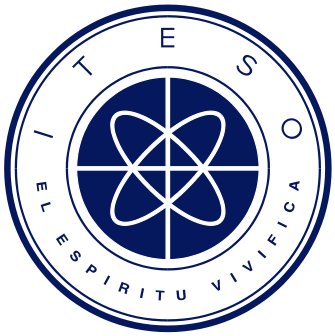
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.



RECONOCIMIENTO DE PATRONES EN IMÁGENES CON UN SISTEMA EMBEBIDO

Tesina para obtener el grado de:

Especialista en sistemas embebidos

Presenta(n)

Rodolfo Santiago Lara Núñez

Martín Jhonatan Mares Ruiz

Director del Trabajo:

Dr. Luis Enrique González Jiménez

San Pedro Tlaquepaque, Jalisco. Diciembre de 2016.

Agradecimientos

Queremos agradecer a las personas que nos apoyaron en el transcurso de la especialidad y el desarrollo del trabajo.

Empezando con nuestras familias que siempre contamos con su ayuda e incondicional apoyo, sobre todo al impulsarnos a seguirnos desarrollando académicamente.

Además agradeciendo a nuestro asesor Dr. Luis Enrique González Jiménez por guiarnos en el desarrollo del proyecto, y por último les damos las gracias al ITESO y al CONACYT por brindarnos la oportunidad de poder seguir preparándonos académicamente y de seguir alcanzando nuestras metas.

Resumen

Este trabajo se enfoca en el desarrollo e implementación de un programa capaz de procesar Imágenes en un sistema embebido. Los objetivos principales fueron el reconocimiento facial y la detección de una señal de tránsito, enfocado en el proceso de identificar las señales de *Stop*. Para cada finalidad se emplearon diferentes algoritmos para procesar las imágenes y obtener el resultado deseado. Se realizaron pruebas funcionales para verificar el comportamiento de los métodos y se reportaron los resultados experimentales.

Abstract

This project is focused in the development and implementation of a program capable of processing images in an embedded system. The principal objectives are the facial recognition and the detection of one traffic signal, focused to identify the road sign of STOP. By each purpose were used algorithms to process the images to obtain the expected result. It were performed functional tests to verify the behavior of the methods implemented and also the experimental results were documented.

Lista de figuras.

Figura 1: Diagrama a bloques del sistema propuesto 6
Figura 2: Referencia archivo formato ISO 9
Figura 3: Referencia formato a SD card. 9
Figura 4: Referencia USB bootlable con Linux. 10
Figura 5: Referencia USB bootlable con Linux configuraciones finales. 10

Lista de tablas

Tabla 1: Características de la tarjeta de desarrollo. 7
Tabla 2: Resultados de reconocimiento de la señal Stop 19
Tabla 3: Continuación de resultados de reconocimiento de la señal Stop 20
Tabla 4: Continuación de resultados de reconocimiento de la señal Stop 21
Tabla 5: Resultados de reconocimiento de rostros 22
Tabla 6: Continuación de resultados de reconocimiento de rostros 23

Contenido

Resumen ii
Abstract iii
Lista de figuras. iv
Lista de tablas v
Contenido vi
Introducción 1
1. Enfoque del sistema 2
1.1. Antecedentes 2
1.2. Objetivos 2
1.3. Descripción Funcional 2
1.4. Fundamentación Teórica 3
2. Desarrollo del diseño 6
2.1. Diagrama a bloques del sistema propuesto 6
2.2. Especificaciones técnicas 7
2.3 Recursos necesarios 8
3. Implementación. 9
3.1. Instalación de Ubuntu 14.04 LTS 9
3.1.1 Actualización de Ubuntu 11
3.1.2 Preparación del entorno para Opencv. 11
3.1.3 Compilación de OpeCV 12
4. Reconocimiento de señal de tráfico “STOP” 13
4.1. Entrenamiento. 13
4.2. Preparación del conjunto de imágenes negativas. 13
4.3. Preparación del conjunto de imágenes positivas. 14
4.4. Creación del vector. 15
4.5. Entrenamiento. 16
5. Identificación de rostros. 17
5.1. Preparación de la base de imágenes por sujeto. 17
5.2. Entrenamiento del identificador de rostros. 17
5.3. Código de implementación. 17
6. Pruebas y Resultados 19
6.1. Pruebas del sistema de reconocimiento de imágenes... **¡Error! Marcador no definido.**
6.2. Prueba de imágenes Stop. 19
6.3. Pruebas de reconocimiento de cara 22
7. Conclusiones y Trabajo Futuro 24

8. Referencias Bibliográficas 25
9. Apéndice..... 26
9.1. Detector de Stop (Imágenes de Fondo) 26
9.2. Detector de Stop (Imágenes positivas) 27
9.3. Identificación de rostro. 29
9.4. Prueba detección de Stop. 33

Introducción

Este trabajo se enfoca en la implementación de OpenCV en un sistema embebido para ser capaz de identificar objetos.

La solución fue planteada para que sea de bajo costo y tenga gran capacidad de procesamiento y reconozca objetos deseados por el usuario, dado el uso de una tarjeta embebida de bajo costo que cualquier usuario pueda adquirir y el lenguaje Python de scripting que corre en múltiples plataformas se considera un sistema con gran flexibilidad y escalabilidad posible.

En esta era en donde el desarrollo de aplicaciones es cada vez más compleja, en las cuales se requieren más poder de procesamiento y más recursos (memoria, periféricos y conectividad), permite que las aplicaciones sean más versátiles y más factibles de incorporar y posicionar, es por eso que un sistema capaz de procesar imágenes en un sistema embebido resulta un medio para la base y complemento de muchos fines, ya que el reconocimiento facial puede aplicarse a medios de seguridad e identidad, y el reconocimiento de señales de tráfico para soluciones de movilidad y seguridad.

El Capítulo 1 describe el enfoque del sistema, menciona los antecedentes y objetivos del trabajo, ofrece una descripción funcional y fundamentos teórico/técnica del sistema.

El Capítulo 2 presenta arquitectura de la solución al problema, describiendo tanto los elementos de hardware, de firmware y los requerimientos detallados de lo necesario para implementar la solución.

El Capítulo 3 se hace una descripción de cómo se llevará acabo la implementación de la solución planteada esto comprende toda la preparación necesaria para poder lograr que OpenCV sea funcional en un sistema embebido.

El Capítulo 4 menciona la implementación del sistema para que sea capaz de reconocer la señal de tráfico de Stop.

El Capítulo 5 describe la implementación de la Identificación de rostros en nuestro sistema embebido.

El Capítulo 6 menciona las pruebas y los resultados de ambos sistemas de detección.

El Capítulo 7 plasma las conclusiones y el trabajo futuro que se espera que en algún momento alcance este proyecto.

El Capítulo 8 menciona las referencias bibliográficas que se tocan a lo largo de este documento.

El Capítulo 9 incluye los apéndices en los cuales podemos encontrar descripciones cortas de los códigos implementados para hacer el sistema y el código fuente.

1. Enfoque del sistema

La siguiente sección describe el concepto y la concepción del sistema de detección y procesamiento de imágenes mediante un sistema embebido, pasando por una breve introducción de los métodos y las herramientas para obtener el resultado en cada finalidad deseada ya sea en la identificación de rostros o detección de señales de *Stop* en la vialidad.

1.1. Antecedentes

Antes de seguir con los detalles de la implementación, es importante el mencionar las razones de por qué se decidió a realizar el sistema. Aunque el procesamiento de imágenes no sea algo nuevo, la implementación de hacerlo portable en un sistema embebido es lo que le da pie a variaciones de aplicaciones tomando en cuenta la versatilidad de la potencia de procesamiento y del tamaño de la tarjeta. Tomando el procesamiento de la imagen como herramienta y apoyo a aplicaciones más amplias, sin meter hardware dedicado o complejo. Así corroborando que las tarjetas que están saliendo al mercado pueden adecuarse a procesamientos más complejos.

