User Manual

Gabriel Anderberg
Karl Hilding
Gustaf Härold
Klara Kastenson
Tai Ta
Michael Yasemi

December 16, 2022

Version 1.0

Status

Reviewed	Gustaf Härold	December 7, 2022
Approved	Martin Skoglund	December 7, 2022

Project Identity

Group E-mail: gusha385@student.liu.se

Homepage: http://www.isy.liu.se/tsrt10/group

Orderer: Martin Skoglund, Linköpings universitet
Phone: +46 13 281890
E-mail: martin.skoglund@liu.se

Customer: Sergi Rotger Griful, Eriksholm Research Centre
Phone: -
E-mail: segr@eriksholm.com

Supervisor: Johanna Wilroth
Phone: +4670-894 48 49
E-mail: johanna.wilroth@liu.se

Course Responsible: Daniel Axehill
Phone: +46 13 28 40 42
E-mail: daniel.axehill@liu.se

Participants of the group

Name	**Responsible**	**E-mail**
Tai Ta	Responsible for the Documentation (DOC)	taita693@student.liu.se
Klara Kastensson	Responsible for the Design (DES)	klaka376@student.liu.se
Karl Hilding	Responsible for the Testing (TEST)	karhi203@student.liu.se
Michael Yasemi	Responsible for the Hardware (HW)	husya078@student.liu.se
Gabriel Anderberg	Responsible for the Software (SW)	gaban592@student.liu.se
Gustaf Härold	Project Leader (PL)	gusha385@student.liu.se

CONTENTS

DOCUMENT HISTORY

Version	Date	Changes made	Sign	Reviewer
0.1	2022-12-01	First draft.	All	GH
0.1	2022-12-07	First version.	All	GH

1 INTRODUCTION

This document is a merge of the user manuals from 2020 [1] and 2021 [2]. It also extends and corrects the previously written manuals, in order to give the reader the necessary information about the project *Sensor Fusion for Hearing Aid Control*. The document assumes that the reader has access to the information given in the *Requirement specification* [3], *Technical report* [4] as well as the documentation available in the *Git repository* [5].

2 HARDWARE

The hardware includes one pair of Tobii Pro Glasses 2 and Tobii Pro Glasses 3. It comes along with battery, hard driver with SD-card slot, SD-card reader, battery charger, equipment for Qualisys and calibration tools. There are a lot of good information about the hardware and how to use the glasses on Tobii Pro:s homepage.

2.1 Tobii Pro Glasses 2

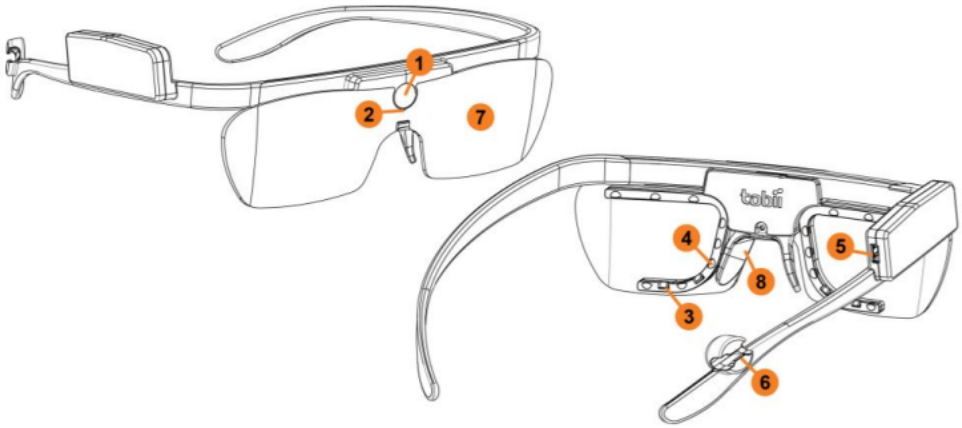

Figure 1: Tobii Pro 2 glasses components. Image rights: Tobii Pro AB

The components from Figure 3 are listed below:

1. **High-definition scene camera** - Captures video of what is in front of the user.
2. **Microphone** - Picks up sounds from the user and the surroundings.
3. **Eye tracking sensors** - Records eye orientation e.g. the direction of the eye gaze.
4. **IR illuminators** - Illuminates the eyes to support the eye tracking sensors.
5. **Micro HDMI connector** - Connects to the recording unit via the supplied HDMI cable.
6. **Cable guide** - Guides the HDMI cable away from the user.
7. **Nose pad** - Exchangeable and different sizes available in the case.

