Advanced Programming of Interactive Systems

Project Implementation

TalkyHand

Arturo Benedetti
If Antonov
Sibilla Domiziana Silbano

Prof James Eagan
TA Mehdi Chakhchoukh

École polytechnique universitaire de l'université Paris-Saclay

20th October 2023

Contents

1 Work Distribution

In our team, we decided to split the work based on individual strengths and areas of interest. Here is a summary of the work distribution:

Arturo:

- **Gesture Recognition:** Arturo focused on the implementation and recognition of gestures, a critical component for our sign language translation system.
- **Motion Recognition:** He also worked on dynamic gesture analysis, contributing to improving the accuracy of gesture detection.
- **Networking:** He was also responsible for implementing the networking functionality, a crucial part in managing communication between system components over the network, ensuring a connection between two different computers.

If:

- **Speech Recognition:** If focused on developing speech recognition, enabling our system to understand and convert spoken language into text.
- **Auto-completion:** He also contributed to implementing autocompletion, enhancing the user experience by providing word suggestions.
- **Networking:** He collaborated with Arturo in managing network communication, ensuring a robust connection between components.

Sibilla:

- **User Interface:** Sibilla crafted the visual aspects of the project, designing an intuitive user interface to facilitate user interaction.
- **Text-to-Speech (TTS):** She also handled the implementation of text-to-speech to enable text translation into audio.
- **Usability:** She contributed to ensuring that the user experience was at the core of the project, optimizing the interaction flow and ease of use.

This work distribution allowed each team member to focus on their core competencies, ensuring effective and successful collaboration to complete the project. Each area of expertise was crucial for the development of an advanced interactive system, and we always aimed to maintain a balanced division of tasks and work.

2 Project Description

2.1 Introduction

For the realization of our project, we not only wanted to implement a highly interactive system but also looked for a relevant and interesting area for all the members of our group. Initially, we had considered developing a mini-games application, but during the second meeting we were inspired by watching a video in which hand tracking was used to interact with a digital map. In that moment, we collectively decided to pivot our project and implement a system based on hand recognition for analyzing sign language gestures. Today, Sign Language Processing, which pertains to languages that use the visual-gestural modality to communicate through manual articulations [15], is an emerging research field that deals with the automatic processing and analysis of sign language content. However, current language technologies primarily focus on spoken languages, ignoring sign languages and the deaf communities [3]. Therefore, we decided to develop our project in this field to explore how to make sign language accessible through appropriate technologies.

Furthermore, this project creatively integrates various forms of interaction, including gesture recognition, motion tracking, speech recognition, text-to-speech, and networking, to offer users a comprehensive experience. The main purpose of our system is to enhance communication within the deaf community by assisting them in using sign language. We believe that this application, designed to address a real issue, has immense value as it directly contributes to improving the lives of people within the deaf community and promotes inclusion.

2.2 Functionalities

Our project idea centers around the development of a sophisticated and advanced system designed to recognize hand gestures and translate them into corresponding meanings in American Sign Language (ASL) alphabet. Essentially, it will function as a sign language translator and an interactive prompter. The primary functionality of this system relies on the camera's ability to detect hand movements, identify specific gestures, and subsequently translate them into corresponding characters, forming words, phrases, and even complete sentences. Both static and dynamic gestures can be analyzed, each associated with its respective meaning.

Recognizing that spelling out complex words using individual alphabet letters can be challenging for users, we have aimed to enhance their experience by implementing an auto-completion feature. Simply by forming the first 2 letters, users will be presented with three different suggestions that will appear on different sides of the screen (top left, top center, top right). Users can then guide their hand to the desired suggestion, and upon contact with the corresponding box, the word will be automatically completed. Additionally, the word that's currently being typed will be displayed at the bottom of the screen.

