به نام خدا

کنترل رایانه با صوت و صورت

آزمایشگاه سخت‌افزار

گروه ۱۱

اعضای گروه:

علیرضا عیسوند

محمد محمدی

امیرمحمد قاسمی

فهرست مطالب

مقدمه

در این پروژه یک ماژول طراحی شده است که جایگزینی برای موس و کیبورد کامپیوتر بوده و این کار با استفاده از فرمان‌های صوتی و حرکات صورت انجام می‌گیرد. در این پروژه یک ماژول به سیستم کاربر اضافه می‌شود که دارای یک میکروفون و یک دوربین بوده که با کمک آن‌ها از کاربر به صورت صوتی یا تصویری ورودی دریافت می‌کند. همچنین یک اپلیکیشن ساده بر روی سیستم قرار می‌گیرد تا کاربر از این ماژول استفاده و آن را کنترل کند. سپس براساس فرمان‌هایی که ماژول از کاربر می‌گیرد موس و کیبورد و بخش‌هایی دیگر از سیستم اصلی را کنترل می‌کند و جایگزین آن‌ها می‌شود. برای مثال کاربر می‌تواند با حرکات صورت خود موس را جابجا کند و با استفاده از حالات چهره یا فرمان‌های صوتی کلیک کند. در ضمن می‌تواند متنی که می‌خواهد تایپ کند را به صورت صوتی به سیستم بگوید و سیستم به صورت خودکار متن را تایپ کند و یا با استفاده از فرمان‌های صوتی سیستم را خاموش کند.

در نتیجه کاربر می‌تواند تجربه راحتی از کار با سیستم خود بدون نیاز به موس و کیبورد داشته باشد درحالیکه این ماژول می‌تواند بر روی هر سیستمی سوار شود. همچنین زبان دستیار صوتی آن فارسی است که یک مزیت برای کاربران فارسی زبان به حساب می‌آید چرا که دستیاران صوتی تا کنون غالبا این زبان را پشتیبانی نمی‌کنند.

در جدول زیر قابلیت‌های سیستم طراحی شده آورده شده است:

جدول ۱ نحوه فعال و غیرفعال کردن ویژگی‌ها

حرکت	توضیح
صدا روشن (یا کلیک بر روی دکمه voice در اپلیکیشن)	فعال کردن دستیار صوتی
صدا خاموش (یا کلیک کردن بر روی دکمه voice)	غیرفعال کردن دستیار صوتی
موس روشن (یا کلیک کردن بر روی دکمه mouse)	فعال کردن جابجایی موس با سر
موس خاموش (یا کلیک کردن بر روی دکمه mouse)	غیرفعال کردن جابجایی موس با سر
کیبورد روشن (یا کلیک کردن بر روی دکمه keyboard)	فعال کردن تایپ متن به وسیله صوت
کیبورد خاموش (یا کلیک کردن بر روی دکمه keyboard)	غیرفعال کردن تایپ متن به وسیله صوت
چشمک روشن (یا کلیک کردن بر روی دکمه blink)	فعال کردن انواع کلیک به کمک چشمک
چشمک خاموش (یا کلیک کردن بر روی دکمه blink)	غیرفعال کردن انواع کلیک به کمک چشمک

جدول ۲استفاده از چشمک به جای کلیک

حرکت	توضیح
چشمک راست	کلیک راست
چشمک چپ	کلیک چپ
بستن هر دو چشم	کلیک جفت

جدول ۳فرمان‌های دستیار صوتی

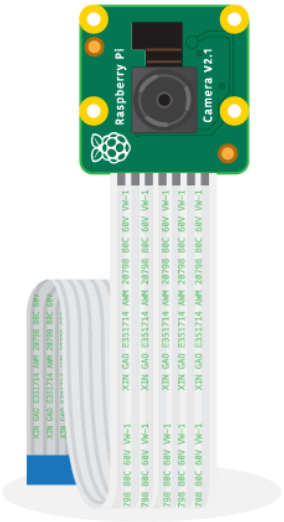
فرمان صوتی	توضیح
کلیک راست	**کلیک راست می‌کند.**
کلیک چپ	**کلیک چپ می‌کند.**
کلیک جفت	**دوبار کلیک می‌کند.**
راه اندازی مجدد	**سیستم را restart می‌کند.**
خاموش	**سیستم را خاموش می‌کند.**

قطعات مورد استفاده

برای ساخت این ماژول از یک برد Raspberry Pi۳ استفاده شده است که تصویر آن را در شکل ۱ مشاهده می‌کنید.

