NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Robot Remote Control based on Augmented Reality Glasses

Charalampos-Michail K. Katimertzis

Supervisor: Stathes P. Hadjiefthymiades, Professor

ATHENS

JUNE 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τηλεχειρισμός ρομπότ βασισμένος σε γυαλιά επαυξημένης πραγματικότητας

Χαράλαμπος-Μιχαήλ Κ. Κατιμερτζής

Επιβλέπων: **Ευστάθιος Π. Χατζηευθυμιάδης**, Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2022

BSc THESIS

Robot Remote Control based on Augmented Reality Glasses

Charalampos-Michail K. Katimertzis

S.N.: 1115201600062

SUPERVISOR: Stathes P. Hadjiefthymiades, Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τηλεχειρισμός ρομπότ βασισμένος σε γυαλιά επαυξημένης πραγματικότητας

Χαράλαμπος-Μιχαήλ Κ. Κατιμερτζής

Α.Μ.: 1115201600062

Επιβλέπων: **Ευστάθιος Π. Χατζηευθυμιάδης**, Καθηγητής

ABSTRACT

In recent years, the rapid development in the field of robotics has brought an impressive potential to the technology environment. It has been developed a wide range of applications for node devices in order to serve the control of information in remote locations but also in places where the human body can not even approach. This development has created the need for new matchings of new technologies and innovative methods in robot control.

In this thesis, we will focus on the remote control of a robot with the technology of augmented reality. Specifically, we will be receiving a live video stream from a ROS robot to Microsoft's AR glasses, Hololens and we will navigate back the robot from them through a Kafka Server.

The experiments were supported by the Ubuntu 16.04 operating system, the Gazebo, and Rviz simulators, a Turtlebot 2 with a raspberry 3 that is running the ROS operating system, as well as a XBOX Kinect sensor with a color camera and a depth sensor.

SUBJECT AREA: Augmented Reality, Mixed Reality, Autonomous Navigation

KEYWORDS: ROS, navigation, Hololens, turtlebot, movement control

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια, η ραγδαία ανάπτυξη στον τομέα της ρομποτικής έχει φέρει εντυπωσιακές δυνατότητες στο περιβάλλον της τεχνολογίας. Έχει αναπτυχθεί ένα ευρύ φάσμα εφαρμογών με συσκευές κόμβους ώστε να εξυπηρετείται ο έλεγχος πληροφορίας σε απομακρυσμένες τοποθεσίες, αλλά και μέρη όπου το ανθρώπινο σώμα δεν μπορεί καν να πλησιάσει. Αυτή η εξέλιξη έχει δημιουργήσει την ανάγκη για πρωτότυπα ταιριάσματα νέων τεχνολογιών και καινοτόμες μεθόδους στον έλεγχο ρομπότ.

Σε αυτή τη διπλωματική εργασία θα επικεντρωθούμε στον τηλεχειρισμό ενός ρομπότ με την τεχνολογία της επαυξημένης πραγματικότητας. Συγκεκριμένα, θα λάβουμε στην AR συσκευή Hololens της Microsoft μία ζωντανή ροή βίντεο από ένα ρομπότ με λειτουρικό ROS, και θα το πλοηγήσουμε από τα γυαλιά μέσω ενός διακομιστή Kafka.

Τα πειράματα υποστηρίχθηκαν από το λειτουργικό σύστημα Ubuntu 16.04, τους προσομοιωτές Gazebo και Rviz, ένα Turtlebot 2 με raspberry 3 που εκτελεί το λειτουργικό σύστημα ROS, καθώς και έναν αισθητήρα XBOX Kinect με έγχρωμη κάμερα και αισθητήρα βάθους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επαυξημένη Πραγματικότητα, Μεικτή Πραγματικότητα, Αυτόνομη Πλοήγηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ROS, πλοήγηση, Hololens, turtlebot, έλεγχος κίνησης

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor professor, Stathes P. Hadjiefthymiades, who gave me the chance to work in the field of robotics and AR and trusted me with this thesis. I would also like to extend my deepest gratitude to Dr. Kyriaki Panagidi for her constant support throughout the duration of this project and Nektarios Deligiannakis for his invaluable contribution to this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Over recent years, significant growth has been observed in the Internet of Things (IoT) field. The advancement of technology in the performance of processors, the size of the devices, and the wireless data transmission speed have created favorable conditions for IoT solutions to a big variety of problems. With the term IoT, we refer to a network of physical objects, that are equipped with sensors, and suitable software and are connected to each other via the Internet. The main purpose of the devices is to collect and exchange data, in order to interact with the physical world. In this Thesis, we will see the UGVs which stands for Unmanned Ground Vehicles which is a subfield of the IoT area. UGVs can be referred to as IoT because there are connected mobile nodes that are equipped with sensors and can navigate in the environment and react to specific events, thus meeting the conditions of the definition.

The second area that we will examine in this thesis, is Augmented reality (AR). AR is an interactive experience of a real-world environment, where the objects in the real world are enhanced by computer-generated perceptual information, sometimes across multiple sensory modalities, including visual, haptic, and auditory somatosensory. For this thesis, AR is an excellent user interface and an innovative way for the remote interaction of the user with the UGV devices. This technology matching opens up a whole new range of possibilities that brings the benefits of augmented reality in the robot navigation field.

The purpose of this thesis is to help mature a software solution for the custom type of devices communication like the one we examined in the previous paragraph, and help to approach extra interest to the extraction of Augmented Reality’s possibilities in other fields such as Robotics and IoT.

The software for this type of solution and the combination of these two technologies is not so mature at the time of writing this thesis. Specifically, a desirable choice would be an open-source solution that combines two devices dominant in their field. The Devices are Microsoft Hololens 2 Augmented Reality Glasses and the TurtleBot 2 which run on ROS operating system. So the problem isn’t a standardized solution, because the need is specific enough. It is necessary to exist a custom solution that allows us to live-stream the robot’s video feed from the ROS environment to Hololens and reliable navigate back the turtlebot from the Glasses.

2. UNMANNED VEHICLES

2.1 Definition of Unmanned Vehicles

The term “unmanned vehicle” defines a vehicle that operates without the need of a human to control it. The operation of these machines can be successful remotely. For example, an operator can use a computer to send messages to a vehicle through a streaming platform like Apache Kafka. A second case is an unmanned vehicle operating autonomously with the help of installed sensors by running algorithms that process these incoming data.

The users can be divided into two categories. At first, the user is in the same area with the device to observe it. In the second category, the person can be far away from the vehicle and keep track of it by relying exclusively on the help of sensors. For example, the user can be in a computer room and look at the vehicle with a camera. The teleoperation is time-sensitive, so it must have smooth streaming of the incoming data to succeed. Many scientists have been working on this problem in the last years to find efficient and innovative ways to teleoperate unmanned vehicles as their applications grow. These vehicles have a lot of applications such as exploration of inaccessible environments like space and ocean, support in rescue teams, military, hobby, and much more. In this Thesis, the control of the vehicle is possible through an augmented reality kit and can be done explicitly based on sensors. The vehicles are categorized depending on the environment that they are in. The categories are unmanned aerial vehicles - UAVs or drones, unmanned Ground Vehicles - UGVs, unmanned sea surface vehicles - USSVs/USVs, unmanned underwater vehicles - UUVs or underwater drones.

2.1.1 Unmanned Ground Vehicles

Unmanned ground vehicles - UGVs perform on the surface without the need of a human to be on the vehicle. They can be controlled remotely or work entirely autonomously with specialized algorithms. UGVs are very useful when we don’t have access to someplace or the environment is too dangerous for a human to be present. For example, rescue situations like building debris or space missions like the Mars Rover, a vehicle to explore Mars’ surface. Another use that has rapid growth is self-driving cars. A ground transportation vehicle can drive itself and avoid obstacles based on the help of sensors and software that interact with its environment. Those specific UGVs will have a massive impact on our daily life. They can release substantial human effort and person-hours from driving. It will be a revolution in cars’ safety and accidents reduction. Many companies give their best on research to make autonomous driving better and much stabler.

2.1.2 Unmanned aerial vehicle

Figure 1 - Unmanned aerial vehicle

Unmanned aerial vehicles - UAVs, commonly known as drones, i.e. an aircraft without the need of a human to be on board. Initially, drones are developed through the twentieth century to help military missions. However, by the twenty-first century, the benefits that drones gave us and the fields that became necessary are a lot more. These include aerial photography, agriculture, science, policing, product deliveries, surveillance, infrastructure inspections, and drone racing.

A field that deserves special mention is autonomous drone transportation, like self-driving cars will be a powerful means of transport. In Asia, the autonomous drone taxis' first-off tests have already begun. Traditional infrastructure such as large airports or runways is not required, suiting the scenario demands of urban air mobility and serving as an effective way to relieve the current traffic congestion pressures. The technology of autonomous flight eliminates the possibility of failure or malfunction caused by man-made errors. Without any concern about controlling or operating the aircraft, the passengers can just sit and enjoy the journey. In an emergency, 5G networks offer the ability of smooth communication for the remote control of the aircraft and real-time transmission of flight data.

Figure 2 - Drone Taxi

2.1.3 Unmanned Underwater Vehicles

Unmanned Underwater Vehicles - UUVs, also known as underwater drones, are submarine-like vehicles that can operate underwater without a human occupant. We have two subcategories for them, remotely operated underwater vehicles - ROUVs and autonomous underwater vehicles - AUVs. The most common uses of UUVs are military purposes and deep-ocean exploration, documentaries, and research.

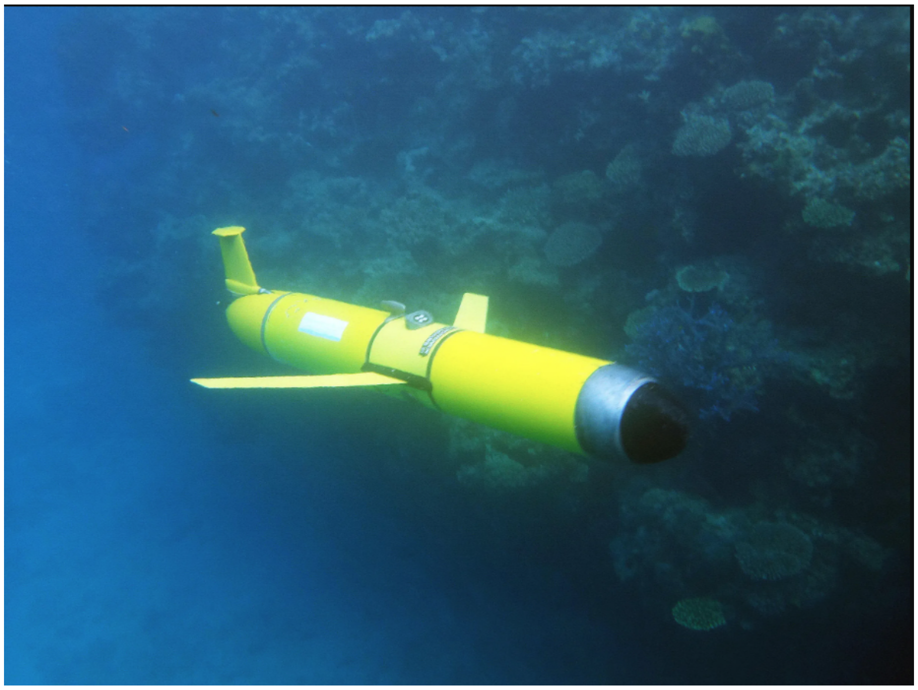


Figure 3 - Unmanned Underwater Vehicle

2.1.4 Unmanned Surface Vehicles

Unmanned Surface Vehicles - USVs, also known as drone ships, are boats that operate at the sea surface without a crew. USVs have various levels of autonomy and can be driven remotely by a human on land or work with autonomous COLREGs compliant navigation. The attention of USVs is on military purposes, oceanography, seaweed farming, and the last years at logistics.

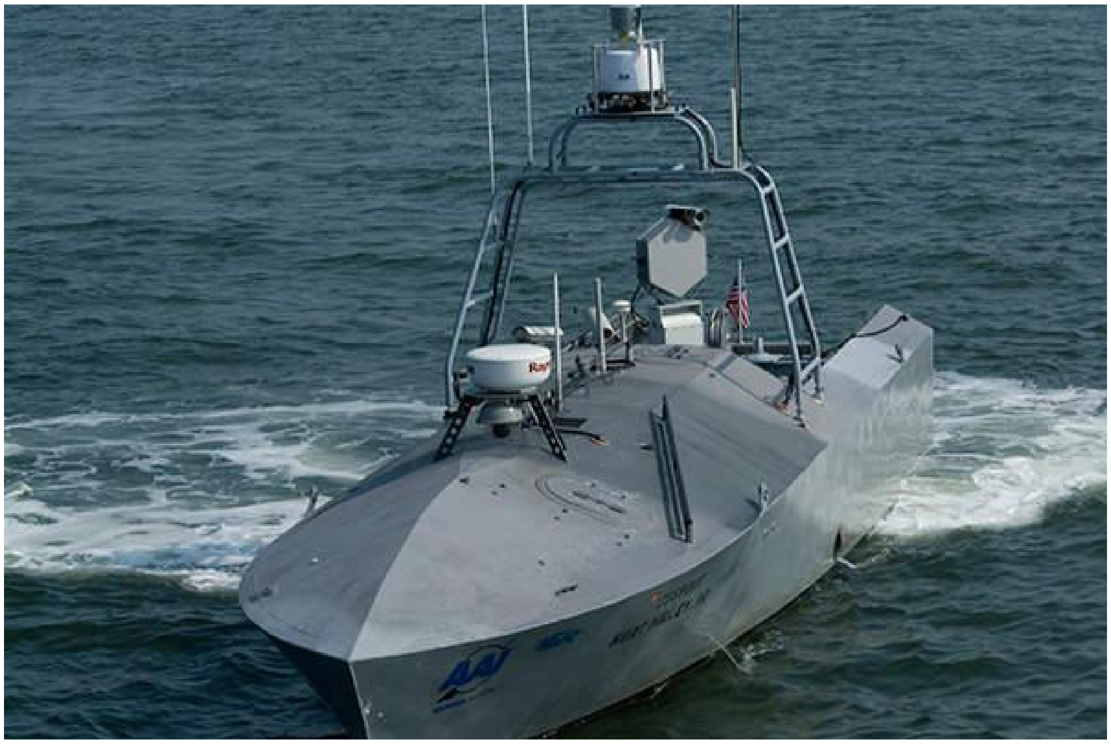


Figure 4 - Unmanned Surface Vehicle

2.2 Robotic Operation System – ROS

The rapid growth of all the unmanned vehicles in plenty of fields has created the need for software solutions that can support the various applications independently from the hardware and the characteristics of the machine. ROS is the dominant solution for this problem. The Robot Operating System - ROS is an open-source platform for building robotic software applications. Also, it has a global open-source community of engineers, developers, and hobbyists who contribute to making robots better and more available to our lives. ROS covers numerous industries from agriculture to medical devices to vacuum cleaners but is escalating to include all kinds of automation and software-defined dynamic use-cases.

2.2.1. Definition of ROS

Robot Operating System (ROS) is a software framework for operating robots. It is an open-source, meta-operating system that assumes an underlying operating system runs alongside. ROS isn’t an operating system. It provides many of the expected services of an operating system, including hardware abstraction, message-passing between processes, and package management, but not the core functionalities of an operating system is supposed to provide. It also provides tools and libraries that help you build, write, and run your project’s code across multiple computers. It is fundamental to understand that ROS isn’t actually an operating system. It is a set of libraries and tools that aims to simplify robotic’s software development. Due to its open-source nature, the best operating system to work with is Linux OS.

The main advantages of ROS are:

1. Distributed communication: ROS provides a reliable procedure for communication between processes regardless of whether they belong to the same computer. This feature is essential for robotics' software because plenty of algorithms span processes on multiple machines.

2. Open Source: The ROS framework is an open-source project, inheriting all the benefits of this feature. Developers worldwide, who use the framework, can contribute their code and reuse the code of others. Reusability of code, ROS's standard, and many other popular packages contain the majority of essential algorithms often used. The result is that developers are not concerned with the same problems and focus on the new parts of their projects.

3. Testing: Software that runs on robotics is inefficient for testing. Sometimes the developers don't have access to the whole equipment. Also, deploying this software to a robot every time a change exists is time-consuming. This problem at the production time is a huge productivity disadvantage. ROS provides a set of simulators and a simple way of recording logs. That improves the software production process and gives us efficient testing times without every time deploying the code on the machines. An overview of ROS main components is presented in Figure 5 below.

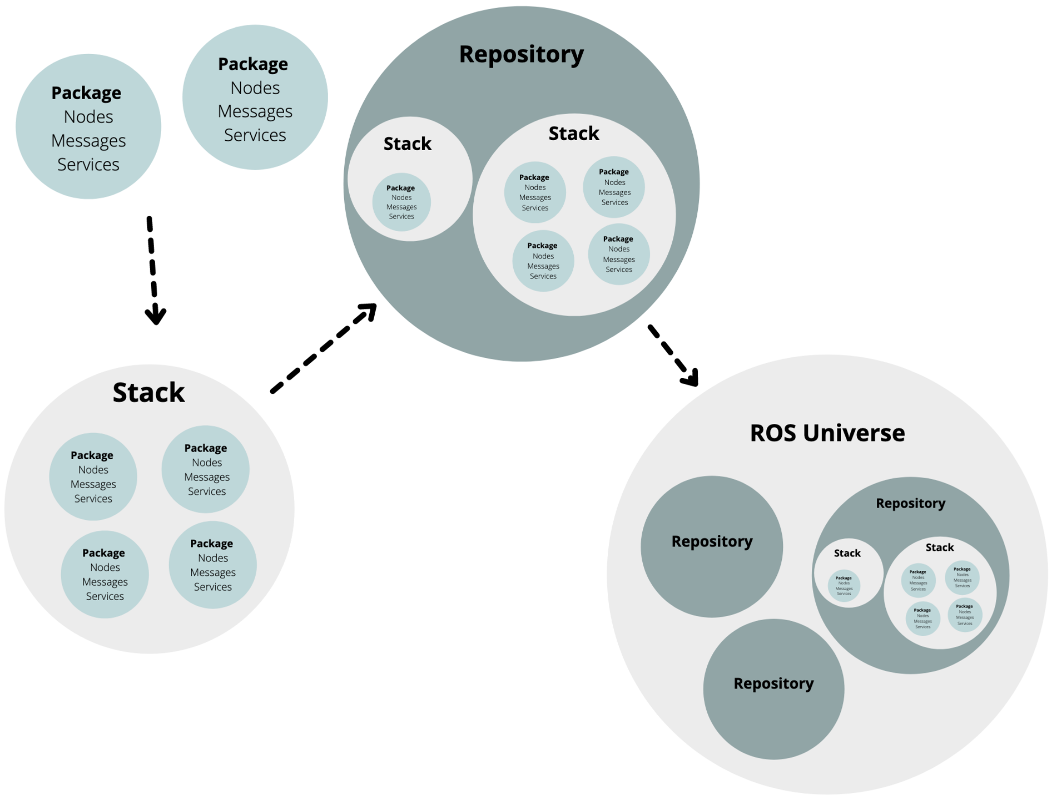


Figure 5 - ROS Universe Structure

2.2.2. ROS Packages

ROS software is separated into packages. A package is a folder that contains all the code and supporting files that serve a business logic or purpose. The directory has Files of ROS parts such as ROS nodes, ROS-independent library, a dataset, and configuration files. The result is that the package is the smallest unit that can build and release. This is an easy way to organize ROS functionalities and maintain the code. Packages are easily reusable and should have only necessary of the functionality that they serve. We can create packages manually or with ROS tools like catkin_create_pkg.

