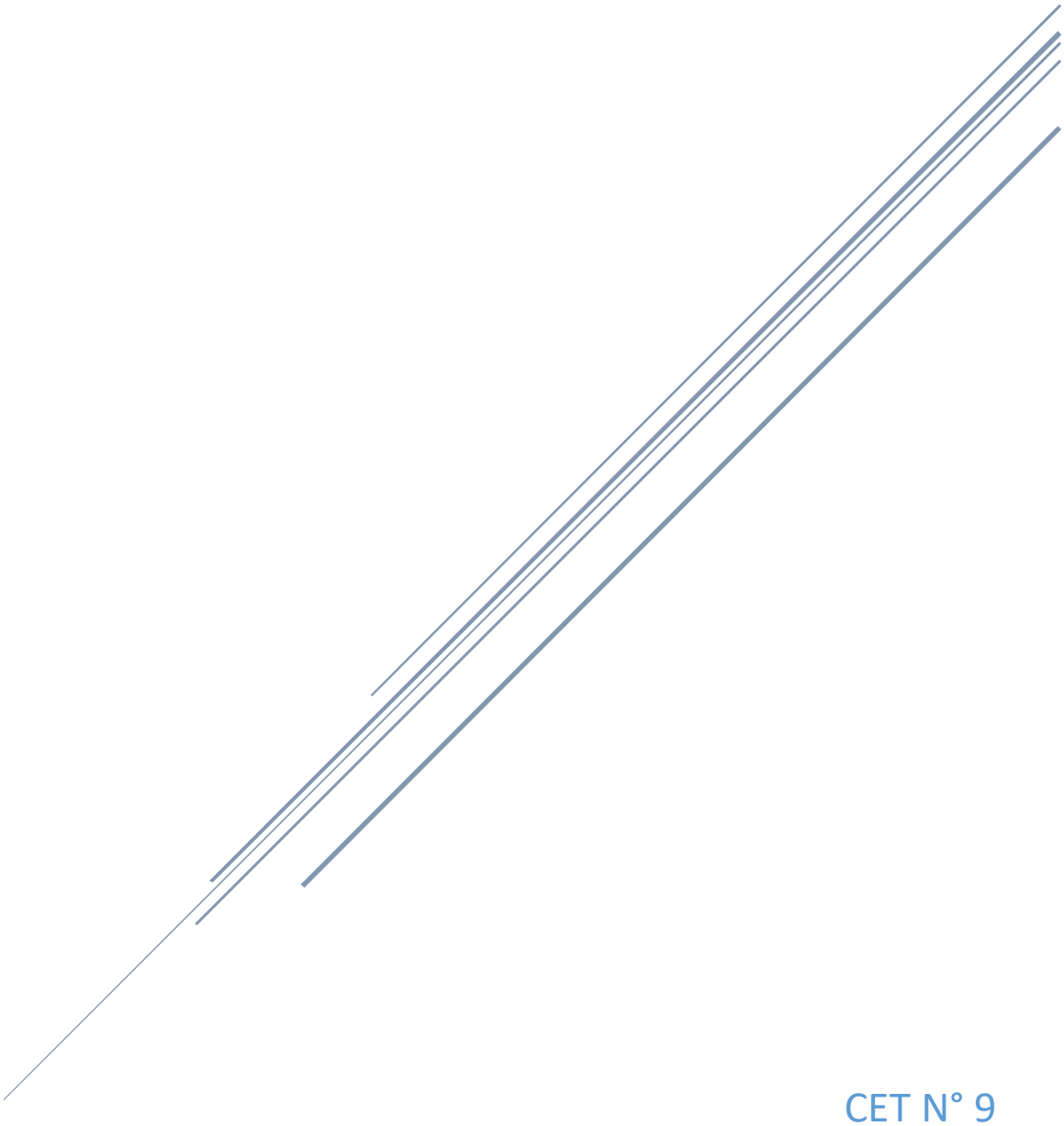
VISIÓN DE LAS MAQUINAS

Sistema de seguridad con reconocimiento facial



CET N° 9
Agustín Sepúlveda

Visión de las maquinas

Proyecto: Sistema de seguridad con reconocimiento facial.

