=1}^{n} \frac{x_i}{n} \tag{3.1}$$

where n is the number of measured points.

Similarly, the gain of the gauge can be obtained using the equation 3.2.

$$x_{gain} = \frac{\sum_{i=1}^{n} \frac{x_i - x_{off}}{n}}{m} \tag{3.2}$$

where m is the known mass applied during calibration.

As mentioned in the subsection Simulation, the masses of the system are not entirely known, making it difficult to determine the forces acting on the transducers. An alternative approach to determining these forces is explained in the following section.

3.3 Force transducer development

Commercial force gauges are often excessively expensive. To reduce costs, one viable approach is to develop a transducer using readily available semi-finished workpieces. Selecting commonly used materials with regular shapes simplifies the design process and minimizes the need for complex calculations. The designed transducer should be validated using the finite element method (FEM) to ensure measurement linearity and safety.

3.3.1 Dimensions consideration

To determine the optimal strain for the transducer, simple equations can be employed to specify the sensor's dimensions. The approximate mass of the excavator with attached tooling is around six tons. Consequently, the force gauge must withstand the associated pressures. Given that the yield strength of steel is approximately 350 MPa, the maximum strain on the sensor can be calculated using the following equation 3.3.

$$\sigma = E\varepsilon \rightarrow \varepsilon = \frac{\sigma}{E} \approx 1.6 \left[\frac{mm}{m}\right] \tag{3.3}$$

Lets assume that dynamic movement of the excavator can generate substantial forces on the gauges, then a larger safety factor should be used. If maximum strain is set to around $300\frac{\mu m}{m}$, the resulting safety factor approximates to around five.

original iteration of the transducer was designed to be made from a pipe workpiece. Various types of piping are commercially available but, from an economic standpoint, a welded pipe was the most cost-effective choice. Because the dimensions were configured in a way that the strain equaled $310\frac{\mu m}{m}$, there was no need for manufacturing. High strain is also beneficial for achieving a high signal to noise ratio (SNR), especially when used with a good data acquisition system. The main concern was the weld failure and buckling of the whole gauge due to uneven load distribution. To mitigate the issue, a FEM experiment was performed to get an idea of potential risks.

3.3.2 FEM simulation

As previously mentioned, to validate the chosen dimensions, a simple FEM experiment was performed. Several cylindrical pipe models were created using a design modeler and imported into static structural and buckling analysis modules. The model was constrained at the bottom and loaded with a force of 50,000 newtons, which is equivalent to the complete mass of the excavator.

Special attention was given to the center of the piece, as this is the optimal location for attaching the strain gauge. The primary concern was whether the cylinder would exhibit linear behavior under load.

The maximum directional strain of the gauge in the measurement region is approximately $35 \left[\frac{\mu m}{m}\right]$ which aligns with the value calculated using the reorganized equation 3.14, given known dimensions, a force of $F = 50\,[kN]$ and $E = 210 \cdot 10^9\,[Pa]$.

3.3.3 Manufacturing

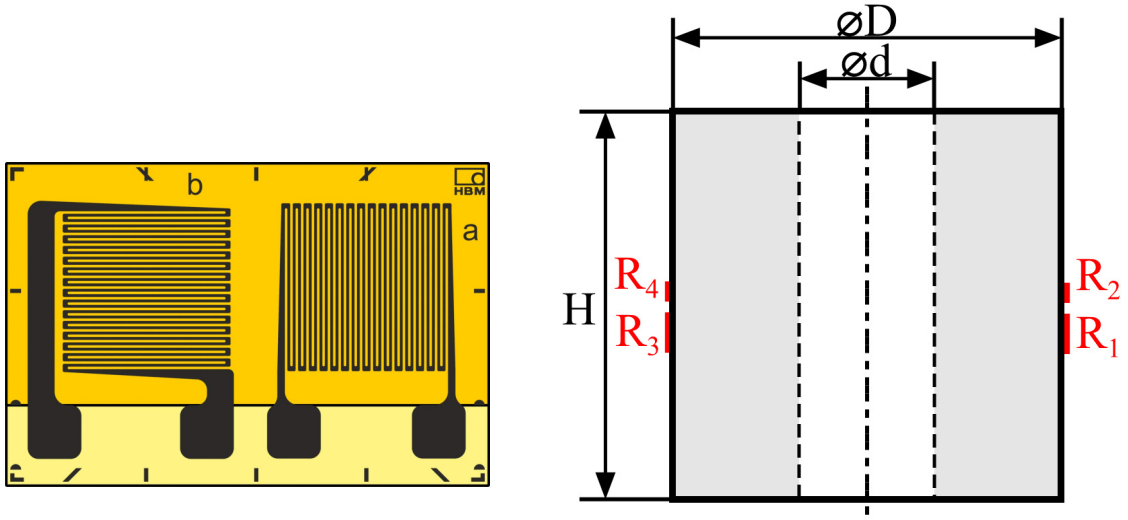
After performing the buckling simulation using FEM and consulting with experts in the field, several modifications were necessary to better align the design with realistic dimensions. According to the professionals, a strain of 30 $\left[\frac{\mu m}{m}\right]$ would be sufficient, provided that the acquisition card offers adequate resolution. To ensure stability during measurement, the outer dimensions had to remain fixed. This left only the inner diameter d, visible in the picture 3.5b, as a variable parameter.

A former mechatronics PhD student offered assistance with the manufacturing of the transducer. Since their primary expertise does not lie in working with hard metals, the available tooling for this task was limited. Nevertheless, the inner diameter was set to 30 mm. Due to manufacturing constraints, the precise dimensional targets were not fully achieved. However, the functional behavior of the workpiece remained unaffected. All dimensions and their corresponding strain values are listed in table 3.1.

	Outer diameter $[mm]$	Inner diameter $[mm]$	Height $[mm]$	Force $[kN]$	Strain $\left[\frac{\mu m}{m}\right]$
original dimensions	100	95	100	50	310
recommended dimensions	100	30	100	50	33
manufactured dimensions	98	29	100	50	35

Table 3.1: Comparison of transducer development iterations

After finalizing the dimensions, the force transducer was completed by applying the strain gauge. The most suitable type for this application would be a rosette strain gauge, as it compensates for potential rotational misalignment during placement. However, to minimize the overall cost of the measurement setup, strain gauges already available locally were used. The strain gauge employed is the 1-XY31-0.6/120 model from HBM [48]. This gauge is capable of measuring strain in two directions, as illustrated in Figure (3.5a).



(a) Strain gauge [48] (b) Workpiece

Figure 3.5: Force sensor dimensions and layout

The strain gauge was connected in a full-bridge configuration (3.6). This setup offers several advantages, including improved sensitivity, as well as better compensation for temperature fluctuations and bending forces. The main drawback of using a full-bridge configuration is the increased number of strain gauges required, which raises the overall cost.

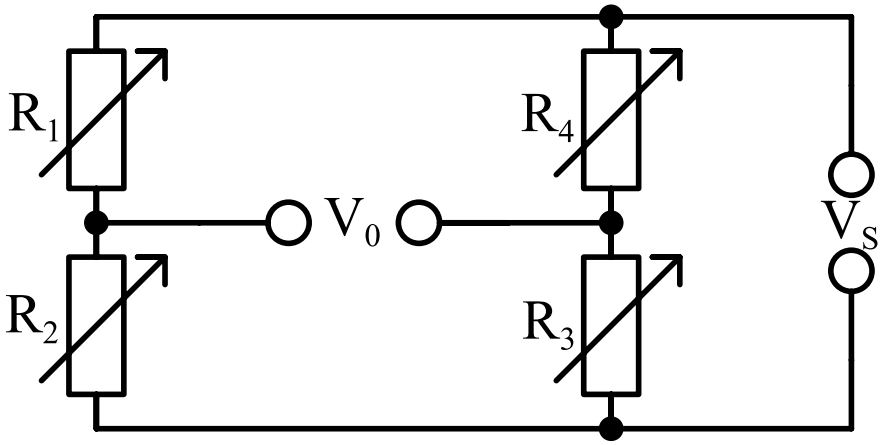


Figure 3.6: Schematic of the connections

The electrical connections and gauge surface bonding were carried out by a specialist at the faculty, as these procedures require specialized skills that could not be acquired within the time frame of this thesis (3.7a). The sensitive strain gauge was mechanically protected using a special compound visible in the picture 3.7b

(a) Gauge application (b) Protective material

Figure 3.7: Force transducer

3.3.4 Force transducer linearity

Using a calibrated force sensor positioned above the manufactured force gauge, it is possible to assess the linearity of the device by applying gradually increasing load. This was done by a small hydraulic press capable of producing up to 50 kN of force. In the figure (3.8a) it is clearly visible that the gauge suffers from slight nonlinearity. This deviation from the linear function is visible in the figure (3.8b).

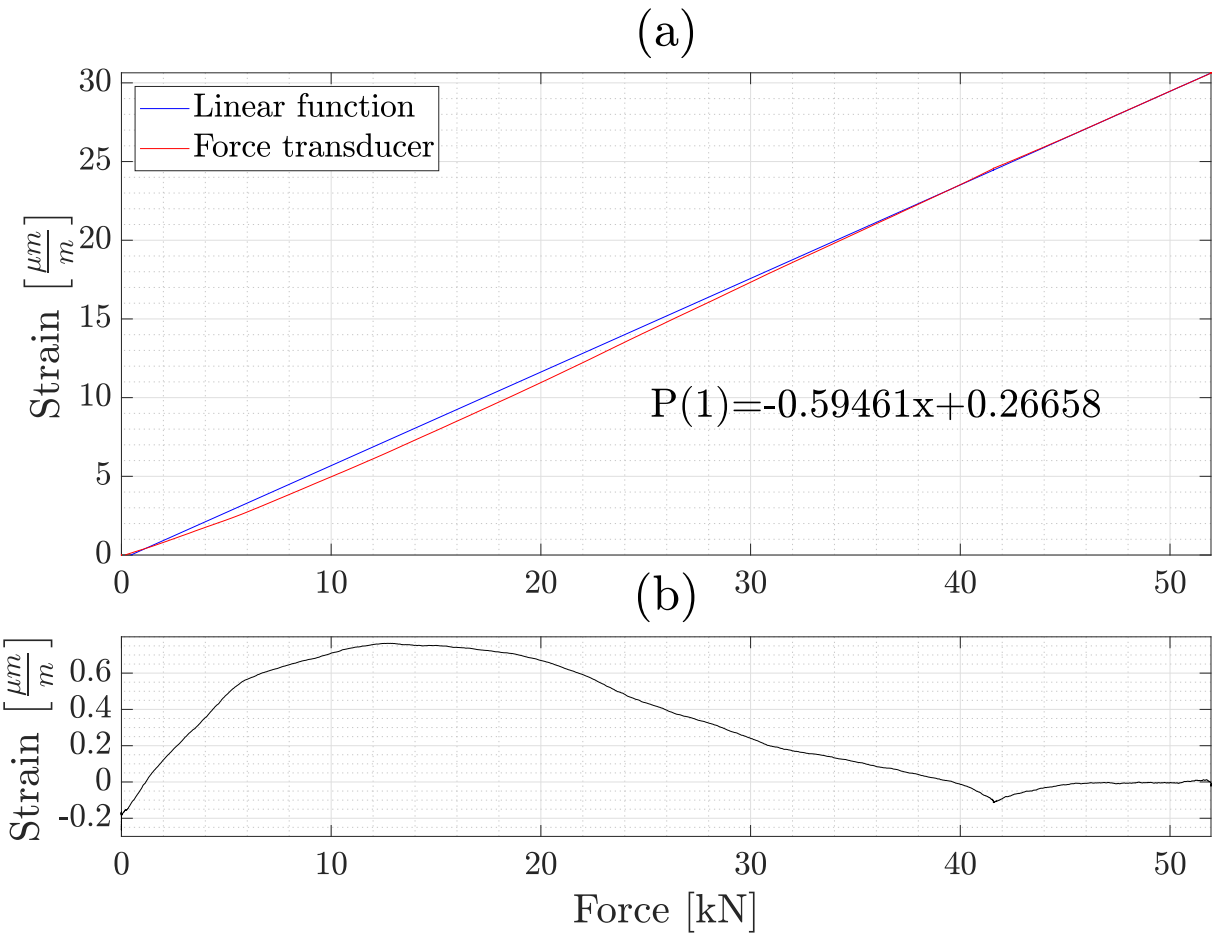


Figure 3.8: (a) Force gauge linearity (b) Linearity error

During measurement, the applied force on each gauge was in the region of 10 kN to 25 kN, which means that the measurement was slightly affected by this error. However, this nonlinearity can be removed with a polynomial function whose coefficients are easily obtainable by solving a system of linear equations (3.4).

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} x_1 & x_1^2 & \cdots & x_1^n \\ x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{3.4}$$

Where c_n are the polynomial coefficients, x_n is the force from the calibrated sensor, and y_n is the corresponding strain from the manufactured gauge.

3.4 Manipulator joint coordinates estimation

Due to constraints in time and resources, machine vision measurement employing ArUco [49] markers was selected. This method provides a viable alternative to traditional measurement techniques for determining the coordinates of excavator arm joints. It requires minimal modifications to the excavator. However, prior to conducting any measurements, a portable computer application is necessary.

3.4.1 OpenCV

The OpenCV library was chosen for the development of the application as it is an open-source standard offering numerous useful features [50]. The core library supports basic image processing tasks such as image format conversion, transformations, filtering, enhancements, camera calibration, and the creation of simple user interfaces. For more advanced functionalities, extension modules can be built from the source. These modules include algorithms for face recognition, tracking, ArUco marker detection, and additional deep neural networks.

An example of a simple color correction is provided below, where the contrast variable functions as the gain of the image and the brightness variable as a value offset. By using the imread method, a variety of image formats can be imported in color, grayscale, or with reduced bit depth.

```
cv::Mat src = cv::imread("inputImage.png", cv::IMREAD_COLOR);
cv::Mat corrected;          // corrected image matrix allocation
double contrast = 1.2;
int brightness = 20;
src.convertTo(corrected, -1, contrast, brigthness);
```

The output of the convertTo method is the corrected matrix. Another useful operation is thresholding, for which OpenCV provides a dedicated function. Initially, the image is converted to grayscale by averaging all colors into a single-channel matrix. Subsequently, the threshold method can be applied, resulting in a binary black-and-white image.

```
cv::Mat gray, thresh;
cv::cvtColor(corrected, gray, cv::COLOR_BGR2GRAY);
double tValue = 100;
double maxValue = 255;
cv::threshold(gray, thresh, tValue, maxValue, cv::THRESH_BINARY);
```

In most machine vision applications, edge detection is a crucial step. An easy method to compute edges involves applying a convolution with a derivative kernel, such as the Sobel operator. OpenCV provides a built-in function, Canny, which performs edge detection by applying gradient calculations, non-maximum suppression, and double thresholding. By setting the high threshold, low threshold, and kernel size, this function returns an image highlighting edges in white.

```
cv::Mat edges;
double lowThreshold = 50;
double highThreshold = 150;
int kernelSize = 3;
cv::Canny(gray, edges, lowThreshold, highThreshold, kernelSize);
```

3.4.2 Video acquisition

Still images are often insufficient for real-time machine vision applications. In such cases, video capture is utilized. Similar to reading an image, OpenCV's VideoCapture class allows for capturing video from various sources. For example, to read from a video file, the following code can be used.

```
cv::VideoCapture cap("C:/videoFile.MP4",cv::CAP_ANY);
```

However, for real-time processing, a live video source, such as a USB camera, is preferable. If only one camera is connected, it can be accessed using index 0. For systems with multiple cameras, it's useful to use the device's specific path to avoid ambiguity. On Windows, this can be achieved by specifying the device's unique ID.

```
Source = "@device_pnp_\\?\usb#vid_046d&pid_085c&mi_00#...
        8&21a08ad1&0&0000#...
        {65e8773d-8f56-11d0-a3b9-00a0c9223196}\global"
cv::VideoCapture cap(Source,cv::CAP_ANY);
```

If the video source is accessible over a network, such as an IP camera streaming via RTSP, OpenCV can capture the stream using the appropriate URL. It's important to ensure that the stream is secured to prevent unauthorized access. An example of capturing a network stream is as follows:

```
Source = "rtsp://username:password@1.1.1.1:500/media/video1"
cv::VideoCapture cap(Source,cv::CAP_ANY);
```

Data from various sources can be raw, compressed, or have low/high frame rates. To avoid unnecessary complications, these attributes must be explicitly set. When using a USB webcam, it may be necessary to specify the CAP_DSHOW backend to ensure proper functionality. To accelerate video transfer, the feed can be set to the Motion JPEG (MJPG) format for compression.

```
cap.set(cv::CAP_PROP_FOURCC, cv::VideoWriter::fourcc('M','J','P','G'));
```

After successfully capturing and compressing the video, the frame rate and resolution can be configured. If these parameters are not set, the video capture defaults to the camera's standard settings.

```
cap.set(cv::CAP_PROP_FPS, 30);
cap.set(cv::CAP_PROP_FRAME_WIDTH, frameWidth);
cap.set(cv::CAP_PROP_FRAME_HEIGHT, frameHeight);
```

Setting a fixed frame rate is ineffective if the exposure time exceeds the frame interval. If the exposure is set to automatic, the frame rate gets adjusted according to the exposure time. To fix the exposure, the following code can be used.

```
cap.set(CAP_PROP_AUTO_EXPOSURE, 0);
cap.set(CAP_PROP_EXPOSURE, -6);
```

By configuring these variables, data consistency is ensured throughout the measurement process.

