2021

DESIGN OF A PLATFORM FOR INTERACTION WITH UNMANNED AERIAL VEHICLES USING VIRTUAL REALITY TECHNOLOGY

Janis Kinzinger
University of Rhode Island, janis.kinzinger@gmail.com

DESIGN OF A PLATFORM FOR INTERACTION WITH UNMANNED AERIAL VEHICLES USING VIRTUAL REALITY TECHNOLOGY

BY

JANIS KINZINGER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING AND APPLIED MECHANICS

UNIVERSITY OF RHODE ISLAND

2021

MASTER OF SCIENCE THESIS

OF

JANIS KINZINGER

APPROVED:

Thesis Committee:

Major Professor Chengzhi Yuan

Paolo Stegagno

Musa Jouaneh

Brennan Phillips

Brenton DeBoef
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2021

ABSTRACT

Unmanned Aerial Vehicles (UAVs) are becoming more and more integrated into our lives as they are able to fulfill new tasks and become more affordable. Human-UAV Interaction (HUI) therefore is an important topic researcher are working on. As a type of UAVs, quadcopters are characterized with uncertainties in their flying behavior and considered dangerous for humans because of their spinning propellers. Interaction through touch is therefore rarely used as interaction method. Hence, in this work Virtual Reality (VR) is used to simulate a quadcopter to create a platform for virtual HUI. In the development process, Unity was chosen to handle the Virtual Reality application and Gazebo running on the Robot Operating System was used to simulate the quadcopter. The user interface is provided by the Oculus Quest 2 VR headset. Finally, two conducted experiments validated that the system is capable of computing and performing new trajectories of the UAV from physical user inputs. Moreover, the performance of the system was analyzed, and delays of dedicated system components were computed.

ACKNOWLEDGMENTS

First of all, I would like to thank my major Professor Chengzhi Yuan for the mentorship and the support during my thesis. Moreover, I thank my co-major Professor Paolo Stegagno for the huge commitment and the advice throughout my time at the URI. I am grateful for all the effort he put into this project and I am very happy to have worked with him. Also, his financial support in the form of technical work equipment contributed to this successful thesis. Additionally, I would like to acknowledge my committee members Professor Musa Jouaneh, Professor Brennan Phillips and committee chair Professor Sigrid Berka. Working on different tasks of the same project, I would also like to thank Matthew Morgan, Austin Clark and Nathan Grantham-Coogan for their contribution to this teamwork.

Most importantly, I would like to express my heartiest thanks and gratitude to my family and particularly my parents. They made this unique time possible and supported me with every step I go. And finally, I wish to thank my girlfriend Nora for all her love, support, and continuous motivation during these hard times. Without you, I would not have had this wonderful time in the US.

TABLE OF CONTENS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The emerging role of Unmanned Aerial Vehicles (UAVs) in industry and for non-commercial use leads to new areas of application. A challenge wherever people encounter UAVs is the Human-UAV Interaction (HUI). Safety issues as well as the form of interaction must be discussed. In this thesis, no real UAV will be used. With Virtual Reality (VR) we have a tool to simulate real-looking scenarios with high immersive effect on the user. Computer-generated models in the virtual world allow us to interact with UAVs without exposing humans to a risk.

This work developed a platform for Human-UAV Interaction using a virtual reality headset. Thereby, the person's field of view is simulated, and the position of their hands is tracked to allow physical interaction with a safe computer-generated model of a UAV. This type of interaction is also referred as Physical Human-UAV Interaction (PHUI).

Starting with the Justification for and Significance of the Study, previous literature on the topic of HUI and PHUI combined with VR or without is presented and reviewed. Moreover, this work is brought into context with the existing studies highlighting the novel aspects of this approach. The subsequent Methodology chapter introduces into the components of the system architecture by giving overviews of their abilities, limitations and how they work. After that, the reader should be able to follow the description of the System Architecture from where the developed components are being explained. All scripts, packages, and tools used are explained starting with the Unity components, followed by the Gazebo components. This section forms the main development work of this thesis. In the Simulation Setup, the process of preparing, recording, and conducting experiments is explained to understand the basis of the following results. Divided into

two sections, the Results chapter first provides an overview on the achieved features of this platform and then presents the recorded data in form of curve diagrams. The findings are explained, and any abnormalities analyzed in detail. Finally, the Discussion and Future Work chapter completes the thesis with evaluating the limitations and with proposing new approaches to improve the developed platform and for further investigations on the topic of PHUI using this system.

2. JUSTIFICATION FOR AND SIGNIFICANCE OF THE STUDY

In recent years, the number of UAVs has increased steadily, attracting the attention of companies, customers, and researchers. The market growth of registered commercial and hobbyist UAVs from 2016 to 2019 in the U.S. was nearly 67%, with the commercial sector showing the most increase according to the Federal Aviation Administration [1, 2]. With this development, UAVs became more and more versatile. Coming from camera-equipped observation UAVs in highly challenging military applications, they now are available for everyone and used as a toy, hobby article or commercial equipment. Through continuous research, applications were extended to the capability of navigating visually impaired persons [3], being used as a hovering ball in sports [4] or controlling a crowd in emergency situations [5].

"As drone usage increases, humans will interact with such systems more often, therefore, it is important to achieve a natural human-drone interaction." [6]. A natural Human-Drone Interaction (also Human-UAV Interaction, short HUI) can consist of multiple control methods. Commonly used controllers have the disadvantage of not being intuitive since every controller layout is different and only a minority of the population is familiar with gamepads and 3D-control through keys. Because of the high level of noise emission from the propellers, speech recognition often is not possible for UAVs. Gesture controlling as another interaction method requires additional sensor systems either on the UAV or set up in the environment that track specific parts of the human body. Those sensors can be cameras for example, but this type of system is not easy to set up and cannot be mounted on every type of UAV. All these limitations in mind, interaction through touch seems ideal, because it is easy to use, and no additional

hardware is needed. Furthermore, physical interaction with UAVs enables new applications, such as programming trajectories by simply guiding it by hand, which is a standard method for collaborative robotic arms (“BitDrones” [7]) Another application could include close work scenarios together in the same workspace with physical interaction as an additional safety feature. A major field of research about the topic of Physical-Human-UAV Interaction (PHUI) is haptic feedback for VR. Although the immersive effect of current VR-Headsets and simulations already is very high, physical feedback going further than vibrating controllers is still in development state. In the following examples, UAVs are used to provide haptic feedback to the user in the virtual word.

In [8], a virtual shopping experience called “HoverHaptics” is created that uses an UAV for different modes of haptic of objects. While the person sees the simulated scenery, a safe-to-touch UAV enriches the VR world with positioning of texture material corresponding to the object in the virtual world that is attached to the UAV and can be touched. Furthermore, the UAV is equipped with real replicates of the virtual objects and can put them into the right position according to the simulation. The a.m. paper also developed the controller for autonomous positioning in this setup.

A framework named “Flyables” designed a tactile user interface based on small UAVs [9]. Thereby quadcopters equipped with a safe-to-touch cage were used to create a spatial user interface. The authors conducted a survey of people interacting with their framework while performing three different tasks, finding out at which position the interaction took place in relation to the user and which fingers were used.

For more general scenarios “TactileDrone” [10] and “VRHapticDrones” [11] developed systems with quadcopters as touchable objects in VR. While TactileDrone

focuses on the implementation of the tactile feedback system and the integration of motion capture systems into VR applications, the scope of VRHapticDrones goes even further and includes two experimental studies. Study 1 determines the effect of haptic feedback in VR on the perception of presence of objects, whereas study 2 investigates the physical compliance between what is sensed through touch and what is seen in the virtual world.

Recapitulating that every referred paper that allowed touching UAVs in their studies had used propeller guards, it has to be stated that touching UAVs such as quadrotors is in general considered unsafe to Humans, because of the spinning propellers. To overcome this problem, propeller guards were designed to provide safe-to-touch UAVs as used in the previous examples. A study “Drone Near Me” [12] investigated the likeliness of using touch as an interaction method for UAVs and how comfortable probands feel while interacting. The authors found out that for interaction with safe-to-touch UAVs, touch is more likely to be chosen than with unsafe-to-touch UAVs. Moreover, when safety was ensured, 58% of the participants preferred touch for interaction compared to gestures or speech input.

Applying the findings of the VR haptic feedback implementations on HUI, a new approach for experimental studies on HUI is formed. With Virtual Reality it is possible to overcome safety concerns when interacting with UAVs in the development phase of new PHUI methods or training scenarios. Not only the appearance and the noise of the UAV, but the whole environment can be simply adapted in the virtual world to get a safe experience while interacting. Multiple simulations and experiments with PHUI can be performed without limitations regarding the flight time of the UAV or spatial

requirements for the experiment. Moreover, the recording of interaction events and tracking of body trajectories in 3D-space can be achieved with low efforts in the VR-simulation. VR-experiences also can be used to sensitize people to touch UAVs or to teach people how to work with them.

An overview of the topic of Human-UAV Interaction is given by D. Tezza and M. Andujar [6]. This survey addresses a huge range of aspects in the field of HUI. Comparing most common UAV models and depicting different interaction methods, over 1600 research papers on HUI are categorized and their results summarized. Research on UAVs as human companions by C. F. Liew and T. Yairi [13] adds social aspects to the otherwise technical consideration. While research on companion robots already is advanced, investigation of companion UAVs still is in its early phase. A first approach of PHUI is described by S. Rajappa, H. Bülthoff and P. Stegagno [14]. Implementing a residual based estimator, the UAV controller estimates all sensed forces, calculates, and performs the desired trajectory. This mathematical model was applied to a touchable quadrotor with force estimation sensor ring around the quadrotor. The model was validated performing hardware-in-the-loop simulations along with experiments in which human interaction disturbs a defined state of the UAV.

The results of the papers mentioned above, using UAVs as haptic feedback to enhance VR experiences, can be used as a basis to simulate not tangible objects with the help of UAVs but to simulate the UAV itself in VR. Figure 1 summarizes the different aspects of the categorized papers presented so far.

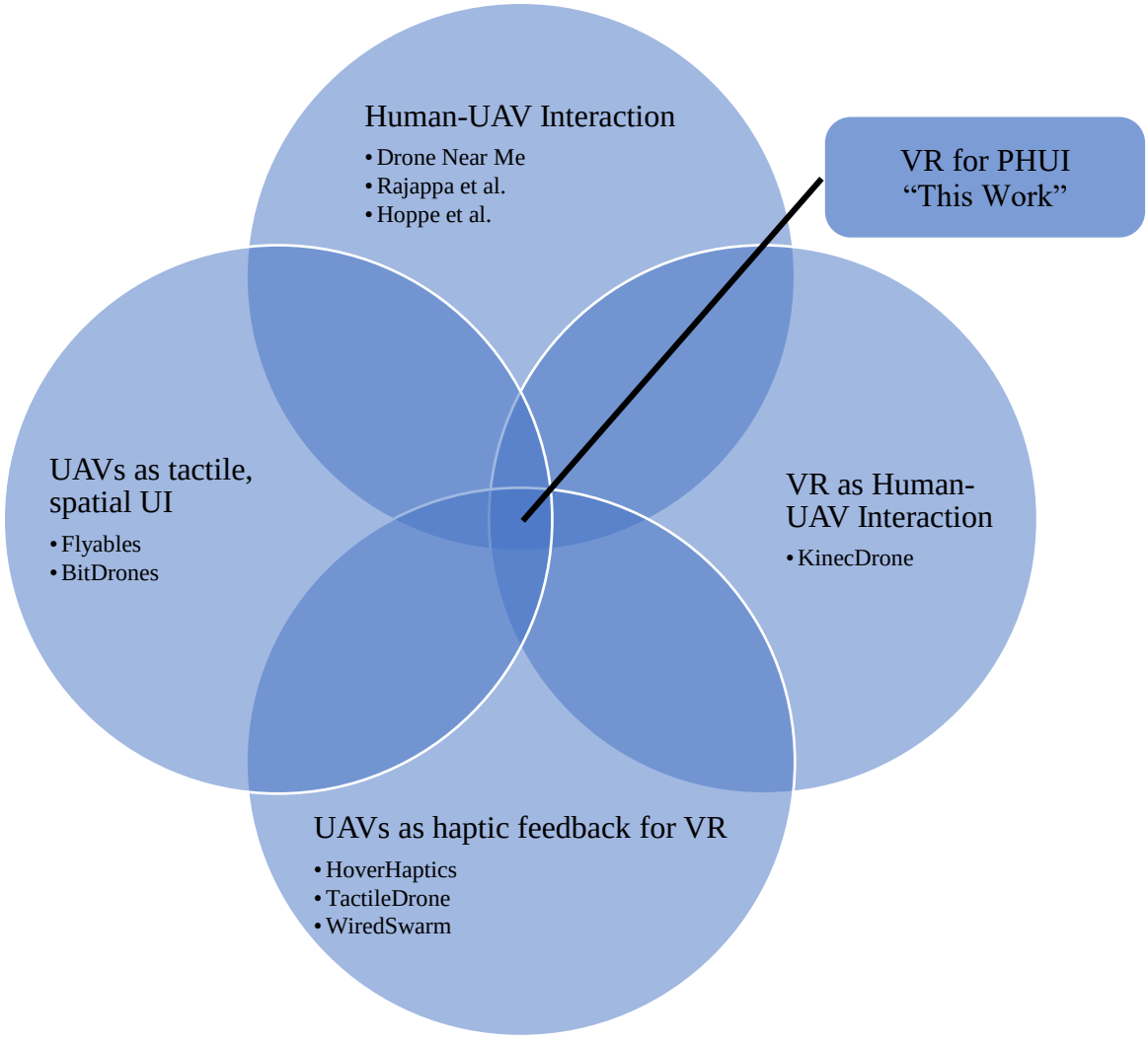

Figure 1: Literature Overview

3. METHODOLOGY

The system proposed for this thesis includes the setup of two different operating systems to connect to each other. On the one side we have a Windows computer on which we run the game engine Unity, that is ideal for VR applications. On the other side there is a Linux machine with the Robot Operating System (ROS) running on it. ROS is a powerful framework to control, simulate and connect robotic platforms. The following chapter introduces the components that were used in this work.

3.1. UNITY

Unity is one of the most popular real-time 3-dimensional development platforms to create games and immersive experiences across industries [15]. Besides games, this graphic engine is used for automotive manufacturing, film animation and cinematics as well as architecture, engineering and construction [16]. Unity therefore covers a huge variety of applications and supports over 25 different operating systems and (gaming) platforms. Projects with Unity can range from mobile applications on smartphones with Android or iOS to complex 3D-simulations in manufacturing process chains. It is used for commercial but also noncommercial purposes. A free version for educational purposes and small businesses makes Unity even more popular. The engine is available on the market since 2004 and thrives on a large online community that constantly releases new expansion packages and open-source code. The software was chosen because it is ideal for most VR developments and offers many interfaces to other software, especially ROS.