1.2. Objetivos

El objetivo del Proyecto es de realizar el procesamiento de imágenes usando una tarjeta embebida con Linux, para obtener la detección de rostros y la identificación de la señal de "STOP". Todo esto aprovechando las librerías de OpenCv[9] y los recursos de la tarjeta, así aplicando métodos de procesamiento de imagen para detectar la finalidad que se requiere. Manejando así un sistema de alto procesamiento y de amplios recursos de conectividad y de almacenamiento, aun relativo bajo costo en comparativa a lo alto de una tarjeta dedicada a solo realizar el procesamiento de la imagen, ampliando el uso de la aplicación, manteniendo estos recursos disponibles.

1.3. Descripción Funcional

El sistema tiene la funcionalidad de obtener una imagen y dependiendo de la finalidad se centraría en reconocer el rostro de alguna persona y localizarla en el plano de la imagen o si no la funcionalidad de identificar si en la imagen está situada una señal de "STOP" de vialidad y así mismo detectar si se encontró y marcarla dentro de la imagen.

La detección de rostros y de la señal de "STOP" se realizaran utilizando el método de los clasificadores Cascada de Haar, mediante el uso de la librería de OpenCV[9]para el procesamiento de imágenes.

La detección de objetos utilizando los clasificadores de cascada Haar es un método de detección de objetos efectivo propuesto por Paul Viola y Michael Jones en su artículo "Detección Rápida de Objetos utilizando una Cascada de Características Simples" [6].

Inicialmente el algoritmo necesita imágenes positivas (imágenes de objetos a ser reconocidos) e imágenes negativas (imágenes sin el objeto) para entrenar al clasificador. Entonces necesitamos extraer características particulares del objeto a reconocer. Cada característica es un valor único obtenido restando la suma de píxeles bajo el rectángulo blanco y de la suma de píxeles bajo el rectángulo negro.

Al estar el sistema entrenado para detectar e identificar los rostros o las señales de “STOP”, el sistema podrá identifica ya sea si se encuentra o no el objeto requerido a detectar en la imagen.

1.4. Fundamentación Teórica

El principal recurso para el desarrollo del sistema es la librería de procesamiento de imágenes de libre uso llamado OpenCV [9].

Open Source Computer Vision Library

OpenCV (Open Source Computer Vision Library) [9] es una biblioteca con licencia BSD de código abierto que incluye varios cientos de algoritmos de visión computarizada. OpenCV tiene una estructura modular, lo que significa incluye varias bibliotecas compartidas o estáticas.

OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. Su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Una gran característica de OpenCV es que es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Por lo que es una gran opción al usarlo en una tarjeta embebida con Linux para la implementación del sistema de reconocimiento de imágenes.

Clasificador Haar

Para poder obtener la detección de objetos se realiza un método llamado clasificador de Haar. Haar son las características de una imagen digital utilizadas en el reconocimiento de objetos. Deben su nombre a su similitud intuitiva con las ondas de Haar y se utilizaron en el primer detector de cara en tiempo real [4].
Una característica de tipo Haar considera las regiones rectangulares adyacentes en una ubicación específica en una ventana de detección, resume las intensidades de píxeles en cada región y calcula la diferencia entre estas sumas. Esta diferencia se utiliza entonces para categorizar las subsecciones de una imagen. Un ejemplo puntual es una base de datos de imágenes con rostros humanos. De los cuales es una característica común que entre todas las caras la región de los ojos es más oscura que la región de las mejillas. Por lo tanto, un rasgo común para la detección de rostros es un conjunto de dos rectángulos adyacentes que se encuentran por encima del ojo y la región de la mejilla. La posición de estos rectángulos se define

en relación con una ventana de detección que actúa como un cuadro delimitador al objeto deseado (la cara en este caso). En la situación de la detección de las señales de “STOP” su característica tipo Haar considera la forma distintiva de la señal siendo un octágono y así también detectando las letras internas de la señal.
Un gran número de características de tipo Haar son necesarias para describir un objeto con suficiente precisión. Las características de tipo Haar se requieren organizan en algo llamado cascada de clasificador para formar un aprendiz o clasificador eficiente.

Entrenamiento clasificador en cascada

Un clasificador en cascada es un clasificador impulsado por las características de Haar, el cual es entrenado bajo las muestras de un objeto particular (usados en este sistema los rostros y las señales de “STOP”), llamados ejemplos positivos y a la par con muestras de ejemplos negativos, los cuales son imágenes arbitrarias. Todas escaladas a un mismo tamaño ya sean positivas o negativas [5].

La palabra "cascada" en el nombre del clasificador significa que el clasificador resultante consta de varios clasificadores más simples que se aplican posteriormente a una región de interés hasta que en algún momento el candidato es rechazado o todas las etapas pasan.

El proceso de un clasificador en cascada incluye dos etapas principales, las cuales son entrenamiento y detección.

Para la etapa de entrenamiento es necesario un conjunto de muestras. Hay dos tipos de muestras: negativas y positivas.

Las muestras positivas corresponden a imágenes con objetos detectados, que en este caso nos interesan los rostros y las señales de “STOP” en la vialidad. Las muestras positivas pueden crearse a partir de una colección de imágenes previamente marcadas.

Las muestras negativas corresponden a imágenes sin objeto. Por lo tanto, las imágenes consideradas como negativas, no deben de presentar algún indicio del objeto que se requiere reconocer. Estas imágenes no deben contener objetos detectados. Las muestras negativas también se denominan muestras de fondo. Las imágenes pueden ser de diferentes tamaños pero de relación igual, cada imagen debe ser más grande o igual que un tamaño de imagen de entrenamiento.

La etapa de detección sigue después de que un clasificador es entrenado, el cual se focaliza en una región de interés (del mismo tamaño usado durante el entrenamiento) en una imagen de entrada. El clasificador emite una serie de posibles coincidencias si es probable que la región muestre el objeto (rostro o la señal de “STOP”) y en caso contrario no emite ningún resultado de probabilidad. Para buscar el objeto en toda la imagen se puede mover la ventana de búsqueda a través de la imagen y comprobar cada ubicación utilizando el clasificador. El clasificador está diseñado para que pueda ser fácilmente redimensionado para poder encontrar los objetos de interés en diferentes tamaños, lo que es más eficiente que cambiar el tamaño de la propia imagen. Por lo tanto, para encontrar un objeto de un tamaño desconocido en la imagen el procedimiento de exploración debe hacerse varias veces a diferentes escalas.

MinnowBoard

La Fundación MinnowBoard.org es una organización sin ánimo de lucro con sede en Estados Unidos que proporciona educación y promoción del diseño y uso de software y hardware de código abierto en computación en arquitectura Intel [1]. MinnowBoard.org apoya los principios de la Asociación de Hardware de Código Abierto (OSHWA.org) y pone sus diseños a disposición del público para que "cualquiera pueda estudiar, modificar, distribuir, fabricar y vender el diseño o hardware basado en ese diseño". Las MinnowBoards están diseñados para ofrecer un rendimiento excepcional, flexibilidad, apertura y estándares. Estas placas integradas tienen una enorme gama de capacidades para desarrolladores de hardware, software y firmware: desde la creación de un proyecto de hobby o profesional, hasta el desarrollo de aplicaciones o productos integrados de alto rendimiento.

Linux

Linux es un sistema operativo GNU. todo su código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera, bajo los términos de la GPL (Licencia Pública General de GNU) [2]. Se utilizó un sistema operativo en base a Linux por la distribución libre, la versatilidad y la compatibilidad con python y el recurso básico para el desarrollo del sistema que es el OpenCV[9], todo esto resulta perfectamente acoplado gracias a la tarjeta de desarrollo minnowboard.

2. Desarrollo del diseño

La solución del problema plateado es utilizar una tarjeta de desarrollo basada en tecnología de microprocesadores Intel, en ella se construirá el entorno funcionalmente de un sistema de reconocimiento de objetos por medio de imágenes previamente establecidas, además de esto el sistema de reconocimiento de objetos contará con un sistema operativo Linux con la distro de Ubuntu en el cual se instalará Python como el leguaje de scripting y OpenCV [9].