Similar to human eyes, though to a lesser extent, digital cameras and their optics have imperfections. By calibrating the camera, the best possible image quality for machine vision applications can be achieved [51],[52]. In OpenCV, several camera parameters are tuned using intrinsic parameters and distortion coefficients. Intrinsic parameters describe the internal characteristics of the camera and are represented by the camera calibration matrix.

$$\mathbf{C} = \begin{bmatrix} f_x m_x & s & p_x \\ 0 & f_y m_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

where f_x and f_y are focal lengths, m_x and m_y are pixel sizes, p_x and p_y are principal points in pixel coordinates, and s is a skew factor

Camera lenses inherently introduce various distortions due to imperfections in lens design, manufacturing, and alignment. These distortions cause deviations from the ideal pinhole camera model, leading to inaccuracies in image representation. To correct these deviations, camera calibration involves estimating distortion coefficients that model the lens's behavior. OpenCV supports several types of distortion coefficients, including radial, tangential (decentering), and thin prism distortions. The number of coefficients used can vary. Four, five, eight, twelve, or fourteen, depending on the desired calibration accuracy and the complexity of the lens distortion.

$$DistCoeff = \left[k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6, s_1, s_2, s_3, s_4\right] \tag{3.6}$$

Where $k_{1,2,3,4,5,6}$ are radial distortion coefficients. Radial distortion occurs when light rays bend more near the edges of the lens than at the center, causing straight lines to appear curved in the image. This distortion is symmetric around the principal point and is modeled using a simplified equation (3.7).

$$x'', y'' = x', y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + ... \tag{3.7}$$

Coefficients $p_{1,2}$ represent tangential distortion, that occurs when the lens and the image sensor are not perfectly parallel, causing the image to appear tilted. This misalignment is modeled using the following expression (3.8).

$$\begin{aligned} x'' &= ... + 2p_1 x' y' + p2(r^2 + 2x'^2) + ... \\ y'' &= ... + 2p_2 x' y' + p1(r^2 + 2y'^2) + ... \end{aligned} \tag{3.8}$$

Thin prism distortion is represented by coefficients $s_{1,2,3,4}$. It describes imperfections in the lens elements that cause additional radial and tangential distortions. This distortion is modeled using part of the equation (3.9)

$$\begin{aligned} x'' &= ... + s_1 r^2 + s_2 r^2 \\ y'' &= ... + s_3 r^2 + s_4 r^2 \end{aligned} \tag{3.9}$$

where r from equations 3.7, 3.8, 3.9 is the distance of the pixel from the image center.

$$r^2 = x'^2 + y'^2 \tag{3.10}$$

3.4.3 ArUco

ArUco markers, named after augmented reality and the University of Córdoba, are binary square fiducial markers primarily used for camera pose estimation [49]. Each marker encodes a unique identifier (ID) within a black-bordered square, enabling identification and estimation of position and rotation in 3D space.

The marker detection algorithm involves several key steps:

1. Image Pre-processing - The input image is converted to grayscale, and adaptive thresholding is applied to segment potential marker regions.
2. Contour Detection - Contours are extracted from the segmented image. Non-convex shapes and those not approximating a square are discarded.
3. Candidate Filtering - Additional filters remove contours that are too small, too large, or too close to each other.
4. Marker Identification - For each candidate, a perspective transformation is applied. The marker is divided into a four by four grid based on the dictionary's specifications, and the bit pattern is analyzed to determine the marker's ID and orientation.

To utilize ArUco in OpenCV, the aruco module from the opencv_contrib repository must be included, as it is not part of the core OpenCV modules. Building OpenCV with the extension modules ensures access to ArUco functionalities. After linking all dependencies and static libraries, ArUco should be accessible. To use this feature, a detector object has to be made.

```
Dictionary dictionary = getPredefinedDictionary(DICT_4X4_50);
DetectorParameters parameters;
RefineParameters refineParams = RefineParameters(10.0f,1.0f,true);


parameters.useAruco3Detection = true;
parameters.cornerRefinementMethod = CORNER_REFINE_SUBPIX;


Ptr<ArucoDetector> detector =
    makePtr<ArucoDetector>(dictionary, parameters, refineParams);
```

Dictionary specifies the bitfield the detector should find. Using the DetectorParameters, it is possible to adjust some functionalities. RefineParameters sets the minimum distances for ArUco candidates, error correction rate, and orientation checking function.

Using the previously mentioned camera calibration, it is possible to solve pose estimation to recover the 3D position and orientation of the marker in space relative to the camera.

To achieve accurate camera calibration, a built-in calibration function can be utilized. Prior to using this function, it is recommended to capture as many images as possible that provide new information. Some sources suggest that capturing at least fifteen images is necessary to attain sub-pixel precision. The validity of these claims is evaluated in the validation chapter.

The images should contain a ChArUco board, which resembles a checkerboard with embedded ArUco markers from a specific dictionary. To minimize false distortions in the images, the board should be mounted on a solid backboard. The calibration process yields the camera matrix and distortion coefficients.

Lets set the size of the ChArUco board based on the known dimensions of standardized paper sizes. An A4 sheet has an aspect ratio of approximately 1.41. To fit as many squares as possible, a seven by five square configuration, resulting in a ratio of 1.4, is employed. To ensure the accuracy of the calibration, the exact sizes of the markers and squares are measured. The following code snippet is used to generate the calibration coefficients.

```
Ptr<aruco::CharucoBoard> charucoBoard = ...
   ... makePtr<aruco::CharucoBoard>(Size(squaresX,squaresY), ...
        ... squareLength, markerLength, *dictionary);

aruco::interpolateCornersCharuco(markerCorners, markerIds, ...
    ... gray, charucoBoard, charucoCorners, charucoIds);

double error = aruco::calibrateCameraCharuco(CharucoCorners, ...
    ... CharucoIds, charucoBoard, imageSize, cameraMatrix, ...
    ... distCoeffs, rvecs, tvecs);
```

The final function returns the mean squared re-projection error of the calibration. It also populates the "cameraMatrix" and "distCoeffs" variables, which are utilized in the subsequent pose estimation stage. The resulting camera calibration matrix and distortion coefficients k_1, k_2, k_3, p_1, and p_2 are presented below.

$$\text{Camera matrix} = \begin{bmatrix} 1417.11 & 0 & 957.22 \\ 0 & 1406.96 & 538.35 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{Distortion coefficients} = \begin{bmatrix} 0.0607 & -0.3455 & 0.0024 & -0.0005 & 0.4084 \end{bmatrix}$$

As stated before, angle measurement using ArUco markers can be hindered due to several reasons. One of these reasons is wrong camera calibration. The following experiment represented by a graph will show this particular issue.

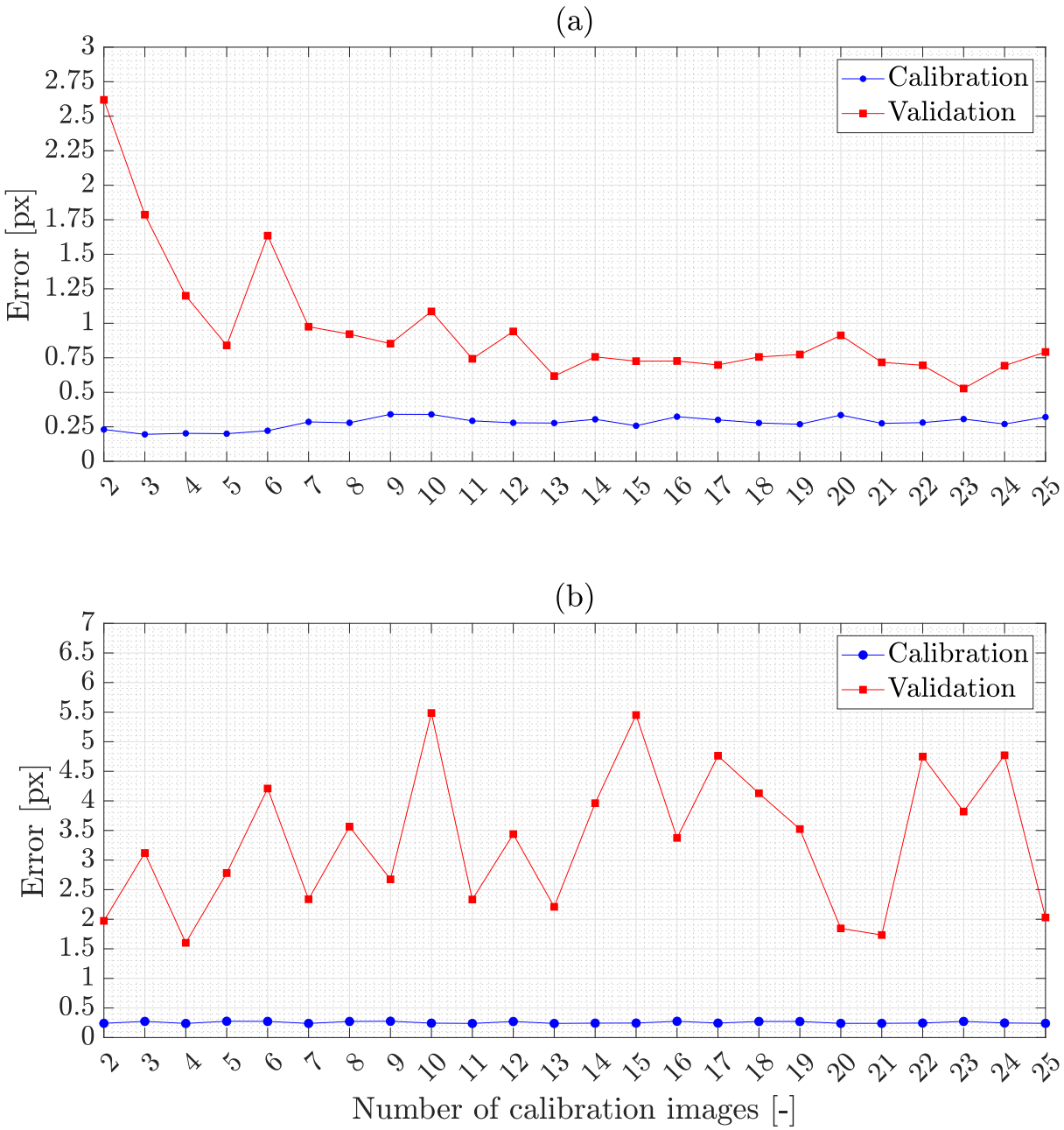


Figure 3.9: Calibration and validation reprojection error

The experiments shown in figures 3.9a and 3.9b present the mean squared estimation error (MSE) of the ArUco markers during the camera calibration. The proper calibration resulted in an MSE of less than one pixel when using approximately ten to fifteen images (3.9a). MSE of an improperly calibrated camera does not converge and stays in the region of four pixels (3.9b). It is important to mention that each value represents the mean of fifteen validations. The deviation of the error on a properly calibrated camera with the maximum number of images was around 0.5 pixels. And the maximum error of the improper calibration was up to 80 pixels.

The core of pose detection using ArUco markers involves estimating the transformation matrix by minimizing the re-projection error. Unless otherwise specified, the Levenberg–Marquardt algorithm is employed for this optimization process [43][44]. As the estimation function, the pinhole camera model is utilized. This model projects 3D world coordinates onto 2D image coordinates using the camera's intrinsic and extrinsic parameters [52].

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x m_x & s & p_x \\ 0 & f_y m_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.11}$$

This equation represents the transformation of a marker's world coordinates into the camera reference frame, followed by the application of the camera matrix to convert 3D coordinates into 2D pixel coordinates.

To reduce the number of parameters estimated during optimization, Rodrigues vectors are employed [53]. These vectors represent rotations using three parameters, in contrast to the nine parameters of a full rotation matrix. The conversion from Rodrigues angles to a rotation matrix is achieved using the following formula.

$$\mathbf{R} = \mathbf{I}\cos\theta + (1 - \cos\theta)\,\mathbf{u}\mathbf{u}^T + \sin\theta\mathbf{K} \tag{3.12}$$

Where $\mathbf{u}$ is a unit vector of the Rodriguez angles. $\mathbf{I}$ is the identity matrix, and $\mathbf{K}$ is a skew-symmetric matrix as follows.

$$\mathbf{K} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \tag{3.13}$$

OpenCV provides a built-in function that estimates the position the same way as mentioned above. The following code snippet demonstrates the basic usage of the pose estimation function.

```
Mat cameraMatrix; Mat distCoeffs; Mat frame;
float markerSize = 0.08
vector<int> markerIds;
vector<vector<Point2f>> markerCorners;
detector->detectMarkers(frame, markerCorners, markerIds);

vector<Vec3d> rvecs, tvecs;
estimatePoseSingleMarkers(markerCorners, markerSize, ...
... cameraMatrix, distCoeffs, rvecs, tvecs);
```

The output Rodriguez vectors can be written to a file, used to position a 3D object on the same device, or be sent through a web interface to a different device. For better compatibility, it is preferable to convert the pose vectors into their corresponding rotation matrices. This conversion can be performed using Equation 3.12 or by utilizing OpenCV's built-in function.

```
Mat Rmatrix;
cv::Rodrigues(rvecs, Rmatrix);
```

A PC application was deployed using Visual Studio C++ to simplify the process of angle measurement on the job site. As this is only a helper application that won't be used by the end user, it is preferable to use simple controls to avoid additional and unnecessary programming. The output of the application is shown in the figure (3.10). There are simple controls for switching the camera input, decreasing or increasing static exposure, toggling auto-focus, and toggling data acquisition. Data are stored in a CSV format for later analysis.

Figure 3.10: ArUco position acquisition APP

3.5 On-site measurement

Goal of the measurement is to gather joint coordinate and force transducer data in order to find their connection to the center of mass of the excavator using estimation methods. The center of mass is necessary feedback for the operator to ensure safety during demolition. To ensure successful measurement, precise planning is required.

Field measurement procedure

- Usage of solid markers with low reflections
- Revision of the camera calibration
- Test measurement of pose estimation and force traducers
- Test of the data logging systems
- **Measurement on an excavator**
- Documentation of measurement procedures

3.5.1 Data acquisition

To acquire ArUco data, the previously mentioned PC application was used. By using the simple UI, the measurement could be easily repeated or adjusted on the fly. CatmanEasy was used for the acquisition of strain data. The sampling frequency of ArUco detection was 30 Hz and strain data 300 Hz.

Figure 3.11: Acquisition station

3.5.2 Measurement process

The measurement procedure was conducted at the job site located near Ostrava. After arriving, the measurement plan was communicated to the head of the site and to the assisting excavator operator. All the necessary equipment was set up near the excavator (3.11). Two independent laptops were used, each carrying out their own task. To ensure redundancy, markers were filmed from several angles.

The machine was lifted using a forklift and positioned onto the force transducer(3.12). The first measurement session involved manually moving each joint to various random positions and measuring the corresponding force while stopped to ensure avoiding dynamic forces. After a short break, second measurement session was carried out. About fifty position changes were measured during each session. After knowing that all the data were saved, the excavator was gradually lifted while measuring the force to measure the difference between the unloaded and loaded force transducer.

All measurements were documented, saved, and backed up for further analysis. Finally, the equipment was packed, and the site was cleared.

