Facemask'd: sistema de reconocimiento automático de mascarillas y lector de temperatura corporal

Alberto Navalón Lillo

24 de junio de 2021

Índice

1. Descripción

Este proyecto es un sistema automático para el reconocimiento de mascarillas (quirúrgicas, FFP2, textiles, etc.), al cual se le ha incorporado la funcionalidad de leer y comprobar que el usuario tiene una temperatura corporal normal (inferior a 37 °C).

Para el reconocimiento de mascarillas, se ha creado un modelo ML de reconocimiento de imágenes con Teachable Machine , que a su vez utiliza la librería Tensorflow.js para generar y entrenar dicho modelo.

1.1. Composición del dataset

Para entrenar el modelo, hemos creado un dataset de 879 imágenes extraídas de diferentes fuentes disponibles públicamente en internet. De estas 879 imágenes:

- 349 muestran a personas con la mascarilla correctamente colocada;
- 199 muestran a personas con la mascarilla puesta de forma incorrecta (bajo la nariz, por debajo de la barbilla, etc.);
- y 331 muestran a personas sin mascarilla.

1.2. Funcionamiento del modelo de predicción

Nuestro modelo, una vez exportado a `p5.js` y `ml5.js`, obtendrá la imagen de la cámara conectada, y realizará una predicción sobre ella, devolviendo así tres porcentajes (la suma de los tres será siempre 100): *Mask worn correctly* ("Mascarilla puesta correctamente"), *Mask worn incorrectly* ("Mascarilla puesta incorrectamente"), y *Without mask* ("Sin mascarilla"); todo ello acorde al contenido de la imagen.

Con el objetivo de ahorrar recursos en el lado del servidor y aumentar la eficiencia y velocidad del modelo, las funciones de análisis y predicción se ejecutan exclusivamente en el lado del cliente.

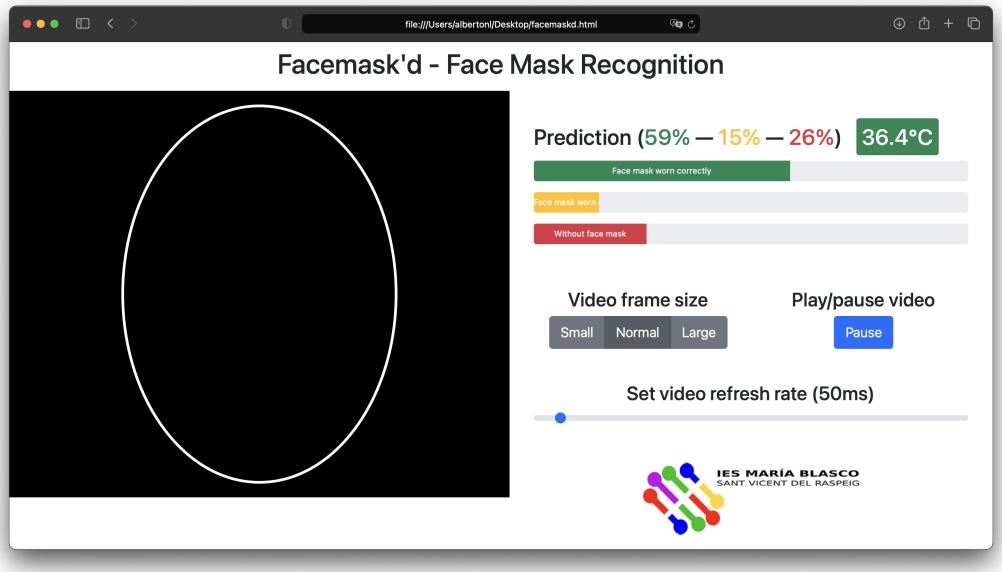
1.3. Funcionamiento de la cámara térmica

La cámara térmica, modelo MLX90641, realizará una lectura de temperatura bajo solicitud por llamada a la función `api_thermal_datapoint()` de la API, disponible en `getpoint/`, bien vía `GET` o `POST`, aunque se recomienda el primer método. El valor se devolverá al cliente en formato JSON, siguiendo esta estructura[1]:

```
{
    "point": 36.4
}
```

1.4. Estructura de la interfaz de usuario

Véase *Fig. A*.