Objetivo: Lograr crear un sistema de seguridad, el cual cuente con reconocimiento facial como método de desbloqueo. Va a contar con una base de datos en la cual se van a registrar los usuarios a los cuales se les permite la entrada. A los que no, luego de haber intentado 3 veces de reconocer su cara y todavía nada, se va a mandar una notificación a algún celular definido por el usuario, diciendo que hay una persona no identificada intentando entrar en el dominio. Una vez logrado el objetivo, se proseguirá con la creación de un sitio web donde se puede ver el paso a paso de cómo ha sido creado, y con las explicaciones necesarias para que otra persona pueda replicarlo.

Materiales necesarios:

- Raspberry Pi 3B o 3B+ (cualquiera de las 2 sirve).
- Alguna cámara web que no utilicemos, o en caso de querer comprar una, procurar que tenga una resolución mínima de 1280x720.
- Memoria SD de al menos 32gb clase 10 (donde se va a guardar el sistema operativo de la Raspberry Pi).
- Una computadora desde donde vamos a entrenar nuestro modelo de inteligencia artificial para que pueda identificar rostros, y más adelante reconocer de quien es el rostro mediante una base de datos.

Se va a trabajar con Python como lenguaje de programación, ya que es bastante simple, que cualquiera puede aprender a utilizar en cuestión de pocas horas.

También vamos a estar trabajando con una librería muy popular llamada “OpenCV”, la cual es utilizada para proyectos realizados con la visión de la computadora.

El sistema operativo que se le va a instalar a la Raspberry Pi es Ubuntu mate (por preferencia).

Todo este proyecto va a estar siendo actualizado constantemente en su GitHub oficial, el cual es: https://github.com/AguuSz/facialRecog-OpenCV

Ambiente de trabajo:

Este proyecto puede ser realizado en cualquier parte donde se tenga acceso a una computadora y a una conexión a internet. De ser posible, lo ideal sería que la computadora desde la cual vamos a entrenar al modelo, tenga una potencia grafica decente, esto con el fin de que sea más fácil su entrenamiento.

Procedimiento:

Para empezar con este proyecto lo primero sería estructurar sus pasos. Lo que primero tiene que ser capaz de realizar, es reconocer la cara de la persona. Más adelante hay que entrenar una red neuronal, que va a estar construida en Keras o Tensorflow, la cual va a estar destinada a, en base a una base de datos, identificar cara y distinguirla de otra. Ejemplo: Si pongo mi cara en la cámara, la idea sería que me reconozca y no me dé el

nombre de otra persona. El proceso de entrenamiento es muy tedioso, por lo que, de ser posible, se va a recurrir a alguna ya creada y entrenada por otra persona. Hay algunas librerías que son de libre uso, las cuales son muy buenas, solo hay que implementarlas y entrenarlas con los datos que nosotros queramos incluir.

Dia 13/04

Se realizan búsquedas acerca de que tecnologías poder implementar para que sea lo más sencillo posible, tanto de implementar, como luego para explicar. Encontré una librería en GitHub del usuario "ageitgey", dicho link es: https://github.com/ageitgey/face_recognition. La cual nos da un primer acercamiento a lo que es la clasificación de personas en imágenes, esto es bastante sencillo de explicar y es que, en una imagen, cuando hay más de una persona, clasifica sus caras en base a unos parámetros que se toman de una imagen del sujeto en cuestión que se le da como dato previo. Dicho esto, se va a estar viendo como poder implementarla junto con la librería "OpenCV" así puede trabajar con imágenes en vivo y en directo (directamente desde la cámara de la computadora y en tiempo real).

Días siguientes hasta el 17/04

Luego de haber intentado durante varios días el integro y análisis de la librería de "ageitgey", se pudo comprobar que funciona, ¡y sorprendentemente bien! Es capaz de, con tan solo una imagen de tu cara con una emoción seria, detectarte si estás haciendo algún gesto, o incluso reconocerte entre otras personas.