To run the glasses together with the existing code, a few steps to get started are listed below:

- Power on the glasses. Connect your computer to the glasses through WiFi. Password: tobiiglassespro
- Run for example run_tobii_gaze.py to test the glasses live

- You can also run calibrate_and_record.py, to record to the tobii SD card. The data includes video stream given as an .mp4 file and gaze and IMU data stored in a .json format.

Tobii Pro Glasses 2 are setup through Wifi and can either be launched by Tobii's software or provided python code. Video and IMU data are saved in the memory card which can easily collected from the SD-card reader. It can be done with the following procedure.

1. Start the hard driver with SD-card slot, connect with a computer through Wifi.
2. Run the script **calibrate_and_record**.
3. When finished, access the data with the SD-card reader. Live data files (.json) for IMU and gaze data and full stream (.MP4) for camera data.

2.1.1 *Glasses data*

The data accessible from glasses are listed in Table 2.

Table 1: Table of the names and properties of .json file collected from Tobii Pro Glasses 2

Type	Name	Property	Dimension
"ts"	time stamp	Monotonic timestamp of the data	μs
"s"	Status	If equal to zero it currently no errors, any non zero value indicate something wrong with data	-
"gidx"	Gaze-index	All gaze messages belonging to a single gaze-event share the same gaze-index	-
"l"	Latency	Expressed in a number of time from the time the image is received from the camera to the time it is queued for transmitting the live data stream	μs
"gp"	Gaze-position	Tells where on the frame the user gaze is	$[0, 1]$
"gp3"	Gaze position in 3D	Tells the 3D coordinate of the users gaze	mm
"pc"	Pupil Center	Specified in 3D coordinates with origo in the scene camera	mm
pd	Pupil diameter	Pupil diameter for each eye	mm
"gd"	Gaze direction	At which direction the user gaze	$[0, 1]$
"gy"	Gyroscope data	Indicates the rotation of the glasses	$[\text{deg}/s]$
"ac"	Accelerometer data	Indicates the rotation of the glasses and if stationary it will be approximately [0, -9,82, 0]	$[m/s^2$

2.2 Tobii Pro Glasses 3

Figure 2: Tobii Pro 3 glasses components. Image rights: Tobii Pro AB

The components from Figure 2 are listed below:

1. **Infrared illuminators** - Illuminates the eyes to assist the eye tracking sensors.
2. **Head unit cable** - Connects to the recorder unit to save and transmit data.
3. **High-definition scene camera** - Camera that captures the scene before the user.
4. **Microphone** - Picks up sound from the user and the immediate surrounding.
5. **Eye tracking camera** - Records eye orientation and movement.
6. **Accessory attachment** - Used for optional accessories for ex. Protective Lens.
7. **Nose pad** - Exchangeable and different sizes available in the case.

Following steps has to be done in order to start the glasses.

1. Connect the glasses to the recorder unit through the HDMI cable.
2. Press and hold the on-button until it stops flashing and stays static green.
3. Connect to the unit through WIFI. The password is **TobiiGlasses**.
4. Open the project application in order to start an experiment.

Further description on how to use the application is in the section below. Installation procedures is described in related gitlab repository.

2.2.1 *Glasses data*

The data accessible from glasses using g3pylib API are listed in Table 2. The data is structured as a dictionary.