Unity's user interface (Figure 2) is optimized for 3D applications. Thanks to its drag-and-drop interface, it allows beginners to learn the most important functions easily.

Simply dragging properties onto selected objects brings them to action. All objects are represented in scenes that describe a fixed 3D space, such as a game level. Every scene consists of “Game Objects” and their children that form the structure of the content in this scene. A Game Object can contain very little but also very much content with thousands of child objects. It could combine individual scripts up to a whole set of visual objects of a game or simulations with their effects. Thereby every object has a position, orientation, and size in the world coordinates of the scene stored in the “Transform” component. Besides this transform, various pre-defined or self-written components can be added to every Game Object. Those components can affect the visualization of the object, the physics or other behavior for example to create interactive content. As mentioned before Game Objects can be assigned to groups so that overall components can be added to impact multiple objects at once.

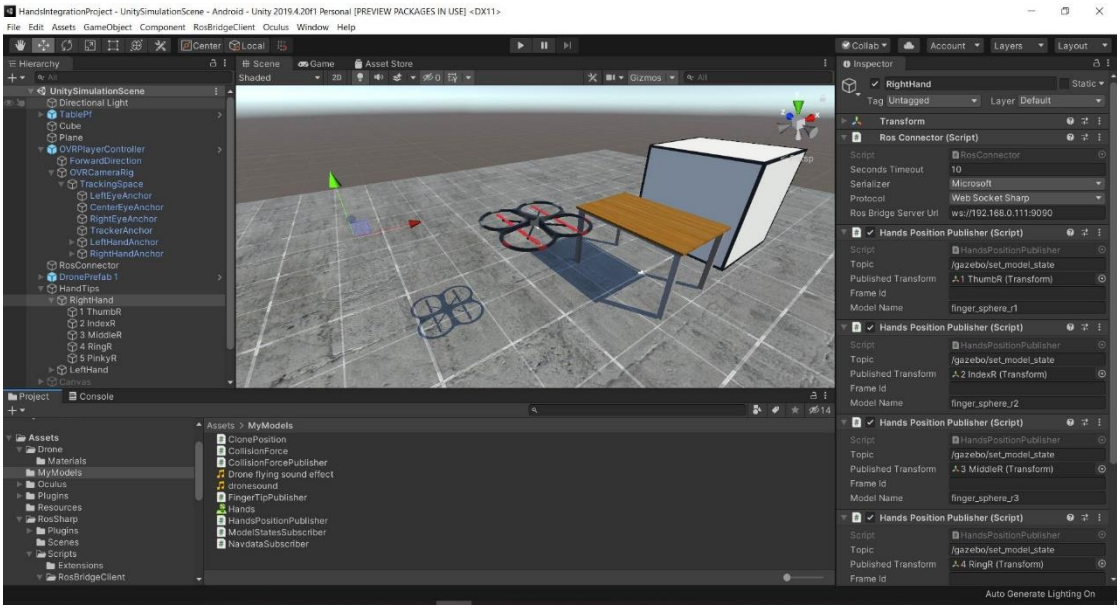

Figure 2: Unity GUI and Simulation Scene Overview. The 3D scene view is located in the center, the Hierarchy containing the Game Objects is on the top left and the components tos each Game Object can be reviewed in the inspector on the right.

Six main areas share the screen in Unity's standard layout. Each user can customize the tabs and position them freely on several monitors to best adapt them to their own workflow. In the default view, there is a "Hierarchy" on the left, the Game- and the Scene View in the middle and the "Inspector" on the right. Console and File Manager share the bottom tab. The hierarchy shows all Game Objects of the scene. These objects are visible in the scene view which is used for positioning of all objects relative to each other. The difference between scene view and game view is that the scene view is shown in the third person, while the game view shows the first-person view, which represents the actual game on the device the project is made for. In case of the work in this thesis, entering the play mode will start streaming the view of the person wearing the VR-Headset into the game view window. This mode allows developing and manipulating the application in real time without the need for compilation or deploying it on the VR-Headset in advance. In the development process this is a time saving feature.

Having access to a large library of predefined objects, so called "Prefabs", Unity makes creating content also faster with its integrated "Asset Store" where all kinds of extension packages and prefabs can be found. Besides importing assets directly from the store into the project, it is also possible to download content manually from various resources. Packages could not only contain complete prefabs, but also components like material properties, meshes or additional plugins.

Every component of a Game Object is the visualization of a script that is running in the background. These scripts have access to all kinds of libraries within Unity, so they can use many features that affect the Game Object itself or other referenced objects. As an example, the position of an object can be accessed and modified with the

expression `gameObject.transform.position`. Thus, it is possible to adapt the code of existing scripts to the requirements of the respective project. In addition, self-written scripts facilitate a wide range of functionalities to achieve the desired behavior of the Game Object.

3.2. ROS

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms." [17]. The Open-Source Software can communicate with almost every kind of actuator or sensor on robots thanks to a wide library of general purpose and specific software drivers. Towards that communication, ROS provides commands and GUIs to make working with robots easier and clear. At the same time ROS can operate 100% without a GUI [18]. Although it only runs natively on Unix-based platforms such as Ubuntu or Mac OS X, the programming languages C++, Python and Lisp are supported. Using an anonymous Peer-To-Peer connection system, ROS communication is coordinated from one software program called "Master". Smaller programs called "Nodes" register with the Master as publishers and/or subscribers of a certain "Topic". The nodes summarize certain functions and provide sensor data or control the robot drives, for example. This publisher and subscriber concept enables nodes to communicate directly with each other via "Messages". Topics bundle messages with the same information e.g., position data of one object. Before a node publishes a message to a topic, the master has informed the publisher which nodes have subscribed to that topic, so that the

publisher can send the message to all the subscribing nodes. The system can be summarized as in Figure 3.

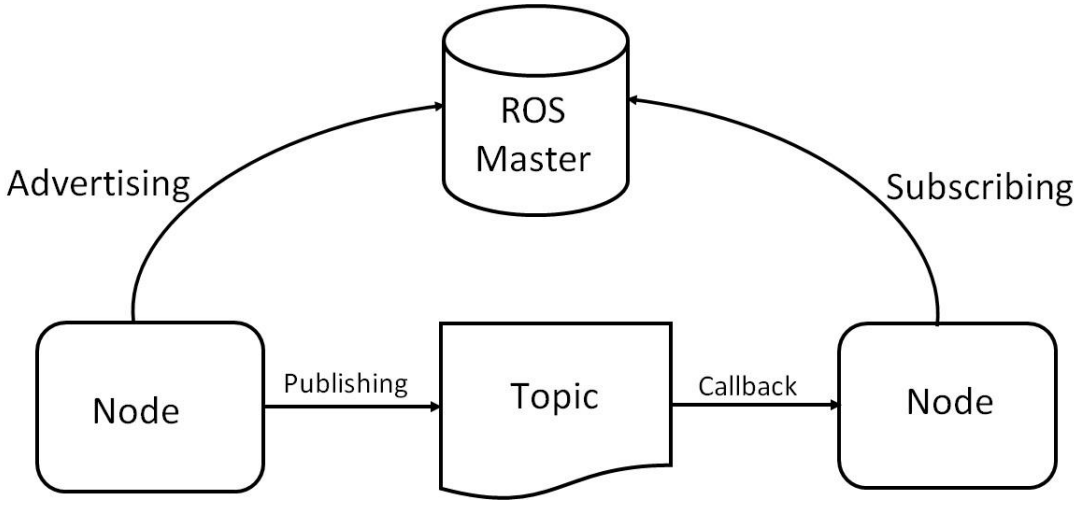

Figure 3: Communication in ROS. Message transfer over topics on the bottom, Node subscription system in the top part of the figure [19]

3.3. GAZEBO

The 3D multi-robot simulator Gazebo is an essential tool in this Thesis. Its powerful physics engine paired with a GUI and multiple available plugins allows fast and realistic testing of robot software as well as designing robots and surrounding environments. Gazebo supports most robots and can emulate their specific sensors generating camera images, contact sensor data or laser range finder data for example [20]. Advanced 3D graphics enhance realistic rendering of all textures and shadows to make the simulation as close as possible to a real setup. Like ROS, Gazebo is a free, open-source software. Its vibrant community supports Gazebo users in solving occurring problems rapidly.

The GUI of Gazebo (Figure 7) mostly consists of the 3D-Scene window that allows moving the camera in space to observe the robot and its environment. On the left side there are three tabs from which the “World” tab is the most important one. It contains a

list of all objects (called “Models”) in the simulation. When selecting a model from the list, properties and their values will be displayed. For example, the position of a model is displayed, or its structure, including the dependent links and the respective properties. Using these tools, it is easy to gather information about the status of the model, recent parameter settings or the model’s structure to determine possible error sources during the simulation. Below the 3D-scene window there is a bar which includes a play/pause button for the simulation, data about the simulation performance and the current time. Some quick tools to manipulate the simulation are available above the 3D-scene window.

Gazebo simulations can be used without displaying the GUI (headless mode) if there is not enough performance on the computer. As part of the ROS ecosystem, Gazebo offers a lot of command line tools to have a quick access on parameter settings and direct outputs. Gazebo and the simulated scene can be launched from one file together with the ROS master and all dependent packages. The simulator uses a Simulation Description Format (SDF) to describe objects and environments that is simple, to make modifications and creating new objects easy.

3.4. OCULUS QUEST 2

The Oculus Quest 2 (OQ2) is the fifth Virtual Reality Headset from the company Oculus. It is designed as an all-in-one device, which means that it can be used together with an external PC but also has a full gaming experience without any external device. Oculus Mobile, the Operating System that runs the Quest 2, is based on Android 10 and draws its applications from the Oculus Store. It also allows installation from foreign

sources through the USB 3 cable for example from Unity in the developer mode of the OS. Connected to a PC via Oculus Link, games can be played on the PC's hardware while the video is streamed to the VR headset.

Figure 4: Oculus Quest 2 and left controller

For controlling the OQ2, some applications require controllers (Figure 4) for each hand to be used. In all other apps, as well as the main menu, direct hand tracking without additional devices makes navigating and scrolling easy and intuitive with the use of gestures. No preliminary calibration or setup is necessary to use hand tracking. The four cameras of the OQ2 capture hands and environment precisely (Figure 5). The safety system called "Guardian" of the headset asks the user to either define a room scale play-safe-zone without any objects in the way or to stay in a smaller stationary boundary. Once the user approaches the boundary with his hands or head, a net gets visible to show the limit of the safe space to avoid hitting surrounding objects. Another safety feature

is the see-through mode, where the headset casts video of the real environment instead of the VR world.

Figure 5: Oculus Quest 2 hand tracking with 4 fish-eye cameras [21]

The LCD type display of the Quest 2 has a resolution of 1832x1920 pixels per eye, a refresh rate of 90 hz and 90-degrees field-of-view [22]. Stereo microphones, stereo speaker inside the head strap as well as 3-Dimensional sound with the use of the head-phone jack give a surrounding sound and allow good communication options within most applications. These specifications allow a high level of immersive experience for the user so that this device facilitates the goal of simulating as realistic as possible.

3.5. SYSTEM ARCHITECTURE

The architecture of the developed system can be separated into two parts. On the one hand there is a Windows operated computer, on the other hand a Linux system was used. The following paragraph will give a brief overview of the complete system to

understand detailed descriptions of the windows system and the Linux system afterwards.

Running on the Linux machine, ROS is capable to execute the previously described robot simulation and visualization software Gazebo. On the Windows side we can create another 3D-Simulation in Unity. The two simulations will be connected to take advantages of their capabilities: the faithful dynamic simulation from Gazebo and the VR-enabled visualization capabilities of Unity. In Gazebo, the UAV can be controlled and simulated while data containing position and orientation is published frequently. These data will be sent to the Windows computer using a RosBridge WebSocket. The Unity simulation also contains an UAV whose position and orientation follows those of the UAV in Gazebo through the published data. This setup is necessary to integrate a virtual reality headset because VR is not yet supported on Linux. The VR-Headset is connected to Unity via an USB link. 3D-Video for the VR-Headset will be rendered in Unity and broadcasted to the headset, while the headset is tracking its motion as well as the user's hands. This information from the headset is passed back into Unity. To enable interaction with the UAV, Unity publishes the position and orientation of the hands to ROS, so that Gazebo can replicate the hands in its simulation. The controller of the UAV evaluates contact forces between the hands and the UAV and induces movement commands to the UAV.

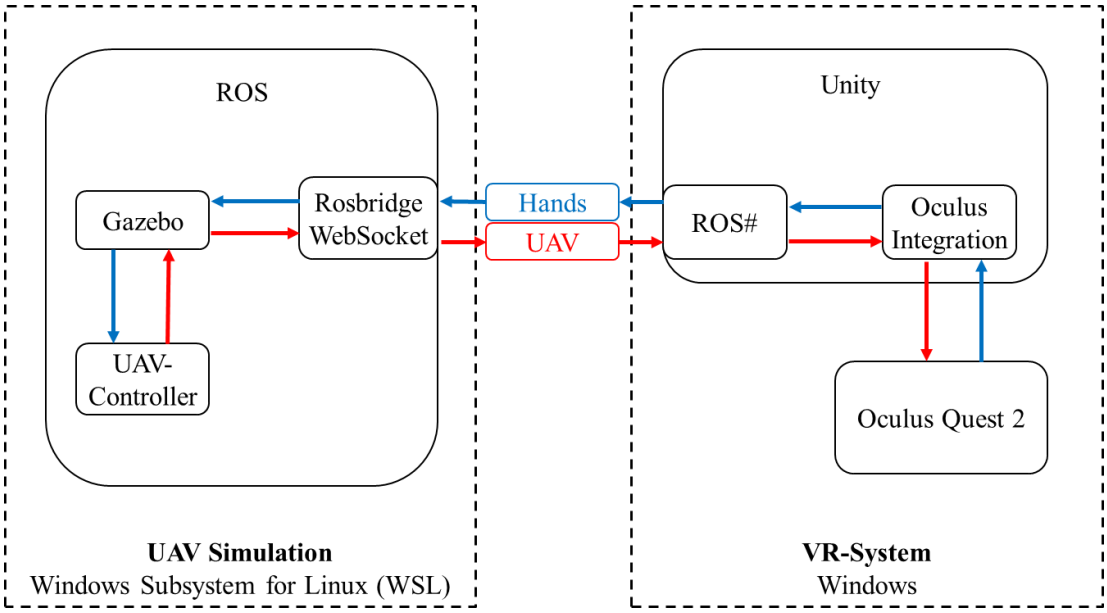

Figure 6: System Architecture. Blue arrows show path of the position of the hands, red arrows show the return path of the position of the UAV through the system components

The interface in ROS to connect to a non-ROS software is provided by a package called “rosbridge_suite”. The matching plugin for the Unity Editor to read the ROS data is provided by a project called “ROS#”. The final platform architecture is shown in Figure 6.