2.1. Diagrama a bloques del sistema propuesto

A continuación, se muestra el diagrama a bloques que conformaría3 el sistema que se propone a realizar.

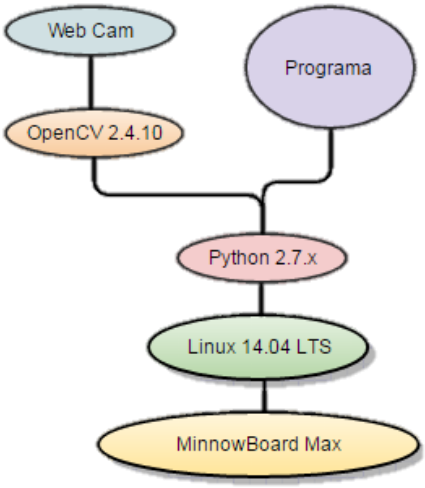


Figura 1: Diagrama a bloques del sistema propuesto.

En la Figura 1 se muestra el diagrama a bloques del sistema propuesto para realizar la identificación y reconocimiento de objetos en imágenes capturadas con un cámara, como se puede observar en el diagrama en la parte inferior tenemos la base o la plataforma de donde iniciaremos el desarrollo, la MinnowBoard Max será nuestra placa base para usar como sistema mínimo, se escogió esta placa debido a las prestaciones y capacidades de procesamiento para llevar a cabo las tareas más críticas como lo será el hecho de procesar una imagen y ser capaz de operar con Linux y una distro de él, Linux será nuestro sistema operativo debido a que el sistema es una distribución libre de licencia lo cual nos permite distribuir y utilizar software libre, debido a la creciente popularidad de scripting se basara este sistema en el lenguaje Python por sus prestaciones y portabilidad ya que el leguaje se utiliza de igual forma entre sistemas operativos soportados. Python será quien albergara todas las sub partes del sistema de reconocimiento e identificación de objetos ya que OpenCV[9] es compatible con Linux y en este caso también lo es con Python, en dicho lenguaje tendremos un scripting que se encargara de llevar acabo los procesos para identificar y detectar un objeto, la cámara WEB será nuestra entrada al sistema ya que como bien sabemos de ellas se obtendrán las imágenes a utilizar o procesar.

2.2. Especificaciones técnicas

MinnowBoard Max

Categoría	Características
Núcleo	1. 64-bit Intel® Atom™ E38xx Series 2. Intel HD Graphics Linux OS
Memoria	1. DDR3 RAM 2. 8 MB SPI Flash System Firmware Memory
Video	1. Intel HD Graphics (1920x1080 máxima resolución) 2. HDMI 1.4a (micro HDMI conector)
Audio	1. Digital vía HDMI
I/O	1. Micro SDSDIO 2. SATA2 3Gb/sec 3. USB 3.0 (host) 4. USB 2.0 (host) 5. Serial debug via FTDI cable 6. 10/100/1000 Ethernet RJ-45
Aditamentos extras	1. 8 x Buffered GPIO pins (2 pins support PWM) 2. I2C & SPI bus 3. 2 x 16550 HS UARTs, one with CTS/RTS 4. System Firmware Flash conector de programación
Dimensiones de la tarjeta	1. 99 x 74mm (2.9 x 3.9in)
Rango de temperatura	1. 0 – 40 Celsius
Poder	1. 5V (min 2.5A) DC
Sistemas operativos soportados	1. Debian GNU, Ubuntu, Fedora, Linux Mint 2. Yocto Project Compatible 3. Android 4.4 (Kitkat) and 5.0 (Lollipop) System 4. Microsoft Windows 8.1 and 10
Arranque	1. UEFI Firmware 2. Coreboot 3. SageBIOS

Tabla 1: Características de la tarjeta de desarrollo.

2.3 Recursos necesarios

Para la correcta implementación del sistema de reconocimiento de objetos será necesario:

1. Conexión a Internet.
2. Computadora con sistema operativo Windows.
3. Tarjeta de desarrollo MinnowBoard Max.
4. Convertidor AC/DC para la alimentación de la Minnoboard (5V @ 3A).
5. Teclado – USB (Para MinnowBoard Max).
6. Mouse – USB (Para MinnowBoard Max).
7. Port Hub – USB 3 conectores mínimo (Para MinnowBoard Max).
8. Adaptador WIFI - USB compatible con Linux (Para MinnowBoard Max).
9. Tarjeta SD mínima capacidad de 16Gb (Para MinnowBoard Max).
10. Monitor compatible con DVI (Para MinnowBoard Max).
11. Cable HDMI a DVI (Para MinnowBoard Max).
12. Webcam Compatible con Linux (Para MinnowBoard Max).
13. Linux 14.04 LTS (versión estable) imagen iso(Para MinnowBoard Max).
14. Opencv 2.4.10
15. Python 2.7.x
16. 1000 Fotografías del objeto a reconocer formato jpg.
17. 1000 Fotografías sin el objeto a reconocer formato jpg.
18. Memoria USB de 4Gb
19. Rufus Installer para la creación de la memoria bootable.

3. Implementación.

En esta sección se llevara el proceso completo de la preparación para el entorno de desarrollo, desde los archivos fuente que el usuario necesitara para poder instalar OpenCV[9] y culminando por la instalación de todos los complementos necesarios para tener el entorno listo.

3.1. Instalación de Ubuntu 14.04 LTS

Descargar la imagen de Ubuntu 14.04 LTS en el siguiente link: releases.ubuntu.com/14.04/ c de preferencia se debe de utilizar un archivo en formato ISO como se muestra en la siguiente imagen.

Figura 2: Referencia archivo formato ISO

Después de este paso procederemos a formatear la unidad en donde esta colada nuestra SD card.

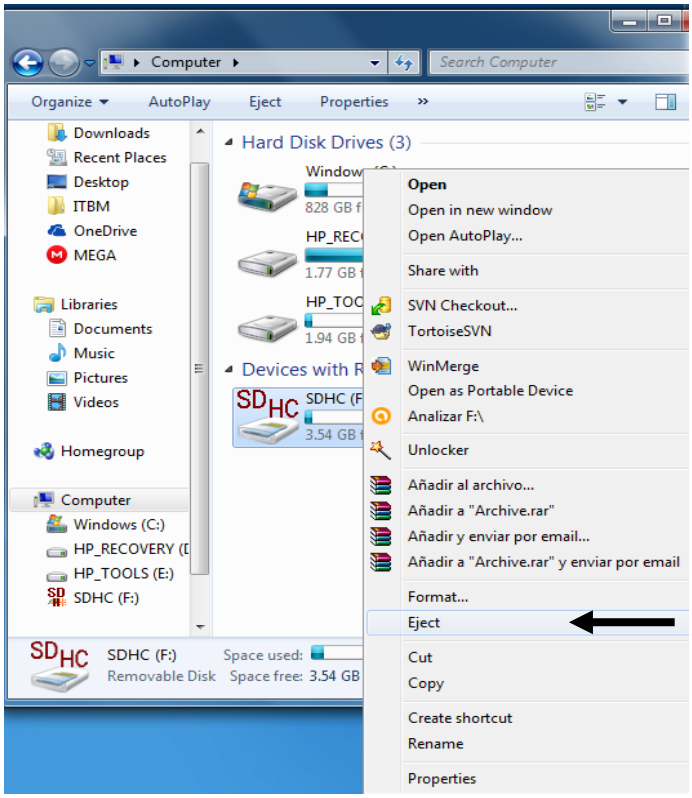


Figura 3: Referencia formato a SD card.

A continuación, con la aplicación Rufus crearemos una USB bootable con el sistema operativo Linux.