Moreover, our system will support a two-way conversation. One user will communicate using sign language, and the signs will be converted into text, with the added option of converting that text into speech to allow the hearing user to listen. The other user will use speech, which is transformed into text for the deaf user to read. To achieve this goal, we have implemented sign-to-text, text-to-speech and speech-to-text translation features. Additionally, we have integrated a networking system that enables the two users to communicate simultaneously using two different computers.

To maintain a coherent and accessible conversation, the entire exchange will be displayed on the screen in a chat-like format. Both the user who uses signs and the speaker one will have the ability to edit their previous messages, ensuring clarity and reducing misunderstandings. For this purpose, we have implemented specific gestures and voice commands that facilitate text modification.

Finally, to provide real-time feedback to users, the system will continuously display its status, helping to understand and ensure a smooth progression of the interaction.

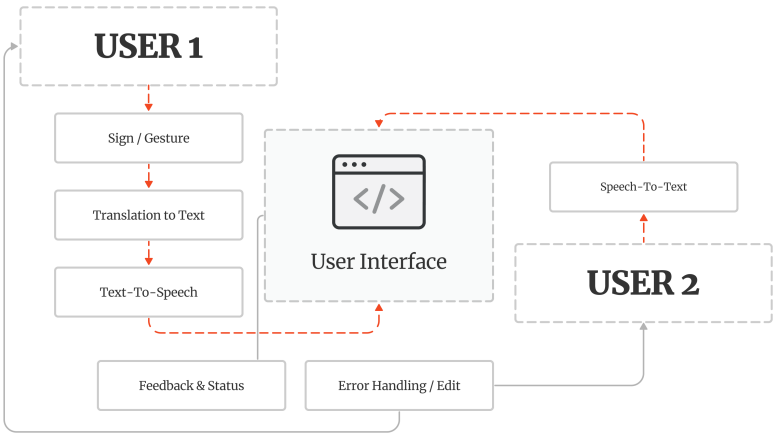
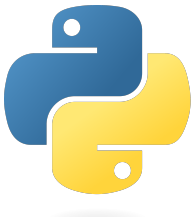


Figure 1: System Architecture Diagram

3 Project structure

To carry out this project, we have chosen to use Python, a programming language known for its simplicity, readability, and extensive support for various libraries and APIs. These attributes make Python an ideal choice for developing a complex system like ours. Additionally, Python also supports the creation of Graphical User Interfaces (GUIs), allowing us to use a single programming language for developing an intuitive and user-friendly front-end for our system. While as part of the course we covered Java Swing, we found it interesting to expand our knowledge by exploring the development of interactive systems with a different programming language that was better suited for our project.



3.1 Gesture Recognition

We used Mediapipe [4] to perform the hand tracking and the recognition of the gestures.

Initially, we considered employing a Convolutional Neural Network (CNN) to directly process webcam images for gesture recognition. However, this approach proved to be slow and inaccurate in handling a continuous webcam feed. To address this, we adopted a hybrid approach where we leveraged Mediapipe for hand tracking, obtaining hand landmarks, and then converted the images into points. We trained the model on these points, achieving high accuracy with test data. However, when applied to a live webcam feed, the model struggled to adapt to variations in hand position. The issue arose because the hand landmarks were saved relative to the position on the screen. Consequently, if the hand was in a different position while making the same gesture, the model failed to recognize the positional shift, leading to recognition problems. This led us to our final model, where we integrated Mediapipe for both hand tracking and recognition.

For training the model, we faced the challenge of classifying around 30 different classes, including alphabet letters and special characters like space or delete. We obtained an ASL alphabet dataset from Kaggle [2], consisting of 87,000 images with 29 classes. To tailor this dataset to our needs, we wrote a script that applies the Mediapipe recognition to each image, creating a dataset suitable for training our model. Notably, images where the hand was not recognized due to factors like lighting were excluded from the training data to ensure the model learned only from relevant examples.

