SAPIENZA
UNIVERSITÀ DI ROMA

Dipartimento di Informatica

M.Sc. in Computer Science

Università degli Studi di Roma, "La Sapienza"

BROWSER HANDLER

Multimodal Interaction Project

Berti Andrea - 1831511

Pappa Massimiliano – 1661249

Supervised by Prof. Maria De Marsico

Contents

1 Introduction

In this document will be presented the project “Browser Handler”. The aim of the project is to develop an efficient system to navigate hands-free in a web browser, thanks to a flow which provides the detection of a hand or fist to enable speech commands for managing the actions on a predefined web browser.

1.1 Technologies

1.1.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms.

OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

1.1.3 Python

Python is a popular general-purpose programming language that can be used for a wide variety of applications. It includes high-level data structures, dynamic typing, dynamic binding, and many more features that make it as useful for complex application development as it is for scripting or "glue code" that connects components together. It can also be extended to make system calls to almost all operating systems and to run code written in C or C++. Due to its ubiquity and ability to run on nearly every system architecture, Python is a universal language found in a variety of different applications.

1.1.4 Azure

Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying and managing applications and services through Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems.

The services we used are the so-called Cognitive Services. Microsoft Cognitive Services let you build apps with powerful algorithms to see, hear, speak, understand and interpret our needs using natural methods of communication. These services are: Decision, Language, Speech, Vision and Web search; for our project we will use the Speech to Text APIs of the speech cognitive service.

This speech service feature allows to quickly and accurately transcribe audio to text in more than 85 languages and variants. It is also possible to customize models to enhance accuracy for domain-specific terminology.

1.1.5 Webbrowser

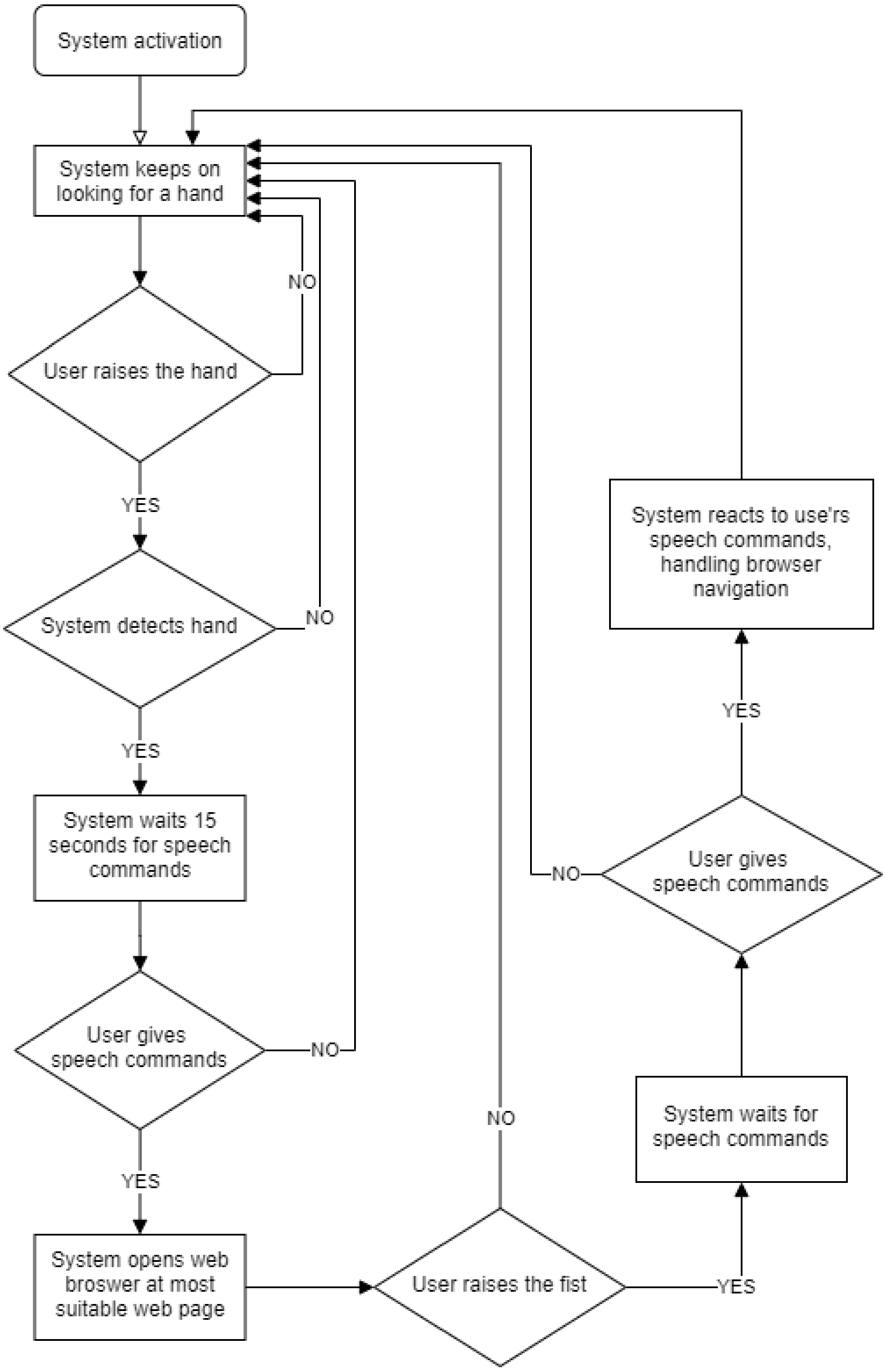
It is a Python Standard Library. The Webbrowser module provides a high-level interface to allow displaying web-based documents to users.