Si bien funciona bien y todo, el hecho de intentar usarla fue bastante complicado, por algunas incompatibilidades que se tenía con el sistema operativo con el que se está trabajando. Relato los problemas:

1. Lo primero que se complicó, fue que, no se podía crear un ambiente virtual de trabajo para poder trabajar adecuadamente. Esto, para una persona que no entienda de programación le puede sonar muy raro, pero en realidad es algo bastante sencillo. La mejor manera de explicarlo, se me ocurre que sería entendiendo la computadora como un mundo, donde todos sus componentes y programas trabajan entre sí, llamaremos ciudades a los programas. Si bien, como dije anteriormente, trabajan entre sí, para este proyecto me conviene aislar una ciudad para componerla como yo quisiera, es decir, armarle los edificios como a mí me guste y que no siga un diseño general del mundo. Dicho esto, la generación de esa "ciudad aislada" fue más complicado de lo que se esperaba, ya que había muchos errores en la compatibilidad de programas y con sus versiones. Para una persona relacionada con la programación, lo que sucedía era que estaba teniendo problemas de compatibilidad con las versiones tanto de Python, como las de las librerías a utilizar. El hecho de poder trabajar con un ambiente virtual, nos permite que, si otra persona crea el mismo ambiente, no va a tener ningún problema con trabajar con las herramientas que se brindan desde el repositorio de GitHub.
2. Una vez se logró instalar, el comando responsable de llamar a la función de identificar las caras no respondía… Cuestión, este problema fue más simple de solventar ya que era cuestión de decirle explícitamente a la computadora donde encontrar la función y listo, solucionado.

Luego de estos problemas, se pudo implementar correctamente el día 17/04. Por las pruebas que se le han hecho funciona bastante bien, pudiendo identificar gente con expresiones raras en cuestión de segundos.

Sin embargo, existe el miedo de que su rendimiento no sea tal cuando se lo implemente con una versión “Live-view”, es decir, en tiempo real. En este caso, también se investigó un modelo pre-entrenado capaz de reconocer caras con un porcentaje de precisión mayor al 96%. Dicho modelo, lo lanzó al público la universidad de Oxford de Estados Unidos, para ser más exactos, el departamento de Ciencia de ingeniería. Se puede encontrar más información acá: http://www.robots.ox.ac.uk/~vgg/software/vgg_face/

Lo bueno que tiene esta librería, es que es la más precisa según la comunidad de inteligencia artificial, cuando a reconocimiento facial se refiere. Aunque, la implementación de este modelo todavía es una incógnita. Si bien se buscó información, no hay mucha, y la que hay por lo general es de gente demostrando su funcionamiento. Este problema igual puede ser solucionado leyendo a fondo la documentación de las tecnologías que utiliza, y averiguando la implementación en Python (que es el lenguaje de programación con el cual se está llevando a cabo el proyecto).

Adjunto una de las pruebas que se le hizo:

Primero se le dio solo estas imágenes como “base de datos”, estas personas son las que nosotros ya conocemos de antemano:

La persona en la imagen izquierda es Jon Snow, mientras que en la derecha esta Samwell Tarly, ambos son personajes de la serie Game of Thrones.

Luego se pone esta imagen para que analice:

Luego de haber importado las imágenes, se procede con invocar al comando para que las analice:

```
agus@agus-X406UAR:~/Desktop$ face_recognition ./conocidos/ ./desconocidos/
```

En este comando se están pasando 2 parámetros, el primero es “./conocidos/” y el segundo es “./desconocidos/”. El primer parámetro que recibe es las fotos de las personas que ya se conoce el nombre, en esa carpeta, estaban guardadas las imágenes de Jon y Samwell, con su nombre en cada una de las fotos. Luego en la carpeta “desconocidos” van las imágenes a analizar. Si ingresamos ese comando vemos lo siguiente:

```
WARNING: More than one face found in ./conocidos/jhon.jpeg. Only considering the first face
```