2.2.3 ROS Stack

A collection of packages make a stack. They are one layer above packages and aim to simplify the process of code contribution. Stack is the original build and release of the Ros Software. They have versions and can declare dependencies on other stacks. Dependencies also have a version number, which adds higher stability in development. ROS distributing software feature based a lot on this mechanism. This information is stored and managed in a .xml file called manifest. Unlike traditional software libraries, stacks are able to add functionality through topics and services, while the robot’s program is running.

2.2.4 ROS Nodes

One layer below of packages, we find nodes. Nodes are processes that perform computations and specific functionalities of a robot. For example, the camera's feed or the robot's movement. Nodes are organized in a graph and communicate with each other with a mechanism called topics.

The nodes work independently, so if some stop working, the others will not be affected. With that in mind, we can debug, add or remove a node without damaging the whole system. As a result, the development has less code complexity.

The running nodes have names as unique identifiers and a specific manually defined type. These make referring much easier.

2.2.5 ROS Catkin

Catkin is the ROS build system used to build all packages in ROS. It is the set of tools to create executables, libraries, interfaces, and scripts so the rest parts can use them.

2.2.6 ROS Topics

The topic is named bus by which nodes can exchange data. Most topics are the means to transmit ROS messages between nodes. Each topic has two nodes, a publisher and a subscriber node. If someone wants to access and consume the data, a node can subscribe to the topic's subscriber node.

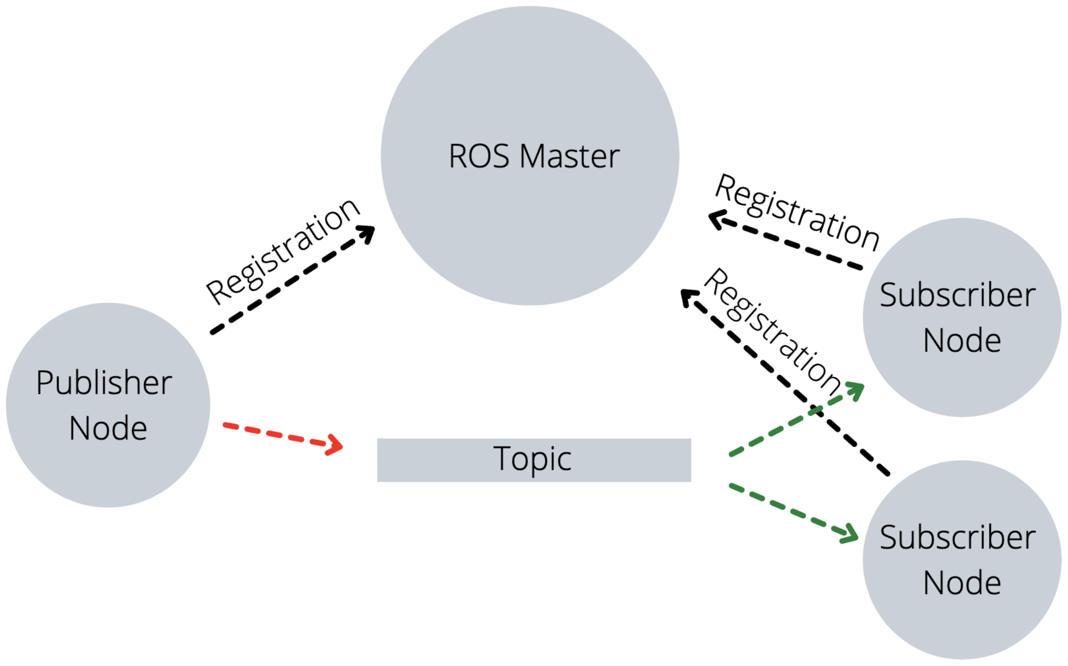


Figure 6 - Topic Communication

Respectively if someone wants to generate and provide data on a topic, publish on it. The publisher and subscriber nodes are mostly unaware of which node will publish or consume the messages. Each topic is able to have multiple subscribers and publishers as shown in Figure 6 insomuch they have different names.

2.2.7 Master Node

Ros contains many files, nodes, and topics. The Master node is responsible for coordinating all these complex operations and the synchronous running of all nodes.

Before any ROS code execution, we run the ROS Master node. With the command “roscore ” the Master will start. This node must be functioning the entire time in the background to provide ROS functionality.

2.2.8 ROS Messages

The nodes’ communication in ROS is attained with Topics. Through them, the nodes can exchange messages. Publisher nodes create the message that will be published to a topic. One or more subscriber nodes can consume this published message.

The messages have some standard types, but we could create some customs types too. To define each message type, standard, or custom, we need to have a msg file for them.

Table 1 - Standard ROS message types

Primitive Type	Serialization	C++	Python2	Python3
bool	unsigned 8-bit int	uint8_t	bool	
int8	signed 8-bit int	int8_t	int	
uint8	unsigned 8-bit int	uint8_t	int	
int16	signed 16-bit int	int16_t	int	
uint16	unsigned 16-bit int	uint16_t	int	
int32	signed 32-bit int	int32_t	int	
uint32	unsigned 32-bit int	uint32_t	int	
int64	signed 64-bit int	int64_t	long	int
uint64	unsigned 64-bit int	uint64_t	long	int
float32	32-bit IEEE float	float	float	
float64	64-bit IEEE float	double	float	
string	ascii string	std::string	str	bytes
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time	
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration	

2.2.9 ROS services

As mentioned above, ROS's communication is based on topics. This type of data sharing is unsuitable if we need synchronous transactions between nodes. ROS services solve this problem.

Services work with the request-response model, so it is needed a pair of defined messages for every service. With the same logic as the messages, the services are be described in a srv file, and we can access them via a string name.

2.2.10 Launch files

The roslaunch tool makes it easier to execute multiple nodes at once and set up parameters for the execution. This tool uses one or more launch files. These are XML configuration files with a .launch extension. The use of launch files is to define the parameters and the nodes we want to spawn. Also, launch files include options to respawn automatic processes that have already died. The location of launch files should be anywhere in the package directory because they are associated with a specific package. The syntax of the roslaunch command for executing a launch file is roslaunch package_name launch_file_name.

2.2.11 World files

As we mentioned above, for the purpose of testing, ROS has a set of Simulators. The World files are used to define a virtual world close to the real world that the robot lives and interacts with. The files detail the whole scene, objects, and various models. Some of the most crucial object characteristics are the size, the location, and the obstacle's name. World files are XML configuration files and have a .world extension. Commonly they are defined in the launch files, so the simulator will be informed what scene must begin with.

2.2.12 Sensors

Robots as unmanned vehicles need to have sensors to understand their surroundings and interact with the world. For example, cameras or lidar sensors would be necessary for the robot’s navigation. In this thesis, we use a Kinect camera which is a camera combined with 3D sensors, to understand the depth.

2.2.13 Turtlebot

TurtleBot is a popular small, low-conscious personal robot kit built for education and early-stage development. It is a repurposed robot vacuum cleaner to keep the product’s budget low. Turtlebot is the most affordable advanced ROS robot on the market. With TurtleBot, you’ll be able to build a robot that can drive around a room, see in 3D, and have enough power to make various applications. Turtlebot was designed in collaboration with the original makers of ROS, so it offers an affordable solution to get started with ROS.

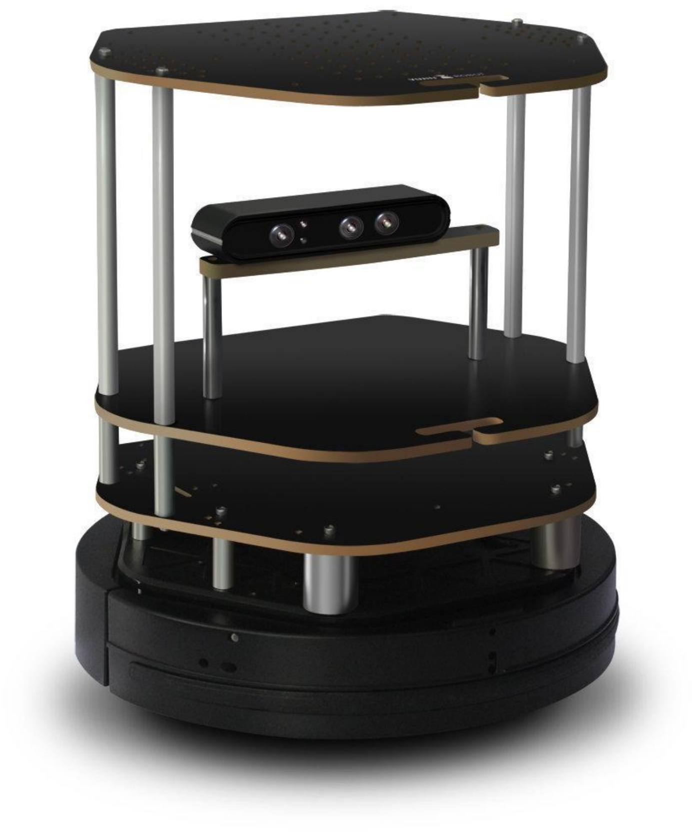


Figure 7 - Turtlebot 2

Turtlebot needs an external computer device to connect with all the sensors and operate. The most common solution that also this thesis has is a Raspberry Pi which can be mounted on the robot and run Linux operating system.

2.2.14 Gazebo Simulator

Robot simulation is a key tool in every robotic development project. A well-designed simulator makes it possible to design robots, test algorithms rapidly, perform regression testing, and train AI systems using realistic scenarios. Gazebo is an open-source simulator that can accurately simulate populations of robots in multiple complex indoor and outdoor environments. In this Thesis Gazebo was the testing tool for the robot's interaction with its environment and, more specifically, for its movement and video source. This simulator is the most popular in the robotic community in order to achieve similar results with the real world. To operate ROS with Gazebo and create the simulation, we need a set ROS packages named gazebo_ros_pkgs. These packages have default files and we can launch them with the command roslaunch turtlebot_gazebo turtlebot_world.launch.

3. AR TECHNOLOGIES AND TOOLS

3.1. Reality Technologies

3.1.1 Virtual Reality

Virtual Reality is a computer-generated environment that covers the entire field of the user’s vision and makes him feel like interacting with the real world. This environment has 3d scenes and objects to make it perceive realistic. The device we use for VR is called Virtual Reality helmet or headset. Virtual reality commonly incorporates auditory and video feedback, but other types of sensory may also be allowed, such as force feedback through haptic technology. Virtual Reality applies to many fields such as entertainment, education, art, medicine, etc.

3.1.2 Augmented Reality

Augmented Reality is a computer-generated environment like VR with the main difference, that augmented reality alters one's ongoing perception of a real-world environment, whereas virtual reality completely replaces the user's real-world environment with a simulated one. The user blends the real world with virtual scenes and 3d objects that are added to it. The physical environment has a leading role in the whole experience because the virtual objects take an accurate position in the space and coexist with the real ones. AR is used in education, communications, medicine, and entertainment. Also, AR is very useful in difficult professions where they need interaction with a computer while space and profession may not have this potential due to the use of their hands or not being able to lose their attention from the environment.

3.1.3 Mixed Reality

Mixed reality is related to augmented reality and it is the merging of the real world and virtual worlds to produce new environments and visualizations where physical and digital objects co-exist and interact in real-time. A noticeable difference between augmented and virtual reality is that the device has to be head-mounted so that its Mixed Reality necessary sensors work properly.

3.2 HoloLens

Microsoft Hololens is a pair of augmented reality glasses or a helmet that has the world's first fully independent holographic computer. HoloLens blends cutting-edge optics and sensors to deliver 3D holograms pinned to the real world around you. Microsoft has introduced two gens of Hololens until now, the Hololens 1 and subsequently the Hololens 2.

3.2.1 HoloLens 2

The Hololens 2 was introduced in 2019 by Microsoft and they were the second generation of the product’s line. The device has holographic lenses that the user can see through, and two IR cameras for eye tracking so can render based on the eye. Also, it is equipped with four light cameras for head tracking and one time-of-flight depth sensor for the environment. The device has a depth sensor, AI, semantic understanding, and eye-tracking sensors that allow the user to manipulate holograms more naturally, which also means less learning curve to use the device.

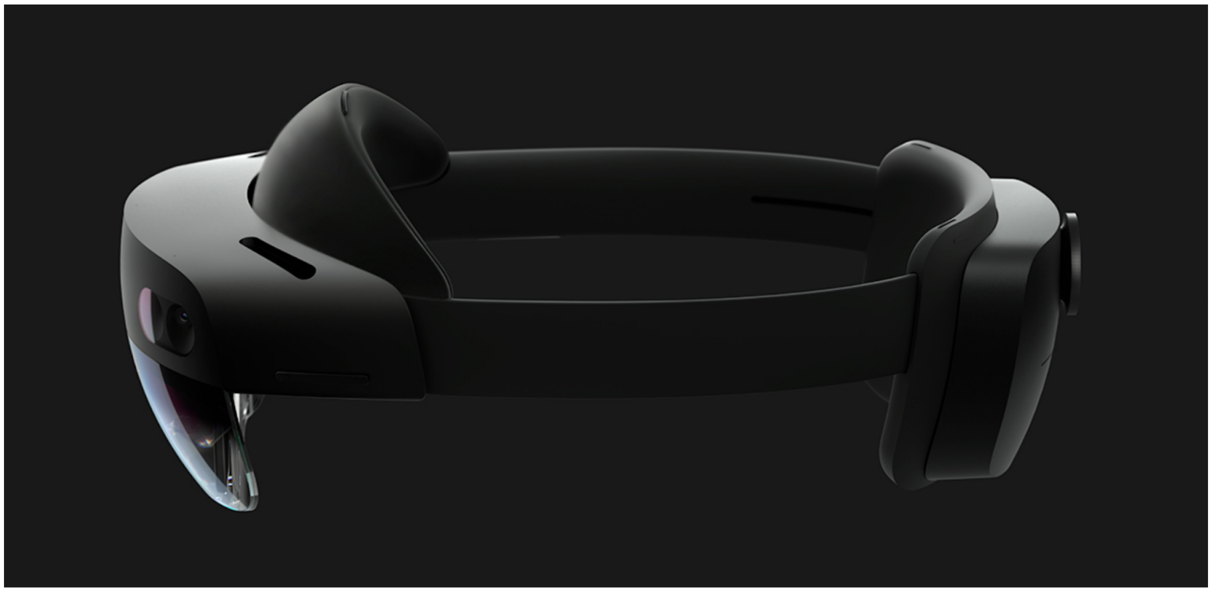


Figure 8 - Hololens 2 Side View

The holograms were created by reflecting images from a screen in the headset into specially made lenses for red, blue, and green light waves. Those light waves were then beamed into the back of your eyes, where your brain would create the final image.

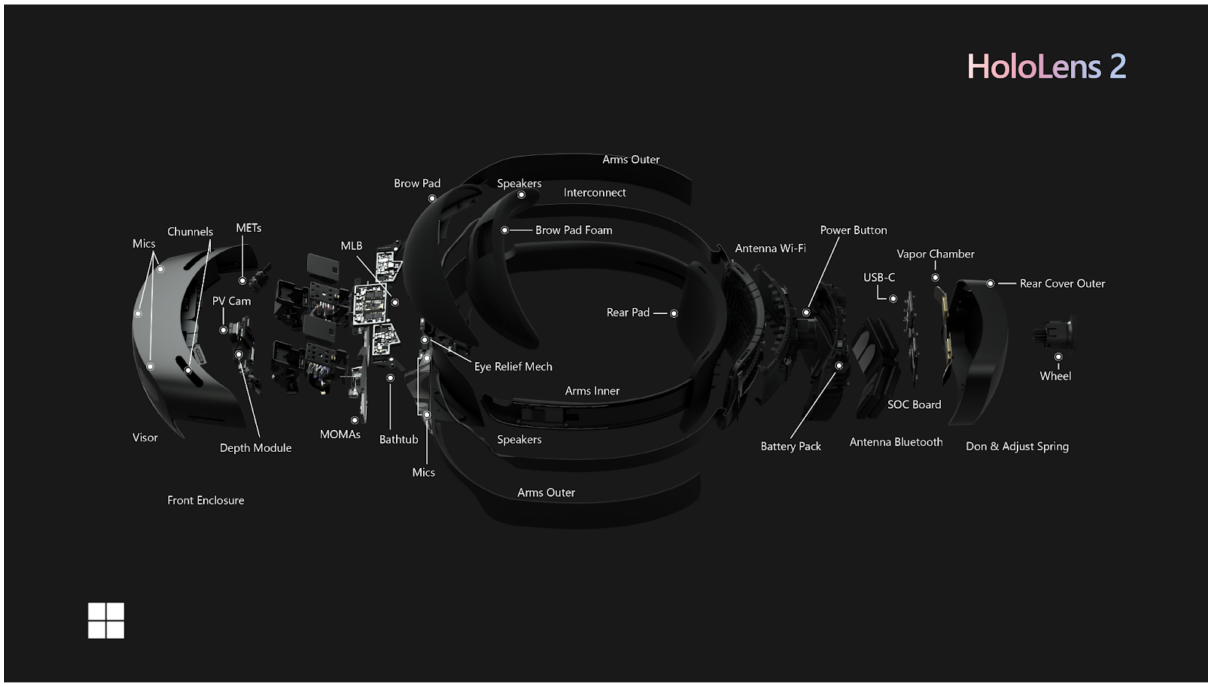


Figure 9 - Hololens 2 Parts

The device has a mirror known as a MEMS (microelectromechanical systems) that moves fast enough to create the illusion of a screen in space. The MEMS creates 120 of these screens each second, filtered to the eye through lenses in the headset. The result is smooth movements, more-believable animations, and quick response if you move your head.

Figure 10 - Hololens 2 Front Glass

The glasses except the CPU/GPU have a special process unit that Microsoft named Hololens Processing Unit or HPU. The HPU conducts the processing that integrates the real world and data for augmented reality. All of the integration of environmental data and user input is handled by the HPU. The HPU receives information from the inertial measurement unit (IMU) which includes an accelerometer, gyroscope, and magnetometer, and combines this with head tracking cameras that follow the user and their environment to augment the user's visual perspective with holographic 3D projections on its transparent lenses. The processing of gestures and voice data is also handled by the HPU. This processing unit is useful to not consume CPU's and GPU's resources, so apps can use them. The Hololens 2 have a Second-generation custom-built holographic processing unit.

The glasses have a version of Windows 10 that is designed for HoloLens 2 with commercial-ready management capabilities and software apps, especially for Hololens like Microsoft edge on holographic version and 3D viewer.