For the Linux machine, external computers, virtual machines, and the Windows Subsystem for Linux (WSL) were tested with the simulation software. The fast work-flow, good access to files and most important the outstanding efficiency in hardware consumption resulted in the use of the WSL. Since it can be treated as a windows pro-gram, clipboard sharing, and window handling allow fast and easy console commanding in Linux.

3.6. UNITY-RELATED COMPONENTS

ROS#

"ROS# is a set of open source software libraries and tools in C# for communicating with ROS from .NET applications, in particular Unity." [23]. Developed by engineers and programmers from Siemens, the package enables communication between ROS-run robots and Windows applications. It provides tools to import the robot's URDF model from ROS into Unity, to visualize the robot's state and sensor data in Unity, to simulate the robot in Unity without ROS, to control the robot via Unity and to train neural networks [24]. The communication system between Unity and ROS is based on a WebSocket. This WebSocket is launched on ROS from a package called RosBridgeClient. In Unity, a script called RosConnector connects to the IP address of the WebSocket and manages all subscriber or publisher components of the same Game Object. Therefore, every Game Object with communication to ROS needs one RosConnector client. Once the connection to the ROS WebSocket is established, publishing or subscribing scripts can send or receive messages of specified topics. Depending on the ROS message type, 20 available scripts from the ROS# package can be used. For example, components to publish or receive images, velocities, positions. Implementing new message types in Unity requires definition in the MessageTypes namespace. Subsequently, components can access the new message type to parse the information into or from the ROS type and publish or subscribe to the specific topic, depending on whether it is a publishing or subscribing script. Besides communication scripts that connect to topics, ROS# also comes with tools to visualize laser scan arrays or to write motor commands in Unity.

In case of this work, the message types gazebo/model_state and gazebo/model_states were not provided by the ROS#. Instead, they were imported from a ROS package. ROS# therefore comes with a tool that automatically generates message types in Unity from a ROS source. In this case the Gazebo 7 package was used to generate 9 scripts that enable using these message types in other scripts inside Unity.

Oculus VR components

In addition to section 3.1 Unity, the following paragraph will give an overview over what is needed to connect the Oculus Quest 2 to a Unity simulation.

Oculus provides a broad toolkit for Unity in the Asset Store called “Oculus Integration” that is available for free [25]. It contains scripts for rendering video streams via Oculus Link, social interaction with other players, audio management, avatar visualization and more. Most important for this project are the Prefabs (predefined Game Objects) that contain different models of the hands. The prefabs were adapted to perform as desired. Features are activation of hand tracking, rescaling of the virtual hands with the size of the user’s hand, and setting their appearance in the virtual world. Also capsules for every bone of the hand were added to the simulation with this prefab to ensure calculation of the physics when interacting with other physical objects. Besides the hands, the Oculus Integration Package was used for creating the camera system in Unity that is needed for the VR-Headset to operate. This “Oculus VR Player Controller” contains a rig of cameras for each eye to be displayed on the Headset device. Features as clipping, which tells the software when to stop rendering images if they are too close to the head, are included in this essential prefab. In general, every component that is related

to the Oculus system such as the hands, controllers or cameras is centralized under this Game Object.

Another important tool that is needed for Oculus VR development is the Oculus XR Plugin that can be installed through the package manager of Unity. While the Oculus Integration Asset described before is used for the practical creation of the VR scene, the XR Plugin is more a backend tool inside the Unity engine. It handles input of the headset on the one hand and outputs stereo video rendering to the device on the other hand. Building the actual application from Unity to an android executable file and installing it on the device in one instance is also a key functionality.

To use all these features, which require the hardware of the headset, the Oculus Link cable connection is necessary. Additional to running Unity, Oculus Link works with a Windows App provided by Oculus. This software accesses the PCs hardware to render stereo video for the headset once the link mode is activated inside the Oculus Quest. The headset is then powered by the PC and can be used to test the created simulation in Unity. With this said, it is also possible to substitute the cable with a powerful Wi-Fi connection. This setup however was not tested.

Special Scripts/Work

As mentioned before, the message type gazebo/model_state was used to transfer positions and orientations of objects between Unity and Gazebo. To subscribe to the position of the UAV in Gazebo, a subscribing ROS# script was adapted to the new message type. Thus, the messages from the topic are translated into Unity format and affect

any Game Object, in this case the UAV in Unity. The same way, a publisher script was adapted to form a Unity “Transform” into a continuously published ROS message.

One special work-around was needed to access the position of the hands in Unity. The hand controlling script uses controller for the Oculus Quest as default input method. In the case that only hands were found instead of controllers, the Unity prefab switches to hand tracking and to the hand input method. Hands are therefore not instantiated at the beginning of each simulation, such that there is no matching Game Object for publication, what would cause errors. The solution is a system of dummy Game Objects that only are used to transport the position of the hands and fingertips once they are recognized. The objects therefore have components that search continuously for the name of the real hand objects inside the hierarchy, that are instantiated later, to adopt their positions and orientations. Then, the positions of those dummy objects are published to ROS. Since the fully articulated hands consist of multiple bones, only the tip of each finger and the wrist of each hand is transferred to ROS to minimize the flow of data over the WebSocket.

To achieve the most immersive experience in this application, another Asset from the Unity Store was used to import a realistic model of an UAV [26]. The 3D model has spinning propellers and its geometry is close to the UAV model in Gazebo. To match collisions and visual model within both simulations, the Unity model was resized manually. Moreover, surround sound effects of a flying drone were added to the Unity model to make it more realistic.

3.7. GAZEBO-RELATED COMPONENTS

ROS#/WebSocket

The WebSocket is part of the rosbridge_suite [27]. It must be launched within the rosbridge_server package to set up the IP-address for non-ROS communication services. This address changes when the network of the computer changes, so it must be updated inside the Unity scripts in this case. Once the WebSocket is running, it connects to the ROS master and displays every client in the console that connects from outside ROS to any published topic.

QLAB

The Gazebo simulation essentially uses the QLAB (Quadrotor LAnding Benchmarking) package. This collection of files provides a "simulated environment for developing and testing landing algorithms for unmanned aerial vehicles" [28]. Based on the simulator of the Technical University of Munich, only the UAV with its visualization and controllers were utilized in this work. The running package provides data of multiple sensors on the UAV which is used in the controller files to coordinate the movements of the UAV. Moreover, the controller creates a realistic behavior of the UAV including disturbances as wind.

In order to implement interaction with external objects, especially fingertips in this setup, a force-torque sensor was added to the UAV Gazebo model. This sensor is connected to a non-visible collision frame around the model, so that contacts with other objects can be detected. To represent the hands in this setup, small spheres for each fingertip and large spheres for each wrist were added to the scene. As already mentioned, they are linked to the position of the hands inside Unity and to the real hands'

movement respectively. When the spheres touch the collision frame of the UAV, the sensor calculates forces in every direction. This force vector is passed to the controller of the UAV to react to this intentional disturbance. The data is also published to a /ft_sensor topic. Instead of a behavior that tries to hold its position, in this case the UAV will take these forces as a command for controlled movement in the direction in which the force was applied. The factor of the applied movement was reduced so that the UAV will stay in a reachable distance inside the simulated space of the VR-Headset.

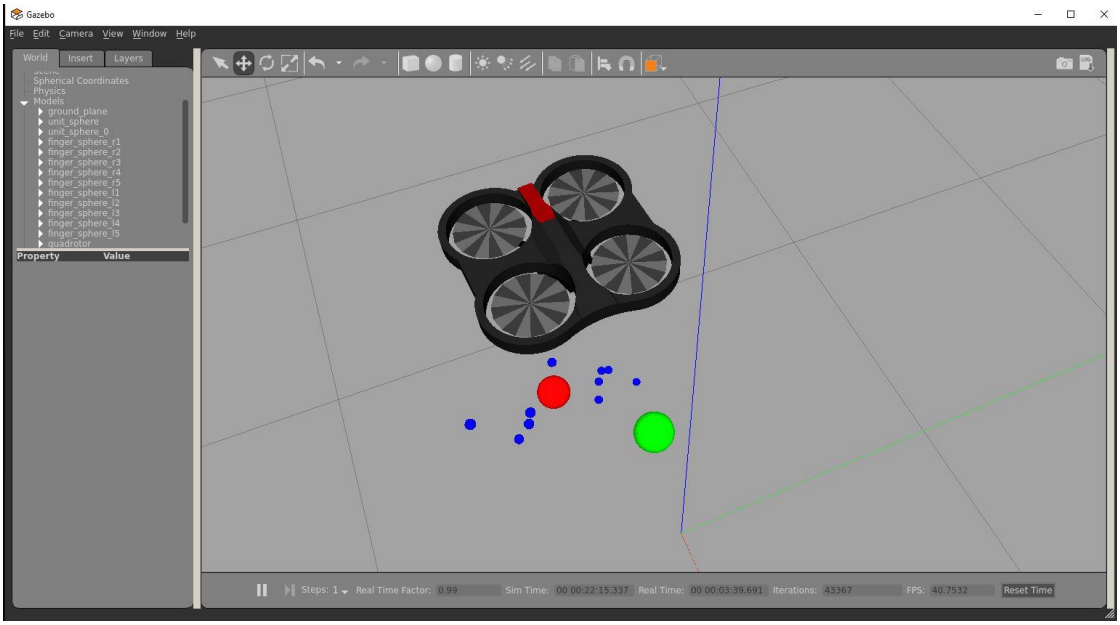

Figure 7: UAV and Hands with Fingertips(smaller blue spheres) and Wrists (big red and green spheres) simulated in Gazebo

3.8. SIMULATION SETUP

To validate the systems functionality and to benchmark its performance, data were gathered during the simulations. Aiming for the delay between the components, coordinates with timestamps were recorded on each system to compare their individual movement. ROS therefore provides a feature called Rosbag. During simulations, ROS stores the message data from either all or previously specified topics into a .bag file. This bag can be re-played to display all messages to the topics, or it can be exported into other file types for example a .txt file. In this case, relevant data were recorded from the /set_model_state, /model_states and /ft_sensor topics. Recalling that /set_model_state was used for passing coordinates from the hands from Unity to Gazebo, that /model_states was used to transmit the UAVs position from Gazebo back to Unity and that /ft_sensor tracks the input into the UAV controller (Figure 6: System Architecture).

Data from Unity were generated inside the debug console. Therefore, output messages for the console were added in the scripts that contain the positioning data. The console output in Unity is stored into a .log file by default and can be accessed easily. Subsequently filtering these data was necessary to create another file that only contains the data required. Another aspect to make all data comparable was to use the Unix Time format in Unity. Since all outputs from ROS use this timestamp format, it also was implemented in the Unity output.

The first analysis showed a linear increasing delay between the published data from Gazebo and the movement data inside Unity. Taking into account that the Gazebo simulation is running with a real time factor of around 0.99 and that the published data timestamp is coupled to the simulated time and not the real time, the Gazebo timestamp

has to be corrected for the analysis. Subsequently, a ROS node was created to publish a new topic containing the real time together with the simulated time. Knowing these timestamps, it is possible to calculate the overall delay of the simulation and to correct all Gazebo timestamps in the recorded data.

For the simulation setup, a powerful gaming laptop was provided. Equipped with the 6 cores CPU Intel i7-9750H and NVIDIA's GeForce RTX2060 (Mobile) coming with 6GB VRAM,16GB RAM and 2TB SSD, running Virtual Reality applications simultaneously to the ROS environment with Gazebo, the speed of each software is at a sufficient level. Thus, the hardware limits did not generate a bottleneck for the simulations.

To calculate the total delay between touching the UAV in the VR Headset and seeing the resulting movement in VR, three steps were necessary. Starting with the process from Unity to Gazebo, positional data from the hands could not be compared directly. Since update frequencies of every system component are different, for each time entry of the sender the matching data on the receiver side was searched. Hence the dedicated timestamps on the Gazebo side could be corrected by means of the published real time topic mentioned before. The second step is to then search for the same positional data on both resources and compare their timestamps to evaluate potential delays. Moreover, the positional data of the UAV gains some delays on its way back from Gazebo to Unity. This lag can be computed as described above – first correct the Gazebo timestamps, then search for the same positions and subtract both times. In this way, data from X, Y and Z axis were recorded and analyzed in order to obtain information on the overall system delay during the course of the simulation.

In addition to the delay, another parameter was evaluated to measure the performance of the system. Since interaction with the UAV is based on contact between hands and UAV, forces are computed to control the UAVs movement. Therefore, the forces in X, Y and Z direction were traced and as well as all Gazebo data corrected with corresponding factors.

4. RESULTS

In the following chapter, the results of the simulations will be presented and discussed. The setup for the simulations was described in the section 3.8 Simulation Setup. Summarizing the previous work, an application for the Oculus Quest 2 was developed that is operated by a computer through the game engine Unity. Furthermore, this application connects to a Gazebo simulation in real time by the use of a WebSocket. The system simulates an UAV that allows interaction through the users' hands while wearing the VR Headset.

4.1. FUNCTIONALITY DESCRIPTION

Describing the functionalities of the developed platform, the following was achieved:

Behavior of the UAV

The UAV is controlled by a PID control script that imitates the movement behavior of real UAVs. Since the thrust of propellers is uniaxial and desired trajectories mostly are not, a more complex movement that is based on the orientation of the UAV is required. In this case, that includes either a change of angle in roll or pitch axis to perform a directed movement in X or Y axis in a cartesian coordinate system respectively. This implemented behavior feels most natural for user in the VR experience, because it is the expected movement.

Interaction

As already mentioned in the beginning of this thesis, interaction through the hands is the most intuitive way to impact on objects as UAVs. Therefore, the implemented UAV model allows interaction via touch. Hands can be directly used inside the VR

application to impact the simulated model. The setup that was implemented to model the hands throughout all software systems is designed for collision with other objects. Subsequently, pushing is the only method to influence the UAV in this system, since the collision frame of the models does not allow pulling or grabbing. The pushing process can be reviewed in the following Figure 8 and Figure 9.

