ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

---

# Remote Driving System

---

**Team #17**

BO PANG (bopang5@illinois.edu)
JIAHAO WEI (jiahaow4@illinois.edu)
KANGYU ZHU (kangyuz2@illinois.edu)

TA: Yi Wang
Sponsor: Liangjing Yang

May 21, 2023

# Abstract

Traditional chauffeuring services provide convenient and secure transportation but have limitations and drawbacks. A remote driving system leveraging communication technology and mixed reality has been developed to address these challenges. This system allows chauffeurs to control vehicles remotely, eliminating the need for their physical presence. The system enhances human capabilities by accessing and operating in previously impractical or inaccessible areas while reducing risks associated with dangerous environments. Deploying human operators to remote locations becomes more cost-effective and time-efficient with this solution.

The remote driving system consists of four subsystems: HoloLens 2, Driving Control, Server, and TurtleBot3. The server acts as the central component, facilitating communication among the subsystems. The Driving Control subsystem handles control commands, while the HoloLens 2 subsystem presents real-time information to the user interface. The TurtleBot3 subsystem performs movements and captures environmental data.

By integrating remote driving systems, the limitations of traditional chauffeuring services can be overcome, enabling safer and more efficient transportation in challenging environments.



**Keywords**: remote driving system, mixed reality, communication technology

# Contents

# 1 Introduction

## 1.1 Purpose

While traditional chauffeuring service offers people a convenient and secure mode of transportation, it is not without its drawbacks. Under some circumstances, the potential dangers and risks typically encountered in dangerous or unsafe environments can put human chauffeurs at risk. In some areas, Humans are limited by their physical capabilities and have difficulty performing specific tasks without endangering themselves. In some cases, deploying human chauffeurs to distant or remote locations can be expensive and time-consuming, requiring significant resources.

Given the advancements in communication technology and mixed reality, it is now feasible to develop a remote driving system where chauffeurs can remotely control the owner's vehicle. This innovative solution eliminates the need for chauffeurs to be physically present. With such a remote driving system, humans can broaden their capabilities, gaining access to and effectively operating in areas otherwise considered impractical or unattainable. It also allows operators to control robots from a safe distance, effectively minimizing the potential dangers and risks associated with such environments. Also, in some cases, it offers a solution to significantly decrease the costs and time required for deploying human operators.

## 1.2 Functionality

We will substitute a TurtleBot3 (a programmable robot) for the automobile to reduce the complexity of the control subsystem. We want to achieve the following high-level project functionality:

- Under a chauffeur's operation, the TurtleBot3 should be able to navigate through the whole campus. This verifies the functionality of remote driving.
- The video delay from the TurtleBot3 should be at least 100 ms. The postponement of the chauffeur's commands should be within 50 ms. This ensures a smooth user experience for chauffeurs and users' safety.
- The server can handle at least two connections from different chauffeurs. All chauffeurs can view the status of the TurtleBot3, and the one who will drive the TurtleBot3 will be selected according to a policy. This enables chauffeurs to take turns during service, thus improving the duration and distance of chauffeuring services.

## 1.3 Subsystem Overview

The system comprises five distinct subsystems: HoloLens 2, Driving Control, Server, TurtleBot3, and Name Server. The server is the central component, receiving, processing, and transmitting information to facilitate interaction among the other three subsystems. Specifically, the Driving Control subsystem is primarily responsible for obtaining and sending control commands. The HoloLens 2 subsystem delivers real-time information to the user interface. The TurtleBot3 subsystem is equipped to execute movements based on instructions and capture real-time environmental data. To achieve a fast data transmission between Turtlebot3 and Server, direct network connections are established between them. This is done with the help of the Name Server subsystem.

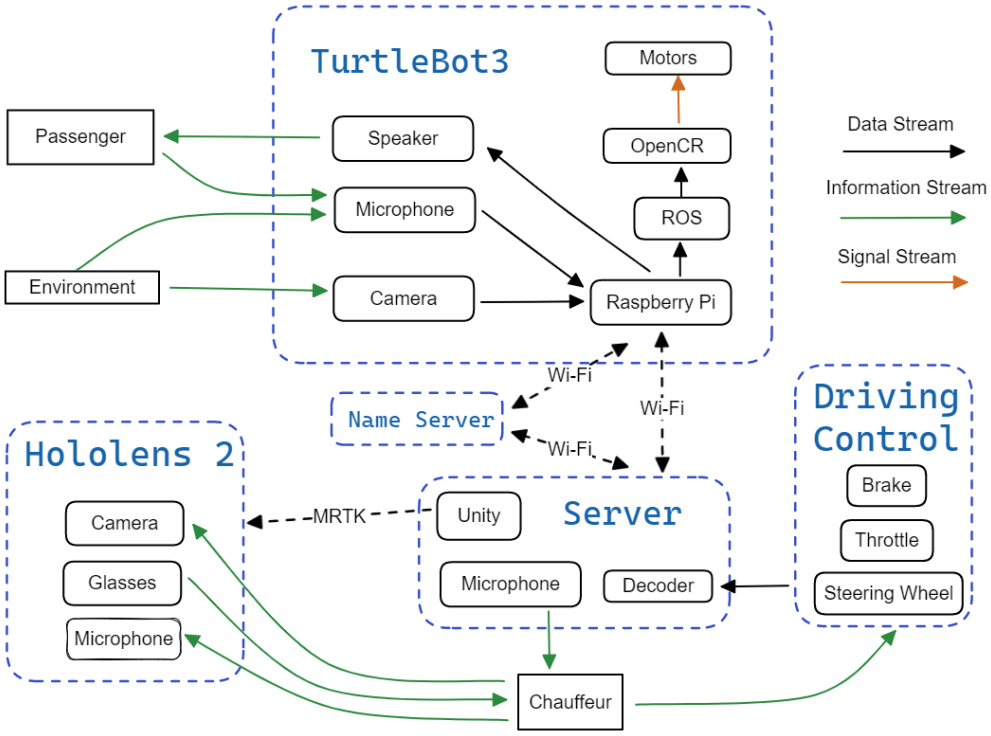


Figure 1: Block Diagram

# 2 Design

## 2.1 HoloLens 2

The HoloLens 2 system encompasses all the knowledge and insights that the driver can acquire. The Server is responsible for transmitting the video signal, which is then displayed on the HoloLens 2 device, enabling the driver to perceive the surrounding environment from various perspectives. The Unity platform facilitates the projection of real-time car information onto the HoloLens 2, serving as a reference for the driver. With HoloLens 2, it is crucial to ensure that the driver can observe a clear video stream captured by the remote TurtleBot3 camera, positioned virtually 1 meter from the driver. The driver's field of view in front should span approximately 120 degrees, extending from left to right. Moreover, they should have the ability to see additional angles by turning their head.