In managing the recognized characters and words, we implemented a straightforward approach based on a time threshold for character recognition and specific events to distinguish words:

- For character saving, we introduced a time threshold mechanism. If a gesture is consistently recognized for a predefined duration (currently set at 0.75 seconds), the recognized character is saved. This approach ensures that users have sufficient time to complete a gesture and that accidental gestures are not erroneously saved.

- To differentiate words, we employed two key events. First, if a space gesture is recognized, it marks the end of a word. This approach allows users to naturally separate words during their signing process. Second, we implemented an autocompletion feature where prompts for completing a word are recognized. When such a prompt is accepted, it signifies the completion of a full word. These autocompletion prompts are designed to be full words, adapting to the word entry process.

The implementation of this character and word management system was encapsulated within a custom class, enhancing modularity and maintainability. This custom class was then integrated into the main script of our user interface, ensuring a cohesive user experience.

3.2 Auto-completion

For the auto-completion, we first found a dataset with the most common English words in speaking, and we even found it sorted by usage frequency [10]. We load the `.csv` file and fill an array with these words.

Every time the gesture recognition recognizes and writes down a character - we take the string that has been formed until then and search in the array all the words that are starting with said substring. We take the first 3 words (as the array is sorted) displaying them in the top middle, top left, and top right respectively. If there are less than 3 words we still show them in the same pattern.

Whenever the user wants to use the auto-complete, he just needs to "touch" it by moving their index finger in the place with the word that he wants. We recognize the position of the pointing finger and, if it collides with the invisible rectangle drawn in the area around the word, we take this word and write it in the input field.

3.3 Motion Recognition

For motion recognition, we tapped into an existing GitHub project [11] that allowed us to extract landmark points from videos. These points became the foundation for our motion recognition model. To integrate this into our project, we used dynamic time warping [7] as a similarity metric between signs.

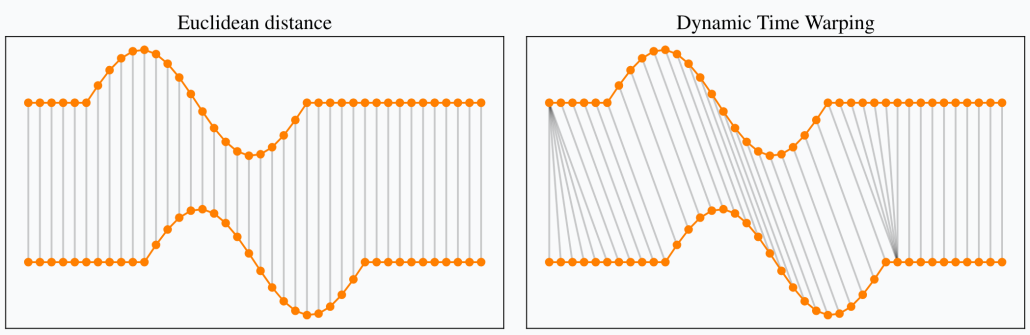


Figure 2: Visual comparison between Euclidean distance and dtw

Initially, we focused on a proof of concept with gestures like "hello", "yes", "no", "goodbye", and "thanks". These signs were chosen for their common use and simplicity. However, our system is flexible and can accommodate additional gestures if provided with enough example videos for training. Once a motion is recognized, the corresponding word is automatically transferred to the input field. Since these motions represent complete words, there is no need for additional features like auto-completion.

To perform the motion recognition, a dedicated button is provided in the user interface. Clicking the button initiates the recording of the motion. The subsequent click on the same button terminates the recording. After a brief processing period, the recognition result is displayed in the text field. In cases where the system fails to recognize the gesture, the status label is updated with a corresponding message, ensuring informative user feedback.

3.4 Speech Recognition

For the speech recognition we used the library Vosk [16]. It is an offline working API which recognizes speech in real time. The library requires a pre-trained model, which we obtained from the official website of Vosk creators [9]. We chose the lightest one as our primary concern was to make sure that it worked correctly for our proof of concept, as well as the fact we did not want to put heavy files in the project. The text processing is separated in another class called SpeechRecognition but the main logic of recognizing the speech is in a function called: speech_recognition() which is run on a different thread.