Under Unix, graphical browsers are preferred under X11, but text-mode browsers will be used if graphical browsers are not available or an X11 display isn't available. If text-mode browsers are used, the calling process will block until the user exits the browser.

If the environment variable `BROWSER` exists, it is interpreted as the *os.pathsep*-separated list of browsers to try ahead of the platform defaults. When the value of a list part contains the string *%s*, then it is interpreted as a literal browser command line to be used with the argument URL substituted for *%s*; if the part does not contain *%s*, it is simply interpreted as the name of the browser to launch.

The script *webbrowser* can be used as a command-line interface for the module. It accepts a URL as the argument.

2 System Flow


System activation
System keeps on looking for a hand
User raises the hand
NO
YES
System detects hand
NO
YES
System waits 15 seconds for speech commands
User gives speech commands
NO
YES
System opens web broswer at most suitable web page
User raises the fist
NO
YES
System waits for speech commands
User gives speech commands
NO
YES
System reacts to use'rs speech commands, handling browser navigation

In brief: when the system is activated, the user can decide whether to raise the hand or do nothing.

Once he/she raises the hand and the system detects it, it will start listening to the user's speech command/query for 15 seconds. If the user does not say anything, the system will keep on looking for a hand to detect, otherwise it will catch the speech commands and translate them to text; then it will open the predefined web browser at the most suitable web page for the user (according to the speech commands he/she gave), and keeps on waiting to detect a hand or a fist.

Once the user raises the fist and the system detects it, it will start listening to the user's speech commands; if the user gives a speech command, the system will either navigate through the opened browser tabs, close the current tab or close the browser, according respectively to the spoken commands. If the user does not say anything, the system keeps on waiting to detect a hand or a fist.