Figure 3.12: Measured excavator

3.6 Analysis

3.6.1 Data processing and correction of strain data

Given that the material is standard structural steel and the dimensions are provided in Table 3.1, the exact force acting on the gauge can be calculated using the following formula.

$$\sigma = \frac{F}{S} \rightarrow F = \sigma S = E\varepsilon S \tag{3.14}$$

Where S is the cross-sectional area of the transducer from the top view, E is Young's modulus for steel, σ is the stress, ε is the strain, and F is the magnitude of the acting force. The area S can be determined using the formula for the area between two concentric circles.

$$S = \frac{\pi}{4}(D^2 - d^2) \tag{3.15}$$

The exact formula for the force magnitude on the gauge is.

$$F = E\varepsilon\frac{\pi}{4}(D^2 - d^2) \tag{3.16}$$

Due to the low strain sensitivity of the transducer, the signal-to-noise ratio (SNR) of the measured signal is relatively low. Since real-time analysis is not required, a simple low-pass filter can be applied to enhance the SNR. A window size of approximately fifty samples was selected, with a sampling frequency of 300 Hz.

The excavator manufacturer specifies a dry weight of 4.5 tons and a tool weight of 300 kg. This aligns with the measured total system mass of approximately 4.8 tons, as illustrated in Figure 3.13.

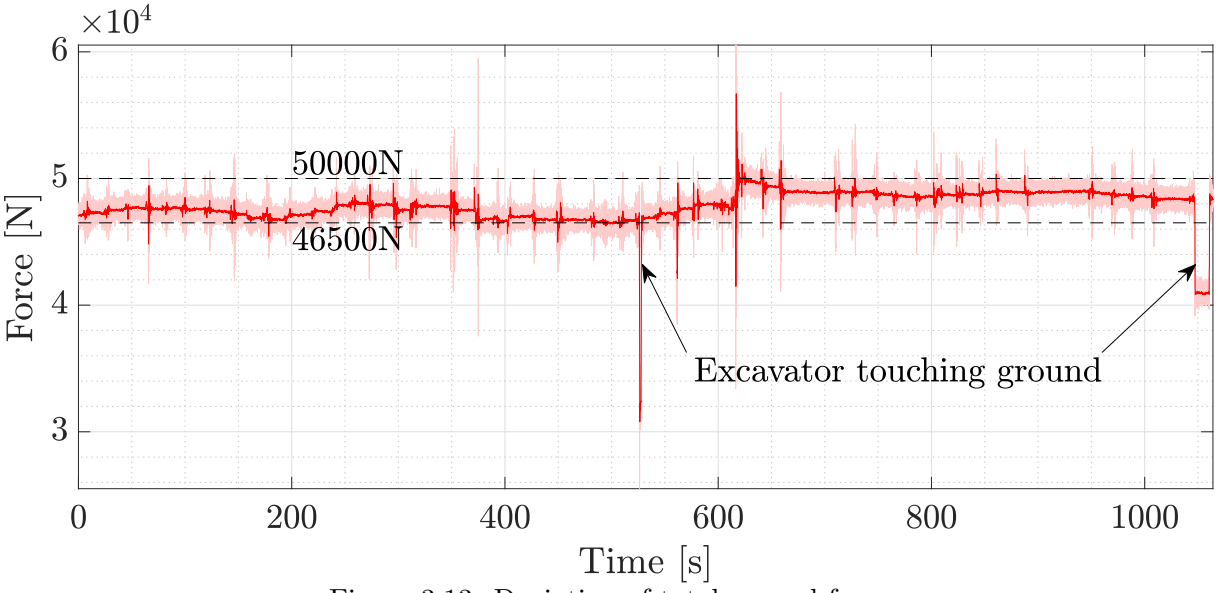


Figure 3.13: Deviation of total normal force

However, significant deviations in the total normal force were observed. In certain instances, these deviations resulted from sudden changes in the excavator arm's position, leading to dynamic oscillations in the measurements. Additionally, variations in static force were detected, which may be attributed to calibration inaccuracies or the influence of forces acting on the gauge in an unexpected way.

As previously mentioned, the SNR of the strain measurements is relatively low, with a noise amplitude of approximately 0.9 $\left[\frac{\mu m}{m}\right]$ and a signal amplitude around 4 $\left[\frac{\mu m}{m}\right]$. However, since the primary objective is to capture the static state of the excavator, the data can be effectively processed using averaging or low-pass filtering to improve the SNR. Filtered data can be seen in the figure (3.14).

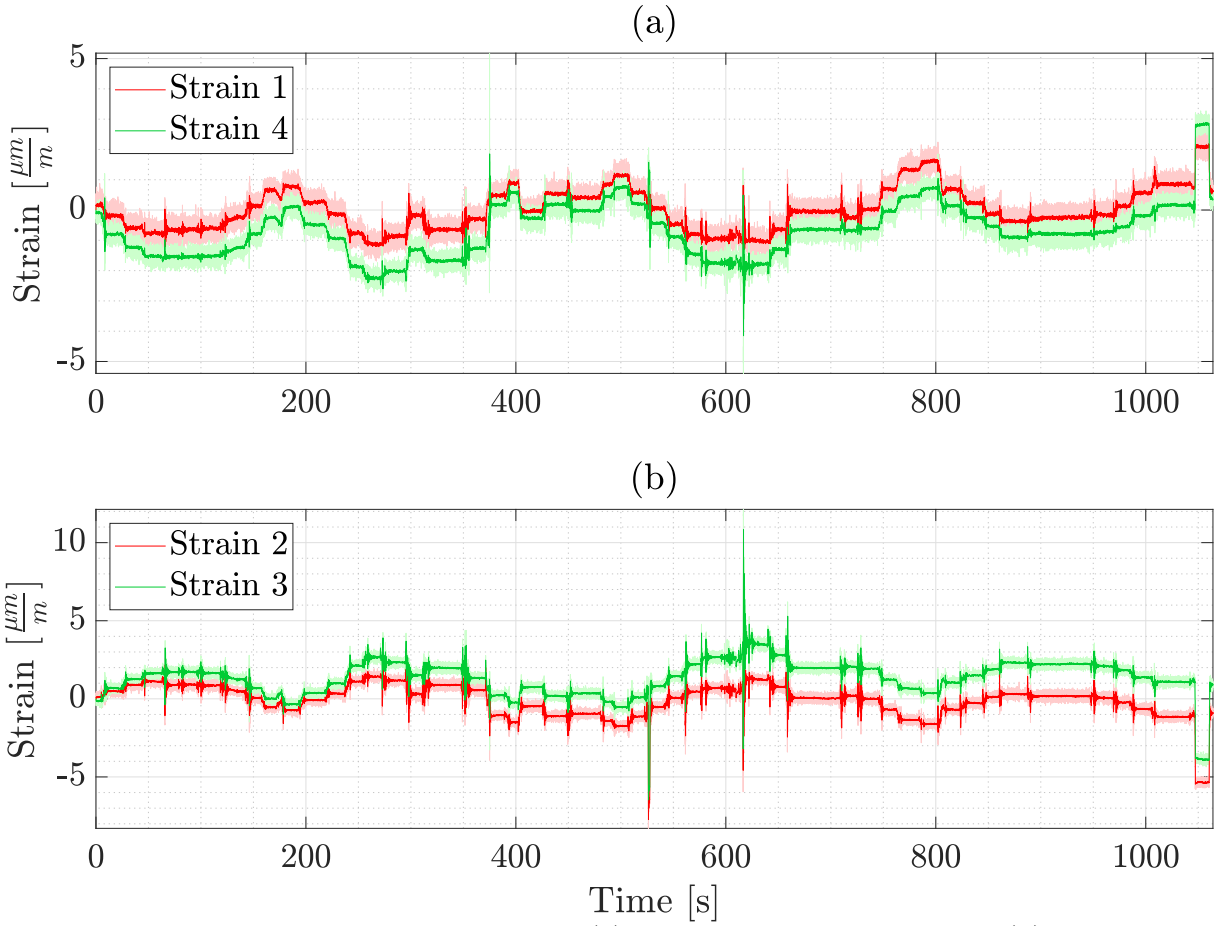


Figure 3.14: Strain on the front (a) and on the back transducers (b)

As previously mentioned in Section 3.5, the excavator was lifted during the measurement process to capture the difference in readings from the force transducer under load and no-load conditions. This procedure enables the calibration of the sensor using the measured data from the machine. The lifting procedure is depicted in Figure 3.15.

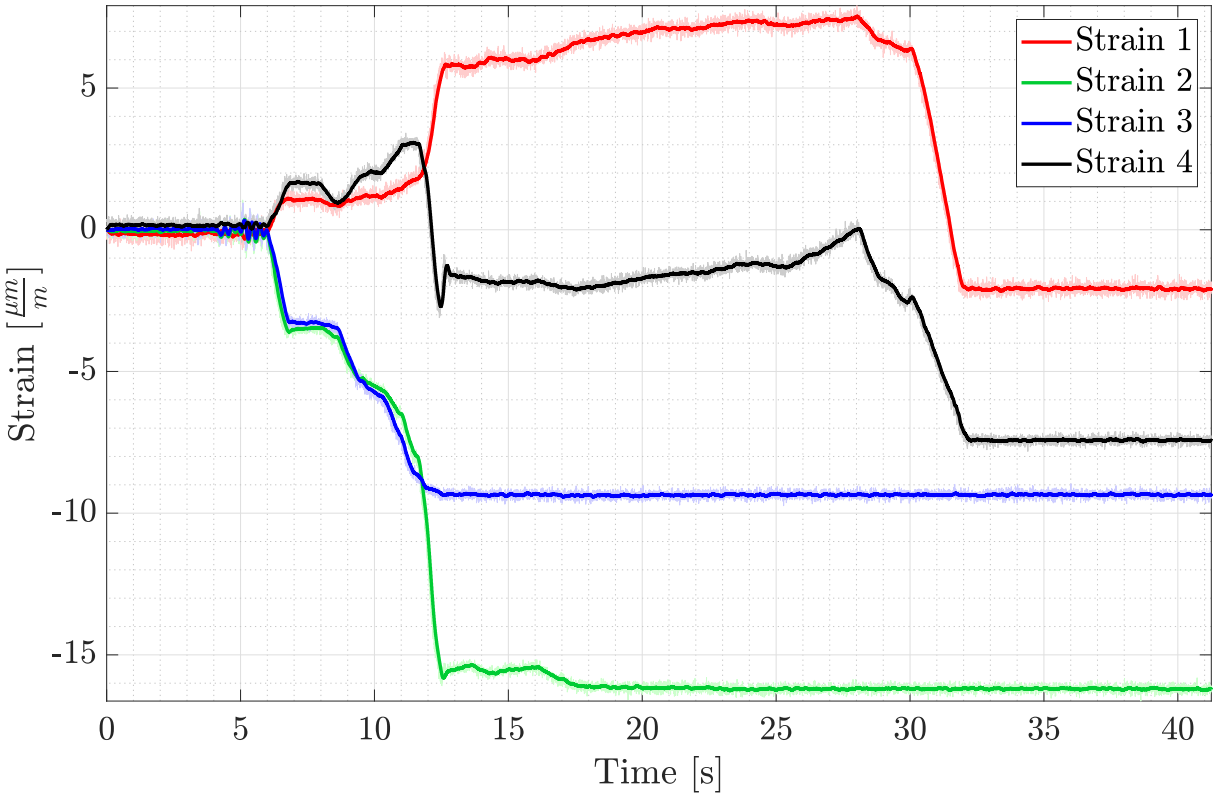


Figure 3.15: Strain after lifting the excavator

3.6.2 Data processing and correction of arm rotation data

Data obtained from ArUco markers represent the rotation and translation of the marker relative to the camera's coordinate system. To utilize this data within the excavator's coordinate system, a transformation from the camera's coordinate frame to that of the excavator is necessary. This transformation can be achieved using rotation matrices derived from the rotation vectors provided by the ArUco marker detection. Rotational matrix has the following form (3.17).

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{3.17}$$

To transform coordinates from the camera frame to the excavator frame, the inverse of the rotation matrix of a base ArUco marker, visible in the picture 3.12, is applied. Since rotation matrices are orthogonal, their inverse is equal to their transpose.

$$\mathbf{R}_{b,c}^{-1} = \mathbf{R}^T \tag{3.18}$$

By applying matrix multiplication of the inverted base rotation and joint ArUco rotation, the orientation data from the ArUco markers can be accurately represented in the excavator's coordinate system.

$$\mathbf{R}_{m,b} = \mathbf{R}_{b,c}^{-1}\mathbf{R}_{m,k} = \mathbf{R}_{b,c}^{T}\mathbf{R}_{m,k} \tag{3.19}$$

Here, m denotes a specific arm marker, b the base marker, and c the camera. $\mathbf{R}_{m,b}$ then represents the rotation of a specific marker with respect to the base marker.

When decomposing the rotation matrix, it is essential to know the order of angles from the original construction. Considering the ZYX convention, the rotation matrix is given by three angles. ψ represents yaw (rotation around the Z-axis), θ pitch (rotation around the Y-axis), and φ roll (rotation around the X-axis). The rotation matrix is then expressed as.

$$R = R_z(\varphi)R_y(\theta)R_x(\psi) \tag{3.20}$$

Assuming perfect measurement, all angles except for the Z-axis rotation (yaw) should remain static and zero. The individual angles are computed using the following formulas, valid for $|\theta| \neq \frac{\pi}{2}$.

$$\varphi = arctan2\left(R_{32}, R_{33}\right) \tag{3.21}$$

$$\psi = arctan2\left(R_{21}, R_{11}\right) \tag{3.22}$$

$$\theta = arcsin\left(-R_{31}\right) \tag{3.23}$$

Due to occasional loss of ArUco marker tracking, some angle data were missing. These gaps were filled using linear interpolation to maintain continuity in the dataset. The resulting angle data were subsequently unwrapped to correct for discontinuities and filtered to reduce noise. To further verify the accuracy of the measurements, angles were also obtained manually. A slight discrepancy between the interpolated and manually measured angles is evident in Figure 3.16. This difference is likely due to human error during manual measurement.

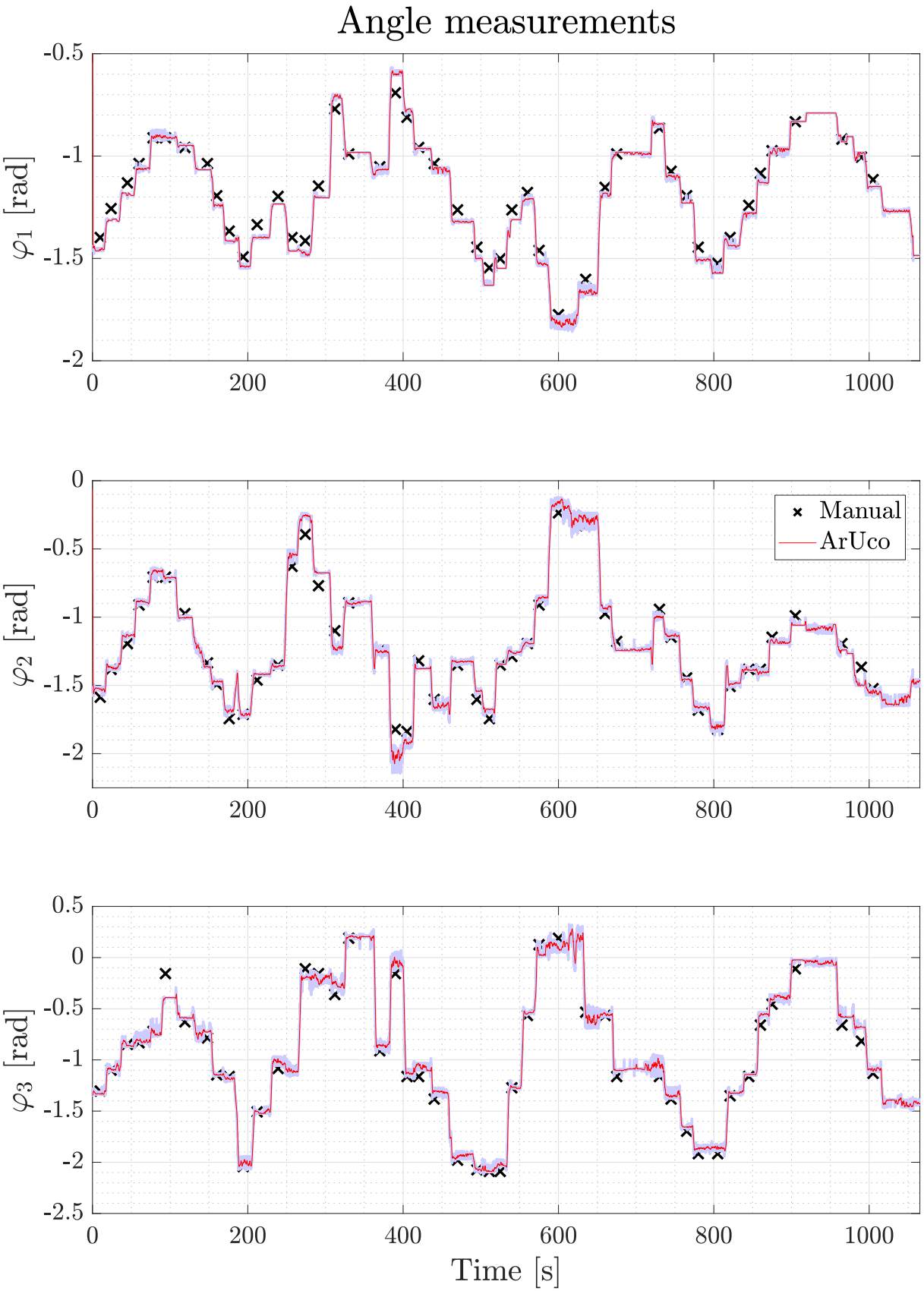


Figure 3.16: Exctracted ArUco angles

3.6.3 Excavator mass distribution estimation

The excavator's arm is a monolithic structure that cannot be easily disassembled. Therefore, estimating the individual component weights requires estimation methods.