After setting up the model, we use the microphone to detect speech which is then processed by the Vosk library. The speech recognition is put on another thread which is stopped when we change modes between gesture recognition and speech recognition.

We provide a dedicated button for recording speech. When the button is pressed it starts recognizing the speech, outputting the result of the recognition as text inside the input field. When we press the button again, it sends the message directly to the chat.

There are also voice-commands:

- `again` - erases the input, so you can speak again new text
- `continue` - return to normal mode after you have used "remove" or "replace".
- `remove` - removes words or phrases
- `replace` - replaces a word or a phrase with another word or a phrase (it is used the following way: "replace word with word")

3.5 Text-To-Speech

For the project, we considered implementing a feature to enable text-to-speech, enhancing the communication experience. This can be achieved simply by clicking on the message one wishes to read in the chat.

Text-to-speech functionality was implemented using the Google API [8].

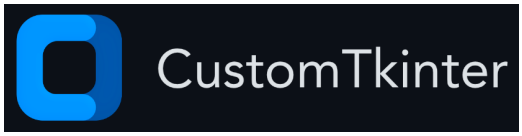
This choice was driven by the outstanding performance of the API in text-to-speech conversion. However, we also explored a viable alternative, the coqui-ai/TTS API [5], which is a deep learning toolkit for Text-to-Speech. Based on our research, it appeared to offer numerous possibilities for voice modulation. Nevertheless, we ultimately chose to use the Google API, as this text-to-speech functionality does not require specific prerequisites such as downloading libraries for different voices. We believed that using a simpler and more basic voice would be sufficiently suitable to achieve our set objectives. In any case, we made adjustments to the voice's accent and reading speed to promote faster and more natural communication.

The API is then integrated by adding a function that extracts the text from each label's message and passes it to the API. The API, in turn, generates the audio file, and we play it using playsound [12], a Python library used to play sound files directly from the script. This mechanism repeats every time a label is clicked. To avoid interrupting the camera's recognition and operation while the system is reading the content of a message aloud, this functionality was put on another thread.

3.6 User Interface

For the development of the interface, the aim was to create a highly usable and accessible product. Given that Python was used as the main programming language, we decided to implement our desktop application using a library that is an integral part of Python and is specially dedicated to supporting software development. This choice was made for several reasons, primarily the desire to learn a new library that could be used for creating high-level GUIs, to avoid future integration problems between the interface and the code, and because our project, as conceived, being a desktop application, did not require specific languages for online publication or for smartphone applications.

Initially, two options were considered for implementing the interface: PyQt5 [13] and Tkinter [17]. After a thorough study of both, it was concluded that these libraries were very similar. While the first had its strengths, such as offering cross-platform support, the second provided comparable functionality with a simpler integration into our Python-based project. Therefore, we opted for the Tkinter library and further extended its capabilities by implementing the CustomTkinter library [6] to meet our specific needs.



We paid special attention to consistently and clearly implement user feedback. This was done not only because the implementation of text-to-speech for a deaf user may not be perceivable without written feedback but also as a form of good design, following the Ben Shneiderman's eight golden rules of interface design [1], which emphasize the importance of offering informative feedback. Following these rules, we also expressly decided to keep all elements visible on the interface to allow the users to always see the available options and what they are doing as an input and what they are getting as an output.

Tkinter itself is a very useful library for creating graphical interfaces in Python projects; however, some limitations were encountered. For instance, it was not possible to create highly advanced and graphic designs due to the library's limited manipulation and preset capabilities. Additionally, for the layout of elements on the interface, the reactive grid method was used, but in some cases, this also posed limitations in arranging elements on the screen, especially when resizing the window. This made it challenging to make the interface content fully responsive.

The development of the interface was greatly influenced by the brainstorming session we conducted in class, during which we listed all the necessary features, resulting in the following image.