3 Project Development

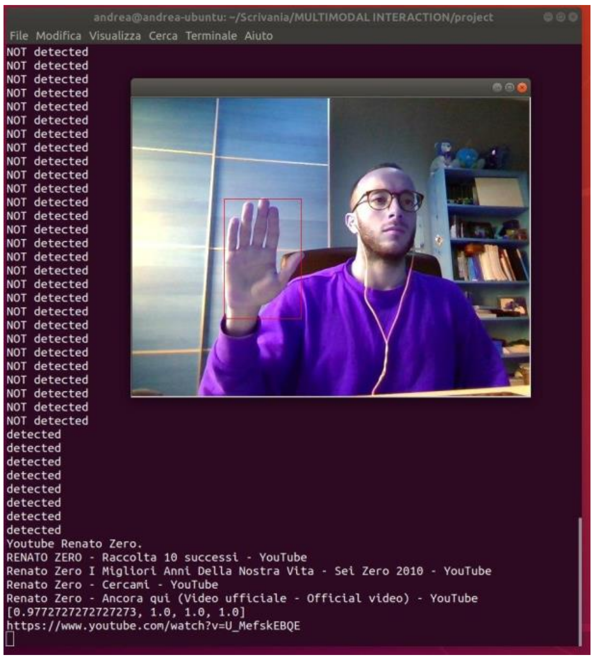
3.1 System structure

The system is structured so that all processes are executed in a sequential order. Firstly, the hand or fist detection process runs as soon as the system is activated. Once this process outputs some information (and so it has detected something), the system runs the speech recognition process and translates the user's speech input to text, through the Microsoft Azure Cognitive Service.

Once the system has the commands translated to text, it processes it and through a matching mechanism it will open the most suitable web page for the user, from the predefined web browser.

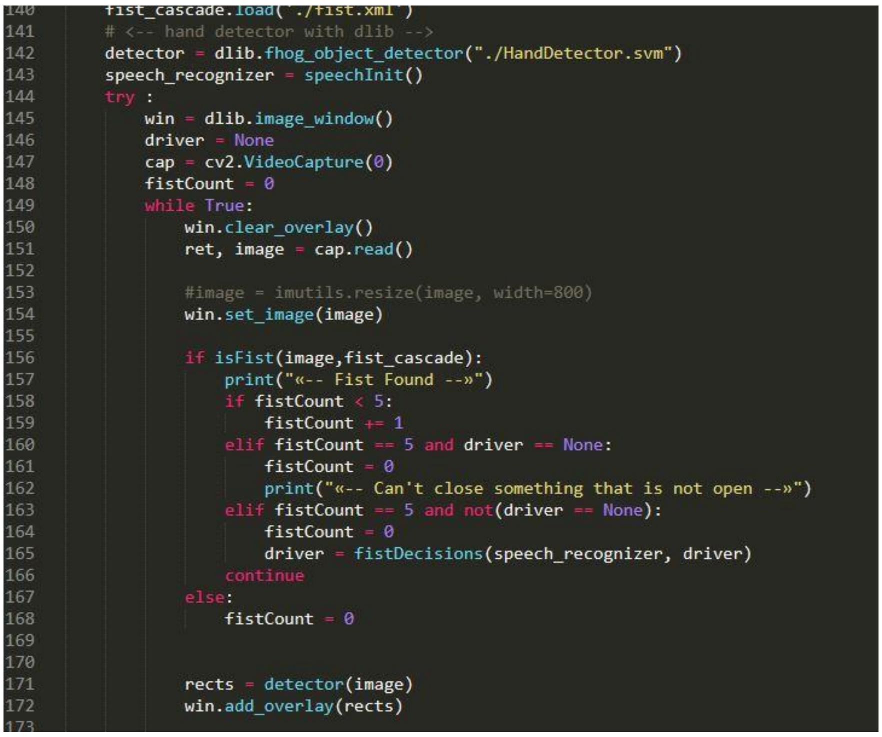
Then, if the user raises the fist, the system runs the fist decision process and starts listening to what the user says again. He/she can either navigate through the browser opened tabs, close the current tab or close the browser and so to exit the navigation.

3.2 Hand detection


andrea@andrea-ubuntu: ~/Scrivania/MULTIMODAL INTERACTION/project
File Modifica Visualizza Cerca Terminale Aiuto

```
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
detected
detected
detected
detected
detected
detected
detected
detected
Youtube Renato Zero.
RENATO ZERO - Raccolta 10 successi - YouTube
Renato Zero I Migliori Anni Della Nostra Vita - Sei Zero 2010 - YouTube
Renato Zero - Cercami - YouTube
Renato Zero - Ancora qui (Video ufficiale - Official video) - YouTube
[0.9772727272727273, 1.0, 1.0, 1.0]
https://www.youtube.com/watch?v=U_MefskEBQE
```