To address this, the Levenberg–Marquardt algorithm, also known as the damped least squares method, was utilized [43][44]. This algorithm is particularly effective for solving nonlinear least squares problems, as it interpolates between the Gauss–Newton algorithm and gradient descent, offering robustness and efficiency in parameter estimation.

The parameter update in the Levenberg–Marquardt method is computed by the following equation.

$$dp = -\left(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{J}^T\mathbf{r}\right) \tag{3.24}$$

In this equation, $\mathbf{J}$ represents Jacobian matrix of size $n \times 4$, λ is the damping coefficient, and $\mathbf{r}$ is residual vector of size $n \times 1$.

The Jacobian matrix can be defined by.

$$\mathbf{J}_{ij} = \frac{\partial r_i}{\partial m_j} \tag{3.25}$$

Each residual is calculated as the difference between the estimated and measured centers of mass.

$$r_i = \hat{T}_{x_i} - T_{x_i} \tag{3.26}$$

The objective of the optimization is to minimize a cost function, which in this case is a sum of the squares of the individual residuals.

$$C = r^T r \tag{3.27}$$

Both estimated and measured centers of mass are required for the computation of the residual. By using a known rule for the center of mass,

$$COM = \frac{\int x dm}{m} = \frac{\sum x_i \cdot m_i}{\sum m_i} \tag{3.28}$$

it is possible to calculate the estimated center of mass of the excavator using the following formula.

$$\hat{T}_x = \frac{m_1 \cdot t_{x_1} + m_2 \cdot t_{x_2} + m_3 \cdot t_{x_3}}{m_1 + m_2 + m_3 + m_4} \tag{3.29}$$

The variables m_1, m_2, m_3, m_4 represent the masses of the individual arm segments and serve as optimization parameters. By adjusting these values using a gradient-based method, the predicted center of mass is modified, resulting in the decrease of residuals and thereby reduction of the cost function.

Assuming that the center of mass of each individual arm segment lies at its geometric center, the missing variables $t_{x_1}, t_{x_2}, t_{x_3}$ can be determined using open manipulator forward kinematics.

$$\begin{aligned} \mathbf{t}_1 &= \mathbf{R}_{\varphi_1}\mathbf{R}_{\frac{L_1}{2}}\mathbf{u}_0 \\ \mathbf{t}_2 &= \mathbf{R}_{\varphi_1}\mathbf{R}_{L_1}\mathbf{R}_{\varphi_2}\mathbf{R}_{\frac{L_2}{2}}\mathbf{u}_0 \\ \mathbf{t}_3 &= \mathbf{R}_{\varphi_1}\mathbf{R}_{L_1}\mathbf{R}_{\varphi_2}\mathbf{R}_{L_2}\mathbf{R}_{\varphi_3}\mathbf{R}_{\frac{L_3}{2}}\mathbf{u_0} \end{aligned} \tag{3.30}$$

Where $\mathbf{u}_0$ is the coordinate vector. The matrix $\mathbf{R}_\varphi$ denotes a rotation matrix parameterized by the joint angles, and $\mathbf{R}_L$ represents a translation matrix determined by the known geometric dimensions of the excavator. When unpacked, $\mathbf{t_1}$ is following (3.31).

$$\begin{bmatrix} t_{1,x} \\ t_{1,y} \\ t_{1,z} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\varphi_1) & -\sin(\varphi_1) & 0 & 0 \\ \sin(\varphi_1) & \cos(\varphi_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{L1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \tag{3.31}$$

These transformation chains can be simplified to express the x coordinates of the segment centers of mass.

$$\begin{aligned} t_{x_1} &= \frac{L_1}{2} sin(\varphi_1) \\ t_{x_2} &= L_1 sin(\varphi_1) + \frac{L_2}{2} sin(\varphi_1 + \varphi_2) \\ t_{x_3} &= L_1 sin(\varphi_1) + L_2 sin(\varphi_1 + \varphi_2) + \frac{L_3}{2} sin(\varphi_1 + \varphi_2 + \varphi_3) \end{aligned} \tag{3.32}$$

The actual center of mass can be calculated from measured forces obtained via strain gauges. According to the equilibrium condition (2.1), for an object to be at rest, the sum of all forces and torques must be zero.

$$\sum F = 0 \qquad \sum M = 0$$

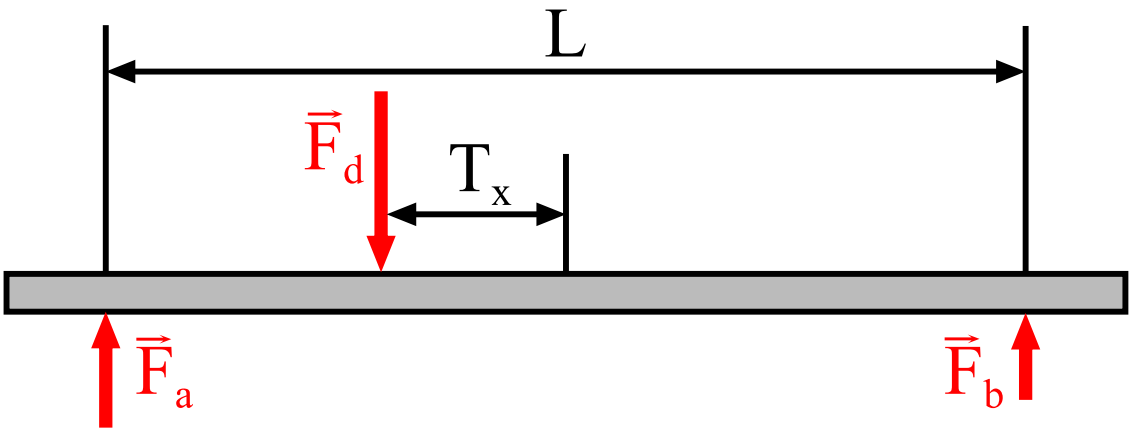


Figure 3.17: Applied forces

As illustrated in the figure 3.17 the force F_d generated by the excavator exerts pressure onto the force gauges, represented by the forces F_a and F_b. The force F_d is applied at the actual center of mass, which is offset from the geometric center. By applying equation (2.1), the following equations are derived.

$$\begin{aligned} \sum F &= F_a + F_b - F_d = 0 \\ \sum M_c &= M_a - M_d - M_b = 0 \end{aligned} \tag{3.33}$$

Rearranging the force equilibrium equation (3.33) yields.

$$F_d = F_a + F_b \tag{3.34}$$

The individual torques can be expressed in terms of their corresponding forces.

$$\begin{aligned} M_a &= F_a \cdot \frac{L}{2} \\ M_b &= F_b \cdot \frac{L}{2} \\ M_d &= F_d \cdot T_x \end{aligned} \tag{3.35}$$

Substituting equations (3.35) and (3.34) into the moment equilibrium equation and solving for T_x yields the equation for the actual center of mass.

$$T_x = \frac{L}{2} \cdot \frac{F_a - F_b}{F_a + F_b} \tag{3.36}$$

3.7 Measurement conclusion

This chapter outlines the development of a measurement methodology aimed at acquiring key states of the excavator, with the center of mass (COM) as the primary focus.

To gain an initial understanding of the problem, a simulation and proof of concept were developed. The simulation revealed that storing COM data in a lookup table was impractical. Achieving meaningful resolution would require over a thousand physical measurements. Therefore, an analytical approach for determining the excavator's COM was chosen. Before investing in or developing expensive force transducers, a proof-of-concept device was created to validate the chosen COM measurement method on a smaller scale.

Following successful validation, full-scale force transducer development was chosen, as it was more economical at the time than purchasing commercial gauges. The development began with analytical calculations to determine the necessary dimensions of the transducer, which were then validated using a simple Finite Element Method (FEM) simulation. Materials were sourced and sent for manufacturing. After applying strain gauges in a full-bridge configuration, the force transducers were prepared for use.

To measure the excavator's joint coordinates, a machine vision approach was selected. This method enabled real-time measurement of all essential data without modifying the excavator. Utilizing C++ with OpenCV and its contrib modules, a measurement application was developed and deployed. The development process included camera calibration, video capture, pose estimation, and the ability to save acquired data to a file. The application was verified on a small locally available excavator before being used for actual measurements.

Upon completing preparations, measurements were conducted using the developed force gauge and C++ application. Three continuous measurements were performed to ensure redundancy. The acquired data were analyzed and used for final parameter estimation. Mass estimation was carried out using both a custom Levenberg–Marquardt algorithm and MATLAB's Parameter Estimation Toolbox for verification. The resulting parameters are as follows.

Estimation using own Levenberg–Marquardt algorithm has produced the following parameters.

$$m_0 \approx 3500kg, m_1 \approx 700kg, m_2 \approx 350kg, m_3 \approx 450kg$$

These figures were validated using the parameter estimation toolbox.

$$m_0 \approx 3850kg, m_1 \approx 450kg, m_2 \approx 250kg, m_3 \approx 450kg$$

The estimation suffers from large deviations of the mass parameters. This is probably caused by the high sensitivity of the model to noise and measurement error.

4 Mixed reality application development

This chapter describes the development of the mixed-reality (MR) application. It begins by defining the system specifications that will guide each subsequent task. Establishing this high-level overview up front ensures that all foundational tasks are aligned and completed in the correct order.

One of the core challenges is close to real-time video streaming from four industrial digital cameras mounted on the excavator. Addressing this requires choosing efficient video compression algorithms, setting up a low-latency streaming server, and integrating a client capable of receiving and rendering the live feeds on virtual displays within the MR headset.

In parallel, the excavator's operational state should be visualized through an animated 3D model. First, a mathematical kinematic model of the excavator is derived to capture its motions accurately. Next, an optimized 3D representation is created using specialized modeling software. This model is then imported into the MR application and animated using real sensor data that is received through the wireless connection.

Finally, to enhance usability, a human interface panel is implemented, providing intuitive controls and feedback to optimize the operator's experience.

Figure 4.1: Meta Quest 3 [18]

4.1 Specification of application scope

- Four video streams with minimal latency
- 3D manipulator integration
- Overview of the states of the excavator
- Good user experience

4.2 System design and architecture

4.2.1 Overall system architecture

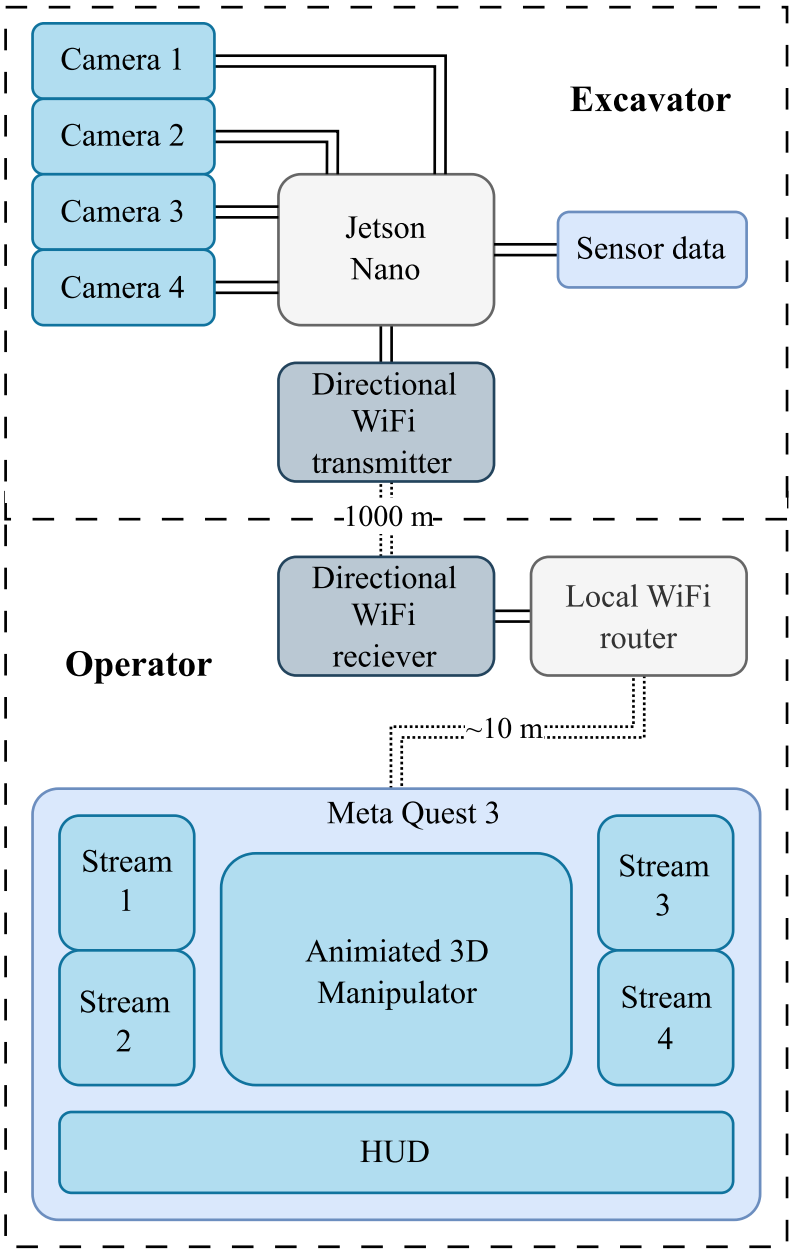
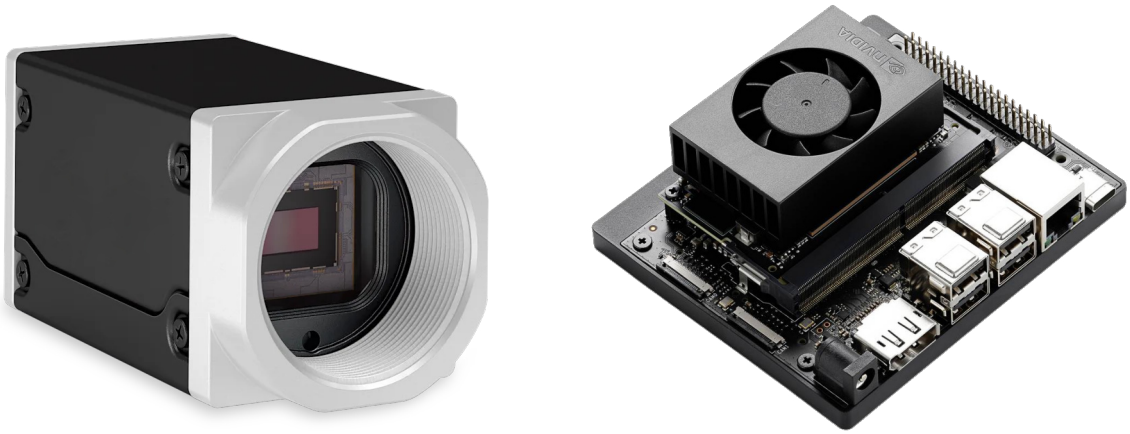


Figure 4.2: Overall system architecture

The architecture was specifically designed to enable data acquisition from at least four industrial cameras, internal sensors and gauges, and specialized sensors for measuring the position of a lifting platform. Although sampling frequencies are not explicitly defined, it is recommended to target rates exceeding 30 FPS for the cameras and 100 Hz for the other sensors to ensure accurate data capture.

The architecture also factors in data transmission over distances of up to a kilometer, while maintaining minimal latency. Additionally, it supports the visualization of video and sensor data through an immersive reality headset.