### 2.1.1 Glasses and Screens

The HoloLens 2 glasses function as a visual interface for experiencing mixed reality. They allow users to view and engage with virtual information superimposed onto the physical world, resulting in a blended reality environment. When using these glasses, it is essential to ensure that the driver has an unobstructed and steady field of view. Specifically, the screen should be positioned 1 meter from the driver, and the immersive virtual interface should be situated directly in front of them. For the screens in HoloLens 2, due to the wide angle of our cameras (the front-facing camera has a 120° field of view), displaying the feed on a flat plane in HoloLens 2 would appear somewhat unusual. To address this, we have designed a slight curvature for the screens displayed in HoloLens 2. By adjusting the screens to a suitable curvature, the field of view appears more realistic, and the driver can also see a wider range of angles by turning their head. Additionally, we have incorporated important information onto the screens, such as the car's speed and battery level. This information helps the driver to have a better understanding of the car's motion status.

### 2.1.2 Microphone and Speech Recognition

The HoloLens 2 device is equipped with three front-facing environmental audio capture microphones and two user voice capture microphones. These microphones can capture both ambient sounds and the voice commands issued by the user. Additionally, the HoloLens 2 includes an embedded speech recognition library, which can capture and rec-

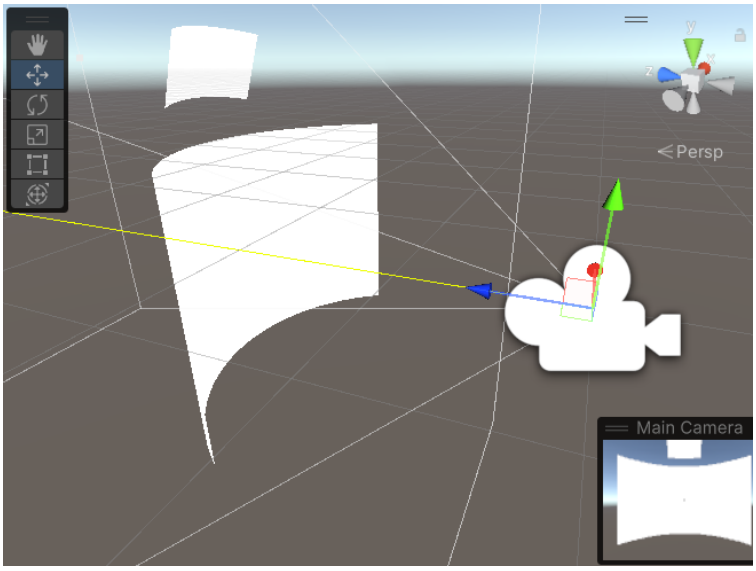


Figure 2: Two screens designed in Unity3D

ognize the user's vocal input, allowing for subsequent actions based on the recognized commands. The functionality of speech recognition is crucial for the remote driving system. When the driver needs to interact with the HoloLens 2 environment while driving, using gesture-based interactions poses certain safety risks. However, by employing voice interactions, the safety of the system can be significantly improved. To address this, we have designed a series of commands such as "reset camera", "play music" and "mode conversion", enabling drivers to interact with the HoloLens 2 device safely.

### 2.1.3 Camera and Position Adjustment

HoloLens 2 has the capability to display real-time footage captured by the TurtleBot3 car's cameras. The car is equipped with both front-facing and rear-facing cameras. The main screen of the HoloLens 2 can show the feed from the front-facing camera, while the feed from the rear-facing camera is displayed on top of the main screen. However, the positions of the two screens in the HoloLens 2 may not align with the driver's direct line of sight during driving. This is because the camera position of the HoloLens 2 may not necessarily be in front of the driver when establishing the connection between the HoloLens 2 and the PC server. Additionally, the driver's seating position may vary. To ensure that the screens in the HoloLens 2 are positioned in front of the driver during driving, we have implemented a "reset camera" method. Through the speech recognition functionality of the HoloLens 2, users can issue the "reset camera" command, which automatically aligns the positions of the two screens with the forward-facing perspective of the HoloLens 2

camera.

For the design detail, when we want to adjust the position and orientation of an object based on the current camera position of HoloLens 2, we need to calculate the adjusted position and rotation states.

For the position state, it can be calculated using the following approach: Assume the camera position vector $\mathbf{p}_c$, camera forward vector $\mathbf{f}_c$, and camera up vector $\mathbf{u}_c$ are represented as:

$$\mathbf{p}_c = \begin{bmatrix} p_{cx} \\ p_{cy} \\ p_{cz} \end{bmatrix}, \quad \mathbf{f}_c = \begin{bmatrix} f_{cx} \\ f_{cy} \\ f_{cz} \end{bmatrix}, \quad \mathbf{u}_c = \begin{bmatrix} u_{cx} \\ u_{cy} \\ u_{cz} \end{bmatrix}$$

We can combine the camera position and the camera forward and up vectors into a matrix:

$$M = \begin{bmatrix} \mathbf{p}_c & \mathbf{f}_c & \mathbf{u}_c \end{bmatrix} = \begin{bmatrix} p_{cx} & f_{cx} & u_{cx} \\ p_{cy} & f_{cy} & u_{cy} \\ p_{cz} & f_{cz} & u_{cz} \end{bmatrix}$$

Then, the target position vector $\mathbf{p}_t$ can be represented as a homogeneous coordinate vector, where the variable 'scale' represents the scaling factor and the variable 'offset' represents the offset. Thus, the target position vector $\mathbf{p}_t$ can be calculated as:

$$\mathbf{p}_t = M \cdot \begin{bmatrix} \text{scale} \\ \text{offset} \\ 0.0 \end{bmatrix}$$

Perform matrix multiplication to obtain the reset position coordinates of the target ob-

ject:

$$\mathbf{p}_t = \begin{bmatrix} p_{tx} \\ p_{ty} \\ p_{tz} \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} \text{scale} \\ \text{offset} \\ 0.0 \end{bmatrix} = \begin{bmatrix} p_{cx} & f_{cx} & u_{cx} \\ p_{cy} & f_{cy} & u_{cy} \\ p_{cz} & f_{cz} & u_{cz} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \text{scale} \\ \text{offset} \\ 0.0 \end{bmatrix}$$