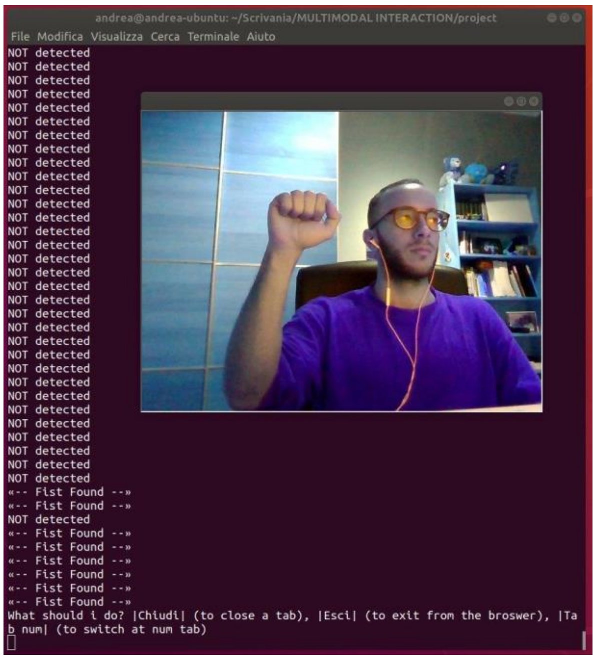
The system, once it is activated, opens a window handled by the OpenCV library capturing the user's webcam live image. If the user raises the hand, the system tries to detect it using a pretrained dlib model (a Python library), which employs the SVM technique. Once the hand is detected, the system draws the bounding box around it and starts running the speech recognition process.



```
fist_cascade.load('./fist.xml')
# <-- hand detector with dlib -->
detector = dlib.fhog_object_detector("./HandDetector.svm")
speech_recognizer = speechInit()
try :
    win = dlib.image_window()
    driver = None
    cap = cv2.VideoCapture(0)
    fistCount = 0
    while True:
        win.clear_overlay()
        ret, image = cap.read()

        #image = imutils.resize(image, width=800)
        win.set_image(image)

        if isFist(image,fist_cascade):
            print("«-- Fist Found --»")
            if fistCount < 5:
                fistCount += 1
            elif fistCount == 5 and driver == None:
                fistCount = 0
                print("«-- Can't close something that is not open --»")
            elif fistCount == 5 and not(driver == None):
                fistCount = 0
                driver = fistDecisions(speech_recognizer, driver)
            continue
        else:
            fistCount = 0


        rects = detector(image)
        win.add_overlay(rects)
```

- *Dlib.fhog_object_detecror* represents the dlib function used for a histogram-of-oriented-gradients based object detector;
- *Cv2.VideoCapture* represents the OpenCV function used to capture the user's webcam live image;
- *Rects = detector(image)* is used to draw the bounding box around the detected hand, from the dlib detector;
- *Dlib.image_window().clear_overlay()* is used to clear all previous bounding boxes, images and layers created in that instance;
- *Dlib.image_window().add:overlay(rects)* is used to add the layer with the bouding box around the hand just detected.