شکل ۱ برد رزبری‌پای۳

همچنین برای دوربین هم از دوربین مخصوص رزبری پای OV۵۶۴۷ استفاده کردیم. شمای کلی دوربین در شکل ۲ آورده شده است. این ماژول یک دوربین ۵ مگاپیکسلی است که مخصوص برد رزبری طراحی شده است.



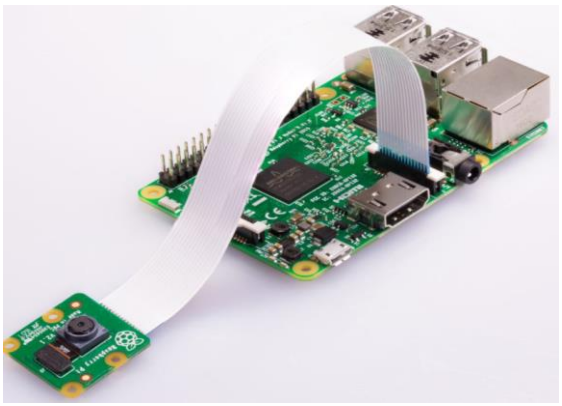
شکل ۲ ماژول دوربین مخصوص رزبری‌پای

برای میکروفون هم از ماژول میکروفون USB مدل M-۳۰۶ استفاده شد که شکل آن را در تصویر ۳ قابل مشاهده است.

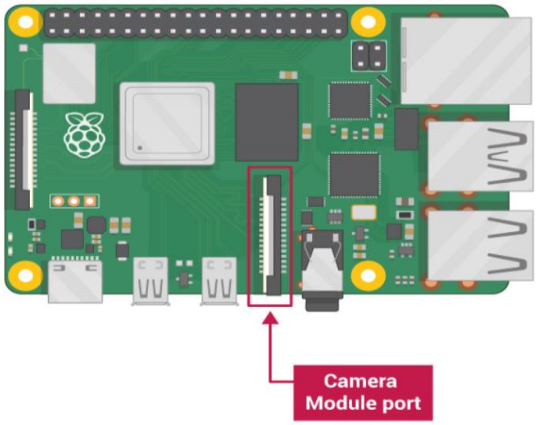
شکل ۳ ماژول میکروفون M306

نحوه اتصال دوربین به رزبری‌پای:

برای اتصال ابتدا باید رزبری خاموش باشد. سپس درب پورت دوربین رزبری را باز کرده و کابل روبانی ماژول را داخل آن قرار می‌دهیم و درب را می‌بندیم. شکل ۴ شکل رزبری بعد از انجام این مرحله و شکل ۵ محل اتصال دوربین به برد رزبری را نمایش میدهد.



شکل ٤ اتصال ماژول دوربین به برد رزبری‌پای



شکل ۵ پورت دوربین بر روی برد رزبری‌پای

سپس رزبری را روشن کرده و به منوی raspberry pi configuration می‌رویم و از بخش camera، interfaces را enable می‌کنیم. حال دوربین ما متصل و آماده استفاده است. برای مثال می‌توان با وارد کردن دستور

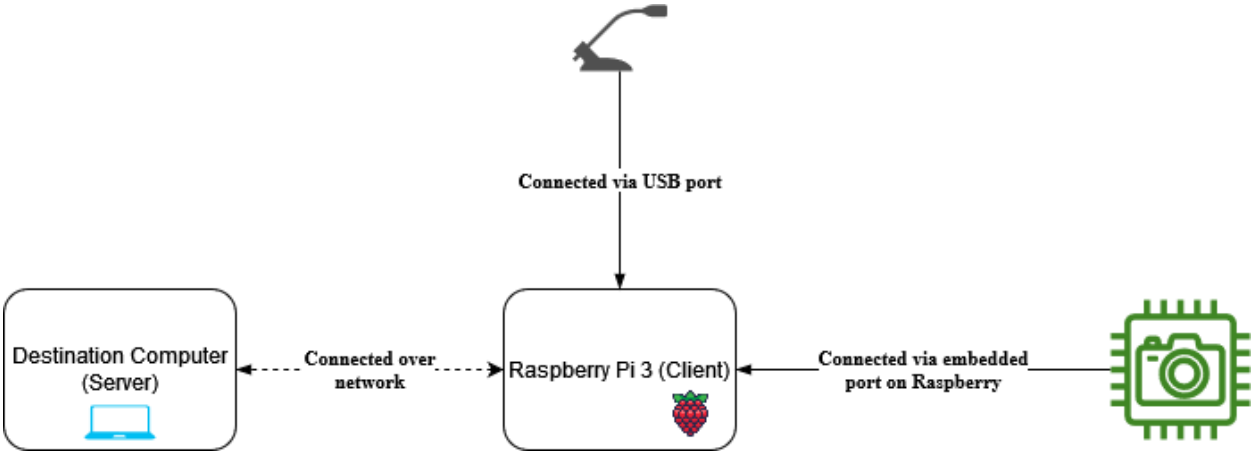
```
raspistill -o Desktop/image.jpg
```

یک عکس با دوربین بگیریم و در دسکتاپ ذخیره و مشاهده کنیم.