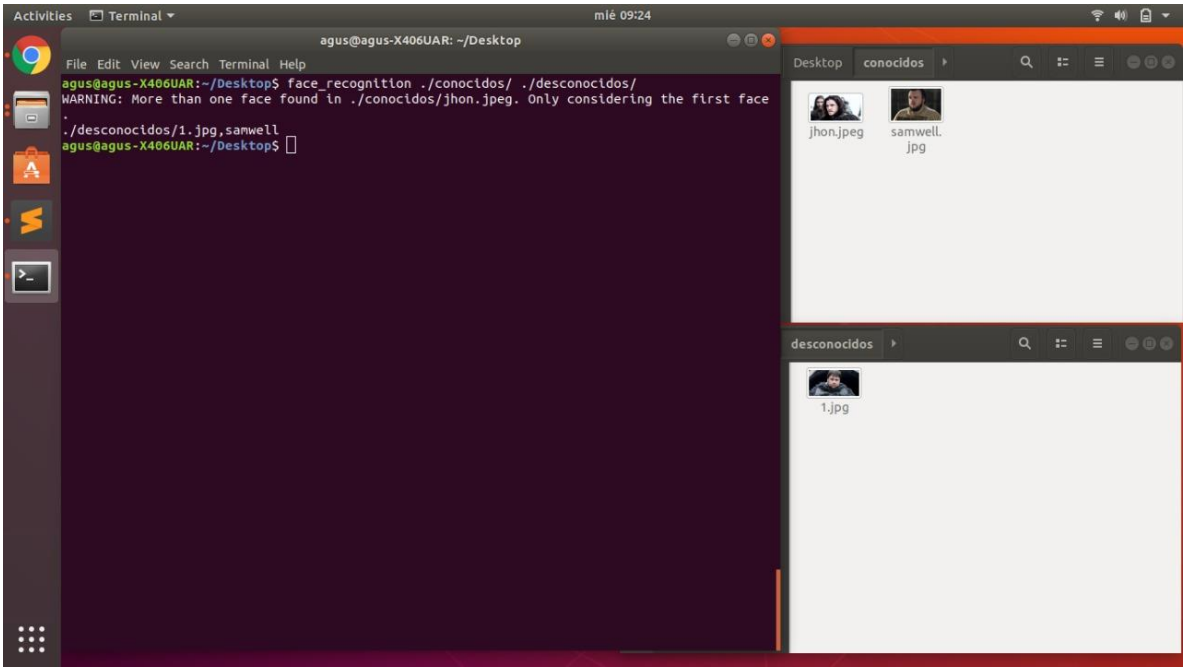
Lo que nos indica, es que en la foto de Jon encontró 2 caras, y esto es correcto, aunque la de atrás está mucho más difuminada, lo cual de por sí, indirectamente nos dice que la calidad no tiene que ser tan alta para que funcione correctamente.

Luego de eso, nos devuelve esto en consola:

```
./desconocidos/1.jpg,samwell
```

Lo que nos dice es que, en la carpeta desconocidos, en la imagen 1, se reconoció a Samwell en la foto, lo cual es genial, ya que de por sí, la segunda imagen era un poco más “tramposa” de adivinar, ya que el pelo era diferente y pasa lo mismo con bigote y barba.

Screenshot final de la prueba:



Dia 18/04

Este día ha sido un excelente día ya que se logro poder realizar el uso de la librería “Face-Recognition” en tiempo real e implementarla con OpenCV para el uso de la cámara integrada de la PC.

Para lograrlo, lo que primero se hizo fue actualizar las dependencias de la librería hasta la versión mas reciente.

Intentare explicar como funciona con palabras sencillas, con el objetivo de que sea comprensible.

1) Lo que primero se hace es cargar la información de las caras las cuales van a estar identificadas. En la mayoría de casos, la database de nuestras imágenes requeriría tener más de 10 imágenes para poder funcionar con una precisión considerable, pero en este caso, esta librería con el uso de una sola ya nos rinde bastante competitiva.

```
# Empezar a capturar video de la camara web 0 (la que viene)
video_capture = cv2.VideoCapture(0)

# Cargar una imagen de prueba y aprende a reconocerla
agus = face_recognition.load_image_file('./imagenes/entrenamiento/agus2.jpeg')
agus_encoding = face_recognition.face_encodings(agus)[0]

# Cargue otra imagen de entrenamiento y que aprenda a reconocerla
ana = face_recognition.load_image_file('./imagenes/entrenamiento/ana.jpeg')
ana_encoding = face_recognition.face_encodings(ana)[0]

# Cargue otra imagen de entrenamiento y que aprenda a reconocerla
johana = face_recognition.load_image_file('./imagenes/entrenamiento/yohana.jpeg')
johana_encoding = face_recognition.face_encodings(johana)[0]

# Cargue otra imagen de entrenamiento y que aprenda a reconocerla
gordo = face_recognition.load_image_file('./imagenes/entrenamiento/gordo.jpeg')
gordo_encoding = face_recognition.face_encodings(gordo)[0]

# Cargue otra imagen de entrenamiento y que aprenda a reconocerla
chad = face_recognition.load_image_file('./imagenes/entrenamiento/chad.jpeg')
chad_encoding = face_recognition.face_encodings(chad)[0]

# Cargue otra imagen de entrenamiento y que aprenda a reconocerla
will = face_recognition.load_image_file('./imagenes/entrenamiento/will.jpeg')
will_encoding = face_recognition.face_encodings(will)[0]
```