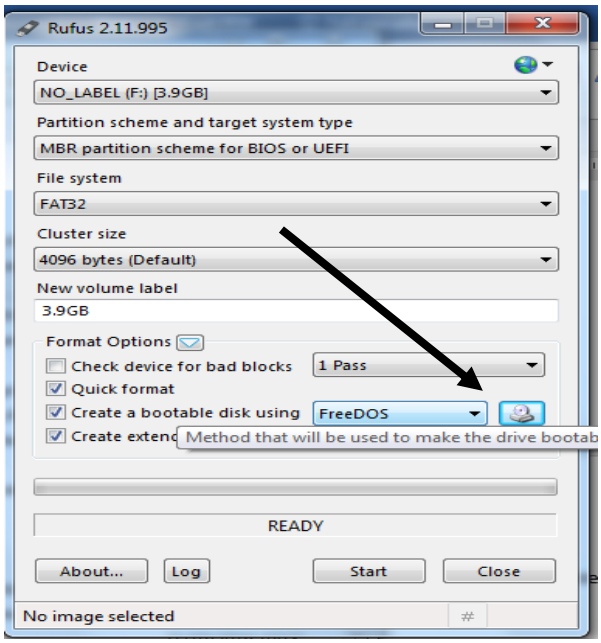


Figura 4: Referencia USB bootlable con Linux.

Donde se encuentra la flecha procederemos a seleccionar la imagen de Ubuntu 14.04 LTS, la aplicación a continuación se mostrará de la siguiente forma y presionaremos Start.

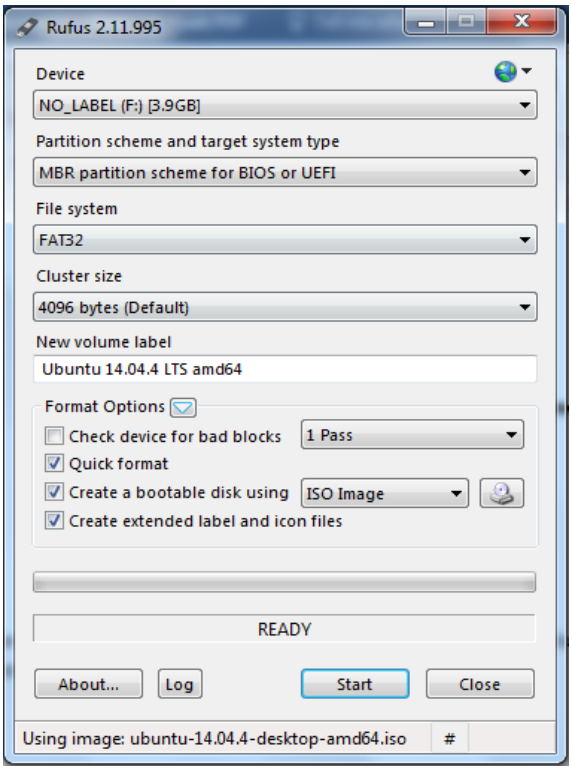


Figura 5: Referencia USB bootlable con Linux configuraciones finales.
releases.ubuntu.com/14.04/ [7]

Estos pasos previos son cruciales para instalar Ubuntu. La instalación de Ubuntu se puede consultar en la siguiente liga: https://www.ubuntu.com/download/desktop/install-ubuntu-desktop [8]

Una vez que ya se tiene la tarjeta funcionando correctamente con el sistema operativo Linux será necesario que esta cuente con una conexión a internet para eso será necesario el adaptador Wifi o en dado caso una conexión de red alámbrica ya que los siguientes pasos se basaran en la instalación de todo el entorno para poder utilizar la librería OpenCV[9] en el sistema Linux de la mano de Python como lenguaje de programación.

3.1.1 Actualización de Ubuntu

Para llevar a cabo esta tarea empezaremos actualizando el repositorio de nuestro sistema eso nos garantizara que contaremos con la más reciente versión disponible todo nuestro entorno, la siguiente línea se debe de introducir desde línea de comandos.

sudo apt-get update

Con nuestros repositorios actualizados será necesario que actualicemos todos los archivos relacionados con esos repositorios para eso ejecutaremos el siguiente comando.

sudo apt-get upgrade

3.1.2 Preparación del entorno para Opencv.

Las distros de Ubuntu por defecto traen instalada la versión de Python 2.7.x con la cual nosotros podremos trabajar, sin embargo, en caso de querer cambiar la versión será necesario investigación propia del individuo.

Empezaremos con los requisitos para OpenCV[9].

Los siguientes paquetes constan de una serie de requeridos y opcionales para nuestra implementación se instalarán en su totalidad.

Paquetes Requeridos:

1. GCC 4.4.x o superior
2. CMake 2.6 o superio
3. Git
4. GTK+2.x o superio, including headers (libgtk2.0-dev)
5. pkg-config
6. Python 2.6 o superior
7. Numpy 1.5 o superio con la paqueteria de desrrollado
8. ffmpeg o libav development packages: libavcodec-dev, libavformat-dev, libswscale-dev
9. [opcional] libtbb2 libtbb-dev
10. [opcional] libdc1394 2.x
11. [opcional] libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev

Todos estos requisitos para el entorno de OpenCV[9] pueden ser instalados desde la línea de comando de Ubuntu.

sudo apt-get install build-essential

sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev

sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

3.1.3 Compilación de OpeCV

Los siguientes comandos nos ayudaran a bajar del repositorio la versión de OpenCv 2.4.10 [9] y alojar la información en una dirección o ruta deseada en nuestra tarjeta de desarrollo [3].

cd ~/<my_working _directory>

git clone https://github.com/opencv/opencv.git

Con nuestro repositorio descargado, continuaremos ingresando a nuestro repositorio de Opencv[9] en cual crearemos un folder llamado release ahora nos cambiamos a la ruta de reléase, en dicha ruta empezaremos la configuración de la compilación.

cd ~/opencv

mkdir release

cd release

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..

Ya configurado como tendremos que compilar OpenCV [9] pasaremos a compilarlo para posteriormente instalarlo como se muestra en los siguientes comandos.

make

sudo make install

4. Reconocimiento de señal de tráfico “STOP”

En esta sección empezaremos a entrar a trabajar con librería OpenCV [9] en la cual se basará todo el procesamiento de imagen. Empezaremos hablando del algoritmo en el cual tomaremos como base para hacer nuestra implementación, este algoritmo se denomina Cascade Classifier y se divide principalmente en 2 etapas las cuales son el entrenamiento y la detección.

4.1. Entrenamiento.

Para el entrenamiento serán necesarias imágenes positivas esto en otras palabras son imágenes que contienen el principal objeto a identificar y a su vez necesitaremos imágenes negativas o de fondo en las cuales no debe de aparecer el objeto a detectar, se debe de considerarse que para poder trabajar con las imágenes de una manera más eficiente deben de ser similares en relación de ancho y alto ya que de lo contrario el procedimiento de entrenamiento se puede prolongar bastante en tiempo, además recomendamos que todas las imágenes sean manejada en escala de grises aunado a estos temas es importante mencionar que la cantidad de imágenes positivas y negativas debe ser de igual valor 1:1 esto quiere decir que por cada fotografía positiva deberá de existir una negativa.

4.2. Preparación del conjunto de imágenes negativas.

Para las imágenes negativas será necesario crear un archivo txt que se denominará negatives.txt el cual contendrá la ruta absoluta de las imágenes de fondo o negativas el formato del texto se debe visualizar parecido al mostrado en la parte inferior.

C:\Users\martin.mares\Downloads\ProyectoDeReconocmientoDeImagenes\TrafficSignalRecognizer\SignalCropper\StopSignal_False\FalseReadyImages\FalseReadyImage0.jpg

C:\Users\martin.mares\Downloads\ProyectoDeReconocmientoDeImagenes\TrafficSignalRecognizer\SignalCropper\StopSignal_False\FalseReadyImages\FalseReadyImage1.jpg

C:\Users\martin.mares\Downloads\ProyectoDeReconocmientoDeImagenes\TrafficSignalRecognizer\SignalCropper\StopSignal_False\FalseReadyImages\FalseReadyImage10.jpg

C:\Users\martin.mares\Downloads\ProyectoDeReconocmientoDeImagenes\TrafficSignalRecognizer\SignalCropper\StopSignal_False\FalseReadyImages\FalseReadyImage100.jpg

Como se puede observar son consecutivas las líneas y solamente cuentan con fin de línea en el último carácter.

