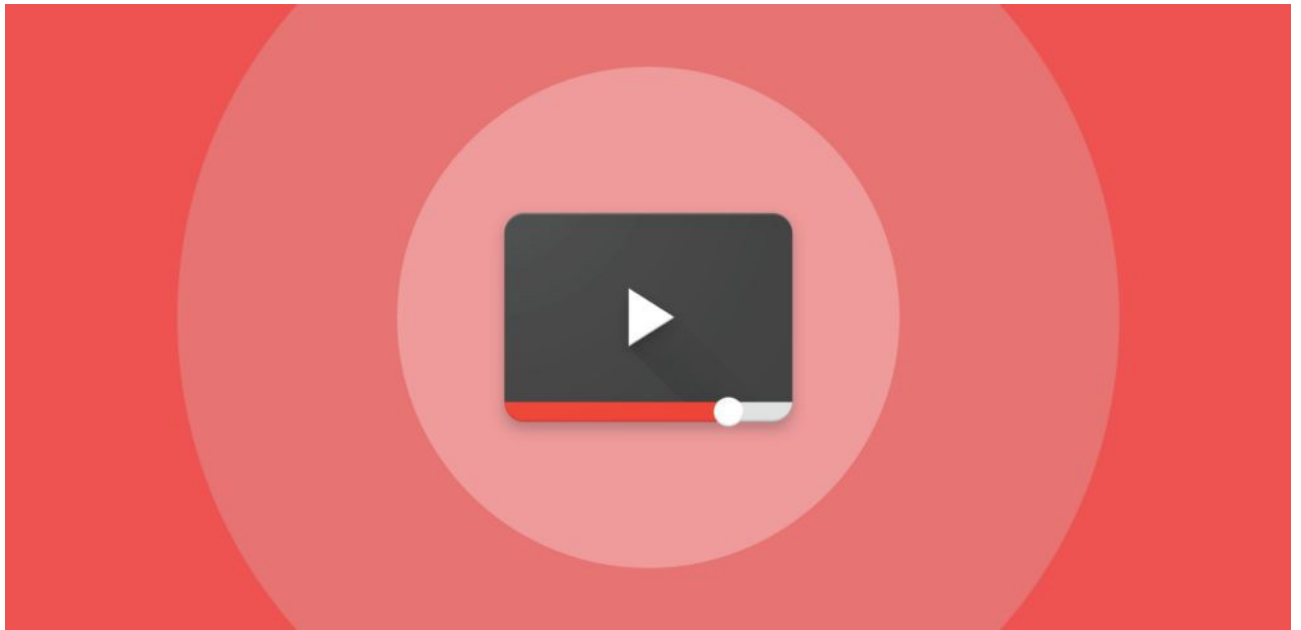


Multimodal YouTube

A YouTube Player for Everyone



Cannistraci 1603090

Ficarra 1659089

A.Y 2019/2020

Biometric Systems and Multimodal Interaction

INTRODUCTION

Multimodal YouTube is an application designed and developed to be accessible by both deaf users and not, that combines *biometric* and *multimodal* methodologies.

The system was conceived to mimic a video music service that can be controlled by using both *Speech Recognition* and *Gesture Recognition* methods. An enrolled user is automatically identified through a *Facial Identification* system, that is also able to understand his/her current mood by using an *Emotion Recognition* approach. Then, the identified user will be automatically redirected to a playlist of videos that reflects his/her mood and he/she will be able to interact with the player (e.g raise the volume if it is too low or skip the song if it is not appreciated) by both using hand gestures or voice commands. During the enrollment phase the user can register himself as a deaf user or not, in this way the system will know if it has to redirect the user to an ASL songs playlist.

CODE AND REQUIREMENTS

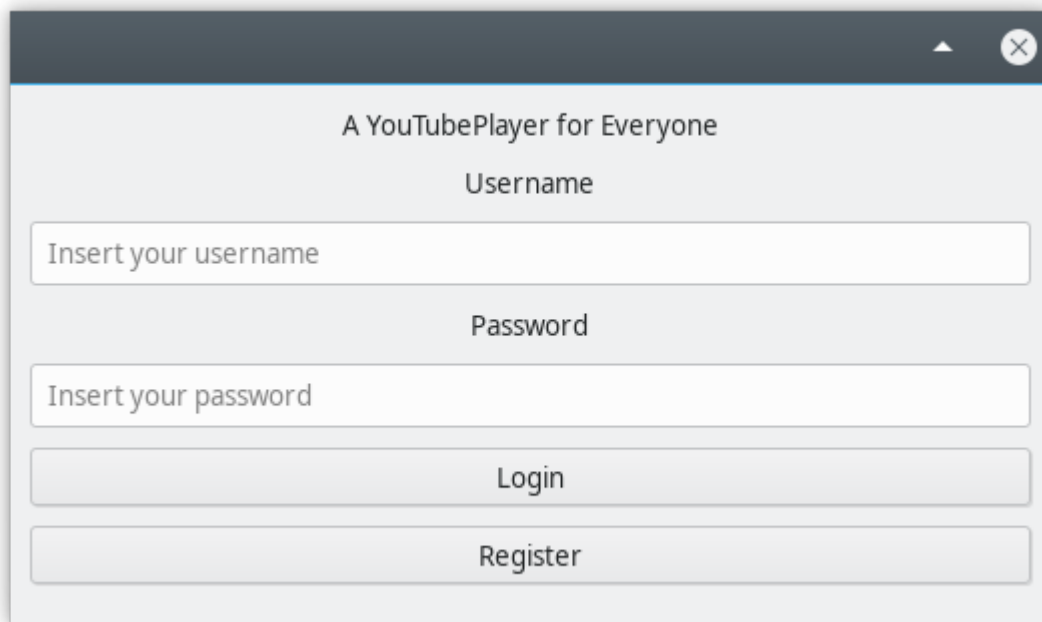
To run the application, your environment must satisfy the following requirements: run a linux system and install the necessary libraries. For more details about the libraries refer to section [API & Services](#). For example, under Ubuntu, you can use:

```
sudo apt install libgirepository1.0-dev gcc libcairo2-dev  
pkg-config python3-dev gir1.2-gtk-3.0 gir1.2-notify-0.7 vlc  
postgresql libgphoto2-dev
```

```
pip install pygobject python-vlc pafy youtube-dl pydbus mutagen  
psycpg2-binary opencv-python requests  
azure-cognitiveservices-speech gphoto2
```

SYSTEM ARCHITECTURE AND FLOW

When the application is started, the *login window* is shown and the system begins taking pictures to automatically **login** the user if it is already enrolled; if it is not recognized by the system, the user can try to login by inserting username and password and clicking on the *Login* button. The system also analyzes the taken picture to identify the current mood of the user, that will be used to redirect him/her to the most suited playlist.



A screenshot of a login window titled "A YouTubePlayer for Everyone". The window has a dark header bar with a close button. Below the title, there are two input fields: "Username" with the placeholder text "Insert your username" and "Password" with the placeholder text "Insert your password". At the bottom, there are two buttons: "Login" and "Register".

If the user is not registered yet, he/she can click on the *Register* button to perform the **enrollment** and the system will show a window in which the user can insert his/her wanted username and password, can select the *Deaf user* option and, by clicking on the *OK* button, the system will automatically take three pictures that will be stored in the gallery. We chose to take three pictures because it allows to have more variety in the gallery.

Registration

Insert your data and then wait until 3 pictures are taken!