(a) Fig. A: recreación de la interfaz de usuario. En un entorno real, dentro del recuadro negro se vería la imagen de la cámara.

La interfaz se divide esencialmente en dos columnas. A la izquierda se muestra la imagen de la cámara, dentro del recuadro. Se incluye una elipse para que el usuario se posicione de modo que la cara esté centrada, para así obtener una predicción más fiable.

[1] La temperatura se devolverá en grados celsius.

A la derecha se muestran los resultados de la predicción, representados de forma gráfica en las barras, y de forma numérica por encima de estas. El color verde representa el porcentaje para la mascarilla bien colocada (*Face mask worn correctly*), el amarillo para la mascarilla incorrectamente colocada (*Face mask worn incorrectly*), y por último, el rojo para la ausencia de mascarilla (*Without face mask*).

Junto a la representación numérica de la predicción, hallamos, dentro de un pequeño recuadro, la temperatura corporal. Este recuadro será de color verde siempre que la temperatura sea menor que 37 °C, y en rojo en el caso contrario.

Finalmente, bajo los resultados de la predicción encontramos los controles:

- **Video frame size**: cambia el tamaño de las dos columnas. Dentro de una escala de 12, *Small* es 5 : 7, *Normal* es 6 : 6, y *Large* es 7 : 5. El tamaño de las columnas por defecto es *Normal*.
- **Play/pause video**: se para la predicción y la obtención de imágenes de la cámara. En la interfaz, la imagen aparecerá congelada hasta que se active de nuevo.
- **Set video refresh rate**: modifica el intervalo de tiempo en que se ejecuta la función de predicción sobre la imagen de la cámara. Por defecto, este valor es de 50 milisegundos, pudiéndose cambiar entre 0 y 1000, aunque la *refresh rate* real puede oscilar dependiendo del uso de recursos de la máquina. La actualización de la temperatura, ajena a este valor, ocurre una vez cada 1000 milisegundos, independientemente de los cambios en este campo.

2. Componentes y montaje

El proyecto incorpora los siguientes componentes:

- Raspberry Pi 4B
- Cámara USB 1080p (cualquier cámara o webcam serviría)
- Cámara térmica MLX90641 (véase *Fig. B*)

La cámara de vídeo (webcam) no requiere ningún tipo de montaje especial. Simplemente hemos de conectarla a uno de los cuatro puertos USB que tiene nuestra Raspberry Pi.

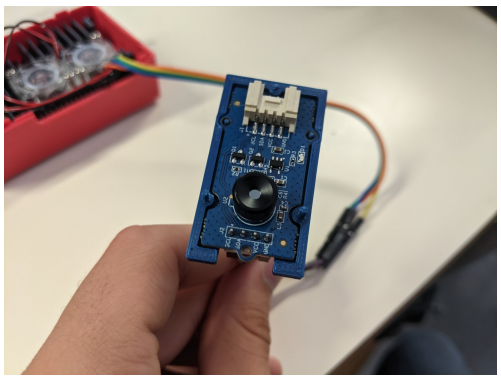
Por el contrario, la cámara térmica debe ser conectada directamente a los pines GPIO de la placa. En primer lugar, el pin GND del sensor deberá ser conectado a uno de los pines GND en la Raspberry Pi (por razones de conveniencia, utilizaremos el pin número 9; véase Configuración de los pines GPIO (en inglés)). Del mismo modo, el pin VCC del sensor será conectado al pin 1 en la placa ("3V3 Power").

Seguidamente, quedan los pines SDA y SCL, que serán conectados, respectivamente, a los pines 3 (SDA0) y 5 (SCL0) de nuestra placa, los cuales conforman la interfaz I2C de la Raspberry Pi.