Table 2: Table of the names and properties of .json file collected from Tobii Pro Glasses 2

Type	Name	Property	Dimension
"ts"	time stamp	Monotonic timestamp of the data	s
"Acceleration"	Acceleration	Three-dimensional acceleration vector.	m/s^2
"Gyroscope"	Gyroscope	Three-dimensional orientation vector.	deg/s
"Magnetometer	Magnetometer	Three-dimensional vector of the magnetic field.	μT
"gaze2d	Gaze-position	Horizontal och vertical position of the gaze on normalized video coordinates.	$[0,1]$
"gaze3d"	Gaze position in 3D	Vergence point of the gaze from right and left eye with respect to scene camera.	mm
"righteye.gazeorigin"	Pupil center	Location of the right eye relative the scene camera.	mm
"lefteye.gazeorigin	Pupil center		mm
"righteye.gazedirection"	Gaze direction	Estimated gaze direction with the right eye as center.	-
"lefteye.gazedirection"	Gaze direction	Indicates the rotation of the glasses	-
"righteye.pupildiameter"	Pupil diameter	Diameter of the pupil.	mm
"lefteye.pupildiameter"	Pupil diamter		mm

Note that all data doesn't arrive at the same time. Acceleration and gyroscope data is sampled at 100 Hz while magnetometer data is sampled at 10 Hz. For the gaze data, there is an option to either sample at 50 Hz and 100 Hz. All data is included with a time stamp.

3 FACE MESH AND SLAM-MODULE

In this section, the setup of the hardware and an explanation of the code for this project will be explained. The code will only be related to the SLAM module, face mesh and how to validate test data for these.

3.1 Script from 2021

The main script main.py runs all functionality that is developed in this project. The code needs data, either using live or recorded data form the glasses, or using webcam. The function can also run for example the facebox implementation from last years (2020) project. The parameters for running different implementations and settings are defined and changed at the end of the main.py code. The inputs and parameters are further described below.

Input:

1. Live tobii:
   Connection to tobii glasses via WiFi.
2. Live webcam:
   Webcam on computer, configured automatically (cap = cv2.VideoCapture(0)). Used for development of code.
3. Recording:
   Recorded data from tobii glasses. These are created using the script calibrate_and_record.py and should be named as the files in the folder test_data and be added as a test in config.ini .

Parameters:

- test_name:
  Name of prerecorded test from tobii glasses. See folder test_data and config.ini for examples
- Save:
  Saves the results as processed_video.mp4 and ekf.mp4. Also saves the states of the ekf as ekf_states_test_name in the folder of test_name.
- video_type:
  VideoType.VIDEO: To run a recorded test.
  VideoType.SIM: To run a simulated test.
  VideoType.GLASSES: To run live from Tobii glasses.
  VideoType.WEBCAM: To run live from Webcam.
- detector_type:
  DetectorType.FACEMESH: To run code with facemesh, witch enables mouth movement detection and improved distance perception
  DetectorType.FACEBOX: Runs last years implementation of face box, and corresponding distance perception.

3.2 Main application for current project

A detailed description on how to download and install the application is on gitlab page for this project (see CDIO-2022 branch). Once the installation is done simply run main_app.py through the python interpreter. Make sure that the Tobii

Pro glasses are connected to your laptop before that (password=tobiiproglasses). The glasses are turned on by holding the activate button on the recorder unit until it shows a static green light.

Before an experiment is performed, what modules to activate has to be chosen. This cannot be done once an experiment is started and can only be changed once you have canceled the experiment. Note that SLAM is only available if the face mesh is activated. If the Tobii Glasses are chosen as the input unit then calibration needs to be done for every new user or every time the unit is turned on. Note: failed calibrations can have multiple causes such as insufficient lightning, calibration marker too far/close, calibration marker not in view etc. In the case that the pupil is not detected by the calibration algorithm, it is recommended to change the nose pad so the glasses position is adjusted.

The application can be closed once the experiment is done. In order to turn off the glasses, press the activate button until it shows a static orange light. The glasses will turn off iteslf after a few seconds.

4 EYE-TRACKING

4.1 Introduction

In this section the eye tracking functions are explained. This years project is included environments light measurement, Detection of smooth pursuit and detection of listening effort using the pupil diameter.