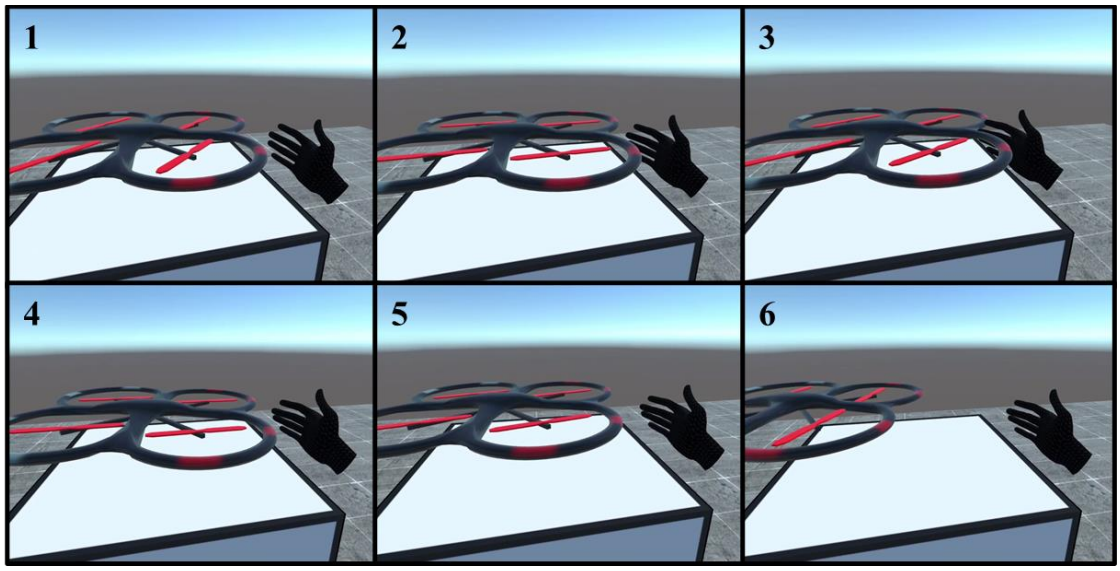

Figure 8: Pushing Reaction of the UAV; Screencast Footage from the OQ2. 1. UAV and right hand next to each other 2. Right hand moving towards UAV 3. Interaction takes place, fingertips are pushing UAV to the left 4. Hand is moving away from UAV 5. Hand is staying at the same position while UAV is tilting towards left 6. Tilting causes the UAV to move away from the Hand

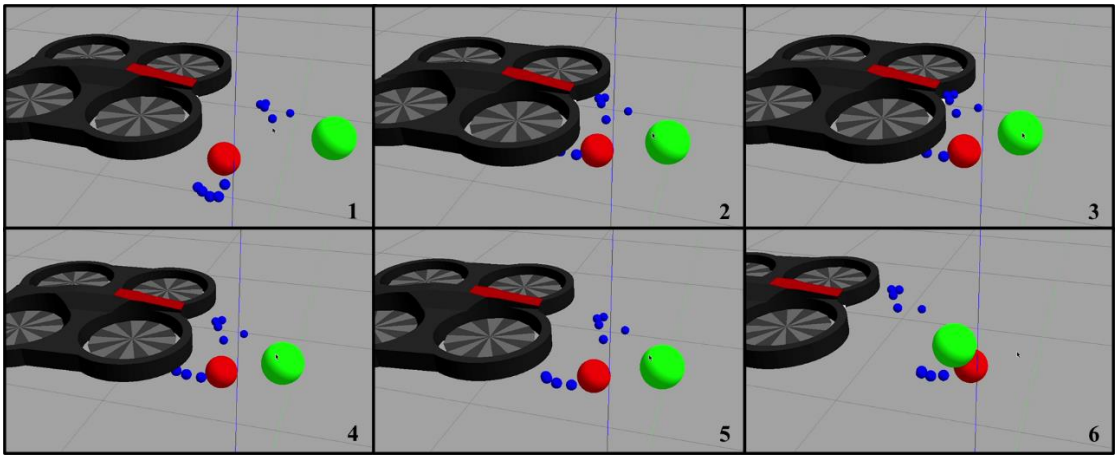

Figure 9: Pushing Reaction of the UAV: Screencast Footage from Gazebo. 1. UAV and hands next to each other, hand consist of one big sphere for the wrist and five smaller blue spheres for the fingertips 2. Right hand moving towards UAV 3. Interaction takes place, fingertips are pushing UAV to the left 4. Hand is moving away from UAV 5. Hand is staying at the same position while UAV is tilting towards left 6. Tilting causes the UAV to move away from the Hand

Force Sensor

Regarding the previous paragraphs, this behavior and the type of interaction were enabled through a force sensor on the UAV model. Detecting contact forces from the hands, the intended trajectory can be calculated and fulfilled in the manner explained above.

VR Experience

One key aspect of the platform is its high performance regarding the VR representation. With the aid of several tools from Unity, a real-looking and real-feeling VR environment was created to achieve the highest possible immersion. In addition, 3D sound emission from the UAV supports this objective, as well as other touchable 3D objects such as a cube or table.

4.2. PERFORMANCE ANALYSIS

To evaluate the system's performance, defined parameters were collected during simulations. The process is described in Simulation Setup. From these parameters, easily interpretable graphic images were generated. Recalling that data from the hands collected through the VR headset is passed via Unity to Gazebo, gets processed and sent back to Unity, there are three different types of charts to visualize the data. The first type shows the delay of the WebSocket connection from Unity to Gazebo. Thereby the position of the hands is subject of the calculation. Moreover, this position data is plotted to get an overview of how well the position of each component is tracked by the other. For the Gazebo position, there are two potential sources to record the data from. There is the /set_model_state topic where Unity publishes its coordinates to, and there is the topic /model_states which represents the actual position of the hands in the simulation. From the data, the delay from /set_model_state to /model_states was estimated to 0.01s. However, the /model_states topic was used to obtain the delay of this entire system. The overall computing delay from Gazebo was also added to the first chart of each experiment to show that all data was corrected so that there is no correlation to the real time factor of Gazebo in the presented data.

The second diagram type on the other hand contains the calculated delay from Gazebo to Unity. This time, the position of the UAV is used for calculations since the hands' positions are not returned to Unity. As well as in the first chart, positional data from both softwares are plotted as they are the calculation source of the delay.

Lastly, a third chart relates the position graph of the UAV to the measured forces of the force torque sensor of the model. This is to validate the sensors measurements as well as the correct implementation into the controller file.

For each of these three diagram types, there are three figures representing the X, Y and Z axis. Furthermore, every diagram only shows data from one experiment to keep each plot as simple as possible. Data was recorded from two simulation sessions, where the duration from simulation 1 was about 4.4 minutes and simulation 2 lasted about 3.5 minutes.

In the following, the results of the data analysis are described and plotted. Beginning with simulation 1, Figure 10 of the first type shows the simulation time in seconds on the X axis, the delay in seconds on the left and the position of the UAV in meters on the right. The simulation time start is as high as 1160 seconds because Gazebo begins every launch with 18 minutes of simulated time. First it is clear that performing the Gazebo simulation with a real time factor less than 1 leads to a rising delay in the timestamps. Since the real time factor is an indicator of the performance of the Gazebo simulation, it oscillates around 0.98 which means that every computing step of the simulation time needs 2% longer than the real time. In the runtime of the simulation, the internal Gazebo delay reached 1.25 s, which had to be corrected only for the analysis of the timestamps. The total system delay is not affected from this internal Gazebo delay, since all messages are sent and received in real time and are not using the simulation time from Gazebo. Furthermore, Figure 10 shows the trajectories of the UAV in X direction, one from Unity and the following in Gazebo. From this, the delay between those softwares is plotted. As the average of the delay in X axis is as small as 0.0353 s, both

trajectories mostly coincide in this graph. Mentioning the correction, it is also clear that the timestamp lag, depending on the computer's performance, has no influence on the determined Gazebo delay.

The same type of graph for Simulation 2 is given with Figure 11. In this setting, the Gazebo simulation ran with a higher real time factor which causes the timestamp delay to stay below 0.6 s. This indicates a better performance of Gazebo in simulation 2 compared to simulation 1. Depending on the computer's hardware and the type of operating system it is running on, Gazebo is part of a powerful setup as mentioned in 3.8 Simulation Setup. In comparison to virtual machines or standard office computers, Gazebo takes advantage of the gaming laptop that was available and the Windows Subsystem for Linux that was used for these simulations. The performance difference could be caused by temporary overheating, other unclosed programs on the computer or randomly emerging reasons. Besides the Gazebo performance, the total delay between Unity and Gazebo also shrinked to an average of 0.0172 s in X direction in simulation 2.

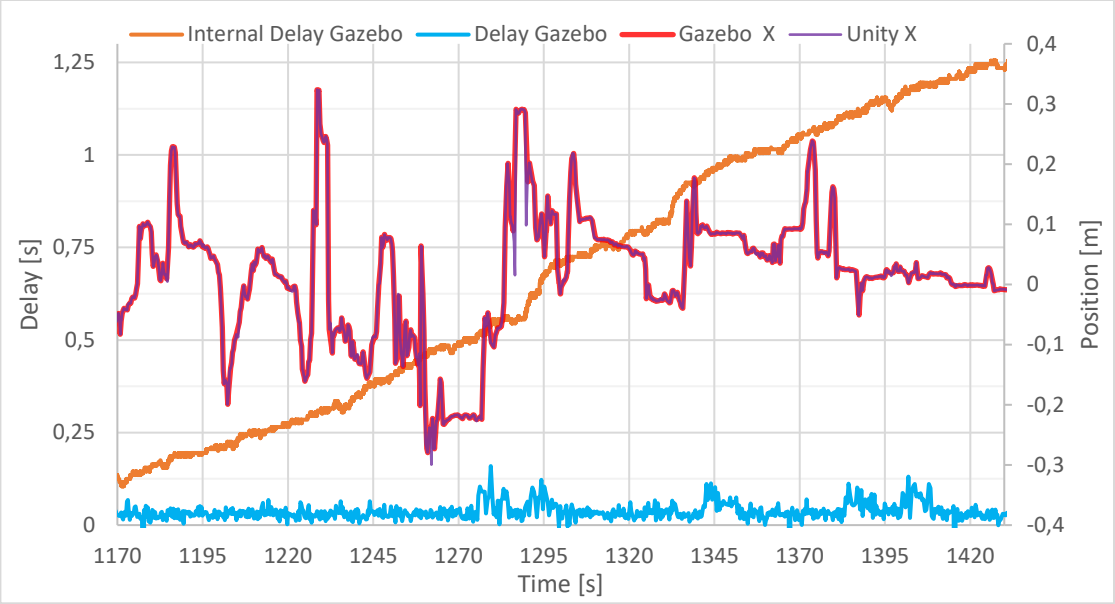

Figure 10: Gazebo Delay Simulation 1 X Axis

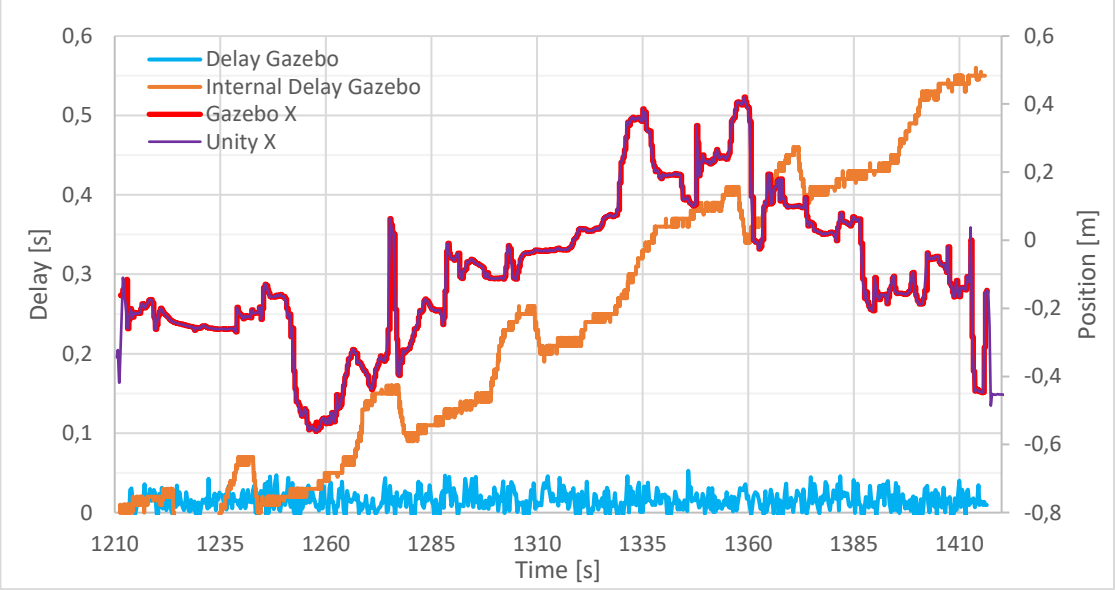

Figure 11: Gazebo Delay Simulation 2 X Axis

After processing the hands' position in Gazebo, interaction with the UAV is possible. The resultant trajectory of the UAV is sent back to Unity and is plotted in the following diagrams in addition to the delay of this process. The axes for the following

figures remain the same as in the first diagram type however, the delay scale was set logarithmic for simulation 1 to make small values in the beginning more visible. During these first 120 seconds, the delay stays at a low average of 0.161 s, whereas the overall average is 3.364 s. This lag is strongly noticeable when interacting with the UAV in the VR world and can be easily identified in Figure 12. Possible reasons for this increasing delay could be a queue of messages that are held back by the WebSocket connection and are cutting off the queue at some points when the delay decreases for a short period of time.