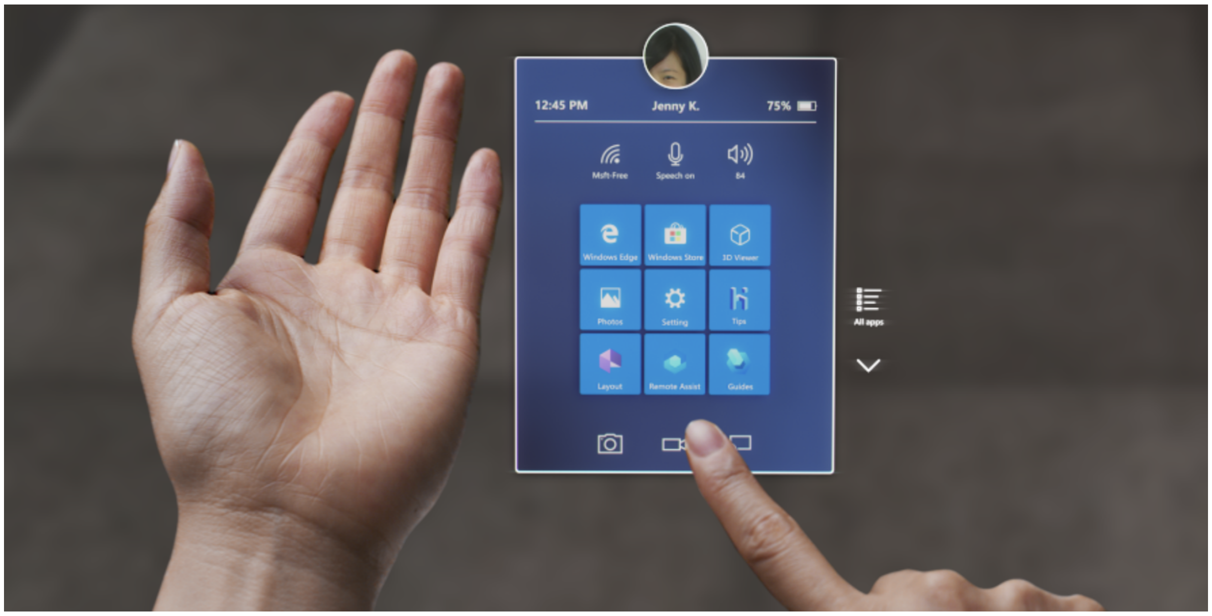


Figure 11 - Start Menu Hologram

3.3 Holograms

With the Hololens, you can create holograms, which are objects made of light and sound that take shape in the environment around you like were real objects. Holograms are made to respond to your input actions that we discuss later. They can even interact with the real surfaces of space you are. Holograms are digital objects that are part of the augmented reality around you.

Figure 12 - Holograms Medicine Case

The Hololens render the hologram in the holographic frame right in front of the user's eyes. The holographic display is not blocking the light from the real world instead adds light to render the holograms. Since Hololens use an additive display that adds light, the black color will be rendered transparent.

Holograms are not only visual objects, but they can produce sounds to have a more realistic experience. The sound comes from the speakers that are placed above the user’s ears. Same as the holographic display, the speakers add sound without blocking the sound from the surroundings.

A hologram can have a fixed location in the room, you can place it exactly at the point in the world you want. As you move inside the space the hologram appears static in relation to the rest virtual environment so it can have a real position in the physical world. You can use a spatial anchor to pin the object in someplace and if you leave the room the device can remember the position of the object to add when you come back.

The other option is that the holograms can follow you. They position themselves based on the user’s location. You can choose to bring a hologram with you, and then place it on the wall once you get to another room.

Holograms are not only objects that we can see but they have a very important active part that users can interact with them and developers can create useful applications. A user can gaze at a hologram, gesture with his hand, follow him, or give it a voice command, and the hologram reply.

Figure 13 - Holograms Browsing Case

Holograms enable personal interactions that aren't possible elsewhere. Because the HoloLens knows where it is in the world, a holographic character can look at you directly in the eyes and start a conversation with you.

Figure 14 - Holograms 3D Model Simulation Case

A hologram can also interact with your surroundings. For example, you can place a holographic bouncing ball above a table. Then, with an air tap, watch the ball bounce, and make a sound as it hits the table.

Holograms can also be occluded by real-world objects. For example, a holographic character might walk through a door and behind a wall, out of your sight.

Tips for integrating holograms and the real world

- Aligning to gravitational rules makes holograms easier to relate to and more believable. For example: Place a holographic dog on the ground & a vase on the table rather than have them floating in space.
- Many designers have found that they can integrate more believable holograms by creating a "negative shadow" on the surface that the hologram is sitting on. They do this by creating a soft glow on the ground around the hologram and then subtracting the "shadow" from the glow. The soft glow integrates with the light from the real world. The shadow is used to ground the hologram in the environment.

3.4 Hololens Actions

The user experience differs a lot in MR applications from the devices that we typically use. The user's interaction is in a 3D environment and the user has to learn new input methods that are only in this field. Here I represent some main ideas that Hololens use for input, such as gaze, gestures, and voice commands.

The user interacts with the device with some actions:

- Gaze tracking
- Gesture input
- Touch
- Hand ray
- Air Tap
- Bloom
- Start Gesture
- Voice support

3.4.1. Gaze tracking

Gaze is an action in which the user can interact with the world based on where is looking. Gaze works in two different ways, head gaze and eye gaze. The second is the head gaze, the device understands where the helmet is oriented in the space so can get input, where the user is looking. The eye gaze is based on eye-tracking knowing where the eye is looking. The applications can intersect this ray with virtual or real-world objects, and draw a cursor at that location to let the user know what they're targeting. The eye gaze was introduced with the Hololens 2 that support eye-tracking.In the first generation, we had only head gaze.

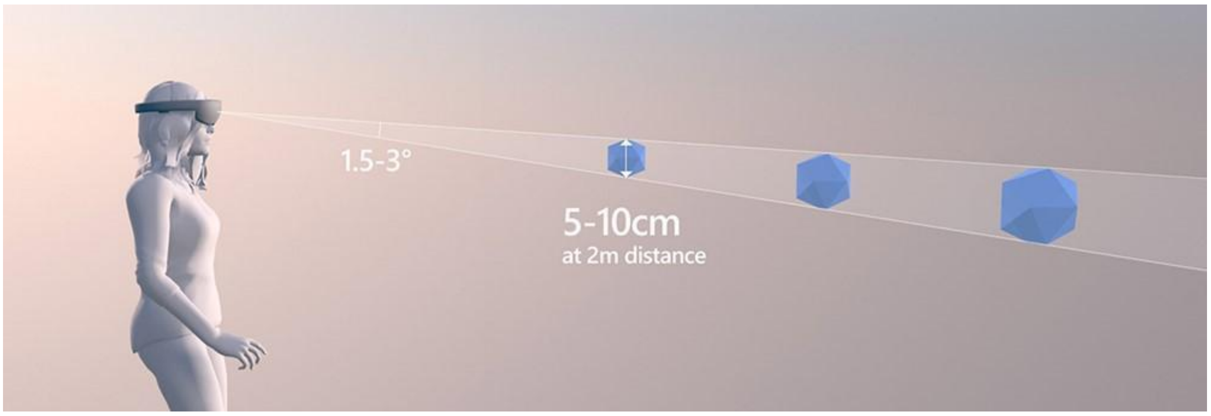


Figure 15 - Hololens Head Gaze

Some ways the gaze can use for:

- Your app can intersect gaze with the holograms in your scene to determine where the user's attention is (more precise with eye-gaze).

- Your app can channel gestures and controller presses based on the user's gaze, which lets the user seamlessly select, activate, grab, scroll, or otherwise interact with their holograms.
- Your app can let the user place holograms on real-world surfaces by intersecting their gaze ray with the spatial mapping mesh.
- Your app can know when the user isn't looking in the direction of an important object, which can lead your app to give visual and audio cues to turn towards that object.

3.4.2 Gesture input

The user navigates in the HoloLens UI with some hand movements. Some gestures are available only in Hololens' one of two editions and others are on both. The glasses to be able to understand the user's gestures have a hand-tracking frame that the user needs to keep his hands in it. As the user moves around, the frame moves with him. The basic gestures are Touch, Hand Ray, Gaze, Air Tap, and Bloom, or for the Hololens 2 Start Gesture.

3.4.2.1 Touch

In Hololens 2 the easiest way to interact with a hologram is by a simple touch. You can touch and grab holograms by targeting objects with the help of a floating pointer that appears near the tip of your index. The touch needs your index to reach the physical position of an element.

3.4.2.2 Hand ray

This gesture needs to hold your hand open in front of you and face away. You can target objects by moving your hand with the help of a laser pointer. The movement is that you point the object with your open palm. After that, you can do several actions with it to manipulate it.

3.4.2.3 Air Tap

To select an app or a hologram is not necessary to touch it, you can air tap it. To do this, gaze at the hologram, hold your hand straight out in front of you in a loose fist, point your index finger straight up toward the ceiling, tap your finger down, and then quickly raise it back up again. Also, you can air tap and hold.

3.4.2.4 Bloom

In Hololens 1 to open the windows start menu, you gather all your fingers together and then open your hand. This gesture is called Bloom.

3.4.2.5 Start Gesture

In Hololens 2 the bloom gesture is replaced by the Start gesture. To perform this action hold out one of your hands with the palm facing up, and look at your wrist. You should see a holographic Microsoft Windows logo. Then with the other hand touch the icon on your wrist. The Start menu will pop up in front of you. You can also open the menu with one hand. After appears the logo, just touch your index finger to your thumb in a pinching motion.

3.4.3 Voice support

The Hololens 2 support voice recognition for many quick tasks. Some voice commands are built-in like taking a photo or opening apps, and others are available through Cortana which is Microsoft’s virtual assistant. Some nice built-in commands to interact with 3D objects are: “Face me” to turn the element to face you, “Move this” to follow your gaze, and “Follow me” to follow your place in the room.

4. ROBOT’S REMOTE CONTROL AND VIDEO STREAMING APPLICATION

The problem was the lack of an open-source application in the Unity environment where an operator can control efficiently ROS devices from the Hololens 2. The goal of this application was to fill this gap.

4.1 Application’s Basic Tools

First they will be analyzed some of the tools that were used for the application.

4.1.1 Unity Engine

Unity is a powerful cross-platform game engine for Software Developers. Unity can provide a lot of tools that an application with rich graphics needs, like physics, 3D object rendering, and collision detection. These basic problems are solved by Unity such as how light should bounce off of different surfaces or creating a new physics engine from scratch–calculating every last movement of each material.

A feature that makes Unity even more powerful, is the Asset Store. It is a place where all developers can upload and share their creations. This community was important for the Unity platform. During development arise needs for project’s components that are accessible as assets from the store. For example, if you want to add tilt controls to your game without going through the laborious process of fine-tuning the sensitivity, you can find them as assets in the store. The result is that developers are free to focus on the unique features of their applications.

For MR applications on Hololens, we have to use Microsoft’s Mixed Reality Toolkit library. This is designed only for Unity, so this platform is the only option for this type of application. Unity is a platform that is updating frequently, these updates are useful to solve known issues but many times create new bugs and incompatibilities so developers are forced to make changes to the project in order to make it buildable again. Therefore, the correct configuration is very important in order to make the application work.

For this application MRTK 2.7 was used, which is the current latest version, so the appropriate Unity version had to be used as well. We have Unity 2019.4.28f1 which is a stable version of Unity for the MRTK version we use. In MRTKS’s official GitHub repository can someone find which unity version is compatible with MRTK.

4.1.2 Mixed Reality Toolkit - MRTK

MRTK-Unity is an open-source Microsoft-driven project that provides components, common building blocks for spatial interactions, and UI features that are used to accelerate the development of cross-platform Mixed Reality apps in Unity. MRTK supports many platforms such as OpenXR, Oculus, OpenVR, and mobile. MRTK is compatible with all Hololens Devices. For this application, we use the current latest MRTK version which is 2.7.

In the MRTK Github Repository, we can find features in areas such as eye and hand tracking, spatial awareness, controls, input system, boundary system, and much more. Also, there are a lot of examples and tools that make the Hololens development much easier.

Figure 16 - Mixed Reality Toolkit Objects

A principal and useful feature of MRTK is the user's capability to test the Application in the Unity Editor. A Hololens Emulator executes the scene with an easy play button on Unity. The emulator has virtual hands and eyes to simulate the Hololens experience. This is important because the build and deployment of the program from Unity to the glasses are time-consuming.

The emulator has a diagnostic tool on the bottom of the screen called Visual Profiler. This tool has some information about the application, such as memory usage and current FPS. It is important always to keep track of this information because It is important to achieve the desired framerate, as outlined by the platform being targeted (i.e., Windows Mixed Reality, Oculus, etc.). For example, on HoloLens, the target framerate is 60 FPS. Low framerate applications can result in deteriorated user experiences such as worsened hologram stabilization, world tracking, hand tracking, and more.

4.1.3 Kafka

Kafka is a distributed streaming platform. The Kafka platform implements queues that can handle events very fast and efficiently. The application uses the Confluent Platform, which implements at its core Apache Kafka which is the only enterprise event streaming platform at that time. The Confluent Platform offers many different connectors that provide more application-specific automation for handling events and has enough documentation that specifies them. To publish or consume messages in real-time to any other application, an application has first to subscribe to the right topic. Kafka platform guarantees some essential assumptions, the most crucial is that the messages will be consumed in the order they were published to the topic.

The application makes use of this assumption and visualizes incoming data from any source that are real-time. The Topics are uniquely named buses that are used for data communication between nodes. Every topic should have a unique name in order to avoid problems and confusion between the same-named topics. Nodes can publish or subscribe to a topic, but data production and consumption are decoupled meaning that there is an indirect connection between the nodes. Finally, each topic is related to a message type, meaning that it only accepts specific message types. In this application is set the JSON data type.

The built-in Kafka REST Proxy connector appeared to be the most efficient way to communicate with a web server that is running Kafka. This proxy allowed the application to consume messages from Kafka topics with code that the frameworks and the Unity Editor can support. The producers in python set automatically the correct configuration properties for Kafka in order for the application to achieve the production of the messages in JSON format. However, on the consumer's side, some configuration is necessary.

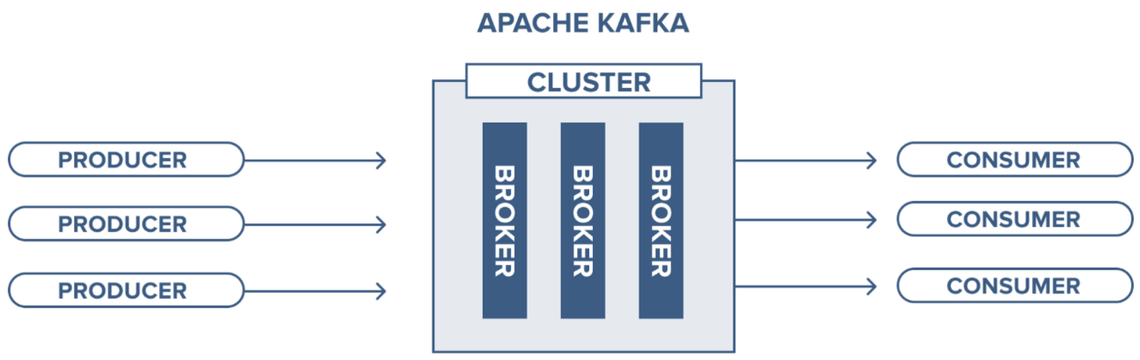


Figure 17 - Kafka Structure

4.1.4 Visual Studio

When you install Unity, by default it is also installing Microsoft's Visual Studio. This feature provides unity with a stable c# script Editor and the necessary compilers for the application's build. Unity supports also the VS code for the editor part as an alternative choice.

The Create Visual Studio Solution build setting allows you to generate a Visual Studio solution instead of a built app. This procedure enables you to change your build process. For example, you can, modify your application's manifest, add C++ code, modify embedded resources and launch your application with the debugger attached. By default, Unity stores the Visual Studio solution you generate in the same directory as your built project. Your generated solution includes three projects when you use the Mono scripting backend, and four when you use the IL2CPP scripting backend. When the code compilation is completed the program is ready for deployment on the glasses.

4.2 Application's Basic Components

In this section will be analyzed how the application works. We split the controller into three parts. The first part is the robot device with a raspberry pi computer for its software, the second part is the Kafka Server as middleware and the third part is the Hololens program that is deployed on the AR glasses.

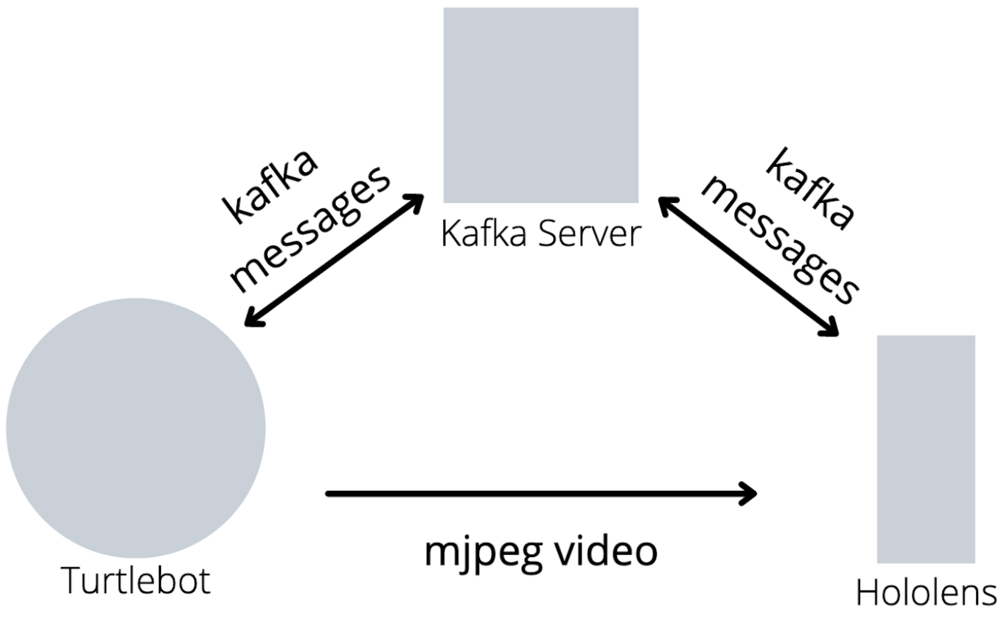


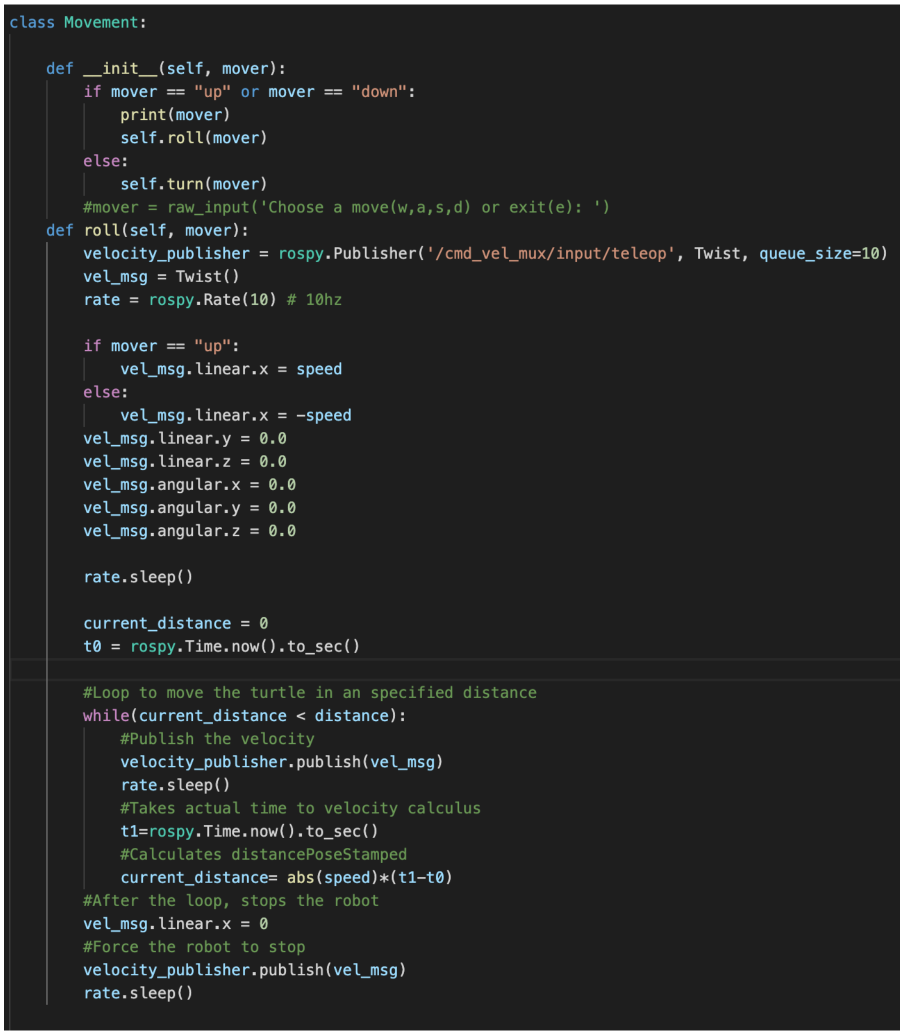
Figure 18 - Application Structure

4.2.1 Turtlebot

As described in Section 2.2.6 ROS operating system works with a channel system. Thus, all functions of the robot are subscribing to topics and exchanging messages there.