For the rotation state, assume the camera rotation matrix is denoted by $R_c$, the downward rotation angle is denoted by $\theta_x$, and the rotation matrix for the target object is denoted by $R_t$. First, we calculate the downward rotation matrix $R_{\text{lookDown}}$, which corresponds to the rotation angle $\theta_x$, and can be represented as:

$$R_{\text{lookDown}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix}$$

Next, we multiply the camera rotation matrix $R_c$ with the downward rotation matrix $R_{\text{lookDown}}$ to obtain the rotation matrix for the target object $R_t$:

$$R_t = R_c \cdot R_{\text{lookDown}}$$

### 2.1.4 Videos and Face Detection

The video information captured by the remote car's camera can be directly transmitted to the HoloLens 2 or undergo certain processing, such as implementing face detection on the video information. This auxiliary method can enhance safety as it enables the system to identify and monitor the presence of pedestrians, cyclists, or other individuals near the vehicle. Therefore, we implemented a network that could perform rapid face detection. When a human face is detected in the video, it can be visually indicated by enclosing it with a red bounding box. Consequently, this can assist the driver in making better judgments regarding the presence of pedestrians in their field of view, thereby enhancing the safety of the system. In addition to that, depending on the quality of the network environment, the driver can choose to activate or deactivate face detection through voice control. Through testing, it has been observed that in a well-functioning network environment, incorporating face detection operations ensures low latency in video transmission.

Specifically, we utilized the Haar cascade in OpenCV for detecting faces. The Haar cascade algorithm is widely recognized as the most popular method for object detection in OpenCV. Its primary advantage lies in its high-speed performance. Haar cascades prove to be highly efficient[1], especially when deployed on resource-limited devices where more resource-intensive object detectors are not feasible. The following diagram illustrates the specific process of detection:

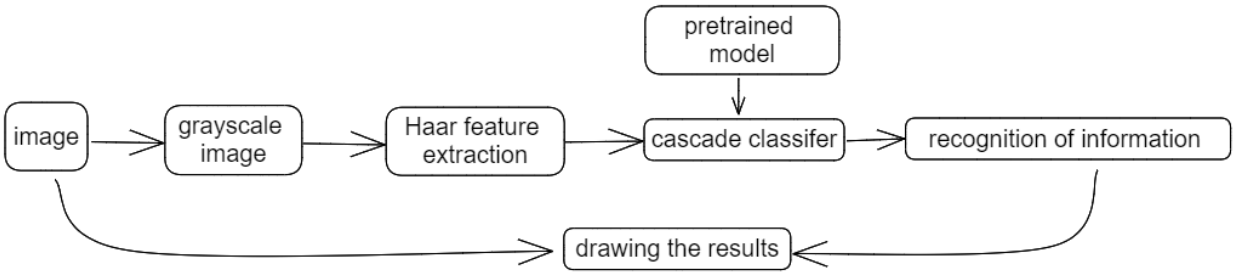


Figure 3: Haar cascade algorithm detection procedure

## 2.2 Server

The Server Subsystem is crucial in facilitating information exchange as a pivotal link for signal control. Its responsibilities encompass decoding the video information transmitted by the TurtleBot3 and displaying it on the HoloLens 2. Additionally, the Server Subsystem is responsible for transmitting signals from the Driving Control Subsystem to the TurtleBot3, enabling effective control over the vehicle.

### 2.2.1 Unity

Unity is a versatile game engine and development environment that finds extensive application in developing augmented reality (AR) applications. In this project, it is imperative to establish a connection between Unity and HoloLens through the MRTK package. Simultaneously, Unity must receive real-time information from the server, encompassing data from the vehicle, video feed, and other pertinent information.

There are two connection methods for connecting HoloLens 2 to a PC server. One is through a Wi-Fi connection, and the other connects the HoloLens 2 to the PC via a USB interface. In the second method, the PC establishes a connection by obtaining the HoloLens 2's IP address on the local network. We have utilized the second connection method because it is more stable and has lower latency.

### 2.2.2 Decoder and Motion Signal Transmission

When the driver manipulates the steering wheel, accelerator, and brake in the driving control subsystem, the motion signals are transmitted via USB to the PC server. The PC server needs to receive these signals in real-time and convert them accordingly. Subsequently, the converted signals are sent to the remote Raspberry Pi on the car via the TCP protocol.

For the specific conversion process:

$$l = \frac{\text{float}(\text{throttle} - \text{brake})}{2} \times \text{MAX\_LIN\_VEL}$$

The $throttle$ and $brake$ reflect the braking and acceleration states in the driving control, and their values range from $[-1, 1]$. By subtracting them and mapping the result to the velocity range that turtlebot3 can achieve, we can control the forward and backward movements of the car.

$$a = \frac{\text{float}(\text{wheel})}{1} \times \text{MAX\_ANG\_VEL} \times (-1)$$

The $wheel$ represents the rotation value of the steering wheel, which also ranges from $[-1, 1]$. We can steer the car by mapping the angular velocity to the angle range that turtlebot3 can achieve.

By sending $l$ and $a$ to the remote car, we can achieve remote control of the car through the driving control.

Name Server is designed to handle three requests, namely `JOIN`, `LEAVE`, and `GET`. Sending request `JOIN hostname` to Name Server will register the hostname and sender's IP in its record. Sending request `LEAVE hostname` will delete the record of `hostname`. By sending request `GET hostname`, the sender will receive the IP address of the specified `hostname` from Name Server. If `hostname` does not exist in Name Server's record, an empty `string` will be returned.

## 2.3 Driving Control

### 2.3.1 Mechanical Part

We used pedals and steering wheel as the interface between automobiles and drivers to provide a direct and accurate control mechanism. The steering wheel is designed with a

larger diameter than other possible controls, such as levers or buttons, which allows the driver to exert greater force and have better control over the steering mechanism. Also, the circular shape of the steering wheel allows for a full range of motion, enabling the driver to turn the wheels smoothly and progressively. This makes it easier to negotiate turns, navigate tight spaces, and maintain vehicle control. Besides, the steering wheel is positioned in a comfortable and intuitive way for the drivers. It is located within easy reach and allows for a natural hand placement, giving the driver a stable grip for precise control.

The placement and design of pedals and the steering wheel are typically standardized across vehicles, making them familiar and intuitive for drivers. This consistent layout allows drivers to adapt quickly to our control system. It also simplifies the learning process for new drivers and facilitates safe and efficient operation.

Figure 4 shows the driving Control Subsystem. The steering wheel enables the driver to control the direction of the Turtlebot3. The pedals let the driver control the forward and backward speed of the Turtlebot3.

