DISEÑO DE UN SOFTWARE EMPLEANDO TÉCNICAS DE VISIÓN POR COMPUTADOR, OPENCV Y GPS PARA EL RECONOCIMIENTO Y GEOLOCALIZACIÓN EN TIEMPO REAL DE SEÑALES DE TRÁNSITO REGLAMENTARIAS

Presentado por:

OSCAR ALBERTO ROA VALBUNA

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERIA
INGENIERIA ELECTRÓNICA

NEIVA
2017

DISEÑO DE UN SOFTWARE EMPLEANDO TÉCNICAS DE VISIÓN POR COMPUTADOR, OPENCV Y GPS PARA EL RECONOCIMIENTO Y GEOLOCALIZACIÓN EN TIEMPO REAL DE SEÑALES DE TRÁNSITO REGLAMENTARIAS

Presentado por:

OSCAR ALBERTO ROA VALBUNA

Trabajo de grado para optar por el título de Ingeniero Electrónico:

Director

JOSÉ DE JESÚS SALGADO PATRÓN

Magister en Ingeniería Electrónica y de Computadores

UNIVERSIDAD SURCOLOMBIANA

FACULTAD DE INGENIERIA

INGENIERIA ELECTRÓNICA

NEIVA

2017

Nota de aceptación:

Firma del director del proyecto

Firma del primer jurado

Firma del segundo jurado

5 de febrero de 2018

Este trabajo es dedicado a mis padres y hermano, por su sacrificio y apoyo incondicional en todo momento.

AGRADECIMIENTOS

Agradezco a familiares cercanos, especialmente a mi tía Emilce Roa J. por su paciencia y apoyo a lo largo de la carrera. Al cuerpo docente de la universidad por su labor educativa y finalmente agradezco a mis amigos y compañeros quienes hicieron de esta experiencia algo digno de recordar.

CONTENIDO

Pág.

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE TABLAS

Pág.

RESUMEN

En este trabajo se desarrolla una aplicación capaz de detectar, reconocer y geolocalizar en tiempo real señales de tránsito reglamentarias de velocidad. Se emplea módulos de la librería OpenCV como el clasificador cascada para detectar objetos y el algoritmo de machine learning kNN (k-Nearest Neighbors o k-vecinos más cercanos), para el reconocimiento de los dígitos de las señales de tránsito. La ubicación de las señales se realiza mediante GPS. De esta manera se logra un desempeño de la aplicación del 92 %.

Palabras clave:
Detección, reconocimiento,ubicación, GPS, OpenCV, Haar Cascade Classifier, kNN

ABSTRACT

In this work, an application capable of detecting, recognizing and geolocating real-time traffic regulatory signals is developed. OpenCV library modules are used as the cascade classifier to detect objects and the kNN machine learning algorithm (k-Nearest Neighbors or nearest k-neighbors), for the recognition of the digits of the traffic signals. The location of the signals is done by GPS. In this way, an application performance of 92 % is achieved.

Keywords:
Detection, recognition, location, GPS opencv, Haar Cascade Classifier, kNN

INTRODUCCIÓN

La visión por computador permite llevar a la práctica conceptos de diferentes áreas como física del color, óptica, electrónica, programación, sistemas de computación, etc. Dando lugar a que ésta ciencia sea capaz de interpretar la estructura de un mundo tridimensional a partir de una o varias imágenes bidimensionales de ese mundo.[1]

En este trabajo se hace uso de la librería OpenCV (Open Source Computer Vision Library). La cual cuenta con un gran enfoque para trabajos en tiempo real y una amplia gama de herramientas para el procesado de imágenes. Entre las más usadas en este trabajo estan: Filtro para la reducción de ruido como el Gaussian Blur, para detección de objetos el clasificador cascada y el algoritmo kNN para reconocimiento de caracteres.

El clasificador cascada usado en OpenCV en del tipo Haar. Para su correcto funcionamiento es necesario tomar muestras positivas (en este caso señales de tránsito reglamentarias) y muestras negativas (imágenes que no son señales de tránsito reglamentarias). De esta forma poder entrenar un clasificador propio capaz de detectar el objeto deseado y cuyo resultado sea usado por un método de machine learning.[2]

El algoritmo kNN (k-nearest neighbors o k-vecinos más cercanos) es un método de maching learning soportado en OpenCV. Este algoritmo compara pixeles para clarificarlos según corresponda.[2]

El resultado de este trabajo es la unión y uso adecuado de las diferentes técnicas, métodos y algoritmos para desarrollar una aplicación de software capaz de detectar, reconocer y ubicar mediante GPS señales de tránsito reglamentarias de velocidad.

[1] Alegre, E, Pajares, G y Escalera, A. (2016). Conceptos y Métodos en Visión por Computador. Grupo de Visión del Comité Español de Automáticas (CEA).

[2] K, Alexander Mordvintsev y Abid. https://readthedocs.org/. [En línea] 02 de Septiembre de 2017. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf.

1. PLANTEAMIENTO DEL PROBLEMA

A lo largo de la historia y en especial desde el momento en que el ser humano decidió establecerse en comunidad en un punto geográfico especifico, ha enfrentado múltiples problemas cuando establecer medios de comunicación se trata. El inconveniente más inmediato y el cual hoy día se sigue combatiendo, es la distancia que separa una población de otra. Para vencer la distancia el ser humano creó diferentes medios y así establecer un canal de comunicación, siendo los caminos la primera red que permitió un contacto entre poblaciones distantes. A medida que la población fue creciendo sus necesidades lo hicieron también, por lo que crearon su propia economía y la necesidad de llevar los productos a diferentes regiones, hicieron que las vías terrestres se convirtieran en parte fundamental del desarrollo.

Existen diferentes estudios donde se demuestra que la infraestructura vial es de suma importancia para el desarrollo de un país [3]. En un país como Colombia donde el 80 % de la carga se transporta por carretera [4] se hace notable su peso para el bienestar económico de la población. Es debido al gran impacto que tiene ésta infraestructura que en teoría debe estar en óptimas condiciones, esto incluye una señalización en la vía adecuada que indique a quienes hacen uso de ella: Curvas, límites de velocidad o cualquier condición que pueda representar algún peligro. Sin embargo, en la práctica esto no siempre es así y según informes de medicina legal la mala señalización es una de las causas de accidentes de tránsito [5]. La falta de una correcta señalización en las vías ha llevado al país a tener que pagar millones de pesos en demandas e indemnizaciones [6] a personas que sufrieron algún tipo de accidentes por este motivo. Es el Consejo de Estado quien ratifica que los organismos estatales son los responsables de estar pendientes del estado de las carreteras, hacerles mantenimiento cuando sea necesario y evitar al máximo los accidentes de lo contrario tendrán que indemnizar a quienes sufran algún tipo de accidente por un mal estado en la vía o mala señalización. [7]

[3] Perez, C. y Yanovich, D. (1999). Sector Carreteras. [online] www.corficolombiana.com. Disponible en: https://www.corficolombiana.com/WebCorficolombiana/Repositorio/Informes/IS01021999.PDF

[4] Pérez V., G. (2005). La infraestructura del transporte vial y la movilización de carga en Colombia. [online] http://www.banrep.gov.co. Disponible en: http://www.banrep.gov.co/docum/Lectura_finanzas/pdf/DTSER-64.pdf.

[5] Margeé Garcia, G. (2002). Lesiones no intensionales, Muertes en accidentes de tránsito. [online] http://www.medicinalegal.gov.co. Disponible en: http://www.medicinalegal.gov.co/documents/10180/51788/Muertestransito(3).pdf2002.pdf

[6] Garibello, A y Veloza Cano, H (2008). Accidentes por huecos y falta de señales en vías hacen que la Nación pierda cada vez más demandas. [online] http://www.eltiempo.com. Disponible en: http://www.eltiempo.com/archivo/documento/CMS-3943922

[7] Elespectador.com (2012). Estado responderá por accidentes en carreteras por mala señalización [online] http://www.elespectador.com Disponible en: http://www.elespectador.com/noticias/judicial/estado-respondera-accidentes-carreteras-mala-senalizaci-articulo-355907

Haciendo uso de la tecnología disponible hoy día, se plantea el desarrollo de una aplicación de software que sea capaz de reconocer señales de tránsito reglamentarias a través de técnicas de visión por computador y librerías como OpenCV y de esta manera geolocalizarlas, como alternativa para ver el estado de la señalización de una vía y así ayudar a solventar este inconveniente.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Diseñar e implementar una aplicación de software para el reconocimiento y geolocalización de las señales de tránsito reglamentarias en tiempo real por medio del uso de visión por computadora.

2.2. OBJETIVOS ESPECÍFICOS

- Determinar el tipo de filtro más conveniente para reducir el ruido de la imagen digital.
- Determinar y aplicar diferentes técnicas y algoritmos para la detección de bordes, segmentación, extracción de características e interpretación.
- Hacer uso de la librería de visión por computador OpenCV dado que opera bajo licencia BSD, la cual es soportada por diferentes lenguajes de programación y tiene un gran enfoque en tiempo real.
- Hacer uso de un sistema de posicionamiento global (GPS), el cual envía datos de latitud y longitud para ser ubicados en un mapa al momento en que el software detecte una señal de transito reglamentaria.
- Realizar pruebas en carretera con el software para evaluar su desempeño.
- Aplicar la teoría de visión por computador para desarrollar una aplicación de software que ofrezca un servicio a quienes diariamente circulan por las carreteras del país.
- Ayudar en la verificación del estado de deterioro de las señales de tránsito reglamentarias.

3. MARCO TEÓRICO

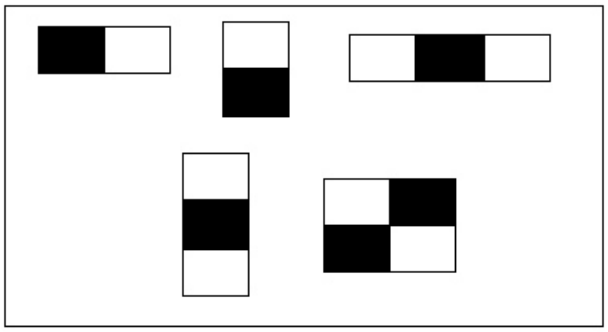
3.1. HAAR CASCADE

El clasificador cascada tipo haar es un método de detección de objetos altamente efectivo propuesto por Paul Viola y Michael Jones en su artículo, "Robust Real-Time Face Detection" en 2004 [8]. Este método se basa en el aprendizaje donde una función en cascada es entrenada a partir de muchas imágenes positivas y negativas. De acuerdo al trabajo de Paul y Michael, este clasificador está compuesto de la siguiente manera.

3.1.1. Filtros de Haar

Los filtros de haar son la combinación de varios rectángulos (entre 2 y 4), del mismo tamaño adyacentes horizontal o verticalmente[8].

Figura 1: Filtros Haar básicos



Fuente: Autor

Los rectángulos en color negro *(RN)* realizan aportes positivos al filtro y los rectángulos en color blanco *(RB)* son aportes negativos. Por tanto, el resultado del filtro es la diferencia en la suma de los valores de la intensidad *(I)*de los píxeles entre zonas en color negro y las zonas en color blanco. Lo anterior se puede representar mediante la ecuación 3.1.

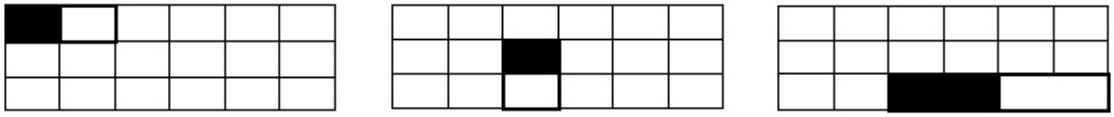
$$F_{haar} = \sum I_{RN}(x, y) - \sum I_{RB}(x, y) \tag{3.1}$$

Este resultado es necesario normalizarlo por el tamaño de la ventana del filtro para que los valores sean invariantes al cambio de tamaños de objetos e igualmente sean comparables a escalas diferentes.

[8]Jones, Paul Viola y Michael J. 2004. Robust Real-Time Face Detection. Netherlands : Kluwer Academic Publishers

Todos y cada uno de los filtros básicos se aplican en la imagen a todas las posibles escala que esta permita en horizontal y vertical. Cabe resalta que cada escala se aplica en todas las posibles posiciones en la imagen.

Figura 2: Filtros haar básicos a escalas y posiciones diferentes en la imagen



(a) Filtro básico, primera posición en la imagen

(b) Filtro básico 2, otra posición en la imagen

(c) Filtro básico, escala y posición diferente en la imagen

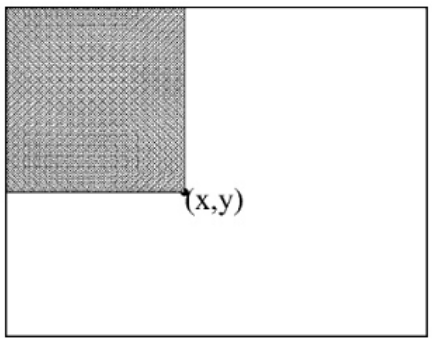
Fuente: Autor

Las características de haar son el resultante de aplicar cada uno de los filtros en cada posición y escala. Esto implica que el descriptor de la imagen sea la unión de todas las características arrojadas por cada filtro haciendo que este tenga un número muy elevado de las mismas[8].

3.1.2. Imagen integral

La imagen integral permite calcular rápidamente las características de cualquier rectángulo en la imagen. Cada píxel que conforma la imagen integral tienen un valor correspondiente a la suma de todos los píxeles arriba y a la izquierda de la imagen original[8]. La figura 3 Imagen integral en el punto x,y, muestra un ejemplo.

Figura 3: Imagen integral en el punto x,y



Fuente: Jones, Paul Viola & Michael J. Robust Real-Time Face Detection. Netherlands : Kluwer Academic Publishers, 2004.

Lo anterior se puede expresar mediante la ecuación 3.2.

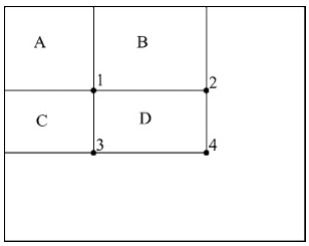
$$ii(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x',y') \tag{3.2}$$

Donde $ii(x, y)$ es la imagen integral y $i(x, y)$ es la imagen original.
Usando las ecuaciones 3.3 y 3.4.

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{3.3}$$
$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{3.4}$$

Donde $s(x, y)$ es la suma de fila acumulativa, $s(x, -1) = 0$ y $ii(-1, y) = 0$. De a cuerdo a lo expuesto por Paul y Michael, la imagen integral se puede calcular en un paso sobre la imagen original. Además utilizando la imagen integral, cualquier suma rectangular se puede calcular en cuatro matrices de referencia. Lo anterior se ilustra en la figura 4 Imagen integral, calculando un rectángulo.

Figura 4: Imagen integral, calculando un rectángulo



Fuente: Jones, Paul Viola & Michael J. Robust Real-Time Face Detection. Netherlands : Kluwer Academic Publishers, 2004.

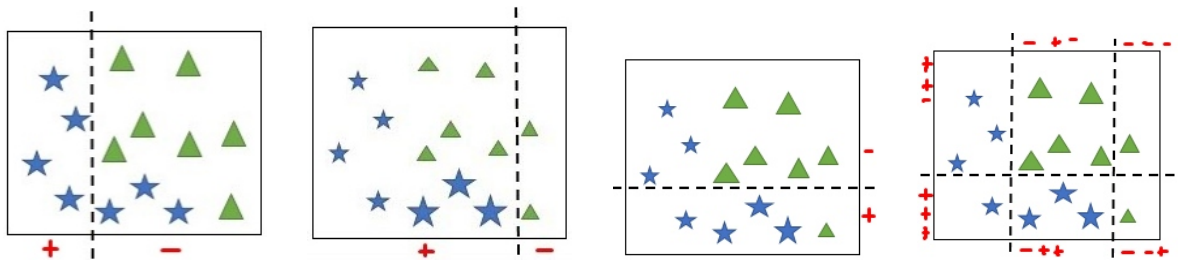
En la imagen anterior la suma de los píxeles en el rectángulo D se calcula con las cuatro matrices de referencia. Por tanto el valor de la imagen integral en la ubicación 1 es la suma de los píxeles en el rectángulo A. El valor en la posición 2 es $A + B$, en la posición 3 es $A + C$, y en la posición 4 es $A + B + C + D$. La suma dentro de D se puede calcular como $4 + 1 - (2 + 3)$ [1].