Para este paso nosotros desarrollamos el siguiente código que se agregó en apéndice 5.1 el cual podemos observar está escrito Python, el cual realiza el siguiente procedimiento:

1. Obtiene todas las rutas absolutas de las imágenes almacenadas en la carpeta.
2. Se procesan de la siguiente forma cada imagen.
    a) Se les realiza un cambio de resolución el cual respeta el formato (4:3) y la resolución de las imágenes se definió a (160:120).
    b) Conversión a escalas de grises.
    c) Creación de una imagen temporal.
    d) Escritura de la imagen temporal a su destino final con un numero consecutivo.
3. Obtiene todas las rutas absolutas de las imágenes obtenidas anteriormente.
    a) Se escriben todas las rutas absolutas en el documento negatives.txt.

4.3. Preparación del conjunto de imágenes positivas.

En caso de las imágenes positivas deben estar contenido en un archivo con el nombre de positives.txt pero para las positivas es muy importante que la información quede en el siguiente formato:

```
img1.jpg  1  140 100 45 45
img2.jpg  2  100 200 50 50   50 30 25 25
```

Como se muestra en la parte superior la primera parte de la line va colocada la ruta del archivo con la ruta absoluta, el siguiente número que se muestra es la cantidad de objetos contenidos en la imagen como se ve en la línea superior la imagen contiene solo un elemento y la linea inferior cuenta con 2 elementos, a continuación se debe de especificar de donde a donde se encuentra la imagen aunado a esto se debe de recordar que la relación ancho y alto debe ser igual en todas las imágenes por lo que posiblemente será necesario realizar un ajuste de tamaño, en orden de izquierda a derecha el primer nombre nos dice dónde empieza nuestro objeto en la coordenada X, seguido de la coordenada en el eje Y, cabe mencionar que la cuenta de pixeles empieza de la esquina superior izquierda hacia la esquina inferior derecha, el siguiente numero especifica el ancho en el eje X seguido de la profundidad en el eje Y resumiendo "AbsPath Object# X Y W H".

Para la preparación de este archivo se desarrolló el siguiente código que se encuentra en el apéndice 5.2.

1. Obtiene todas las rutas absolutas de las imágenes almacenadas en la carpeta.
2. Se procesan de la siguiente forma cada imagen.
    a. Se detecta en la imagen donde se encuentra la señal de “STOP”, dicha detección es muy básica por lo que es necesario corroborar si en todas las imágenes se ubicó de forma correcta el objeto.
    b. Se extrae esa parte de la imagen.
    c. Se le realiza un cambio de resolución el cual respeta el formato (4:3) y la resolución de las imágenes se definió a (40:30).
    d. Conversión a escalas de grises.
    e. Creación de una imagen temporal.
    f. Escritura de la imagen temporal a su destino final con un numero consecutivo.
3. Obtiene todas las rutas absolutas de las imágenes obtenidas anteriormente.
    a) Se escriben todas las rutas absolutas en el formato adecuado en el documento negatives.txt.

4.4. Creación del vector.

La creación del vector se ejecuta con la siguiente línea de comando:

opencv_createsamples.exe -info positives.txt -vec stopsignalvector.vec -num # -maxxangle 0.1 -maxyangle 0.1 -maxzangle 0.1 -maxidev 100 -bgthresh 0 -w 40 -h 30

En comando realiza una seria de distorsiones y modificaciones de las propiedades de la imagen para de esta forma crear una simple identidad que represente todas las posibles variaciones de luz y entorno y de esta forma tener un vector solido en otras palabras a mayor cantidad de imágenes mayor precisión, como resultado obtendremos un archivo en formato vec el cual será necesario para nuestro siguiente paso.

4.5. Entrenamiento.

El siguiente párrafo representa la línea de comando a utilizar para poder entrenar el reconocedor de imágenes, deberemos de tener en cuenta que e l número de stages es un factor muy importante en este proceso ya que en base a este número será la precisión que el sistema pueda tener.

opencv_traincascade.exe -data cascades -vec stopsignalvector.vec -bg negatives.txt -numStages 12 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos # -numNeg # -w 40 -h 30 -mode ALL -precalcValBufSize 2048 -precalcIdxBufSize 2048

Una vez ejecutada esta línea obtendremos un archivo XML el cual será nuestro archivo que engloba todo para poder identificar objetos en este caso la señal de “STOP.”

5. Identificación de rostros.

El reconocimiento de rostro se basa en 2 pasos, entrenar el reconocedor y verificar si la persona se encuentra en la imagen en realidad es un proceso bastante sencillo más sin embargo conlleva bastante trabajo preparar la información, para poder entrenar nuestro reconocedor es necesario tener fotos de la cara del sujeto a identificar, debemos tomar en cuenta que a mayor cantidad de fotos aumenta considerablemente el porcentaje de eficiencia esto en otras palabras representa mayor asertividad del reconocedor.

5.1. Preparación de la base de imágenes por sujeto.

En la preparación de la base del sujeto será necesario utilizar el reconocimiento de rostros que ya provee la librería de OpenCV [9] con esto seriamos capaces de identificar donde se encuentra el rostro para posteriormente extraerlo, escalarlo a una resolución deseada, pasarlo a escala de grises para posteriormente alojarlo en un directorio deseado.

5.2. Entrenamiento del identificador de rostros.

El entrenamiento del identificador consiste en pasarle las rutas de las imágenes al identificador para que el crees la identidad del sujeto además de cada imagen que se le provee es necesario pasarle como argumento cual sujeto es el que se está reconociendo.

5.3. Código de implementación.

En el apéndice 5.3 se puede observar el código que de manera muy superficial se explicara, este script corre en un entorno Python por lo que la portabilidad es transparente y eso nos ayuda a que cualquier problema que se presente en la tarjeta pueda ser reproducido y corregido en una computadora de escritorio si es que así se deseara realizar la labor.

El script se basa en un menú de 6 opciones las cuales se describen a continuación:

1. Agregar una persona a la base de reconocimiento.

   Esta función se encarga de crear todos los recursos para poder llevar acaba el entrenamiento posterior.

2. Entrenar al reconocedor con una persona existente.

   Ya que se agrega la persona el siguiente paso es entrenar al sistema para que sea capaz de identificar el rostro de la persona, para llevar a cabo este entrenamiento es necesario tomarse una sesión reducida en número de fotos, recordemos que entre mayor sea el número de fotografías mayor será la posibilidad de identificar al sujeto.

3. Eliminar a una persona de la base de reconocimiento.

   Como su nombre lo menciona simplemente con esta función seremos capaces de eliminar a alguna persona de la base de reconocimiento.

4. Ver personas existentes en la base.

   Con facilidad se puede entender que el comando simplemente desplegara en la consola el nombre de las personas existentes a reconocer.

5. Jugar a reconocer persona.

   Este comando nos da la facultad de tomar una imagen y ver si es posible reconocer a alguna persona, de ser posible también nos mostrara en consola el porcentaje con el cual fui identificada la persona.

6. Salir.

   Este comando cierra la aplicación.

6. Pruebas y Resultados

En este capítulo se muestran los resultados obtenidos en base a las experimentaciones realizadas en el reconocimiento de la señal de tráfico Stop, luego observaremos los resultados de la experimentación con el script que se encarga de identificar rostros, estas pruebas se realizaron con la tarjeta de desarrollo previamente mencionada en capítulos anteriores, como entrada del sistema tenemos imágenes aleatorias obtenidas con una cámara web.

6.1. Prueba de imágenes Stop.

Esta prueba consiste en introducirle al sistema 10 fotos que a continuación se mostraran y el sistema debe ser capaz de reconocer. Se probó con el código de prueba que se encuentra en el apéndice 9.3.