4.2.1.1 Controller Ros Package

This is a custom package for ROS that allows the robot's application to control the robot and exchange messages with the device through topics. We can see most of the functionality and the package core that is located in the controller's core python script.



```python
class Movement:

    def __init__(self, mover):
        if mover == "up" or mover == "down":
            print(mover)
            self.roll(mover)
        else:
            self.turn(mover)
        #mover = raw_input('Choose a move(w,a,s,d) or exit(e): ')
    def roll(self, mover):
        velocity_publisher = rospy.Publisher('/cmd_vel_mux/input/teleop', Twist, queue_size=10)
        vel_msg = Twist()
        rate = rospy.Rate(10) # 10hz

        if mover == "up":
            vel_msg.linear.x = speed
        else:
            vel_msg.linear.x = -speed
        vel_msg.linear.y = 0.0
        vel_msg.linear.z = 0.0
        vel_msg.angular.x = 0.0
        vel_msg.angular.y = 0.0
        vel_msg.angular.z = 0.0

        rate.sleep()

        current_distance = 0
        t0 = rospy.Time.now().to_sec()

        #Loop to move the turtle in an specified distance
        while(current_distance < distance):
            #Publish the velocity
            velocity_publisher.publish(vel_msg)
            rate.sleep()
            #Takes actual time to velocity calculus
            t1=rospy.Time.now().to_sec()
            #Calculates distancePoseStamped
            current_distance= abs(speed)*(t1-t0)
        #After the loop, stops the robot
        vel_msg.linear.x = 0
        #Force the robot to stop
        velocity_publisher.publish(vel_msg)
        rate.sleep()
```

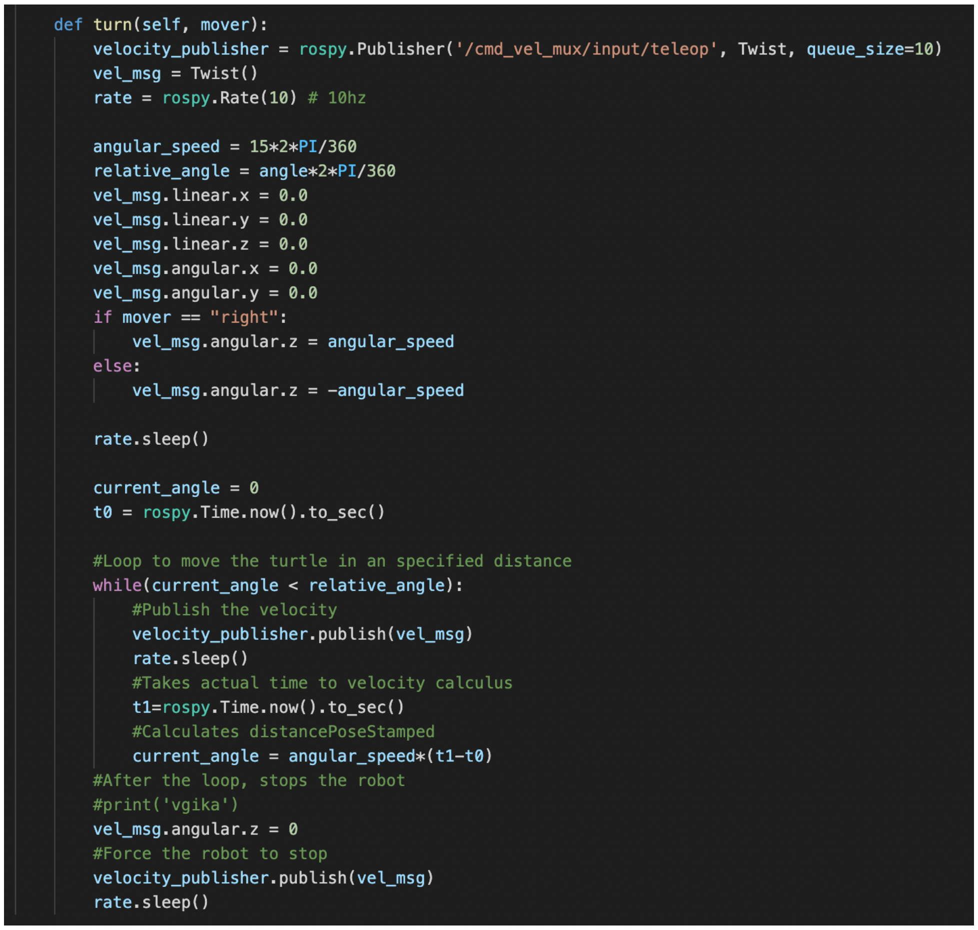
Figure 19 - Class Movement

We have one topic the /cmd_vel_mux/input/teleop, which is responsible for the turtlebot2's movement and the robot is subscribed to it. In this topic, depending on the command and the move we want the robot to make, we publish the corresponding messages for this topic. The two moves that the robot supports are roll and turn.

The messages that this topic waiting for are called Twist messages. This message is represented by two vector3. These two vectors are the linear and angular velocities in the free space. So the message Twist has a Twist.linear (x, y, z) part and a Twist.angular (x, y, z), each one has 3 axes which are mutually perpendicular to each other and their point of intersection is called the origin (x = 0, y = 0, z = 0). This can be a frame of reference i.e. you can define various points and directions w.r.t. them.

In the case of Turtlebot which is a ground Robot, the z i.e. in angular velocity (Twist.angular.z) will be the robot's turning speed. And the x i.e in linear velocity vector (Twist.linear.x) will be the robot's moving straight speed.

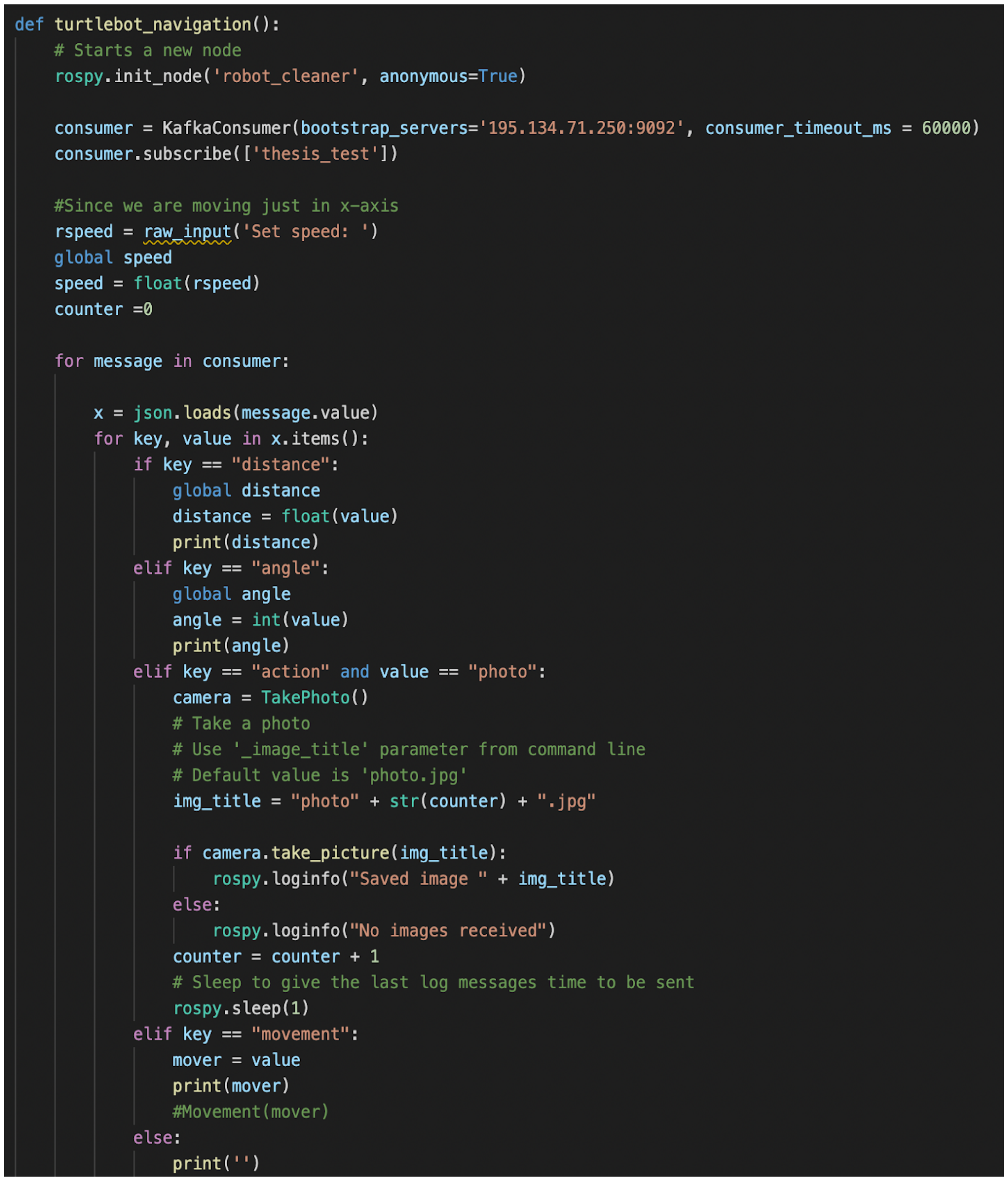
The class Movement [20] is responsible for the robot's moves. In method roll, we create a publisher where we publish periodically messages that move the robot at a given distance and speed back or front. This is accomplished by publishing in a loop, twist linear messages, with a value on the x-axis until the given step distance will be covered. The distance definition formula is "distance = absolute(speed) * time".



```
def turn(self, mover):
    velocity_publisher = rospy.Publisher('/cmd_vel_mux/input/teleop', Twist, queue_size=10)
    vel_msg = Twist()
    rate = rospy.Rate(10) # 10hz

    angular_speed = 15*2*PI/360
    relative_angle = angle*2*PI/360
    vel_msg.linear.x = 0.0
    vel_msg.linear.y = 0.0
    vel_msg.linear.z = 0.0
    vel_msg.angular.x = 0.0
    vel_msg.angular.y = 0.0
    if mover == "right":
        vel_msg.angular.z = angular_speed
    else:
        vel_msg.angular.z = -angular_speed

    rate.sleep()

    current_angle = 0
    t0 = rospy.Time.now().to_sec()

    #Loop to move the turtle in an specified distance
    while(current_angle < relative_angle):
        #Publish the velocity
        velocity_publisher.publish(vel_msg)
        rate.sleep()
        #Takes actual time to velocity calculus
        t1=rospy.Time.now().to_sec()
        #Calculates distancePoseStamped
        current_angle = angular_speed*(t1-t0)
    #After the loop, stops the robot
    #print('vgika')
    vel_msg.angular.z = 0
    #Force the robot to stop
    velocity_publisher.publish(vel_msg)
    rate.sleep()
```

Figure 20 - Method Turn

In method turn as shown in the figure 22, we create a publisher where we publish on the same topic as before periodically messages that turn the robot in a given turn angle measured in degrees. The angular velocity value is defined in the twist angular's z-axis. The loop breaks when the current angle reaches the relative one. The formula for the radian conversion that is needed is 1 rad = 1 degree * π /180 degrees.



```
def turtlebot_navigation():
    # Starts a new node
    rospy.init_node('robot_cleaner', anonymous=True)

    consumer = KafkaConsumer(bootstrap_servers='195.134.71.250:9092', consumer_timeout_ms = 60000)
    consumer.subscribe(['thesis_test'])

    #Since we are moving just in x-axis
    rspeed = raw_input('Set speed: ')
    global speed
    speed = float(rspeed)
    counter =0

    for message in consumer:

        x = json.loads(message.value)
        for key, value in x.items():
            if key == "distance":
                global distance
                distance = float(value)
                print(distance)
            elif key == "angle":
                global angle
                angle = int(value)
                print(angle)
            elif key == "action" and value == "photo":
                camera = TakePhoto()
                # Take a photo
                # Use '_image_title' parameter from command line
                # Default value is 'photo.jpg'
                img_title = "photo" + str(counter) + ".jpg"

                if camera.take_picture(img_title):
                    rospy.loginfo("Saved image " + img_title)
                else:
                    rospy.loginfo("No images received")
                counter = counter + 1
                # Sleep to give the last log messages time to be sent
                rospy.sleep(1)
            elif key == "movement":
                mover = value
                print(mover)
                #Movement(mover)
            else:
                print('')
```

Figure 21 - Method turtle_navigation

The turtle_navigation method as shown in the figure 21 above is the core of the controller. A Kafka consumer is created and is subscribed to the Kafka server, ready to listen to all the commands in form of Kafka messages that will be published on the server. In the loop, the program checks the type of the message and depending on the command, makes an action. Here also is implemented the input of the parameters of speed, angle, and distance.

The command “rosrun position myMovingScriptGzbKfk.py” is necessary to enable the ROS controller.

4.2.1.2 Kinect Camera Stream

The turtlebot has a Kinect camera. To successfully take the stream from the Kinect camera and publish it to a ROS topic, we use the freenect_camera ROS package, which is a libfreenect-based ROS driver for the Microsoft Kinect. This package is publishing messages on a topic for a consumer video server that we will mention later.

We execute a launch file with the command

“roslaunch freenect_launch freenect.launch”.

4.2.1.3 Turtlebot activation

The turtlebot_bringup package provides all the software that turtlebot needs to start up. Specifically, we run the minimal.launch file for starting the TurtleBot base functionality.

We execute a launch file with the command

“roslaunch turtlebot_bringupminimal.launch”.

4.2.1.4 ROS web video server

web_video_server is a ROS package that helps us to transfer the video feed to hololens. This package provides a video stream of a ROS image transport topic that can be accessed via HTTP.

The web_video_server frees a local port and waits for incoming HTTP requests. Following that a video stream of a ROS image topic is requested via HTTP, it subscribes to the corresponding topic and creates an instance of the video encoder. The encoded raw video packets are served to the client. Parameters can be specified by adding additional them to the query string.

The web_video_server tries to minimize internal coding latency by avoiding a B-frame encoding scheme and by forcing the codec to keep its internal network buffer as small as possible. On the browser side, however, HTML5 does not allow any control of the video playback buffer. Depending on the Internet connection, the browser might cache a few seconds of video data first before starting the video playback. Best performance concerning latency and stability has been achieved with recent versions of the Chrome browser.

From this package we use the mjpeg stream

```
/stream?topic=/camera/rgb/image_rect_color
```

This package starts with the command

“rosrun web_video_server web_video_server”

4.2.2 Kafka

We have set up the Kafka Server as a Broker. With the Kafka platform, the ROS application on turtlebot is able to communicate with the Hololens in order to receive movement commands and input parameters.

As described in Section 4.1.3 in The Unity application as we will see above uses Kafka Rest to publish messages on a topic that the ROS Application is subscribed on. The Kafka choice provides us with a reliable live-time data streaming solution without the two devices needing to be in the same network due to the Kafka server being accessible from everywhere. Also, Kafka guarantees us that the messages will be consumed in the order that will be published. This is important because the commands are translated to the physical robot’s moves.

4.2.3 Hololens

The parts that are related to AR and and more specifically to the Hololens Device.

4.2.3.1 UI

The application has two scenes. The first scene is the variables’ input panel as shown in Figure 23 that accepts three parameters.