نحوه اتصال میکروفون به رزبری‌پای:

برای اتصال کافی است از طریق پورت USB این میکروفون را به رزبری‌پای متصل کنیم.

در قسمت زیر، بلوک دیاگرام ماژول و سیستم کاربر را مشاهده می‌کنید.



شکل ۶ بلوک دیاگرام سیستم

همانطور که در این شکل آورده شده است، کامپیوتر مقصد که توسط ماژول قرار است کنترل شود، به کمک شبکه و پترن کلاینت سرور به ماژول طراحی شده متصل می‌شود و از این طریق ارتباطات دو طرفه‌ی بین کامپیوتر مقصد و ماژول برقرار می‌شود. دقت کنید که ارتباط ماژول با میکروفون و دوربین یک طرفه است زیرا تنها به عنوان دیوایس‌های ورودی از آن‌ها استفاده می‌کند.

راه اندازی Raspberry Pi

در ابتدا باید سیستم‌عامل Raspbian را بر روی این برد نصب کنیم:

۱. **دانلود Raspbian: می‌توان با مراجعه به لینک زیر Disk Image این سیستم‌عامل را دانلود کرد:**
https://www.raspberrypi.com/software/

۲. **در این مرحله Disk Image دانلود شده را بر روی یک microSD قرار می‌دهیم.**

۳. **در این مرحله microSD را داخل Raspberry Pi قرار داده و سپس رزبری را روشن می‌کنیم.**

٤. **بعد از روشن کردن سایر مراحل نصب به صورت اتوماتیک انجام می‌شود. نام کاربری به صورت پیشفرض pi و رمز عبور raspberry است.**

اجرا کردن کد

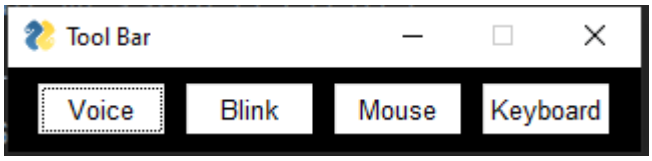
قبل از این که به سراغ اجرا کردن کد برویم، باید نیازمندی‌های این پروژه را نصب کنید. برای این کار می‌توانید از کتاب‌خانه‌هایی که در فایل requirements آورده شده است استفاده کنید. دقت کنید در صورتی که میکروفون به درستی بر روی رزپری‌پای شما کار نمی‌کند، اروری دریافت می‌کنید که در آن گفته شده است نیاز است که flac نصب شود. برای این کار از دو خط زیر می‌توانید استفاده کنید.

```
sudo apt-get update
sudo apt-get install flac
```

توجه کنید که با توجه به کتاب‌خانه‌های پیشین نصب شده روی سیستم شما، ممکن است که مجبور باشید ورژن برخی از آن‌ها را به روز رسانی کنید. در این صورت با کمک متن نوشته شده در ارور هنگام اجرا که سیستم عامل به شما می‌دهد، تغییرات لازم را انجام دهید.

حال به سراغ اجرای کد می‌رویم. بعد از اینکه ماژول‌ها را به رزبری متصل کردیم و رزبری را روشن کردیم و آماده بود، باید کد server.py را بر روی سیستم کاربر اجرا کنیم. سپس باید فایل main.py را بر روی رزبری اجرا کنیم. حال یک toolbar بر روی سیستم کاربر نمایش داده می‌شود که ۴ دکمه دارد که تصویر آن در شکل ۶ آورده شده است:

- Voice: با فشار دادن این دکمه حالت دستیار صوتی تغییر کرده و از فعال به غیرفعال (و بالعکس) تغییر می‌کند.
- Mouse: با فشار دادن این دکمه حالت «جابجایی موس با صورت» تغییر می‌کند.
- Blink: با فشار دادن این دکمه حالت «کلیک کردن با استفاده از چشمک» تغییر می‌کند.
- Keyboard: با فشار دادن این دکمه حالت «تایپ کردن صوت ورودی» تغییر می‌کند.