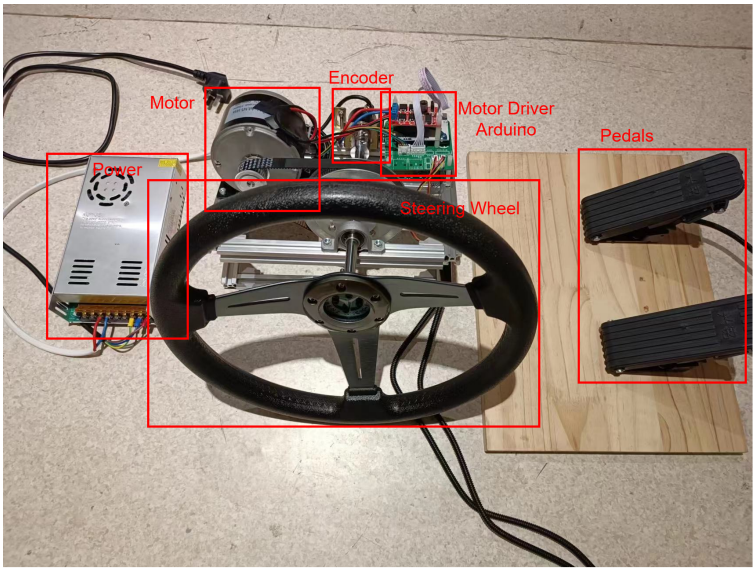


Figure 4: Driving Control

The encoder captures the rotation angle of the steering wheel. The two pedals will seize the degree to which they are depressed. Arduino will then get and process these signals and send them to Server. As a result, Server can get the control commands from the driver and send them to Turtlebot3 to control its motion.

A motor with power is used to simulate the mechanism of a steering wheel. This motor is

controlled by the signals sent from Arduino. In this way, the motor can be programmed to restrict the rotation angle of the wheel. Specifically, if the driver rotates the wheel to an angle greater than 540 degrees, the motor will apply a force in the opposite direction. Thus, the driver cannot turn it further. Also, the motor can apply force during rotation to simulate the rotation-related properties such as friction, i.e., the amount of force the driver needs to apply to rotate the wheel.

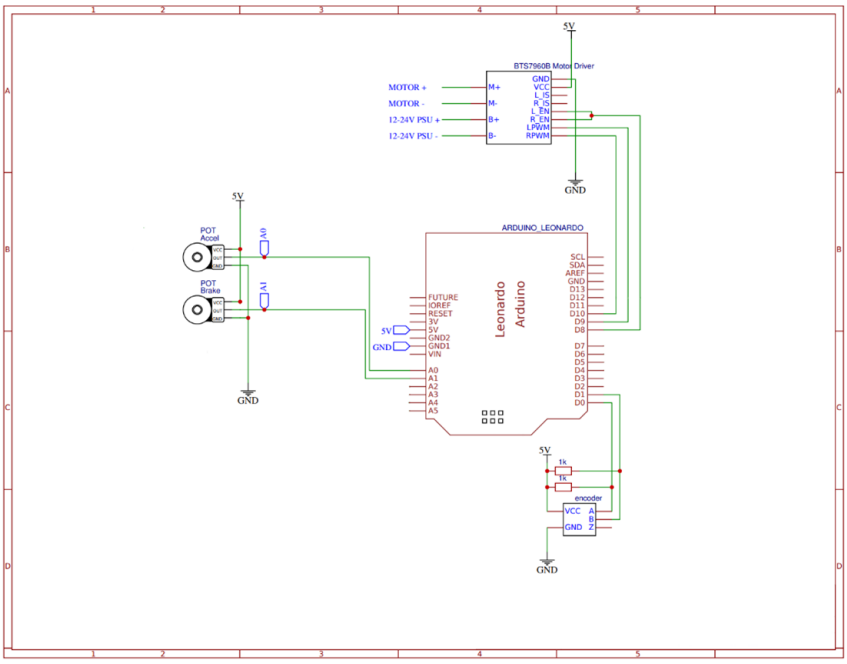


Figure 5: PCB Design

## 2.4 Turtlebot3

### 2.4.1 Motions and Status

Turtlebot3 has ROS preinstalled. ROS (Robot Operating System) provides standard operating system services such as hardware abstraction, device drivers, implementation of commonly used features including sensing, recognizing, mapping, motion planning, message passing between processes, package management, visualizers, and libraries for development as well as debugging tools [2].

In ROS, the master acts as a name server for node-to-node connections and message communication. A node refers to the smallest unit of the processor running in ROS. A message is a bundle of data used to exchange data between nodes. A package is the basic

unit of ROS. The ROS application is developed on a package basis, and the package contains either a configuration file to launch other packages or nodes. The topic is literally like a topic in a conversation. The publisher node first registers its topic with the master and then starts publishing messages on a topic. Subscriber nodes that want to receive the topic request information from the publisher node corresponding to the topic's name registered in the master. Based on this information, the subscriber node directly connects to the publisher node to exchange messages as a topic.

The `serial` package is used to utilize motors and simplify development. As shown in Figure 6, the `rosserial` is a package that converts ROS messages, topics, and services to be used in serial communication [3]. It achieves message communication between a microcontroller and a computer using ROS. The node `serial_python` in package `rosserial_python` will subscribe to the messages in topic `cmd_vel` and can send control signals to the OpenCR board, which will then control the motors. Thus, We can publish messages on this topic to control the motion of Turtlebot3. To get the status of the battery and speed of Turtlebot3, we can subscribe to topic `battery_state` and `joint_states`, respectively.

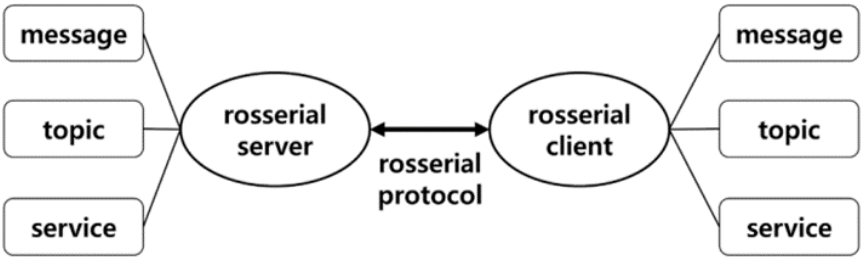


Figure 6: Rosserial Server (for PC) and Client (for Embedded System)

Figure 7 shows the correlation between active nodes and messages transmitted on the ROS network.