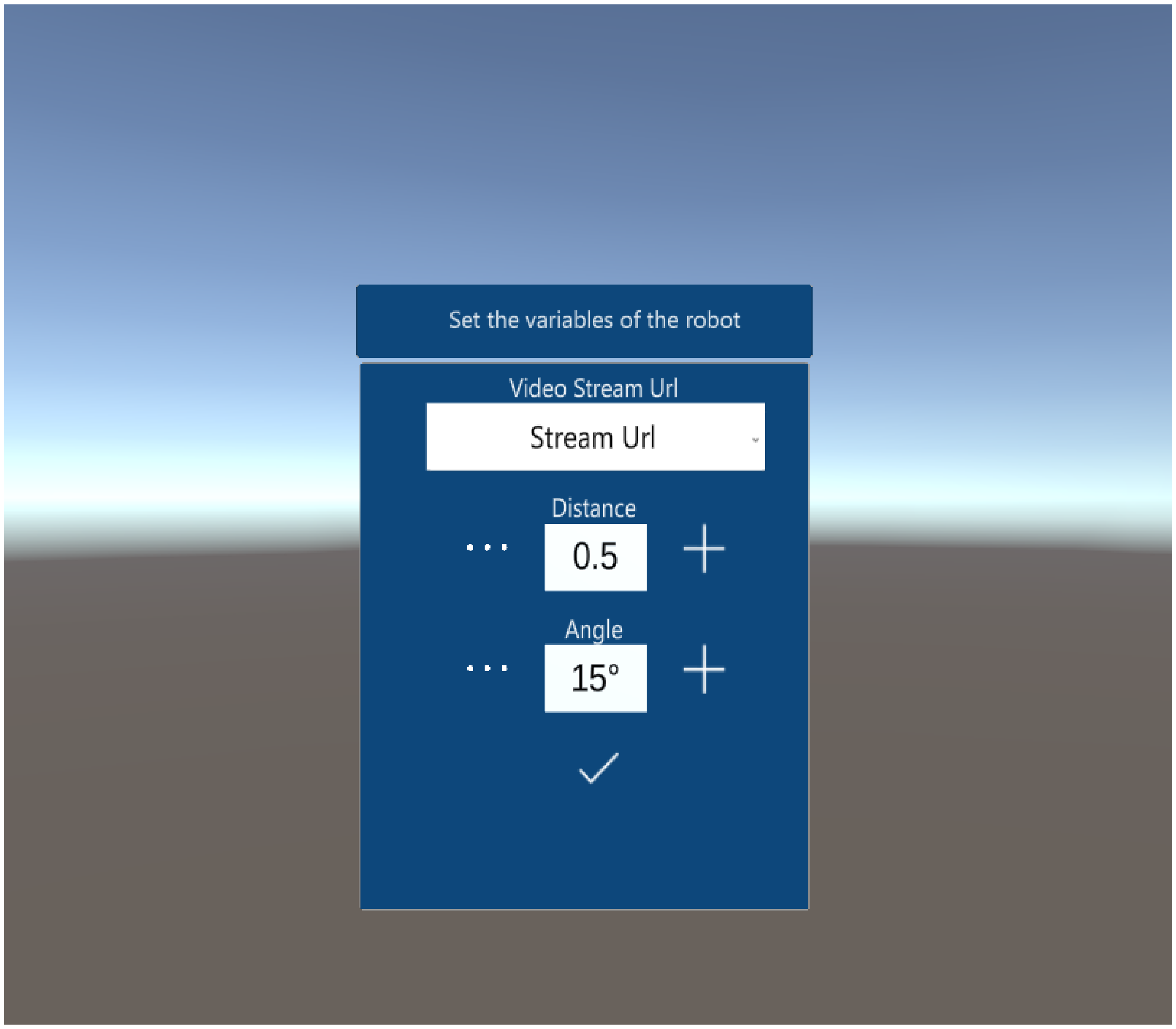


Figure 22 - Variables Panel

These are the distance of the step that the robot will move in each command, the angle that the robot will turn in each command, and a dropdown with possible the URLs from which it will consume the video stream.

The second scene is the main scene [24] where the user sees the feed from the turtlebot's camera and the control buttons for the turtlebot navigation. More specifically the main scene has a virtual screen that floats in the user's physical environment and a cross of arrow buttons that could give the navigation commands. Also below at the left exists a button for a quick photo capture of the user's surroundings.

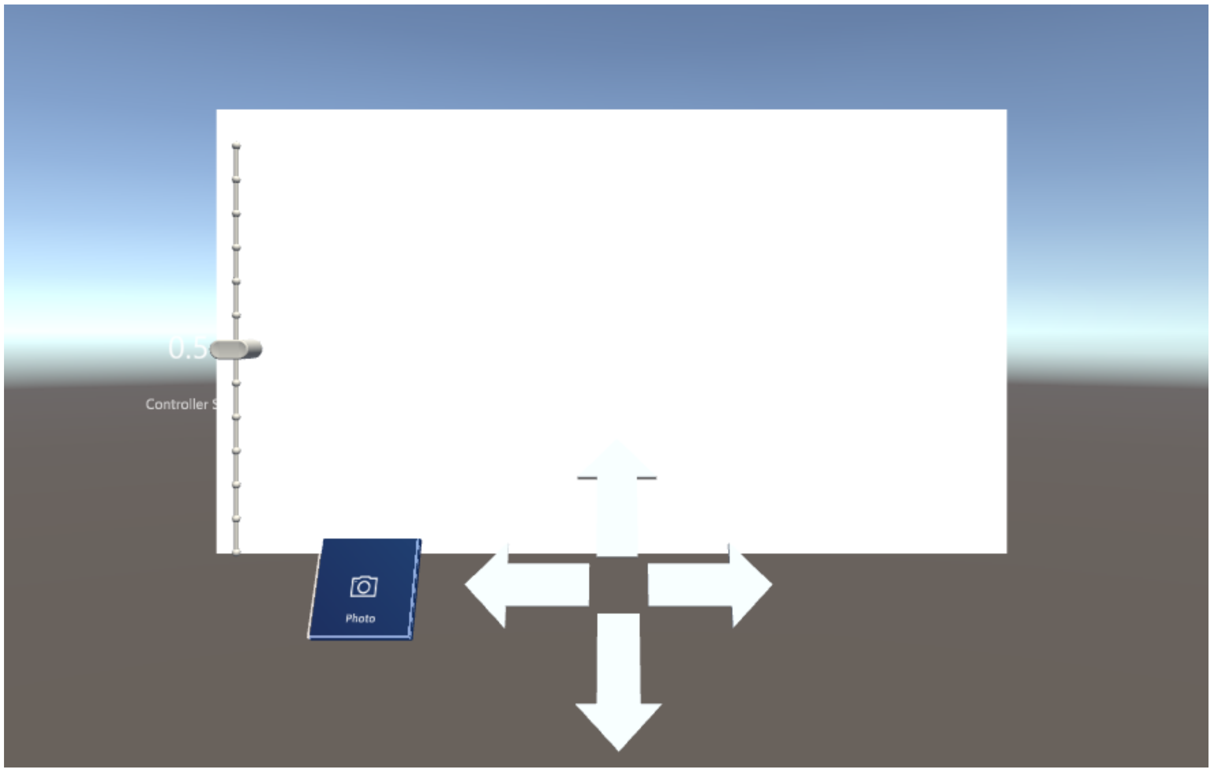
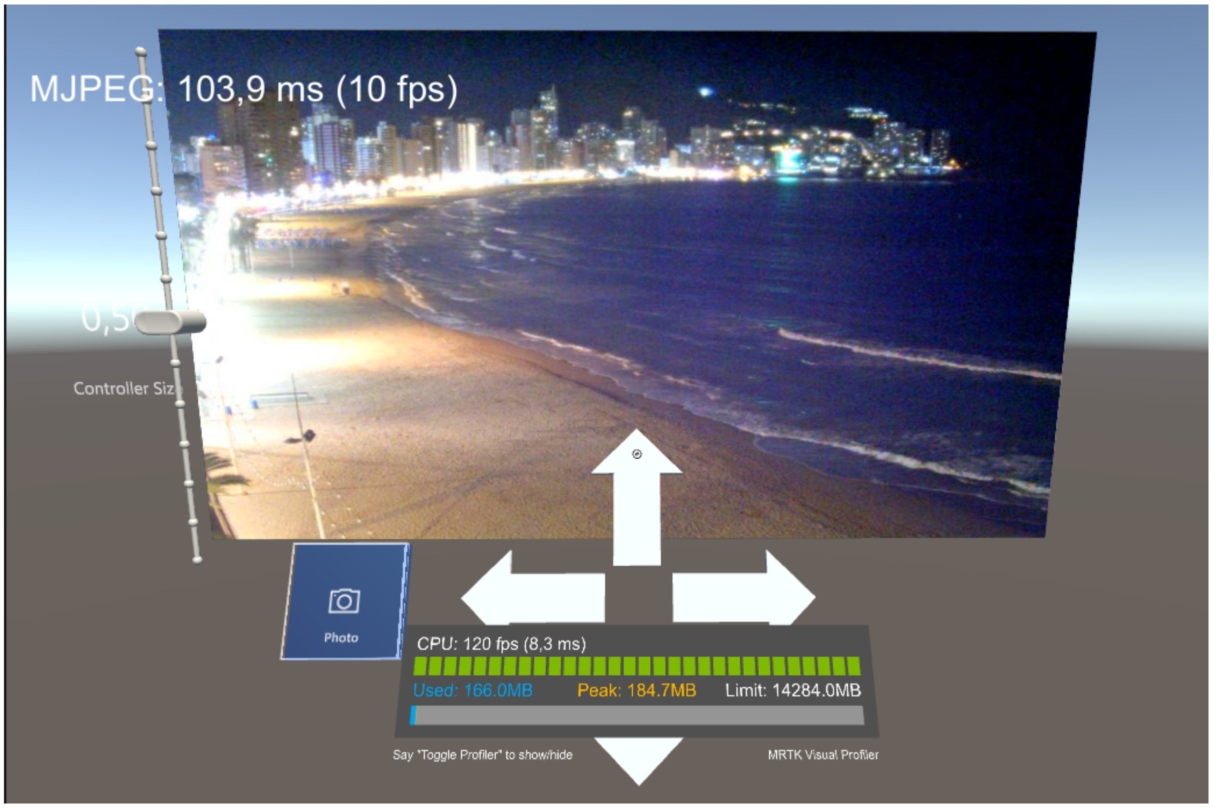


Figure 23 - Main Scene

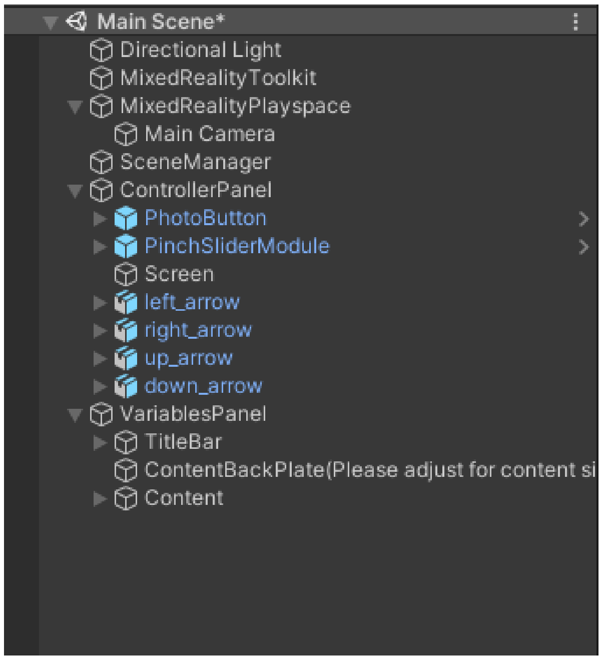


```
CPU: 120 fps (8,3 ms)
Used: 166.0MB    Peak: 184.7MB    Limit: 14284.0MB
```

Figure 24 - Main Scene on Play Mode

4.2.3.2 Scene Manager Script

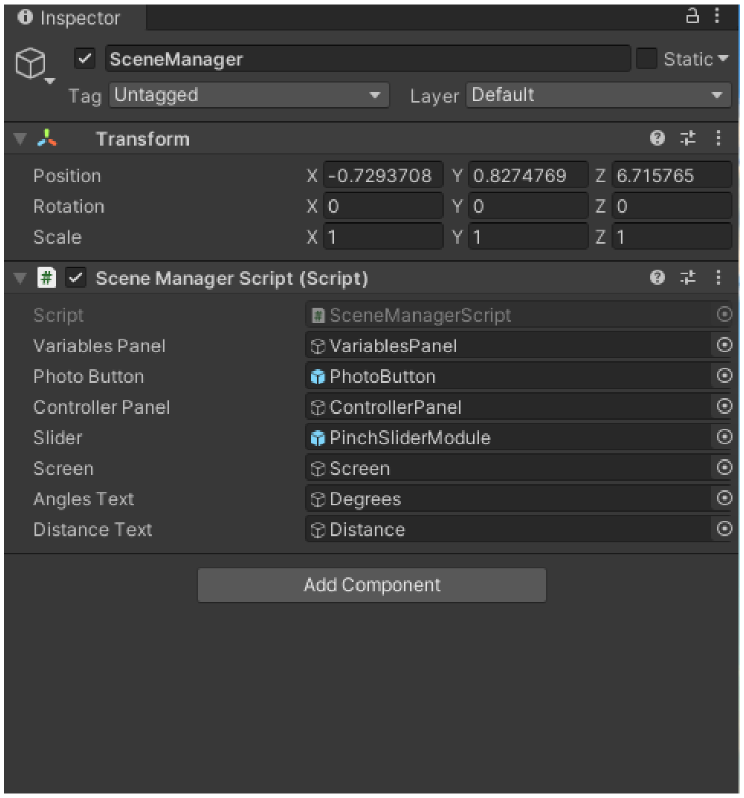
The components as shown in the figure 26 below is the basic structure of the Hololens application. The Main Scene has the Camera which is the way the user sees the objects, the SceneManager which is the application's main script, the ControllerPanel which has all the interaction objects for the user and the last one is the Variables' Panel that we saw above.



```
Main Scene*
  Directional Light
  MixedRealityToolkit
  MixedRealityPlayspace
    Main Camera
  SceneManager
  ControllerPanel
    PhotoButton
    PinchSliderModule
    Screen
    left_arrow
    right_arrow
    up_arrow
    down_arrow
  VariablesPanel
    TitleBar
    ContentBackPlate(Please adjust for content si
    Content
```

Figure 25 - Application Files

The basic functionalities of the AR application are described well from the main C# script that we have made and the Unity application executes.



Inspector

SceneManager Static

Tag Untagged Layer Default

Transform

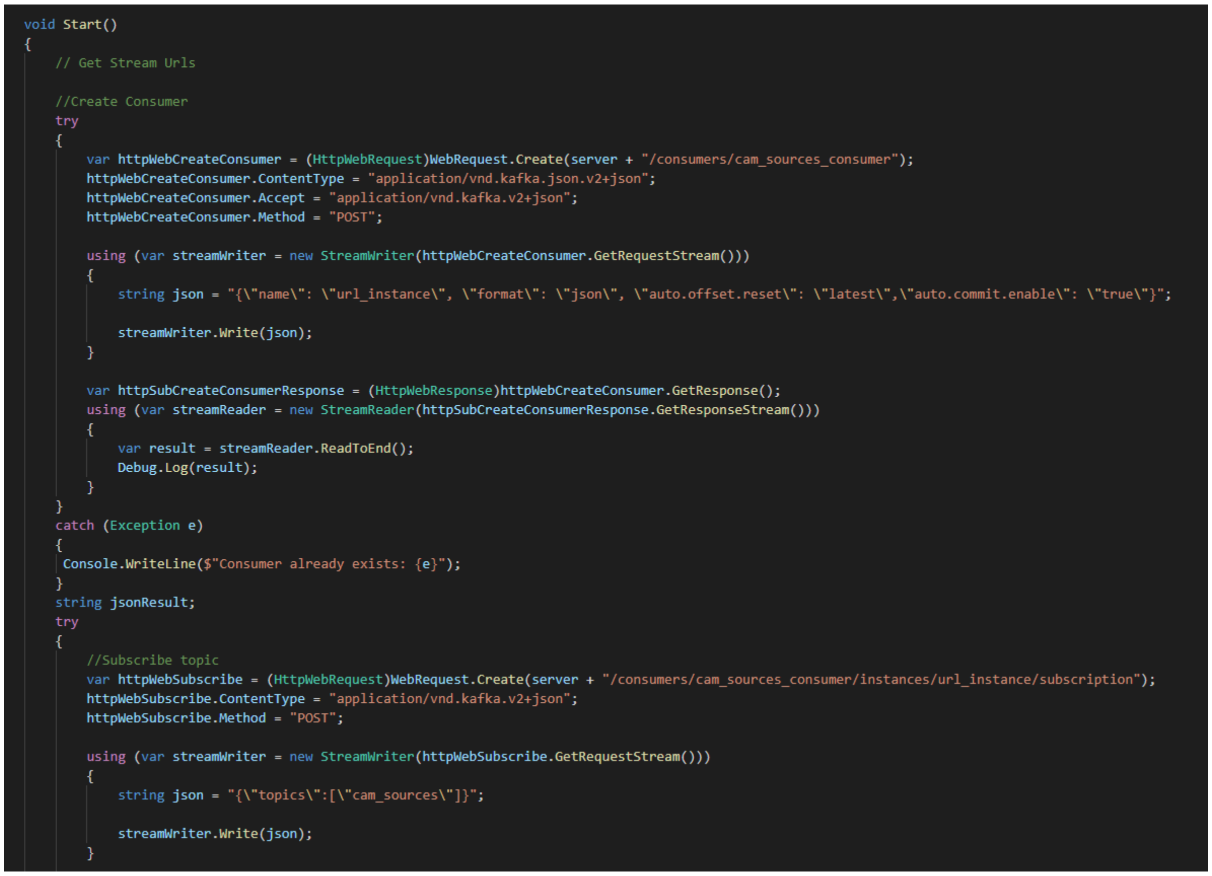
Position	X -0.7293708	Y 0.8274769	Z 6.715765
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Scene Manager Script (Script)

Script	SceneManagerScript
Variables Panel	VariablesPanel
Photo Button	PhotoButton
Controller Panel	ControllerPanel
Slider	PinchSliderModule
Screen	Screen
Angles Text	Degrees
Distance Text	Distance