New Username

New Password

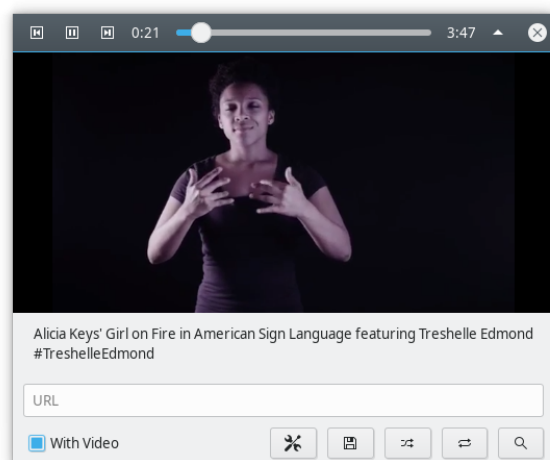
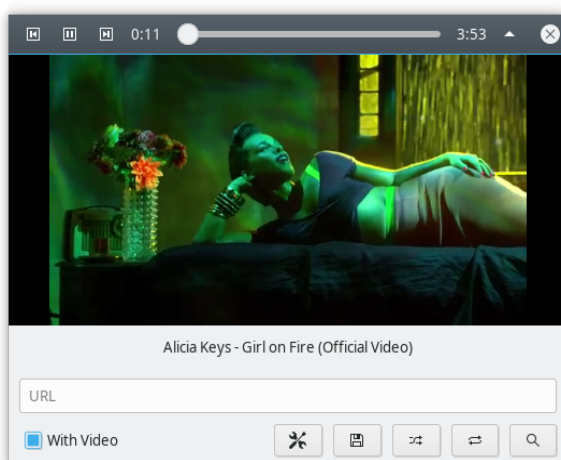
Insert your desired password

☐ Deaf user

Cancel OK

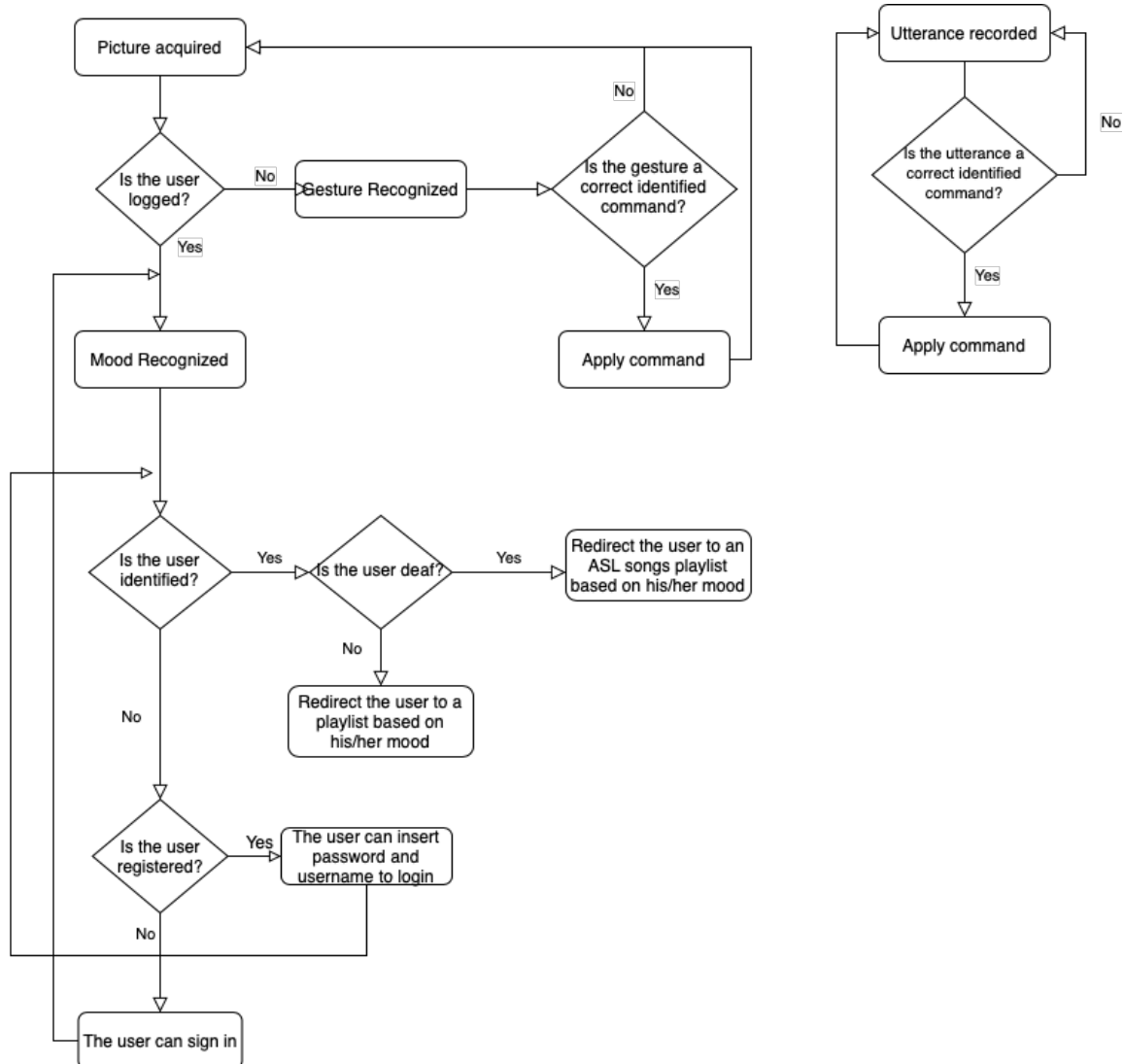
Username and passwords are stored in a local database, where passwords are encoded with a *hash function*, whereas pictures are hosted on a cloud server offered by the service (Face++) we used to perform facial recognition. Locally is saved only the identification token provided by the service, in this way the gallery is protected too.

After the registration, the user is automatically authenticated by the system. At this point, the system guides the user to the best suited playlist according to his/her mood, that was previously identified, and if he/she is deaf or not. The system can recognize the following moods: neutral, happy and sad. If the user is enrolled as a deaf user, he/she will be redirected to an ASL song playlist.



The playlists we created are reachable at the following links: [ASL “happiness”](#), [“happiness”](#), [ASL “sadness”](#), [“sadness”](#), [ASL “neutral”](#) and [“neutral”](#). The playlists contain, for the same mood, the same set of songs. For example the first song of *ASL “happiness”* is *Super Bass - Sign Language* and the first song of *“happiness”* is *Nicki Minaj - Super Bass*.