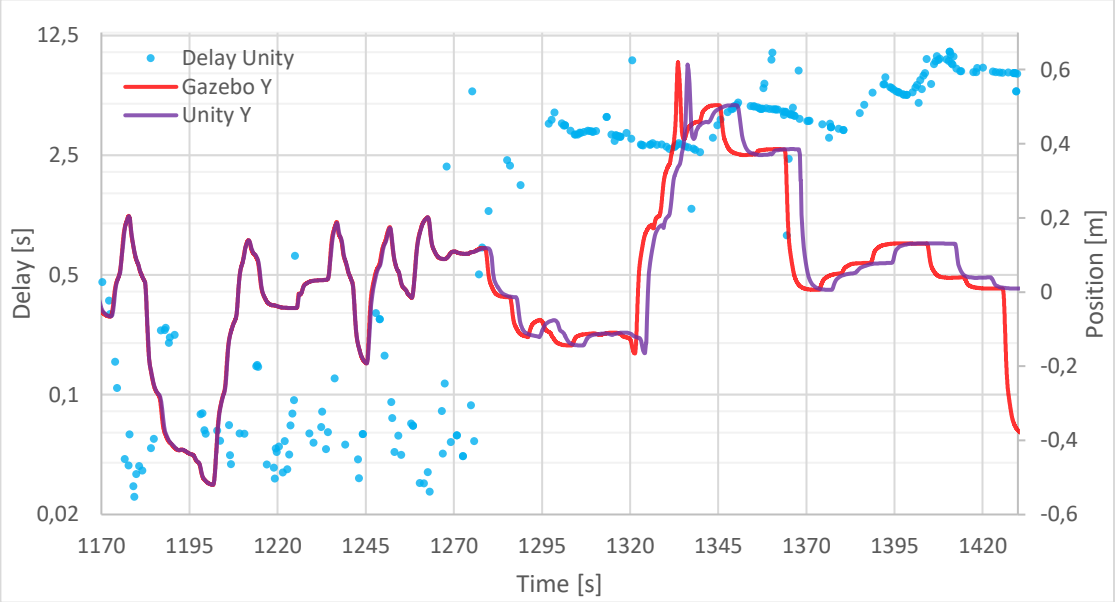

Figure 12: Unity Delay Simulation 1 Y Axis

Evaluating simulation 2, conclusions can be drawn that the first 100 seconds again have a good performance with a minimal delay of 0.0854 s. After that, Figure 13 illustrates a trend comparable to Figure 12, i.e. increasing to a peak with a following decline to the starting level. This again could be caused by a buffer of messages. Other reasons

for that could be found in windows background services, an unsteady network connection inside the network driver or throttling down of the CPU after heating up. However, as compared in Figure 14, on a different level of values since the highest increase in simulation 2 is only 0.62 s, whereas it is around 2.5 s in simulation 1. A conclusion that Figure 14 implies is that both simulations have the same order of magnitude in the delay during the first 120 seconds. The curves in the second part of each simulation differ, but the first section suggests consistently good performance at the start of each simulation. It has to be mentioned that the Y axis was selected to conclude these results because the plot of the delay was the cleanest of all three axes. Since that is the case for all diagrams the text only shows a selection of the recorded data, it also has to be referenced that all other figures can be found in the Appendix. Figure 15 therefore shows the relation between the delay curve of each axis in the same simulation. That being said, all three data sets have an almost similar distribution of points that form one single curve, although they have unique patterns around it. Especially the calculated delay in Z axis has almost vertically growing peaks at some points. One reason for that could be the different trajectory profile in Z axis which characteristic it is to stay at the same height for a longer time. These same positions can cause the delay calculation algorithm to output those peaks as a noise, even without being the messages actually delayed.

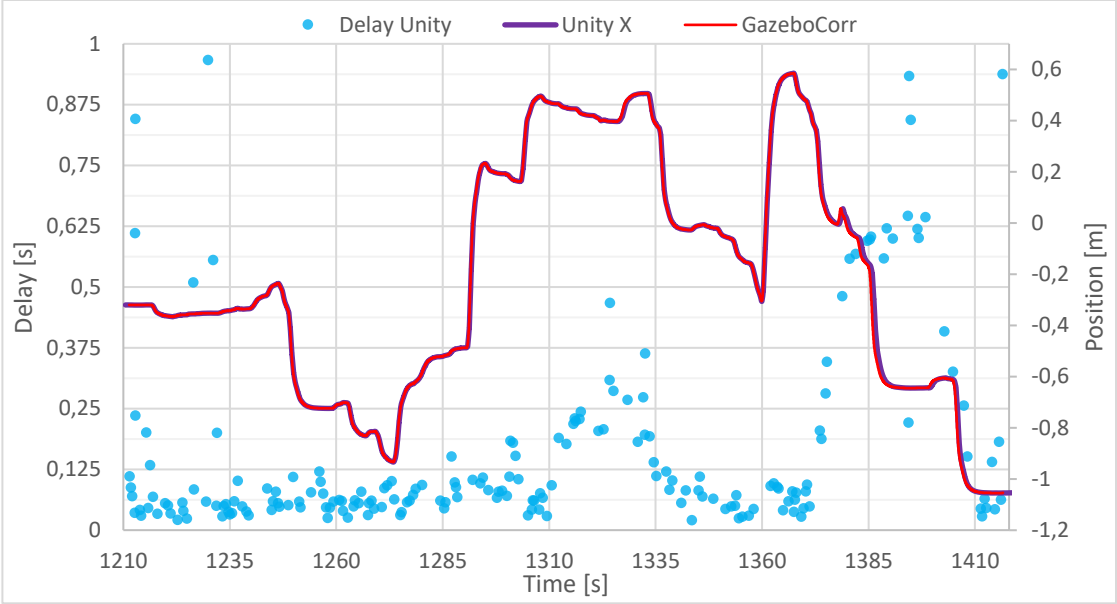

Figure 13: Unity Delay Simulation 2 Y Axis

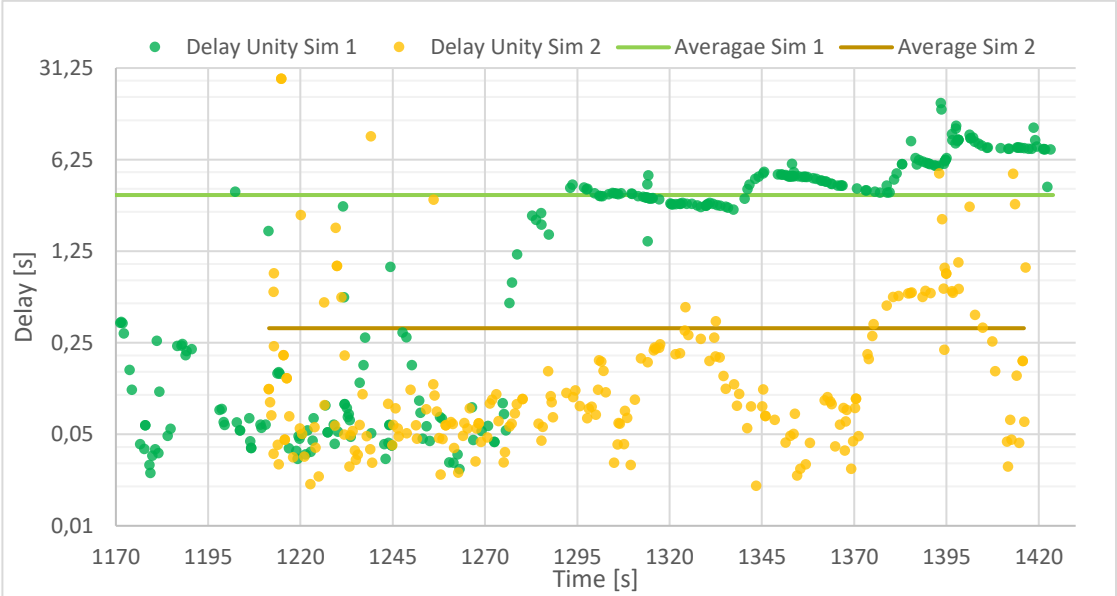

Figure 14: Unity delay comparison between Simulation 1 and 2 for Y Axis positions

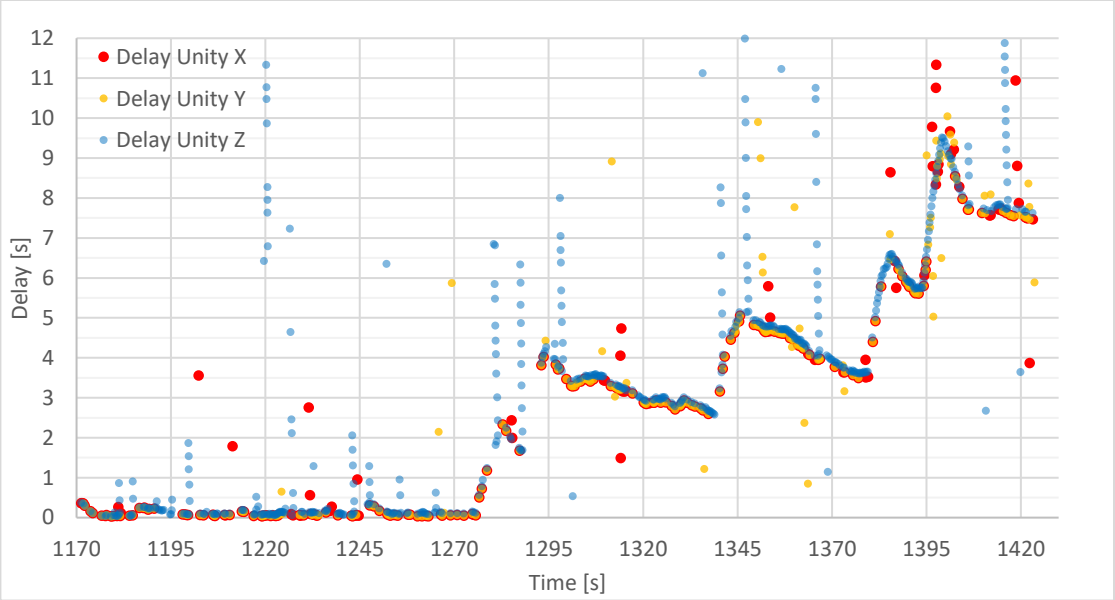

Figure 15: Unity delay comparison between X, Y and Z Axis Simulation 1. It shows the increasing delay after around 120 seconds.

Summarizing the results of the two simulations, the performance can vary with each simulation session. Hence, the delay in the first two minutes period is acceptable for the purpose of the application, which is a sufficiently fast response of the UAV to the contacts that are made within the Human-UAV-Interaction. The total delay of the platform therefore can be added to 3.4017 s in the first and 0.7732 s in the second simulation. Table 1 gives a short summary of all measured delays in the specific section of the system.

Table 1: Summary of Delays in Simulation 1 and 2

Measured Direction		Simulation 1	Simulation 2
←	Average Unity to Gazebo	0.0377 s	0.0172 s
→	Average Gazebo to Unity	3.364 s	0.756 s
→	Average Gazebo to Unity in first 120 seconds	0.161 s	0.0854 s
← →	**Total System Delay in first 120 seconds**	**0.1987 s**	**0.1026 s**
← →	**Total System Delay**	**3.4017 s**	**0.7732 s**

Finishing the results regarding the delay parameter, the third diagram type will be discussed in the following. Another performance index of the simulation in this thesis is the relationship between the forces measured by the force sensor on the UAV and the trajectory that was ultimately performed. Therefore, the following diagrams show the position of the selected axis in meters on the left, the force indicated in newton on the right and the simulation time in seconds again on the horizontal axis at the bottom. The expected behavior is that each change in position follows to a force peak and the amplitude of movement is proportional to the applied force.

The data of the first simulation shows that most of the executed movements are induced by a preceding force. However, the amplitude of the force in Figure 16 for example at time 1165 s stands in the very contrast to the performed motion in the positive X direction since the force is negligibly small but the motion is not. Generally speaking, the measured forces of the sensor are not the only cause of positional changes of the UAV.

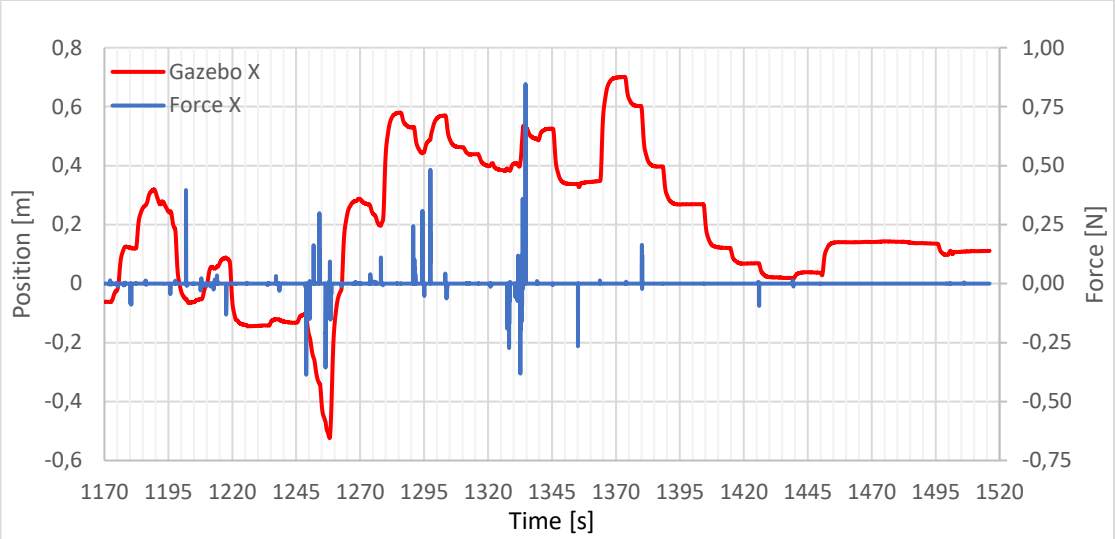

Figure 16: Force Trajectory Relation Simulation 1 X Axis

That phenomenon can be reasoned by taking other forces of the same sensor into account. Besides pushing the UAV along the desired direction, a movement can also be enforced through lifting or lowering the rim of the UAV so that its thrust is directed outwards the current position. It will then also hover into this direction without seeing the force into this specific axis. Instead, there is a force that was applied in the Z axis. This maneuver is illustrated in Figure 17. With this said, Figure 18 in addition shows the forces in Z axis to see the described process. On the other side, motion in Z direction cannot be influenced by some disturbance comparable to the just described one. In the Z axis diagrams of both simulations (Figure 19 and Figure 20), the direct response of the controller to the applied forces is visible.