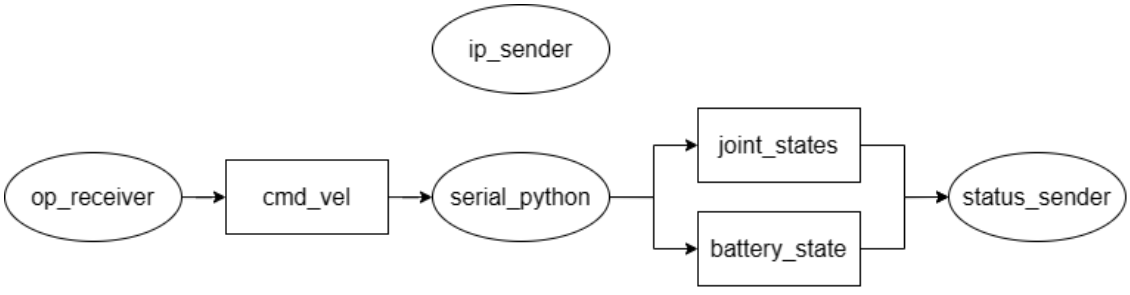


Figure 7: Graphical Representation of Message Communication

There are four nodes running in ROS, namely `serial_python`, `ip_sender`, `op_receiver`, and `status_sender`. Node `op_receiver` will receive control commands from Server and publish messages to topic `cmd_vel`. Node `serial_python` will subscribe to messages published in topic `cmd_vel` and communicate these messages (commands) with the OpenCR board, which will, in turn, control the motion of Turtlebot3. The OpenCR board will also send battery status and speed information to node `serial_python`. After processing them, `serial_python` will publish these messages in topic `joint_states` and `battery_state`. By subscribing to these two topics, node `status_sender` can get the status of Turtlebot3 and send them to Server. These messages will then be shown to the driver.

There is a node called `ip_sender`. This node does not interact with other nodes. It periodically sends requests to Name Server to update its IP address. In this way, Server can fetch the correct IP address of Turtlebot3 from Name Server.

## 2.5 Name Server

Direct network connections are established between the Server subsystem and Turtlebot3 to achieve fast data transmission. To resolve their IP addresses, the Name Server subsystem is needed. This system records the mapping from host names to their IP addresses and handles requests from other hosts. As the IP addresses of Turtlebot3 and Server Subsystem are subjected to DHCP (Dynamic Host Configuration Protocol), they will be changed periodically. With the help of the Name Server Subsystem, they can update their current IP addresses and get others' correct IP addresses. Thus, direct network connections can be established.

Concerning transport protocol, UDP (User Datagram Protocol) is used instead of TCP (Transmission Control Protocol). First, UDP has a smaller header size than TCP, resulting in lower packet overhead. The requests and responses are small in size, so using UDP helps reduce the overall network traffic and latency. Second, UDP is a connectionless protocol, meaning it doesn't establish a dedicated connection before sending data. This lack of connection setup and teardown processes makes UDP faster. Also, this name service operates statelessly, where each request and response is independent of others. UDP's stateless nature suits the requirements of this service's communication. Finally, TCP has built-in congestion control mechanisms to prevent network congestion and ensure reliable data delivery. Using UDP allows this service to bypass TCP's congestion control mechanisms, which can lead to faster response times and less network congestion.

## 2.6 Video Transmission

For the video transmission system, my task was to ensure that the video had low latency and high resolution to meet the project's requirements. We developed my own modules to handle video transmission, using the Socket framework for sending information and Opencv for encoding and decoding the image data.

Firstly, after clarifying the goals of video transmission, we decided to adopt a layered approach to implement the code. That is, we utilized the object-oriented features of C++ to achieve code reusability. This allowed us to build my functionality more steadily. We built four primary classes to do the video encoder and transmission.

### 2.6.1 Socket Connection Class

In this video transmission implementation, the difficulty level for socket transmission is relatively low, as the most basic UDP connection was used to send' vector¡unsigned char¿' data. As the implementation process is not particularly complicated, we will not provide a detailed explanation here.

### 2.6.2 VideoFrame Class

To begin with, we implemented the VideoFrame class to retrieve images. The images are represented using two data structures, namely cv::Mat and vector¡char¿. These structures can be interconverted between each other. The constructor of the VideoFrame class, "explicit VideoFrame($conststd :: vector < unsignedchar > frame_bytes$)", uses cv::imdecode to convert vector¡char¿ to cv::Mat. Additionally, the "$vector < unsignedchar > GetJPEG()$" function converts the cv::Mat member variable of the class to vector¡char¿.

### 2.6.3 VideoCapture Class

If in VideoFrame, we can open an image, save its data in an object, and convert it into a different data type. The next step is to get real-time information from each frame in the camera. VideoCapture's job is to get data from the camera for each frame and convert it into a usable data type. Therefore, VideoFrame is used in VideoCapture to return usable image data. We use the API cv::VideoCapture(0) to get the camera's image information, save it as a cv::VideoCapture type data, convert it to a cv::Mat data type, and then call the VideoFrame constructor.

### 2.6.4 BasicProtocol Class

After collecting video data, the next step is to use the $Basic_Protocol$ class to easily convert the collected data into the required data type for sockets. In UDP, the data type used for transmission is $vector < unsignedchar >$. Once you have this data type, you can use sockets for communication.

### 2.6.5 Working Principle

We also implemented stitching of cv::Mat images through functions for both horizontal and vertical stitching. This was implemented after We achieved multi-threaded image transmission, which can improve the video resolution.

After implementing basic video transmission functionality, you made improvements to allow Turtlebot to simultaneously transmit video to multiple clients. To achieve this, We developed a server for video transmission.

The server uses TCP for client connections, ensuring reliable data delivery. Video transmission is done using UDP, which is suitable for real-time streaming. To handle multiple clients efficiently, We implemented a thread pool. Each client connection is assigned two video-transmission threads, allowing concurrent processing.

By combining TCP for connection establishment, UDP for video transmission, and a thread pool implementation, Turtlebot can effectively share video with multiple clients simultaneously. This setup allows our server to transmit video information to multiple drivers and utilize different video compression rates. This allows for monitoring the environment's safety, and our drivers can transfer control of the Turtlebot, making driving safer.

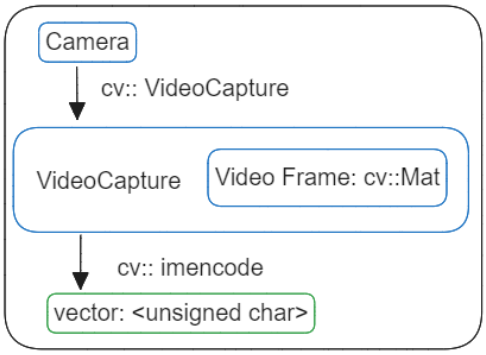


Figure 8: Top level of video transformation

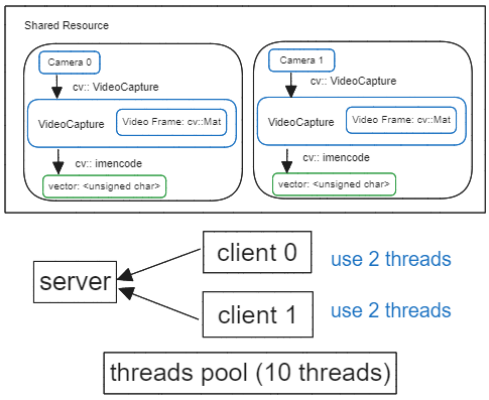


Figure 9: Top level of video transmission

## 2.7 Audio Transmission

To enable real-time communication between drivers and passengers, We implemented audio transmission. The server accepts connections from multiple clients, each providing audio data. The server broadcasts the audio data to other clients using multiple threads, facilitating audio transmission. We utilized the PyAudio library for audio encoding and decoding. In Linux, external devices are treated as files, so the playback process involves writing the data to the speaker. This completes the audio transmission process.