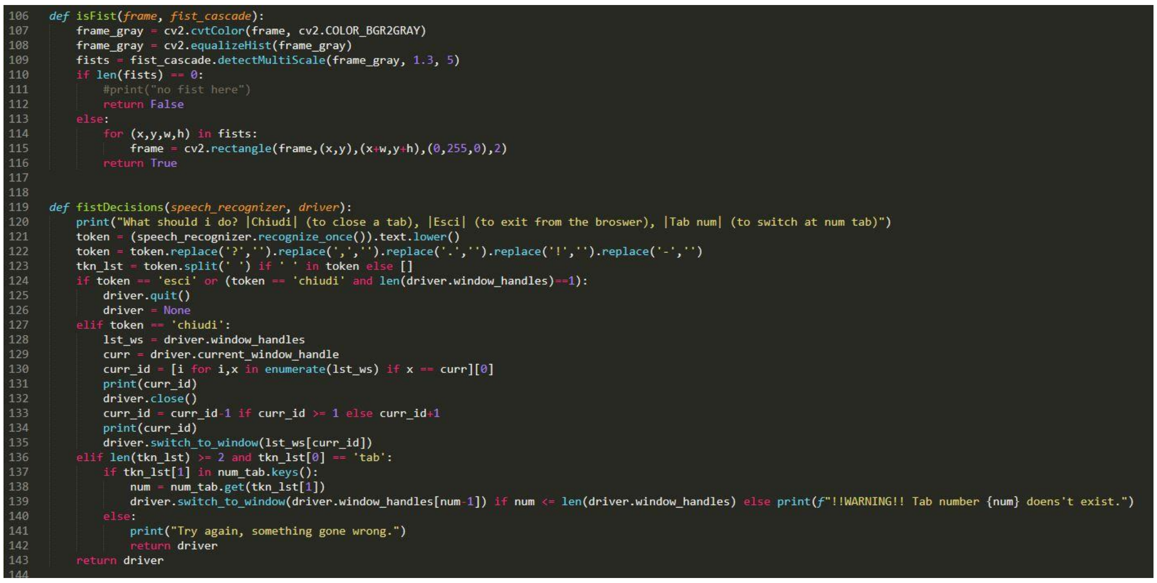
3.3 Fist detection


andrea@andrea-ubuntu: ~/Scrivania/MULTIMODAL INTERACTION/project
File Modifica Visualizza Cerca Terminale Aiuto

```
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
NOT detected
«-- Fist Found --»
«-- Fist Found --»
NOT detected
«-- Fist Found --»
«-- Fist Found --»
«-- Fist Found --»
«-- Fist Found --»
«-- Fist Found --»
«-- Fist Found --»
What should i do? |Chiudi| (to close a tab), |Esci| (to exit from the broswer), |Ta
b num| (to switch at num tab)
```

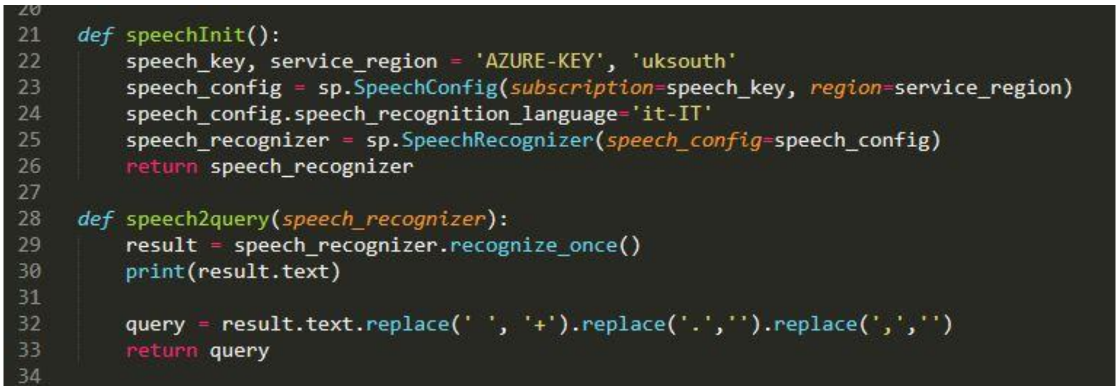
Once the browser is opened, the user can raise the fist in order to navigate through it. The system keeps on trying to detect a hand or a fist while it is activated, so if the user raises the fist, it is detected thanks to an OpenCV Cascade Classifier for object detection, pretrained on detecting fists. It uses the Viola-Jones model, so the fist is detected using Haars Cascade Features. Once the fist detection process has ended, the system starts running the speech recognition process.