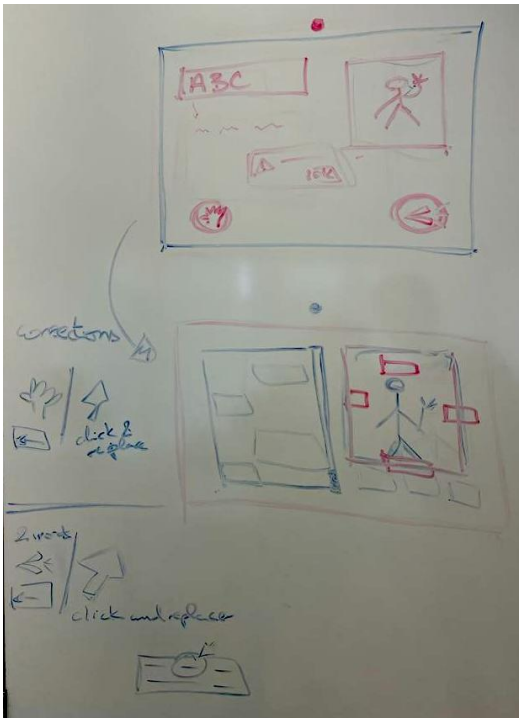


Figure 3: Draft of the interface made during in-class brainstorming

Based on this structure, we first created a low-fidelity prototype and then a high-fidelity one using Figma.