Imágenes a reconocer

Imagen	Reconocimiento	Resultado
		Reconocido
		No Reconocido

Tabla 2: Resultados de reconocimiento de la señal Stop.

Imagen	Reconocimiento	Resultado
		No reconocida
		No reconocida
		Reconocida
		Reconocida

Tabla 3: Continuación de resultados de reconocimiento de la señal Stop.

Imagen	Reconocimiento	Resultado
		Reconocida
		No reconocida.
		No reconocida.
		No reconocida

Tabla 4: Continuación de resultados de reconocimiento de la señal Stop.

En estos resultados como se puede observar podemos observar que no tas las señales de STOP fueron reconocidas y esto se debe a la cantidad de fotografías positivas y numero de stages para realizar el entrenamiento, las imágenes positivas nos brinda una serie de escenarios posibles para el objeto a identificar por ende entre mayor número de imágenes con diferentes ángulos intensidad luminosa y todas las características que una fotografía puede albergar entre mayor sea el número de variaciones mayor probabilidad de reconocimiento tendrá el sistema, el número de stages también es un factor clave en el entrenamiento ya que a mayor número de stages más refinado se volverá el resultado, en el proceso de creación se experimentaron con diversos número de stages para los cual se obtuvo una relación que a mayor números de stages mayor probabilidad de reconocimiento es posible pero este número va de la mano con el tiempo

que toma para entrenar el sistema en este caso el sistema conto con 20 stages y requirió aproximadamente 3 dias, también se debe de tomar en cuenta la capacidad de procesamiento en el cual se llevaran a cabo la creación del XML.

6.3. Pruebas de reconocimiento de cara.

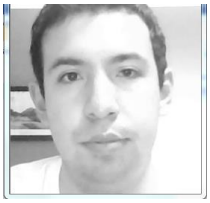
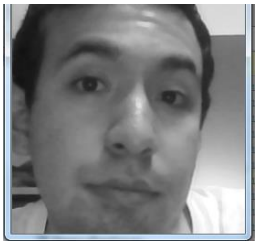
Fotografía Tomada	Sujeto Encontrado	Identificación
	['[Martin]'] 281.971684466	Correcta
	['[Martin]'] 136.392197891	Correcta
	["['[Martin]']"] 84.9605929906	Correcta
	["['[Martin]']"] 149.979099813	Correcta
	["['[Martin]']"] 212.077101353	Correcta

Tabla 5: Resultados de reconocimiento de rostros

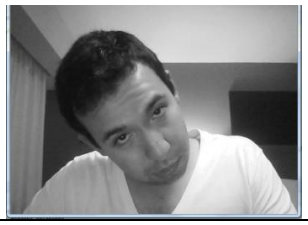
Fotografía Tomada	Sujeto Encontrado	Identificación
	["['[Martin]']"] 254.133598504	Correcta
	["['[Martin]']"] 124.184054953	Correcta
	No reconocido	Incorrecta

Tabla 6: Continuación de resultados de reconocimiento de rostros

En la tabla 3 observamos la clara identificación del sujeto este resultado solamente se obtuvo con un acervo de aproximadamente 12 imágenes la cual nos brindó un excelente resultado pese a la cantidad, de igual forma en este caso se repite la misma conclusión que en pocas palabras es entre mayor número de imágenes con la mayor variación posible son empleada para el entrenamiento mayor capacidad tendrá de identificar el objeto.

7. Conclusiones y Trabajo Futuro

A lo largo de este proceso que nos ha sido grato realizar, se han observador bastantes áreas de oportunidad para proyectos futuros ya que OpenCV [9] es una herramienta poderosa capaz de realizar diferentes procesamientos a las imágenes, brindando una versatilidad al desarrollo de sistemas, teniendo así una amplia diversificación de usos finales. El procesamiento de imágenes puede ser el principal objetivo de un sistema o el complemento de un compuesto más complejo. Brindando la posibilidad de implementar variadas propuestas de soluciones a problemas que tenga la sociedad, como aplicaciones de seguridad, confirmación de identidades, actuadores en detecciones de errores o de peligro, mapeo de rutas para accesos, etc. Con el desarrollo de este proyecto se ampliarán las posibilidades a resolver más problemas que se cuentan en la actualidad.

En un futuro próximas generación serán capaces de emplear sistemas similares, pero con mayor precisión y enfocadas a aplicaciones específicas debido a la creciente capacidad de procesamiento en sistemas de bajo costo.

Esta implementación nos deja con las bases y el conocimiento práctico, que nuestro sector laboral demanda, debido a que día a día se busca la inteligencia artificial en los vehículos de uso común, y este tipo de subsistemas brindan grandes aportaciones para poder cumplir con todas esas nuevas metas que nos resta por alcanzar laboralmente y tecnológicamente.

8. Referencias Bibliográficas

[1]. http://wiki.minnowboard.org/MinnowBoard_Wiki_Home

[2]. http://www.linux.org/

[3]. http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html

[4]. http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

[5]. https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/

[6]. https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[7]. http://releases.ubuntu.com/14.04/

[8]. https://www.ubuntu.com/download/desktop/install-ubuntu-desktop

[9]. http://opencv.org/

9. Apéndice

9.1. Detector de Stop (Imágenes de Fondo)

Este código produce una lista de imágenes de fondo o negativas, dicha lista se coloca en un archivo txt en el cual la lista está compuesta de las rutas absolutas de las imágenes, una vez producido este documento las imágenes se redimensionan para poder adoptar una relación de 4:3 y una resolución de 160*120, una vez terminado este proceso la imagen se cambia a escala de grises.

```
import sys,os,glob,cv2,glob
import numpy as np

def stop():
    lst_BaseDirectory    = os.path.abspath(os.curdir)
    lrg_DirectoryList    = os.listdir("./")
    lub_StopSignalIndex  = lrg_DirectoryList.index("StopSignal")
    lub_NegativeIndex    = lrg_DirectoryList.index("StopSignal_False")
    lst_NegativePath     = os.path.abspath(lrg_DirectoryList[lub_NegativeIndex])
    lst_StopSignalPath   = os.path.abspath(lrg_DirectoryList[lub_StopSignalIndex])
    os.chdir(lst_NegativePath)
    lrg_FalseStopSignalPictureList = os.listdir("./")
    lub_FalseReadyImagesIndex = lrg_FalseStopSignalPictureList.index("FalseReadyImages")
    lst_FalseReadyImagesPath = os.path.abspath(lrg_FalseStopSignalPictureList[lub_FalseReadyImagesIndex])
    os.chdir(lst_StopSignalPath)
    lrg_StopSignalPictureList = os.listdir("./")
    os.chdir(lst_NegativePath)
    lrg_NegativeImagesList = os.listdir("./")

    for files in lrg_NegativeImagesList:
        if not files.endswith(".jpg"):
            lrg_NegativeImagesList.remove(files)

    for lub_FalsePictureIndex in range(lrg_NegativeImagesList.__len__()):
        lrg_FalseStopSignal = cv2.imread(lrg_NegativeImagesList[lub_FalsePictureIndex])
        lrg_FalseStopSignal = cv2.resize(lrg_FalseStopSignal, (160,120), fx=1, fy=1, interpolation=cv2.INTER_CUBIC)
        lrg_FalseStopSignal = cv2.cvtColor(lrg_FalseStopSignal, cv2.COLOR_BGR2GRAY)
        cv2.imwrite('ReadyImageTemp.jpg', lrg_FalseStopSignal)
        os.rename(os.path.abspath('ReadyImageTemp.jpg'),lst_FalseReadyImagesPath + '//FalseReadyImage' +
str(lub_FalsePictureIndex) + '.jpg')

    os.chdir(lst_FalseReadyImagesPath)

    lrg_FalseStopSignalPictureList = os.listdir("./")

    for files in lrg_FalseStopSignalPictureList:
        if not files.endswith(".jpg"):
            lrg_FalseStopSignalPictureList.remove(files)

    with open('negatives.txt', 'w') as in_files:
        for FalseImages in range(lrg_FalseStopSignalPictureList.__len__()): in_files.write(
            os.path.abspath(lrg_FalseStopSignalPictureList[FalseImages]) + '\n')


def main():
    stop()
if __name__ == "__main__":
    main()
```