The selected cameras are industrial-grade variants from Basler [54]. According to the specifications, these cameras offer Full HD resolution at up to 30 FPS. Selecting higher resolutions would be unnecessary and could introduce increased system latency. Basler provides various configurations, including different connection types, sensor sizes, and sensor types (infrared or color).



(a) Basler [54] (b) Jetson [55]

Figure 4.3: Basler camera body, Jetson Nano

The internal and external sensors consist of position sensors in the excavator's legs, an oil temperature sensor, a hydraulic fluid pressure sensor, ultrasonic distance sensors, and IMUs mounted on the lift platform.

All inputs are collected via a Data Acquisition (DAQ) system and transmitted to a single-board computer. The chosen processing unit is NVIDIA's Jetson Nano 4.3b, which features four USB 3.0 Type-A ports suitable for connecting Basler cameras [56]. It also includes a Gigabit Ethernet port utilized for communication with a directional Wi-Fi antenna.

The directional Wi-Fi setup enables bidirectional communication between the Jetson Nano and the Mixed Reality headset. Data transmission from the excavator to the headset benefits from a relatively wide transmission field, whereas data sent from the headset to the excavator requires a more directional approach.

For the immersive reality headset, the Meta Quest 3 [18] was selected (4.1). Among available platforms, this headset offers an optimal balance of cost, performance, and ease of development.

4.2.2 MR googles design environment

Research has identified several methods for programming and developing applications for mixed reality headsets. The ideal approach to implement the required specifications involves using the OpenXR API [42] in conjunction with Android-compatible OpenCV [50]. This combination offers several advantages, including complete control over the programmed device, the capability to perform client-side ArUco pose estimation, and the opportunity to gain experience in low-level C++ mobile programming. However, this approach poses a steep learning curve for beginners, as it requires advanced knowledge of C++ and the OpenGL Shading Language (GLSL). The challenges associated with GLSL can be mitigated by employing game engines such as Unreal Engine 5 (UE5), Unity, or Godot. However, using these engines limits the ability to integrate OpenCV and handle video streams without reverting to low-level programming [31],[32],[33].

Due to these constraints and the selected hardware, the chosen design environments are the Meta Spatial SDK, Meta Spatial Studio, and Android Studio. The Spatial SDK utilizes Kotlin, a modern, high-level programming language developed by JetBrains, which is fully compatible with Java as it runs on the Java Virtual Machine [57], [58].

Android Studio serves both as the programming environment and as the design platform for the application's interface [59].

Meta Spatial editor enables the integration of assets into the MR environment without the need for custom shaders. Additionally, the studio provides a preview of the application and its assets, as illustrated in Figure 4.4.

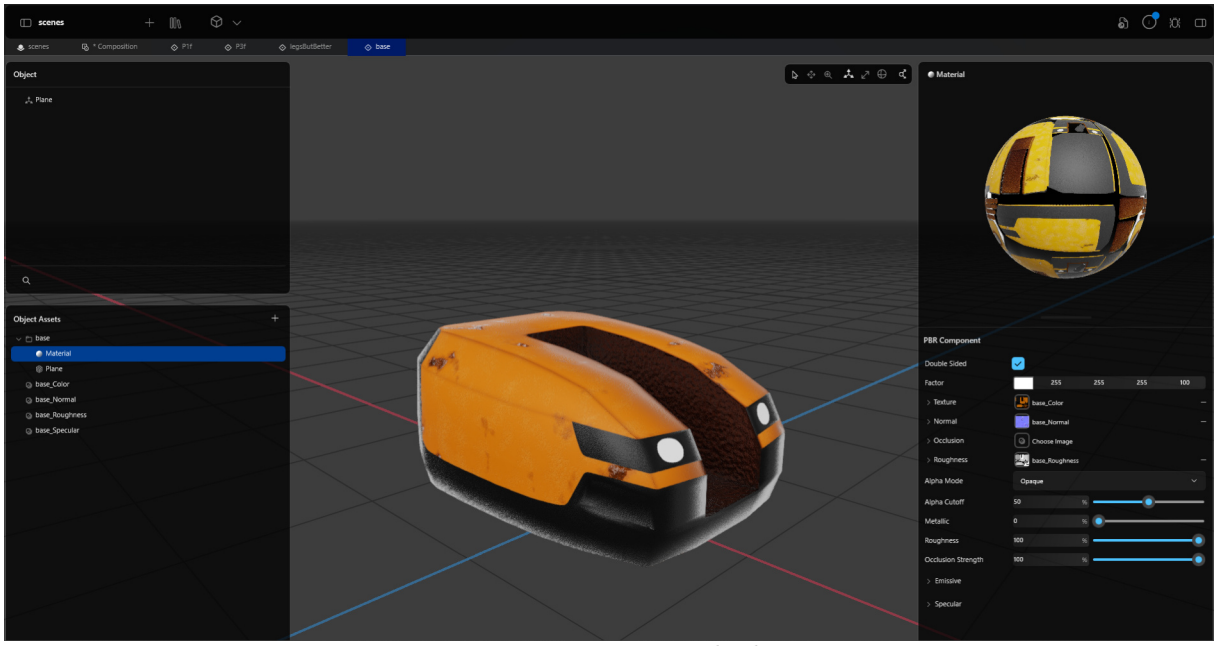


Figure 4.4: Meta spatial editor

Additional environments

Given the selected workflow, integrating machine vision directly within the goggles is not achievable at the time of development. Machine vision processing is conducted remotely on the excavator using Jetson Nano depicted in the diagram 4.2. Marker detection has been implemented in both C++ and Python, allowing straightforward deployment [56].

4.3 Video stream and data transfer implementation

To achieve data transfer from the Jetson Nano to the MR environment, a WiFi directional antenna was used. Selecting an appropriate protocol is important to ensure robust transmission with minimal latency. Additionally, suitable colour and compression formats must be utilized to ensure data usability. A simplified overview of the video and data handling process is illustrated in the diagram below.

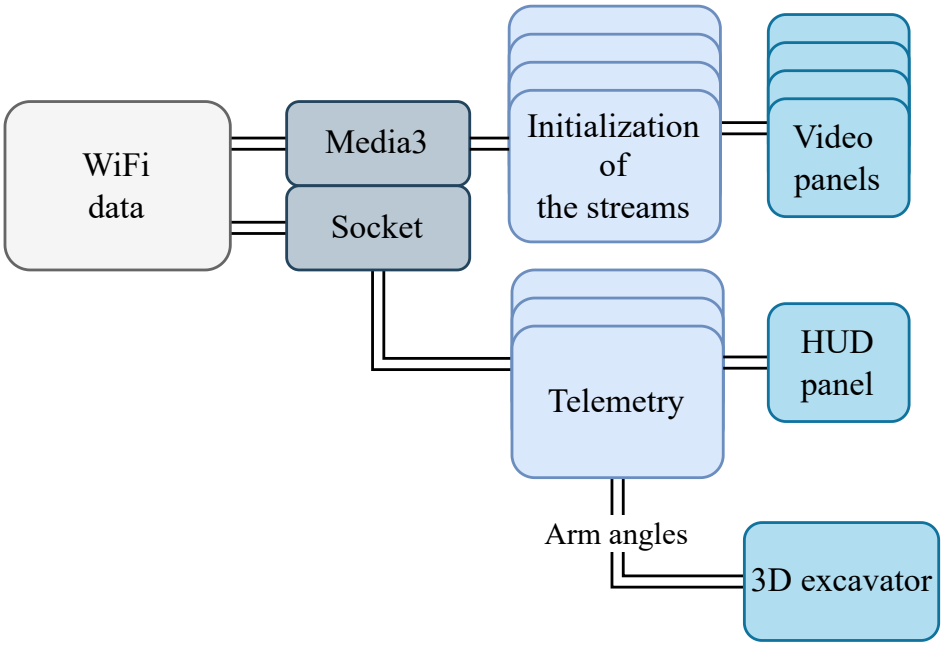


Figure 4.5: Video stream setup

4.3.1 Client side video and data handling

The Meta Quest 3 headset runs on a specialized operating system, with Android OS at its core. Android provides various extensions to streamline development, one of which is Media3 ExoPlayer. This component enables applications to receive video data in multiple media formats. The table below compares these formats in terms of latency.

Media type	Typical latency	Live offset tracking
DASH	10 - 30	Yes
DASH - LL	3 - 10	Yes
HLS	20 - 30	Yes
HLS - LL	2 - 8	Yes
SS	25 - 30	Yes
SS - LL	2 - 3	Yes
RTSP	∼2	No
UDP	2 - 6	No

Table 4.1: Latency comparison

As shown in Table 4.1, many sources report video feed latencies exceeding two seconds [60],[61],[62]. For this application, latencies above 500 milliseconds are nearly unusable. To achieve lower latencies, compromises in image quality, compression ratio, and resolution are necessary.

Supported video formats include H.263, H.264, H.265, and MPEG-4. Android supports various container formats; however, Meta specifies that the .mp4 container is required. The recommended frame rate ranges between 30 and 60 fps. The stream's pixel format must be YUV 420p to ensure proper image rendering [63],[64]. These specifications are important when configuring the server. To implement video streaming to the MR goggles, the necessary dependencies were defined in the app-level `build.gradle.kts file`.

```
implementation("androidx.media3:media3-exoplayer:1.6.1")
implementation("androidx.media3:media3-ui:1.6.1")


implementation("androidx.media3:media3-exoplayer-rtsp:1.6.1") //for RTSP
implementation("androidx.media3:media3-exoplayer-hls:1.6.1") // for HLS
implementation("androidx.media3:media3-exoplayer-dash:1.6.1") // for DASH
```

Subsequently, these packages were imported into the main program file, alongside additional functionalities.

```
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.ui.PlayerView
import androidx.media3.common.MediaItem
import androidx.media3.common.Player
import androidx.media3.common.util.Log
```

After all dependencies were met, a property of type ExoPlayer was created. This property was declared as lateinit, which means that the creation of the object instance is yet to be performed. Since the property was instantiated elsewhere, it was declared as a non-nullable var type. The object was instantiated within the onCreate method, a member function of the main activity class. The complete object creation and basic usage are demonstrated in the following code snippet.

```
private lateinit var objPlayer: ExoPlayer
...
override fun onCreate(savedInstanceState: Bundle?) {
    objPlayer = ExoPlayer.Builder(baseContext).build()

    val mediaItem = ...
       ... MediaItem.Builder()
            .setUri("streamType://123.456.0.789:1234/live")

    objPlayer.setMediaItem(mediaItem)
    objPlayer.prepare()
    objPlayer.play()
}
```

In this example, mediaItem is a non-nullable, read-only instance of the MediaItem class, encapsulating the specified stream URI. This mediaItem is assigned to the objPlayer instance, as a carrier of the setup data, which then prepares and initiates playback.

Depending on the specific media type and usage requirements, objPlayer and mediaItem can be configured with various parameters, such as MIME types, DRM settings, or additional metadata, to accommodate different streaming scenarios. To collect time-sensitive data from both the excavator and the Jetson device, fast data communication is required. Utilizing JavaScript Object Notation (JSON) in connection with WebSocket technology allows for efficient and structured transfer of multiple variables.

Establishing a WebSocket connection on the client involves a process similar to setting up the video stream. This begins with adding the necessary dependency in the `build.gradle.kts` file.

```
implementation("io.socket:socket.io-client:2.0.0")
```

Subsequently, the required classes are imported into the main Kotlin file.

```
import io.socket.client.IO
import io.socket.client.Socket
import io.socket.emitter.Emitter
import org.json.JSONObject
```

As before, a property is declared and instantiated in the onCreate method. To ensure that a new WebSocket is created every time, options carrier object is created with a forceNew parameter set to true. The address of the server and the options are then passed to the socket class instantiating the mySocket object. By setting mySocket.on(attribute,fcn) a function can be called during runtime when the attribute is received. It is possible to set the attribute server-side, so every time the "update" string is sent, the onUpdate function is called.

```
private lateinit var mSocket: Socket
...
override fun onCreate(savedInstanceState: Bundle?) {
    val opts = IO.Options().apply {forceNew = true}
    mySocket = IO.socket("http://123.456.0.789:1234", opts)
    mySocket.on(Socket.EVENT_CONNECT, onConnect)
    mySocket.on("update", onUpdate)
    mySocket.connect()
}
```

onConnect function serves as a helper function for debugging and providing contextual feedback to the developer and the end user. onUpdate function updates transferred variables in a specific global variable. This variable is packaged in an object visible in the code snippet.

```
object GlobalData {var numbers = NumberData(0.0, 0.0, 0.0, 0.0)}
```

The onUpdate function is structured as a lambda expression that is invoked whenever a specific event fires, which in this case is the socket event with the string "update". The function first checks that the argument is not empty and that it is a JSON object. If these conditions are met, the data in the global variable is replaced with the data from the JSON object after converting the values to doubles.

```
private val onUpdate = Emitter.Listener { argument ->
    if (argument.isNotEmpty() && argument[0] is JSONObject) {
        val data = argument[0] as JSONObject
        // Parse the JSON data and update the global variable
        GlobalData.numbers = NumberData(
            var1 = data.optDouble("var1"), ... , ... ,
            var4 = data.optDouble("var4")
        )
    }
}
```

After completing the setup, various media types were evaluated. The first protocol examined was RTSP/RTP, primarily because, according to Table 4.1, it is expected to provide the lowest average latency. However, testing revealed that the system's latency consistently exceeded four seconds, rendering it unusable for this application. This high latency was partially due to Media3's RTSP implementation missing support for live offset tracking.

In contrast, DASH and HLS protocols support live offset tracking, allowing latency optimization using functions seekTo 4.6. This function gradually reduces latency by adjusting the playback speed. Utilizing this approach decreased overall latency to approximately 800 ms, which is barely acceptable.

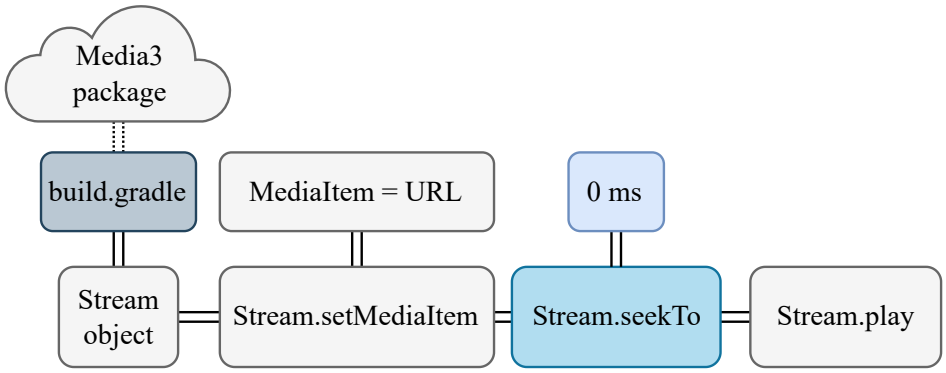


Figure 4.6: latency minimiyation using seekTo

As a final option, streaming via UDP without TCP fallback was explored. Disabling TCP fallback resulted in data transmission, susceptible to packet loss and corruption. However, this approach achieved lower latency than HLS. By implementing server-side optimizations, such as reducing buffer sizes and discarding late frames, the final latency was reduced to approximately 200 ms. This performance is acceptable, provided that occasional image corruption due to packet loss is tolerable. The code included an additional loadControl parameter, and specific configurations to the mediaItem for further improvements.

```
val loadControl = DefaultLoadControl.Builder().setBufferDurationsMs(...
    ... 50, 100, 0, 0).build()

val mediaItem = MediaItem.Builder().setUri("udp://@123.456.0.789:1234")
    .setLiveConfiguration(MediaItem.LiveConfiguration.Builder()
    .setTargetOffsetMs(0).setMinOffsetMs(0).setMaxOffsetMs(150).build())
    .build()
objPlayer = ExoPlayer.Builder(this).setLoadControl(loadControl).build()

objPlayer.setPlaybackSpeed(1.05F)
```

4.3.2 Serverside video and data handling

The video data needs to be transmitted to be received. This requires a server that is able to provide the necessary data. For modern protocols like Low-Latency HLS (LL-HLS), a straightforward and robust tool such as MediaMTX can be employed [65]. MediaMTX functions as a media server or proxy, supporting various streaming protocols including SRT, WebRTC, RTSP, RTMP, and LL-HLS. In the case of a UDP stream, an additional server is unnecessary. However, transcoding and format conversion are required.