2) Luego se crea un array (es un tipo de dato estructurado que nos permite almacenar un conjunto de datos homogéneos) con las caras conocidas, y otro con los nombres de las caras, siempre respetando el orden en el que van.

```
# Crear una array con los encodings de las caras y sus nombres
known_face_encodings = [
    agus_encoding,
    ana_encoding,
    johana_encoding,
    gordo_encoding,
    chad_encoding,
    will_encoding

]
known_face_names = [
    "Agustin",
    "Ana",
    "Johana",
    "Gordo",
    "Chad",
    "Will"
]
```

3) Luego se crea un bucle que va a hacer el cual mantenga encendida la cámara y el revisado de la existencia de un rostro en la imagen o no. Luego realiza el checkeo para ver si el rostro coincide con alguno de los que conocemos o no.

```
while True:
    # Toma un solo frame de video
    ret, frame = video_capture.read()

    # Cambia el tamaño del frame a 1/4 para que el procesamiento sea mas rapido
    small_frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)

    # Convierte la imagen de un perfil BGR a RGB (que face_recognition usa)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Solo procesa cada otro frame del video para ahorrar tiempo
    if process_this_frame:
        # Encuentra todas las caras y los encodings del frame actual del video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # Mira si el rostro hace match con alguna de los rostros conocidos, sino le da el nombre de "Desconocido"
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Desconocido"

            # Usa el rostro conocico y le da el nombre correspondiente
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame
```

4) Seguido, vuelve a redimensionar la imagen al tamaño predeterminado (ya que antes estaba a ¼ por temas de velocidad), luego dibuja un cuadro alrededor del rostro con su respectivo nombre, siendo desconocido si no logro coincidir con ninguno de los conocidos (que previamente están guardados dentro de una carpeta la cual viene dentro del repositorio).

```
    # Que muestre los resultados
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Volver a resolucion original el frame, ya que el que detectamos estaba a 1/4
        top *= 2
        right *= 2
        bottom *= 2
        left *= 2

        # Dibujar una caja alrededor de la cara
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

        # Dibujar un campo de texto con el nombre de la persona abajo
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 145), cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 0.7, (255, 255, 255), 1)
```

5) Ya las últimas líneas de código son para que se este mostrando la imagen final (es decir, la imagen de video y la que contiene la información del nombre reconocido) todo el tiempo, y que, además, que al pulsar la tecla “Q” el programa se cierre y, cierre todas sus ventanas.

```
        # Que muestre la imagen final
        cv2.imshow('Video', frame)

        # Apretar Q para salir
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

video_capture.release()
cv2.destroyAllWindows()
```

Dia 22/04

El cambio del día de hoy ha sido uno chiquito. Lo que se hizo fue cambiar la versión del le guaje de programación el cual se estaba usando, por la ultima versión. Si bien existen algunos cambios, no afectan al uso de la librería ni de las dependencias.

Dia 02/05

Cambios menores a la ubicación de los archivos para que la carpeta de trabajo sea más organizada y entendible.

Dia 08/05

Se empieza a pensar en la posibilidad de que, si bien el uso de la librería fase-recognition es muy bueno y facilita bastante las cosas, quizás haya que recurrir al uso de algún modelo ya creado de “Deep-Learning” o bien, crear uno desde 0 utilizando una red neuronal convolucional. Mas adelante redactare un poco de como es su funcionamiento y porqué se usa este modelo de red neuronal cuando se trabaja con la clasificación de imágenes, en este caso, de imágenes de rostros.