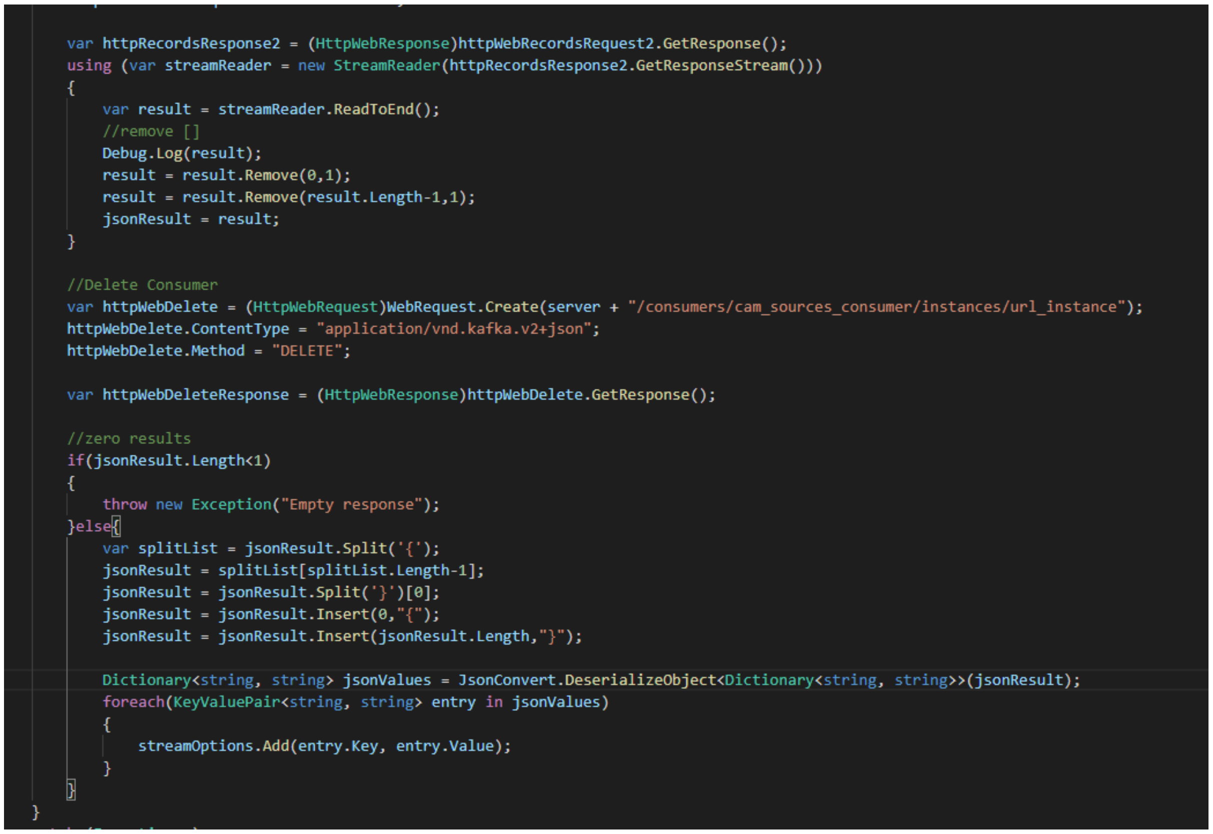
Figure 26 - Scene Manager Components

“Start” is the first function that will be run with the application’s execution. In this function, we create a new consumer and a topic that is subscribed to. Here we receive from a python script the stream URLs so it isn’t needed to change the links in code and redeploy the application in the glasses.



```
void Start()
{
    // Get Stream Urls

    //Create Consumer
    try
    {
        var httpWebCreateConsumer = (HttpWebRequest)WebRequest.Create(server + "/consumers/cam_sources_consumer");
        httpWebCreateConsumer.ContentType = "application/vnd.kafka.json.v2+json";
        httpWebCreateConsumer.Accept = "application/vnd.kafka.v2+json";
        httpWebCreateConsumer.Method = "POST";

        using (var streamWriter = new StreamWriter(httpWebCreateConsumer.GetRequestStream()))
        {
            string json = "{\"name\": \"url_instance\", \"format\": \"json\", \"auto.offset.reset\": \"latest\",\"auto.commit.enable\": \"true\"}";

            streamWriter.Write(json);
        }

        var httpSubCreateConsumerResponse = (HttpWebResponse)httpWebCreateConsumer.GetResponse();
        using (var streamReader = new StreamReader(httpSubCreateConsumerResponse.GetResponseStream()))
        {
            var result = streamReader.ReadToEnd();
            Debug.Log(result);
        }
    }
    catch (Exception e)
    {
     Console.WriteLine($"Consumer already exists: {e}");
    }
    string jsonResult;
    try
    {
        //Subscribe topic
        var httpWebSubscribe = (HttpWebRequest)WebRequest.Create(server + "/consumers/cam_sources_consumer/instances/url_instance/subscription");
        httpWebSubscribe.ContentType = "application/vnd.kafka.v2+json";
        httpWebSubscribe.Method = "POST";

        using (var streamWriter = new StreamWriter(httpWebSubscribe.GetRequestStream()))
        {
            string json = "{\"topics\":[\"cam_sources\"]}";

            streamWriter.Write(json);
        }
```

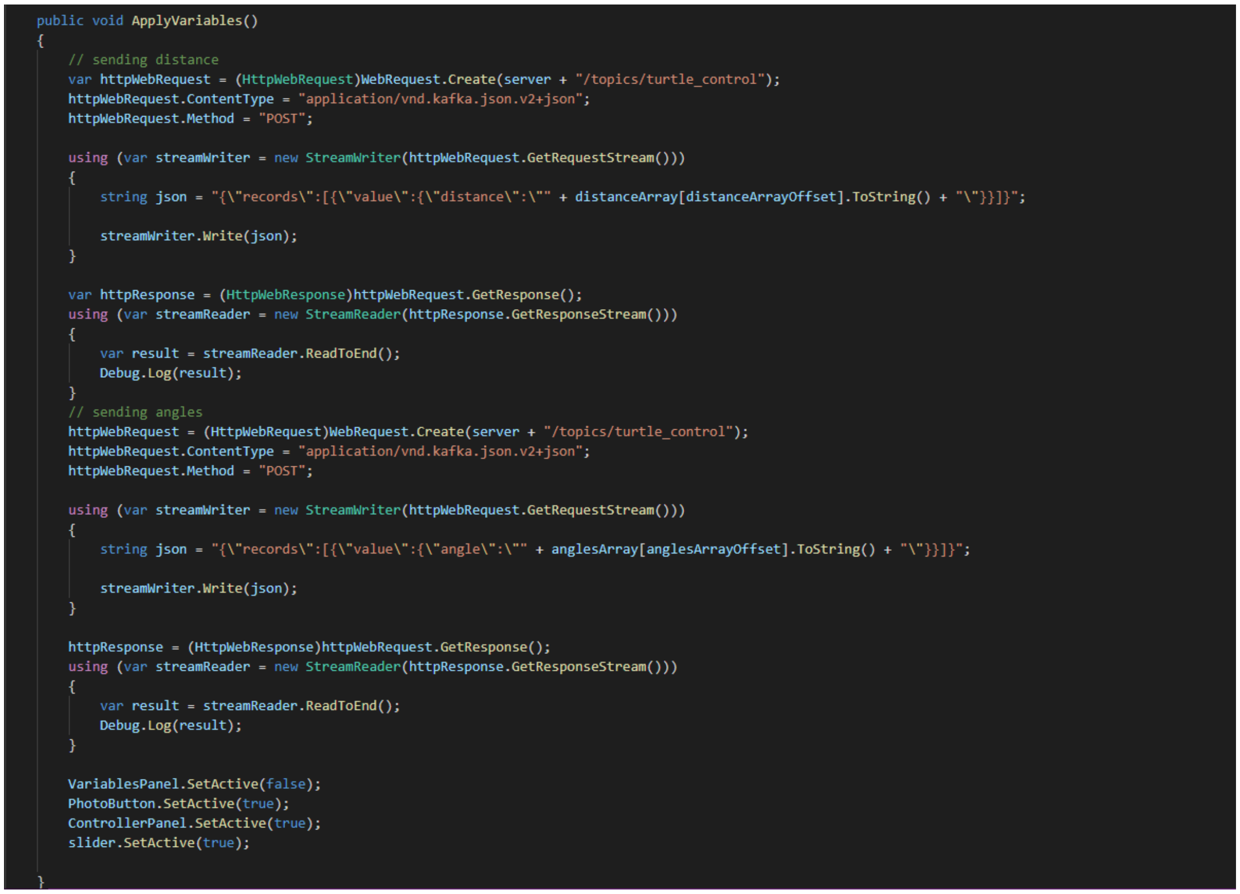
Figure 27 - Scene Manager Script Part 1



```
        var httpRecordsResponse2 = (HttpWebResponse)httpWebRecordsRequest2.GetResponse();
        using (var streamReader = new StreamReader(httpRecordsResponse2.GetResponseStream()))
        {
            var result = streamReader.ReadToEnd();
            //remove []
            Debug.Log(result);
            result = result.Remove(0,1);
            result = result.Remove(result.Length-1,1);
            jsonResult = result;
        }

        //Delete Consumer
        var httpWebDelete = (HttpWebRequest)WebRequest.Create(server + "/consumers/cam_sources_consumer/instances/url_instance");
        httpWebDelete.ContentType = "application/vnd.kafka.v2+json";
        httpWebDelete.Method = "DELETE";

        var httpWebDeleteResponse = (HttpWebResponse)httpWebDelete.GetResponse();

        //zero results
        if(jsonResult.Length<1)
        {
            throw new Exception("Empty response");
        }else{
            var splitList = jsonResult.Split('{');
            jsonResult = splitList[splitList.Length-1];
            jsonResult = jsonResult.Split('}')[0];
            jsonResult = jsonResult.Insert(0,"{");
            jsonResult = jsonResult.Insert(jsonResult.Length,"}");

            Dictionary<string, string> jsonValues = JsonConvert.DeserializeObject<Dictionary<string, string>>(jsonResult);
            foreach(KeyValuePair<string, string> entry in jsonValues)
            {
                streamOptions.Add(entry.Key, entry.Value);
            }
        }
    }
```

Figure 28 - Scene Manager Script Part 2

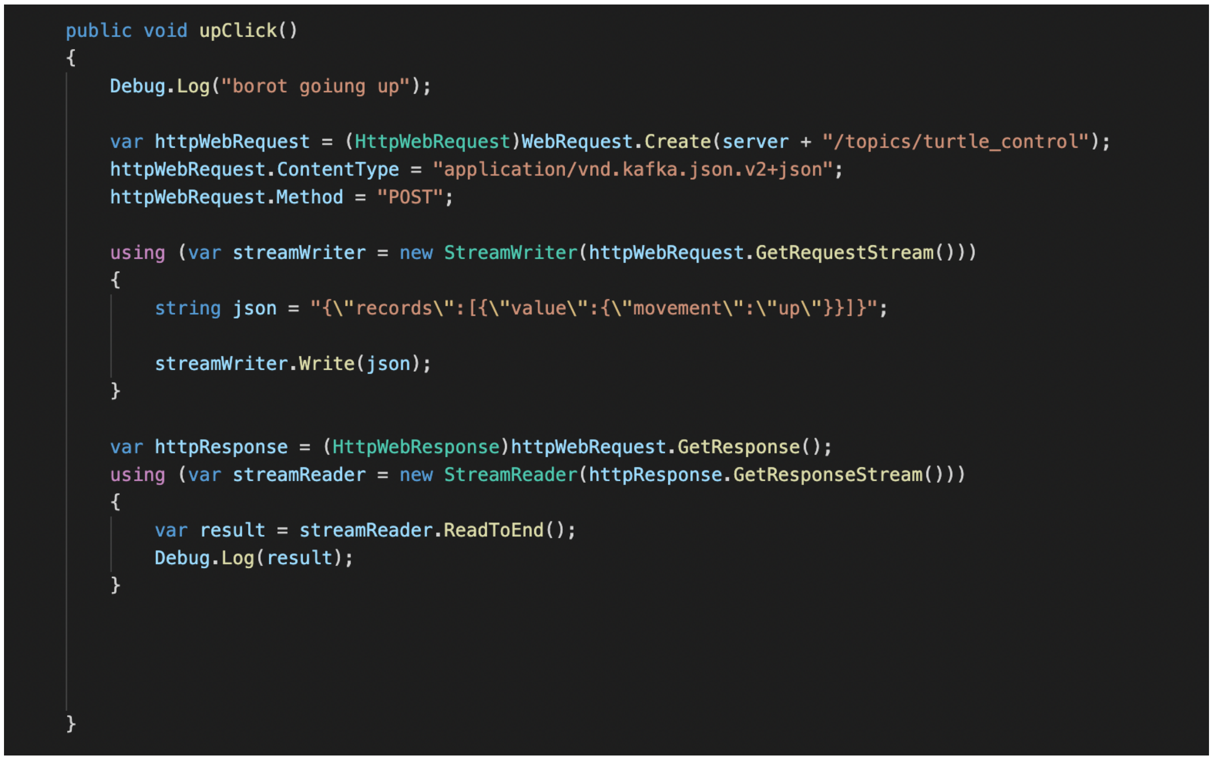
In the figure 29 below we see the parameter’s panel. We make two Kafka post requests in the /turtle_control topic with the variables that we get from the input fields. At the end of this scene, we disable the first scene and we enable all the other necessary components for the second screen.



```
public void ApplyVariables()
{
    // sending distance
    var httpWebRequest = (HttpWebRequest)WebRequest.Create(server + "/topics/turtle_control");
    httpWebRequest.ContentType = "application/vnd.kafka.json.v2+json";
    httpWebRequest.Method = "POST";

    using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
    {
        string json = "{\"records\":[{\"value\":{\"distance\":\"" + distanceArray[distanceArrayOffset].ToString() + "\"}}]}";

        streamWriter.Write(json);
    }

    var httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
    using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
    {
        var result = streamReader.ReadToEnd();
        Debug.Log(result);
    }
    // sending angles
    httpWebRequest = (HttpWebRequest)WebRequest.Create(server + "/topics/turtle_control");
    httpWebRequest.ContentType = "application/vnd.kafka.json.v2+json";
    httpWebRequest.Method = "POST";

    using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
    {
        string json = "{\"records\":[{\"value\":{\"angle\":\"" + anglesArray[anglesArrayOffset].ToString() + "\"}}]}";

        streamWriter.Write(json);
    }

    httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
    using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
    {
        var result = streamReader.ReadToEnd();
        Debug.Log(result);
    }

    VariablesPanel.SetActive(false);
    PhotoButton.SetActive(true);
    ControllerPanel.SetActive(true);
    slider.SetActive(true);

}
```

Figure 29 - Scene Manager Script Part 3

In the figure 30 below we see a sample of how a navigation button is implemented, which makes a post request with a Kafka message on the turtle_control topic.

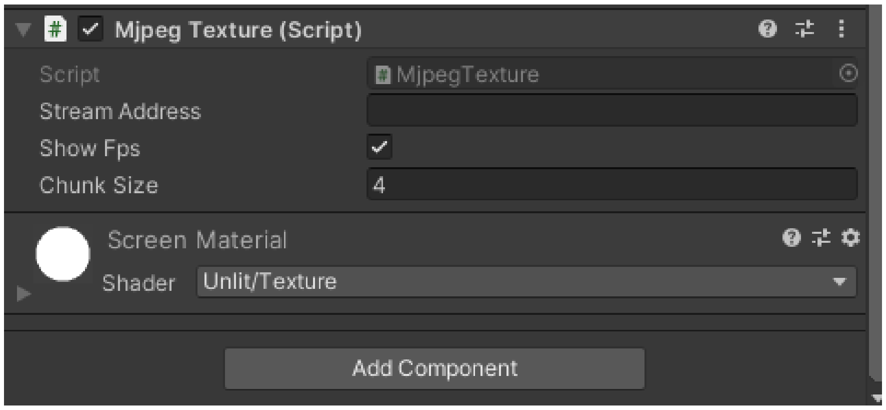


```
public void upClick()
{
    Debug.Log("borot goiung up");

    var httpWebRequest = (HttpWebRequest)WebRequest.Create(server + "/topics/turtle_control");
    httpWebRequest.ContentType = "application/vnd.kafka.json.v2+json";
    httpWebRequest.Method = "POST";

    using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
    {
        string json = "{\"records\":[{\"value\":{\"movement\":\"up\"}}]}";

        streamWriter.Write(json);
    }

    var httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
    using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
    {
        var result = streamReader.ReadToEnd();
        Debug.Log(result);
    }




}
```

Figure 30 - Scene Manager Script Part 4

4.2.3.3 Mjpeg Video Stream

The ROS web_video_server serves the camera's feed as a mjpeg video stream. It wasn't an available way to display the stream's mjpeg content in a UI mesh. So it used a custom-made script that can receive mjpeg video from an HTTP source and render it to this mesh as depicted in Figures 31 and 32.

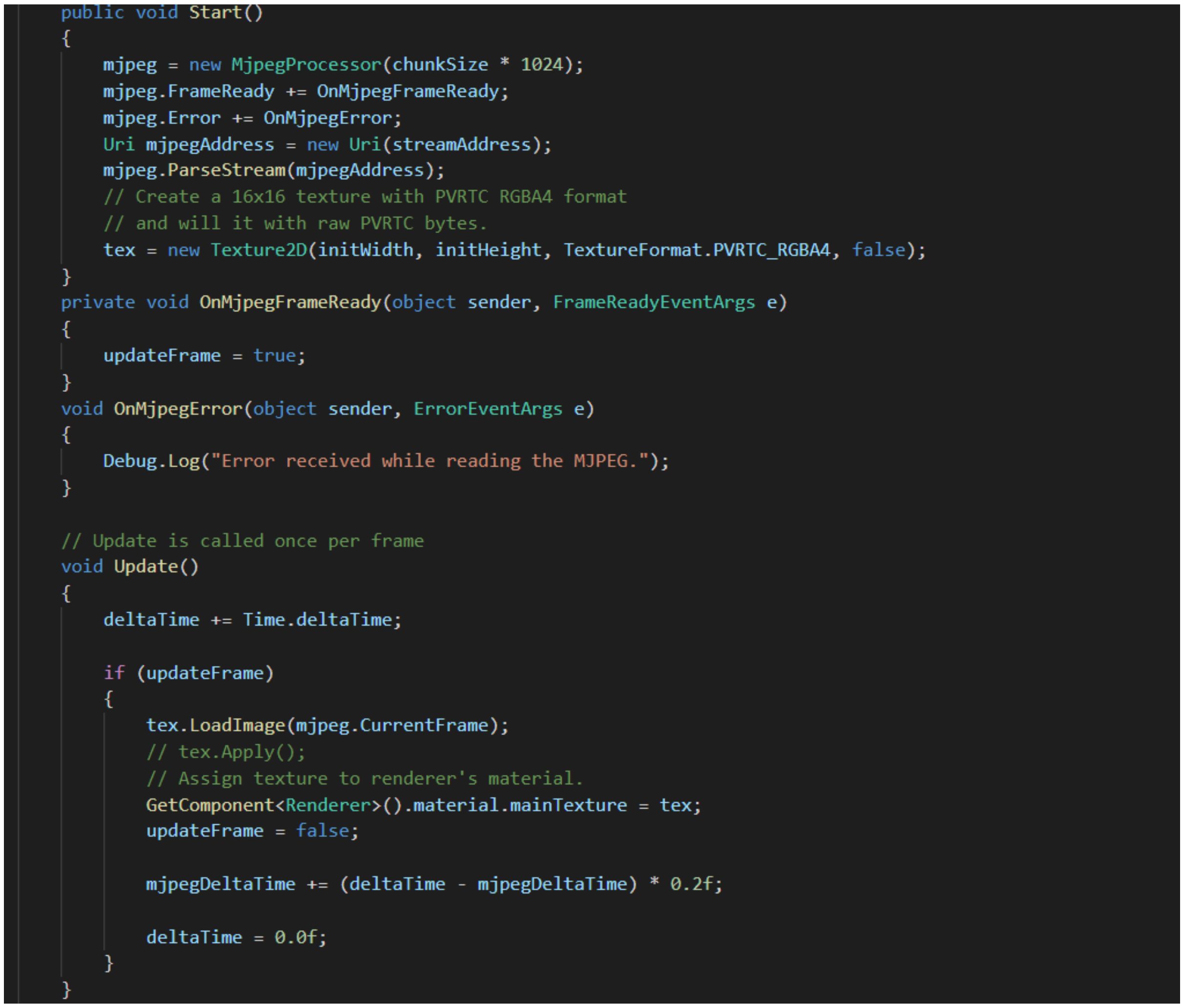


Mjpeg Texture (Script)