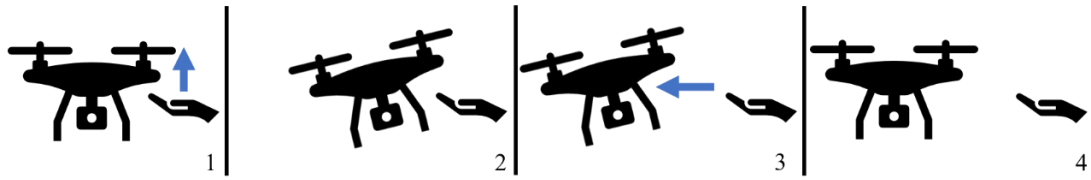

Figure 17: From force sensor unrecognized UAV manipulation. 1. Fingertips apply force in z-direction at a point on the UAV that is not the center 2. UAV tilts away from the hands 3. Thrust is directed not only in z-direction to the ground but also in the direction of the hands so that the UAV will move away from the hand 4. Velocity controller stabilizes UAV in new position

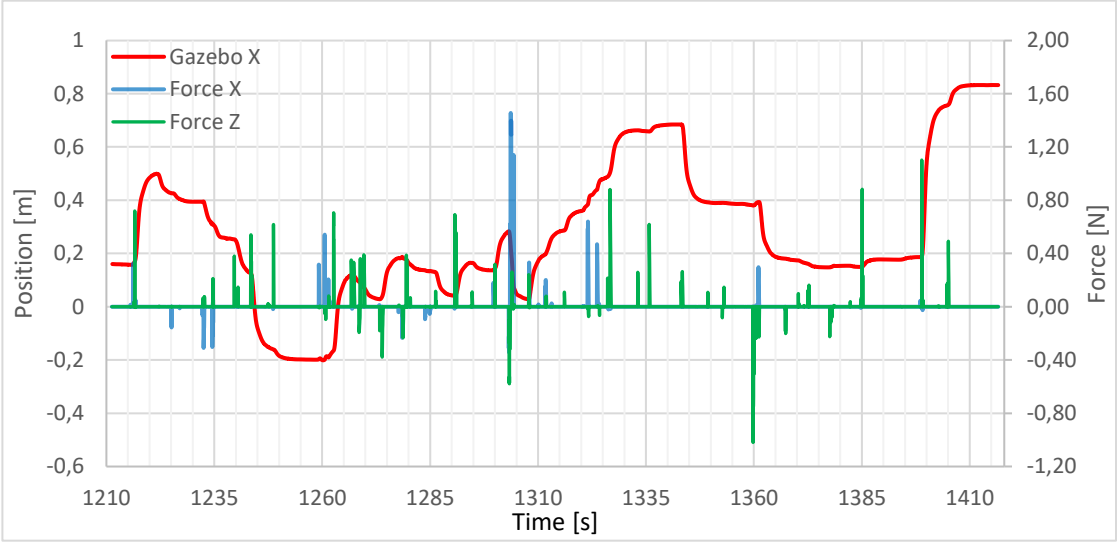

Figure 18: Force Trajectory Relation, Simulation 2 X Axis

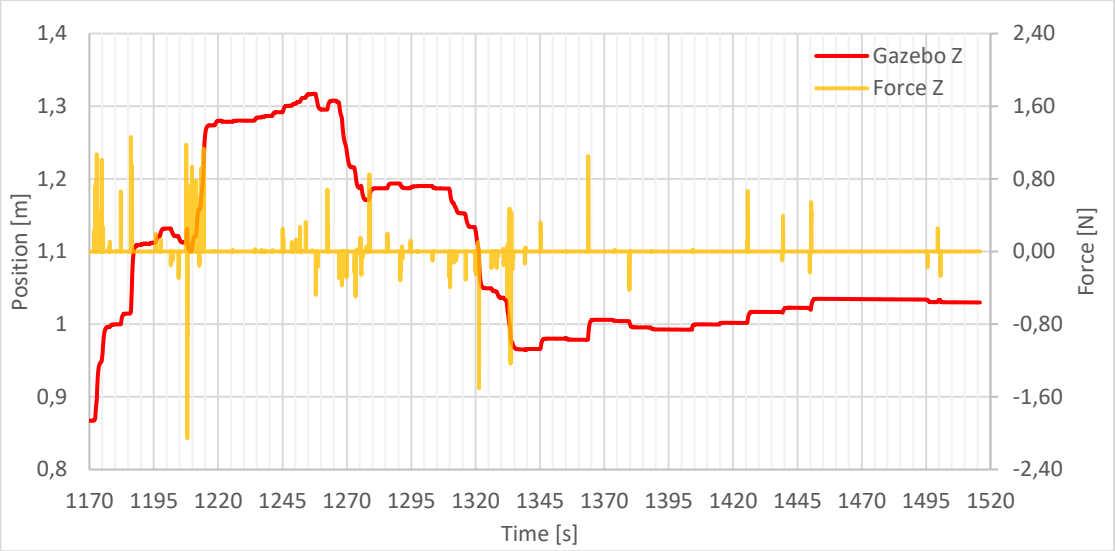

Figure 19: Force Trajectory Relation, Simulation 1 Z Axis

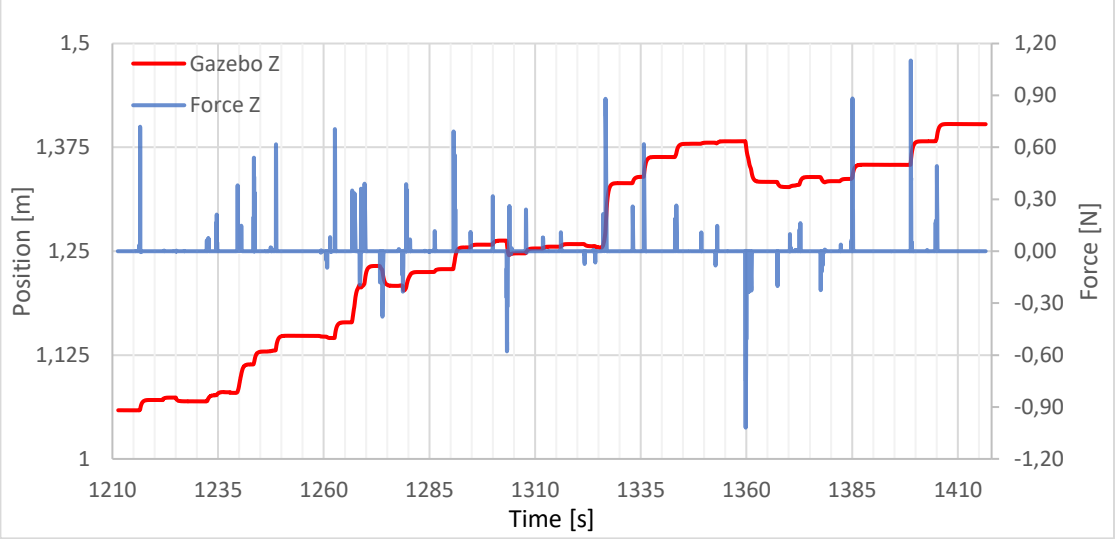

Figure 20: Force Trajectory Relation Simulation 2 Z Axis

Summarizing the performance analysis of the developed platform, all components demonstrated their capability of running the VR application in a sufficient speed so that Human interaction with UAVs can be further investigated. Regarding the accuracy of the force detection or in general the accuracy of the interaction, the refresh rates of all

components should be set to the highest possible level to achieve the best resolution at a sufficient performance for the VR experience. Especially the first time of each simulation and the performance increase between the first and the second simulation seem to offer further improvements. At the same time this improvement was not caused by any intended changes on the system so that it can be concluded that a repeatable performance is not given. This topic amongst others will be discussed in the next chapter.

5. DISCUSSION AND FUTURE WORK

In conclusion, this thesis showed the development process of a VR platform for interaction with Unmanned Aerial Vehicles, conducted two simulations to test the performance of the system and analyzed the recorded data. Besides the positive findings from the results, there are some limitations that will be presented in the following chapter as well as an outlook for further works.

The main limitation is the lack of intuitive interaction methods within the VR simulation. Since the hands are tracked and visualized in the OQ2 with great detail and accuracy, naturally the user tries to grab or pull the UAV towards himself at some time. This is not possible in this system architecture, because the model of the hands in Gazebo which controls the UAV is only based on collisions and does not allow grabbing into the model or accept other gesture input. Although the hands are fully articulated and tracked inside Unity, only the position of the fingertips and the wrists is used for the interaction and force sensing unit on the UAV, so that the possible accuracy in collision detection is not fully used.

As already discussed in the results chapter, the delay between Gazebo and Unity is sufficient but still has room for improvement. The WebSocket seem to be an important factor when it comes to latency generation, so that there may be some possible parameters on this connection to minimize the delay. Another option that must be taken into account at this point is the recently launched Unity Robotics Hub. This collection of tools is published from Unity and therefore directly is integrated into the game engine [29]. Furthermore, the biggest advantage of the Unity solution over ROS# could be the

faster connection with a faster serialization type for the ROS messages. Switching this platform over to the Unity solution can be subject to further investigation.

Another apparent limitation of the platform is the lack of research applications. Thinking of investigating the Human-UAV-Interaction, experimental scenarios have to be developed in which humans are supposed to complete some tasks involving the UAV and to record data from it for further evaluation. Those scenarios are not yet developed nor implemented in the software. This is an issue for future research to explore.

Although widely common in VR headsets, the required cable connection limits the range of motion and subsequently the form of interaction. While wearing the headset, user will have to pay attention to the cable what restricts the grade of immersion. A solution already is available with the possibility to run the application on the OQ2 itself without the cable connection but a Wi-Fi connection to the ROS simulation instead. This setup, however, performs in an unsteady and lagging manner that no single test was able to achieve the wanted behavior of the UAV. Future studies could investigate the bottleneck causing this issue.

Future work also could include the idea of haptic feedback while interacting with the virtual world. A collaborative robot arm in the real environment can move accordingly to the position of the UAV in the simulation so that it will be possible to touch the UAV with the haptic feeling of touching a real object. A robot arm may be added to the ROS simulation, that will follow the position of the UAV relative to the person's position. At a later stadium of this attempt, a real robotic arm could be connected to the setup to perform experiments. In the Intelligent Control and Robotics Laboratory at the University of Rhode Island, the collaborative robot arm Sawyer from Rethink Robotics

is available. Regarding Human-Robot-Interaction, safety aspects must be considered when working with real robotic arms. A feature that ensures an additional level of safety and opens new possible interaction modes is multi-camera-based body tracking. The OQ2 already tracks head and hand movements, whereas such camera system is able to track the whole body. With full body tracking not only the range of interaction methods is extended but also the human behavior during experiments can be examined more in depth with this additional information. This work thus offers many approaches for further research projects on Virtual Reality and Human-UAV Interaction.

6. REFERENCES

[1] Federal Aviation Administration, *FAA Aerospace Forecasts: Fiscal Years 2017-2037*. [Online] Available: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2017-37_FAA_Aerospace_Forecast.pdf. Accessed on: Jan. 31 2021.

[2] Federal Aviation Administration, *FAA Aerospace Forecasts: Fiscal Years 2020-2040*. [Online] Available: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/Unmanned_Aircraft_Systems.pdf. Accessed on: Jan. 31 2021.

[3] M. Avila, M. Funk, and N. Henze, "DroneNavigator," in *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, Lisbon, Portugal, 2015, pp. 327–328.

[4] K. Nitta, K. Higuchi, and J. Rekimoto, "HoverBall," in *Proceedings of the 5th Augmented Human International Conference*, Kobe Japan, 2014, pp. 1–4.

[5] S. Schneegass, F. Alt, J. Scheible, A. Schmidt, and H. Su, "Midair displays," in *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, Toronto Ontario Canada, 2014, pp. 2035–2040.

[6] D. Tezza and M. Andujar, "The State-of-the-Art of Human–Drone Interaction: A Survey," *IEEE Access*, vol. 7, pp. 167438–167454, 2019.

[7] A. Gomes, C. Rubens, S. Braley, and R. Vertegaal, "BitDrones," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, San Jose California USA, 05072016, pp. 770–780.

[8] P. Abtahi, B. Landry, J. Yang, M. Pavone, S. Follmer, and J. A. Landay, "Beyond The Force," in *CHI 2019: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, May 4-9, 2019, Glasgow, Scotland UK*, Glasgow Scotland Uk, 2019, pp. 1–13.

[9] P. Knierim, T. Kosch, A. Achberger, and M. Funk, "Flyables," in *Proceedings of the Twelfth International Conference on Tangible, Embedded, and Embodied Interaction*, Stockholm Sweden, 2018, pp. 329–336.

[10] P. Knierim, T. Kosch, V. Schwind, M. Funk, F. Kiss, S. Schneegass, and N. Henze, "Tactile Drones - Providing Immersive Tactile Feedback in Virtual Reality through Quadcopters," in *CHI'17 Extended Abstracts: Proceedings of the 2017 ACM SIGCHI Conference on Human Factors in Computing Systems*, Denver Colorado USA, 2017, pp. 433–436.

[11] M. Hoppe, P. Knierim, T. Kosch, M. Funk, L. Futami, S. Schneegass, N. Henze, A. Schmidt, and T. Machulla, "VRHapticDrones," in *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia*, Cairo Egypt, 2018, pp. 7–18.

[12] P. Abtahi, D. Y. Zhao, J. L. E., and J. A. Landay, "Drone Near Me," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 1–8, 2017.

[13] C. F. Liew and T. Yairi, "Companion Unmanned Aerial Vehicles: A Survey," 2020. [Online] Available: https://www.semanticscholar.org/paper/Companion-Unmanned-Aerial-Vehicles%3A-A-Survey-Liew-Yairi/630630c4feefaf78e6370736b99cac1082041c4b. Accessed on: Feb. 02 2021.

[14] S. Rajappa, H. Bülthoff, and P. Stegagno, “Design and implementation of a novel architecture for physical human-UAV interaction,” *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 800–819, 2017.
[15] U. Technologies, *Wondering what Unity is? Find out who we are, where we've been and where we're going | Unity.* [Online] Available: https://unity.com/our-company. Accessed on: Jun. 09 2021.
[16] U. Technologies, *Unity - Unity.* [Online] Available: https://unity.com/. Accessed on: Jun. 04 2021.
[17] *ROS.org | About ROS.* [Online] Available: https://www.ros.org/about-ros/. Accessed on: Jun. 10 2021.
[18] *ROS.org | Core Components.* [Online] Available: https://www.ros.org/core-components/. Accessed on: Jun. 10 2021.
[19] designnews.com, *ROS 101: An Intro to the Robot Operating System.* [Online] Available: https://www.designnews.com/gadget-freak/ros-101-intro-robot-operating-system. Accessed on: Jun. 28 2021.
[20] Open Source Robotics Foundation, *Gazebo - Features.* [Online] Available: gazebosim.org. Accessed on: Jun. 11 2021.
[21] VRScout, *Hands-On: Oculus Quest Hand Tracking Feels Great, But It's Not Perfect - VRScout.* [Online] Available: https://vrscout.com/news/hands-on-oculus-quest-hand-tracking/. Accessed on: Jun. 28 2021.
[22] Victor D, “Oculus Quest 2 review,” *GSMArena*, 15 Nov., 2020, https://www.gsmarena.com/oculus_quest_2_review-news-46255.php.
[23] GitHub, *siemens/ros-sharp.* [Online] Available: https://github.com/siemens/ros-sharp. Accessed on: Jun. 17 2021.
[24] GitHub, *siemens/ros-sharp.* [Online] Available: https://github.com/siemens/ros-sharp/wiki/Info_Showcases. Accessed on: Jun. 17 2021.
[25] *Oculus Integration | Integration | Unity Asset Store.* [Online] Available: https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022. Accessed on: Jun. 21 2021.
[26] *Quadcopter controller (PC||Mobile) | 3D Characters | Unity Asset Store.* [Online] Available: https://assetstore.unity.com/packages/3d/characters/quadcopter-controller-pc-mobile-61829. Accessed on: Jun. 25 2021.
[27] *rosbridge_suite - ROS Wiki.* [Online] Available: http://wiki.ros.org/rosbridge_suite?distro=kinetic. Accessed on: Jun. 26 2021.
[28] GitHub, *pulver22/QLAB.* [Online] Available: https://github.com/pulver22/QLAB. Accessed on: Jun. 26 2021.
[29] GitHub, *Unity-Technologies/Unity-Robotics-Hub.* [Online] Available: https://github.com/Unity-Technologies/Unity-Robotics-Hub. Accessed on: Jul. 10 2021.