Dia 09/05

Se encargan las cosas necesarias para poder llevar el proyecto afuera de la PC con la que se esta trabajando, dichas cosas son:

- Raspberry Pi 3B+
- Memoria microSD de 32gb Clase 10
- Cable HDMI (para poder conectarlo a un monitor)

Tiempo estimado de llegada de productos: 1 semana.

Dia 17/05

Hoy han llegado los productos, por lo que se procede a instalar un sistema operativo en el RaspBerry. Por preferencia, se instala Ubuntu MATE y se clona el repositorio dentro de la Raspberry Pi.

Dia 08/06

Luego de unas charlas con los profes de prácticas profesionalizantes, se ha pegado un cambio profundo al proyecto. Ahora va a ser un sistema de ascensor para personas ciegas, el cual detecte la cara de una persona ciega y a partir de ahí, active los comandos por vos, cosa que la persona pueda decir “ir al tercer piso” o “subir al piso 3” y que, de esta manera, el ascensor interprete esta información y lleve a esta persona al piso 3.

Este cambio supone el uso de una maqueta para representar lo que seria el ascensor, además de un motor de 6v con el que se pueda elevar dicha maqueta.

Entonces para crear este sistema de reconocimiento de voz, debido al apuro que existe, se decidió buscar una librería que haga este trabajo. Fue ahí cuando se encontró como una de las primeras opciones “Speech_recognition”, la cual usa varios motores para reconocer el sonido. Puede usar uno de Google, uno que puede funcionar sin internet, entre otros.

Dia 13/06

Se sigue intentando fusionar speech_recognition con el proyecto. Actualmente se le hicieron unos cambios menores para que pueda funcionar con otros lenguajes y que maneje posibles errores que se puedan aparecer sin crashear, tales como que no haya una conexión a internet (en caso de usar un motor que si utilice internet) o que no comprenda lo que dijo la persona, o cuando dice algo inentendible.

Además, para integrar bien el como se usa la librería, se hizo un pequeño juego donde el programa elige una fruta al azar de una lista, y vos tenés que averiguar qué fruta eligió, todo esto por medio de la voz.

Dia 18/06

Se observa que el uso de speech_recognition con Google es medianamente lento para funcionar en el ascensor, por lo que se observa el uso de PocketSphinx como motor. Este motor lo bueno que tiene es que funciona sin internet, tiene una muy buena velocidad y no consume mucho, por lo que su uso en una RaspBerry es bastante acertado. Aunque las partes negativas afectan peor, la precisión que tiene deja mucho que desear, aunque esto se puede arreglar grabando las cosas con tu propia voz y haciendo un diccionario chico, el cual comprenda las posibles palabras que la persona va a decir y no más. Además, hay que enseñarle como son las entonaciones para las palabras y el sonido que hacen. Tiempo estimado de este trabajo: 20 horas, incluso para un diccionario chico.

También se analiza el hecho de como va a ser la GUI (Graphical User Interface), básicamente es la parte grafica del programa. En primera instancia se había pensado que la configuración del dispositivo se hiciera en un programa y este esté almacenado en el mismo RaspBerry. Luego apareció una idea aun mejor, ya que nos permitiría configurar el dispositivo desde cualquier computadora que este conectada a la misma red Wifi. El problema con este método, el cual es el mas atractivo, es que no se tiene ni siquiera un mínimo de conocimiento como poder hacerlo, porque se pensó que podía ser una pagina donde vos te crees una cuenta y vincules el ID de tu dispositivo con tu cuenta, y de ahí tengas un panel de control donde modificar tanto la base de datos de las personas, como poder añadir, eliminar, configurar tiempos de respuesta, etc.

Este trabajito requeriría una semana mínimo, ya que es bastante pesado y además hay que leer muchísimo y practicar bastante para que no pueda ser vulnerado con facilidad. Esto supone también un trabajito de seguridad informática, el cual no se va a cubrir con profundidad ya que tampoco es el fin del proyecto.

Dia 22/06

Se instala PocketSphinx luego de varios intentos fallidos, ya que depende de varias librerías, las cuales algunas no son fáciles de instalar.

Una vez instalado, lo primero que se observa es que la precisión es bastante baja, reconociendo lo que decís 1 de cada 5 veces (20% de las veces). Lo segundo es que es bastante rápida, lo cual nos viene de 10 para el proyecto.