- *Cv2.CascadeClassifier()* represents the opencv function to call the cascade classifier class for fist detection;
- *Cv2.CascadeClassifier().detectMultiScale()* is used to detect fists of different sizes in the input image and then returned as a list of rectangles.



```python
def isFist(frame, fist_cascade):
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_gray = cv2.equalizeHist(frame_gray)
    fists = fist_cascade.detectMultiScale(frame_gray, 1.3, 5)
    if len(fists) == 0:
        #print("no fist here")
        return False
    else:
        for (x,y,w,h) in fists:
            frame = cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
        return True


def fistDecisions(speech_recognizer, driver):
    print("What should i do? |Chiudi| (to close a tab), |Esci| (to exit from the broswer), |Tab num| (to switch at num tab)")
    token = (speech_recognizer.recognize_once()).text.lower()
    token = token.replace('?','').replace(',','').replace('.','').replace('!','').replace('-','')
    tkn_lst = token.split(' ') if ' ' in token else []
    if token == 'esci' or (token == 'chiudi' and len(driver.window_handles)==1):
        driver.quit()
        driver = None
    elif token == 'chiudi':
        lst_ws = driver.window_handles
        curr = driver.current_window_handle
        curr_id = [i for i,x in enumerate(lst_ws) if x == curr][0]
        print(curr_id)
        driver.close()
        curr_id = curr_id-1 if curr_id >= 1 else curr_id+1
        print(curr_id)
        driver.switch_to_window(lst_ws[curr_id])
    elif len(tkn_lst) >= 2 and tkn_lst[0] == 'tab':
        if tkn_lst[1] in num_tab.keys():
            num = num_tab.get(tkn_lst[1])
            driver.switch_to_window(driver.window_handles[num-1]) if num <= len(driver.window_handles) else print(f"!!WARNING!! Tab number {num} doens't exist.")
        else:
            print("Try again, something gone wrong.")
            return driver
    return driver
```

3.4 Speech recognition

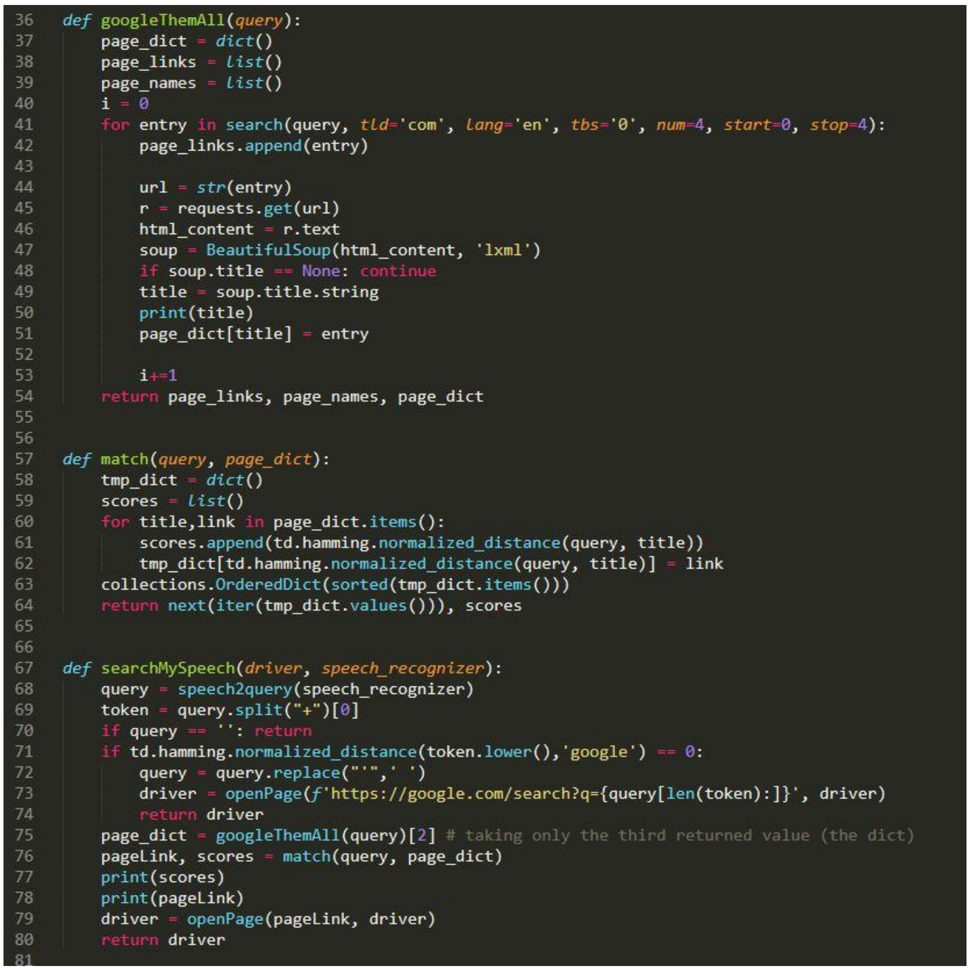


```python
def speechInit():
    speech_key, service_region = 'AZURE-KEY', 'uksouth'
    speech_config = sp.SpeechConfig(subscription=speech_key, region=service_region)
    speech_config.speech_recognition_language='it-IT'
    speech_recognizer = sp.SpeechRecognizer(speech_config=speech_config)
    return speech_recognizer

def speech2query(speech_recognizer):
    result = speech_recognizer.recognize_once()
    print(result.text)

    query = result.text.replace(' ', '+').replace('.','').replace(',','')
    return query
```