4.2 Plots

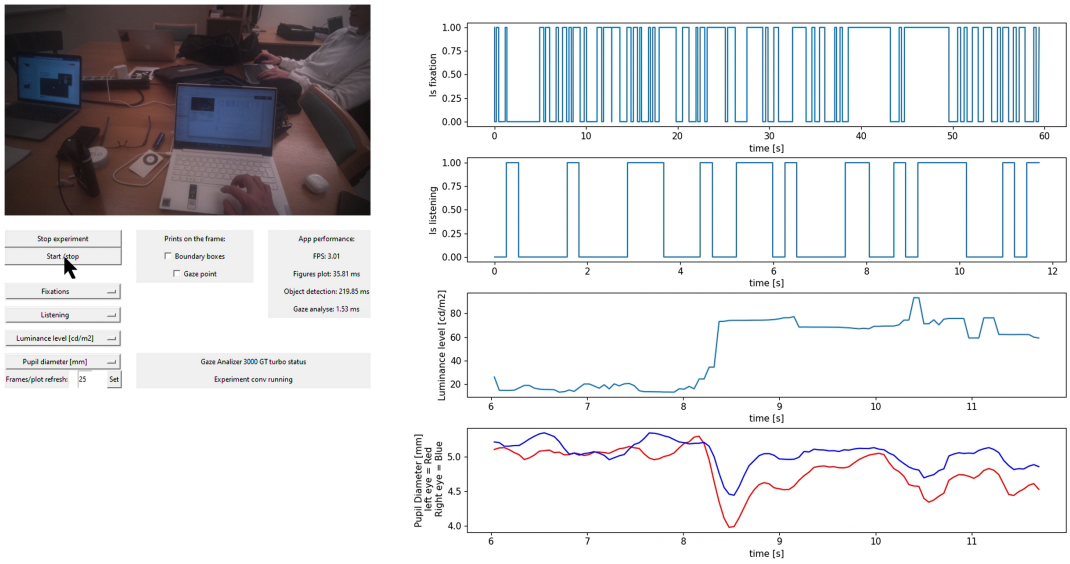

Figure 3: A view of GUI interface. In this case the Environment light has been changed from the dark to light after 8 secconds.

- Listening : Shows if the user is listening or not listening.
- Luminance level : Shows the Brightness of the environment.
- Pupil Diameter : Shows the pupil diameter for both left and right eye.
- Average saccade frequency : Shows the users saccade frequency in Hz.
- Movement classification : 1 indicates that the users is in fixation, 0.5 shows that the user is in smooth pursuit and 0 means that the user is in saccade.

4.3 Functions

To measure environments brightness the function `measur_lum` has been used. In order to remove the outlier in the light measurements, the function `reject_outliers(data,c)` is used. Greater c means removing less values as

outlier.
Function `listening_sett` is used to determine if user is listening. 1 means that the user is listening and 0 means the user do not listening. The Code will determine the listening set one time in five instances. That is why the function returns one vector with five values. This is more convenient when plotting the data.

5 SIMULATION ENVIRONMENT

5.1 Introduction

This section will cover the user manual of the simulation environment. It will include an installation guide and an overview on how to run and tweak the simulation. The simulation environment mainly uses Gazebo for the physics engine and ROS2 for controlling models and data handling. This guide will not go into detail over how the software works and it is therefor recommended to read through beginner manuals for ROS2 and familiarize with the web-page `https://docs.ros.org/en/foxy/Tutorials.html`. Do notice that this project is using ROS2 and not ROS, it is needed to be aware of this when searching for soulutions and troubleshooting. Unfortunately the simulation enviroment is not very user friendly, and neither is ROS2. It is our hope that this manual will provide some help to future users or developers for this specific project with some understanding of ROS2 and Gazebo.

5.2 Installation

This section installs the needed packages for running ROS2 and Gazebo. By running and launching the command `sudo bash installscript.bash` it downloads the following packages:

- ROS2-Foxy-Fitzroy
- Gazebo 11
- Python 3.x
- ffmpeg
- colcon
- Gazebo_gazebo_ros_pkgs

As well as the following python libraries.

- kivymd
- pyaudio
- audio2numpy
- sounddevice
- numpy
- pynput
- webrtcvad (for voice activity detector)
- pydub (for voice activity detector)
- contextlib (for voice activity detector)

If there are any problems with installing the dependencies, install the dependencies manually as discussed in section 5.11.