9.2. Detector de Stop (Imágenes positivas)

Este código se encarga de reconocer la señal de tráfico STOP el reconocimiento está basado en las cualidades de la señal como su forma la cantidad de lados los colores y las letras más sin embargo no es lo suficientemente robusto para poderse usar como reconocedor de señal, de la misma manera este script crea una lista de imágenes con las cuales entrenaremos el detector de Stop, la lista contiene sus rutas absolutas y además cabe destacar que se redimensiona la imagen para cumplir con una relación 4:3 además de ajustar el tamaño a 40*30 una vez que se tienen en su tamaño y relación necesarios se procede a pasarla a escalas de grises.

```
import sys,os,glob,cv2,glob
import numpy as np

def stop():
   lst_BaseDirectory    = os.path.abspath(os.curdir)
   lrg_DirectoryList    = os.listdir("./")                              # current directory
   lub_StopSignalIndex  = lrg_DirectoryList.index("StopSignal")                 # get the index of the
StopSignal
   lst_StopSignalPath   = os.path.abspath(lrg_DirectoryList[lub_StopSignalIndex])       # get the full path of
the StopSignal
   os.chdir(lst_StopSignalPath)                                    #change to the actual working directory
   lrg_StopSignalPictureList = os.listdir("./")                          #get the complete list of  files on the
current working folder
   lub_ReadyImagesIndex  = lrg_StopSignalPictureList.index("ReadyImages")            # get the index of
the ReadyImages
   lst_ReadyImagesPath = os.path.abspath(lrg_StopSignalPictureList[lub_ReadyImagesIndex])  # get the
full path of the ReadyImages

   for files in lrg_StopSignalPictureList:
      if  not files.endswith(".jpg"):
         lrg_StopSignalPictureList.remove(files)

   for lub_PictureIndex in range(lrg_StopSignalPictureList.__len__()):
      lrg_CurrentPicture = cv2.imread(lrg_StopSignalPictureList[lub_PictureIndex])
      luw_Widht = np.size(lrg_CurrentPicture,1)
      luw_Height = np.size(lrg_CurrentPicture,0)

      lrg_AreasContours = []
      lrg_FilterContour = []
      lrg_CurrentPicture =
cv2.resize(lrg_CurrentPicture,(640,480),fx=1,fy=1,interpolation=cv2.INTER_CUBIC)
      lrg_CurrentPicture_HSV = cv2.cvtColor(lrg_CurrentPicture, cv2.COLOR_BGR2HSV)

      lub_LowerRed = np.array([0, 0,0],dtype=np.uint8)
      lub_UpperRed = np.array([125, 90,255], dtype=np.uint8)
      lrg_Mask = cv2.inRange(lrg_CurrentPicture_HSV,lub_LowerRed,lub_UpperRed)
      lrg_Mask = cv2.morphologyEx(lrg_Mask, cv2.MORPH_CLOSE,None)

      cv2.imshow('Current Picture Mask', lrg_Mask)
      luw_Contours,lul_trash = cv2.findContours(lrg_Mask,1,2)

      for cnt in luw_Contours:
         approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
         if len(approx)== 8:
```

```
            lrg_AreasContours.append(cv2.contourArea(cnt))

        lrg_AreasContours.sort(None,None,True)

        for cnt in luw_Contours:
            approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
            if len(approx)== 8:
                if lrg_AreasContours[0] == cv2.contourArea(cnt):
                    lrg_FilterContour = cnt

        luw_RecX, luw_RecY, luw_RecW, luw_RecH =cv2.boundingRect(lrg_FilterContour)
        lrg_StopSignalCropped = lrg_CurrentPicture[luw_RecY:(luw_RecY+luw_RecH),luw_RecX:
(luw_RecX+luw_RecW)]
        lrg_StopSignalCropped = cv2.resize(lrg_StopSignalCropped, (40, 30), fx=1, fy=1,
interpolation=cv2.INTER_CUBIC)
        lrg_StopSignalCropped = cv2.cvtColor(lrg_StopSignalCropped, cv2.COLOR_BGR2GRAY)

#cv2.rectangle(lrg_CurrentPicture,(luw_RecX,luw_RecY),(luw_RecX+luw_RecW,luw_RecY+luw_RecH),(0,25
5,0),2)
        cv2.imwrite('ReadyImageTemp.jpg', lrg_StopSignalCropped)
        os.rename(os.path.abspath('ReadyImageTemp.jpg'), lst_ReadyImagesPath + '//ReadyImage'+
str(lub_PictureIndex) + '.jpg' )

    os.chdir(lst_ReadyImagesPath)
    lrg_ReadyImagesList = os.listdir("./")

    for files in lrg_ReadyImagesList:
        if  not files.endswith(".jpg"):
            lrg_ReadyImagesList.remove(files)

    with open('positives.txt', 'w') as in_files:
        for ReadyImages in range(lrg_ReadyImagesList.__len__()):
in_files.write(os.path.abspath(lrg_ReadyImagesList[ReadyImages])+'\n')

def main():
    stop()
if __name__ == "__main__":
    main()
```