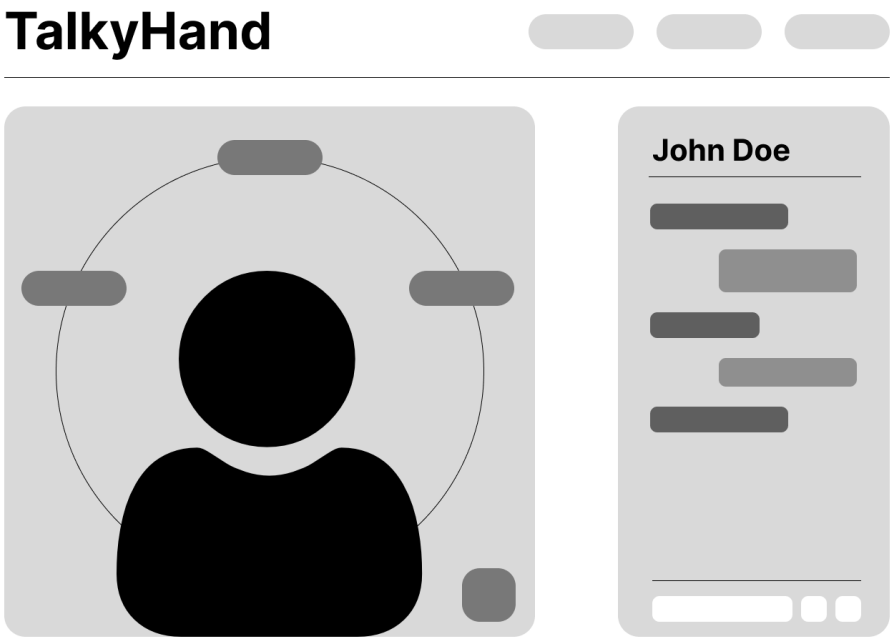


Figure 4: Low-fidelity prototype of the interface made in Figma

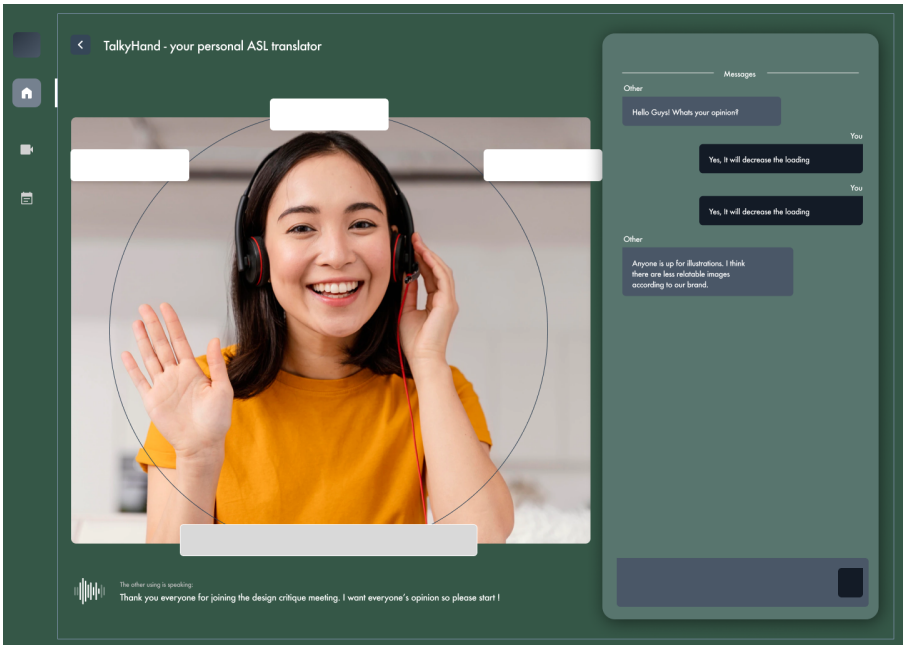


Figure 5: High-fidelity prototype of the interface made in Figma

In addition, paper prototypes and sketches were made to keep track of the features to be implemented and the work to be done in composing the interface.

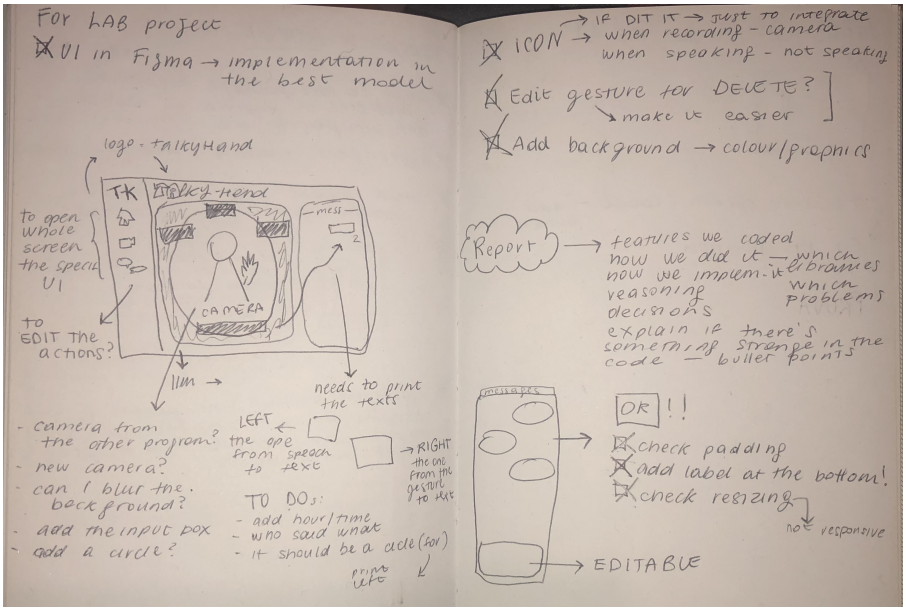


Figure 6: Sketches made throughout the interface implementation process

Below are some details regarding the graphical interface of our system.