شکل ۷ تصویر نوار ابزار

دقت کنید که نیاز است آی‌پی سرور مقصد در کد فایل‌های server.py و network.py مشخص شوند. از آنجا که آی‌پی تخصیص داده شده ثابت نیست، لازم است که مقدار مورد نظر به درستی مشخص شده باشد. در غیر این صورت ارتباط بین سرور و کلاینت به درستی برقرار نمی‌شود.

کتابخانه‌ها

در قسمت زیر کتاب‌خانه‌های اصلی استفاده شده در کدها و توضیحات آن‌ها را مشاهده می‌کنید.

numpy: **برای انجام عملیات‌های ریاضی بر روی ماتریس‌ها**

dlib: **کتابخانه‌ای دارای الگوریتم‌های یادگیری ماشین**

opencv: **کتابخانه‌ای مناسب برای کاربردهای پردازش تصویر به صورت** real time

PyAudio: **کتابخانه‌ای مناسب برای ورودی و خروجی صدا در پایتون**

PySimpleGUI: **کتابخانه‌ای برای کارهای گرافیکی در پایتون**

SpeechRecognition: **کتابخانه‌ای برای پردازش صوت**

Socket: **کتابخانه‌ای برای انجام کارهای شبکه**

Threading: **کتابخانه‌ای برای کار با ریسه‌ها و مدیریت آن‌ها**

شبکه

نحوه برقراری ارتباط بین رزبری و سیستم به این صورت است که با اجرای server.py بر روی سیستم یک سرور اجرا می‌شود و با اجرای main.py بر روی رزبری به آن سرور متصل می‌شویم. حال پردازش‌ها بر روی رزبری انجام شده و فرمان‌ها از طریق شبکه به سیستم فرستاده می‌شوند. سیستم این فرمان‌ها را دریافت کرده و بر روی سیستم اعمال می‌کند.

همچنین هنگامیکه دکمه‌های Toolbar فشار داده می‌شوند، یک پیام از طریق شبکه از سمت سرور به سمت رزبری فرستاده می‌شود که باعث می‌شود وضعیت ماژول تغییر کند.

قطعه کد زیر برای ساخت سرور است:

```
host = '۱۲۷,۰,۰,۱'
port = ۸۵۵۰

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen(۱)

client, address = server.accept()
```

ساختار پیام‌های رد و بدل شده در شبکه به صورت زیر است که type مشخص‌کننده نوع پیام است و درصورتی که type=move mouse آنگاه move_x و move_y مقادیر معتبر دارند و برابر مقداری هستند که موس باید جابجا شود. همچنین درصورتی که type=keyboard آنگاه typed_text مقدار معتبر دارد و متنی است که باید تایپ شود.

```
class Message:
    def __init__(self, type: str = None, move_x: int = ۰, move_y:int =
۰, typed_text: str = None):

        self.type = type
        self.move_x = move_x
        self.move_y = move_y
        self.typed_text = typed_text
```

پیام‌ها را در شبکه به صورت json میفرستیم و برای خواندن آن‌ها نیز به صورت بایت به بایت پیام را خوانده و با کمک {} که ابتدا و انتهای json را مشخص می‌کنند پیام‌ها را تفکیک می‌کنیم. علت این کار این است که اگر یکدفعه بخوانیم ممکن است بیش از یک پیام کامل را بخوانیم و با مشکل روبرو شویم. قطعه کد زیر برای همین منظور است:

```
def read_message(client: socket.socket):
    msg = ""
    while True:
        ch = client.recv(۱).decode("utf-۸")
        msg += ch
        if ch == '}':
                break
     return msg
```

قطعه کد زیر هم برای Toolbar کاربر است که یک پنجره با ۴ دکمه می‌سازد و هرکدام که فشار داده شود یک پیام به رزبری ارسال می‌شود و اطلاع داده می‌شود.

```
def handle_menu():

    sg.theme('Black')  # Add a touch of color
    # All the stuff inside your window.
    layout = [[sg.Button('Voice', size=(١٠, ٢)),
               sg.Button('Mouse', size=(١٠, ٢)),
               sg.Button('Blink', size=(١٠, ٢)),
               sg.Button('Keyboard', size=(١٠, ٢))]]

    # Create the Window
    window = sg.Window('Tool Bar', layout, grab_anywhere=True)
    # Event Loop to process "events" and get the "values" of the
inputs
    while True:
        event, values = window.read()
        print(event, values)
        msg = Message()
        if event == sg.WIN_CLOSED:  # if user closes window
            break
        elif event == 'Mouse':
            msg.type = 'Mouse'
        elif event == 'Keyboard':
            msg.type = 'Keyboard'
        elif event == 'Voice':
            msg.type = 'Voice'
        elif event == 'Blink':
            msg.type = 'Blink'

        client.send(json.dumps(msg.__dict__).encode('utf-٨'))
```