To experiment with different settings before implementing the final code, FFmpeg can be utilized. It serves as an efficient prototyping tool for capturing video data from various sources, converting formats, and streaming. For instance, to capture media using FFmpeg, the following command can be used.

```
ffmpeg -f dshow -i video="Webcam"
```

This command captures the webcam video source available on a Windows machine. It's possible to specify the resolution, frame rate, and other parameters of the input device using specific commands.

```
-video_size 1920x1080 -framerate 30
```

While setting resolution and frame rate is beneficial, the primary focus of this thesis is minimizing latency. To configure the server for minimal latency, the following adjustments should be applied.

```
-c:v libx264 -preset ultrafast -tune zerolatency -pix_fmt yuv420p
```

This configuration sets the encoding type and speed, targets zero latency, and specifies the video format. The encoding speed depends on the specific hardware capabilities. If the hardware is powerful, slower encoding can be used to produce smaller file sizes. Conversely, if the hardware has limited encoding capabilities, the ultrafast setting is preferable to prevent the hardware from being the system's limitation.

If network bandwidth is the main source of latency, the amount of transmitted data can be adjusted by setting target and maximum bit rates alongside the buffer size.

```
-b:v 500k -maxrate 500k -bufsize 1000k
```

The final step involves setting up the output by specifying the data wrapper type and the output address. In this case, the MPEG transport stream and a UDP address are used.

```
-f mpegts udp://123.456.0.789:1234
```

4.4 User interface and visualization

Developing an application in Android Studio includes backend programming, typically written in Java or Kotlin, and frontend design using XML markup language. Android Studio also provides a graphical user interface that provides automatic code generation, streamlining the development process [59].

4.4.1 Graphical implementation of video stream

The Meta Spatial SDK offers functionality to create instances of 3D spatial panels. These panels function similarly to mobile phone screens. They are fully interactive and can be manipulated using hand gestures or controllers. However, utilizing them requires a specific approach [57].

To instantiate a panel, it is possible to use the Meta Spatial Editor, a graphical interface, to spawn the panel. For greater control, panels can be created in program using Kotlin code, as demonstrated in the following example [58].

```
panelEntity = Entity.createPanelEntity(
    R.layout.PlayerView,
    Transform(Pose(Vector3(0f, 0f, 0f), Quaternion(0f, 0f, 0f))),
    Grabbable(true),
)
```

This code snippet creates a panel entity using the XML layout named player_view_layout. The panel is positioned at the origin of the global coordinate system with default rotation. The Grabbable(true) parameter enables the panel to be movable via controller input.

To make the panel interactive and visible within the 3D scene, it must be registered using the registration process. It associates the panel with a unique identifier and ensures it is properly integrated into the scene. An illustration of this registration process is shown in Figure 4.7.

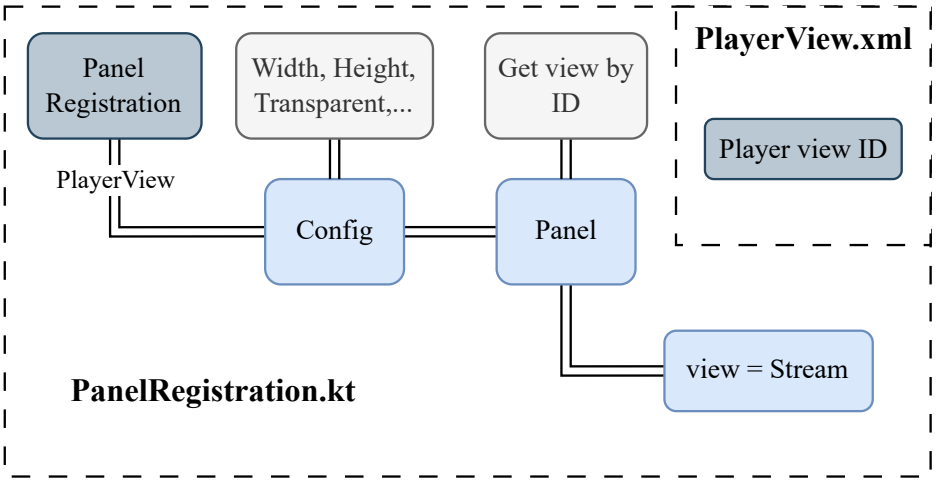


Figure 4.7: Panel registration

To register the panel, the precompiled PanelRegistration object from the Meta Spatial SDK is used. A registration ID must be provided to associate the object with the corresponding layout file [57].

The panel registration includes configuration attributes such as width, height, and transparency. To pass the video object, the panel argument is employed. The following snippet demonstrates the panel registration:

```
PanelRegistration(R.layout.player_view_layout) {
    config {
        width = 1920
        height = 1080
        enableTransparent = false
    }
    panel {
        val playerView = ...
            ...rootView?.findViewById<PlayerView>(R.id.player_view)
        playerView?.player = objPlayer
    }
}
```

In this snippet, the PanelRegistration associates the player_view_layout with specific configuration attributes such as width, height, and transparency. Inside the panel block, the PlayerView component is retrieved and linked to the objPlayer instance, enabling video playback functionality. The PlayerView component is defined in the XML layout as follows.

```
<androidx.media3.ui.PlayerView
    android:id="@+id/player_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:surface_type="surface_view"
/>
```

Layout PlayerView specifies the maximum width and height by the parent layout and assigns the surface type for the video rendering.

4.4.2 User interface layout

The application requires a graphical user interface (GUI) that displays key variables and the states of the excavator (4.8). All important data are positioned on the right side of the GUI. This includes the individual angles of the excavator arm, essential telemetry data, the lifting platform angle, and stream latency. Using this information and previously estimated masses, the center of mass (COM) is calculated and visualized in the bottom-center area of the GUI. The COM calculation also considers the influence of the lifting platform angle.

The GUI includes slider inputs for rotating and scaling the simulated excavator, buttons for enabling or disabling passthrough, and for switching between stationary and attached modes of the GUI. The online tool Photopea was used for graphical aesthetics [66].

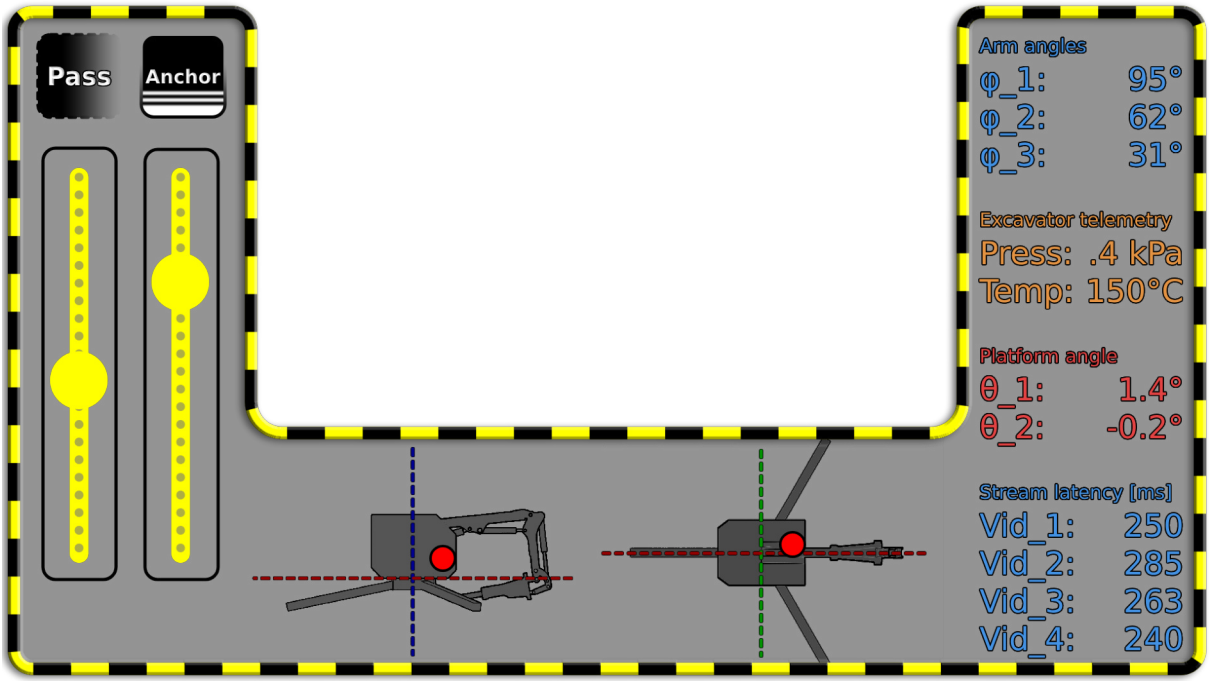


Figure 4.8: Graphical user interface

When the attached mode is selected, the panel entity updates its position based on the headset's pose. The transformation refresh rate must not exceed approximately 120 Hz, as higher rates can cause the application to crash. The following code snippet demonstrates the process of retrieving and applying the viewer pose:

```
val transform = panelEntity?.getComponent<Transform>()
val viewerPose = scene.getViewerPose()

... //pose update

panelEntity?.let{panel -> panel.setComponent(Transform(updatedPose))}
```

4.5 System latency

Due to the choice of a UDP video stream, there is no built-in method to measure latency directly. Additionally, the Meta Quest 3 operating system does not support synchronized wall-time clocks over the network, making accurate time alignment more difficult. To overcome this, a simple measurement method was devised, as illustrated in Figure 4.9. The server encodes a timestamp into the video stream's visuals and simultaneously sends the same timestamp over a WebSocket connection. By visually comparing these two timestamps, the latency can be determined.

Additionally, the client sends a ping message to the server every ten milliseconds, including a nanosecond counter that begins at the client's startup. Upon receiving the ping, the server immediately sends the value back. Once the client receives this "pong" response, it subtracts the original timestamp from the current time and divides the result by two to estimate the WebSocket network delay.

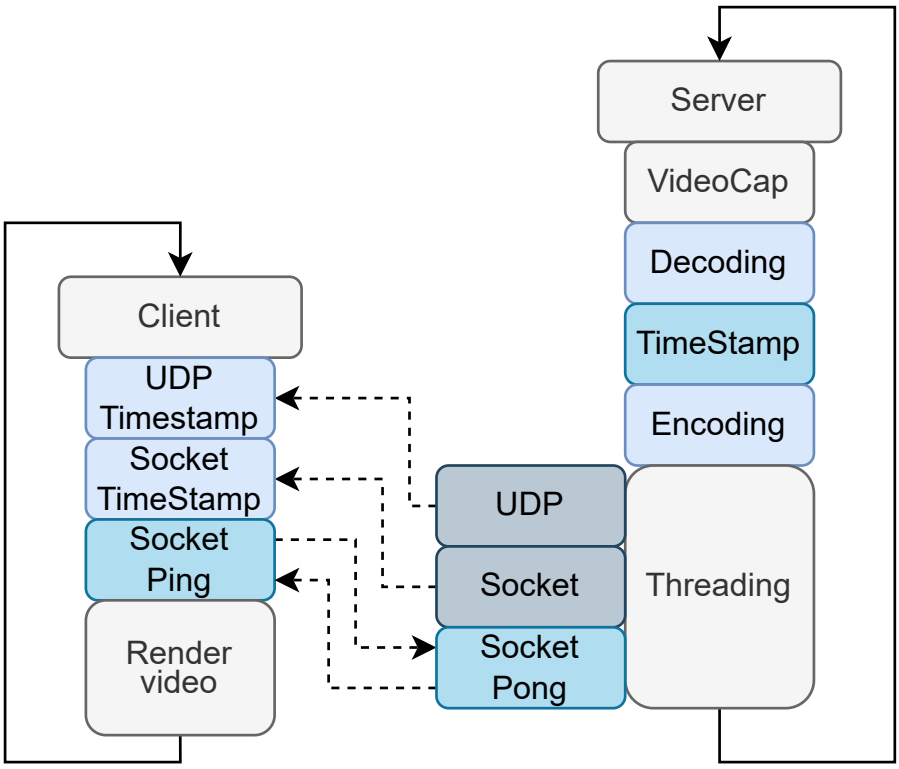


Figure 4.9: Sequence of the network latency measurement

Fortunately, both the server and client support measuring decoding, encoding, and rendering times. This enables a full assessment of system latency. However, the latency introduced by the video capture device remains unknown due to the camera's construction. However, the video capture latency was measured using a high-speed camera, providing an estimate of this value.

The network latency was also measured with varying distances or signal strengths. Interestingly, the median network latency appeared to be higher when the receiver was close to the source WiFi transmitter. The optimal range, where latency was lowest, seems to lie between -40 dBm and -70 dBm. Although the latency remained approximately consistent across the entire range, it became noticeably less stable near the maximum distance, as shown in figure 4.10.

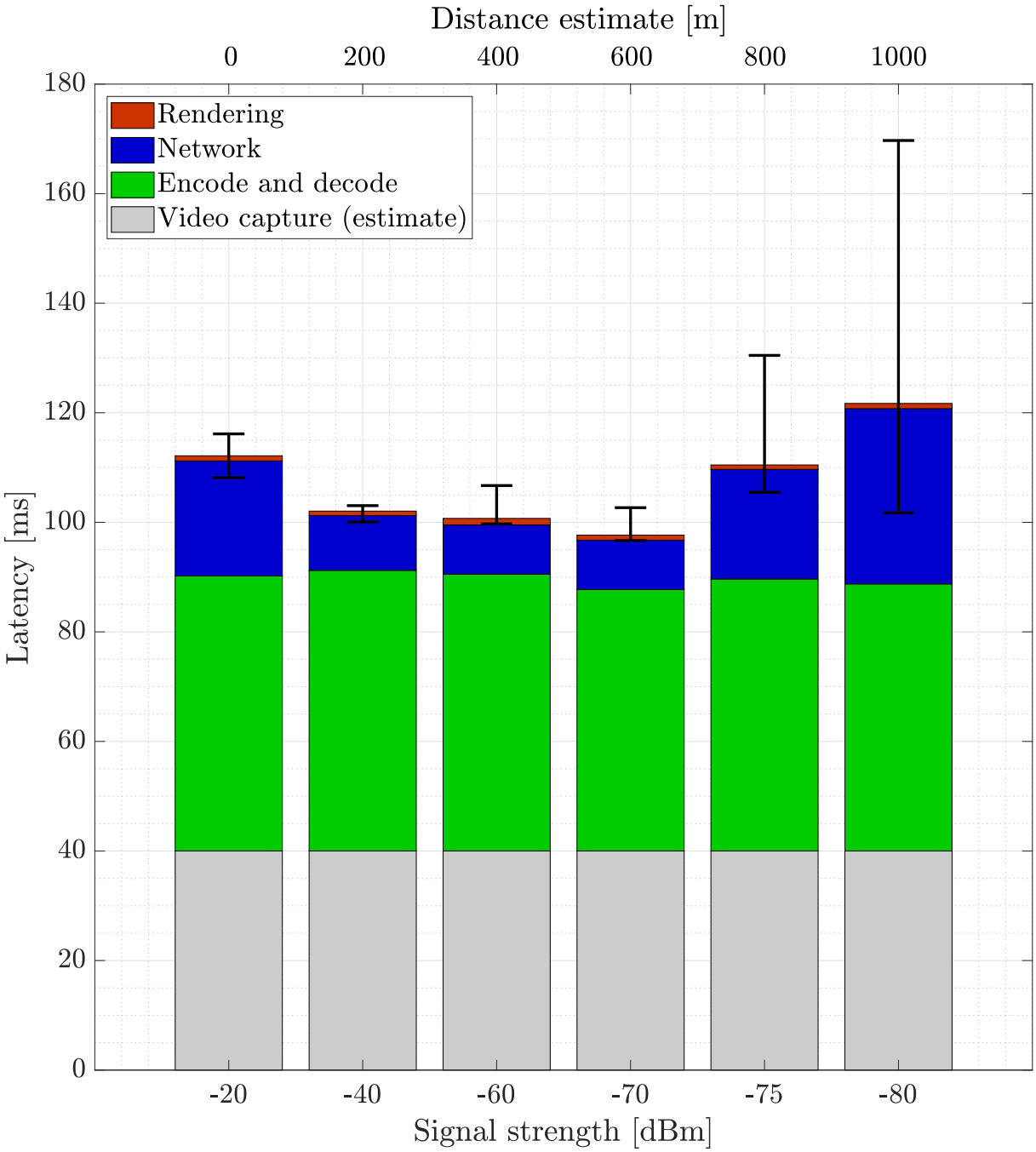


Figure 4.10: Latency with distance

4.6 Excavator kinematic model implementation

As previously stated, implementing a 3D simulation of the excavator in the MR headset can considerably assist the operator with spatial orientation in the demolition environment. The implementation can be divided into two main tasks. Mathematical derivation of the manipulator's kinematics, and creation of a game-optimized 3D model.