3.1.3. Método de clasificación: Adaboost

El algoritmo de aprendizaje Adaboost propuesto por Paul y Michael[8] es un proceso iterativo donde se entrenan clasificadores débiles con una única característica para combinarlos y crear un clasificador fuerte.

Un clasificador se denomina débil porque su función de clasificación tiene una tasa de error muy alta (usualmente del 49 %)[1]. La forma de establecer si la clasificación es positiva (características del objeto que se desea detectar) o negativas (características que no son del objeto), se hace estableciendo un umbral. Una vez terminada esta clasificación, para el entrenamiento del siguiente clasificador se hace énfasis en las características que quedaron mal clasificadas y se da menor énfasis a las que quedaron bien. En la figura 5 Proceso de clasificación a nivel general de Adaboost se ilustra este proceso[8].

Figura 5: Proceso de clasificación a nivel general de Adaboost



(a) Clasificador débil 1 (b) Clasificador débil 2 (c) Clasificador débil 3 (d) Clasificador fuerte

Fuente: Autor

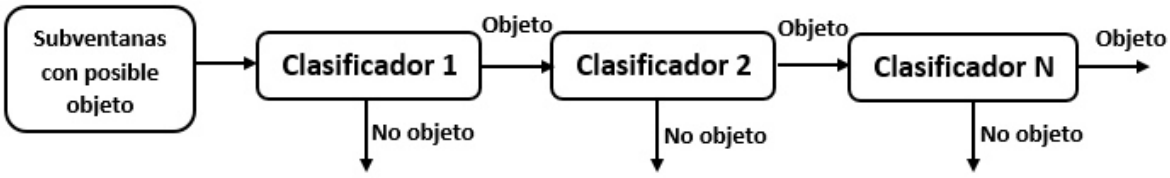
Como se observa en la figura 5, el clasificador fuerte, basado en el resultado de cada clasificador débil es capaz de delimitar correctamente la zona donde están las características positivas (mayoría de resultados positivos) y la zona de las negativas (mayoría de resultados negativos).

Cada clasificador débil determina el umbral óptimo de tal forma que obtenga la menor cantidad posible de características mal clasificadas.

3.1.4. Cascada de clasificadores

El algoritmo de cascada de clasificadores permite aumentar la eficiencia en la detección de objetos, mientras reduce de manera importante el tiempo de computo[8]. La clave radica en que los primeros clasificadores de cascada tienen pocas características, sin embargo, son capaces de descartar una gran cantidad de subventanas negativas de la imagen y detectar casi todas las positivas[8]. La figura 6 Cascada de clasificadores muestra un esquema general del clasificador.

Figura 6: Cascada de clasificadores



Fuente: Autor

Cada clasificador de cascada es entrenado mediante el método clasificación adaboost, el aprendizaje varia puesto que cada clasificador se encarga de determinar una cantidad establecida de falsos positivos (falsas detecciones del objeto) y falsos negativos (objetos no detectados)[8].

Para cumplir con el número de falsos positivos y negativos permitidos, el clasificador debe calcular un umbral óptimo que permita llegar a éste objetivo, si no es posible, se vuelve a entrenar un nuevo clasificador y se aumentan las características del mismo para lograrlo, sin embargo, puede que aún así no sea posible alcanzar el objetivo, por lo que se hace necesario establecer un número máximo de características para cada clasificador[8].

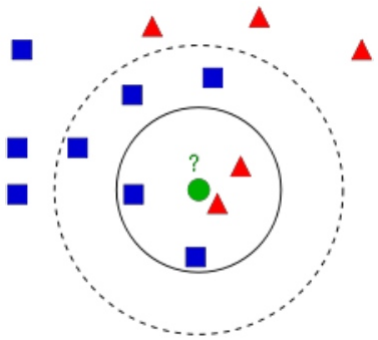
Una vez se llegue al objetivo o número máximo de características el siguiente clasificador, se centrará en las imágenes negativas que el anterior detecto como positivas repitiendo el proceso aprendizaje. A medida que el aprendizaje avanza se van añadiendo nuevos clasificadores hasta llegar al nivel de la cascada establecido o hasta llegar a un nivel aceptable de detecciones de falsos positivos y negativos[8].

3.2. MACHINE LEARNING: ALGORITMO kNN

Existen múltiples definiciones para expresar lo que es machine learning (aprendizaje automático), en términos simples, se trata de un conjunto de técnicas para enseñar a las máquinas a realizar tareas por si mimas. [9]

k-Nearest Neighbour (kNN) es un algoritmo de clasificación basado en el aprendizaje supervisado, [10] es decir, que se dispone de un conjunto de datos con sus respectivas etiquetas, indicando para cada dato la clase a la que pertenece. [11] El objetivo de este algoritmo es encontrar las coincidencias más cercanas de un dato de prueba en el espacio de características, como se muestra en la figura 7 Datos de prueba algoritmo kNN.

Figura 7: Dato de prueba algoritmo kNN



Fuente: K, Alexander Mordvintsev y Abid. https://readthedocs.org/. [En línea] 02 de Septiembre de 2017. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf.

[9]Coelho, Willi Richert Y Luis Pedro (2013). Building Machine Learning Systems with Python. s.l. : Packt Publishing Ltd., 1200713.

[10]K, Alexander Mordvintsev y Abid. https://readthedocs.org/. [En línea] 02 de Septiembre de 2017. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf.

[11]Alegre, E, Pajares, G y Escalera, A. (2016). Conceptos y Métodos en Visión por Computador. Grupo de Visión del Comité Español de Automáticas (CEA).

Como se puede observar en la figura 7 Datos de prueba algoritmo kNN, existen dos clases, una es el conjunto de cuadrados azules y la otra, el conjunto de los triángulos rojo. Cada dato de cada clase está proyectado en un espacio de características 2D (coordenadas x,y). El circulo verde representa el dato de prueba que se quiere clasificar.

Para clasificar el dato de prueba kNN tiene en cuenta principalmente dos aspectos como son: El primero es la distancia entre los datos, el cual por defecto la calcula con la fórmula euclidiana[12]. Ecuación 3.5.

$$D_{(p,q)} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \tag{3.5}$$

El segundo aspecto es la cantidad de datos más cercanos (neighbour o vecinos). Esta cantidad se define mediante k[13]. Aplicando lo anterior al dato de prueba con un k = 3, el resultado se centra en la zona más cercana (circulo continuo) arrojando que este dato esta cerca a 2 triángulos rojos y 1 cuadrado azul, por tanto el dato es un triángulo rojo. Sin embargo, al varia k, por ejemplo k = 7, resultado abarca una zona más amplia (circulo entrecortado), arrojando que el dato está cerca de 5 cuadrados azules y 2 triángulos rojos, por lo que el ahora el dato de prueba pertenece a la clase de cuadrados azules[13].

Para tener resultados adecuados es necesario probar con diferentes valores de k, y al trabajar con un algoritmo kNN básico es recomendable usar valores de k impares para evitar dilemas al quedar empatados en datos de clases de diferentes, por ejemplo en la imagen, si k = 4, el resultado es 2 cuadrados azules y 2 triángulos rojos [13].

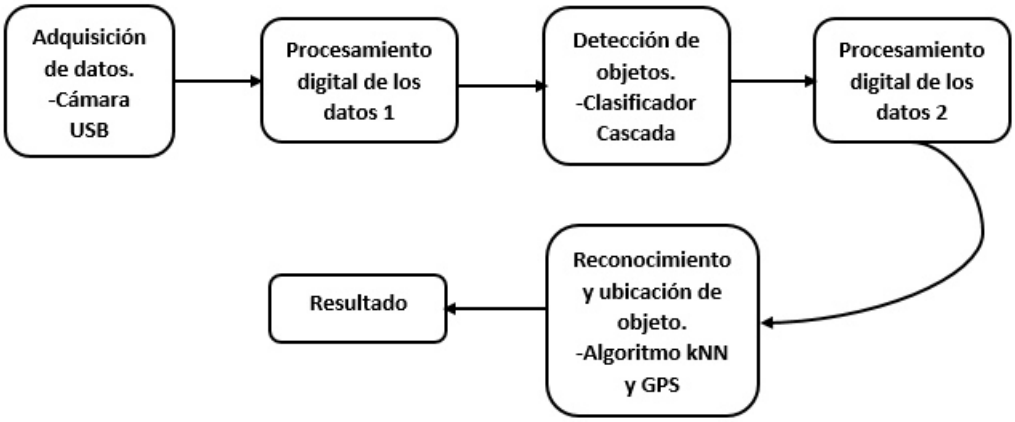
[12]Zhang, Z. (2016). Introduction to machine learning: k-nearest neighbors. Annals of Translational Medicine, 4(11), 218. http://doi.org/10.21037/atm.2016.03.37

[13]K, Alexander Mordvintsev y Abid. https://readthedocs.org/. [En línea] 02 de Septiembre de 2017. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf.

4. DISEÑO DE LA APLICACIÓN

El siguiente esquema, muestra a nivel general la estructura en la que se basa la aplicación de detección y reconocimiento de señales de transito reglamentarias. En la figura 8 Esquema general de la aplicación se puede ver el diagrama de bloques. *Todos los códigos de programación desarrollados para la elaboración del programa se adjuntan en la sección de anexos.*

Figura 8: Esquema general de la aplicación



Fuente: Autor

Para profundizar en cada una de las etapas, en primera instancia se especifican las herramientas usadas para el desarrollo de las mismas.

4.1. HERRAMIENTAS PARA EL DESARROLLO DE LA APLICACIÓN

4.1.1. Cámara USB: Adquisición de datos

La figura 9 Cámara Logitech C615 HD Webcam muestra la cámara seleccionada para la adquisición de datos como fotos y vídeo.

Figura 9: Cámara Logitech C615 HD Webcam.

Fuente: Fabricante(2013)[En Linea]
https://secure.logitech.com/assets/36589/hd-webcam-c615-gallery-5.png

Las especificaciones de la cámara de acuerdo a la página web del fabricante se muestra en la tabla 1 Características cámara C615

Tabla 1: Características Cámara C615

Características	
Tipo de conección	USB
Tipo USB	USB 2.0
USB VID_PID	082C
Soporte UVC	Si
Microfono	Si
Tipo de Microfono	Mono
Lente y Tipo de Sensor	Glass
Tipo de Foco	Auto
Resolución Óptica	True = 2MP, Interpolated = 8MP
Diagonal Campo de visión (FOV)	74°
Captura de imagen (4:3 SD)	640 x 480, 1.2 MP, 2.0 MP, 8 MP
Captura de imagen (16:9 W)	360P, 480P, 720P, 1080P
Captura de Video (4:3 SD)	320 x 240, 640 x 480
Captura de Video (16:9 W)	360P, 480P, 720P, 1080P
Tasa de Frames (max)	30 FPS
Right Light	Si
Indicator Lights (LED)	Si
Privacy Shade	Si
Tripod Mounting Option	Si
Universal Clip Adjustability (range)	1 200 degrees

Fuente: Fabricante(2013). https://www.logitech.com. [En línea] [Citado el: 12 de 06 de 2017.]
http://support.logitech.com/en_us/product/hd-webcam-c615/specs.

Los principales motivos para escoger esta cámara son, su tecnología Right Light, el cual permite adaptare a los diferentes cambios de luminiscencia en el ambiente haciendo que las fotos y vídeos adquieran mayor nitidez, su framerate de 30 fotogramas por segundo permite visualizar de manera fluida los vídeos. La cámara también permite elegir entre múltiples resoluciones dando la ventaja de probar cuál de ellas brinda mejores resultados.

4.1.2. OpenCV: Procesamiento digital de datos

Para este apartado se usa la librería OpenCV (Open Source Computer Vision Library). El lenguaje de programación seleccionado para manipularla y sacar provecho de su prestaciones es Python 3.5. Cabe resaltar el uso de la librería numpy, esta permite trabajar con arrays, por tal motivo es un gran complemento para OpenCV dado que la capacidad para manipular y tratar imágenes se incrementa de manera considerable.

4.1.3. Haar Cascade: Detección de objetos

Basados en el material de apoyo de OpenCV [14], la librería brinda varias aplicaciones para poder entrenar un clasificador cascada propio y adaptarlo a una necesidad especifica. En este caso se enfatiza en dos. La primera de ellas es **opencv_createsamples**, se usa para preparar todos los datos de muestras positivas. La salida de este archivo es de formato .vec, el cual es un archivo binario que contiene imágenes, además de ser compatible con la segunda aplicación, **opencv_haartraining**. Escrita en c++, permite entrenar una cascada de clasificadores, para ello se debe disponer del archivo .vec donde están preparadas la muestras positivas, se necesita preparar la muestras negativas, definir cuántos clasificadores se desea tener en la cascada, definir si solo se trabajan con los filtros de haar básicos o con los rotados 45 grados, entre otros parámetros definidos más adelante.

4.1.4. Reconocimiento: Algoritmo kNN

En esta sección el reconocimiento consiste en diferenciar las señales de tránsito reglamentarias de velocidad de las que no lo son. El algoritmo kNN permite reconocer los números de las señales de tránsito, para ello se debe igualmente preparar los datos con los que se definirá las clases (números de 0 a 9) y así etiquetar las mismas. La aplicación para implementar este algoritmo es brindada por OpenCV, siendo el método KNearest, más adelante se profundiza en su uso.

4.1.5. Ubicación: GPS

La figura 10 BU-353 USB GPS Navigation Receiver muesta el GPS seleccionado para obtener la ubicación de las señales de transito reglamentarias.

[14]Blueprints, OpenCV 3. http://docs.opencv.org. [Online] [Consultado: 6 20, 2017.] http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html.