The complete flow of the system is the following:

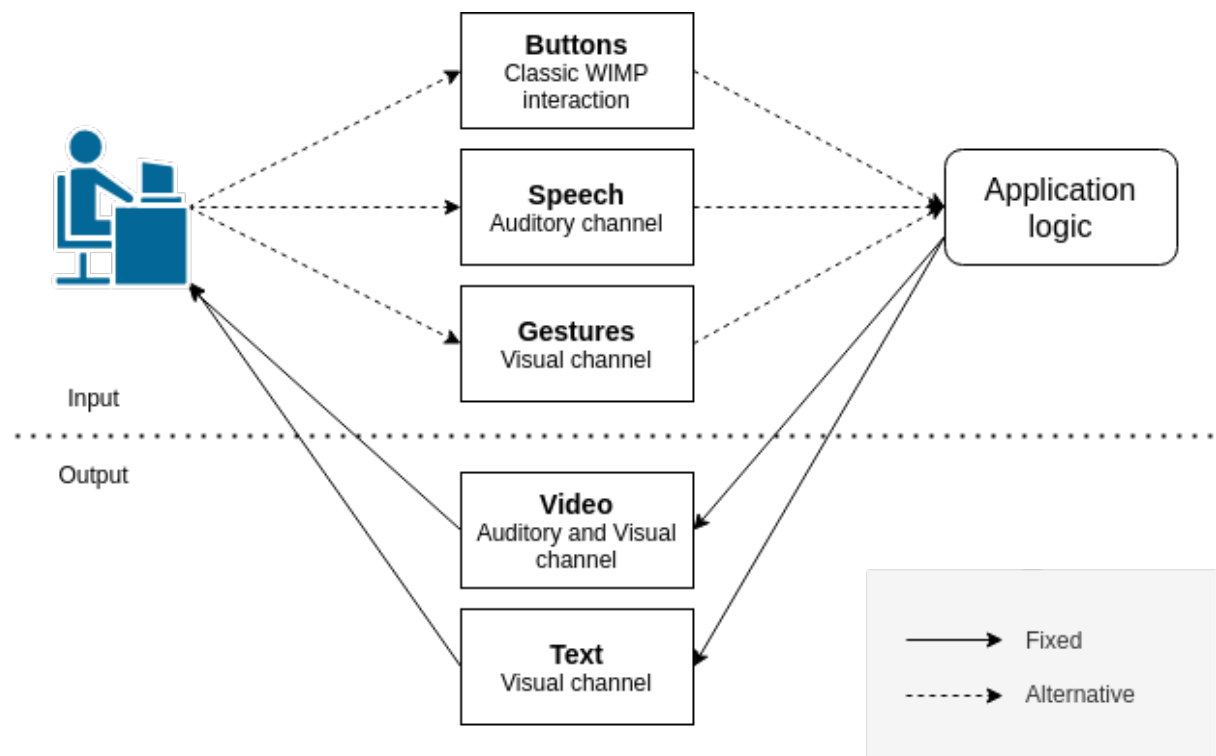


After being redirected to the playlist, the user can interact with the YouTube player in three different ways:

- using *gesture* commands,
- using *voice* commands,
- using the player *buttons*.

The following table shows all the available commands:

OPERATION	GESTURE	VOICE
Play	Hand open	Play
Pause		Pause/Stop
Volume up	Thumb up	Up
Volume down	Thumb down	Down
Next song	Index finger up	Skip/Next
Previous song	Victory	Previous
Mute	Fist	Mute
Unmute		Unmute



As shown in the table and the figure above, the three interaction *modalities* are alternative, and exploit Visual and Auditory channels.

API & SERVICES

To develop our system, we took advantage of existing services such as libraries and APIs. In details:

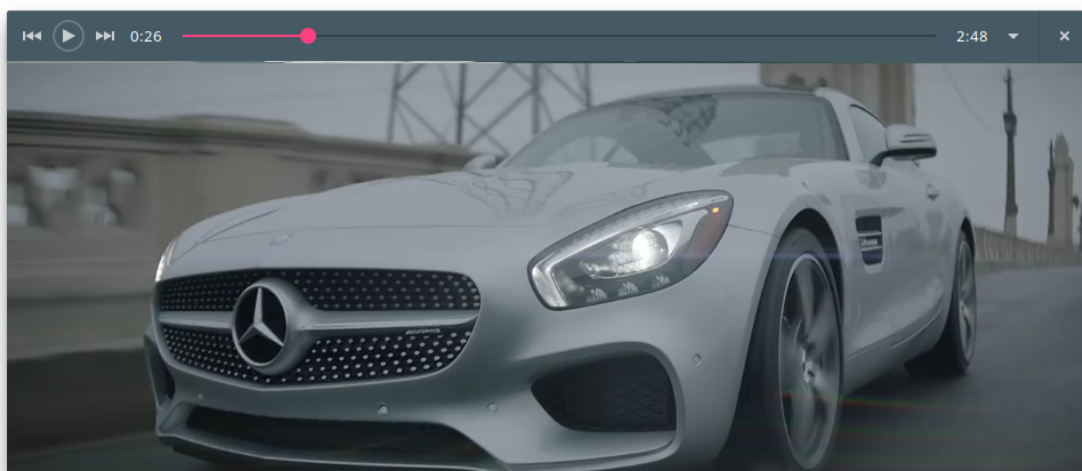
- **Programming language:** Python 3.8
- **YouTube player:** libvlc, Pafy, youtube-dl, gtk (pyGObject), pydbus, mutagen.
- **Database:** PostgreSQL (psycopg2).
- **Image acquisition:** openCV, gphoto2.
- **Face and Emotion Recognition:** Face++ Face Detection and Search.
- **Gesture Recognition:** Face++ Gesture Recognition.
- **Speech Recognition:** Microsoft Azure Cognitive Services (Speech to Text).

Let's see in detail the main services we used.

YouTube player

To avoid developing from scratch a YouTube player, we decided to adapt and integrate the one published [on GitHub by Vishnunarayan K.I.](#) The interface is quite **nice and simple**, and the player provides most of the features any user needs, including support of YouTube **playlists**, the most important one for our application.

The only one (but not negligible) **issue** was that this player exploits [GTK toolkit](#) for the GUI, which is quite hard to use under systems different from Linux. So, after a number of tries to adapt different environments and the application to each other, we decided to switch to Linux.



Face++

Face++ offers free and straightforward APIs for many tasks related to face, body and text recognition. We used [Gesture Recognition API](#) for gesture recognition and [Face Detection](#) and [Face Searching](#) APIs for face recognition.

The **first** API allows to simply recognize users' gestures by sending an image with an HTTP POST request and properly parsing the JSON response:

```
url = 'https://api-us.faceplusplus.com/humanbodypp/v1/gesture'
files = {
    'api_key': (None, util.get_property("gest_api_key")),
    'api_secret': (None, util.get_property("gest_api_secret")),
    'image_file': (img_name, open(img_name, 'rb')),
    'return_gesture': (None, '1'),
}
x = requests.post(url, files=files)
hands = json.loads(x.text)['hands']
print("hands:", hands)
gesture = Counter(hands[i]["gesture"]).most_common(1)[0][0]
```

The **second** API allows to detect a face in a picture sent in a POST request, recognize the user's expression, analyze the face and obtain a *face token* which can be locally stored in the database and associated to a user (in registration phase) or sent to the third API for recognition (in login phase):

```
url = 'https://api-us.faceplusplus.com/facepp/v3/detect'
files = {
    'api_key': (None, util.get_property("gest_api_key")),
    'api_secret': (None, util.get_property("gest_api_secret")),
    'image_file': (img_name, open(img_name, 'rb')),
    'return_attributes': (None, 'smiling,emotion'),
}
x = requests.post(url, files=files)
res = json.loads(x.text)
face_token = res['faces'][0]['face_token']
smile = res['faces'][0]['attributes']['smile']
emotions = res['faces'][0]['attributes']['emotion']
emotion = Counter(emotions).most_common(1)[0][0]
```


Optionally (during registration), the computed image features can be stored in a specific cloud gallery (called *faceset*):

```
url = 'https://api-us.faceplusplus.com/facepp/v3/faceset/create'
files = {
    'api_key': (None, util.get_property("gest_api_key")),
    'api_secret': (None, util.get_property("gest_api_secret")),
    'outer_id': (None, set_name),
    'force_merge': (None, '1'),
    'face_tokens': (None, ",".join(face_tokens))
}
x = requests.post(url, files=files)
```

The **third** API receives a new *face token* (not stored in the gallery) and checks if it matches with a face associated with an enrolled user, present in the required *faceset*. To do that, it returns the best matching token with a confidence score and a set of three thresholds:

- 1e-3: confidence threshold at the 0.1% error rate,
- 1e-4: confidence threshold at the 0.01% error rate,
- 1e-5: confidence threshold at the 0.001% error rate.