9.3. Identificación de rostro.

Este script se encarga de identificar el rostro en base a una base de imágenes que pertenecen a diferentes sujetos, principalmente se compone de 3 bloques, el primero es la creación de la base de conocimiento o imágenes por sujeto entre más imágenes por sujeto se aumenta el nivel de eficiencia e identificación de rostro, el siguiente bloque se encarga de entrenar el identificador en base a las imágenes proporcionadas, el tercer bloque básicamente se especializa en capturar una imagen detectar si en él hay un rostro y si existe ver si existe alguna relación entre el rostro detectado y algún sujeto en la base de conocimiento.

```
import cv2
import csv
import os
import shutil
import time
import numpy as np
from PIL import Image
#********************************************** Variables para la
definicion del Crop
BaseWidth = 255
BaseHeight = 255
AverageWidth = 0
AverageHight = 0
#********************************************** Variable para saber la
informacion de la imagen
ImageProperties = 0
#********************************************** Variables para la base
de datos de la informacion
DB_Subject = list()
DB_Labels = list()
DB_Images = list()
EstadoPersona =
{'Feliz','Triste','Sorprendido','Natural','Lentes','Adormecido','Perfil_
Izquierdo','Perfil_Derecho','Frontal_Superior','Frontal_Inferior','Cansa
do'}
#********************************************** LBPH Face Recognizer
recognizer = cv2.createLBPHFaceRecognizer()
#********************************************** Carga el path del
archivo xml para poder identificar rostros
faceCascade =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
#********************************************** Misceleanea de
Funciones
def cls():
    os.system('cls' if os.name=='nt' else 'clear')
#********************************************** Funciones para manejo
de la base de datos(archivo)
'''
Abre el archivo que contiene los sijetos y los carga en DB_Subject
'''
def SubjectDataBaseReader():
    global DB_Subject
    file = open("SubjectDataBase.csv","r+")
    DB_Subject = csv.reader(file)
    DB_Subject = list(DB_Subject)
    file.close()
'''
Guarda el archivo que contiene los sijetos y los carga en DB_Subject
'''
def SubjectDataBaseSaver():
    file = open("SubjectDataBase.csv","wb")
```

```
    spamwriter = csv.writer(file, delimiter=' ',quotechar='|',
quoting=csv.QUOTE_MINIMAL)
    for subject in DB_Subject:
       spamwriter.writerow([subject])
    file.close()
#******************************************** Funciones para manejo
de personas
'''
Inserta en la base de datos un nombre
'''
def Append_Subject_DB(subject):
    DB_Subject.append(str(subject))
    FolderManager('DataBase',subject)
'''
Remove de la base de datos un nombre
'''
def Remove_Subject_DB(subject):
    DB_Subject.remove(str(subject))
    remove_dir((subject))
#******************************************** Funciones para manejo de
directorios
'''
Crea los folders necesarios para el sistema
'''
def FolderManager(Kind,subject):
    ensure_dir(str(Kind))
    if subject:
        ensure_dir(str(Kind)+'/'+str(subject))
def ensure_dir(path):
    if not os.path.isdir(path):
        os.mkdir(path,0755)
def remove_dir(subject):
    if os.path.isdir('Database/'+ str(subject)):
        prevdir = os.getcwd()
        os.chdir("Database")
        shutil.rmtree(str (subject))
        os.chdir(prevdir)
#******************************************** Funciones de alto Nivel
def CrearPersona():
    cls()
    Append_Subject_DB(str(raw_input("Introdusca el nombre de una
Persona: ")))
    SubjectDataBaseSaver()
def EliminarPersona():
    cls()
    print DB_Subject
    Remove_Subject_DB(str(raw_input("Introdusca el nombre de la Persona
a ELIMINAR: ")))
    SubjectDataBaseSaver()
def ObtenerCara():
    global BaseWidth
    global BaseHeight
    global faceCascade
    #Capturamos  una Foto y leemos un fotograma
    frame = cv2.VideoCapture(0)
    #Tomamos una foto
    _, image = frame.read()
    #La convertimos  escalas de grises
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #Detectamos caras en la imagen
    face = faceCascade.detectMultiScale(
        image_gray,
        scaleFactor=1.4,
        minNeighbors=5,
        minSize=(40, 40),
        flags = cv2.cv.CV_HAAR_SCALE_IMAGE
    )
    print "Caras Encontradas {0}".format(len(face))
```

```
    #Cargamos las cordenadas y hacemos el crop
    crop_img = image_gray[face[len(face)-1][1]:face[len(face)-
1][1]+face[len(face)-1][3],face[len(face)-1][0]:face[len(face)-
1][0]+face[len(face)-1 ][2]]
    cv2.imshow("Fotografia" ,crop_img)
    #Dibujamos en la imagen original  un rectanbgulo verde
    for (x, y, w, h) in face:
        cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 1)
    cv2.imshow("Cara Extraida" ,image)
    # Escalamos la foto para tener una resolucion standar definida por
BaseWidth y BaseHeight
    ImageProperties = crop_img.shape
    AverageWidth = float(ImageProperties[1])/float(BaseWidth)
    AverageHight = float(ImageProperties[0])/float(BaseHeight)
    crop_img=
cv2.resize(crop_img,(int(float(ImageProperties[1])/AverageWidth),int(flo
at(ImageProperties[0])/AverageHight)))
    cv2.imshow("Cara Extraida y Ajustada al tamano" ,crop_img)
    cv2.waitKey(1000)
    cv2.destroyAllWindows()
    frame.release()
    return crop_img
def CapturarPersona():
    global EstadoPersona
    print DB_Subject
    subject =str(raw_input("Introdusca el nombre de la Persona a
Entrenar: "))
    prevdir = os.getcwd()
    if os.path.isdir('Database/'+ str(subject)):
        os.chdir("Database/"+ str(subject))
        for estado in EstadoPersona:
            try:
                print ("Perfil o con " + estado)
                os.system('pause')
                CaraRec = ObtenerCara()
                cv2.imwrite(subject + '.'+ estado+ ".jpeg",CaraRec)
                print ('Perfil o con ' + estado + ' identificado')
            except:
                print ("Perfil o con " + estado)
                os.system('pause')
                CaraRec = ObtenerCara()
                cv2.imwrite(subject + '.'+ estado+ ".jpeg",CaraRec)
                print ('Perfil o con ' + estado + ' identificado')
        os.chdir(prevdir)
def Entrenar(path):
    global DB_Subject
    global DB_Labels
    global DB_Images
    global recognizer
    BaseDir = os.getcwd()
    subject_paths = os.listdir(path)
    os.chdir("DataBase/")
    SubjDir = os.getcwd()
    for subjects in subject_paths:
        subject_image =  os.listdir('./'+subjects)
        os.chdir(subjects+"/")
        for subject_images in subject_image:
            image_pil = cv2.imread(subject_images, 0)
            DB_Images.append(image_pil)
            subject_indx = [i for i, s in enumerate(DB_Subject) if
subjects in str(s)]
            DB_Labels.append( int(subject_indx[0]))
        os.chdir(SubjDir)
    os.chdir(BaseDir)
    subject_paths = os.listdir(path)
    recognizer.train(DB_Images, np.array(DB_Labels))
def Reconocer():
    global recognizer
```

```
    global BaseWidth
    global BaseHeight
    global faceCascade
    frame = cv2.VideoCapture(0)
    #Tomamos una foto
    _, image = frame.read()
    #La convertimos  escalas de grises
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #Detectamos caras en la imagen
    face = faceCascade.detectMultiScale(
        image_gray,
        scaleFactor=1.4,
        minNeighbors=5,
        minSize=(40, 40),
        flags = cv2.cv.CV_HAAR_SCALE_IMAGE
    )
    print "Caras Encontradas {0}".format(len(face))
    #Cargamos las cordenadas y hacemos el crop
    crop_img = image_gray[face[len(face)-1][1]:face[len(face)-
1][1]+face[len(face)-1][3],face[len(face)-1][0]:face[len(face)-
1][0]+face[len(face)-1 ][2]]
    cv2.imshow("Fotografia" ,crop_img)
    ImageProperties = crop_img.shape
    AverageWidth = float(ImageProperties[1])/float(BaseWidth)
    AverageHight = float(ImageProperties[0])/float(BaseHeight)
    crop_img=
cv2.resize(crop_img,(int(float(ImageProperties[1])/AverageWidth),int(flo
at(ImageProperties[0])/AverageHight)))
    cv2.imshow("Cara Extraida y Ajustada al tamano" ,crop_img)
    nbr_predicted, conf = recognizer.predict(crop_img)
    cv2.waitKey(1000)
    cv2.destroyAllWindows()
    print str(DB_Subject[nbr_predicted])
    print conf

#****************************************** Programa
SubjectDataBaseReader()
Entrenar('./DataBase')
var = 0
while(var != 6):
    print 'Sistema de reconocimiento facial\n'
    print '1 - Agregar una persona a la base de reconocimiento'
    print '2 - Entrenar al reconocedor con una persona existente'
    print '3 - Eliminar a una persona de la base de reconocimiento'
    print '4 - Ver personas existentes en la base'
    print '5 - Jugar a reconocer personas'
    print '6 - Salir\n'
    var = int(raw_input("Elija una opcion: "))
    if var == 1:
        CrearPersona()
    elif var == 2:
        CapturarPersona()
    elif var == 3:
        EliminarPersona()
    elif var == 4:
        cls()
        print DB_Subject
        os.system('pause')
        cls()
    elif var ==5:
        cls()
        Reconocer()
        os.system('pause')
        cls()
```

9.4. Prueba detección de Stop.

Este script se encarga de leer el XML producido por nuestro entrenador y analizar la imagen en curso y reconocer la señal de Stop si se reconoce la señal de Stop se colocara un recuadro.

```
import cv2
import numpy as np

stop_cascade = cv2.CascadeClassifier('cascade.xml')
img= cv2.imread('prueba(1).jpg')
img= cv2.resize(img, (640, 480), fx=1, fy=1, interpolation=cv2.INTER_CUBIC)
stops = stop_cascade.detectMultiScale(img,1.3, 1, cv2.cv.CV_HAAR_SCALE_IMAGE, (40,30))
print stops
for (p, q, r, s) in stops:
   cv2.rectangle(img, (p, q), (p + r, q + s), (255, 0, 0), 6)
cv2.imshow("output", img)
cv2.waitKey(0)

cv2.destroyAllWindows()
```