Script: MjpegTexture
Stream Address:
Show Fps: ✓
Chunk Size: 4

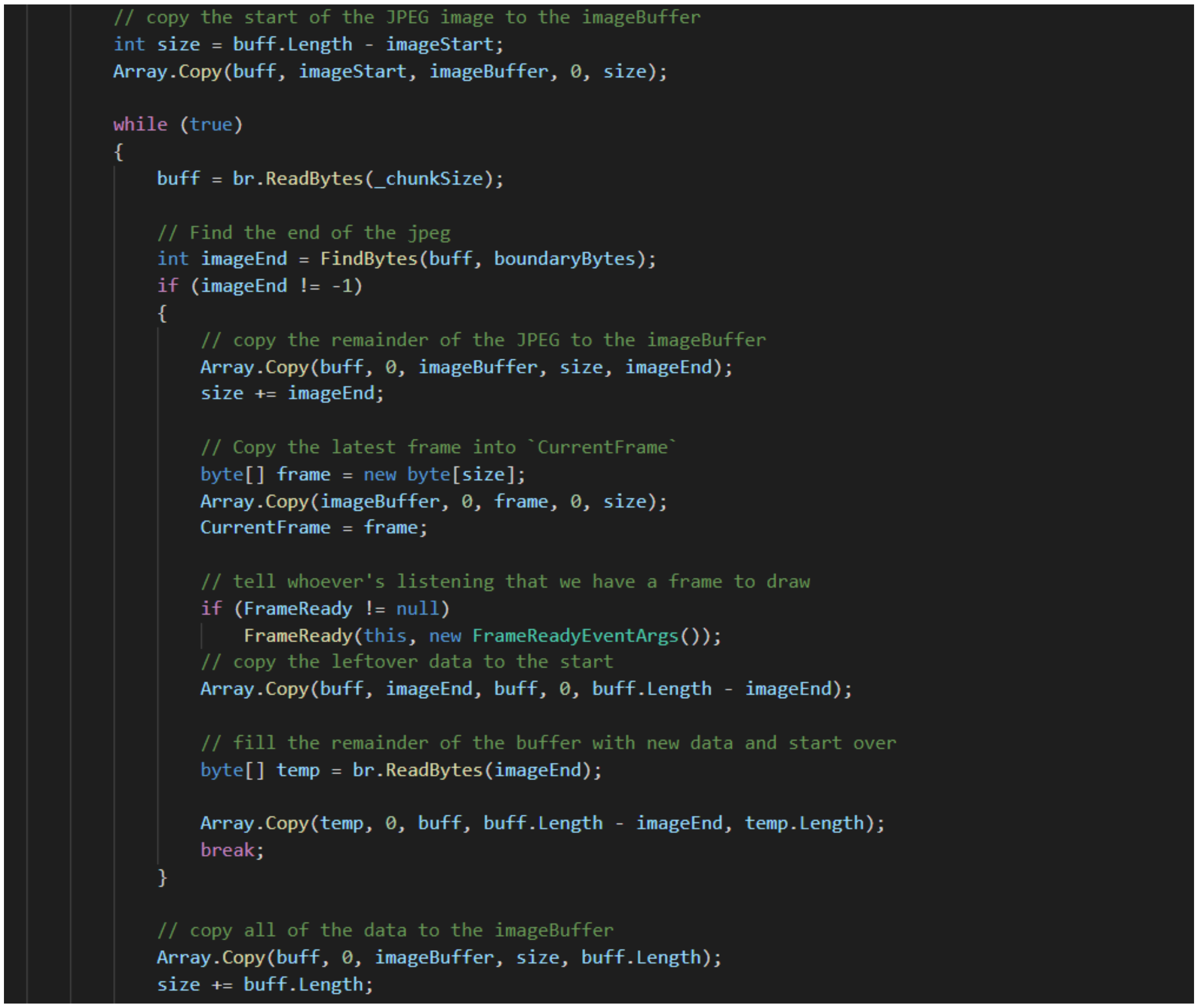
Screen Material
Shader: Unlit/Texture

Figure 31 - Mjpeg Texture



```
public void Start()
{
    mjpeg = new MjpegProcessor(chunkSize * 1024);
    mjpeg.FrameReady += OnMjpegFrameReady;
    mjpeg.Error += OnMjpegError;
    Uri mjpegAddress = new Uri(streamAddress);
    mjpeg.ParseStream(mjpegAddress);
    // Create a 16x16 texture with PVRTC RGBA4 format
    // and will it with raw PVRTC bytes.
    tex = new Texture2D(initWidth, initHeight, TextureFormat.PVRTC_RGBA4, false);
}
private void OnMjpegFrameReady(object sender, FrameReadyEventArgs e)
{
    updateFrame = true;
}
void OnMjpegError(object sender, ErrorEventArgs e)
{
    Debug.Log("Error received while reading the MJPEG.");
}

// Update is called once per frame
void Update()
{
    deltaTime += Time.deltaTime;

    if (updateFrame)
    {
        tex.LoadImage(mjpeg.CurrentFrame);
        // tex.Apply();
        // Assign texture to renderer's material.
        GetComponent<Renderer>().material.mainTexture = tex;
        updateFrame = false;

        mjpegDeltaTime += (deltaTime - mjpegDeltaTime) * 0.2f;

        deltaTime = 0.0f;
    }
}
```

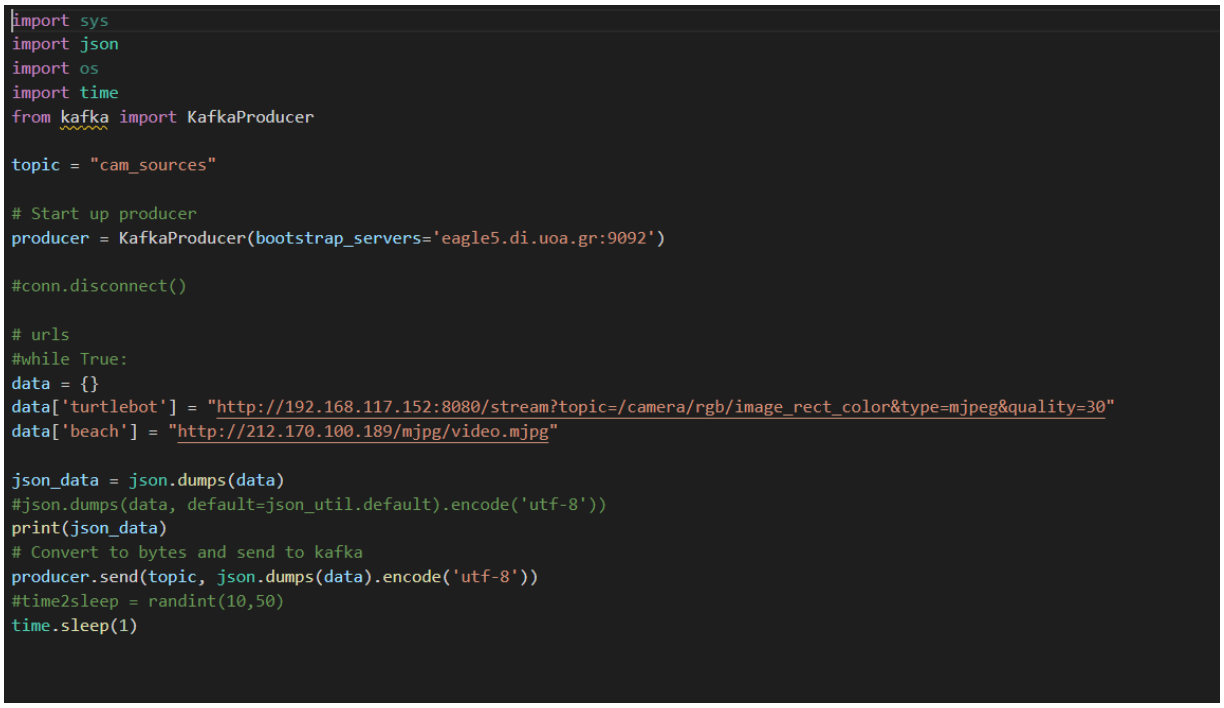
Figure 32 - Mjpeg Script Part 1



```
        // copy the start of the JPEG image to the imageBuffer
        int size = buff.Length - imageStart;
        Array.Copy(buff, imageStart, imageBuffer, 0, size);

        while (true)
        {
            buff = br.ReadBytes(_chunkSize);

            // Find the end of the jpeg
            int imageEnd = FindBytes(buff, boundaryBytes);
            if (imageEnd != -1)
            {
                // copy the remainder of the JPEG to the imageBuffer
                Array.Copy(buff, 0, imageBuffer, size, imageEnd);
                size += imageEnd;

                // Copy the latest frame into `CurrentFrame`
                byte[] frame = new byte[size];
                Array.Copy(imageBuffer, 0, frame, 0, size);
                CurrentFrame = frame;

                // tell whoever's listening that we have a frame to draw
                if (FrameReady != null)
                    FrameReady(this, new FrameReadyEventArgs());
                // copy the leftover data to the start
                Array.Copy(buff, imageEnd, buff, 0, buff.Length - imageEnd);

                // fill the remainder of the buffer with new data and start over
                byte[] temp = br.ReadBytes(imageEnd);

                Array.Copy(temp, 0, buff, buff.Length - imageEnd, temp.Length);
                break;
            }

            // copy all of the data to the imageBuffer
            Array.Copy(buff, 0, imageBuffer, size, buff.Length);
            size += buff.Length;
```

Figure 33 - Mjpeg Script Part 2

To send the ROS server’s stream addresses in the unity program that runs on Holelens, we publish the addresses with a python script in the /cam_sources topic on the Kafka server. Next, when the Hololens application starts up, the Scene Manager Script that is subscribed to the topic will receive the URLs. In the figure 33 below we have a python script example for URL publication.



```python
import sys
import json
import os
import time
from kafka import KafkaProducer

topic = "cam_sources"

# Start up producer
producer = KafkaProducer(bootstrap_servers='eagle5.di.uoa.gr:9092')

#conn.disconnect()

# urls
#while True:
data = {}
data['turtlebot'] = "http://192.168.117.152:8080/stream?topic=/camera/rgb/image_rect_color&type=mjpeg&quality=30"
data['beach'] = "http://212.170.100.189/mjpg/video.mjpg"

json_data = json.dumps(data)
#json.dumps(data, default=json_util.default).encode('utf-8'))
print(json_data)
# Convert to bytes and send to kafka
producer.send(topic, json.dumps(data).encode('utf-8'))
#time2sleep = randint(10,50)
time.sleep(1)
```

Figure 34 - Url Publisher Python Script

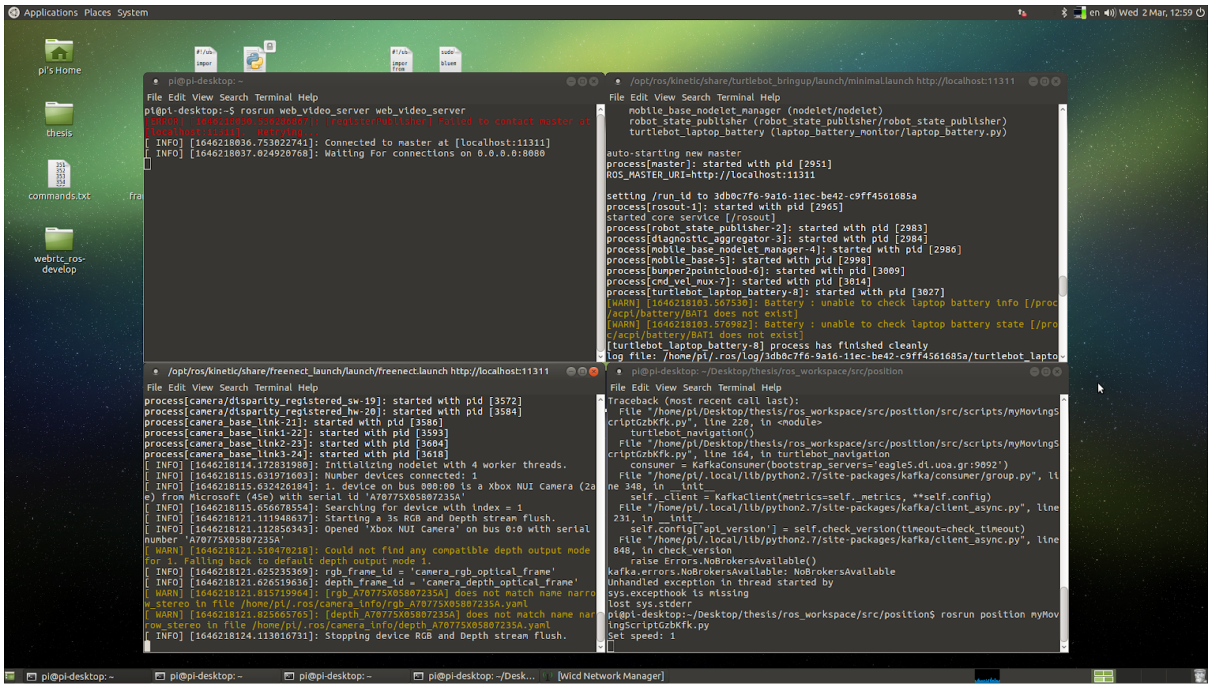
5. EXPERIMENTAL EVALUATION

5.1 Experiment Scenario

The experiments tested the installation and the functionalities of our application. The goal was the navigation of the turtlebot inside a room surrounded by obstacles without the operator having visual contact with the robot.

The 3 main nodes of the experiment are a turtlebot 2 with a raspberry version 3 tied on it, the Microsoft's hololens 2 for the navigation, and a Kafka Server for the pub/sub command streaming.

To begin, all the necessary software components have to executed in the robot's raspberry. This includes the turtlebot launch file, the Kinect camera, the ROS package controller, and the ros_web_video_server. We check that the Kafka Server for the navigation is ready and we deploy the Unity application on Hololens. The next step is the publication of the video stream's URL on the right topic and the opening of the program on Hololens.



```
pi@pi-desktop:~$ rosrun web_video_server web_video_server
[ERROR] [1646218030.536286867]: [registerPublisher] Failed to contact master at
[localhost:11311].  Retrying...
[ INFO] [1646218036.753022741]: Connected to master at [localhost:11311]
[ INFO] [1646218037.024920768]: Waiting For connections on 0.0.0.0:8080
```

```
    mobile_base_nodelet_manager (nodelet/nodelet)
    robot_state_publisher (robot_state_publisher/robot_state_publisher)
    turtlebot_laptop_battery (laptop_battery_monitor/laptop_battery.py)

auto-starting new master
process[master]: started with pid [2951]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 3db0c7f6-9a16-11ec-be42-c9ff4561685a
process[rosout-1]: started with pid [2965]
started core service [/rosout]
process[robot_state_publisher-2]: started with pid [2983]
process[diagnostic_aggregator-3]: started with pid [2984]
process[mobile_base_nodelet_manager-4]: started with pid [2986]
process[mobile_base-5]: started with pid [2998]
process[bumper2pointcloud-6]: started with pid [3009]
process[cmd_vel_mux-7]: started with pid [3014]
process[turtlebot_laptop_battery-8]: started with pid [3027]
[WARN] [1646218103.567530]: Battery : unable to check laptop battery info [/proc
/acpi/battery/BAT1 does not exist]
[WARN] [1646218103.576982]: Battery : unable to check laptop battery state [/pro
c/acpi/battery/BAT1 does not exist]
[turtlebot_laptop_battery-8] process has finished cleanly
log file: /home/pi/.ros/log/3db0c7f6-9a16-11ec-be42-c9ff4561685a/turtlebot_lapto
```

```
process[camera/disparity_registered_sw-19]: started with pid [3572]
process[camera/disparity_registered_hw-20]: started with pid [3584]
process[camera_base_link-21]: started with pid [3586]
process[camera_base_link1-22]: started with pid [3593]
process[camera_base_link2-23]: started with pid [3604]
process[camera_base_link3-24]: started with pid [3618]
[ INFO] [1646218114.172831980]: Initializing nodelet with 4 worker threads.
[ INFO] [1646218115.631971603]: Number devices connected: 1
[ INFO] [1646218115.632426184]: 1. device on bus 000:00 is a Xbox NUI Camera (2a
e) from Microsoft (45e) with serial id 'A70775X05807235A'
[ INFO] [1646218115.656678554]: Searching for device with index = 1
[ INFO] [1646218121.111948637]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1646218121.112856343]: Opened 'Xbox NUI Camera' on bus 0:0 with serial
number 'A70775X05807235A'
[ WARN] [1646218121.510470218]: Could not find any compatible depth output mode
for 1. Falling back to default depth output mode 1.
[ INFO] [1646218121.625235369]: rgb_frame_id = 'camera_rgb_optical_frame'
[ INFO] [1646218121.626519636]: depth_frame_id = 'camera_depth_optical_frame'
[ WARN] [1646218121.815719964]: [rgb_A70775X05807235A] does not match name narro
w_stereo in file /home/pi/.ros/camera_info/rgb_A70775X05807235A.yaml
[ WARN] [1646218121.825665765]: [depth_A70775X05807235A] does not match name nar
row_stereo in file /home/pi/.ros/camera_info/depth_A70775X05807235A.yaml
[ INFO] [1646218124.113016731]: Stopping device RGB and Depth stream flush.
```

```
Traceback (most recent call last):
  File "/home/pi/Desktop/thesis/ros_workspace/src/position/src/scripts/myMovingS
criptGzbKfk.py", line 220, in <module>
    turtlebot_navigation()
  File "/home/pi/Desktop/thesis/ros_workspace/src/position/src/scripts/myMovingS
criptGzbKfk.py", line 164, in turtlebot_navigation
    consumer = KafkaConsumer(bootstrap_servers='eagle5.di.uoa.gr:9092')
  File "/home/pi/.local/lib/python2.7/site-packages/kafka/consumer/group.py", li
ne 348, in __init__
    self._client = KafkaClient(metrics=self._metrics, **self.config)
  File "/home/pi/.local/lib/python2.7/site-packages/kafka/client_async.py", line
 231, in __init__
    self.config['api_version'] = self.check_version(timeout=check_timeout)
  File "/home/pi/.local/lib/python2.7/site-packages/kafka/client_async.py", line
 848, in check_version
    raise Errors.NoBrokersAvailable()
kafka.errors.NoBrokersAvailable: NoBrokersAvailable
Unhandled exception in thread started by
sys.excepthook is missing
lost sys.stderr
pi@pi-desktop:~/Desktop/thesis/ros_workspace/src/position$ rosrun position myMov
ingScriptGzbKfk.py
Set speed: 1
```

Applications Places System
en Wed 2 Mar, 12:59
pi's Home
thesis
commands.txt
webrtc_ros-develop
pi@pi-desktop: ~
/opt/ros/kinetic/share/turtlebot_bringup/launch/minimal.launch http://localhost:11311
/opt/ros/kinetic/share/freenect_launch/launch/freenect.launch http://localhost:11311
pi@pi-desktop: ~/Desktop/thesis/ros_workspace/src/position
File Edit View Search Terminal Help
pi@pi-desktop: ~/Desk...
[Wicd Network Manager]

Figure 35 - Application's Running Commands

During the execution of the application, the user was pressing the navigation arrows so that the robot follows the right path. As mentioned above, the movement of the turtlebot is not continuous and with each command, it travels a specific distance that we have set at the beginning of the application. This way any slight delay that the video streaming may have was prevented. This has resulted that the first version of this application could be completely functional. So the test was completed successfully and the application ended normally.