Figura 10: BU-353 USB GPS Navigation Receiver

Fuente: Fabricante(2014)[En Linea]
http://usglobalsat.com/images/supportimages/BU353_FRONT_MED.jpg

Las características del GPS según la página del fabricante se muestra en la tabla 2 Características BU-353 USB GPS Navigation Receiver

Tabla 2: Características BU-353 USB GPS Navigation Receiver

Características	
Chipset	SiRF Star III
Channels	20
Protocol	NMEA
Default Baud Rate	4800
SBAS	WAAS/EGNOS
Frequency	1575.42 MHz
C/A Code	1.023 MHZ Chip Rate
Antenna Type	Built-In Patch
RoHS Compliant	Yes
Sensitivity - Tracking	-159dBm
Hot Start	1 sec, average
Warm Start	38 sec, average
Cold Start	42 sec, average
Reacquisistion	0.1 sec, average
Operation Temperature (C)	-40C - 85C
Operation Humidity	Up To 95 % Non-Condnesing
Dimensions (inches)	2.08 X 0.75
USB Cable Length (inches)	58
Weight (oz)	2.2

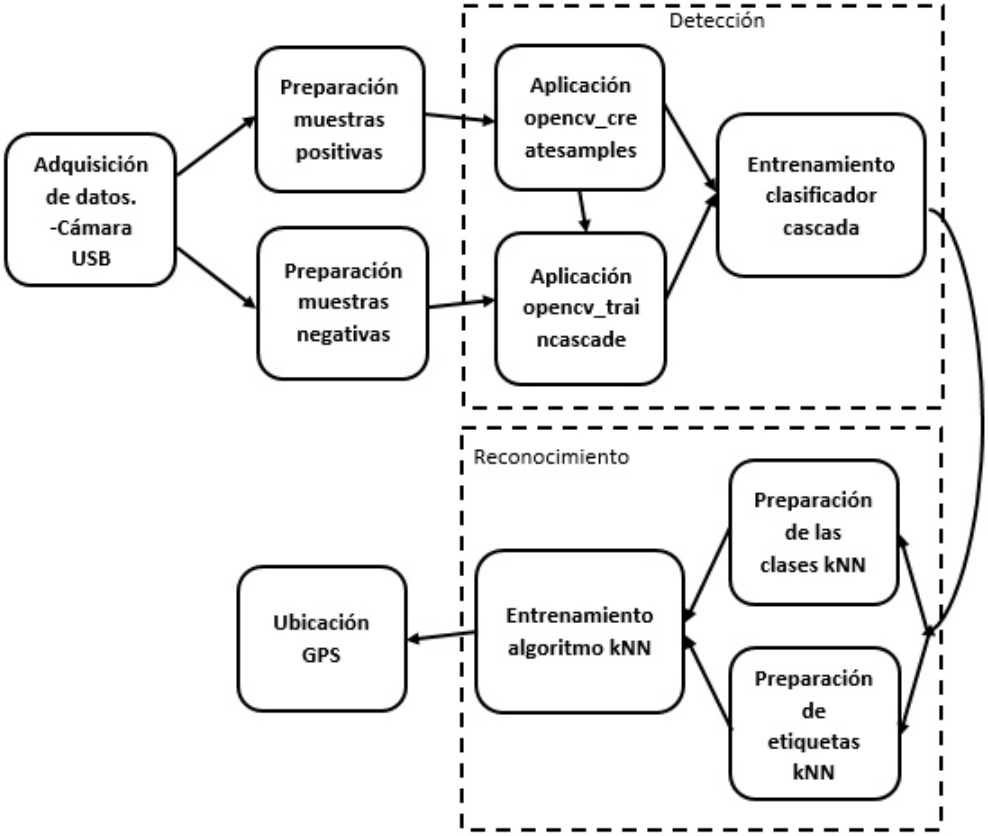
Fuente: Fabricante(2014). http://usglobalsat.com/. [En línea] [Citado el: 12 de 06 de 2017.]
http://usglobalsat.com/store/usb_comparison/usb_comp.html._MED.jpg

Los principales motivos para elegir este modelo de GPS es su conexión mediante puerto USB, esto presenta una gran ventaja debido a que la transmisión de datos y alimentación ocurren en el mismo medio, no necesita fuentes de alimentación externas. Su protocolo NMEA lo hace compatible con diferentes aplicaciones que necesiten hacer uso de sus datos de salida como latitud, longitud, velocidad. Finalmente su alta sensibilidad y sus tiempos de actualización, permiten usarlo para aplicaciones de tiempo real.

4.2. DESARROLLO DE LA APLICACIÓN

En la figura 11 Esquema del desarrollo de la aplicación, se muestra el procedimiento que se llevo a cabo para elaborar la aplicación de detección y reconocimiento de señales de tránsito reglamentarias.

Figura 11: Esquema del desarrollo de la aplicación



Fuente: Autor

4.2.1. Adquisición y preparación de los datos

La adquisición de datos inicial, consiste en una serie de fotos y vídeos tomados en diferentes calles y carreteras de señales de tránsito reglamentarias y todo lo que le rodea,

como pueden ser vehículos, peatones, edificios, árboles, postes eléctricos, etc. Este material, se usa para la preparación de muestras positivas y negativas y así entrenar el clasificador cascada.

Las señales de tránsito reglamentarias que se pretende detectar se muestran en la siguiente lista:

- **Señales de prohibición:** Señales representadas con un circulo blanco con borde rojo cruzado por una diagonal también roja, descendente desde la izquierda la cual forma un ángulo de 45° con la horizontal[15].
- **Señales de restricción:** En general, están compuestas por un círculo de fondo blanco y borde rojo en el que se inscribe el símbolo que representa la restricción, por ejemplo: velocidad máxima permitida, velocidad mínima permitida, peso máximo bruto permitido, ancho máximo permitido, circulación en ambos sentidos, etc[5].
- **Señales de obligación:** Compuestas generalmente por un círculo de fondo blanco y borde rojo en el que se inscribe el símbolo que representa la obligación, por ejemplo: Giro a la izquierda solamente, giro a la derecha solamente, giro en "U" solamente, vehículos pesados a la derecha, peatones a la izquierda, etc[5].
- **Señales de autorización:** Se caracterizan por el color rojo del círculo en el que se inscribe el símbolo o leyenda, autorizando algunas acciones a determinados vehículos, lo que constituye una excepción dentro de las señales reglamentarias, por ejemplo: zona de estacionamiento de taxis, zona exclusiva de paradero, zona de cargue y descargue[5].

4.2.1.1. Preparación de muestras positivas

La preparación de muestras positivas, consiste en primera instancia en obtener imágenes de señales de transito reglamentarias en primer plano, en condiciones de luminosidad optimas y no optimas, ángulos de visión sutilmente diferentes e incluso tamaños diferentes. La figura 12 Muestras positivas sin tratar muestra ejemplos.

Figura 12: Muestras positivas sin tratar

(a) Señal 1 (b) Señal 2 (c) Señal 3 (d) Señal 4 (e) Señal 5

[15]Transporte, Ministerio. 2015. Manual de Señalización Vial. Bogotá : MinTrans.

Fuente: Autor

Para poder usar este conjunto de muestras en el entrenamiento del clasificador, es necesario normalizarlas. Este procedimiento consiste en dejar a todas las imágenes del mismo tamaño o resolución (en este caso $50 * 50$). También se mejora su calidad, quitando todo el ruido posible de las muestras, finalmente se resaltan sus principales características para mitigar los variados efectos que provoca los cambios de iluminación.

El procedimiento se lleva a cabo dejando todas las muestras en una misma carpeta, posteriormente se aplican los siguientes métodos de opencv que se muestran en la tabla 3 Métodos OpenCV: Preparación de muestras.

Tabla 3: Métodos OpenCV: Preparación de muestras

Métodos OpenCV	
cv2.imread	Permite leer las imágenes de muestra
cv2.resize	Permite cambiar el tamaño de las muestras
cv2.GaussianBlur	Filtro para reducir ruido gaussiano
cv2.adaptiveThreshold	Permite resaltar Características
cv2.imwrite	Guarda el resultado en una imagen nueva

Fuente: Autor

Resultados de diferentes muestras se presentan en la figura 13 Muestras positivas tratadas.

Figura 13: Muestras positivas tratadas

(a) Muestra 1 (b) Muestra 2 (c) Muestra 3 (d) Muestra 4 (e) Muestra 5

Fuente: Autor

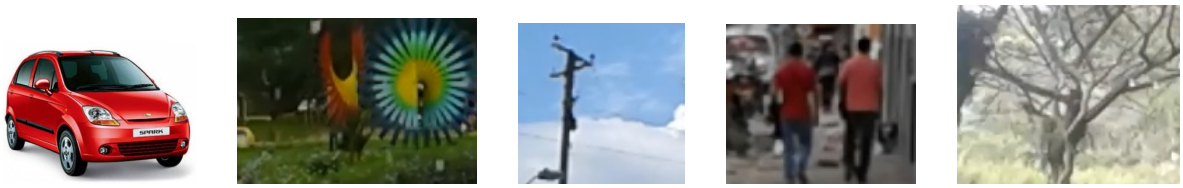
Para crear el archivo .txt, el cual se usa en el entrenamiento del clasificador, se usa el método **open** y el método **write** para escribir en él. Este archivo contiene:

- Ubicación de cada imagen de muestra en el computador.
- Cantidad de veces que aparece el objeto en la imagen.
- Coordenadas del objeto en la imagen.

4.2.1.2. Preparación de muestras negativas

El procedimiento para la preparación de muestras negativas es similar al de las positivas. Los métodos usados son los mismos e igualmente se debe crear un archivo .txt. Las diferencias radican en el tamaño al que se normalizan es de $100 * 100$ y por último el archivo .txt solo contiene la ruta de ubicación de cada imagen en el ordenador. En la figura 14 Muestras negativas sin tratar y figura 15 Muestras negativas tratadas se muestra imágenes del proceso.

Figura 14: Muestras negativas sin tratar



(a) Muestra 1 (b) Muestra 2 (c) Muestra 3 (d) Muestra 4 (e) Muestra 5

Fuente: Autor

Figura 15: Muestras negativas tratadas

(a) Muestra 1 (b) Muestra 2 (c) Muestra 3 (d) Muestra 4 (e) Muestra 5

Fuente: Autor

4.2.2. Entrenando el clasificador cascada

En este apartado se entreno un total de cuatro clasificadores cascada. El primero contó con 200 muestras positivas y 600 negativas, el segundo 200 positivas y 1000 negativas, el tercero 270 positivas y 2000 negativas y el cuarto 400 positivas y 4000 negativas. El procedimiento para entrenar cada uno de ellos es el mismo, se debe preparar el archivo .vec con opencv_createsamples.exe.

4.2.2.1. Aplicación opencv_createsamples

La aplicación opencv_createsamples sirve para preparar las muestras positivas. De acuerdo al material de apoyo de opencv[16] el resultado de esta aplicación indica al clasificador lo que debe buscar cuando trata de encontrar el objeto de interés.

Existe dos modos en que esta aplicación genera un conjunto de datos de muestras positivo. El primero, consiste en generar varias muestras positivas a partir de una sola imagen donde este el objeto de interés, el segundo consiste en obtener una colección de imágenes donde este el objeto de interés y organizarlas en un archivo .txt.

En este caso se enfatiza en el segundo modo dado a que es mucho más robusto y por tanto confiable para detectar objetos que no son fijos. Para usar la aplicación de esta manera se hace necesario usar los comandos mostrados en la tabla 4 Comandos y argumentos opencv_createsamples como linea de argumentos.

Tabla 4: Comandos y argumentos opencv_createsamples

opencv_createsamples	
Comando	**Argumento**
-info	Archivo .txt, colección de imágenes positivas
-num	Número de muestras
-w	Ancho de las muestras en pixeles
-h	Alto de las muestras en pixeles
-vec	Genera archivo .vec. Se define el nombre

Fuente: Blueprints, OpenCV 3. http://docs.opencv.org. [Online] [Consultado: 6 20, 2017.] http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html.

Un ejemplo de la linea de comando para prepara un conjunto de muestras positivas de 200 es: opencv_createsamples.exe -info positivos.txt -num 200 -w 50 -h 50 -vec positivos_200.vec

4.2.2.2. Aplicación opencv_traincascade

La aplicación opencv_traincascade es quien automatiza el proceso de aprendizaje del clasificador cascada. Los elementos que usa para cumplir este objetivo, son los expuesto en el marco teórico como son, filtros de haar, imagen integral, adaboost y cascada de clasificadores.

Igualmente como la aplicación opencv_createsamples, para el correcto funcionamiento de opencv_traincascade ésta cuenta con una serie de comandos como linea de argumentos. Los comandos mostrados en la tabla 5 Comandos y argumentos opencv_traincascade hace referencia a los del tipo haar.

[16]Blueprints, OpenCV 3. http://docs.opencv.org. [Online] [Consultado: 6 20, 2017.] http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html.

Tabla 5: Comandos y argumentos opencv_traincascade

opencv_traincascade	
Comando	**Argumento**
Parámetros Generales	
-data	Ubicación donde se almacena el resultado .xml
-vec	Archivo .vec generado con opencv_createsamples.exe
-bg	Archivo .txt, colección muestras negativas
-numPos	Número de muestras positivas usadas en cada etapa
-numNeg	Número de muestras negativas usadas en cada etapa
-numStages	Número de etapas de la cascada para ser entrenadas
Parámetros de la Cascada	
-w	Ancho de la muestra en pixeles
-h	Alto de la muestra en pixeles

Fuente: Blueprints, OpenCV 3. http://docs.opencv.org. [Online] [Consultado: 6 20, 2017.] http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html.

Un ejemplo de la linea de comando para prepara un clasificador de 200 muestras positivas y 600 negativas es:
opencv_traincascade.exe -data resultado -vec positivos_200.vec -bg negativos.txt -numPos 200 -numNeg 600 -numStages 10 -w 50 -h 50

Cabe resalta que esta aplicación tiene más comandos y argumentos, sin embargo no se tratan en este caso porque hacen parte de otro modo de entrenamiento o sencillamente vienen predefinidos con valores convenientes. Este es el caso de comando -mode que por defecto trabaja con los filtros de haar básicos para obtener las caracteristicas de haar[17].

4.2.3. Entrenando algoritmo kNN

Este algoritmo se entrena con el objetivo de reconocer los números del 0 al 9. Para ello se realiza el siguiente procedimiento.

4.2.3.1. Preparación de clases

En este caso las clases son cada uno de los números correspondientes de 0 a 9. El proceso de preparación consiste en primera medida en obtener una imagen con todos los dígitos repetidos varias veces y cada paquete de dígitos tenga fuentes diferentes para ampliar las probabilidades de un correcto reconocimiento de cada número.

Se realizó tres imágenes donde se cambio la cantidad de dígitos en cada una de ella, esto con el fin de elegir entre las tres, el mejor resultado.

[17]Blueprints, OpenCV 3. http://docs.opencv.org. [Online] [Consultado: 6 20, 2017.] http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html.

El paso siguiente consiste en tomar cada uno de los números de la imagen (el proceso es igual para cada una de las imágenes), normalizarlos a un tamaño de $10 * 10$, esto con el fin de crear un vector de características (cada píxel es una característica)de 100 píxeles y de esta manera crear cada clase. Los métodos usados para cumplir este objetivo son de la librería opencv y numpy. En la tabla 6 Métodos: Preparación de clases se listan cada uno de ellos.

Tabla 6: Métodos: Preparación de clases

Métodos para creación de clases	
Métodos OpenCV	
cv2.imread	Permite leer las imágenes de muestra
cv2.adaptiveThreshold	Permite resaltar Características
cv2.findContours	Permite detectar los bordes de cada objeto
cv2.boundingRect	Permite trazar un rectángulo alrededor del objeto
cv2.imresize	Permite cambiar el tamaño de una imagen
Métodos Numpy	
np.empty	Permite crear una matriz vacía
np.reshape	Permite cambiar las dimensiones de una matriz
np.append	Permite añadir valores a una matriz
np.savetxt	Permite guardar resultados en archivo .txt

Fuente: Autor

Cada vector generado de cada número que conforma las clases es almacenado es un archivo .txt. Este archivo se usa posteriormente para indicarle al algoritmo kNN las posibles categorías donde puede clasificar el objeto reconocido.

4.2.3.2. Preparación de etiquetas

Las etiquetas se asocian a cada elemento de la clase para definir la respuesta cuando el algoritmo kNN designe a qué clase pertenece el objeto reconocido, es decir, si hay 50 elementos por clase, se necesitan 50 etiquetas por clase y de esta manera asociar una respuesta a cada uno ellos. Los métodos para crear las etiquetas son de la librería numpy y se muestran en la tabla 7 Métodos: Preparación de etiquetas.

Tabla 7: Métodos: Preparación de etiquetas

Métodos para creación de clases	
Métodos Numpy	
np.arange	Permite crear un vector con n cantidad de elementos
np.repeat	Permite repetir cada elemento n cantidad de veces
np.savetxt	Permite guardar resultados en archivo .txt

Fuente: Autor

Al igual que con las clases, las etiquetas se guardan en un archivo .txt para posteriormente indicarle al algoritmo kNN la respuesta que debe asociar a la clase al cual halla clasificado el nuevo elemento.