We decided to consider as correct the matches with confidence \geq the intermediate threshold:

```
url = 'https://api-us.faceplusplus.com/facepp/v3/search'
files = {
    'api_key': (None, util.get_property("gest_api_key")),
    'api_secret': (None, util.get_property("gest_api_secret")),
    'outer_id': (None, set_name),
    'face_token': (None, face_token)
}
x = requests.post(url, files=files)
res = json.loads(x.text)
threshold = res['thresholds']['1e-4']
confidence = res['results'][0]['confidence']
match_face_token = res['results'][0]['face_token']
```



We had to face a **problem** related to image quality: on an old computer with a not so good webcam, the service has some difficulties in recognizing gestures and faces, so we briefly tried some alternatives:

- [Filipe Foschini Borba's hand recognition](#): 1/13 correct recognitions,
- [Suhask A's HandGestureRecognition](#): 2/13 correct recognitions,
- [Kageyama's Hand Gesture Recognition CNN](#): 1/13 correct recognitions,
- [Suyash Gupta's hand gesture recognition](#): 0/13 correct recognitions,
- [William Mailhot's Gesture Recognition Using Keras](#): 2/13 correct recognitions,
- Face++: 2/13 correct recognitions.

Since the results were almost all the same for each model we tested, we concluded that the problem was mostly dependent on the webcam rather than on the model, so we decided to keep our system based on the *lighter* Face++ and to connect an external camera (using gphoto2 APIs):

```
camera = gp.Camera()
try:
    text = str(camera.get_summary())
    if "Nessuna fotocamera" in text or "No Image Capture" in text:
        self.thread_web =
threading.Thread(target=self.web_capture)
    else:
        self.thread_web =
threading.Thread(target=self.cam_capture)
except:
    self.thread_web = threading.Thread(target=self.cam_capture)
self.thread_web.start()
```

By looking at the code it is possible to understand that the integration of the gphoto2 library is not exclusive, we let the user choose which device to use. By doing so, the user can still decide to use the webcam of his computer if it has a good resolution or if he/she does not have an external camera to connect.

Microsoft Azure Cognitive Services

[Azure Cognitive Services](#) are a comprehensive family of AI services and APIs that you can easily integrate in your application in order to build an intelligent application.

The offered service covers a wide range of applications such as Decision, Language, Speech, Vision and Web Search. For our project we take advantage of the [Speech to Text API](#) that quickly **transcribes** audible speech into readable text in more than 40 languages and variants. Microsoft allows you to create a new *resource* that can be accessed by a pair (*subscription, region*) where the first one is one of the two keys generated by the system for the resource you created, and the second one is the region in which your service is deployed (e.g west-europe).



Once the application is running, it is possible to create a new **speech recognizer session** by inserting the correct (*subscription, region*) pair that are taken from a JSON configuration file we created:

```
speech_config = speechsdk.SpeechConfig(  
    subscription=util.get_property("speech_key_1"),  
    region=util.get_property("service_region")  
)
```

```
speech_recognizer = speechsdk.SpeechRecognizer(
    speech_config=speech_config
)

speech_recognizer.recognizing.connect(handle_event)
```

The command of the last row is used by the system to recognize the event signals and extract the keyword used to manipulate the player

```
if evt.result.text == "play":
    GLib.idle_add(self.youtube.play, None)
    self.show_info("Play")
```

Then, with the following command

```
speech_recognizer.start_continuous_recognition()
```

the system will start continuously listening and recognize the real-time audio in order to translate it into text.

In our application we used the translated text to manipulate the YouTube player and we decided to restrict the system to recognize only english voice commands (see the [table](#) for the available commands).

A **difficulty** we found while developing the Speech Recognition task was that if the audio was too high and the vocal part of the song started, then the system started recognizing both the words of the song and the commands spoken by the user that is using the system. In order to overcome this limitation we decided to set the initial volume to 75 over 100, this choice allowed us to solve the problem. Furthermore, if the user wants a more accurate result can use headphones, that will completely avoid this kind of problem.

EVALUATION

Unfortunately we had some limitations in testing the application since, due to time constraints and due to the Covid-19, we were not able to see a lot of people, so we decided to generate *three custom datasets* where users have various ages and english proficiency. Pictures and audios are divided as follows:

- **Identification_and_Emotion:** for each of the *31 users* we have at least 3 pictures with the three possible emotions (neutral, sad, happy); the dataset contains *103 pictures*.
- **Gesture_dataset:** for each of the *30 users* we have at least 6 pictures with the six possible gestures (see the [table](#)); the dataset contains *187 pictures*.
- **Speech_dataset:** for each of the *31 users* we have one audio in which they pronounce in a random mode and with a random pause between the words all the possible voice commands (see the [table](#)); the dataset contains *31 audios*.

The number of pictures and audios is not the same for each dataset since someone decided to give us only the audio, and others captured more than the minimum number of pictures we asked for. Moreover, each user used a different device in a different environment (i.e., not anyone had a notebook with a webcam at hand, and each webcam/smartphone can be different), so images have different qualities.

The evaluation was performed on all the different tasks as follows:

- **Open Set Identification:** the user does not claim an identity and the system has to assess if the taken picture matches the face of someone in the gallery.
 - The set of **Genuine Users** consists of 25 random *enrolled* users (all the genuine users were previously enrolled by using the application);
 - The set of **Impostors** consists of 6 random *not enrolled* users.
- **Emotion recognition:** the system recognizes the emotion from 103 pictures;
- **Semaphoric Gesture recognition:** the system recognizes the gesture from 187 pictures;
- **Speech recognition:** the system recognizes the speech from 31 audios.

In order to perform tests on all the above mentioned tasks, we developed a Python script. Let's see each test in detail.

Open Set Face Identification

In the open set identification task the goal of the system is to determine if the probe subject is in the database and, if so, who the probe subject is. Thus, there are different possible error situations, depending on the matcher and on the recognition threshold.

Possible scenarios are:

- **Correct Identification:** there are some individuals above the threshold and the individual with the highest score is the right one;
- **No Identification:** there isn't any individual above the threshold, thus we don't care about looking at the top individual;
- **False Alarm:** there are some individuals above the threshold but the individual with the highest score is not the right one.

We decided to evaluate the system on all this possible scenarios by using three common error measures for this kind of identification:

- **Detection and Identification Rate:** the number of times in which the system yields a correct result over all the trials with probes belonging to subjects in the database:

$$\text{DIR} = \text{correct_matches} / \text{n_genuine}$$

- **False Reject Rate:** the probability of a false reject at rank 1:

$$\text{FRR} = 1 - \text{DIR}$$

- **False Acceptance Rate:** the number of times in which the system returns an incorrect alarm running trials with probes belonging to subjects not in the database:

$$\text{FAR} = \text{false_alarms} / \text{n_impostors}$$

The following is the script we used to test the Identification:

```
# enrolling genuine users
for user in genuine:
    gallery = tokens[user][:2]
    for g in gallery:
        face.faceset([g], FACESET)
        time.sleep(1.1)

# trying to log genuine users
for user in genuine:
    gallery = tokens[user][:2]
    probe = tokens[user][2:]
    for p in probe:
        n_genuine += 1
        match = face.search(p, FACESET)
        if match in gallery:
            correct_matches += 1
            print("correct match")
        else:
            print(f"False reject: {user} mistaken for {rev_tokens[match]}")
        time.sleep(1.1)
```

We performed a first test with the intermediate threshold 71.8%, and we got:

- **DIR = 35/35 = 1**, this means that the system was able to recognize all the genuine users;
- **FRR = 1 - 1 = 0.0**, this means that the system had never reject a genuine user;
- **FAR = 8/18 = 0.44**, this means that the system had accepted users that were impostors.