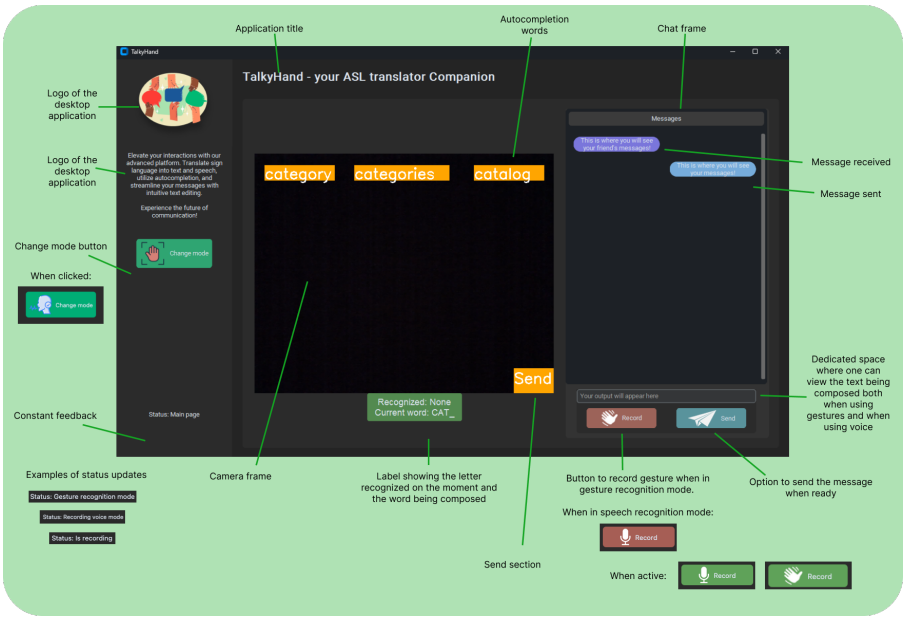


Figure 7: Detailed description of the user interface

Due to time limitations, we weren't able to optimize the user interface to be responsive across various screen resolutions, scales, or resizing scenarios. Testing was focused on specific settings where we ensured that the elements appeared and behaved correctly. While the UI might not adapt seamlessly to all configurations, it functions well under the tested conditions.

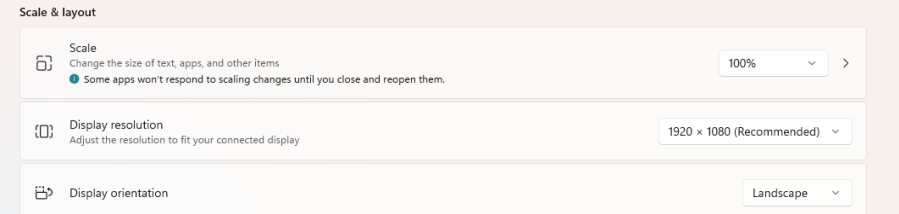


Figure 8: Specific settings for the screen

3.7 Networking

Our system incorporates networking capabilities, enabling two computers to connect, exchange messages, provided they are on the same network, and the network is not restricted by complex firewalls. We successfully tested this functionality using a mobile data hot-spot without encountering issues. This means that, with a proper configuration, the communication can also be carried out between devices connected over the internet, enabling for a full-scale remote interaction.

To achieve this, we implemented a dedicated class for message sending, known as `ChatSender`. This class facilitates the establishment of connections and the transmission of messages on a separate thread. On the receiving end, the App class manages message reception on another thread, setting up the system to listen for incoming messages. Upon receiving a message, it promptly decodes it and updates the chat interface.

The underlying connection is established using Python's Socket functionalities [14], ensuring an efficient communication process between connected devices. We have a separate file called .env which contains two configuration variables for hostname and port number, so we can change between them easily with localhost(127.0.0.1) when testing.

4 Implementation process

When we had our idea well-structured on paper we decided to divide the tasks. We had 3 main functionalities on mind - the gesture recognition, speech recognition and the user interface. We split them up: Arturo - gestures, If - speech, Sibilla - UI. We created a GitHub repository, and we set up the environment in the second week when we had our idea fixed. After that we started our research on what libraries to use, and we tried on different files our functionalities. In the meantime we also started working on the 3 subtasks which were: auto-completion, text-to-speech and motion recognition. If - autocompletion, Arturo - motion recognition and Sibilla - text-to-speech.