7. APPENDIX

The two simulations accumulated a big amount of data. Since not all figures of the simulations were required to point out the results, they can be reviewed here for further interest. Each simulation generated three diagrams for the X,Y and Z axis for each of the three type of diagrams described in the Results chapter.

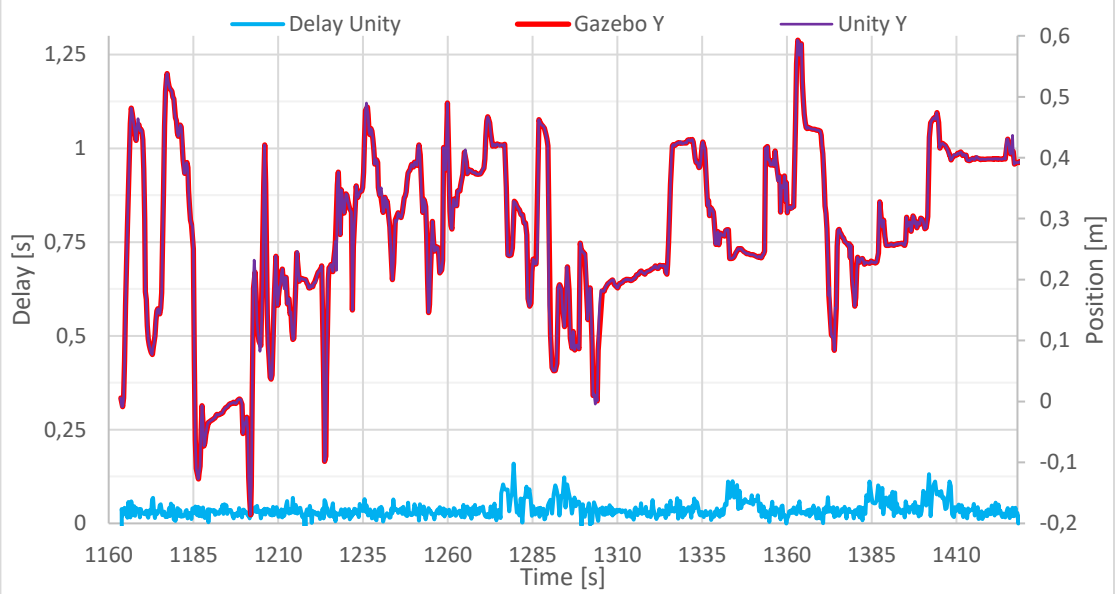

Figure 21: Gazebo Delay Simulation 1 Y Axis

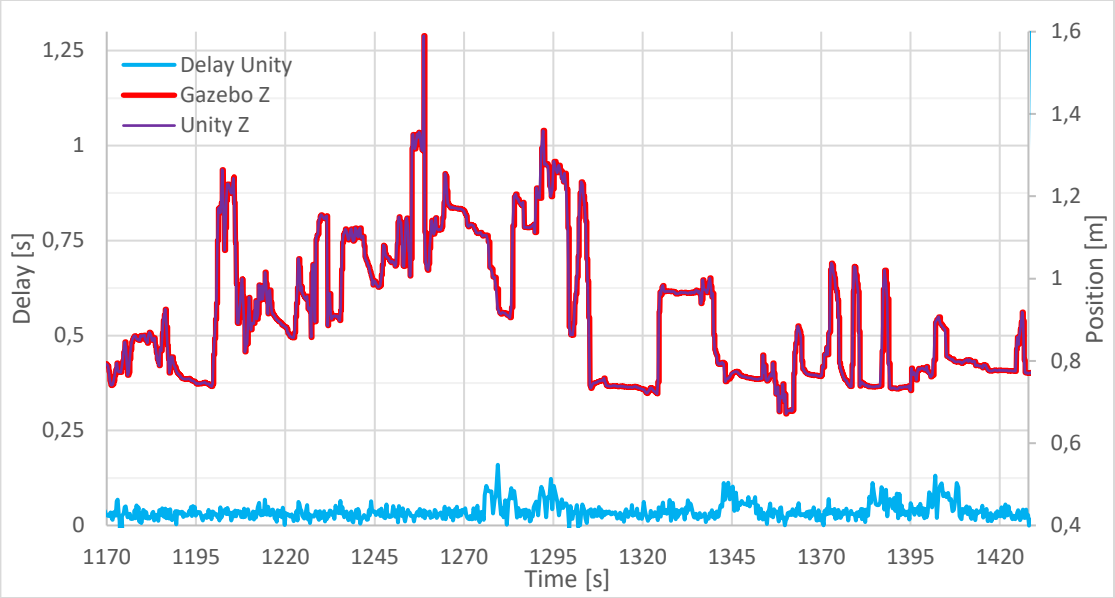

Figure 22:Gazebo Delay Simulation 1 Z Axis

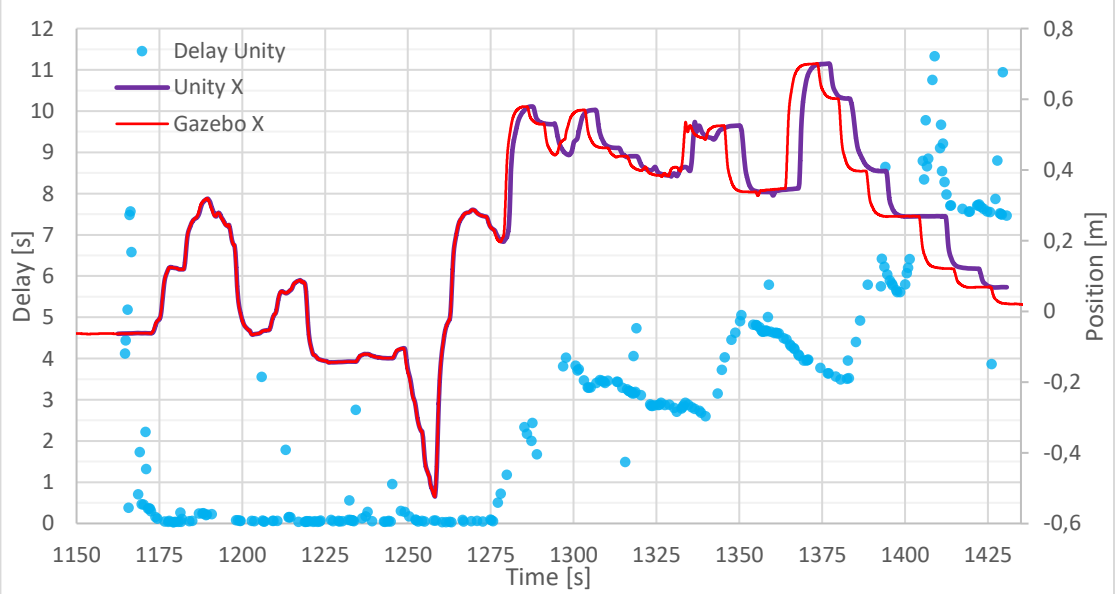

Figure 23: Unity Delay Simulation 1 X Axis

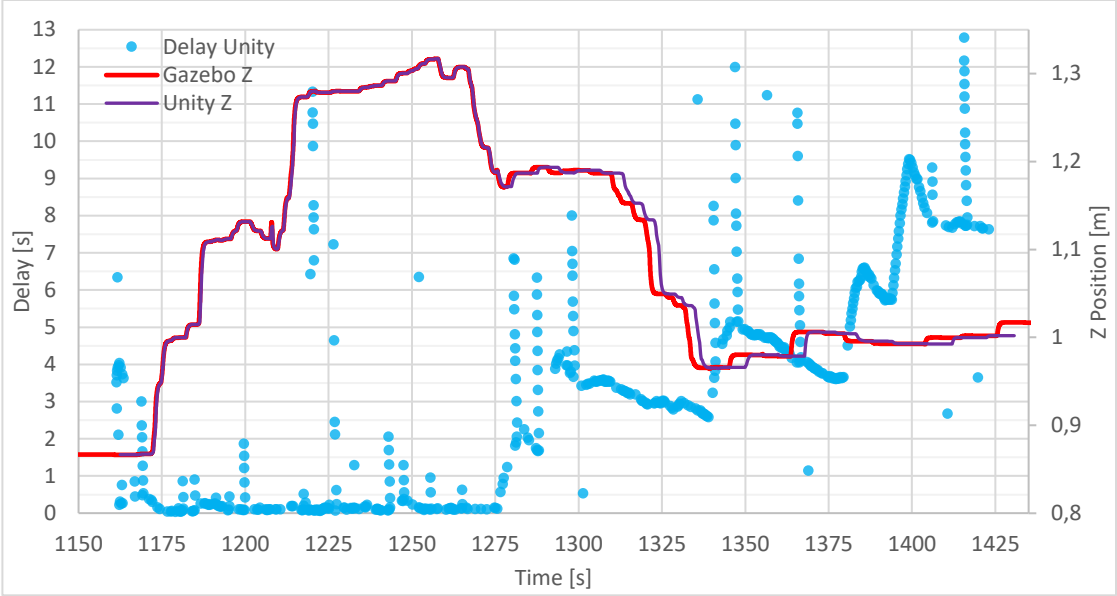

Figure 24: Unity Delay Simulation 1 Z Axis

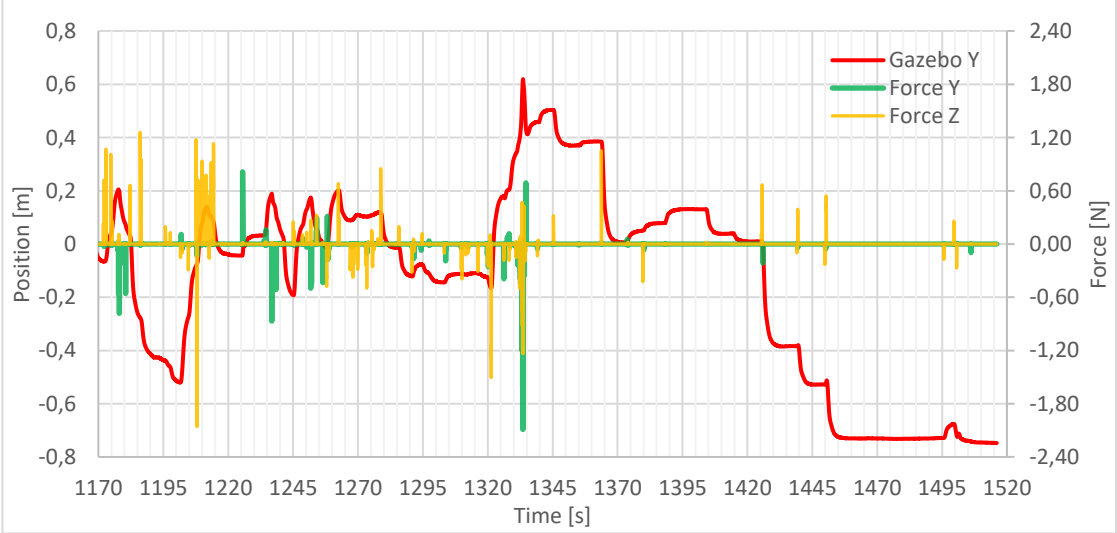

Figure 25: Force Trajectory Relation Simulation 1 Y Axis

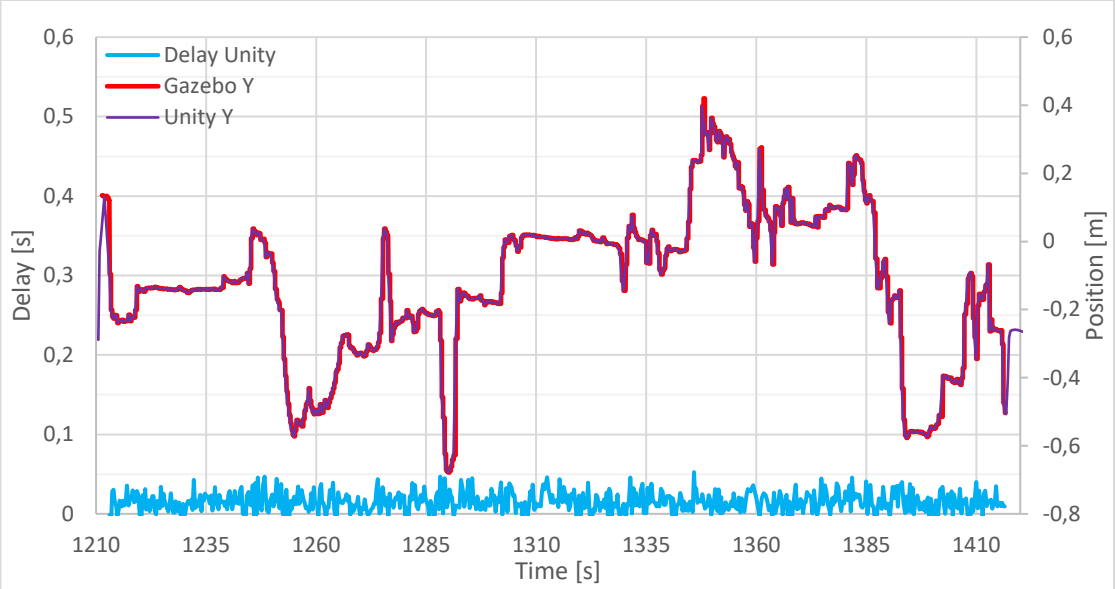

Figure 26: Gazebo Delay Simulation 2 Y Axis

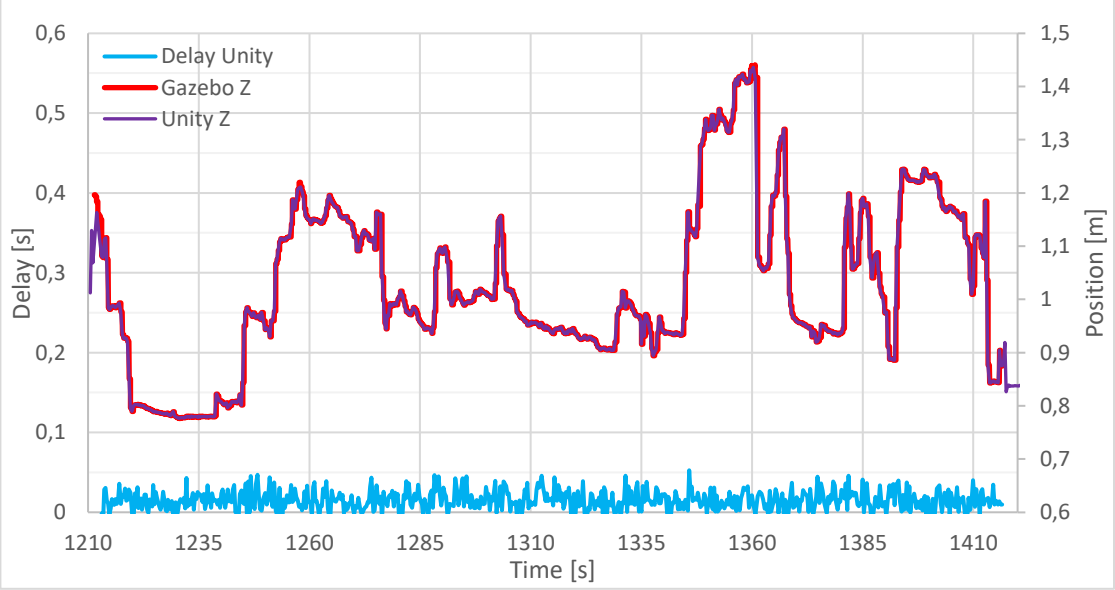

Figure 27: Gazebo Delay Simulation 2 Z Axis

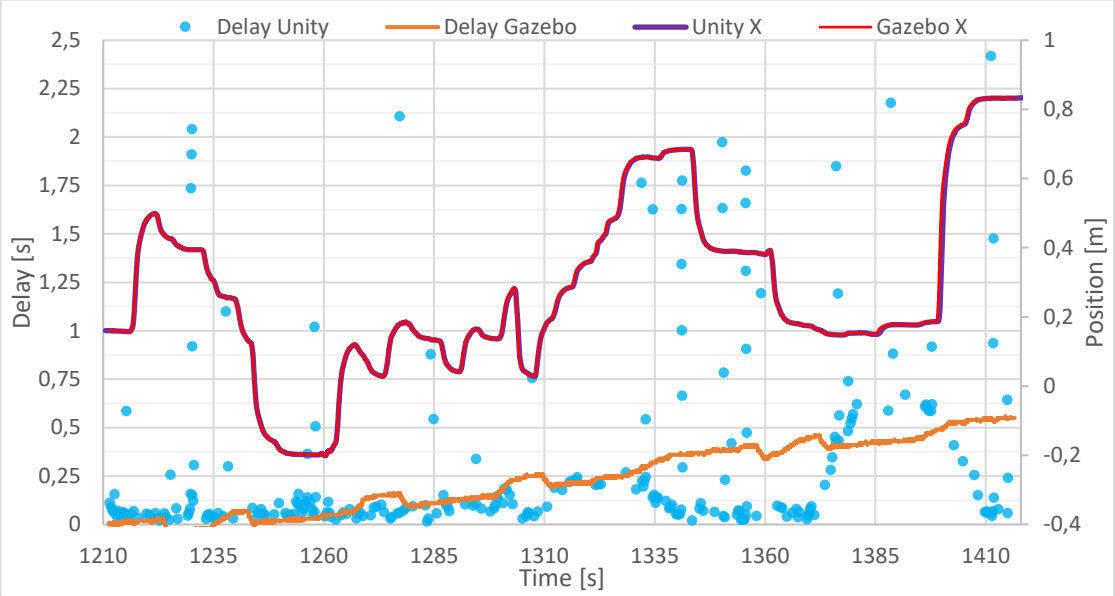

Figure 28: Unity Delay Simulation 2 X Axis

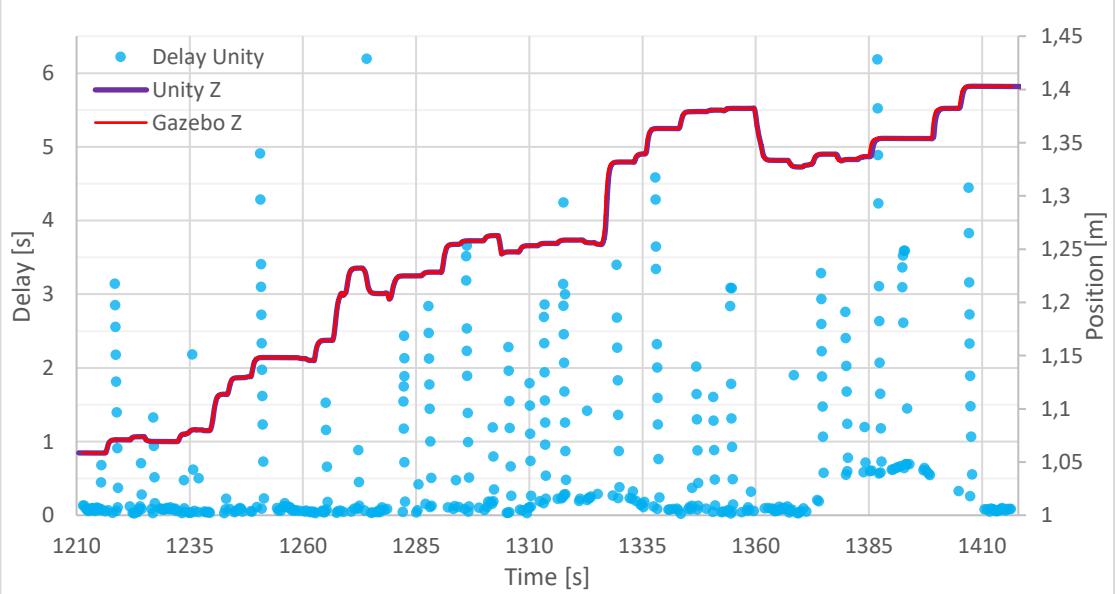

Figure 29: Unity Delay Simulation 2 Z Axis

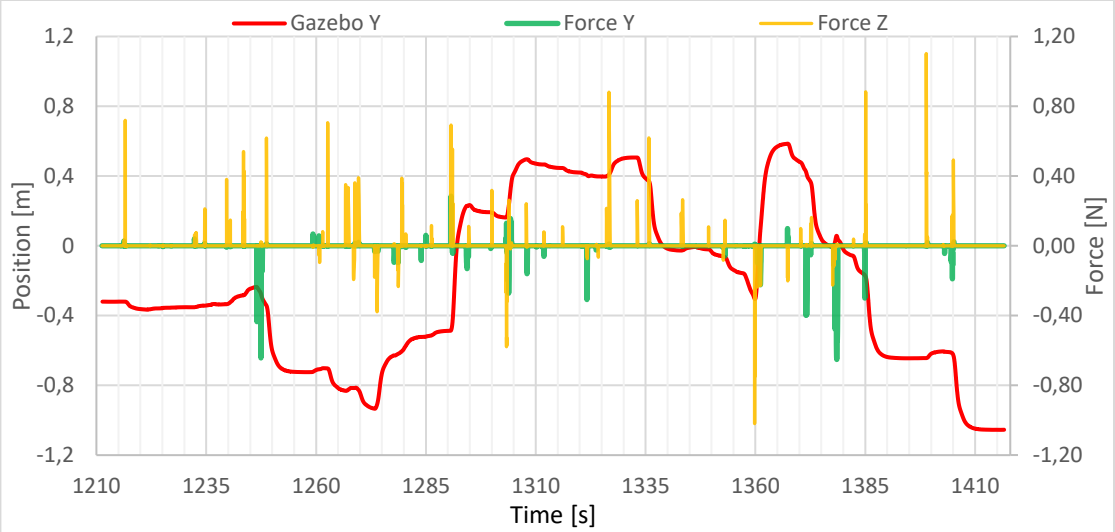

Figure 30: Force Trajectory Relation Simulation 2 Y Axis

8. BIBLIOGRAPHY

Abtahi, P., Landry, B., Yang, J., Pavone, M., Follmer, S., & Landay, J. A. (2019). Beyond The Force. In S. Brewster, G. Fitzpatrick, A. Cox, & V. Kostakos. New York, NY: ACM.

Abtahi, P., Zhao, D. Y., E., J. L., & Landay, J. A. (2017). Drone Near Me. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 1*(3), pp. 1–8.

Avila, M., Funk, M., & Henze, N. (2015). DroneNavigator. In Y. Yesilada, & J. P. Bigham. New York, NY: ACM.

C. F. Liew, & T. Yairi. (2020). Companion Unmanned Aerial Vehicles: A Survey.

designnews.com. (2019). *ROS 101: An Intro to the Robot Operating System*. Retrieved 6 28, 2021, from https://www.designnews.com/gadget-freak/ros-101-intro-robot-operating-system

Federal Aviation Administration. (2017). *FAA Aerospace Forecasts*. Retrieved 1 31, 2021, from Fiscal Years 2017-2037: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2017-37_FAA_Aerospace_Forecast.pdf

Federal Aviation Administration. (2020). *FAA Aerospace Forecasts*. Retrieved 1 31, 2021, from Fiscal Years 2020-2040: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/Unmanned_Aircraft_Systems.pdf

GitHub. (n.d.). *pulver22/QLAB*. Retrieved 6 26, 2021, from https://github.com/pulver22/QLAB

GitHub. (n.d.). *siemens/ros-sharp*. Retrieved 6 17, 2021, from https://github.com/siemens/ros-sharp/wiki/Info_Showcases

GitHub. (n.d.). *siemens/ros-sharp*. Retrieved 6 17, 2021, from https://github.com/siemens/ros-sharp

GitHub. (n.d.). *Unity-Technologies/Unity-Robotics-Hub*. Retrieved 7 10, 2021, from https://github.com/Unity-Technologies/Unity-Robotics-Hub

Gomes, A., Rubens, C., Braley, S., & Vertegaal, R. (05072016). BitDrones. In J. Kaye. New York NY: ACM.

Hoppe, M., Knierim, P., Kosch, T., Funk, M., Futami, L., Schneegass, S., . . . Machulla, T. (2018). VRHapticDrones. In S. Abdennadher. [Place of publication not identified]: ACM.