By printing the **False Acceptance**, that are impostors recognized as genuine users we noticed that:

- 3 over 3 pictures of the user *andrea* were mistaken for an enrolled user;
- 2 over 3 pictures of the user *isabella* were mistaken for an enrolled user;
- 2 over 3 pictures of the user *davide* were mistaken for an enrolled user;
- 1 over 3 pictures of the user *camilla* were mistaken for an enrolled user.

By referring to the Doddington Zoo, that is a biometric menagerie that defines and labels user groups with animal species to reflect their behavior with the biometric systems, we deduced that *andrea*, *isabella* and *davide* can be identified as the **wolf**: it is good at impersonation and it is likely to generate a higher than average match score when compared to a stored biometric of a different person.

Since we got **DIR = 1**, we decided to run the test again but this time we used the higher threshold, **76.5%**. Let's see the result:

- **DIR = 5/33 = 0.15**, this means that the system was not able to recognize most of the genuine users;
- **FRR = 1 - 0.15 = 0.85**, this means that the system reject almost all the genuine users;
- **FAR = 18/20 = 0.9**, this means that the system had accepted just a few users that were impostors.

By looking at these results we can see that by increasing the threshold the system rejected most of the users even if they were genuine users, so we decided to restore the intermediate threshold.

It is important to highlight that the denominator of both the DIRs and FARs is not the same since some users captured more than the 3 required pictures and we divided the images in probes and gallery as follows:

- *genuine users*: for each user put the first 2 pictures in the gallery and use the remaining picture/s as probe/s;
- *impostors*: use all the pictures as probes.

Thus, the denominator depends on which users are chosen to be genuine or impostors. For example in the first case we have 22 genuine users with 3 pictures and 3 genuine users as follows: *alessia* with 8 pictures, *mel* 4 with pictures and *angelo* with 4 pictures (10 “additional” pictures).

Emotion Recognition

The emotion recognition task is a computer-generated paradigm for measuring the recognition of basic facial emotional expressions, in our project we focused on happiness, sadness and neutral expression.

We tested the system on all the 109 available pictures and we evaluated it based on:

- **Accuracy:** the number of correct classified images over all the samples in the gallery,

$$\text{Accuracy} = \text{correct_emotion} / \text{n_faces}$$
- **Confusion matrix:** it is a table that reports the number of *false positives*, *false negatives*, *true positives*, and *true negatives*.

The following is the script we used to test the Emotions:

```
for f in faces:
    gt_user, gt_emotion0 = os.path.splitext(os.path.basename(f))[0].split("_")[0:2]
    gt_emotion = EMOTIONS[gt_emotion0]
    face_token, smile, emotion = face.detect(f)
    tokens[gt_user] += [face_token]
    rev_tokens[face_token] = gt_user
    if gt_emotion == emotion:
        correct_emotions += 1
    print(f"{gt_user},{gt_emotion},{emotion}")
    time.sleep(1.1)
```

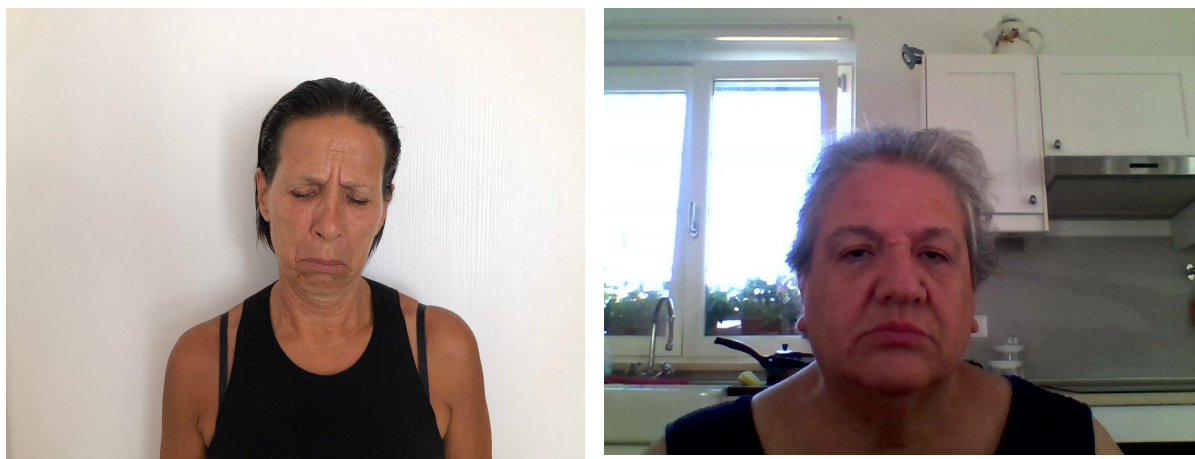
By doing tests we got an **Accuracy** of **69%** since the system was able to recognize **71 correct emotions** over **103 pictures**. Let's see the confusion matrix:

	ACTUAL CLASS			
		NEUTRAL	HAPPINESS	SADNESS
	NEUTRAL	29	2	25
	HAPPINESS	1	34	1
	SADNESS	1	0	8

Notice that each column has a different number of elements, since some testers sent more images for one of the expressions, in particular we had 31 neutral expressions, 36 happy expressions and 34 sad expressions. Moreover, the sum of all the elements of the matrix is not 103 since we had 2 instances that were classified with two classes that we had not considered, the instances were:

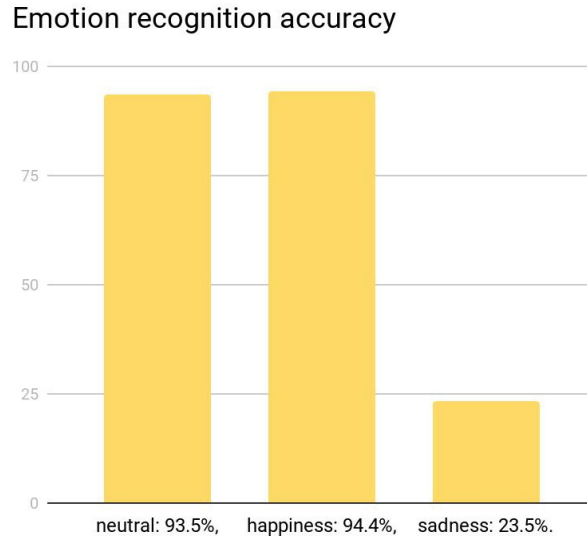
- **user:** alessandro, **actual_class:** sadness, **predicted_class:** fear;
- **user:** leonardo, **actual_class:** neutral, **predicted_class:** surprise.

By looking at the confusion matrix we noticed that the emotion that was more hard to recognize was **sadness**, that was often misclassified as neutral. We think that it strictly depends on how much the user that takes the picture “forced” his facial expression to be sad, or not (i.e., a sad expression that is not forced is really hard to recognize). In fact, for example, in the following images the first one was correctly classified as sad, while the second was considered neutral.