4.6.1 Analytical solution

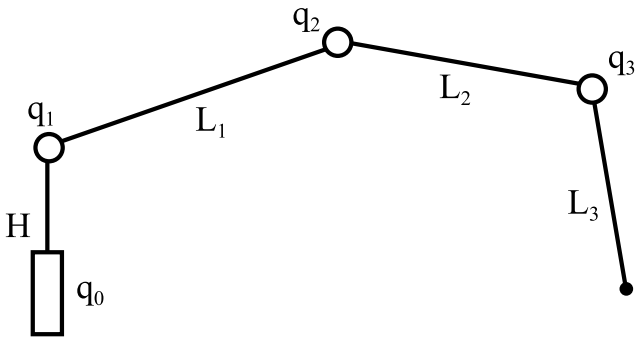


Figure 4.11: Simple manipulator

The simplified arm of the excavator qualifies as an open-chain manipulator with three degrees of freedom (4.11). Since the individual joint angles are known, the position of the end effector can be expressed using the following formula.

$$\mathbf{u}_{eff} = \mathbf{R}_{q_0,q_{eff}} \mathbf{T}_{o,eff} \mathbf{u}_0 \tag{4.1}$$

However, for visualization purposes, all intermediate joint translations and rotations must be calculated. This leads to the following.

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{R}_{q_0} \mathbf{T}_H \mathbf{u}_0 \\ \mathbf{u}_2 &= \mathbf{R}_{q_0} \mathbf{T}_H \mathbf{R}_{q_1} \mathbf{T}_{L_1} \mathbf{u}_0 \\ \mathbf{u}_3 &= \mathbf{R}_{q_0} \mathbf{T}_H \mathbf{R}_{q_1} \mathbf{T}_{L_1} \mathbf{R}_{q_2} \mathbf{T}_{\mathbf{L}_2} \mathbf{u}_0 \\ \mathbf{u}_{eff} &= \mathbf{R}_{q_0} \mathbf{T}_H \mathbf{R}_{q_1} \mathbf{T}_{L_1} \mathbf{R}_{q_2} \mathbf{T}_{L_2} \mathbf{R}_{q_3} \mathbf{T}_{L_3} \mathbf{u}_0 \end{aligned} \tag{4.2}$$

The corresponding rotation matrices are:

$$\begin{aligned} \mathbf{r}_1 &= \mathbf{R}_{q_0} \\ \mathbf{r}_2 &= \mathbf{R}_{q_0} \mathbf{R}_{q_1} \\ \mathbf{r}_3 &= \mathbf{R}_{q_0} \mathbf{R}_{q_1} \mathbf{R}_{q_2} \\ \mathbf{r}_{eff} &= \mathbf{R}_{q_0} \mathbf{R}_{q_1} \mathbf{R}_{q_2} \mathbf{R}_{q_3} \end{aligned} \tag{4.3}$$

Implementing this simplified manipulator model is straightforward. However, a more accurate representation, visible in the figure (4.12), requires additional derivations.

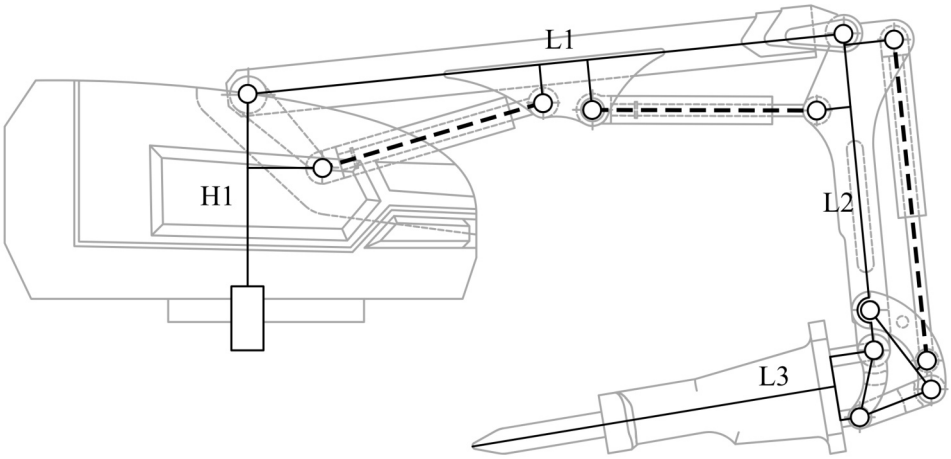


Figure 4.12: More accurate manipulator

To determine the joint coordinates of the pistons and the effector mechanism, several trigonometric functions must be solved. The entire first section of the excavator shown in figure (4.13) needs to be analyzed to compute the rotations of the first piston base and its follower.

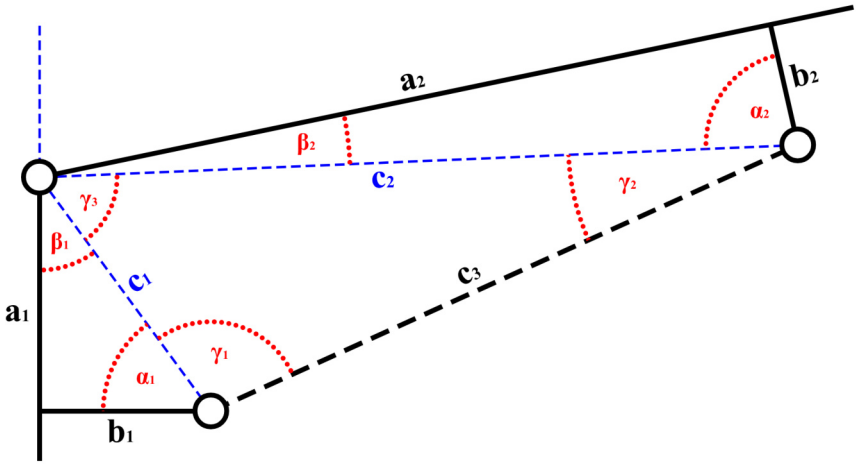


Figure 4.13: Detail at the first portion of manipulator

The rotation angle θ_1 is given by:

$$\theta_1 = \alpha_1 + \gamma_1 - \frac{\pi}{2} \tag{4.4}$$

where

$$\alpha_1 = arctan\left(|a_1|, |b_1|\right) \tag{4.5}$$

a and b are constants.

γ_1 can be derived from the reorganized rule of cosines.

$$\gamma_1 = \arccos\left(\frac{c_1^2 + c_3^2 - c_2^2}{2c_1c_3}\right) \tag{4.6}$$

Where c_1, c_2 and c_3 are.

$$c_1 = \sqrt{a_1^2 + b_1^2} \tag{4.7}$$

$$c_2 = \sqrt{a_2^2 + b_2^2} \tag{4.8}$$

$$c_3 = \sqrt{c_1^2 + c_2^2 - 2c_1c_2\cos\gamma_3} \tag{4.9}$$

To calculate γ_3, two helper angles β_1 and β_2 are required.

$$\beta_1 = \frac{\pi}{2} - \alpha_1 \tag{4.10}$$

$$\beta_2 = \frac{\pi}{2} - \alpha_2 \tag{4.11}$$

Finally γ_3 can be calculated using.

$$\gamma_3 = \pi - \varphi_1 - \beta_1 - \beta_2 \tag{4.12}$$

Follower of the first piston can be calculated using the following equation.

$$\theta_2 = \frac{3}{2}\pi - \alpha_2 + \gamma_2 \tag{4.13}$$

Angle α_2 is computed similarly to equation 4.5, but with a_2 and b_2 as inputs. The angle γ_2 is derived by rearranging equation 4.14.

$$\gamma_2 = \arccos\left(\frac{c_2^2 + c_3^2 - c_1^2}{2c_2c_3}\right) \tag{4.14}$$

The same equations can be applied to calculate the motion of the second and third pistons. Finally, the transformation of the end-effector must be described. This transformation can be performed either using inverse kinematics during runtime or by deriving an analytical solution for the mechanism's kinematics. Assuming that the motion of the effector is constrained between 0 and π, an analytical solution for the four-bar mechanism of the effector can be derived.

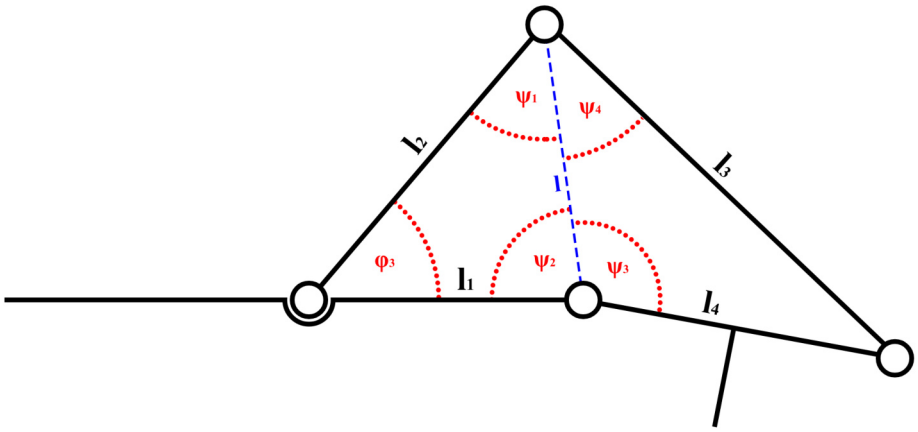


Figure 4.14: Detail at effector mechanism

Given the known link lengths l_1, l_2, l_3, l_4 and the measured angle φ_3, the length l of the diagonal linkage connecting opposite joints in the four-bar mechanism can be calculated.

$$l = \sqrt{l_1^2 + l_2^2 - 2l_1 l_2 \cos(\varphi_3)} \tag{4.15}$$

Knowing that the first link is rotated by angle φ_3, and having all the link lengths, the rotation of the second link can be obtained using.

$$\delta_1 = \pi - \psi_1 - \psi_4 \tag{4.16}$$

Where ψ_1 and ψ_2 are.

$$\psi_1 = \arccos\left(\frac{l^2 + l_2^2 - l_1^2}{2ll_2}\right) \tag{4.17}$$

$$\psi_4 = \arccos\left(\frac{l^2 + l_3^2 - l_4^2}{2ll_3}\right) \tag{4.18}$$

Using Equations 4.17 and 4.19, the remaining angles required to determine the rotation of the effector can be calculated. The second rotation angle is:

$$\delta_2 = \psi_2 - \psi_3 \tag{4.19}$$

4.6.2 Modeling of an excavator

The excavator was recreated using the open-source all-in-one 3D modeling software Blender. This software offers a wide range of tools for creating game-ready assets. The basic workflow begins with preparing a 2D sketch, blueprint, or photograph. These 2D representations can be placed in multiple viewports as references [34].

3D modeling typically starts with the construction of simple shapes. These shapes are then improved by adding finer details, often through mesh sculpting. While sculpting is commonly used for organic forms, it can also be applied to technical objects.

The detailed 3D model is not suitable as a game asset because it usually contains millions of polygons. To optimize the model, it must be re-meshed, either automatically or manually. Automatic re-meshing can lead to lighting artifacts, so manual re-meshing is often preferred.

Since re-meshing significantly reduces detail, a method for detail preservation is required. A high-polygon mesh is used to generate a normal map, a texture that stores surface detail through image data. This approach enables much higher visual quality than what the simplified mesh geometry would allow.

In addition to geometry, the model includes various colors and material properties, which are stored in texture maps. The albedo map contains the base color of the object, either with or without indirect lighting. The roughness map defines the amount of surface reflectivity. The specular map specifies the shininess of highlights on the surface. The textures are visible in the figure (4.15). To generate these textures, the 3D object must be projected onto a 2D plane with a process known as UV unwrapping.

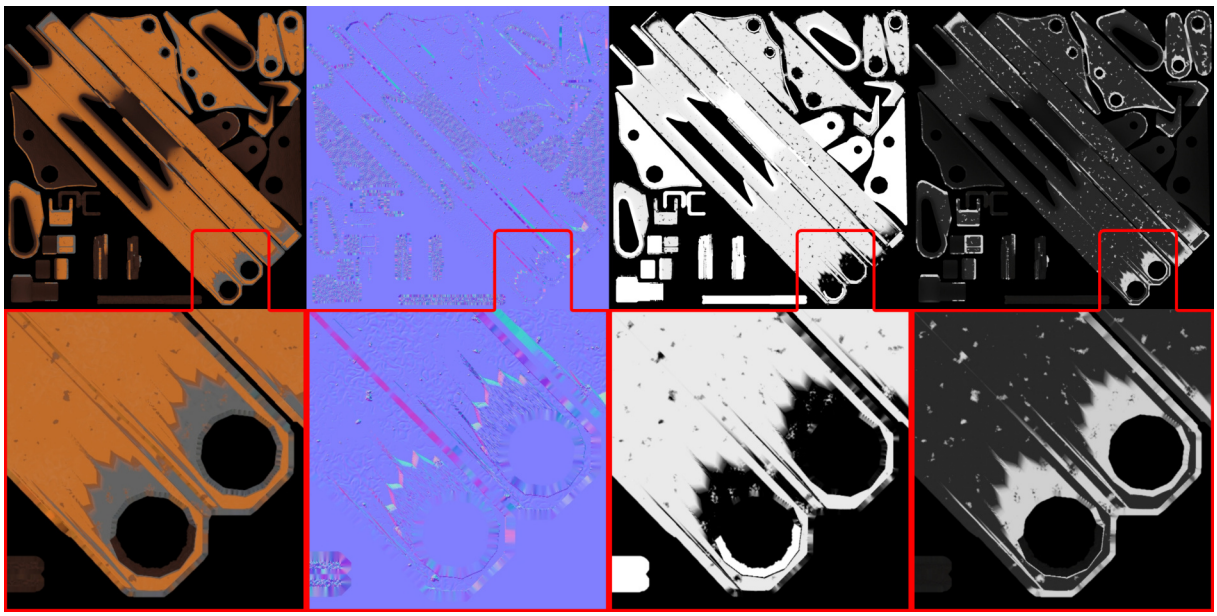


Figure 4.15: Textures for the arm segment

The origin of each object was placed at the joint, and all objects were initially positioned at the global coordinate system's origin. This setup enables the use of rotation matrices for subsequent positioning and animation. The final models are shown in Figure 4.16.

Figure 4.16: Individual objects

4.6.3 Implementation and animation

The finalized models were exported in GLTF format, and the textures in PNG format, which supports alpha blending. Importing everything into the Meta Spatial Editor is handled via a single import button. After assigning the textures, the scene with the models is ready for programming. To enable movement of individual objects, the meshes must be saved into an entity object. The following code demonstrates this workflow.

```
private var arm3: Entity? = null
arm3 = composition.getNodeByName("R3").entity
```

To enable the use of the kinematic model, several helper functions were implemented. These functions create four-by-four identity, rotation, and translation matrices.

```
fun Ident(): Matrix44{...}
fun Ry(): Matrix44{...}
fun Rz(): Matrix44{...}
fun Ty(): Matrix44{...}
fun Tx(): Matrix44{...}
```

Using these helper functions, it is possible to exactly recreate the kinematic model described by equations 4.2 and 4.3. The implementation of the remaining computations follows a analogous approach.

```
//-----------------Arm 3 translation---------------
myMatrix = Ident() *
        Ty(h+H) *
        Ry(GlobalData.numbers.var4.toFloat()) *
        Rz(phi1) *
        Ty(L1) *
        Rz(phi2) *
        Ty(L22)
myVec = myMatrix * Vector4(0.0F, 0.0F, 0.0F, 1.0F)
//-----------------Arm 3 rotation---------------
myMatrix = Ident() *
        Rz(delta2 - PI.toFloat()) *
        Rz(phi2) *
        Rz(phi1) *
        Ry(GlobalData.numbers.var4.toFloat())
```

The Pose of the arm3 entity must be set using a translation vector and a quaternion representing the rotation. The quaternion can be derived from the rotation matrix. The following code demonstrates how to apply the transformation.

```
arm3?.setComponent(
    Transform(
        Pose(
            Vector3(-myVec.x, myVec.y, myVec.z),
            myMatrix.decompose().first.q
        )
    )
)
```

These calculations are encapsulated in the Fkine function, which is called every time there is a change in joint angle variables.

5 Conclusion

The goal of this thesis was to develop a mixed reality (MR) application capable of visualizing surroundings, center of mass, and the internal states of an excavator operating in a demolition environment.

To begin, research into immersive technologies was conducted. It provided an understanding of various commercially available devices. These were compared based on optical and display quality, computational capabilities and software support. The chosen headset, **Meta Quest 3**, offered a good compromise between cost and capabilities, broad developer support, and an accessible SDK.