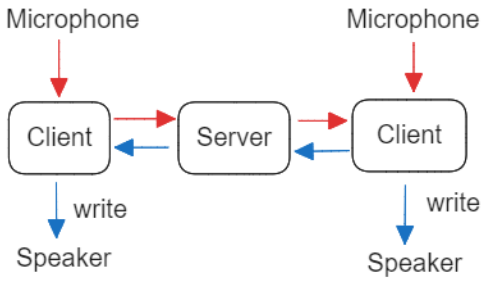


Figure 10: Top level of audio transmission

# 3 Verification

## 3.1 HoloLens 2

### 3.1.1 Screens in HoloLens 2

One requirement is that the screen showing the remote view should be positioned at an optimal distance from the driver to ensure clear visibility of the content. The recommended distance is approximately one meter. To verify that, one way is to get a person to stand one meter away from the driver and see whether the person and the screen overlapped from the driver's view with HoloLens 2. Another way is to check the relative position of the "main camera" and "curverdCam" in Unity:

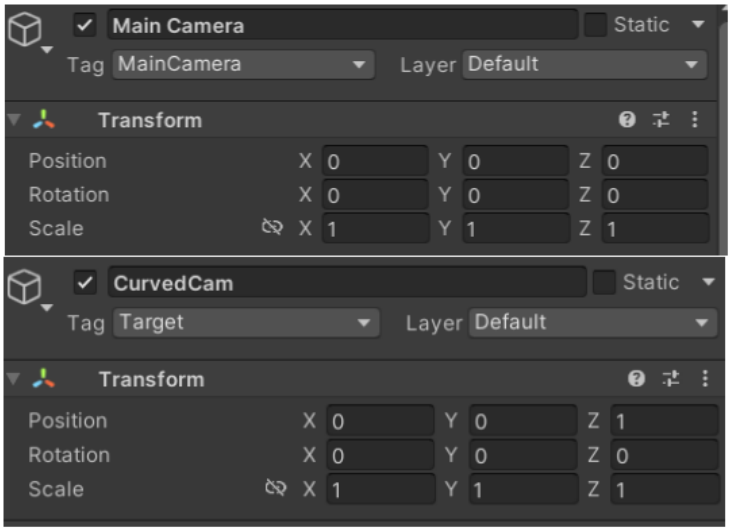


Figure 11: Relative position of two objects in Unity

Another requirement is that the driver needs to have a 120° field of view in the forward direction:

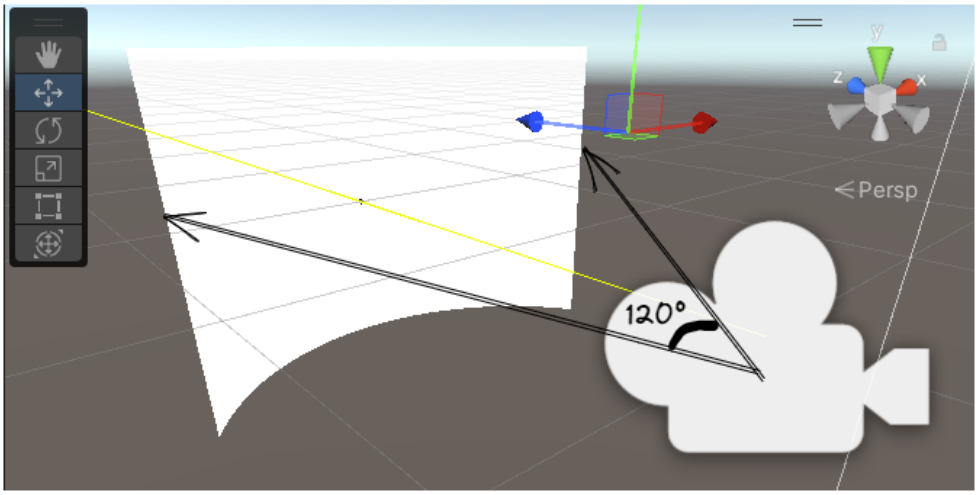


Figure 12: The driver's field of view.

To meet this requirement, two students stood at opposite ends of the screen according to the driver's instructions. By calculating the angle between them, we adjusted the position and curvature of the screen in the HoloLens 2 to meet the requirement.

### 3.1.2 Speech Recognition and Face Detection

To test the effectiveness of speech recognition, we repeated various commands (e.g., "reset camera," "play music," "stop music," "detect mode," and "normal mode") to see if the HoloLens 2 could perform the corresponding actions. We also increased the environmental noise to 70dB, proving that HoloLens 2 could still distinguish our commands in such a noisy environment.

To test the effectiveness of face recognition, we first activated the face detection mode using a voice command. Then, we controlled multiple variables, such as having one or multiple individuals appear at different distances within the field of view of the car's camera. The results showed that the face detection performed well as the red bounding box was drawn on each person's face.

## 3.2 Driving Control

module `joystick` in `pygame` is used to decode the output signals from the Driving Control subsystem. Figure **??** shows the output from `python` script `teleop`. This script can interpret different kinds of controllers, decode their outputs, and send them to Turtlebot3.

As regards the Driving Control subsystem, the steering wheel and two pedals are interpreted as axes in a joystick. By rotating the steering wheel and depressing the pedals, the values of the axes will change accordingly. This indicates that Driving Control is running correctly.

## 3.3 Turtlebot3

### 3.3.1 Motion

Node `op_receiver` will accept connections from other servers and receive their operation commands. It will then parse these commands and publish them in topic `cmd_vel`.