After installing the software mentioned above, a few more things are needed to be done. From the project files from the git SIM branch, navigate to Sim_ws/src/Simulator_launch/models and copy the content. In your home directory there is a hidden .Gazebo folder. Here you can find another directory called model, paste the copies files into this folder and overwrite if necessary. Unfortunately, this process have to be repeated if any change to the models are made.

Camera photos have been stored in tmp/camera_save_sim. Use ffmpeg to turn the photos into a video. For a 25 fps video (which is the capture rate of the modelled camera and the tobii-glasses camera), type "ffmpeg -r 25 -pattern_type glob -i '/tmp/camera_save_sim/default*.jpg' -c:v libx264 Video_name.mp4". There can sometimes be large differences between simulation data output time and camera video time. This seems to be fixed when running both ffmpeg and the simulation on more powerful hardware. If this "bug" is due to ffmpeg error or simulation has not yet been fully investigated.

5.3 Starting up the simulator

Using a bash script will help you to open the Gazebo simulator quickly with the least amount of commands in the terminal. The first thing you do is to open a terminal in the Sim_ws folder `./startup_simulator.bash .` In the terminal you will see a selection of different launchfiles you can choose from

- launch_2021
- dynamic_launch
- semi_launch
- static_launch
- launch_2022

To simulate the different cases you can for example write `./startup_simulator.bash dynamic_launch` in the terminal. This means that we tell ROS2 to use the launch package "Simulator_launch", and specifically the file "dynamic_launch". The simulator is now starting and to see the visuals, i.e. open the Gazebo environment, you must open a new terminal and write `gzclient`. To quit the simulation, press ctrl+c in the terminal window where it is running. Sometimes you will have to press it several times. If that for some reason does not work, open another terminal window and type `killall gzserver`. Data from the simulation will be saved to `Sim_ws/saved_data/`.

We recommend you to look at a launch file to gain a better understanding of what is happening and being build. If you want to do it step by step in the terminal the procedure will be: In the Sim_ws terminal type `source /opt/ros/foxy/setup.bash`. This has to be done every time you open a new terminal and intend to use ROS2 functionality. After that you write `colcon build` and `source install/local_setup.bash`. Colcon will now build the files and packages. If any changes are made to the python files, this has to be done again. While not always necessary, if changes to models have been made, it is advised to remove the install, log and build folders (but not the src!) and rerun `colcon build`. This is to avoid conflicting models. After building the workspace through colcon, open Sim_ws in the terminal and type `.install/setup.bash`.

Finally, we use a slightly altered version of the Gazebo planar_move-plugin, for Gazebo to able to find it and not use the normal Gazebo_Ros_pkgs-plugin, in the terminal, write:`export GAZEBO_PLUGIN_PATH=/[PATH TO Sim_ws]/build/gazebo_plugins:$GAZEBO_PLUGIN_PATH`, where "[PATH TO Sim_ws]" should be replaced with your path to Sim_ws.

There are different bashscripts to run. Go to the Sim_ws folder in the terminal and type `ls` to see the names of the different bash-scripts, in the terminal you will see:

- startup_all.bash
- startup_simulator.bash
- startup_gui.bash

One bash-script starts the simulator, one starts the GUI and one starts both of them. However when starting both the GUI and the simulator in the same terminal it has problems with closing the GUI window after you have shut down the terminal. The GUI was created by the group of 2021 and can ease the understanding of the the simulated environment since you can see directly in the simulation environment what is happening. We do recommend you to focus on understanding ROS2 and Gazebo instead of improving the GUI since it will ultimately lead to an extra step in the process of development.

Data from the simulation will be saved to Sim_ws/saved_data. This will include IMU-data and eye-data with gaze vectors to all speakers and true data on the position of both the user and all three speakers. The IMU and Gaze-data are formatted to mimic the format of the Tobii-glasses, while the true position data mimics that of "Qualisys" from LiUs Visionen.