3. Configuración rápida

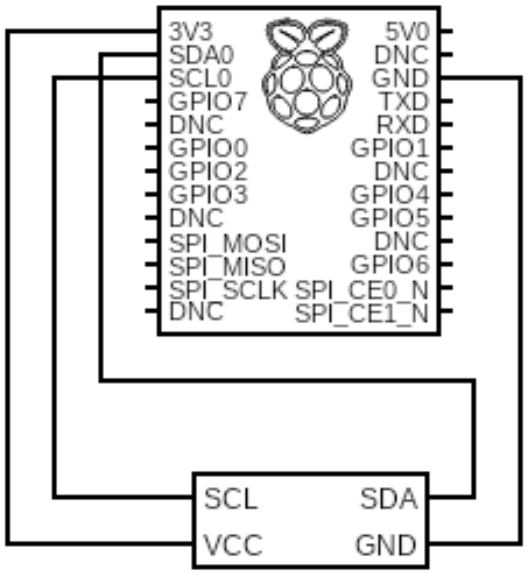
En primer lugar, es necesario instalar una distribución de Linux en nuestra Raspberry Pi (se recomienda emplear Raspberry Pi OS/Raspbian), haciendo uso de una tarjeta MicroSD que haga la función de memoria interna para el sistema operativo.

Tras esto, comprobaremos que Python está instalado (y por consiguiente, `python-pip`), y en una versión igual o superior a Python 3.6. Para ello, ejecute en una consola Bash la instrucción



(b) Fig. B: vista de la cámara térmica MLX90641.

(c) Fig. C: vista del proyecto ensamblado y terminado.



(d) Fig. D: esquema de conexiones para la cámara térmica MLX90641.

`python --version`. En caso de que Python no esté instalado, o bien sí lo esté, pero en una versión antigua, instale una nueva versión antes de continuar.

A continuación, descargue una copia del código de la última versión de Facemask'd en https://github.com/albertonl/facemaskd/releases. Si así lo desea, es posible configurar un servidor de producción para servir nuestra aplicación de Django. Sin embargo, aquí se describe el proceso para configurar un servidor temporal de desarrollo, el cual permitirá el acceso a través de `http://localhost:8000`.

3.1. Ejecución del servidor de desarrollo

Si todavía no se encuentra instalado, instale `virtualenv` vía `pip`:

```
$ python3 -m pip install virtualenv
```

Tras esto, sitúese en el directorio base de nuestro proyecto, cree un entorno virtual y actívelo:

```
$ venv env && source env/bin/activate
```

Una vez el entorno virtual esté activado, instale las dependencias del proyecto:

```
$ pip install -r requirements.txt
```

Aún desde el directorio base, aplique las migraciones iniciales:

```
$ python manage.py migrate
```

Y por último, active el servidor de desarrollo:

```
$ python manage.py runserver
```

Ahora, el proyecto estará disponible en la dirección `http://localhost:8000`. Para apagar el servidor y desactivar el entorno virtual, haga `Ctrl+C`, seguido de:

```
$ deactivate
```

Para volver a activarlo, desde el directorio base del proyecto ejecute:

```
$ source env/bin/activate
$ python manage.py runserver
```

3.2. Funcionamiento del servidor

El cliente descarga desde nuestro servidor toda la parte estática del proyecto (HTML, CSS, JavaScript, etc.), en la cual, además de la interfaz de usuario, se incluye el modelo de predicción, que se ejecuta exclusivamente de forma local, mediante las librerías `p5.js` y `ml5.js`.

Dado que no podemos interactuar con la interfaz I2C de la Raspberry Pi directamente desde JavaScript, cada 1000 milisegundos se hará una solicitud al servidor, que enviará la temperatura corporal al cliente en formato JSON, el cual será decodificado y mostrado al usuario.

Se han propuesto varias alternativas a esto, como la de enviar el dato de la temperatura mediante sockets, pero dada la naturaleza del framework Django, desarrollar un sistema asíncrono como este sería una tarea muy difícil.

En adición a esto, también se ha meditado la opción de convertir el proyecto en una aplicación nativa basada en Chromium, utilizando el framework Electron. No obstante, ante la dificultad de interactuar con la interfaz I2C de la Raspberry Pi para obtener los datos recogidos por la MLX960641 con Node.js, esta opción ha sido descartada por el momento.

4. Código fuente

El código fuente, recogido bajo la licencia GNU General Public License v3.0 (GNU GPLv3), está disponible en GitHub:

`https://github.com/albertonl/facemaskd`

También podrá encontrar la última versión estable del proyecto (se recomienda usar esta a la hora de configurar una placa) en la página *Releases*:

`https://github.com/albertonl/facemaskd/releases`

La última versión estable, a 23 de junio de 2021, es la versión `v1.0`:

`https://github.com/albertonl/facemaskd/releases/tag/v1.0`