So we can conclude that the overall accuracy was quite good, but it changed considerably from class to class. In particular, for *sadness* it is very lower than for the other classes:

- *neutral*: 93.5%,
- *happiness*: 94.4%,
- *sadness*: 23.5%.



Gesture Recognition

In our system the gesture interaction is based on a *semaphoric* style, where semaphoric gestures means that the system employs a stylized dictionary of static or dynamic hand, or arm gestures.

The idea of using gestures to interact with the system comes from the fact that we want our application to be accessible even by deaf users. Many deaf people have the ability to speak, or learn how to speak, but many of them choose not to speak. In the American Deaf community, some people believe that using speech is a betrayal of Deaf pride and cultural values: to speak is to indicate that American Sign Language (ASL) is inferior. Of course, for some people, the choice is made for them through the circumstances of their education and training approach and for some, it is simply too late (see this [link](#)).

Sometimes, not speaking is a natural consequence of deafness; other times, it's a choice. In order to satisfy and respect all the choices, we decided to integrate gesture interaction so that people that are not able to speak, or simply don't want to, can use gestures to manipulate the player.

We tested the system on all the 187 available pictures and, also in this case, we decided to evaluate the system on:

- **Accuracy:** the number of correct classified images over all the samples in the gallery,

$$\text{Accuracy} = \text{correct_gestures} / \text{n_photo_gestures}$$

- **Confusion matrix:** it is a table that reports the number of *false positives*, *false negatives*, *true positives*, and *true negatives*.

The following is the script we used to test the Gestures:

```
for p in pictures:
    gt_user, gt_gesture0 = os.path.splitext(os.path.basename(p))[0].split("_")[0:2]
    gt_gesture = GESTURES[gt_gesture0]

    files = {...}
    x = requests.post(url, files=files)
    hands = json.loads(x.text)['hands']
    for h in hands:
        num_hands += 1
        gesture = Counter(h["gesture"]).most_common(1)[0][0]
        gestures_count[gt_gesture] += [gesture]
        if gt_gesture == gesture:
            correct_gestures += 1
        print(f"{gt_user},{gt_gesture},{gesture}")
```

By doing tests we got an **Accuracy** of **58.82%** since the system was able to recognize **110 correct gestures** over a total of **187 pictures**. We also realized that the total number of hands identified by the system was **252, 65** more than they should be. This means that the system found more than one gesture (hand) in some pictures.

We decided to consider the total number of images, rather than the number of detected hands, for two reasons:

1. For our task it is more relevant to find the highest number of correct classifications (maximize the true positives) than to avoid misclassifications (minimize the false positives),
2. In most cases the false positives (inexistent detected hands) were classified as unknown or as gestures we don't consider.

The images in the dataset are not the same number for each class since some users sent us both right and left hand gesture, in particular we had:

- *hand open*: 30 pictures,
- *thumb up*: 32 pictures,
- *thumb down*: 31 pictures,
- *index finger up*: 33 pictures,
- *victory*: 30 pictures,
- *fist*: 31 pictures.

Even if the number of pictures is not the same, the dataset is not *unbalanced*, so we can use the accuracy metric. Let's see the confusion matrix:

	ACTUAL CLASS						
		HAND OPEN	THUMB UP	THUMB DOWN	INDEX FINGER UP	VICTORY	FIST
PREDICTED CLASS	HAND OPEN	19	0	1	0	1	0
	THUMB UP	2	18	0	6	0	0
	THUMB DOWN	0	0	14	1	0	1
	INDEX FINGER UP	0	2	1	16	0	1
	VICTORY	0	0	0	2	22	0
	FIST	0	0	0	1	0	21
	OTHER	9	12	12	7	9	6
	UNKNOWN	7	12	14	11	8	12

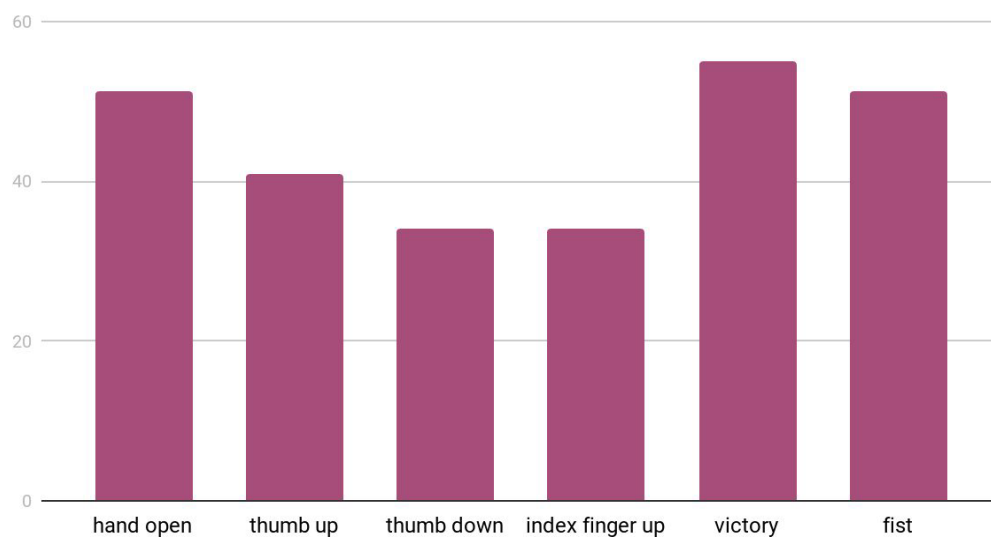
By looking at the confusion matrix and by calculating the accuracy for each gesture, we noticed that the one that was more hard to recognize was the **index finger up**. We think that this was due to the fact that a lot of users made a *wrong gesture* since they created an "L" with their hand, instead of raising only the index finger. In the following pictures it is possible to understand the difference:



This time the overall accuracy was not so good and it changed considerably from class to class. In particular, for *index finger up* and *thumb down* gestures, it is very low:

- *hand open* = 51.35%,
- *thumb up* = 40.90%,
- *thumb down* = 34.14%,
- *index finger up* = 34.04%,
- *victory* = 55%,
- *fist* = 51.21%.

Gesture recognition accuracy



By analyzing the results for each user and looking at the corresponding pictures, we can conclude that:

- the system performs better when the user took the picture of the gesture and himself, rather than giving us only the picture of the hand. For example:

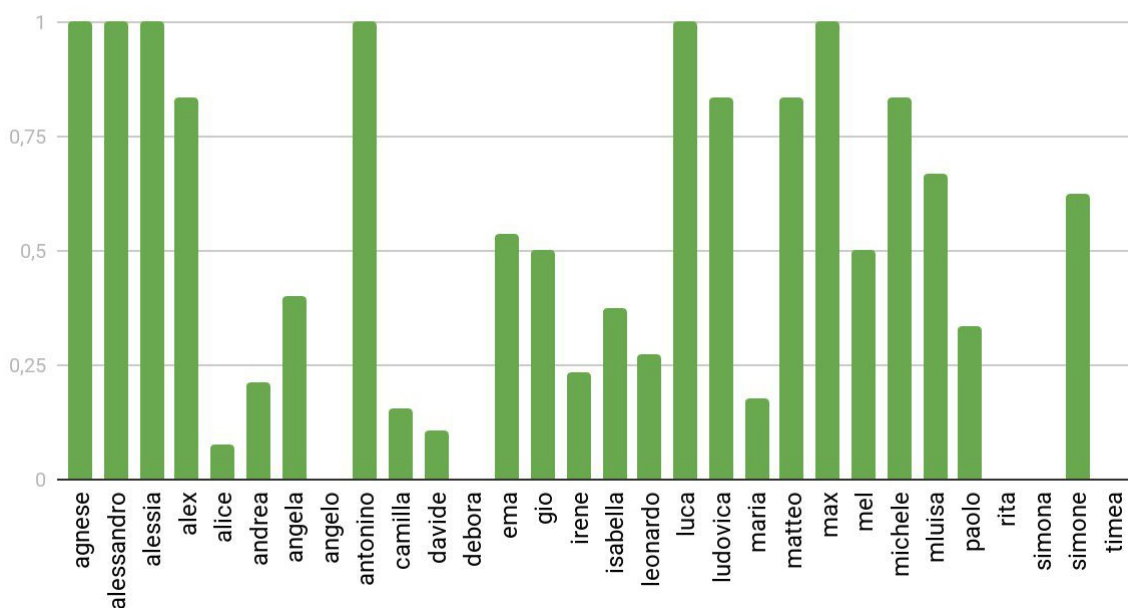


- we also noticed that the most recognized pictures were the ones taken with a good camera (such as the IphoneX or new smartphones);
- even with a not so good camera, if the background isn't cluttered, or if there is a high contrast between the hand and the background, the system is able to recognize the gestures with a good accuracy. For example:



- when the system succeeds in finding all the right gestures (many true positives) it does not look for other gestures where there aren't any (few false positives), for example:
 - *agnese*: 6 hands found and 6 correct gestures,
 - *alice*: 13 hands found and only 1 correct gesture.

Correct gestures over found hands per user



Speech Recognition

In order to have a vocal interaction with the system it is required speech recognition, in our case with semaphoric trigger words instead of a more articulated composition of them. For this kind of system problems are related to variation of the context, speakers and environment. Our system is actually an Isolated and Speaker Independence system, where Isolated means that commands are single words, while Speaker Independence means that the system was not intended for use by a single speaker but by any speaker.

Our system, since it is used to manipulate songs from your own pc that can be everywhere, have several adverse conditions:

- **environmental noise:** e.g. song played in background, noise from other people in the house, dogs barks and phone ring;
- **different microphones:** e.g. close-speaking and bad microphones;
- **altered speaking manners:** e.g. english pronunciation, shouting, whining and speaking too fast.

For this task we decided to use two common evaluation measures:

- **Accuracy:** the number of correct classified commands over all the samples in the gallery,

$$\text{Accuracy} = \text{correct_actions} / \text{n_utterances}$$

Arguably the most important measurement of accuracy is whether the desired end result occurred. For example, if the user said "skip," the engine returned "skip," and the "SKIP" action was executed, it is clear that the desired end result was achieved.

- **Recognition accuracy:** the number of times in which the engine recognized the utterance exactly as spoken. This measure of recognition accuracy is expressed as a percentage and represents the number of utterances recognized correctly out of the total number of utterances spoken.

$$\text{Recognition Accuracy} = \text{correct_utterances} / \text{n_utterances}$$

This time we didn't use a Python script, but we tested the application in real-time. We tested the system on all the *31 available registrations* that contained all the voice commands, so the *total number of voice commands* was 310.

By doing tests we got an **Accuracy** of **81.29%** since the system was able to perform **252 correct actions** over a total of **310 utterances**, and a **Recognition Accuracy** of

79.68% since the system was able to recognize **247 correct utterances** over a total of **310 utterances**.

The following table shows, for each utterance, the number of time it was correctly recognized and performed by the system:

VOICE COMMAND		RECOGNIZED	NOT RECOGNIZED	CORRECT ACTION	INCORRECT ACTION
	DOWN	26	5	26	5
	MUTE	29	2	29	2
	NEXT	30	1	30	1
	PAUSE	17	14	17	14
	PLAY	30	1	30	1
	PREVIOUS	28	3	28	3
	SKIP	29	2	29	2
	STOP	30	1	30	1
	UNMUTE	18	13	23	8
	UP	26	5	26	5

By looking at the table we noticed that **next**, **play** and **stop** are always recognized (except for one case), whereas **pause** was the utterance that was more hard to recognize. It was often recognized as *pose*, *post*, *bulls*, *those* and others. But also *down* and *up* were hard to recognize, in fact the first one was often recognized as *done*, while the second one was often recognized as *app* or *out*.

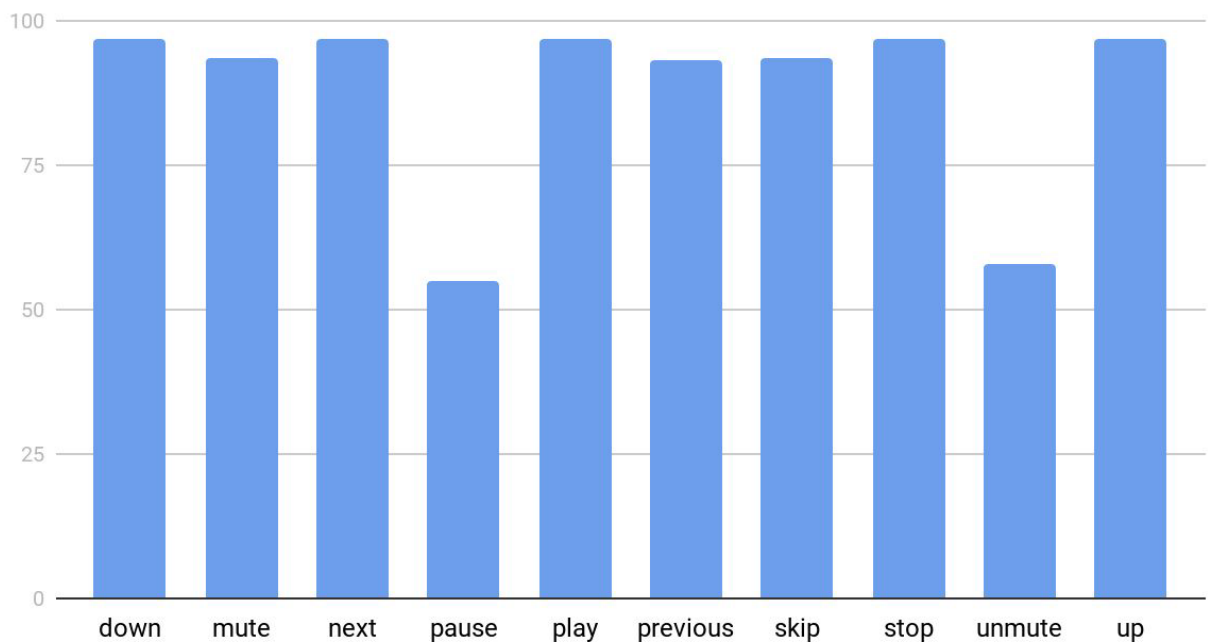
Next, play and stop are not 100% correctly recognized since 1 out of the 31 audios had a too low volume and the system was not able to recognize what the user said. Moreover, we can say that our system is quite flexible, since in some cases it performed the correct action even if the command was not perfectly recognized (e.g. when it understands *I'm mute* instead of *unmute*).