The speech recognition process runs only once a hand or a fist is detected. In order to work properly, we need to initialize our Microsoft Azure Cognitive Service. We set the language to Italian only because otherwise our bad pronunciation of English words could mislead the tests on the system speech recognition. Once the speech service is initialized, the system can translate into text what the user says in that temporal window and pass the query to the other functions.

- *speechInit()* is the function used to initialize Microsoft Azure Cognitive Services, using a subscription key for the speech service and the Italian language as the speech recognition language.

- *Speech2query()* is the function used to perform recognition in a blocking (synchronous) mode. Returns after a single utterance is recognized. The end of a single utterance is determined by listening for silence at the end or until a maximum of 15 seconds of audio is processed. The task returns the recognition text as result.



```python
def googleThemAll(query):
    page_dict = dict()
    page_links = list()
    page_names = list()
    i = 0
    for entry in search(query, tld='com', lang='en', tbs='0', num=4, start=0, stop=4):
        page_links.append(entry)

        url = str(entry)
        r = requests.get(url)
        html_content = r.text
        soup = BeautifulSoup(html_content, 'lxml')
        if soup.title == None: continue
        title = soup.title.string
        print(title)
        page_dict[title] = entry

        i+=1
    return page_links, page_names, page_dict


def match(query, page_dict):
    tmp_dict = dict()
    scores = list()
    for title,link in page_dict.items():
        scores.append(td.hamming.normalized_distance(query, title))
        tmp_dict[td.hamming.normalized_distance(query, title)] = link
    collections.OrderedDict(sorted(tmp_dict.items()))
    return next(iter(tmp_dict.values())), scores


def searchMySpeech(driver, speech_recognizer):
    query = speech2query(speech_recognizer)
    token = query.split("+")[0]
    if query == '': return
    if td.hamming.normalized_distance(token.lower(),'google') == 0:
        query = query.replace("'",' ')
        driver = openPage(f'https://google.com/search?q={query[len(token):]}', driver)
        return driver
    page_dict = googleThemAll(query)[2] # taking only the third returned value (the dict)
    pageLink, scores = match(query, page_dict)
    print(scores)
    print(pageLink)
    driver = openPage(pageLink, driver)
    return driver
```

Once the service is initialized and the speech is recognized, our system processes few more functions in order to get the best result to match the user's query. The user, in the 15 seconds temporal window, can either say something to look for straight from the web (as a precise website name and path) or use the keyword "google" in order to look for the query as a google search input.

The system works in a way that once the speech is recognized, the resulting text is then matched to a series of google results title (the first 4 in this case), through a scraping process made by Beautiful Soup Python library.

These results are then matched to the user's query so that by measuring the Hamming Distance among them, the most suitable resulting title is found, and it is then linked to the user in the web page output form. The same mechanism works for the query processed after the "google" keyword, but of course it results quicker than the other scraping process, as it is only a matter of looking up on Google what the user is saying.

- *googleThemAll()* is used to retrieve the first 4 Google title results, by scraping the page using Beautiful Soup, which is a Python library useful for pulling data out of HTML and XML files. The results of the parser are stored into a dictionary which becomes handy in the next functions.
- *Match()* is used to match the user's query with the titles retrieved by the previous function. The best title is returned after using the Hamming distance (so the number of bit positions in which they differ) among all of the strings, to get the most probable useful link.
- *searchMySpeech()* is used to run all the speech recognition parts and also to define "google" as the keyword command to start the Google search, with the rest of the query as argument of the query.