To verify that node `op_receiver` has successfully received the driver's commands and uses them to control the motors, command `rostopic -n one echo cmd_vel` is used. This command will get one published message in topic `cmd_vel` and print them in the terminal. As shown in Figure 13, node `op_receiver` had received commands and published them to topic `cmd_vel` in the form of linear and angular velocity. Another command `rostopic info` shows the topic information `cmd_vel`. It shows that node `op_receiver` is the only publisher, and the message published above must be from node `op_receiver`.

```
pi@raspberrypi:~/catkin_ws/devel $ rostopic echo -n 1 cmd_vel
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
pi@raspberrypi:~/catkin_ws/devel $ rostopic info cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /op_receiver (http://localhost:40821/)

Subscribers:
 * /turtlebot3_core (http://localhost:43947/)
```

Figure 13: op_receiver Test

### 3.3.2 Status

Node `status_sender` will periodically fetch the IP addresses of other hosts and send them the status of Turtlebot3. It uses receivers' port `8082` and `8083` to send battery

status and current speed, respectively. By registering our machine in Name Server and then listening on a pre-configured port, we can receive the current status of Turtlebot3 sent from it.

Figure 14 examines the functionality of node `status_sender`. A machine joins Name Server with hostname `Dell`. A socket in this machine is created and bound to port `8082`. Then, it receives the battery voltage (12.1100) sent from Turtlebot3, as shown in the figure.

```
>>> s = socket(AF_INET, SOCK_DGRAM)
>>> s.bind(('10.107.135.205', 8082))
>>> addr = '10.105.100.215', 8080
>>> s.sendto('JOIN Dell'.encode(), addr)
9
>>> s.recvfrom(128)
(b'12.1100', ('10.107.7.214', 33849))
```

Figure 14: status_sender Test

## 3.4 Name Server

Figure 15 demonstrates a test on Name Server Subsystem. The Name Server's IP address is `10.105.100.215`, and the port it used is `8080`. We first registered the IP address with hostname `Turtlebot3`. The IP address of the local host will be used to register. After sending `GET Turtlebot3`, we received the machine's IP address running this test. As expected, we tried to resolve the IP address of `Turtlebot2`, and we got an empty string. Then, we unregistered `Turtlebot3`. A request acquiring the IP address of `Turtlebot3` will return an empty string. This test verified that the objectives of Name Server were achieved.

## 3.5 Video transmission

Based on extensive testing and validation, our server has demonstrated the capability to effectively transmit video data to multiple clients, supporting a maximum of five clients concurrently. During our impressive demo, we showcased the seamless integration of the video transmission feature. One client's video feed was streamed and displayed on the immersive Hololens2 device, while another client's video feed was presented on a conventional screen.

The implementation yielded exceptional results, as all connected clients received clear

```
>>> addr = '10.105.100.215', 8080
>>> s.sendto('JOIN Turtlebot3'.encode(), addr)
15
>>> s.sendto('GET Turtlebot3'.encode(), addr)
14
>>> s.recvfrom(128)
(b'10.107.135.205', ('10.105.100.215', 8080))
>>> s.sendto('GET Turtlebot2'.encode(), addr)
14
>>> s.recvfrom(128)
(b'', ('10.105.100.215', 8080))
>>> s.sendto('LEAVE Turtlebot3'.encode(), addr)
16
>>> s.sendto('GET Turtlebot3'.encode(), addr)
14
>>> s.recvfrom(128)
(b'', ('10.105.100.215', 8080))
```

Figure 15: Name Server Test

and uninterrupted video streams. This successful transmission of high-quality video data serves as a testament to the robustness and efficiency of our video transmission functionality.

By enabling real-time video communication between Turtlebot and multiple clients, we have opened possibilities for various applications. For instance, the Hololens2 display empowers one client to experience an augmented reality view, enhancing situational awareness, while another client can monitor the video feed on a standard screen. The flexibility to accommodate diverse client setups demonstrates the versatility and scalability of our solution.

## 3.6 Audio transmission

We can confidently assert that we have successfully implemented audio transmission functionality through our validation and demonstration. Our system achieves precise and reliable audio transmission, enabling real-time conversations between drivers and passengers, enhancing the overall experience with greater flexibility.

# 4 Cost and Schedule

## 4.1 Cost Analysis

- Labor
  Assume salary per hour is ¥45. Then the total labor cost is $45 \times 3 \times 85 = 11475$ ¥.
- Bill of Materials

| Name | Description | Price | Qty | Total |
|---|---|---|---|---|
| Pi Camera | wide-angle camera | ¥40 | 2-3 | ¥100 |
| Speaker | Normal Speaker | ¥15 | 1 | ¥15 |
| Microphone | Normal microphone | ¥9 | 1 | ¥9 |
| Power supply | 24V20A500W | ¥88 | 1 | ¥88 |
| Belt | HTD3M480*15MM | ¥13 | 1 | ¥13 |
| Motor | MY1016 24V300W | ¥65 | 1 | ¥65 |
| Development Board | Leonardo_R3 | ¥26 | 1 | ¥26 |
| Steering wheel | | ¥68 | 1 | ¥68 |
| DC motor drive board | 4f0W DC motor drive board | ¥52 | 1 | ¥52 |
| Pedals | Normal Pedals | ¥18 | 2-3 | ¥45 |
| Base support | Just normal wood and iron | N/A | 1 | N/A |
| Screws and nuts | Just normal screws and nuts | N/A | 1 | N/A |
| Total | | | | ¥481 |

- Total
  The total cost is 11956 ¥.

## 4.2 Schedule

| Week | Bo Pang | Jiahao Wei | Kangyu Zhu |
|---|---|---|---|
| 3/27 | Learn OpenCV programming under the C++ framework. Choose appropriate encoding and decoding methods. | Program OpenCR to manipulate the motion of TurtleBot3. | Establish a connection between Unity and HoloLens. |
| 4/3 | Roughly completed the framework for UDP video transmission. Build a PCB architecture based on the working principle and complete component procurement. | Understand the working principle of the steering control system and main control solutions. | Complete the video streaming from the server into Unity for rendering. |
| 4/10 | Complete the framework for the UDP video transmission function. | Brainstorm the working plan for the driving system. | Implement video stitching for multiple camera images. |
| 4/17 | Understand the working principle of firmware and flash the firmware. Print PCB and solder components. | Understand the working principle of the driving system, and attempt to program to drive the analysis of the steering wheel motion signal. | Optimize the display of remote videos in Unity, including features such as following movement of head rotation. |
| 4/24 | Install cameras in TurtleBot3 and make Raspberry Pi collect video signals. | Understanding how firmware works and burning firmware. | 1. Transmit information such as battery level and speed of the turlebot to Unity.<br>2. Model AR components in Unity and display real-time information such as battery level and speed. |

| Week | Bo Pang | Jiahao Wei | Kangyu Zhu |
|---|---|---|---|
| 5/1 | Set up Raspberry Pi and establish a connection between Raspberry Pi and the server. | Complete the driving programming and transmit the control signal in real-time. | Collect external audio using a microphone on the car and transmit it to the server. |
| 5/8 | Conduct testing. | Conduct testing. | Transmit the microphone signal from the server to the car radio. |

# 5 Conclusion

## 5.1 Accomplishments

We met all the high-level requirements in our final design and prototype device. We have successfully implemented all subsystems and confirmed their accuracy. Some notable achievements are outlined below:

- The server can handle at least two connections from different chauffeurs. All chauffeurs should be able to monitor the status of the TurtleBot3, and the selection of the chauffeur who will drive the TurtleBot3 will be determined based on a predetermined policy.
- The delay of the video sent back from the TurtleBot3 is within 100 ms. The postponement of the chauffeur's commands is within 50 ms.
- The Turtlebot 3 can operate effectively on the campus with the chauffeur located nearly 500 meters away. The chauffeur can successfully control the vehicle with low latency.