همچنین کد network.py اجرا می‌شود تا اتصال رزبری به سروری که بالا آمده انجام شود. قطعه کد زیر برای این کار است.

```
host = '١٢٧,٠,٠,١'
port = ٨٥٥٠

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((host, port))
```

همچنین در این کد پیام‌هایی که از سمت سرور می‌آید خوانده می‌شود و وضعیت mouse یا keyboard یا voice یا blink تغییر می‌کند. قطعه کد زیر برای این کار است.

```
def read():
    while True:
        # try:
```

```
        data = read_message(Config.client)
        M = json.loads(data)
        msg = Message(**M)

        if msg.type == 'Mouse':
            Config.mouse = not Config.mouse
        elif msg.type == 'Keyboard':
            Config.keyboard = not Config.keyboard
        elif msg.type == 'Blink':
            Config.blink = not Config.blink
        elif msg.type == 'Voice':
            Config.speech_recognition = not Config.speech_recognition
```

همچنین یک فایل به نام config داریم که بر روی رزبری است و وضعیت ماژول در لحظه را دارد. مثلا اینکه دستیار صوتی فعال است یا خیر و کلاینتی که رزبری از طریق آن به سرور سیستم کاربر متصل است. ساختار آن به صورت زیر است:

```
class Config:
    speech_recognition = False
    mouse = False
    keyboard = False
    blink = False

    client = None
```

دستیار صوتی

در این بخش از طریق ماژول میکروفون صوت را ورودی گرفته و با کمک دستیار صوتی گوگل این صوت را به متن تبدیل کرده و پردازش‌های لازم را روی آن انجام می‌دهیم. برای این کار هم در بازه‌های زمانی که کاربر حرف می‌زند، کد ورودی صوتی دریافت کرده و با توقف حرف زدن کاربر صوت ورودی در این بازه، پردازش می‌شود. درصورتیکه صوت ورودی یکی از فرمان‌های معتبر بود، آن فرمان بر روی سیستم اصلی اعمال می‌شود.

فرمان‌های صوتی معتبر عبارتند از:

- صدا روشن: برای فعال‌کردن دستیار صوتی باید ابتدا این دستور صوتی گفته شود.
- صدا خاموش: برای غیرفعال‌کردن دستیار صوتی است.
- کلیک راست
- کلیک چپ
- کلیک جفت: برای انجام دوبار کلیک پشت هم یا همان double click است.
- موس روشن: برای فعال‌کردن جابجایی موس با حرکت صورت است. در واقع با چرخاندن صورت به سمت بالا، پایین، چپ و راست موس به آن سمت حرکت می‌کند.
- موس خاموش: برای غیرفعال‌کردن جابجایی موس با حرکت صورت است.
- چشمک روشن: برای فعال کردن استفاده از چشمک به جای کلیک موس است. اگر کاربر با چشم چپ چشمک بزند کلیک چپ، اگر با چشم راست چشمک بزند کلیک راست و اگر هر دو چشم را ببندد double click انجام می‌شود.
- چشمک خاموش: برای غیرفعال کردن جایگزینی چشمک با کلیک موس است.
- راه اندازی مجدد: برای restart کردن سیستم است.
- خاموش: برای خاموش کردن سیستم است.
- کیبورد روشن: برای فعال کردن تایپ کردن صوت ورودی است. در واقع اگر از این گزینه استفاده کنیم، هر حرفی که بزنیم در سیستم کاربر تایپ می‌شود.
- کیبورد خاموش: تایپ کردن صوتی غیر فعال می‌شود.

حال به توضیح کد و پیاده‌سازی این قسمت می‌پردازیم. تمامی کد این قسمت در فایل speech_detection.py قرار داده شده است.

در ابتدای این فایل، مقادیر اولیه‌ی کانفیگ میکروفون و حساسیت ست شده است.

تابع main: این تابع این پراسه را شروع می‌کند.

```
def main():
    logging.basicConfig(filename='speech.log', level=logging.INFO)
    start_recognizer()
```

تابع start_recognizer: این تابع یک ترد در بک‌گراند ایجاد میکند که به صدای کاربر گوش می‌دهد.

```
def start_recognizer():
    r.listen_in_background(source, callback)
```

```
    while True:
        pass
```

تابع callback: زمانی که کاربر صحبت کند، صدای او به عنوان یک فایل آودیو به این تابع داده می‌شود. سپس بر اساس API گوگل، این صوت به متن فارسی در می‌آید. اگر متنی دریافت نشد، ارور می‌دهد و در غیر این صورت، متن تفسیر می‌شود.