4.2.3.3. OpenCV y kNN

Una vez se tiene las clases y etiquetas en sus respectivos archivos .txt, solo resta carga cada uno de estos archivos y usar los métodos que opencv dispone para hacer uso del algoritmo kNN. En la tabla 8 Métodos: Métodos para usar kNN se muestra los comandos usados.

Tabla 8: Métodos: Métodos para usar kNN

Métodos uso kNN	
np.loadtxt	Permite cargar archivos .txt
cv2.ml.KNearest_create()	Realiza el llamado al algoritmo kNN
cv2.ml.ROW_SAMPLE	Indica clases y etiquetas
findNearest	Se define la imagen con los elementos a reconocer y k

Fuente: Autor

4.2.4. Ubicación GPS

El proceso para obtener los datos de latitud y longitud arrojados por el GPS, se sintetizan en los siguientes pasos.

- Instalación del driver
- Verificación del puerto serial (puerto COM)
- Definir la tasa de baudios (Estandar 4800)
- Mediante el lenguaje de programación Python, se importa la librería serial y de esta manera se lee los datos necesarios que entrega el GPS (Código en la sección de anexos).

5. PRUEBAS, MEJORAS Y RESULTADOS

Las pruebas realizadas están divididas en dos etapas. La primera etapa corresponde a la detección de las señales de transito reglamentarias y la segunda al reconocimiento de las mismas. (El código final para cada etapa se adjunta en la sección de anexos)

5.1. DETECCIÓN DE SEÑALES DE TRANSITO REGLAMENTARIAS

En el proceso de detección de objetos no existe método fijo que indique los parámetros adecuados para satisfacer una necesidad puntual. Las herramientas que ayudan a esta tarea se deben calibrar de tal forma que los resultados obtenidos sean satisfactorios.

Para probar los clasificadores cascada y elegir el de mejor desempeño, se dispone de una serie de vídeos donde se muestra parte del trayecto recorrido en la vía Pitalito-Timaná en el departamento del Huila con condiciones de iluminación variadas y múltiples objetos en escena.

5.1.1. Procesamiento digital de imágenes 1

Este es el paso previo a usar el clasificador cascada, a continuación se muestran los métodos de opencv usados para el tratamiento de los vídeos. Cabe resaltar que para cada clasificador los parámetros de los métodos cambian para obtener mejores resultados que estos pueden entregar, esto se abarca más en detalle cuando se hable de cada clasificador. En la tabla 9 se muestra el tratamiento inicial del vídeo.

Tabla 9: Métodos OpenCV: Tratamiento inicial de vídeo

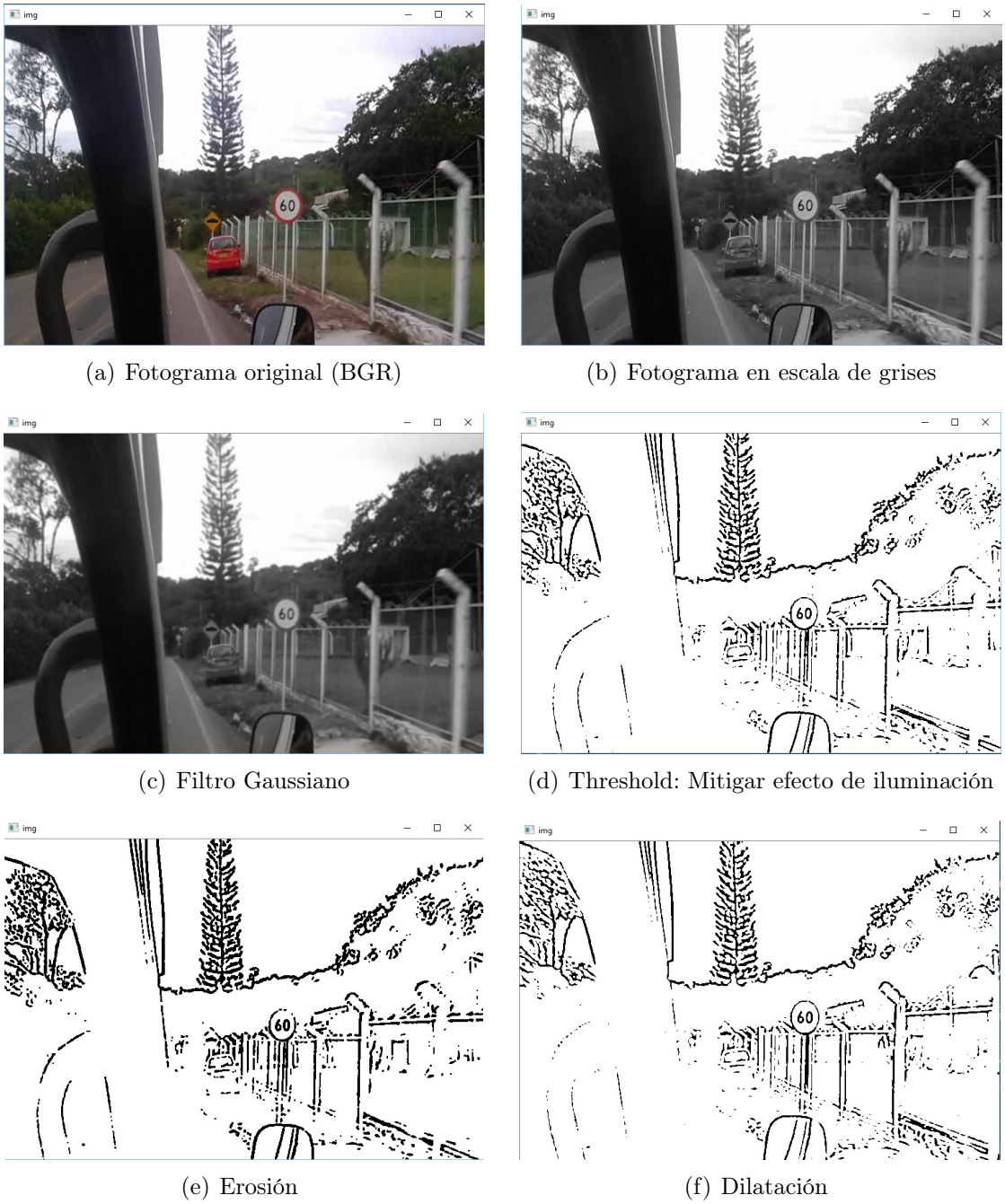
Métodos OpenCV	
cv2.VideoCapture	Permite leer vídeos en formato .avi
cv2.cvtColor	Permite cambiar de un espacio de color a otro
cv2.GaussianBlur	Filtro para reducir ruido gaussiano
cv2.adaptiveThreshold	Resalta Características
cv2.erode	Contrae la imagen
cv2.dilate	Expande la imagen

Fuente: Autor

La resolución original de los vídeos es de $1280 * 720$, sin embargo, esta resolución es demasiado grande para procesarla a 30 frames por segundo, el equipo donde se realizo este trabajo no cuenta con tanta capacidad de procesamiento. Para resolver este inconveniente la resolución de los vídeos se bajo a $720 * 480$.

El proceso del tratamiento a los vídeos se muestra en la figura 16 Procesamiento digital vídeo.

Figura 16: Procesamiento digital vídeo



(a) Fotograma original (BGR)

(b) Fotograma en escala de grises

(c) Filtro Gaussiano

(d) Threshold: Mitigar efecto de iluminación

(e) Erosión

(f) Dilatación

Fuente: Autor

El espacio de color original es RGB[18], sin embargo al momento de leerlo en opencv este cambia automáticamente a BGR debido a que la librería trabaja de manera preestablecida en este espacio de color. Como el clasificador cascada no se basa en el color para detectar objetos, los pasos siguientes se centran en eliminar todo lo posible el ruido de los fotogramas y los efectos de la iluminación sobre los objetos.

Para la supresión de ruido el filtro gaussiano y el efecto de aplicar una erosión seguida de una dilatación (openning[19]) son los encargados de esta tarea. Mitigar los efectos de la iluminación se hace mediante el método threshold adaptativo , este permite binarizar la imagen y adaptarse a los cambios de intensidad para resaltar las formas de los objetos[17].

5.1.2. Clasificador cascada 1

En la tabla 10 Parámetros clasificador 1, se muestra la configuración del clasificador cascada 1.

Tabla 10: Parámetros clasificador 1

Clasificador cascada 1	
Muestras de entrenamiento positivas	200
Muestras de entrenamiento negativas	600
Clasificadores	6
Tiempo de entrenamiento	5 horas

Fuente: Autor

El vídeo que sirvió de test para este clasificador cuenta con las siguientes características.

- Duración de 1 minuto
- 7 señales de transito reglamentarias (6 de velocidad, 1 de prohibido adelantar)
- Zonas de alta vegetación
- Múltiples vehículos
- Diferentes estructuras (casas, mallas, rejas, etc)
- Otros tipo de señales de transito

[18] Alegre, E, Pajares, G y Escalera, A. (2016). Conceptos y Métodos en Visión por Computador. Grupo de Visión del Comité Español de Automáticas (CEA).

[19] K, Alexander Mordvintsev y Abid. https://readthedocs.org/. [En línea] 02 de Septiembre de 2017. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf.

Los parámetros de procesamiento de vídeo con los que se obtuvo los mejores resultados se muestras en la tabla 11 Tratamiento inicial de vídeo: Parámetros.

Tabla 11: Tratamiento inicial de vídeo: Parámetros

Métodos OpenCV	
cv2.cvtColor	Cambio de BGR a Gris
cv2.GaussianBlur	Kernel de 5*5
cv2.adaptiveThreshold	Tamaño de la vecindad para umbral 11, constante 5
cv2.erode	Kernel 2*2
cv2.dilate	Kernel 2*2

Fuente: Autor

Para usar el archivo .xml que resulta del entrenamiento del clasificador, opencv dispone del método **cv2.CascadeClassifier** cuyo argumento es la ruta donde se encuentra almacenado el .xml. Para detectar los objetos **detectMultiScale** cuyos parámetros son, el vídeo ya tratado, el siguiente especifica cuánto se reduce el tamaño de la imagen en cada escala de imagen. (**1.3 para este caso**) y por último se especifica cuántos vecinos debe tener cada rectángulo candidato para retenerlo (**14 para este caso**).

La razón de que sea un rectángulo es porque la salida de *detectMultiScale* son las coordenadas x,y y w,h, mediante estos datos se puede trazar un rectángulo en el objeto detectado por el clasificador.

Los resultados se muestran en la tabla 12 Resultados: Clasificador 1.

Tabla 12: Resultados: Clasificador 1

Clasificador 1	
Positivos detectados	7
Falsos positivos detectados	20

Fuente: Autor

El clasificador fue capaz de detectar satisfactoriamente todas las señales de transito reglamentarias de la vía, sin embargo, la cantidad de falsos positivos es muy elevada.

Es importante tener en cuenta que estos resultados solo pretenden mostrar la eficacia del clasificador en un entorno cambiante. De momento no se pretende reducir la taza de falsos positivos.

5.1.3. Clasificador cascada 2

En la tabla 13 Parámetros clasificador 2 se muestra su configuración.

Tabla 13: Parámetros clasificador 2

Clasificador cascada 2	
Muestras de entrenamiento positivas	200
Muestras de entrenamiento negativas	1000
Clasificadores	7
Tiempo de entrenamiento	9 horas

Fuente: Autor

El vídeo de test para este clasificador es el mismo que el del anterior.
Los parámetros de procesamiento de vídeo con los que se obtuvo los mejores resultados se muestra en la tabla 14 Tratamiento inicial de vídeo: Parámetros 2.

Tabla 14: Tratamiento inicial de vídeo: Parámetros 2

Métodos OpenCV	
cv2.cvtColor	Cambio de BGR a Gris
cv2.GaussianBlur	Kernel de 5*5
cv2.adaptiveThreshold	Tamaño de la vecindad para umbral 11, constante 5
cv2.erode	Kernel 2*2
cv2.dilate	Kernel 2*2
detectMultiScale	Escala 1.3, Vecindad 15

Fuente: Autor

La tabla 15 Resultados: Clasificador 2 se pueden ver su desempeño.

Tabla 15: Resultados: Clasificador 2

Clasificador 2	
Positivos detectados	7
Falsos positivos detectados	4

Fuente: Autor

Para efectos de este vídeo el clasificador mejora de manera considerable comparado con el clasificador anterior. Por tanto este clasificador se someterá a otras pruebas y los resultados se mostrarán junto con los demás clasificadores.

5.1.4. Clasificador cascada 3

La tabla 16 Parámetros clasificador 3 muestra su configuración.

Tabla 16: Parámetros clasificador 3

Clasificador cascada 3	
Muestras de entrenamiento positivas	270
Muestras de entrenamiento negativas	2000
Clasificadores	8
Tiempo de entrenamiento	12 horas

Fuente: Autor

El vídeo de test para este clasificador es el mismo que el del anterior.
Los parámetros de procesamiento de vídeo con los que se obtuvo los mejores resultados se muestran en la tabla 17 Tratamiento inicial de vídeo: Parámetros 3.
Tabla 17: Tratamiento inicial de vídeo: Parámetros 3

Métodos OpenCV	
cv2.cvtColor	Cambio de BGR a Gris
cv2.GaussianBlur	Kernel de 5*5
cv2.adaptiveThreshold	Tamaño de la vecindad para umbral 11, constante 4
cv2.erode	Kernel 2*2
cv2.dilate	Kernel 2*2
detectMultiScale	Escala 1.3, Vecindad 15

Fuente: Autor

La tabla 18 Resultados: Clasificador 3 muestra su desempeño.

Tabla 18: Resultados: Clasificador 3

Clasificador 3	
Positivos detectados	7
Falsos positivos detectados	7

Fuente: Autor

Este clasificador también será puesto a más pruebas, más adelante se muestran los resultados.

5.1.5. Clasificador cascada 4

La tabla 19 Parámetros clasificador 4 muestra su configuración.
Tabla 19: Parámetros clasificador 4

Clasificador cascada 4	
Muestras de entrenamiento positivas	400
Muestras de entrenamiento negativas	4000
Clasificadores	8
Tiempo de entrenamiento	18 horas

Fuente: Autor

El vídeo de test para este clasificador es el mismo que el del anterior.
Los parámetros de procesamiento de vídeo con los que se obtuvo los mejores resultados se muestra en la tabla 20 Tratamiento inicial de vídeo: Parámetros 4.

Tabla 20: Tratamiento inicial de vídeo: Parámetros 4

Métodos OpenCV	
cv2.cvtColor	Cambio de BGR a Gris
cv2.GaussianBlur	Kernel de 5*5
cv2.adaptiveThreshold	Tamaño de la vecindad para umbral 11, constante 5
cv2.erode	Kernel 2*2
cv2.dilate	Kernel 2*2
detectMultiScale	Escala 1.3, Vecindad 10

Fuente: Autor

En la tabla 21 Resultados: Clasificador 4 se muestra su desempeño.