4 Multimodal Fusion & Fission

4.1 Multimodal Fusion

The process of integrating information from various input modalities and combining them into a complete command is referred as multimodal fusion. Of course, the most important part of the fusion is the input interpretation process, which involves 4 phases, corresponding to the main architectural levels of a multimodal system: the acquisition, the recognition, the integration and the decisions phases. Although the acquisition, recognition and decisions are consecutive phases, the same does not occur for the integration phase, where the fusion process takes place, because it can take place prior to the other phases.

This integration process can take place at acquisition level (which consists in mixing two or more signals), at recognition level (which consists in merging the outcomes of each recognizer by using integration mechanisms), at decision level (which means merging directly the semantic information that are extracted) and at a hybrid multi-level fusion (in which the integration of input signals is distributed among the acquisition, recognition and decision level).

In our project, in order to make the system work properly, we decided to use inputs fusion at the decision level, as it is mostly suitable for modalities that differ both in their nature and in the time scale (gesture and speech in this case). So, each input modality is separately recognized and interpreted; in particular, in this case, the input modalities are interpreted sequentially as the system decides whether to call some functions rather than others, depending on the hand and fist detection results.

4.2 Multimodal Fission

Multimodal fission refers to the information output arrangement and organization. It determines the best modalities and actions based on the given context and evaluation of events. So basically, multimodal fission is the process of physically acting or reacting to the data and information provided. Some issues to consider in order to design and configure fission are the information structure, intonation and emphasis for the output by speech, the spatio-temporal coordination of pieces of information for visual outputs, the design of appropriate output for each kind of modality and the synchronization of the different outputs modalities.

Multimodal fission process needs to consider what information has to be presented according to the interaction context and how the information can be presented in terms of information structure, the chosen modalities for the output and then coordination and synchronization.

For this project, we decided to adopt the WWHT (What, Which, How, Then) Model in order to sequentially study and analyze the best results to provide to the user. It is a conceptual model for multimodal presentation of information and for the design of the multimodal system output. For our project we considered this model:

- *What* is the information to present? The most suitable web page for the user's speech request;
- *Which* modality or modalities combination should we use to present this information? We decided to go with visual communication, as the desired output would be a visually presented web page;
- *How* to present the information using the chosen modalities? The system opens the predefined web browser and links the user to the desired page;
- *Then*, how to handle the evolution of the resulting presentation? The user can either continue querying, navigate through the browsers' tabs or close the tab and the browser to end the navigation.

5 Conclusions

5.1 Future Implementations

Having more time, we came with few ideas to improve and enlarge the implementation of the Browser Handler project.

It would be possible to use PCA (Principal Component Analysis) to replace the Hamming distance while looking for the most suitable title result. For example, instead of taking the first 4 titles, we can take the first 20 titles and plot them in a vectorial space and, through the cosine similarity, to maximize the precision of the matching phase.

We can also use embeddings, useful for finding results based on context; in this way, we are able to work with precise time windows for chronological purpose and the user's browser history, to refine the matching phase.

It would be a good idea to add a biometric verification system, for example through face recognition and identification, so that only the owner of that computer will be able to activate the "browser handler". It would be possible also to implement a system that starts recognizing user's gestures or listening to user's speech commands, only while a determined user is in front of that computer and stops if the user is momentarily away from it.

5.2 General Conclusions

In conclusion, we think that the "Browser Handler" project is a good example of a multimodal system overall. It takes different inputs through different modalities and processes them sequentially to give the user a visual interpretation of the results of their fusion.

In our studies, we were concerned about the precision of the hand detection, but it turned out to be precise and accurate. Although, the fist detection is a little bit less precise as (probably) the pretrained classifier was not trained too well, but for our purpose it works just fine.

Therefore, we believe that this system could be useful for those cases in which the user is not so familiar with the keyboard or does not want to use it for the navigation on the web. It could be possible to improve it even further by adding more functionalities, more gestures or more modalities, but with the time schedule we had, we though this system could be a good compromise between a simple, useful and efficient multimodal system.