One of the objectives of this thesis was to develop a measurement methodology for acquiring the **center of mass** of the excavator. This involved the design and fabrication of a force gauge capable of measuring a five-ton excavator, as well as the creation of a C++ application for joint coordinate measurement using OpenCV's ArUco markers. Early testing confirmed successful data acquisition for both joint coordinates and force measurements. However, the subsequent mass estimation of the excavator showed low accuracy, likely due to measurement noise and sensor imprecision. Interestingly, the center of mass proved less sensitive to arm positioning than expected, reducing the impact of the estimation error.

The **MR application was developed** with a focus on integrating four parallel video streams with minimal latency. RTSP, HLS, and UDP transfer protocols were tested to identify the most effective streaming approach. RTSP proved to be insufficient due to high latency above two seconds and the inability to use live offset tracking. UDP streaming with minimal buffering yielded the lowest latency, achieving a minimum of 100 ms and an average of around 200 ms. Although HLS offered higher latency, it proved more robust and supported live offset tracking. UDP was chosen for its speed. In parallel, bidirectional data transfer using WebSockets was also implemented, achieving a latency of 20 ms.

Additionally, a GUI was implemented, providing the operator with critical information such as temperature, pressure, and the center of mass.

Using the open-source application Blender, a **3D model** of the excavator was created. A mathematical kinematic model was subsequently derived and implemented into the MR environment, allowing the virtual excavator to accurately mirror the movements of its real-world equivalent.

Testing on a real system was conducted. First, the accuracy of ArUco pose estimation was evaluated using various camera calibration procedures. Poor calibration methods resulted in high reprojection errors, which caused tracking failures. In contrast, effective calibration, requiring at least ten well-captured images, achieved reprojection errors of less than one pixel, ensuring reliable tracking. Secondly, network latency was measured in relation to the distance from the WiFi transmitter. While latency remained relatively consistent, connection stability declined once the signal strength dropped below -70 dBm.

5.1 Improvements

- Measurement:
  - Increase of measurement accuracy in order to improve mass estimation precision.
- Application:
  - Utilization of OpenXR CPP development.
  - Direct implementation of ArUco marker tracking within the MR headset.
  - Overall improvement of the application reliability.
- User feedback:
  - Transition to more intuitive controls using a physical controller instead of hand tracking.
  - Introduction of depth masking to improve visual realism in MR.
  - Addressing occasional jitter issues for a smoother user experience.

List of Abbreviations

2D Two Dimensional

3D Three Dimensional

API Application Programming Interface

AR Augmented Reality

COM Center Of Mass

CR Contrast Ratio

DAQ Data AcQuisition

DASH Dynamic Adaptive Streaming over HTTP

DRM Digital Rights Management

FEM Finite Element Method

FOV Field Of View

FPS Frames Per Second

GLSL Graphics library Shading Language

GUI Graphical User Interface

HLS HTTP Live Streaming

HMD Head Mounted Display

HTTP HyperText Transfer Protocol

HW HardWare

ID IDentifier

JSON JavaScript Object Notation

LCD Liquid Crystal Display

LiDAR Light Detection and Ranging

LL Low Latency

MEMS Micro Electro Mechanical System

MIME Multipurpose Internet Mail Extensions

MR Mixed Reality

OLED Organic Light Emitting Diode

OS Operating System

PC Personal Computer

PPD Pixels Per Degree

PPI Pixels Per Inch

RGB Red Green Blue (Color)

RTP Real Time Protocol

RTSP Real Time Streaming Protocol

SDK Software Development Kit

SNR Signal to Noise Ratio

SoC System on Chip

TCP Transmission Control Protocol

UDP User Datagram Protocol

URI Uniform Resource Idetifier

VR Virtual Reality

XML eXtensible Markup Language

XR eXtended Reality

References

[1] *bigscreenvr.com* [https://www.bigscreenvr.com/]. [N.d.]. [Accessed 21-05-2025].

[2] *MeganeX superlight 8K — en.shiftall.net* [https://en.shiftall.net/products/meganex8k]. [N.d.]. [Accessed 21-05-2025].

[3] *Virtual reality - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Virtual_reality]. [Accessed 30-04-2025].

[4] *Augmented reality - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Augmented_reality]. [Accessed 30-04-2025].

[5] RAGHAVAN, Rithesh. *What Is XR (Extended Reality) and Application of XR in Various Industries — acowebs.com* [https://acowebs.com/what-is-extended-reality-xr/]. [Accessed 30-04-2025].

[6] SPEICHER, Maximilian; HALL, Brian D.; NEBELING, Michael. What is Mixed Reality? In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, pp. 1–15. CHI '19. Available from DOI: 10.1145/3290605.3300767.

[7] JOYJAZ. *Microsoft HoloLens — learn.microsoft.com* [https://learn.microsoft.com/cs-cz/hololens/]. [N.d.]. [Accessed 21-05-2025].

[8] *Ray-Ban Meta Wayfarer AI Glasses and Sunglasses — meta.com* [https://www.meta.com/ai-glasses/wayfarer/]. [N.d.]. [Accessed 21-05-2025].

[9] *Spatial Anchors Overview — developers.meta.com* [https://developers.meta.com/horizon/documentation/unity/unity-spatial-anchors-overview/]. [N.d.]. [Accessed 21-05-2025].

[10] *Lens Material Properties - EyeWiki — eyewiki.org* [https://eyewiki.org/Lens_Material_Properties]. [N.d.]. [Accessed 21-05-2025].

[11] *3 Types Of VR Headset Lenses Explained – Which One Is Best For You? - Vskel.com — vskel.com* [https://vskel.com/3-types-of-vr-headset-lenses/]. [N.d.]. [Accessed 21-05-2025].

[12] *Fresnel lens - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Fresnel_lens]. [N.d.]. [Accessed 21-05-2025].

[13] PASCHOTTA, Dr. Rüdiger. *Fresnel lenses — rp-photonics.com* [https://www.rp-photonics.com/fresnel_lenses.html#Optical-Performance]. [N.d.]. [Accessed 21-05-2025].

[14] *Ray Optics Simulation - PhyDemo — phydemo.app* [https://phydemo.app/ray-optics/]. [N.d.]. [Accessed 21-05-2025].

[15] *Aspheric lens - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Aspheric_lens]. [N.d.]. [Accessed 21-05-2025].

[16] *The world's most advanced virtual and mixed reality devices. — varjo.com* [https://varjo.com/]. [N.d.]. [Accessed 21-05-2025].

[17] $\mathrm{SCT}_{A}DMIN$. *Looking Behind the Lens of Meta's VR Optics - Smart Cities Tech — smartcitiestech.io* [https://smartcitiestech.io/2022/06/looking-behind-the-lens-of-metas-vr-optics/]. [N.d.]. [Accessed 21-05-2025].

[18] *Meta - Shop MR, VR Headsets & AI Glasses — meta.com* [https://www.meta.com/quest/quest-3/]. [N.d.]. [Accessed 21-05-2025].

[19] LANG, Ben. *DigiLens is Developing a Waveguide Display for 150 Degree XR Headsets — roadtovr.com* [https://www.roadtovr.com/digilens-developing-150-degree-waveguide-display-for-ar-vr-xr-headsets/]. [N.d.]. [Accessed 21-05-2025].

[20] *Field of view - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Field_of_view]. [N.d.]. [Accessed 21-05-2025].

[21] MUSIL, Richard. *HMD Geometry Database — risa2000.github.io* [https://risa2000.github.io/hmdgdb/]. [N.d.]. [Accessed 21-05-2025].

[22] CHEN, Haiwei; TAN, Guanjun; WU, Shin-Tson. Ambient contrast ratio of LCDs and OLED displays. *Optics Express*. 2017, vol. 25, p. 33643. ISSN 1094-4087. Available from DOI: 10.1364/OE.25.033643.

[23] *MicroOLED and MicroLED: The Future of AR/VR Displays — displaydaily.com* [https://displaydaily.com/micro-oled-and-microled-the-future-of-ar-vr-displays/]. [N.d.]. [Accessed 22-05-2025].

[24] *VRcompare - The Internet's Largest VR & AR Headset Database — vr-compare.com* [https://vr-compare.com/]. [N.d.]. [Accessed 22-05-2025].

[25] LTD., Arm. *Building the Future of Computing — arm.com* [https://www.arm.com/]. [N.d.]. [Accessed 22-05-2025].

[26] *Qualcomm Snapdragon XR2 5G Platform — Qualcomm — qualcomm.com* [https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/snapdragon-xr2-5g-platform]. [N.d.]. [Accessed 22-05-2025].

[27] OSTHOFF, Andreas. *Qualcomm Snapdragon X Elite Analysis - More efficient than AMD & Intel, but Apple stays ahead — notebookcheck.net* [https://www.notebookcheck.net/Qualcomm-Snapdragon-X-Elite-Analysis-More-efficient-than-AMD-Intel-but-Apple-stays-ahead.850221.0.html]. [N.d.]. [Accessed 22-05-2025].

[28] *Frostbite (game engine) - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Frostbite_(game_engine)]. [N.d.]. [Accessed 22-05-2025].

[29] *Snowdrop (game engine) - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Snowdrop_(game_engine)]. [N.d.]. [Accessed 22-05-2025].

[30] CASSAING, Jonathan. *Which architecture should be implemented to manage data from the real world, in an Unreal Engine 5 simulator and in the context of mixed reality?* 2023.

[31] *Unreal Engine 5 — unrealengine.com* [https://www.unrealengine.com/en-US/unreal-engine-5]. [N.d.]. [Accessed 22-05-2025].

[32] *Unity (game engine) - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Unity_(game_engine)]. [N.d.]. [Accessed 22-05-2025].

[33] ENGINE, Godot. *Godot Engine - Free and open source 2D and 3D game engine — godotengine.org* [https://godotengine.org/]. [N.d.]. [Accessed 22-05-2025].

[34] FOUNDATION, Blender. *blender.org - Home of the Blender project - Free and Open 3D Creation Software — blender.org* [https://www.blender.org/]. [N.d.]. [Accessed 22-05-2025].

[35] MARLENAKLEIN-MSFT. *Mixed Reality Toolkit 3 Developer Documentation - MRTK3 — learn.microsoft.com* [https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/]. [N.d.]. [Accessed 22-05-2025].

[36] *SteamVR - Valve Developer Community — developer.valvesoftware.com* [https://developer.valvesoftware.com/wiki/SteamVR]. [N.d.]. [Accessed 22-05-2025].

[37] *Build new augmented reality experiences that seamlessly blend the digital and physical worlds — ARCore — Google for Developers — developers.google.com* [https://developers.google.com/ar]. [N.d.]. [Accessed 22-05-2025].

[38] *Resources — PICO Developer — developer.picoxr.com* [https://developer.picoxr.com/resources/]. [N.d.]. [Accessed 22-05-2025].

[39] *Qualcomm Announces Vuforia for Digital Eyewear, Powering the Next Generation of Augmented Reality Experiences — Qualcomm — qualcomm.com* [https://www.qualcomm.com/news/releases/2014/09/qualcomm-announces-vuforia-digital-eyewear-powering-next-generation]. [N.d.]. [Accessed 22-05-2025].

[40] *Varjo Native SDK — developer.varjo.com* [https://developer.varjo.com/docs/native/varjo-native-sdk]. [N.d.]. [Accessed 22-05-2025].

[41] *WebXR - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/WebXR]. [N.d.]. [Accessed 22-05-2025].

[42] *OpenXR - High-performance access to AR and VR —collectively known as XR— platforms and devices — khronos.org* [https://www.khronos.org/openxr/]. [N.d.]. [Accessed 22-05-2025].

[43] LEVENBERG, Kenneth. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*. 1944, vol. 2, pp. 164–168. ISSN 0033-569X. Available from DOI: 10.1090/qam/10666.

[44] MARQUARDT, Donald W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*. 1963, vol. 11, pp. 431–441. ISSN 0368-4245. Available from DOI: 10.1137/0111030.

[45] GAVIN, Henri P. *The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems c ©*. 2013. Available also from: https://api.semanticscholar.org/CorpusID:5708656.

[46] DAHLIN, Johan; SCHÖN, Thomas B. Getting Started with Particle Metropolis-Hastings for Inference in Nonlinear Dynamical Models. *Journal of Statistical Software*. 2019, vol. 88. ISSN 1548-7660. Available from DOI: 10.18637/jss.v088.c02.

[47] *Brokk 520D - Brokk Global — brokk.com* [https://www.brokk.com/product/brokk-520d/]. [N.d.]. [Accessed 22-05-2025].

[48] *XY (T Rosettes) strain gauges — hbkworld.com* [https://www.hbkworld.com/en/products/transducers/strain/experimental-testing/y-series/xy#!ref_hbm.com]. [N.d.]. [Accessed 22-05-2025].

[49] GARRIDO-JURADO, S.; MUÑOZ-SALINAS, R.; MADRID-CUEVAS, F.J.; MARÍN-JIMÉNEZ, M.J. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*. 2014, vol. 47, pp. 2280–2292. ISSN 00313203. Available from DOI: 10.1016/j.patcog.2014.01.005.

[50] *Home — opencv.org* [https://opencv.org/]. [N.d.]. [Accessed 22-05-2025].

[51] WENG, J.; COHEN, P.; HERNIOU, M. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992, vol. 14, no. 10, pp. 965–980. Available from DOI: 10.1109/34.159901.

[52] *OpenCV: Camera Calibration and 3D Reconstruction — docs.opencv.org* [https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html]. [N.d.]. [Accessed 22-05-2025].

[53] *Rodrigues' rotation formula - Wikipedia — en.wikipedia.org* [https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula]. [N.d.]. [Accessed 22-05-2025].

[54] *acA1920-50gm — Basler AG — baslerweb.com* [https://www.baslerweb.com/en/shop/aca1920-50gm/]. [N.d.]. [Accessed 23-05-2025].

[55] *Nvidia Super Developer Kit Jetson Orin Nano 8 GB 6 x 1.5 GHz — Conrad Electronic — conrad.cz* [https://www.conrad.cz/cs/p/nvidia-super-developer-kit-jetson-orin-nano-8-gb-6-x-1-5-ghz-2998506.html]. [N.d.]. [Accessed 23-05-2025].

[56] *Jetson Orin Nano Super Developer Kit — nvidia.com* [https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/]. [N.d.]. [Accessed 23-05-2025].

[57] *Building with Meta Spatial SDK — developers.meta.com* [https://developers.meta.com/horizon/develop/spatial-sdk]. [N.d.]. [Accessed 23-05-2025].

[58] *Meta Spatial Editor for Windows — developers.meta.com* [https://developers.meta.com/horizon/downloads/package/meta-spatial-editor-for-windows/]. [N.d.]. [Accessed 23-05-2025].

[59] *Download Android Studio & App Tools - Android Developers — developer.android.com* [https://developer.android.com/studio]. [N.d.]. [Accessed 23-05-2025].

[60] *Low Latency, LL-HLS, LL-DASH — ottball.com* [https://ottball.com/low-latency-ll-hls-ll-dash/]. [N.d.]. [Accessed 23-05-2025].

[61] OPTIVIEW, Dolby. *Low Latency DASH (LL-DASH) — optiview.dolby.com* [https://optiview.dolby.com/resources/blog/streaming/low-latency-dash/]. [N.d.]. [Accessed 23-05-2025].

[62] OVENMEDIAENGINE. *Low-Latency HLS: The Era of Flexible Low-Latency Streaming — OvenMediaEngine* [https://medium.com/@OvenMediaEngine/low-latency-hls-the-era-of-flexible-low-latency-streaming-ec675aa61378]. [N.d.]. [Accessed 23-05-2025].

[63] *Supported media formats — Android media — Android Developers — developer.android.com* [https://developer.android.com/media/platform/supported-formats]. [N.d.]. [Accessed 23-05-2025].

[64] *Meta Quest for Creators — creator.oculus.com* [https://creator.oculus.com/media-studio/documentation/rectangular-video-spec/]. [N.d.]. [Accessed 23-05-2025].

[65] *GitHub - bluenviron/mediamtx: Ready-to-use SRT / WebRTC / RTSP / RTMP / LL-HLS media server and media proxy that allows to read, publish, proxy, record and playback video and audio streams. — github.com* [`https://github.com/bluenviron/mediamtx`]. [N.d.]. [Accessed 23-05-2025].

[66] *Photopea — Online Photo Editor — photopea.com* [`https://www.photopea.com/`]. [N.d.]. [Accessed 23-05-2025].

List of Figures