Ikeuchi, K., Otsuka, T., Yoshii, A., Sakamoto, M., & Nakajima, T. (2014). KinecDrone. In T. Terada, M. Inami, K. Kunze, & T. Nojima. New York, NY: ACM.

Knierim, P., Kosch, T., Achberger, A., & Funk, M. (2018). Flyables. In Y. Fernaeus. New York, NY: ACM.

Knierim, P., Kosch, T., Schwind, V., Funk, M., Kiss, F., Schneegass, S., & Henze, N. (2017). Tactile Drones - Providing Immersive Tactile Feedback in Virtual Reality through Quadcopters. In G. Mark, S. R. Fussell, C. Lampe, M. C. Schraefel, J. P. Hourcade, C. Appert, & D. Wigdor. New York: ACM.

Matthias Hoppe, Thomas Kosch, Pascal Knierim, Markus Funk, & Albrecht Schmidt. (2019). Are Drones Ready for Takeoff? Reflecting on Challenges and Opportunities in Human-Drone Interfaces.

Nitta, K., Higuchi, K., & Rekimoto, J. (2014). HoverBall. In T. Terada, M. Inami, K. Kunze, & T. Nojima. New York, NY: ACM.

Oculus Integration | Integration | Unity Asset Store. (n.d.). Retrieved 6 21, 2021, from https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022

Open Source Robotics Foundation. (2014). *Gazebo - Features*. Retrieved 6 11, 2021, from gazebosim.org

Quadcopter controller (PC||Mobile) | 3D Characters | Unity Asset Store. (n.d.). Retrieved 6 25, 2021, from https://assetstore.unity.com/packages/3d/characters/quadcopter-controller-pc-mobile-61829

Rajappa, S., Bülthoff, H., & Stegagno, P. (2017). Design and implementation of a novel architecture for physical human-UAV interaction. *The International Journal of Robotics Research, 36*(5-7), pp. 800–819.

ROS.org | About ROS. (n.d.). Retrieved 6 10, 2021, from https://www.ros.org/about-ros/

ROS.org | Core Components. (n.d.). Retrieved 6 10, 2021, from https://www.ros.org/core-components/

rosbridge_suite - ROS Wiki. (n.d.). Retrieved 6 26, 2021, from http://wiki.ros.org/rosbridge_suite?distro=kinetic

Schneegass, S., Alt, F., Scheible, J., Schmidt, A., & Su, H. (2014). Midair displays. In A. Schmidt, M. A. Jones, P. Palanque, & M. Jones. [Place of publication not identified]: ACM.

Technologies, U. (n.d.). *Unity - Unity*. Retrieved 6 4, 2021, from https://unity.com/

Technologies, U. (n.d.). *Wondering what Unity is? Find out who we are, where we've been and where we're going | Unity*. Retrieved 6 9, 2021, from https://unity.com/our-company

Tezza, D., & Andujar, M. (2019). The State-of-the-Art of Human–Drone Interaction: A Survey. *IEEE Access, 7*, pp. 167438–167454.

Tsykunov, E., & Tsetserukou, D. (2019). WiredSwarm: High Resolution Haptic Feedback Provided by a Swarm of Drones to the User's Fingers for VR interaction. In T. Trescak. New York,NY,United States: Association for Computing Machinery.

Victor D. (2020). Oculus Quest 2 review.

VRScout. (2019). *Hands-On: Oculus Quest Hand Tracking Feels Great, But It's Not Perfect - VRScout*. Retrieved 6 28, 2021, from https://vrscout.com/news/hands-on-oculus-quest-hand-tracking/

Yamaguchi, K., Kato, G., Kuroda, Y., Kiyokawa, K., & Takemura, H. (2016). A Non-grounded and Encountered-type Haptic Display Using a Drone. In C. Sandor, R. Teather, E. Suma, & K. Johnsen. New York, NY: The Association for Computing Machinery.