5.4 Configuring the GUI

The GUI for the simulation is based on the Kivy and KivyMD framework. In the gui_menu-package there are two files used. One Kivy file for the appearance, gui.kv, and one python file for implementation of the nodes and functions being published when some interaction with the menu is made, gui.py. Kivy have lots of information and documentation on their website if you want to implement new functions or change already existing ones in the GUI.

At the time of writing only speaker objects can be added and removed using the GUI. Have a look at the existing code for implementation of other object types. Kivy can not open multiple windows, but you can use a "new page" option for more space and new functions. This can be seen in the GUI as the buttons New Object and Back. Pay attention to how the new pages are created in the kivy-file.

Kivy has an inbuilt function called on_press and on_release which can be used in the kivy-file to call a function defined in the python file. This function can then publishes a topic, a Bool or a String for example. This is how the kivy and python files communicate with each other.

5.5 Configuring the Gazebo environment

To change the aesthetics and the .world files you can build it and save it directly in the Gazebo environment and the call upon the world in your launch-files. There are multiple tutorials and gazebo_models to use on the web. When

saving the environment as a new world it creates a file in `/oticon_2022/Desktop/oticon/Sim_ws/src/simulator_launch/worlds/`.

5.6 Configuring the simulator

The easiest way to configure a specific scenario in a simulation can be done in the python file "dynamic_path_controller.py". The file can be find inside `C:Sim_ws/src/dynamic_path_controller/dynamic_path_controller/`. When launching with dynamic options this is the file that controls the movement of the user's body and head, as well as the movement of the speakers. Within the file there is a class: "Dynamic Path Controller". This class is a node for ROS2. It subscribes to "demo/odom_user". It controls *User* head and body aswell as *Speaker* body. What actual message that are pub- lished to these topics are defined within the class's callback function "path_controller_callback(self, msg)". Here you can see several parameters be defined, such as user_lin_vel_x or speaker3_ang_vel_z, which are the linear velocity to be applied to the user in the x-direction and the angular velocity to be applied to speaker 3 in the z-direction (ergo turning).

To create a new scenario another case in the if-statement must be created. As for now there are 7 cases and only case 6 and 7 were used in the project of 2022. To launch the specific scenario the chosen case must be selected and the save the file before running it in the terminal. If you are not using the *.bash* script you must rebuild with `colcon build` in the terminal in Sim_ws folder in order to make the changes apply.

5.7 Make a new package

First make sure that in the terminal you stand in the src folder. This can be done using the cd command to maneuver to the correct folder or by left-clicking in the designated folder in the file manager and selecting Open in Terminal. When in the src (source) folder, write:`ros2 pkg create –build-type ament_python <package_name>` which will create a new folder with all necessary files. Let's say you name the package gui_menu, then this is what the new folder will be named. Inside this folder an additional folder with the same name will be created. This is where you will add your python-files for the nodes. For every python-file you add to your package you will need to add a line in the file setup.py. In the bottom of the setup-file there will be a line of code saying 'console scripts': [..." in which you will add the following: *name_of_executable = name_of_package.name_of_python_file:main*. See existing packages for examples of implementation.

5.8 Configuring sound-physics

At the time of writing, there are two good songs available for testing the sound-physics, located at – . When launching the simulation environment, these soundfiles are converted to vector format, i.e. a numpy-array. On runtime, these vector are sent chunk by chunk at the preset frequency over topics by the node *speaker_node*. Hence, each sound gets its own topic. The length of these chunks are arbitrary but note that too long chunks might yield unrealistic sound as the physical properties of the sound will be determined at the start of each chunk. For example a 5 second chunk would use the same distance between users over the full 5 second sound regardless of how the simulation environment changed during those seconds. On the contrary, too small chunks could lead to performance instabilities as a typical digital soundfile has a sample frequency of fs $\approx$ 48000 Hz. Therefore the ideal chunksize of 1 vector element would

require 48000 topic updates each second.