با توجه به این که صدای ورودی ممکن است نویز داشته باشد، دستورات ورودی نیازمند بهبود و اصلاح می‌باشند. برای همین دو تابع get_similarity و correct_input پیاده سازی شده‌اند که متن اولیه را ورودی می‌گیرد. در صورتی که کیبورد روشن باشد، ممکن است همان متن مورد نظر کاربر بوده است و به همین دلیل تغییری نمی‌کند. اما اگر کیبورد خاموش باشد، صدای ورودی صدای یک دستور است و به همین دلیل شبیه‌ترین دستوری که به متن ورودی وجود دارد، به عنوان ورودی تشخیص داده می‌شود. در زیر کد این توابع را می‌بینید.

```
def callback(recognizer, audio):
    try:
        text = recognizer.recognize_google(audio, language='fa')
        interpret_text(text)

    except sr.UnknownValueError:
        print("Oops! Didn't catch that")
```

تابع interpret_text:

در ابتدا فعال/غیرفعال شدن دستیار صوتی چک می‌شود و سپس متن داده شده تفسیر می‌شود.

```
def interpret_text(text):
    check_for_speech_recognition_enabling(text)

    if not Config.speech_recognition:
        return
    check_for_speech_commands(text)
```

تابع check_for_recognition_enabling:

این تابع بررسی می‌کند که آیا تکست ورودی دستیار صوتی را فعال/غیرفعال می‌کند یا خیر.

تابع check_for_speech_commands و check_for_system_commands و check_for_keyboard_commands:

تابع اول تکست ورودی را تفسیر می‌کند. تفسیر می‌تواند منجر به فعال کردن فیچرهای دیگر مانند کیبورد، موس یا چشمک شود یا این که دستوراتی مانند تایپ با کیبورد، راه‌اندازی مجدد یا خاموش اجرا شوند. تابع دوم و سوم نیز برای ساده‌تر شدن کد این قسمت استفاده شده‌اند.

ویدیو

قسمت ویدیو متشکل از دو بخش است: کنترل کردن کلیک‌ها با چشمک - کنترل موس به کمک حرکت سر

با استفاده از کدهای فراهم شده برای این دو بخش، در فایل video_controller.py تصویر از وبکم گرفته می‌شود و سپس تصویر گرفته شده به دو تابع مختلف برای چشمک و موس (به شرط فعال بودن) داده می‌شود. برای قسمت چشمک، در صورتی که عملیاتی ایجاد شود، تا ۲ ثانیه دیگر چشمکی پذیرفته نمی‌شود که دلیل آن این است که کلیک‌های پشت هم انجام ندهد. برای موس نیز، یک ترد ایجاد شده است که هر ۰.۱، با توجه به آخرین پوزیشن سر، موس را در راستای مورد نظر حرکت می‌دهد. در صورتی که حرکت موس کند باشد، می‌توان مدت زمان رفرش را کاهش داد و اگر دقت مورد اهمیت نباشد، می‌توان تنها مقدار جابه‌جایی را افزایش داد.

هر دو قسمت ویدیو، با کمک کدهای از پیش آماده که توسط دیگران در اینترنت به صورت رایگان قرار داده شده است استفاده شده است.

کلیک کردن با چشم

برای این قسمت، از کد blink_detection استفاده شده است. این کد در اینترنت نیز موجود است. مکانیزم کارکرد این کد به این شکل است که با تشخیص صورت، تعدادی landmark بر روی دور چشم‌ها قرار می‌گیرد. سپس وقتی که شخص چشمش را ببند، با توجه به تغییر مکان landmarkها، مشخص می‌شود که به چه میزان چشم شخص بسته شده است و معیاری برای بسته بودن چشم بدست می‌آید. سپس برای هر چشم، ترشهولدی در نظر گرفته شده و وقتی که معیار از ترشهولد بیشتر شود، به معنای بسته بودن چشم است. این ترشهولد به صورت تجربی بدست آمده است.

توابع کمکی: این دو تابع کمک می‌کند تا مرکز دو نقطه و فاصله‌ی اقلیدسی دو نقطه محاسبه شود.