Tabla 21: Resultados: Clasificador 4

Clasificador 4	
Positivos detectados	7
Falsos positivos detectados	3

Fuente: Autor

5.1.6. Comparando los clasificadores

En la tabla 22 Comparación de clasificadores se presenta con más detalles los resultado del primer test realizado a los clasificadores.
Tabla 22: Comparación de clasificadores

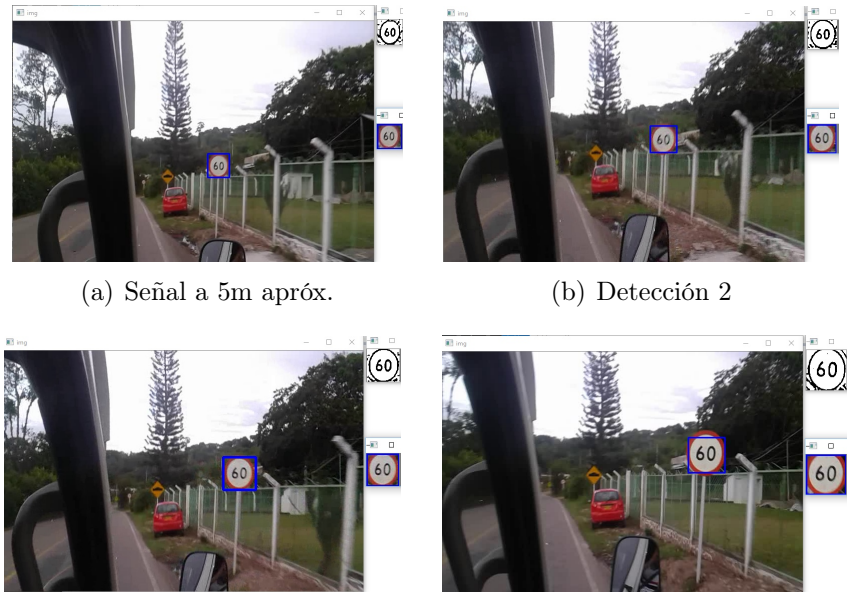
Clasificadores					
	Positivos	**Falsos positivos**	**Dist. máx. Detección**	**Dist min. Detección**	**Velocidad promedio**
Clasificador 1	7	20	5m	1.5m	50km/h
Clasificador 2	7	4	5m	1.5m	50km/h
Clasificador 3	7	7	5m	1.5m	50km/h
Clasificador 4	7	3	5m	1.5m	50km/h

Fuente: Autor

La distancia máxima y mínima de detección a la que se hace referencia en la tabla anterior, son las distancias aproximadas a la cual se empieza a detectar y se deja de detectar la señal de transito. Sin embargo, estas distancias pueden variar (siempre dentro del rango de 5m y 1.5m) debido a los cambios de brillo y contraste a las que están sometidas las señales.

Todos los clasificadores al momento de detectar una señal de tránsito reglamentaria, tuvieron el mismo comportamiento, por lo que el rango de detección es el mismo para todos.

En la figura 17 Detección de señal de tránsito iluminación media muestra un ejemplo de detección con iluminación diferente.
Figura 17: Detección de señal de tránsito iluminación media



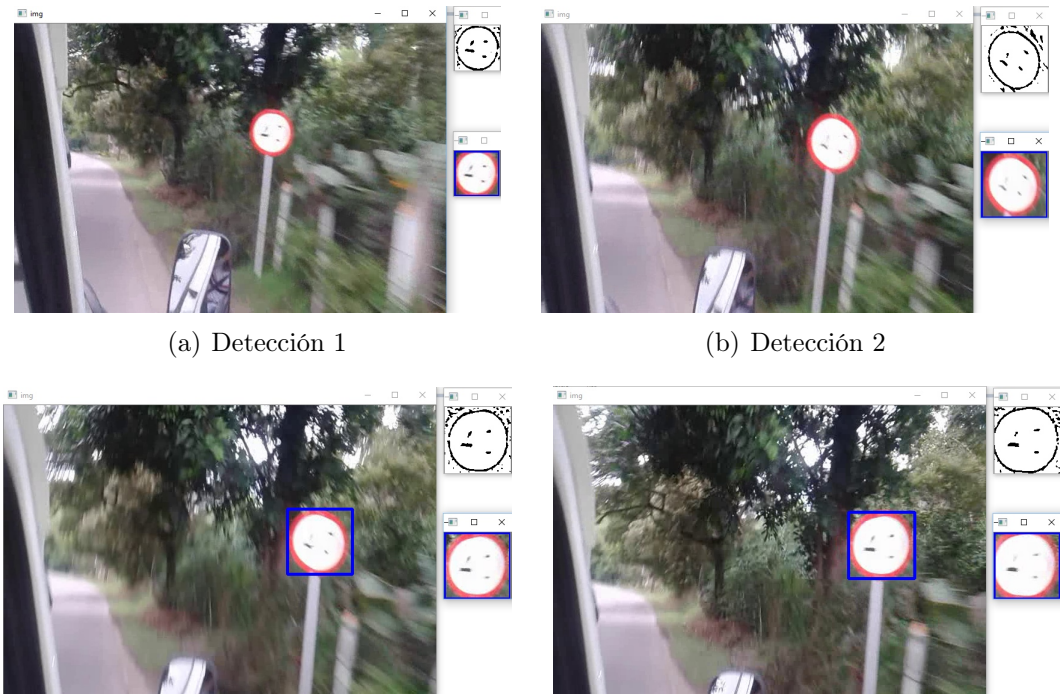
(a) Señal a 5m apróx. (b) Detección 2

(c) Detección 3 (d) Señal a 2m apróx.

Fuente: Autor

En la figura 17 se muestran cuatro fotogramas donde se puede observar el comportamiento de los clasificadores cuando detectan una señal de transito con iluminación media, por tanto su contraste y brillo no son elevados.

Figura 18: Detección de señal de tránsito iluminación alta



(a) Detección 1

(b) Detección 2

(c) Detección 3

(d) Detección 4.

Fuente: Autor

En la figura 18 Detección de señal de tránsito iluminación alta, se muestra condiciones en la que muchas señales de tránsito se detectan bajo estas condiciones, por lo que el número es completamente ininteligible. Sin embargo, en próximos procesos se hace todo lo posible para mejorar sus condiciones de lectura.

5.1.6.1. Elección del mejor clasificador

Para elección de clasificador con el que se trabajara finalmente, los clasificadores (Excepto el primer clasificador), se someterán a otra prueba el cual consiste en pasar un vídeo de larga duración con iluminación y escenario cambiante. El clasificador que obtenga la mayor cantidad de positivos detectados y la menor cantidad de falsos positivos será el elegido.

El perfil del vídeo que sirvió de test fue el siguiente.

- Ruta: San Agustin - Pitalito.
- Duración 5 minutos.
- 12 señales de transito (9 de velocidad, 3 de prohibido adelantar).
- Zonas de alta vegetación.
- Zonas de baja y alta iluminación .
- Múltiples vehículos,estructuras y otro tipo de señales.
- Cámara fija enfocada al carril por el que se transita.

El resultado de este test es mostrado en la tabla 23.

Tabla 23: Resultados: Prueba de elección

Clasificadores		
	Positivos	**Falsos positivos**
Clasificador 2	12	43
Clasificador 3	11	30
Clasificador 4	12	24

Fuente: Autor

De acuerdo al resultado el clasificador adecuado para la detección de señales es el número cuatro. Es importante tener en cuenta que en 5 minutos se procesaron un total de 540.000 imágenes, en las cuales solo 12 objetos son válidos y la cantidad de objetos no válidos es demasiado grande. Obtener un acierto del 100 % en los objetos válidos y solo detectar 24 objetos no válidos como válidos es un resultado bastante bueno. El paso siguiente es disminuir mediante un segundo tratamiento digital la detección de estos falsos positivos.

5.1.7. Mejorando la detección de señales

Mejorar la detección de las señales de transito reglamentarias es un proceso de ensayo y error, no existe un método el cual paute de manera rigurosa los pasos a seguir para obtener los resultados deseados, si no, que es necesario realizar poco a poco ajustes hasta alcanzar el objetivo.

Los ajustes que se realicen deben ser equilibrados, es decir, que disminuyan todo lo posible la detección de falsos positivos afectando lo mínimo posible la detección de objetos correctos.

5.1.7.1. Delimitación de detección por tamaño

Las mejoras se llevan a cabo, estableciendo las características de los falsos positivos que se puedan tratar. En este caso el **tamaño de la matriz** que contiene los **falsos positivos** es muy buen elemento para delimitar la detección a un rango de tamaño más especifico. Estas matrices están en un rango aproximado de:

- **142*465 hasta 620*250**

Las matrices que contienen a las **señales de transito** reglamentarias están en un rango aproximado de **346*304 a 632*211**.

Después de varios intentos y probando diferentes vídeos el rango del tamaño de las matrices para **delimitar la detección** que mejor dio resultado disminuyendo los falsos positivos sin afectar demasiado la detección de positivos son, **matrices cuyas filas estén entre 430 y 520**.

Al tomar los dos vídeos de referencia anteriores en la tabla 24 Resultados: Rango entre 430 y 520, se muestra su desempeño es.

Tabla 24: Resultados: Rango entre 430 y 520

Resultados		
	Positivos	**Falsos positivos**
Test vídeo 1	5	2
Test vídeo 2	12	15

Fuente: Autor

Es importante tener en cuenta que el rango de detección establecido hace que las señales sean detectadas desde los 3m - 3.5m hasta los 2m - 2.5m aproximadamente. Este el motivo por el cual en ocaciones no se detecta la señal de tránsito. Como en el caso del primer vídeo, las dos últimas señales no se detectaron porque quedaron fuera del rango de detección.

En cuanto a los falsos positivos de ve una disminución de los mismo en un 40 % aproximadamente, sin embargo no es suficiente. Para reducir más estas detecciones se da lugar a la validación de las detecciones.

5.1.7.2. Validando detección

La mayoría de los falsos positivos detectados tienen como característica que son ramas de árboles, partes de vehículos, partes de estructuras, cables, nubes, entre otros.

La validación consiste en comprobar si cada objeto detectado por el clasificador dentro del rango establecido anteriormente tiene el color rojo característico de las señales de tránsito reglamentarias.

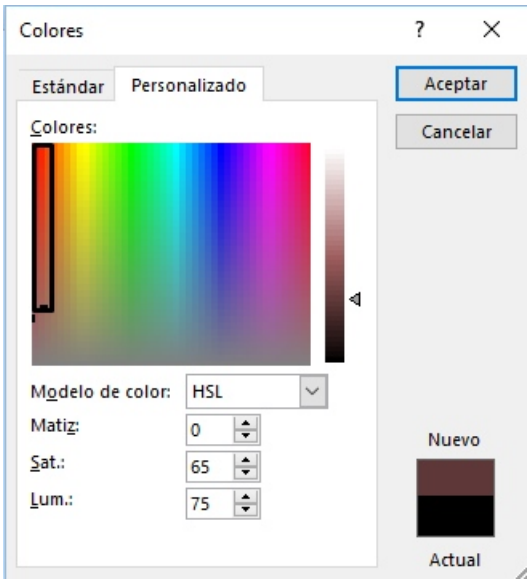
La detección del color rojo se realiza cambiando el espacio de color del objeto de **BGR a HSV**[20]. La elección de este espacio de color se debe a que se obtiene mejores a pesar de los cambios de iluminación que afectan al objeto.

El rango de color establecido es el siguiente.

- **Rojo_bajos** = ([0, 65, 75])
- **Rojo_altos** = ([12, 255, 255])

Opencv da valores de color a los píxeles de 0 a 255, para encontrar este rango se empleó una paleta de colores como la mostrada en la figura 19 Rango de color rojo en hsv.

Figura 19: Rango de color rojo en hsv



Fuente: Paleta de color word 2016

Tomando una vez más los vídeos de referencia los resultados de detección de positivos y falsos positivos se muestran en la tabla 25 Resultados: Rango y validación.

[20] Alegre, E, Pajares, G y Escalera, A. (2016). Conceptos y Métodos en Visión por Computador. Grupo de Visión del Comité Español de Automáticas (CEA).

Tabla 25: Resultados: Rango y validación

Resultados		
	Positivos	**Falsos positivos**
Test vídeo 1	5	0
Test vídeo 2	11	3

Fuente: Autor

Es un hecho que para reducir la tasa de detección de falsos positivos, lamentablemente se deja detectar el 100 % de las señales de tránsito, sin embargo, las condiciones de iluminación e incluso las distancia de detección son muy aleatorias en un ambiente tan cambiante como es una carretera.

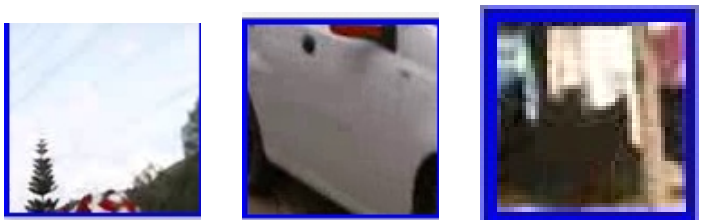
Lo que se ha intentado hacer durante todo este proceso es darle la capacidad a la detección de señales ser lo máximo posible inmune a esa aleatoriedad del ambiente. Esto es lo que se puede apreciar con los resultados de los vídeos de referencia, en donde en el primer vídeo no se detectan falsos positivos, sin embargo, no se logró detectar todas las señales de tránsito. En el caso del segundo vídeo se puede observar que a pesar de las condiciones establecidas para dar la detección como válidas hubo tres casos donde incluso tenia secciones de color rojo dentro del rango establecido para la validación.

Las causas para que no sea detectada una señal de tránsito pueden ser:

- Las condiciones de iluminación hace que el brillo y contraste de una tonalidad al color rojo que no esta dentro del rango permitido.
- La señal de tránsito aparece en escena a una distancia lejana (más de 5 metros) por tanto no su tamaño no entra dentro del rango permitido.

Estas causas a pesar de que también son aleatorias ocurren en menor medida por lo que permite que la tasa de éxito de detección este alrededor del 90 % cuando las condiciones de iluminación no son optimas.

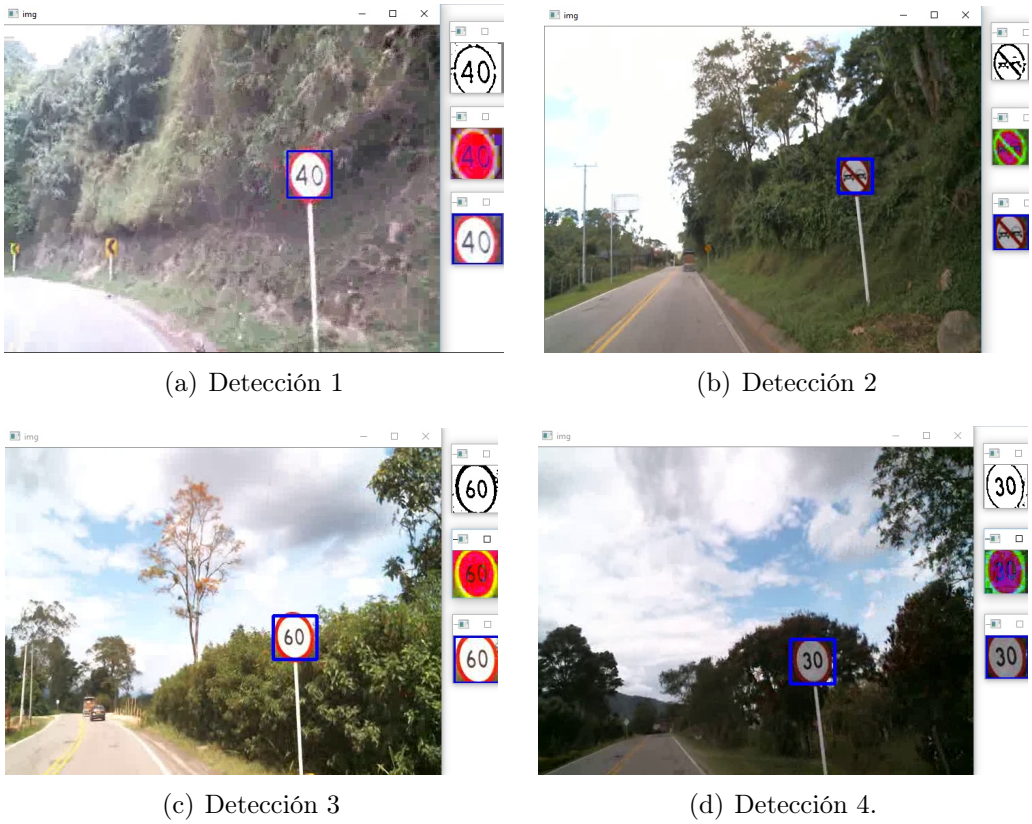
Figura 20: Falsos positivos vídeo 2



(a) Detección 1 (b) Detección 2 (c) Detección 3

Fuente: Autor

Figura 21: Detección en diferentes condiciones de iluminación



(a) Detección 1

(b) Detección 2

(c) Detección 3

(d) Detección 4.