For the end-purpose of providing prerequisites to implement the microphone of the Tobii-glasses, the above functionality should be sufficient. However, to provide some intuition of what the user hears in the simulation environment, an audio-player is implemented in the node user_hearing. This audio-player is severely limited by not being an open stream, i.e, the stream opens and closes each time a chunk is received. This results in unpleasant audio for chunk-sizes lesser than about 0.5 seconds. Making this an open stream could therefore be an area of focus for someone interested in the future.

While the simulation runs in an 3-dimensional environment, sound is scaled according to typical field intensity in 2-d space i.e $Amplitude_{listener} = \frac{Amplitude_{source}}{r^2}$. Scaling by room r^3 instead of plane r^2 resulted in an unreasonable dampening at longer distances. This amplitude scaling model can be found at the user_hearing node. Some conventient audio functions are located in `Sim\_ws\/src/sound\_functions.py`.

5.9 Movement controllers

It is possible to add a controller to a speaker from the GUI. For example if you type "1" and then press "set controllers" the controller will be added to speaker number 1. If you type "12" and then press "set controllers" then the controller will be added to speaker 1 and 2. There is three different controllers. One reads inputs from the keyboard and moves the speaker accordingly. Only two speakers at the time can be controlled from the keyboard. One with "w,a,s,d" and one with "up,down,right,left". The other two controllers doesn't have a maximum limit of speakers except the fact that the maximum number of speakers in the simulation environment is at the moment 10. One will give an oscillating linear motion and another will make the speaker move in a circle.

5.10 Adding and removing speakers

To add a speaker need to input x and y position given in meters as well as orientation in z given in radians. Then press "spawn speaker" which will add a speaker to the environment. In the beginning there is three speakers named speaker_head_1, speaker_head_2 and speaker_head_3. The next one will be named speaker_head_4 and then speaker_head_5 and so on. Removing a speaker won't change the order of naming of speakers. However if you have spawned speakers till the maximum limit of 10 and then remove speaker_head_4 the next spawned speaker will be named speaker_head_4.

5.11 Troubleshooting

- Ubuntu can't find ROS2

This is usually due to ROS2 not being sourced. In the terminal, write: source /opt/ros/foxy/setup.bash

- Simulation is not launching.

This can be due to several reasons, such as incorrect python code (can usually be seen in the terminals error messages), a simulation already being run (or at least not fully terminated) in another terminal (try killall gzserver), faults in the model/world-files or conflicting model/world-files. Make sure the model and world files are correct. Make sure model files in .gazebo and in the Sim_ws are the identical. Remove the build, install and log map from Sim_ws and run colcon build again in order to make sure that there are no old versions causing conflicts there.

- Head is not moving when it should.

This could either be due to the wrong case being applied in the path controller or that the plugin path is not exported. Check path controller and type "export GAZEBO_PLUGIN_PATH=/[PATH TO Sim_ws]/build/gazebo_plugins:$GAZEBO_PLUGIN_ into the terminal. It is also possible that it could be to old conflicting models.

- Complains about missing package or wrong version.

This can be because a system dependency or python dependency is missing. If there is a problem with a python package, you can run "pip show <package_name>", and if the package is installed, it will output the version of the package. If it has the wrong version, you can run "pip install <package_name> –upgrade" to update the package. If the package is missing you can run "pip install <package_name>" to install the package.

If it is a system dependency that is missing, you can run "dpkg -s <package_name>". If the package is missing or needs to be upgraded you can run "sudo apt-get install <package_name>"
.
Check for conflicting models or error in python code (such as wrong case chosen).

Here are some general tips and tricks for troubleshooting.

If you go into the same folder as a world file you can launch just that world, without launching all the other scripts. This is great for making sure the simulation world or model files are not causing problems. Open "...Sim_ws/src/Simulator_lanuch/worlds" in the terminal and write "gazebo sim_world.world" to launch with GUI, or "gzserver sim_world.world" to launch without GUI (if you open another terminal and type gzclient, the GUI will be launched). You can also add –verbose after "gazebo/gzserver" to get more information.

If you have started a world without the launch file you can open another terminal and type "ros2 run dynamic_path_controller dynamic_path_controller dynamic_path_controller.py. This makes sure that, aside from the world, models and their related nodes, only the dynamic_path_controller node is started.