## 5.2 Ethical Considerations:

We continuously improve the stability, responsiveness, and safety of the remote driving system based on the feedback we receive from testing. This adherence to the IEEE Code of Ethics 6 "to maintain and improve our technical competence and to undertake technological Tasks for others only if qualified by training or experience, or after full disclosure Of pertinent limitations"[4].

We have also carefully considered other potential safety issues. In the remote driving system, drivers wear virtual reality devices that differ in size and weight from everyday devices. These devices can cause distractions and fatigue, with prolonged use leading to dizziness and compromising safety. Such issues may violate the IEEE Code of Ethics principles, "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices" in IEEE Code of Ethics[4]. Therefore, we recommend rigorous screening and training for drivers before system operation. Adequate rest after extended driving periods is also crucial. By prioritizing these measures, we minimize risks and enhance system performance.

In addition, our system utilizes cameras to capture real-time driving situations and transmit them to a remote driver. The remote driver can interact with the vehicle using virtual

reality technology, ensuring passenger safety. However, this technology presents challenges. According to the ACM code of ethics, "the emergent properties of systems should be carefully analyzed[5]." In our system, the transmission of live video feeds may suffer from instability and delays, posing safety risks. Weak signal areas may require reducing video resolution, limiting the remote driver's ability to make informed decisions. To address these issues, we must enhance camera hardware for improved resolution and optimize the transmission network. Our project proposes using 5.0 GHz channels, employing JPEG compression for video data, and UDP for reliable and timely transmission. These improvements enhance effectiveness and safety, providing passengers with a seamless and secure driving experience.

## 5.3 Future Work

Although we have successfully met all the high-level requirements, there are still opportunities to optimize the system for a more immersive driving experience and enhanced safety in remote operations. The following areas can be further improved:

- **Algorithm Design**: Intelligent automatic resolution adjustment can be implemented in different network environments to reduce latency. Increasing the number of servers can allow for automatic allocation of the central server controlling the car to minimize delays as the vehicle reaches different locations, enabling seamless driver transitions.
- **Self-driving and Cruise Control**: On road segments with low traffic and high safety factors, the car can engage cruise control and automated driving functions, which may further enhance the driving experience and improve overall system safety.
- **Automatic switch between Wi-Fi and cellular network**: When the Wi-Fi signal is weak, or the congestion level of networking is high, the server in the car can automatically switch to a cellular network to maintain a good network connection.

By implementing these optimizations, we aim to create a driving system that is not only more immersive but also prioritizes user safety.

# References


[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[2] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim, *ROS Robot Programming*.

[3] Ken Conley, Dirk Thomas, Jacob Perron. "rospy." (), [Online]. Available: http://wiki.ros.org/rospy.

[4] IEEE. "IEEE Code of Ethics." (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html.

[5] ACM. "ACM Code of Ethics and Professional Conduct." (2018), [Online]. Available: https://www.acm.org/code-of-ethics.

# Appendix A Requirement and Verification Table

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. Retrieve video information from the server. | 1. Check the display on HoloLens2. If the displayed image on HoloLens2 is the same as the simulated image in Unity, and the simulated functions, such as clicks, can be performed in HoloLens2, then the verification is successful. | Y |
| 2. Retrieve data information from the Server sent by the distant car (e.g., car battery, speed). | 2. Check whether Unity can display the real-time image transmitted from the remote car camera, focusing on frame rate and latency. The verification is successful if there is no disconnection or the frame is dropping for a long time. | Y |
| 3.HoloLens2 can display a stable immersive virtual interface (e.g., dashboard, driving environment) in front of and below the user (at a close distance). | 3. Continuously change the running status of the car, and check whether the data information about the car in Unity changes dynamically accordingly. | Y |
| 4. Using natural language, The driver could reset the screen's position in HoloLens 2. | 4. Use the command "reset camera" to test whether the camera can be reset. | Y |

| | | |
|---|---|---|
| 5. The screen displaying the remote view needs to be at an appropriate distance from the driver so that they can see the content. The distance is set to about one meter, and the alignment of the screen displaying the remote view can be reset when the driver changes position. | 5. When the driver is driving or slightly moving, check whether the remote view screen is fixed. Verify the distance between the screen and the driver by first recording the driver's current position, then having the driver pass through the screen, and finally measuring the distance between the two recorded positions to see if it is 1 m. | Y |
| 6. It can provide drivers with a stable view of the remote driving environment (captured by the car camera), with the interface displayed in a curved shape, surrounding the driver's field of view (approximately 124°). | 6. Ask the driver to look straight ahead and have someone stand about 120 degrees off the driver's front, ensuring that the driver can see the scene from this angle on the screen. | Y |
| 7. The driver could enable music to get a more immersive experience with commands in natural language. | 7. Use the command ″play music″ and ″stop music″ to see whether the music function is okay. | Y |
| 8. The driver could enable human face detection mode with a command in natural language so that the box will be shown to detect human faces in the main camera screen of the Turtlebot 3. | 8. Use the command ″detect mode″ to see whether there is a box on the human face when the main camera on Turtlebot 3 captures a human. | Y |

| | | |
|---|---|---|
| 9. The driver could enable human face detection mode with a command in natural language so that the box will be shown to detect human faces in the main camera screen of the Turtlebot 3. | 9. Use the command "detect mode" to see whether there is a box on the human face when the main camera on Turtlebot 3 captures a human. | Y |
| 10. Establish a reliable connection with the Server to transmit the control signals. The connection delay should not exceed 10ms. | 10. Correctly receive and process various control signals sent from the Server and convert them into control signals of the car. | Y |
| 11. Test the communication delay between the OpenCR module and the Server to ensure that the delay does not exceed 10ms. | 11. Send various types of control signals to the OpenCR module to verify whether it can accurately convert them into control signals for the car and control the car. Testing can be done using a simulation environment or an actual car. | Y |