Fuente: Autor

5.2. RECONOCIMIENTO DE SEÑALES DE TRANSITO REGLAMENTARIAS

Como se explico en la sección del entrenamiento del algoritmo kNN, el objetivo del reconocimiento de las señales de tránsito reglamentarias consiste en discernir entre las señales que son de velocidad y las que no lo son, además de identificar cada señal de velocidad.

5.2.1. Elección de la imagen de entrenamiento

Para la elección de la imagen que sirva como referencia para entrenar el algoritmo kNN se tomó en cuenta dos parámetros.

- Precisión
- Velocidad de reconocimiento en segundos

Una vez realizado el proceso de entrenamiento explicado en el capitulo anterior para cada imagen los resultados se muestran en la tabla 26 Resultados: Entrenamiento kNN.

Tabla 26: Resultados: Entrenamiento kNN

Resultados		
Dígitos	**Precisión**	**Velocidad de reconocimiento**
5000	92 %	3.02s
2000	92 %	1.05s
500	91 %	0.48s

Fuente: Autor

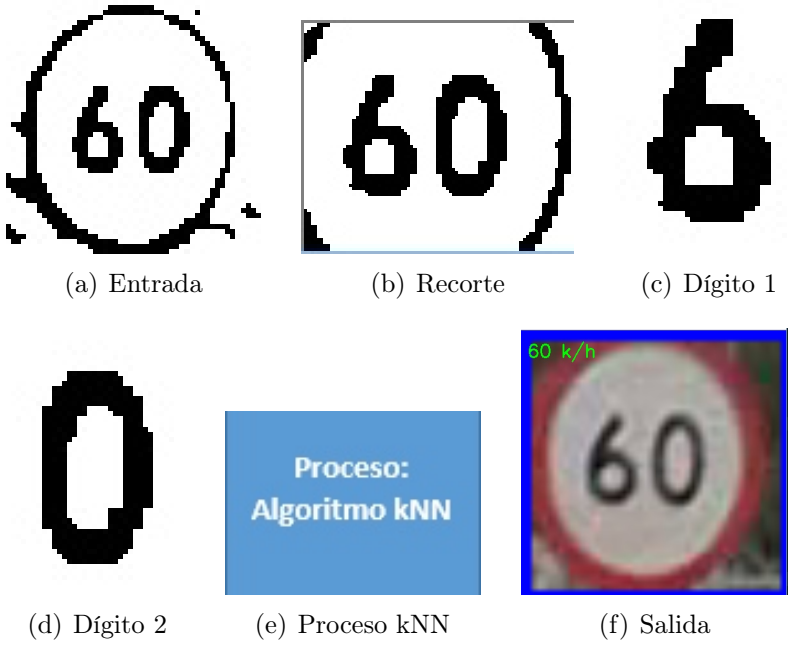
Después de hacer que reconociera varios números, se observa que el algoritmo entrenado con las dos primeras imágenes tiene las misma precisión, sin embargo, se toman mucho tiempo para lograr dar un resultado, especialmente el primero entrenado con una imagen de 5000 dígitos. Ahora el algoritmo entrenado con la imagen de 500 dígitos tiene un tiempo de respuesta aceptable a pesar de que la precisión disminuyo en un 1 % a efecto prácticos brinda el mismo nivel de confianza que el algoritmo entrenado con la primera o segunda imagen. Por tal motivo se elige el algoritmo entrenado con la imagen de 500 dígitos.

5.2.2. Integrando kNN a la detección de señales

Para integrar el algoritmo kNN a la detección de señales, en Python se crea una clase llamada reconocimiento, el cual se importa en el código de detección y de esta manera se hace un llamado a la función de permite reconocer la señal de transito.

La figura 22 Reconocimiento de señales muestra el proceso para reconocer señales de tránsito reglamentarias.

Figura 22: Reconocimiento de señales



(a) Entrada (b) Recorte (c) Dígito 1

(d) Dígito 2 (e) Proceso kNN (f) Salida

Fuente: Autor

El proceso que se muestra en el conjunto de imágenes anterior funciona de la siguiente manera.

- La entrada es el objeto detectado, gracias al clasificador cascada. Figura 5.7a.
- Empleando los métodos de erosión y dilatación se busca mejorar el aspecto del objeto detectado.
- Se enfoca el área del centro del objeto y se recorta lo que no es de interés. Figura 5.7b
- En caso de que en el área seleccionada existan dígitos (cada dígito tiene un rango de área característico, de esta manera se identifica si es carácter o no) se recorta cada uno. Figura 5.7 c y d
- *El proceso del algoritmo kNN* consiste en tomar cada carácter, normalizarlo a un tamaño de 10*10 y crear un vector de 1*100. De esta manera empleando el método findNearest se procede a comprobar los vecinos más cercanos y así reconocer el carácter.
- Finalmente el algoritmo arroja un resultado el cual se puede visualizar en el objeto detectado. Figura 5.7f
- En caso de que el objeto detectado **NO** sea una señal de velocidad. Las áreas en el centro de la imagen no encajan dentro del rango de áreas de un dígito, por consiguiente la señal será reconocida como una señal de transito que no es de velocidad.

5.2.3. Pruebas de reconocimiento

Teniendo en cuenta el nivel de detección que se ha logrado, se toma de nuevo los vídeos de referencia con los que se ha trabajado.

5.2.3.1. Reconocimiento vídeo 1

En este vídeo aparece un total de 7 señales de tránsito reglamentarias, de las cuales se detectan 5 y 0 falsos positivos. Al integrar la sección de reconocimiento la tabla 27
Resultados: vídeo 1, se muestra el desempeño.
Fuente: Autor

En este primer vídeo el reconocimiento a funcionado perfectamente puesto que fue capaz de dar el valor de cada señal de velocidad de manera correcta y la señal de prohibido adelantar la clasifico como "No es de velocidad". Es necesario tener en cuenta que el nivel de reconocimiento va ligado al nivel de detección. En la siguiente prueba de vídeo se ve claramente.

Tabla 27: Resultados: vídeo 1

Resultados vídeo 1						
	60km/h	**40km/h**	**30km/h**	**30km/h**	**Prohibido Adelantar**	**Acierto**
Test video 1	correcto	correcto	correcto	correcto	correcto	100 %

5.2.3.2. Reconocimiento video 2

Este vídeo es más exigente que el anterior debido a sus muchos cambios en la iluminación, tiempo de duración y objetos en escena.

En este recorrido existen 12 señales de tránsito en total de las cuales se detectan 11 y 3 falsos positivos. Las señales de transito detectadas en escena son las siguiente.

- 3 señales de 60km/h.
- 3 señales de 40km/h.
- 1 señal de 30km/h.
- 1 señal de 20km/h.
- 3 señales de prhobido adelantar.
- 3 falsos positivos.

Al integrar la sección de reconocimiento se obtuvo los siguientes resultados.

- De las 8 señales de velocidad 7 fueron reconocidas correctamente.
- La señal incorrecta fue 1 de 60km/h reconocida como de 50km/h.
- Cada una de las 3 señales de prohibido adelantar se reconocieron como "No es de velocidad".
- Cada uno de los 3 falsos positivos de reconocieron como "No es de velocidad".

De acuerdo a los resultados se puede apreciar que el reconocimiento es altamente confiable aún y con los incesantes cambios de iluminación. A pesar de que existen falsos positivos la sección de reconocimiento es capaz de categorizarlos como señales de tránsito que no son de velocidad.

Teniendo en cuanta las imágenes que reconoció en este vídeo el nivel de aciertos esta alrededor del 94 %. A nivel general y teniendo en cuenta más pruebas realizadas el nivel de confianza de la sección de reconocimiento queda en un 95 %.

5.3. Integrando GPS

Este es el paso más sencillo de todos, porque una vez realizado el código para obtener la latitud y longitud del GPS (se explico en el capitulo anterior) se realiza los siguientes pasos:

- Creación de la clase GPS y función ubicación,
- Se importa la clase GPS al código de detección y se hace llamado a la función ubicación cada vez que el clasificador detecta una señal. De esta manera se obtiene la ubicación aproximada de la señal y el archivo .kml generado se podrá ubicar en google earth.

6. FUNCIONAMIENTO EN CAMPO

Con todos los módulos de la aplicación integrados y funcionando se realiza varios recorridos con la aplicación trabajando en tiempo real. Los recorridos que se presentan aquí son de Pitalito-Timaná(vídeo 1), Timaná-Pitalito(vídeo 2), Pitalito-Villa Lobos(vídeo 3).

6.1. RECORRIDO VÍDEO 1

Del tiempo de grabación de este recorrido se analiza 20 minutos del desempeño de la aplicación. Se tiene en consideración que en el momento de realizar la grabación, se estaban realizando obras en la vía por lo que hubo muchas más señales de reglamentarias en escena y demás objetos que dificultaron su detección y reconocimiento.

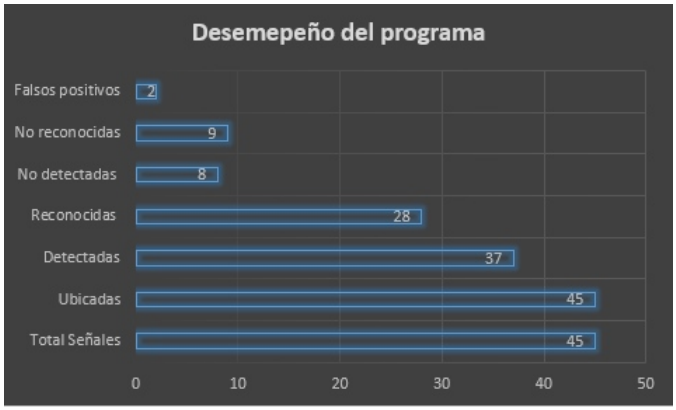
El perfil de los 20 minutos de grabación cuenta con un total de 45 señales reglamentarias a una velocidad promedio de 55km/h distribuidas de la siguiente manera.

- 2 señales de 20km/h.
- 9 señales de 30km/h.
- 7 señales de 40km/h.
- 1 señal de 50km/h.
- 7 señales de 60km/h.
- 4 señales de 80km/h.
- 15 señales de otro tipo (prohibido adelantar, girar a la derecha, etc)

El desempeño de la aplicación en este caso está evaluado por la cantidad de señales detectadas, la cantidad reconocida, falsos positivos y estableciendo las causas de los fallos.

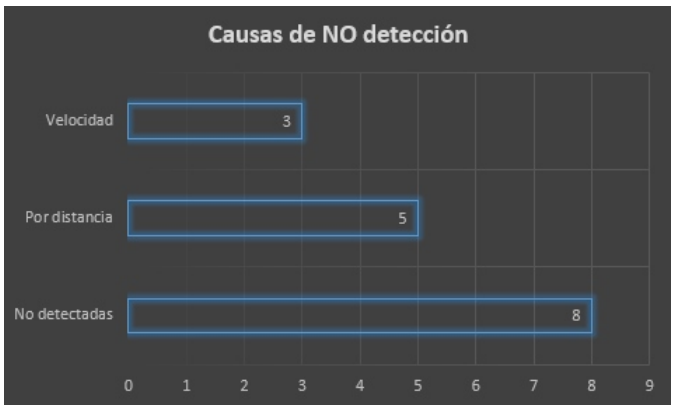
En la figura 23 Desempeño de la aplicación, figura 24 Causas de no detección y figura 25 Causas de no reconocimiento, se puede apreciar su funcionamiento.

Figura 23: Desempeño de la aplicación



Fuente: Autor

Figura 24: Causas de no detección



Fuente: Autor

Figura 25: Causas de no reconocimiento

Fuente: Autor

Se puede apreciar que la aplicación fue capaz de detectar el 82 % de las señales de tránsito en la vía y la principal causa para no haber detectado más es la distancia, debido a que se tuvo que cambiar de carril o sencillamente las señales aparecían a distancias muy largas (a más de 5m), por lo que el tamaño de la señal quedaba fuera del rango de detección. La segunda causa debe a la velocidad con la que se viaja en el vehículo puesto que las señal de tránsito se distorsionaba evitando así su detección. Este efecto de distorsión ocurre a partir de los 60km/h.

El reconocimiento se vio gravemente afectado por el brillo y la velocidad con la que se viajaba. Solo se reconoció un 75 % del total de las señales detectadas. El ángulo en que los rayos del sol daban al parabrisas aumentaba considerablemente el brillo y contraste con que se detectaban las señales. Esto causo un mal reconocimiento de las señales de transito.

Algo importante a tener en cuenta es que en todo los 20 minutos de grabación solo se detecto 2 falsos positivos. Si se compara con el total de señales detectadas en la vía, este tipo de detección corresponde a un 5.5 %, lo cual indica un buen desempeño de la aplicación este apartado.

Finalmente la ubicación de las señales detectadas funciono en un 100 %, el GPS funciono bien durante todo el trayecto.

Realizando un promedio con cada uno de los apartados se tiene que a nivel global la aplicación en este recorrido tuvo una tasa de éxito del 87.8 %

6.2. RECORRIDO VÍDEO 2

Para el caso de este recorrido las condiciones en las que se realiza el vídeo es similar al anterior, puesto que existen obras en la vía, el cual hace que existan muchos más elementos y señales reglamentarias.

Igualmente se analiza 20 minutos de grabación, en escena aparece un total de 50 señales de tránsito reglamentarias y la velocidad promedio de 50km/h. Las señales están distribuidas de la siguiente manera.

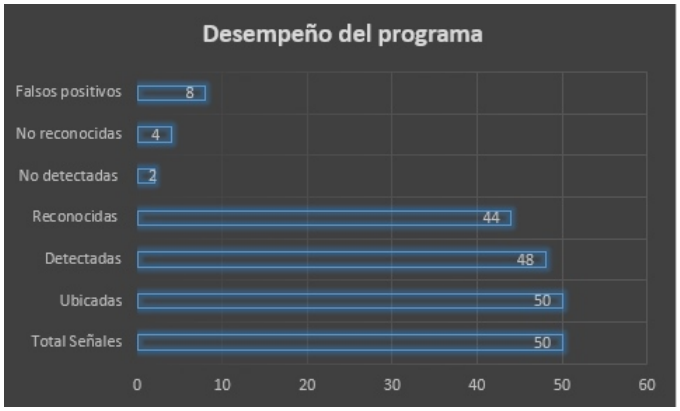
- 1 señal de 20km/h.
- 10 señales de 30km/h.
- 9 señales de 40km/h.
- 6 señales de 60km/h.
- 14 señales de 80km/h.

- 10 señales de otro tipo (prohibido adelantar, girar a la derecha, etc)

Al igual que en el vídeo anterior el desempeño de la aplicación en este caso está evaluado por la cantidad de señales detectadas, la cantidad reconocida, falsos positivos y estableciendo las causas de los fallos.

En la figura 26 Desempeño de la aplicación vídeo 2, se puede apreciar su funcionamiento.

Figura 26: Desempeño de la aplicación vídeo 2



Fuente: Autor

En esta ocasión el desempeño de la aplicación a presentado un mejor rendimiento. El GPS sigue siendo muy confiable, dando así una tasa de ubicación de las señales detectadas del 100 %.

El apartado de detección ha tenido rendimiento del 96 % y las causas por la que no se detectaron la totalidad de las señales fue por la distancia, dado que estas salieron del rango de detección. Es importante tener en cuenta que al no ir tan rápido en el vehículo la tasa de detección ha aumentado también.