In your home folder there is a hidden .bash.rc file. If you go to the end of the file and enter, for example, "source /opt/ros/foxy/setup.bash", then this will be entered automatically in all terminals you start. Some may find it useful not to have to enter source or export commands every time a new terminal is started.

With the command "ros2 topic list", the running topics will be listed. If you instead write "ros2 topic list -t" the message type of the topic will also be displayed. Similar, "ros2 topic echo <topic_name>" will show any information the topic receives in the terminal. The add-on rqt for ROS2 is very useful and can be used to publish messages directly to topics. This means you can use it to see if subscribers works as intended. rqt_graph can be used to visualise the network of nodes and topics within ROS2. If installed both rqt and rqt_graph can be launched by typing "rqt" or "rqt_graph" in the terminal.

6 QUALISYS

This section aims to provide the basic information to run a test in Visionen with the Qualisys system. To gain information about ground true data for the glasses wearer and users of interest Qualisys was used.

6.1 Setup and calibration

If you are new to testing with qualisys we suggest contacting someone who can explain how it works. Since tests have been done in previous groups a node between Qualisys and ROS2 is already in place. In order to be able to make it work some arrangements needs to be done.

- All objects of interest needs to have at least three reflecting balls mounted on itself (i.e. on a cap or hat)
- For the glasses there is a special stand for the reflecting balls, try to use bigger balls to make sure that you establish a stable connection to qualisys.
- The file save_6dof.py, is the script that retrieves data from qualisys (Position and rotation of all users). For the script to work bodies in qualisys needs to be named "User", "Speaker1", "Speaker2" and "Speaker3". More speakers can be added.
- calibrate_and_record is the script that retrieves data from glasses. (MEMS and video stream).
- Get in touch with someone who can explain in depth how qualisys work i.e. setup, calibration and recording.

6.2 Testing

The test procedure was as follows:

1. Start calibrate_and_record.
2. Arrange persons involved as test description.
3. One "speaker" cover a reflecting ball (this way eye ware data and qualisys that can be synchronized).
4. Start Qualisys recording and save_6dof.py.
5. Uncover the reflective ball.
6. Do the test.
7. Stop Qualisys recording.
8. Stop glasses recording.

6.3 Data processing

To make the data from qualisys and glasses more work-friendly it has to be processed. Mainly to sync them and cut away unnecessary parts. To this a movie editing software is need. There is two part for this, data from qualisys and data from glasses.

- Edit away data which was before the uncover of reflect ball.
- Edit the data so that "ts" starts from 0. It can be done in clean_data.py

The data for the glasses must also be edited as:

- By look at the video, edit out parts before uncover of reflect ball. Both for livedata and video stream.
- Edit the data so that **"ts"** starts from 0.

7 OTHER FEATURES

Other features which have been implemented.

7.1 Voice activity detection

A simple python file for voice activity detection has been made which is based on the google library webrtcvad which is supposed to be one of the best. To use this file you need a recording which needs to be converted to wavformat. In the bottom of the python-script under "if __name__== '__main__'" there is a variable called path_to_wav where you will write the name of your file. Also don't forget to add the file to the folder. Since changes to the file has to be done before using the webrtcvad library the altered wav-file will use another file which is in the variable "target_path_to_wav". You don't need to change anything in that variable. The output from the script is a list with booleans. Each boolean corresponds to 30 ms in the sound file and says true if the script thinks that there is a person talking and false if it believes there is no one talking.

REFERENCES

[1] (2020) User manual 2020. [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2020/oticon/documentation/user_manual.pdf

[2] (2021) User manual 2021. [Online]. Available: http://www.isy.liu.se/edu/projekt/tsrt10/2021/oticon/documents/User_manual.pdf

[3] (2022) Requirement specification. [Online]. Available: https://www.overleaf.com/project/6335409646a638172bef6d32

[4] (2022) Technical specification. [Online]. Available: https://www.overleaf.com/project/6373cc348e4040df8a9a2b6d

[5] (2022) Oticon gitlab wiki. [Online]. Available: https://gitlab.liu.se/tsrt10/2022/oticon/-/wikis/Oticon-Wiki