تابع get_blink_ratio: این تابع با توجه به landmark های دور چشم، مقدار فاصله‌ی افقی چشم تقسیم بر فاصله‌ی عمودی چشم را بدست می‌آورد و در صورتی که از حدی بیشتر شود، به معنای بسته بودن چشم است.

```
def get_blink_ratio(eye_points, facial_landmarks):
    #-----Step ٥: Getting to know blink ratio

    #loading all the required points
    corner_left  = (facial_landmarks.part(eye_points[٠]).x,
                    facial_landmarks.part(eye_points[٠]).y)
    corner_right = (facial_landmarks.part(eye_points[٣]).x,
                    facial_landmarks.part(eye_points[٣]).y)

    center_top    = midpoint(facial_landmarks.part(eye_points[١]),
                             facial_landmarks.part(eye_points[٢]))
    center_bottom = midpoint(facial_landmarks.part(eye_points[٥]),
                             facial_landmarks.part(eye_points[٤]))
```

```
    #calculating distance
    horizontal_length = euclidean_distance(corner_left,corner_right)
    vertical_length = euclidean_distance(center_top,center_bottom)

    ratio = horizontal_length / vertical_length

    return ratio
```

تابع check_face_blink: این تابع در فریم‌های موجود، بسته بودن هر یک از چشم‌ها را بر اساس ترشهولد گزارش می‌کند.

تابع check_frame_for_blink: این تابع فیس‌های مورد نظر را با کمک تابع dlib بدست می‌آورد و بسته بودن چشم را گزارش می‌کند. ما در کد خود مستقیما از این تابع استفاده کرده‌ایم.

کنترل موس با حرکت سر:

برای حرکت سر، همانطور که توضیح داده شد، با توجه به جهت سر می‌توان موس را در راستای مورد نظر حرکت داد. برای این کار از فایل head_orientation_triangles.py استفاده شده است. دقت کنید که این عملیات، بسیار زمان‌گیر است و رزبری‌پای امکان محاسبه‌ی این مورد را ندارد. به همین دلیل، این فیچر تنها بر روی سیستم‌های قوی‌تر قابل استفاده است.

head_orientation_triangles: این فایل کد مربوطه برای جابجایی موس با حرکات سر است.

pose_estimation.py: این فایل کدهای مربوط به محاسبه نقاط حساس چهره و توابع کمکی برای کار کردن با آن‌ها است.

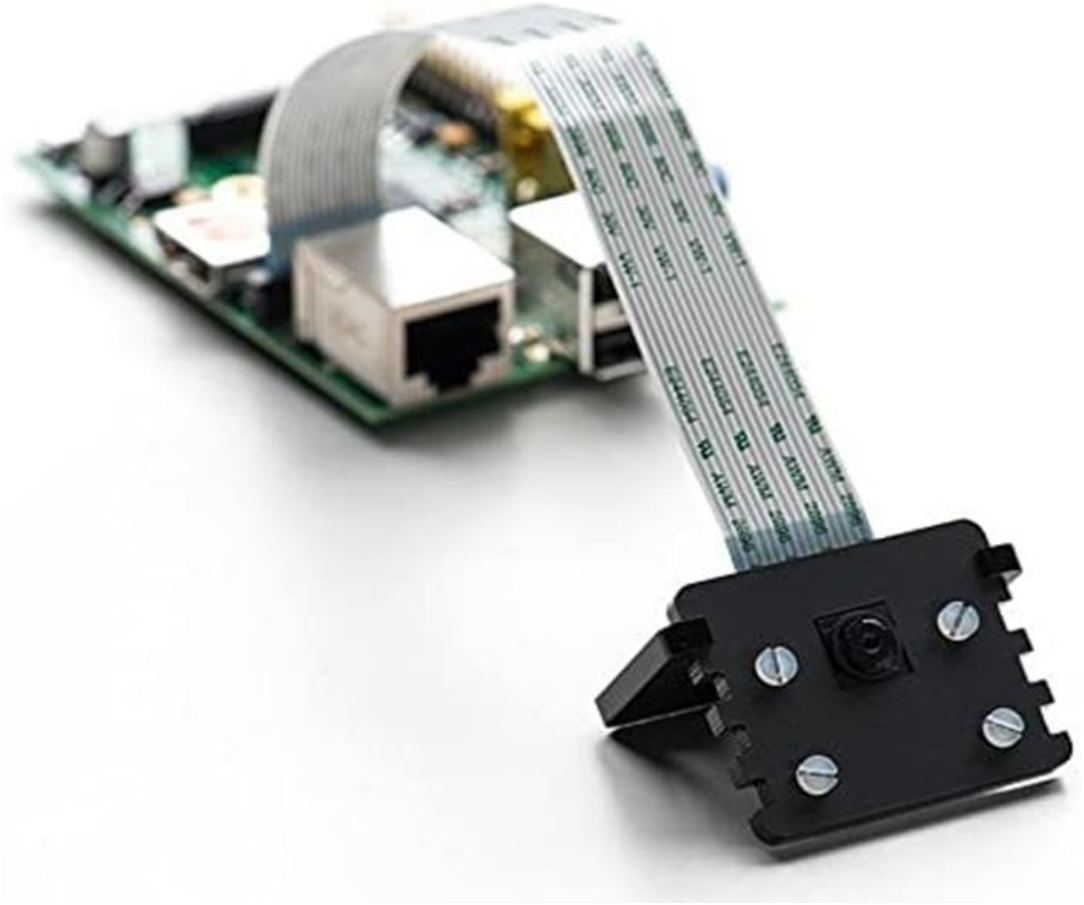
توابع:

calc_ratio_ud(img, detector, predictor, temp): این تابع یک تصویر از چهره به همراه یک مدل برای انتخاب نقاط آن را ورودی می‌گیرد و بر اساس مختصات نقاط و نحوه قرارگیری آن‌ها با یک معیار مشخص می‌کند که به چه میزان چهره رو به بالا است.

calc_ratio_lr(img, detector, predictor, temp): این تابع هم مشابه calc_ratio_ud است با این تفاوت که مشخص می‌کند به چه میزان چهره رو به سمت چپ است.

بسته‌بندی

از آنجایی که کل محصول شامل یک رزبری پای، یک عدد میکروفون و یک عدد دوربین است، بسته‌بندی این محصول پیچیدگی ندارد و به راحتی برای کاربر قابل استفاده است. خود رزبری پای در قابل خود قرار دارد که در برابر ضربات احتمالی از خود رزبری پای محافظت کند.برای نگهداری میکروفون از آنکه خود میکروفون یک پایه دارد و به راحتی قابل جابجایی است نیاز به مورد اضافه‌تری نداریم. برای نگهداری دوربین نیازمند یک پایه هستیم که به دلیل کمبود امکانات نتوانستیم آن را فراهم کنیم اما عکسی از پایه را در زیر مشاهده می کنید. این پایه به عنوان نگهدارنده‌ی دوربین‌های رزبری پای استفاده می‌شود. به کمک این پایه دوربین بر روی یک سطح مستقر می شود و می تواند تصویر باکیفیت از شخص را ذخیره کند.



شکل ۷ بسته بندی پیشنهادی برای دوربین

همچنین خود رزبری را نیز داخل جعبه‌ای قرار می‌دهیم که شکل آن در شکل ۱ آورده شده است.

شکل ابسته بندی رزبری

جمع‌بندی

خرید

تخمین اولیه‌ای که از قیمت محصول داشتیم برابر بود با صد هزار تومان برای دوربین، صد هزار تومان برای میکروفون و چهار میلیون و پانصد هزار تومان برای رزبری که حدودا چهار میلیون و هفتصد هزار تومان می‌شود.

در نهایت این محصول با صد هزار تومان برای میکروفون و نود هزار تومان برای دوربین به سرانجام رسید:

رزبری مورد استفاده از دانشگاه گرفته شد، دوربین را از سایت thecaferobot.com سفارش دادیم و میکروفون را از سایت torob.com خریداری کردیم.

تست

همچنین تمامی ویژگی‌های این محصول از قبیل حرکت موس با سر، انواع فرمان‌های صوتی و همچنین تایپ‌کردن متن با صوت تست شد و تست‌ها بر روی سیستم غیر رزبری با موفقیت طی شدند و تنها مشکل این بود که توان پردازشی رزبری برای انجام کارهای پردازش تصویر به اندازه کافی نبود و کاربر تجربه روانی در استفاده از عملکردهای تصویری نداشت. در نتیجه برای عملکرد روان و سریع بر روی برد رزبری‌پای، باید بخش استفاده از موس با سر و چشمک را غیرفعال کرد. درحالیکه این محصول بر روی سیستم‌های عادی مثل لپتاپ میتواند به صورت کامل اجرا و استفاده شود.

منابع

https://stackoverflow.com/

https://pysimplegui.readthedocs.io/

https://towardsdatascience.com/real-time-head-pose-estimation-in-python-e٥٢db١bc٦٠٦a

https://www.geeksforgeeks.org/python-convert-speech-to-text-and-text-to-speech/

https://docs.python.org/

https://projects.raspberrypi.org/en/projects/getting-started-with-picamera