Esta vez el brillo y la velocidad no han jugado excesivamente en contra, puesto que del total de señales detectadas se logrado reconocer un 92 %. La velocidad con la que se viaja toma un papel importante debido a que si la velocidad no es superior a 50km/h da tiempo a la cámara para adaptare mejor a los cambios de iluminación, permitiendo así un alza en la tasa de reconocimiento.

En cuanto al apartado de la detección de los falsos positivos, el desempeño disminuyo un poco, al detectar 8, comparado con el total de detecciones correctas corresponde a un 17 % aproximadamente, por lo que en este caso la aplicación decae un poco.

Promediando el desempeño en cada uno de los apartados, se tiene que para este recorrido el rendimiento global de la aplicación esta alrededor de un 92.8 %

6.3. RECORRIDO VÍDEO 3

Esta vez la ruta es completamente diferente y se adentra más al sur, igual que en los casos anteriores se analizará 20 minutos de grabación para tener una muestra del rendimiento de la aplicación.

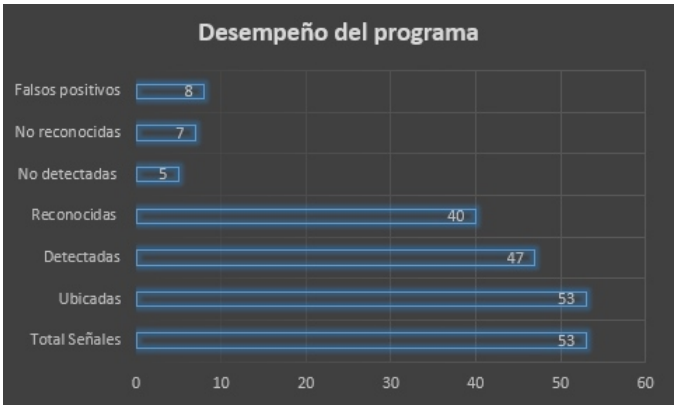
El vídeo cuenta con un total de 53 señales reglamentarias, la velocidad promedio fue de 60km/h, la distribución de las señales de tránsito es.

- 15 señales de 30km/h.
- 7 señales de 40km/h.
- 6 señales de 50km/h.
- 9 señales de 60km/h.
- 7 señales de 80km/h.
- 9 señales de otro tipo (prohibido adelantar, girar a la derecha, etc)

Al igual que en los vídeos anteriores el desempeño de la aplicación en este caso está evaluado por la cantidad de señales detectadas, la cantidad reconocida, falsos positivos y estableciendo las causas de los fallos.

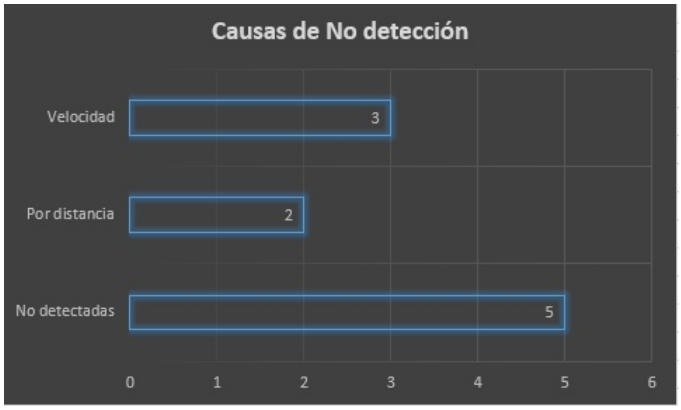
En la figura 27 Desempeño de la aplicación vídeo 3, figura 28 Causas de no detección vídeo 3 y figura 29 Causas de no reconocimiento vídeo 3, se puede apreciar su funcionamiento.

Figura 27: Desempeño de la aplicación vídeo 3



Fuente: Autor

Figura 28: Causas de no detección vídeo 3



Fuente: Autor

Figura 29: Causas de no reconocimiento vídeo 3

Fuente: Autor

El apartado de ubicación vuelve a ser el mejor de todos con 100 % de las señales detectadas ubicadas, esto indica que con el GPS no hay problemas y hay cobertura en cualquier dirección.

La detección de las señales decae a un 89 %. Si bien la distancia jugo un papel importante al quedar señales por fuera del rango de detección, la velocidad con la que se iba fue la principal causa para que la tasa de detección bajara.

La velocidad también afecto el reconocimiento y esto se puede palmar más si se tiene encuenta que 7 de las señales no reconocidas 5 eran de 60km/H. Estas señales fueron reconocidas como de 50km/h, la distorción que se empieza generar a esta velocidad afecta negativente la capacidad de clasificar la señal correctamente, haciendo que la

tasa de éxito fuese del 86 %.

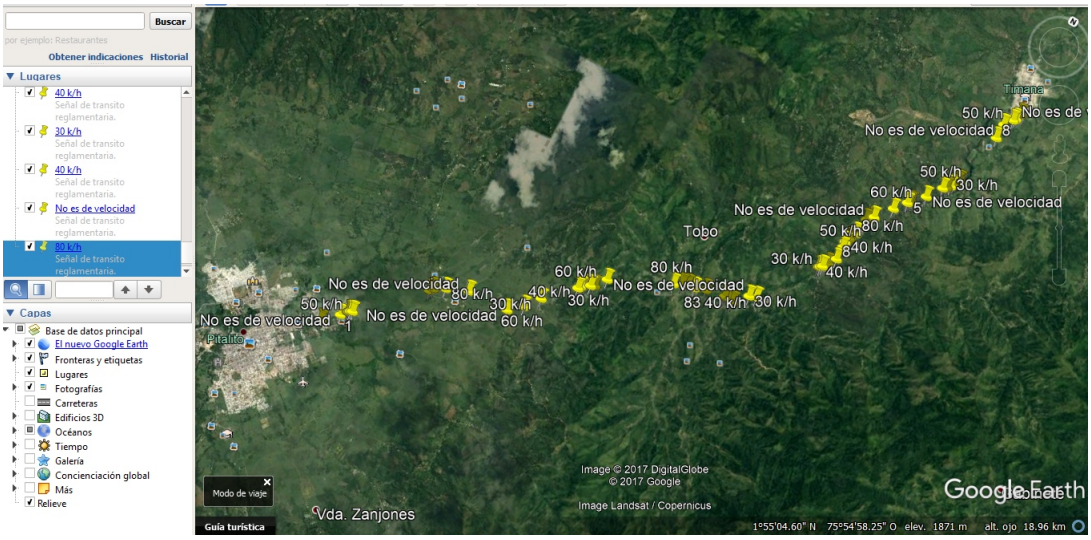
Los falsos positivos vuelven a aparece en un 17 % aproximadamente si se tiene en cuenta la cantidad de señales detectadas. La aparición de estos es aleatoria por lo que en ocasiones son mucho menos, en este caso el desempeño en este apartado decae un poco.

Una vez más el promedio global del funcionamiento de la aplicación a una velocidad promedio de 60km/h de acuerdo con cada uno de los apartados es de 89.5 %.

6.4. UBICACIÓN DEL LOS RECORRIDOS DEN EL MAPA

6.4.1. Video 1

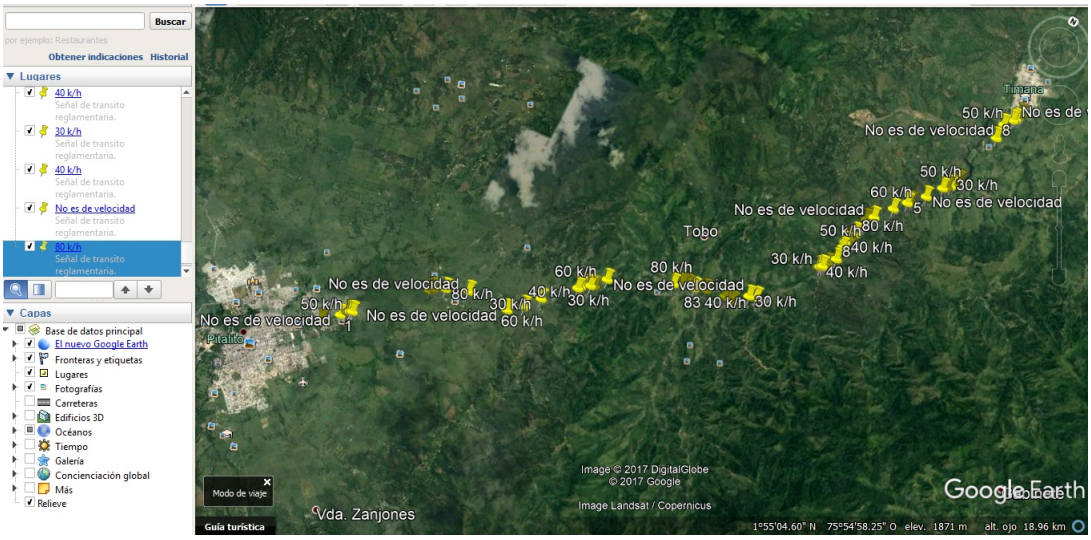
Figura 30: Pitalito-Timaná



Fuente: Autor

6.4.2. Video 2

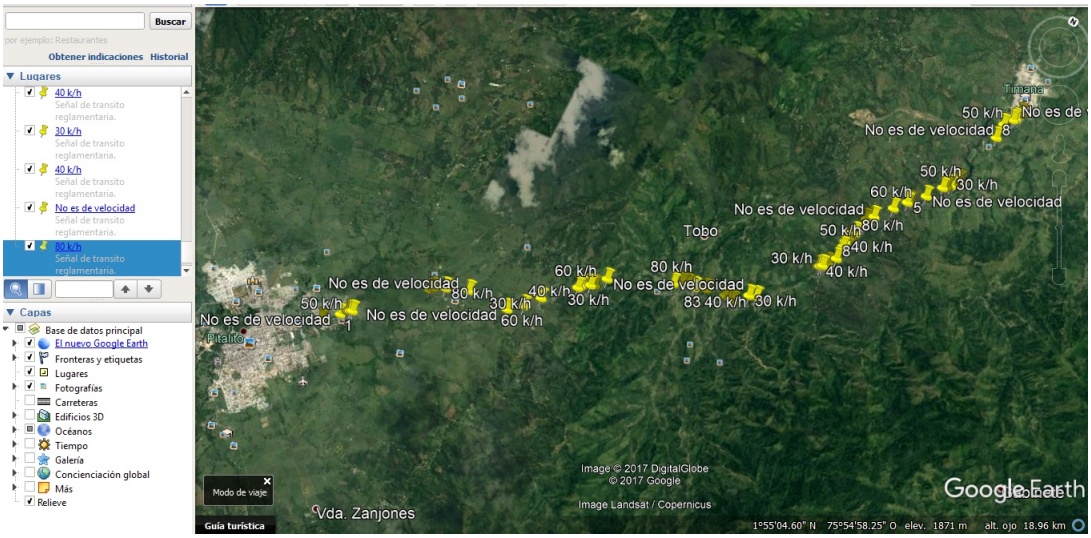
Figura 31: Timaná-Pitalito



Fuente: Autor

6.4.3. Video 3

Figura 32: Pitalito-Villa Lobos



Fuente: Autor

7. CONCLUSIONES

La preparación de datos para entrenar un clasificador cascada o el algoritmo kNN, es la parte más importante del proceso porque el rendimiento de los mismo se ve afectado por la calidad de datos que se dispongan para esta tarea.

El clasificador cascada resulta ser un método de detección de objetos altamente efectivo, el sistema adaboost permite tratar con grandes cantidades de características ahorrando tiempo de computo, el cual permite que se pueda usar en aplicaciones de tiempo real.

Para que un clasificador cascada tenga mejor desempeño a la hora de mermar la tasa de detección de falsos positivos, es conveniente que las muestras negativas sean de objetos que aparezcan en la escena.

El ruido es un mal que siempre va a afectar cualquier aplicación en tiempo real, por tanto se debe hacer todo lo posible por reducir los daños que este puede causar en el procesamiento de la imagen. El filtro gaussiano es efectivo para cumplir esta función gracias al suavizado que da a las imágenes. Para quitar aun más ruido una vez binarizada la imagen un efecto de opening mejora el resultado de la misma.

En espacios exteriores el ambiente es realmente hostil para la detección y reconocimiento de objetos, especialmente por los fuertes cambios de iluminación que se producen en todo momento. Para poder solventar este inconveniente lo máximo posible, es necesario aplicar métodos capaces de adaptare a estos cambios. Este es el caso del método Adaptive Threshold de OpenCV, este es capas de calcular un umbral cada vez que la iluminación cambia para resaltar mejor la imagen.

El algoritmo kNN es la técnica más básica de machine learning, sin embargo, es efectiva a la hora de emplearla en clasificaciones simples, como es el caso de los dígitos. Su uso para aplicaciones en tiempo real, resulta no ser la más conveniente puesto que debe calcular muchas características hasta determinar un resultado, tomando así mucho tiempo de calculo.

El lenguaje de programación python es sumamente versátil al contar con una comunidad amplía que desarrolla librerías para el uso de cualquier usuario, permitiendo de esta manera realizar proyectos ambiciosos.

El uso apropiado de los métodos y técnicas mostrados permiten que el programa de detección, reconocimiento y ubicación de señales de tránsito reglamentarias funcione de manera óptima y este ne la capacidad de brindar un excelente servicio.

8. RECOMENDACIONES

En el momento de usar la aplicación es importante que el usuario tenga en cuenta los siguientes aspectos.

- Velocidad promedio del vehículo menor o igual a 50km/h
- Enfocar la cámara al carril donde aparecen las señales de tránsito, así reducir un poco los objetos en escena.
- Es importante que la cámara quede ubicada de tal modo que las señales de tránsito aparezcan desde el centro de la pantalla.
- No usar el programa si está lloviendo, las gotas de agua en el parabrisas aumentan la dificultad de detección y reconocimiento.

9. TRABAJOS FUTUROS

Como trabajos futuros para mejorar aún más esta aplicación de detección, reconocimiento y ubicación de señales de tránsito reglamentarias se puede realizar:

- Capacidad para identificar las demás señales de tránsito, no solo las de velocidad.
- Usar un mejor algoritmo de machine learning capaz de tener un mejor desempeño sin importar la iluminación del entorno.
- Ampliar la gama de detección y reconocimiento a más tipos de señales de tránsito.
- Realizar una nueva versión capaz de funcionar en plataformas como Android e ios.

BIBLIOGRAFIA

Alegre, E, Pajares, G y Escalera, A. (2016). Conceptos y Métodos en Visión por Computador. Grupo de Visión del Comité Español de Automáticas (CEA).

K, Alexander Mordvintsev y Abid. https://readthedocs.org/. [En línea] 02 de Septiembre de 2017. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf.

Perez, C. y Yanovich, D. (1999). Sector Carreteras. [online] www.corficolombiana.com. Disponible en: https://www.corficolombiana.com/WebCorficolombiana/Repositorio /Informes/IS01021999.PDF

Pérez V., G. (2005). La infraestructura del transporte vial y la movilización de carga en Colombia. [online] http://www.banrep.gov.co. Disponible en: http://www.banrep.gov.co /docum/Lectura_finanzas/pdf/DTSER-64.pdf.

Margeé Garcia, G. (2002). Lesiones no intensionales, Muertes en accidentes de tránsito. [online] http://www.medicinalegal.gov.co. Disponible en: http://www.medicinalegal.gov.co /documents/10180/51788/Muertestransito(3).pdf2002.pdf

Garibello, A y Veloza Cano, H (2008). Accidentes por huecos y falta de señales en vías hacen que la Nación pierda cada vez más demandas. [online] http://www.eltiempo.com. Disponible en: http://www.eltiempo.com/archivo/documento/CMS-3943922

Elespectador.com (2012). Estado responderá por accidentes en carreteras por mala señalización [online] http://www.elespectador.com Disponible en: http://www.elespectador.com /noticias/judicial/estado-respondera-accidentes-carreteras-mala-senalizaci-articulo-355907

Coelho, Willi Richert Y Luis Pedro (2013). Building Machine Learning Systems with Python. s.l. : Packt Publishing Ltd., 1200713.