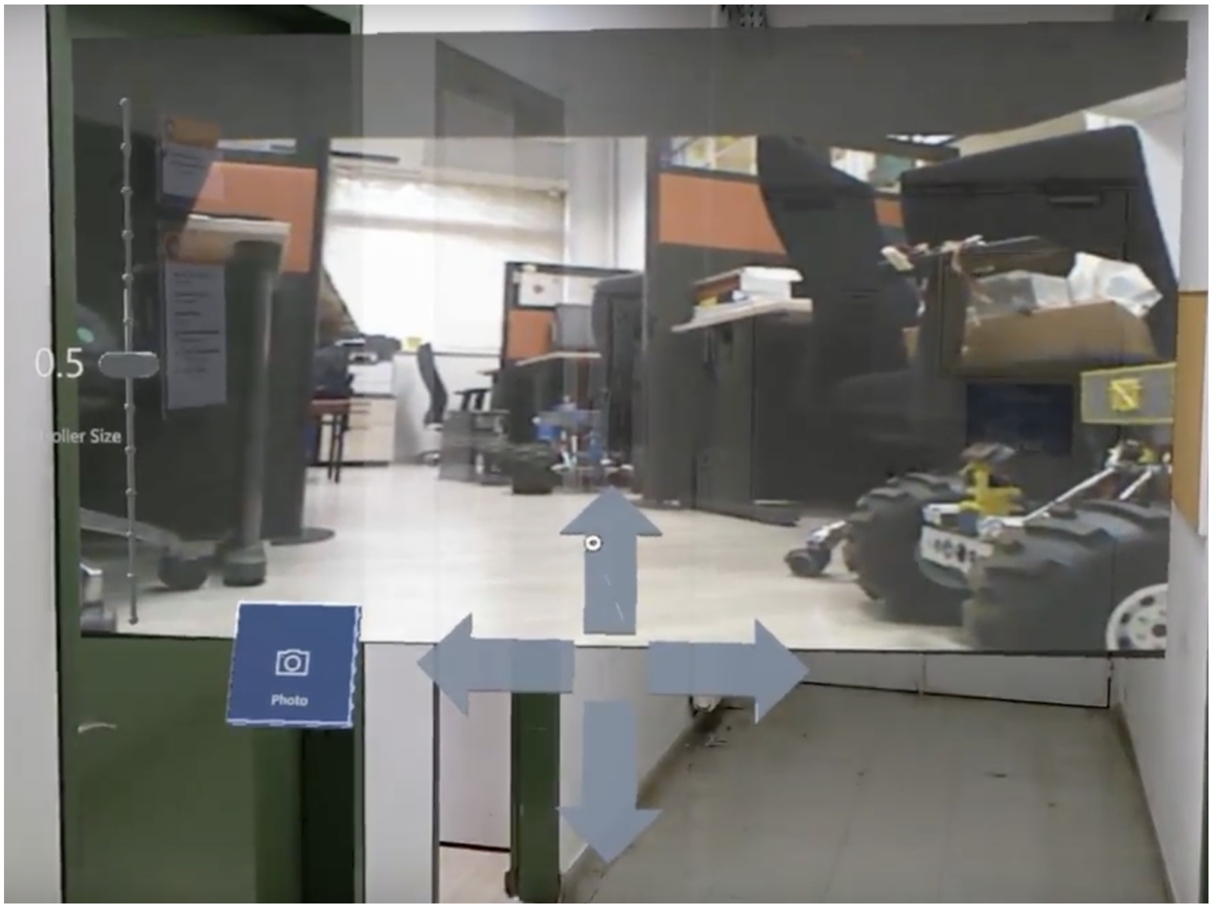
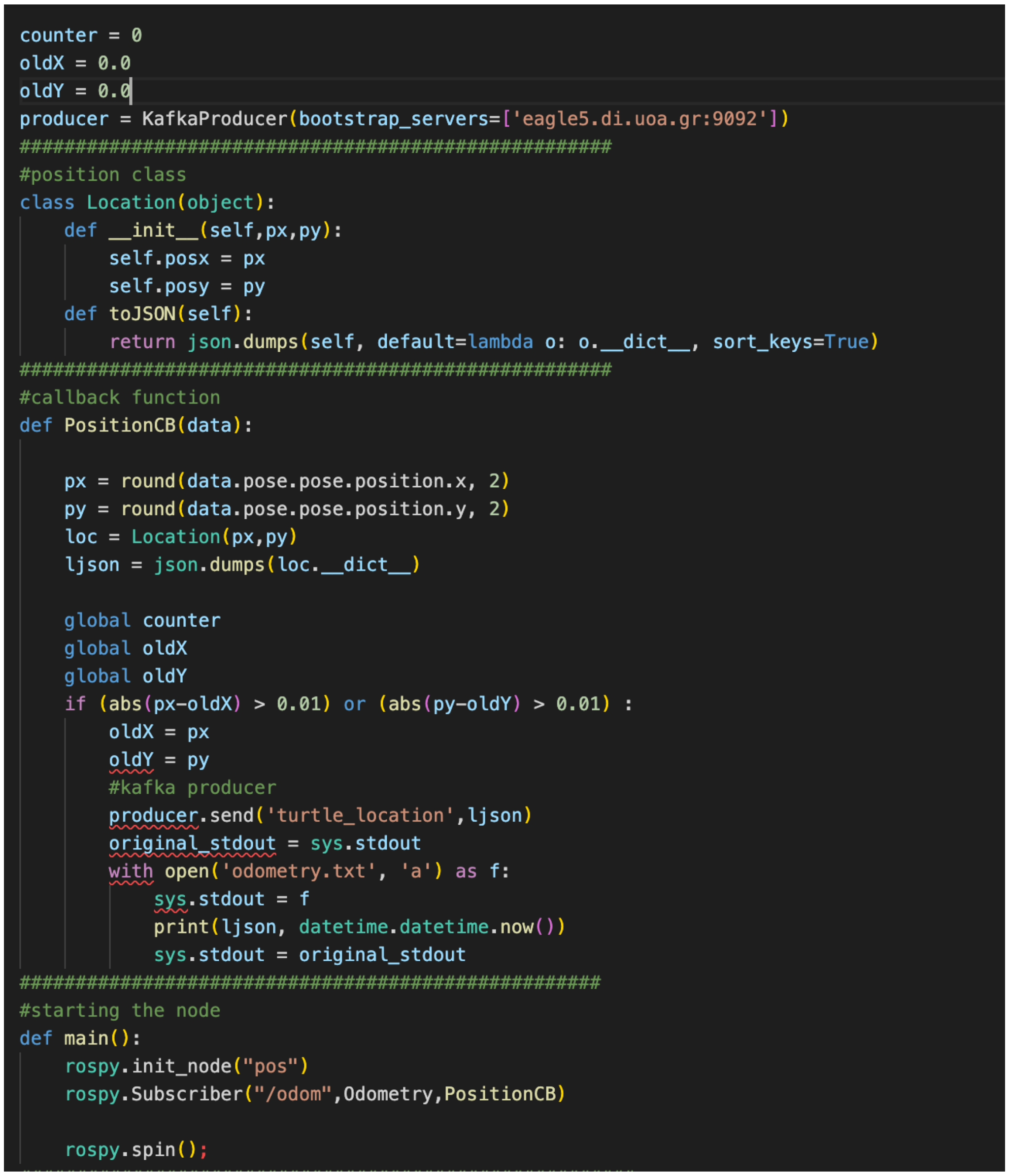


Figure 36 - Experiment's Screen

5.2 Precision Check

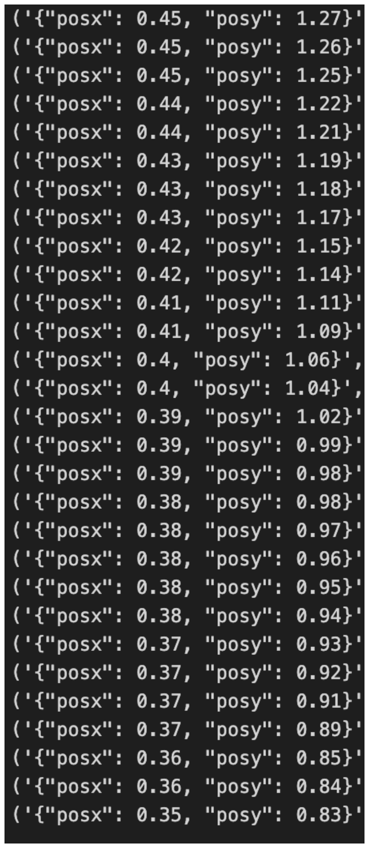
Eventually in the experiment, we want to have a way of checking if the turtlebot is moving at the predetermined distance step after each navigation command. The solution to this problem is called Odometry and it is the main tool that turtlebot has to find its location in the environment. With odometry, the robot uses the data from its motion sensors to estimate the change in its position over time. The TurtleBot estimates its orientation and position relative to a starting location that is given in terms of an x and y position and an orientation around the z (upward) axis. As mentioned earlier, all the data from turtlebot are circulating through topics, so for the odometry, we have the topic /odom. A python script as shown in figure 37 called position subscribes to the

/odom topic and publishes periodically on a Kafka topic the change of the robot's location. Also, the output is saved locally in JSON format as shown in figure 38.



```
counter = 0
oldX = 0.0
oldY = 0.0
producer = KafkaProducer(bootstrap_servers=['eagle5.di.uoa.gr:9092'])
#####################################################
#position class
class Location(object):
    def __init__(self,px,py):
        self.posx = px
        self.posy = py
    def toJSON(self):
        return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True)
#####################################################
#callback function
def PositionCB(data):

    px = round(data.pose.pose.position.x, 2)
    py = round(data.pose.pose.position.y, 2)
    loc = Location(px,py)
    ljson = json.dumps(loc.__dict__)

    global counter
    global oldX
    global oldY
    if (abs(px-oldX) > 0.01) or (abs(py-oldY) > 0.01) :
        oldX = px
        oldY = py
        #kafka producer
        producer.send('turtle_location',ljson)
        original_stdout = sys.stdout
        with open('odometry.txt', 'a') as f:
            sys.stdout = f
            print(ljson, datetime.datetime.now())
            sys.stdout = original_stdout
####################################################
#starting the node
def main():
    rospy.init_node("pos")
    rospy.Subscriber("/odom",Odometry,PositionCB)

    rospy.spin();
```

Figure 37 - Position Script



```
('{"posx": 0.45, "posy": 1.27}'
('{"posx": 0.45, "posy": 1.26}'
('{"posx": 0.45, "posy": 1.25}'
('{"posx": 0.44, "posy": 1.22}'
('{"posx": 0.44, "posy": 1.21}'
('{"posx": 0.43, "posy": 1.19}'
('{"posx": 0.43, "posy": 1.18}'
('{"posx": 0.43, "posy": 1.17}'
('{"posx": 0.42, "posy": 1.15}'
('{"posx": 0.42, "posy": 1.14}'
('{"posx": 0.41, "posy": 1.11}'
('{"posx": 0.41, "posy": 1.09}'
('{"posx": 0.4, "posy": 1.06}',
('{"posx": 0.4, "posy": 1.04}',
('{"posx": 0.39, "posy": 1.02}'
('{"posx": 0.39, "posy": 0.99}'
('{"posx": 0.39, "posy": 0.98}'
('{"posx": 0.38, "posy": 0.98}'
('{"posx": 0.38, "posy": 0.97}'
('{"posx": 0.38, "posy": 0.96}'
('{"posx": 0.38, "posy": 0.95}'
('{"posx": 0.38, "posy": 0.94}'
('{"posx": 0.37, "posy": 0.93}'
('{"posx": 0.37, "posy": 0.92}'
('{"posx": 0.37, "posy": 0.91}'
('{"posx": 0.37, "posy": 0.89}'
('{"posx": 0.36, "posy": 0.85}'
('{"posx": 0.36, "posy": 0.84}'
('{"posx": 0.35, "posy": 0.83}'
```

Figure 38 - Position JSON Output

6. RELATED WORK

6.1 Concept and architecture for programming industrial robots using augmented reality with mobile devices like Microsoft HoloLens

This work analyses techniques about the human-robot interaction, based on mobile devices, such as smartphones that are using augmented reality, or the Hololens which are using Mixed Reality. The combination of the device's sensors' data with the perception and abilities of a human operator gives to developers the opportunity for creating new applications with impressive capabilities. The visualization of the current robot state and the robot environment that is captured by different sensors and processed with both machine and human vision can increase the assisted workplaces or robot installations. Due to the robots' remote connection, it is possible for a remote maintenance procedure or a faster startup of the industrial robots.

6.2 Interactive Robots Control Using Mixed Reality

This work is one more mixed reality-based approach for interactive control of robotic manipulators and mobile platforms. In particular, they designed an interactive and understandable interface for human-robot interaction. The interface visualizes and also provides the tools for the robot path programming. The path visualization assists workers understand the robot behavior, it is important for the safety of human-robot interaction. The paper presents an architecture of that system and the implementation for an industrial robot KUKA iiwa and mobile robot platform Plato. The main issue of a multi-platform system is related to the synchronization of coordinate frames for all elements. The three setting options that ended up for dealing with this problem are manual, a camera with markers, and point clouds processing. The interface is implemented on Microsoft HoloLens and evaluated on a sample of users.

6.3 Remote Supervision of an Autonomous Surface Vehicle using Virtual Reality

This approach is more GUI-focused, they compared three different Graphical User Interfaces (GUI) that they have designed and implemented to enable human supervision of an Autonomous Surface Vehicle (ASV). Special attention has been paid to providing tools for safe navigation and giving the user a good overall understanding of the surrounding world while keeping the cognitive load at a low level. Their findings indicate that a GUI in 3D, presented either on a screen or in a Virtual Reality (VR) setting provides several benefits compared to a Baseline GUI representing traditional tools.

6.4 Improving Collocated Robot Teleoperation with Augmented Reality

As we discuss before robot teleoperation can be a challenging task, often requiring a great deal of user training and expertise, especially for platforms with high degrees of freedom e.g. industrial manipulators and aerial robots. Users regularly battle to synthesize the information robots collect (e.g., a camera stream) with relevant knowledge of how the robot moves in the environment. So this work investigates how the progress in augmented reality technologies is creating an unused design space for mediating robot teleoperation by enabling novel forms of intuitive, visual feedback. They prototype several aerial robot teleoperation interfaces using AR, which are evaluated in a forty-eight-hour participant user study where participants completed an environmental inspection task. Their new interface designs provided several objective and subjective performance benefits over existing systems, which often force users into an undesirable paradigm that divides user attention between monitoring the robot and monitoring the robot's camera feed.

7. FUTURE WORK

The application has a lot of potential for usage in a wide range of applications. For Future work, the priority has been the improvement of delays on the video level. In this implementation the video's stream protocol is mjpeg. The improvements can be achieved with other types of streaming like webRTC (RFC 7478).

This standard is supported by Microsoft and at this moment has implemented Unity components and examples. It is called MixedReality-WebRTC Unity integration and implements the WebRTC project.

WebRTC is a free open-source project that provides real-time communication. Initially, it was developed for web browsers but later it was expanded to other applications too, such as VoIP and peer-to-peer file sharing. An important feature of WebRTC that benefits our application is the reliable session establishment.

This is true for Network Address Translators (NAT), something that hinders and may block other communications and collaboration protocols. The reliable operation avoids server-relayed media and thereby reduces latency and increases quality. It also reduces the server load.

The main components of this implementation are a PeerConnection, a Signaler, and the local and remote video. The PeerConnection component is provided by the Unity integration of MixedReality-WebRTC and has various settings to configure the Connection's behavior. The Signaler is an essential part because is a way to discover and select a remote peer and to send to and receive from it the SDP messages that are necessary to establish that connection. The WebRTC standard specifies how a peer-to-peer connection can be established using the Session Description Protocol (SDP) but does not enforce a particular signaling solution. So MixedReality-WebRTC offers a built-in solution in the form of the NodeDssSignaler component, but also allows any other custom implementation to be used.

To conclude, WebRTC could be the main feature and improvement for future work so the application will have smaller video streaming delays and consequently to be achieved faster navigation at all levels.

8. CONCLUSION

In summary, the MR application that was developed is capable of remote controlling an unmanned ground vehicle with ROS operating system through the Hololens devices 1 and 2. The application is made for a more specific case where the existing software wasn't enough sufficient for the scenario’s needs. It is a basic controller with the functionalities of the robot’s navigation and Kinect camera’s video streaming. These functionalities are enough for the application to be usable. On this basis, they can add a lot of add-ons on both platforms Ros and Unity.

The Kafka server gives the MR application a nice abstraction and adaptiveness for other types of vehicles different from ROS. So it is a generic MR controller for Hololens that provides them the capability to control different devices that can listen to Kafka messages. Also, the video streaming can be from different sources that are on the Mjpeg protocol. The source of the video can also change through Kafka without making any configuration on the application’s code

To conclude, the results of the experiments showed that the application can be used in a real-life scenario even with some small delay in the video feed. This latency and some precision deviation in the robot’s position are dependent on the case of use and the type of unmanned vehicle that the user wants to navigate.

This thesis’s goal is to provide even a minor contribution to the ongoing research on having more practical applications of these technologies.

ABBREVIATIONS - ACRONYMS

IoT	Internet of Things
UxV	Unmanned Vehicle, x stands for aerial, ground, or sea
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
USV	Unmanned Surface Vehicle
ROS	Robot Operating System

REFERENCES

1. J. Guhl, S. Tung and J. Kruger, "Concept and architecture for programming industrial robots using augmented reality with mobile devices like microsoft HoloLens," 2017 22nd 8. IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2017, pp. 1-4, doi: 10.1109/ETFA.2017.8247749.

2. Hooman Hedayati, Michael Walker, and Daniel Szafir. 2018. Improving Collocated Robot Teleoperation with Augmented Reality. In Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction (HRI '18). Association for Computing Machinery, New York, NY, USA, 78–86.

3. Mårten Lager, Elin A. Topp, Remote Supervision of an Autonomous Surface Vehicle using Virtual Reality, IFAC-PapersOnLine, Volume 52, Issue 8, 2019, Pages 387-392, ISSN 2405-8963

4. M. Ostanin, R. Yagfarov, A. Klimchik, Interactive Robots Control Using Mixed Reality **The work presented in this paper was supported by the grant of Russian Science Foundation 17-19-01740., IFAC-PapersOnLine, Volume 52, Issue 13, 2019, Pages 695-700, ISSN 2405-8963

5. https://medium.com/swlh/apache-kafka-what-is-and-how-it-works-e176ab31fcd5

6. https://docs.microsoft.com/en-us/windows/mixed-reality/

7. https://github.com/microsoft/MixedRealityToolkit-Unity

8. https://webrtc.org/

9. Figure 14 - https://docs.microsoft.com/en-us/windows/mixed-reality/design/gaze-and-commit-head

10. Figure 15 - https://github.com/microsoft/MixedRealityToolkit-Unity

11. Maximilian Speicher, Brian D. Hall, and Michael Nebeling. 2019. What is Mixed Reality? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (*CHI '19*). Association for Computing Machinery, New York, NY, USA, Paper 537, 1–15. https://doi.org/10.1145/3290605.3300767

12. Figure 1 - https://www.uavos.com/uavos-fixed-wing-uav-sitaria-completed-flight-tests/

13. Figure 2 - https://www.business-standard.com/article/technology/drone-taxis-and-deliveries-science-fiction-gets-closer-to-reality-in-seoul-120111100920_1.html

14. Figure 3 - https://www.wired.com/story/hurricane-florence-underwater-drone-slocum-glider/

15. Figure 4 - https://www.naval-technology.com/projects/fleet-class-common-unmanned-surface-vessel-cusv/