After a while when we had something stable and working - we decided to put things together. First the gesture recognition was integrated with the UI, followed by the speech recognition. Then, we added the auto-completion and motion recognition, and at the very end we added the text-to-speech functionality. While merging one by one we encountered some issues, but little by little we managed to solve them.

We were gathering every week on Wednesday class and we were working all 3 hours. At home, we worked at least 2 times a week for some hours, increasing with the deadline getting short. In the last three weeks we also had some online meetings to fix some bugs and to discuss how to continue.

5 Conclusions

In conclusion, we believe that we have developed an advanced interactive system that incorporates all the functionalities we initially set out to implement. Our project to create a sign language translator system can certainly represent a significant step forward in the field of Sign Language Processing and may become a tool that could have a positive impact on the sign language community.

Nevertheless, we recognize that there is potential for further enhancements and development in our project. Our progress was constrained by both the project's time constraints and the necessity to concentrate on mas-

tering the new programming language, which was initially unfamiliar to certain team members. However, we identified some opportunities for further improvements that could enrich our system. For instance:

- The number of supported gestures can be expanded, enabling communication based not only on individual characters but primarily on gestures.
- Consideration can be given to incorporating facial expression analysis since American Sign Language (ASL) communication often involves not only hands but also body and facial expressions.
- The implementation of a second camera for more in-depth sign analysis can also be an option.
- The desktop application can also be converted to a web or mobile application with the purpose of improving the GUI.
- It is also worth considering the extension of network capabilities to improve communication and connectivity among users.

Regardless, we feel satisfied with the project that was carried out, and we think it was a great way to learn skills that will be useful to us in our future careers, such as learning how to apply and implement different libraries, by reading their documentation and reusing them in a customized way within our own project.

Bibliography

References

[1] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 4th. Pearson Addison Wesley, 2004. ISBN: 978-0-201-69497-0.

[2] *ASL Alphabet Dataset on Kaggle*. 2018. URL: https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

[3] Danielle Bragg et al. "Sign Language Recognition, Generation, and Translation: An Interdisciplinary Perspective". In: *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 2019, pp. 16–31. URL: https://www.researchgate.net/publication/336793034_Sign_Language_Recognition_Generation_and_Translation_An_Interdisciplinary_Perspective.

[4] *Mediapipe: Cross-platform, customizable ML solutions*. 2019. URL: https://developers.google.com/mediapipe.

[5] *Coqui-ai/TTS*. URL: https://github.com/coqui-ai/TTS.

[6] *CustomTkinter: a modern and customizable python UI-library based on Tkinter*. URL: https://github.com/TomSchimansky/CustomTkinter.

[7] *Dynamic Time Warping*. URL: https://en.wikipedia.org/wiki/Dynamic_time_warping.

[8] *Google Text-to-Speech API*. URL: https://pypi.org/project/gTTS/.

[9] *Model for speech recognition*. URL: https://alphacephei.com/vosk/models.

[10] *Most common English words*. URL: https://www.kaggle.com/datasets/rtatman/english-word-frequency.

[11] *Motion Recognition Project on GitHub*. URL: https://github.com/gabguerin/Sign-Language-Recognition--MediaPipe-DTW.

[12] *Playsound library*. URL: https://pypi.org/project/playsound/.

[13] *PyQt5*. URL: https://pypi.org/project/PyQt5/#:~:text=PyQt5%20is%20a%20comprehensive%20set,platforms%20including%20iOS%20and%20Android.

[14] *Python Socket*. URL: https://docs.python.org/3/library/socket.html.

[15] *Sign Language Processing*. URL: https://research.sign.mt/.

[16] *Speech recognition*. URL: https://buddhi-ashen-dev.vercel.app/posts/offline-speech-recognition.

[17] *Tkinter library*. URL: https://docs.python.org/3/library/tkinter.html.