For this task the overall accuracy was very good and it does not change a lot from command to command. The only one utterance that was a little bit problematic was *pause*, where the accuracy is quite lower than for the other voice commands:

- *down* = 96.77%,
- *mute* = 93.54%,
- *next* = 96.77%,
- *pause* = 54.84%,
- *play* = 96.77%,
- *previous* = 93.33%,
- *skip* = 93.54%,
- *stop* = 96.77%,
- *unmute* = 58.06%,
- *up* = 96.77%.

For which concern the Recognition Accuracy, we got the same results except for the *unmute* command, for which we got 74%. This happens because we implemented the *unmute* operation using the *in* operator, so that one can also toggle the volume by saying just *mute* more times.

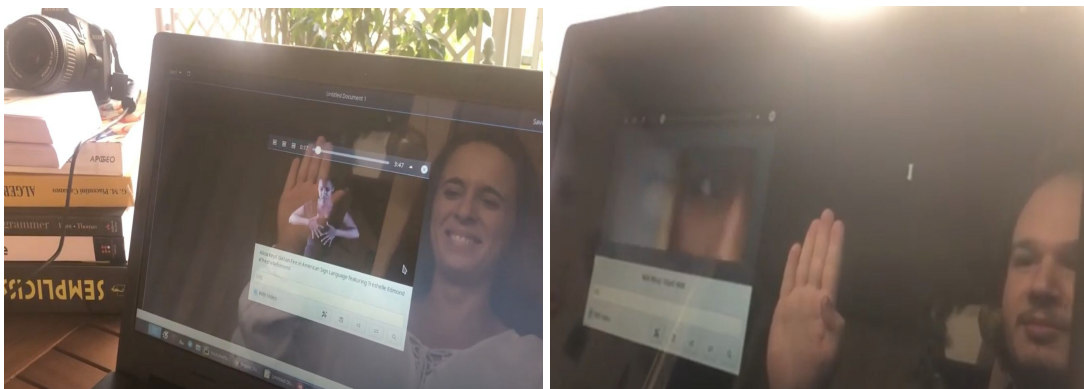
Speech recognition accuracy



DEMO

Both the demo are available at the following links: demo [user deaf](#) and [demo user \(not deaf\)](#). The two videos are organized as follows:

- **Demo deaf user:** the user is not already enrolled.
 1. The user performs the enrollment with the username “irene” and checks the “deaf” option.
 2. The system correctly recognized the mood of the user as neutral (the correct one is the first taken picture).
 3. The user try the following commands:
 - a. Gesture “volume up” -> not recognized -> ignored by the system
 - b. Gesture “pause” -> correctly recognized
 - c. Gesture “volume down” -> correctly recognized
 - d. Gesture “play” -> correctly recognized
 - e. Gesture “next” -> correctly recognized
 - f. Gesture “volume down” -> wrongly recognized as mute -> mute
 - g. Gesture “unmute” -> correctly recognized
- **Demo user (not deaf):** the user is already enrolled in the database as “giovanni”.
 1. The system automatically logs in the user with the correct identity.
 2. The system correctly recognized the mood of the user as happy.
 3. The user tried the following commands:
 - a. Gesture “pause” -> correctly recognized
 - b. Utterance “play” -> correctly recognized
 - c. Utterance “skip” -> correctly recognized
 - d. Gesture “previous” -> correctly recognized
 - e. Utterance “next” -> correctly recognized
 - f. Utterance “mute” -> correctly recognized
 - g. Gesture “unmute” -> correctly recognized
 - h. Utterance “stop” -> correctly recognized



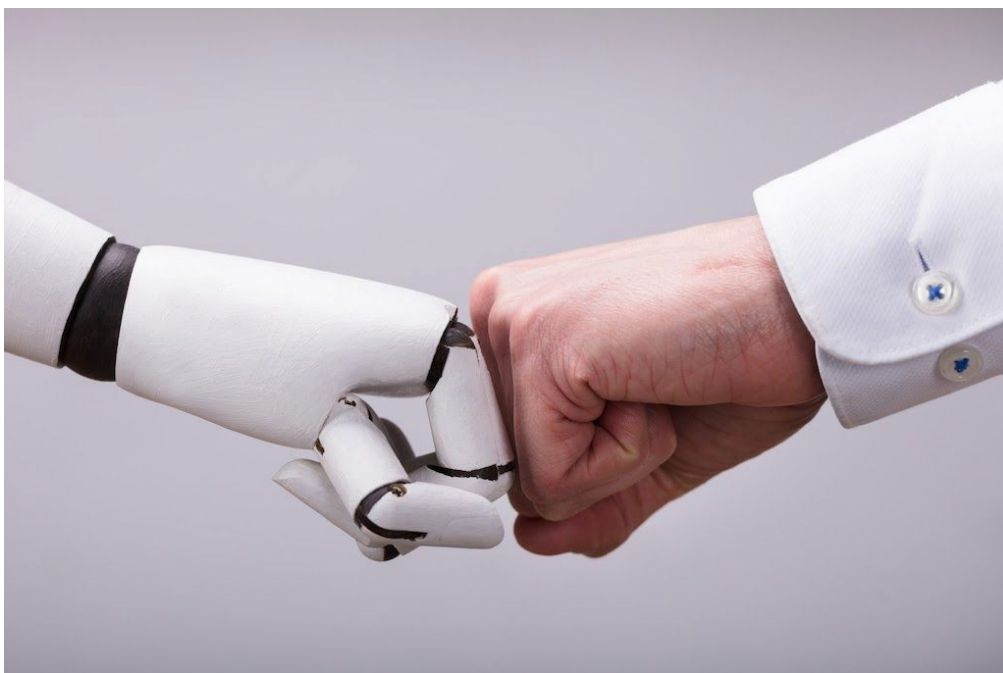
CONCLUSIONS

To sum up, the results obtained by the application are quite good. In particular, the voice interaction task achieved an accuracy of **81.29%** and a recognition accuracy of **79.68%**. The second task that performed well is the Open Set Identification that got a Detection Identification Rate of **1**, but it does not reach a good score of False Acceptance Rate, that was actually **0.44**. This last score is not a real problem for us since the application does not contain any personal data and it will be installed on the pc of the user. Because of that we think that there will not be a lot of impostors, so the application does not require a high level of security. Thus, we prefer to have a high DIR even if the FAR is not so good. Also the emotion recognition result was quite good since it reached the **69%** of accuracy. Moreover the gesture interaction task does not perform very well, in fact the maximum accuracy achieved was **58.82%**.

By looking at the results we also realized that the application could be improved in some aspects, in particular:

- **More intuitive gestures:** use a custom and more complex model in order to recognize more intuitive gestures that better represents the commands that are performed on the player;
- **Usability:** could be an interest point to carry out a survey to understand if the more intuitive gestures are good for the mute community too, and ask them to real-time try the application;
- **More responsiveness:** for which concern the gesture interaction task the application is a little bit slow.

In general we are satisfied with the results since the application satisfies its requirements.



References

- YouTube player: <https://github.com/vn-ki/YouTubePlayer>
- GTK toolkit: <https://www.gtk.org>
- youtube-dl: <https://github.com/ytdl-org/youtube-dl>
- openCV: <https://opencv.org>
- gphoto2: <http://www.gphoto.org>
- PostgreSQL: <https://www.postgresql.org>
- Face++ (Gesture Recognition):
<https://www.faceplusplus.com/gesture-recognition/>
- Face++ (Face Detection): <https://www.faceplusplus.com/face-detection/>
- Face++ (Face Searching): <https://www.faceplusplus.com/face-searching/>
- Microsoft Azure Speech To Text:
<https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/>