Zhang, Z. (2016). Introduction to machine learning: k-nearest neighbors. Annals of Translational Medicine, 4(11), 218. http://doi.org/10.21037/atm.2016.03.37

Blueprints, OpenCV 3. http://docs.opencv.org. [Online] [Consultado: 6 20, 2017.] http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html.

Jones, Paul Viola y Michael J. 2004. Robust Real-Time Face Detection. Netherlands : Kluwer Academic Publishers.

ANEXOS

ANEXO A: CÓDIGOS DISEÑADOS PARA EL PROGARMA

Código preparación de muestras positivas:

```
import cv2

ruta = "ruta de la ubicación de las imagenes"
txt = open('positivos.txt','a')
for archivos in os.listdir(ruta):
    imagen_pos = cv2.imread(ruta + "/" + archivos,cv2.IMREAD_GRAYSCALE)
    imagen_pos_redi = cv2.resize(imagen_pos,(50,50))
    imagen_ruido = cv2.GaussianBlur(imagen_pos_redi, (5, 5), 0)
    imagen_binarizada = cv2.adaptiveThreshold(imagen_ruido, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                        cv2.THRESH_BINARY, 11, 5)

    #cv2.imshow('ventana'+archivos,imagen_binarizada)
    cv2.imwrite('ruta de salida de las imagens /')
    txt.write('ruta' + archivos + ' 1 0 0 50 50' + '\n')
txt.close()
```

Código preparación de muestras negativas:

```
import cv2
ruta = " ruta de la ubicación de las imagenes "
ruta2 = "ruta salida_resultados/"
txt = open('negativos.txt','a')
for archivos in os.listdir(ruta):
    imagen_neg = cv2.imread(ruta + "/" + archivos,cv2.IMREAD_GRAYSCALE)
    imagen_neg_redi = cv2.resize(imagen_neg,(100,100))
    imagen_ruido = cv2.GaussianBlur(imagen_neg_redi, (5, 5), 0)
    imagen_binarizada = cv2.adaptiveThreshold(imagen_ruido, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                            cv2.THRESH_BINARY, 11, 5)
    #cv2.imshow('ventana'+archivos,imagen_pos_redi)
    cv2.imwrite(ruta2 + archivos,imagen_binarizada)
    txt.write(ruta2 + archivos + '\n')
txt.close()
```

Código entrenamiento algoritmo kNN:

```
import numpy as np
import cv2

imagen = cv2.imread('digitos4.png')
imagen2 = imagen.copy()
imagen2 = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
imagen2 = cv2.adaptiveThreshold(imagen2, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                        cv2.THRESH_BINARY, 11, 9)
thresh = imagen2.copy()
_,cnt,_ = cv2.findContours(imagen2,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
numeros = []
for i in cnt:
    area = cv2.contourArea(i)

    #Rango de area para reconocer los numeros, el area grende es porque
reconoce el contorno de la hoja
    if area >= 40 and area < 149100.0: #3300165.0  481374.0
        [x, y, w, h] = cv2.boundingRect(i)
        marcar = cv2.rectangle(imagen, (x, y), (x + w, y + h), (0, 255,
0), 2)
        numeros.append(i)

#Procedemos a ordenarlos

numeros2 = []
numeros2.extend(numeros[480:500])#1er pack de ceros
numeros2.extend(numeros[442:472])#2do pack de ceros
numeros2.extend(numeros[472:480])#1er pack de unos
numeros2.extend(numeros[430:442])#2do pack de unos
numeros2.extend(numeros[400:420])#3er pack de unos
numeros2.extend(numeros[384:394])#4to pack de unos
numeros2.extend(numeros[420:430])#1er pack de dos
numeros2.extend(numeros[394:400])#2do pack de dos
numeros2.extend(numeros[374:384])#3er pack de dos
numeros2.extend(numeros[346:360])#4to pack de dos
numeros2.extend(numeros[326:336])#5to pack de dos
numeros2.extend(numeros[360:374])#1er pack de tres
numeros2.extend(numeros[336:346])#2do pack de tres
numeros2.extend(numeros[320:326])#3er pack de tres
numeros2.extend(numeros[290:300])#4to pack de tres
numeros2.extend(numeros[268:278])#5to pack de tres
numeros2.extend(numeros[300:320])#1er pack de cuatro
numeros2.extend(numeros[278:290])#2do pack de cuatro
numeros2.extend(numeros[210:228])#3er pack de cuatro
numeros2.extend(numeros[228:268])#1er pack de cinco
numeros2.extend(numeros[160:170])#2do pack de cinco
numeros2.extend(numeros[170:210])#1er pack de seis
numeros2.extend(numeros[150:160])#2do pack de seis
numeros2.extend(numeros[130:150])#1er pack de siete
numeros2.extend(numeros[92:122])#2do pack de siete
numeros2.extend(numeros[122:130])#1er pack de ocho
numeros2.extend(numeros[80:92])#2do pack de ocho
numeros2.extend(numeros[34:44])#3er pack de ocho
numeros2.extend(numeros[50:70])#4to pack de ocho
numeros2.extend(numeros[70:80])#1er pack de nueve
numeros2.extend(numeros[44:50])#2do pack de nueve
```

```
numeros2.extend(numeros[0:34])#3er pack de nuve

print(len(numeros2))

muestras = np.empty((0,100)) #Matriz vacia donde se almacenara los
caracteres el 100 es debido
                             # al tamaño de cada caracter es 10*10
#j= 0
for i in numeros2: #Cilco para almacenar cada caracter
    [x, y, w, h] = cv2.boundingRect(i)
    roi = thresh[y:y+h,x:x+w]

#cv2.imwrite('D:/usuario\Mr_roa\Documents\pycharm\knn\orden/'+str(j)+'.jp
g',roi)
    #j = j+1
    roismall = cv2.resize(roi,(10,10))  #Redimencionamos cada caracter a
un tamaño de 10*10
    muestra = roismall.reshape((1, 100)) #Cambiamos las dimenciones para
que que de 1 fila y 100 columnas los valores del caracter
    muestras = np.append(muestras, muestra, 0) #Almancemos el caracter
dentro de la matriz


#etiquetas, creamos la respuesta para cada muestra, va de 0 a 9, cada
número se repite la misma cantidad de
# veces que el de la muestra

eti = np.arange(10)
train_labels=np.repeat(eti,50)[:,np.newaxis]

np.savetxt('muestras_50.txt',muestras)
np.savetxt('respuestas_50.txt',train_labels)

cv2.imshow('digitos',imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Código reconocimiento de caracteres, algoritmo kNN:

```
import numpy as np
import cv2

class reconocomineto():
    def numeros(imagen):
        x = 17       # x = 15
        y = 22       #y = 20
        w = 84       #w = 82
        h = 71       #h = 69

        #imagen = cv2.imread('senal_dec.jpg')
        imagen1 = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
        imagen1= cv2.adaptiveThreshold(imagen1, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                       cv2.THRESH_BINARY, 11, 6)
        kernel = np.ones((2, 2), np.uint8)
        kernel2 = np.ones((4, 4), np.uint8)
        #imagen = cv2.erode(imagen, kernel, iterations=1)
        #imagen = cv2.dilate(imagen, kernel, iterations=2)
        imagen1 = cv2.morphologyEx(imagen1, cv2.MORPH_OPEN, kernel2)
        imagen1 = imagen1[y:y + h, x:x + w]
        imagen2 = imagen1.copy()
        #cv2.imshow('image', imagen1)
        #cv2.waitKey(0)
        #cv2.destroyAllWindows()
        _, cont, _ = cv2.findContours(imagen1, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
        #are1 = []
        ##-------Determinar las areas mas grandes (6)--> solo los
caracteres------
        #for i in cont:
            #area1 = cv2.contourArea(i)
            #are1.append(area1)
            #are1.sort(reverse=True)  # ordenar de manera descendetes
elementos
        #print(are1)
        #rango = are1[3]  # Rango para determinar los caracteres

        ##---------------Extracción de caracteres-----------
        caracter = []
        for i in cont:
            area1 = cv2.contourArea(i)
            #print(area1)
            if area1 >= 350 and area1 <= 1000: # 400 / 980
                x, y, w, h = cv2.boundingRect(i)
                #print(x, y, w, h)
                # cv2.rectangle(img_cor, (x, y), (x + w, y + h), (255, 0,
0), 1)
                carac = imagen2[y - 5:y + h + 5, x - 5:x + w + 5]
                #b = cv2.resize(carac, (96, 146))
                #cv2.imshow('image', carac)
                #cv2.waitKey(0)
                #cv2.destroyAllWindows()
                caracter.append(carac)
        if len(caracter) <= 1:
            return "No es de velocidad"
```

```
        cv2.imwrite('image/caracter1.jpg', caracter[0])
        cv2.imwrite('image/caracter2.jpg', caracter[1])

        caracter1 = cv2.imread('image/caracter1.jpg',0)
        caracter2 = cv2.imread('image/caracter2.jpg',0)

        caracteres = []
        caracteres.append(caracter1)
        caracteres.append(caracter2)

#cv2.imshow('image',imagen)
#cv2.waitKey(0)
#cv2.destroyAllWindows()
        j = 0
        resultado = ''
        for i in range(0,2):

            try:
                imagen_resize = cv2.resize(caracteres[j],(10,10))
            except:
                continue
            j = j + 1
            #cv2.imshow('image',imagen_resize)
            #cv2.waitKey(0)
            #cv2.destroyAllWindows()
            imagen_reshape = np.reshape(imagen_resize,(1, 100))
            imagen1 = np.float32(imagen_reshape)
            #print(np.shape(imagen))
            entreno =
np.loadtxt('D:/usuario\Mr_roa\Documents\pycharm/muestras_50.txt',np.float
32)
            resp =
np.loadtxt('D:/usuario\Mr_roa\Documents\pycharm/respuestas_50.txt',np.flo
at32)
            resp = resp.reshape((resp.size,1))
            #print(np.shape(resp))
            #print(np.shape(entreno))
            modelo = cv2.ml.KNearest_create()
            modelo.train(entreno,cv2.ml.ROW_SAMPLE,resp)
            ret, results, neighbours ,dist =
modelo.findNearest(imagen1,1)
            resultado = resultado + str(int(results[0][0]))

            entrenos =  imagen
            correct = np.count_nonzero(entrenos)
            precision = correct*100.0 / entrenos.size

            #print(precision)



        if resultado == '01' or resultado == '10':
            return '10 k/h'
        if resultado == '02' or resultado == '20':
            return '20 k/h'
        if resultado == '03' or resultado == '30':
            return '30 k/h'
```

```
        if resultado == '04' or resultado == '40':
            return '40 k/h'
        if resultado == '05' or resultado == '50':
            return '50 k/h'
        if resultado == '06' or resultado == '60':
            return '60 k/h'
        if resultado == '07' or resultado == '70':
            return '70 k/h'
        if resultado == '08' or resultado == '80':
            return '80 k/h'
        if resultado == '09' or resultado == '90':
            return '90 k/h'
        return resultado
```

Código de ubicación mediante GPS

```
import serial

class GPS():
    def localizar(nombre):
        gps = serial.Serial('COM3', baudrate = 4800)
        while True:
            line = str(gps.readline())
            datos = line.split(',')
            #print(datos)
            if datos[0] == "b'$GPRMC":
                if datos[2] == 'A':
                    latgps = float(datos[3])
                    if datos[4] == 'S':
                        latgps = -latgps

                    latdeg = int(latgps/100)
                    latmin = latgps - latdeg*100
                    lat = latdeg + (latmin/60)

                    longps = float(datos[5])
                    if datos[6] == 'W':
                        longps = -longps

                    londeg = int(longps / 100)
                    lonmin = longps - londeg * 100
                    lon = londeg + (lonmin / 60)

                    #print(lat)
                    #print(lon)

                    with open ((str(lon)+'.kml'),'w') as pos:
                        pos.write("""<?xml version="1.0" encoding="UTF-
8"?>
        <kml xmlns="http://www.opengis.net/kml/2.2"> <Placemark>
         <name> %s </name>
         <description>Señal de transito reglamentaria.</description>
         <Point>
         <coordinates> %s,%s </coordinates>
         </Point>
         </Placemark> </kml>""" % (nombre,lon,lat))

                        return str(lat),str(lon)
```

Código principal

```
import cv2
import numpy as np
from time import time
import reconomiento_caracter
import gps

senal_cascade = cv2.CascadeClassifier('cascade_4000_380.xml')
#cascade_4000_380 cascade_dosmil_270
capturar = cv2.VideoCapture('video_11.avi') #señales2.avi
#fourcc = cv2.VideoWriter_fourcc(*'XVID')
#video_salida = out=cv2.VideoWriter('señal_detectada6.avi',fourcc,30.0,
(720,480))
verifcar_tiempo = []
while True:
    ret, frame = capturar.read()
    frame_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_gris = cv2.GaussianBlur(frame_gris, (5, 5), 0)
    frame_gris = cv2.adaptiveThreshold(frame_gris, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                       cv2.THRESH_BINARY, 11, 8)
    kernel = np.ones((2, 2), np.uint8)
    frame_gris = cv2.erode(frame_gris, kernel, iterations=1)
    frame_gris = cv2.dilate(frame_gris, kernel, iterations=1)

    senal = senal_cascade.detectMultiScale(frame_gris, 1.3, 6)


    tiempo_inicial = time()
    for (x,y,w,h) in senal:

        vacia = np.empty((x+w, y+h))
        #print(len(vacia))

        if len(verifcar_tiempo) <= 10:
            verifcar_tiempo = []
            break

        roi_color = frame[y:y + h, x:x + w]
        hsv = cv2.cvtColor(roi_color, cv2.COLOR_BGR2HSV)
        rojo_bajos1 = np.array([0, 65, 75], dtype=np.uint8)
        rojo_altos1 = np.array([12, 255, 255], dtype=np.uint8)
        #rojo_bajos1 = np.array([160, 0, 0], dtype=np.uint8)
        #rojo_altos1 = np.array([255, 45, 45], dtype=np.uint8)
        mask = cv2.inRange(hsv, rojo_bajos1, rojo_altos1)
        kernel = np.ones((10, 10), np.uint8)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

        # Difuminamos la mascara para suavizar los contornos y aplicamos
filtro canny
        blur = cv2.GaussianBlur(mask, (7, 7), 0)
```

```
        edges = cv2.Canny(blur, 1, 2)
        _, contornos, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

        if len(contornos) > 0:

            if (len(vacia)) >= 430 and len(vacia) <= 520:  #430 / 530
                cv2.rectangle(frame, (x,y), (x+w, y+h), (255,0,0),2)

                roi_gray = frame_gris[y:y + h, x:x + w]


                cv2.imwrite('senal_dec.jpg',roi_gray)
                senal_dec = cv2.imread('senal_dec.jpg')

                senal_dec = cv2.resize(senal_dec,(110,110))
                roi_color = cv2.resize(roi_color,(440,440))
                cv2.imshow('señal', roi_color)


                resultado =
reconomiento_caracter.reconocomineto.numeros(imagen=senal_dec)
                localizar = str(gps.GPS.localizar(resultado))
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.putText(roi_color,resultado,(10,44),font,

fontScale=1,color=(0,255,0),thickness=2,lineType=2)
                #cv2.putText(roi_color, localizar, (10, 94), font,
                            #fontScale=0.5, color=(0, 255, 0),
thickness=2, lineType=2)

                cv2.imshow('señal', roi_color)
                print(resultado)
                #print(localizar)
                verifcar_tiempo = []
                break




    tiempo_final = time()
    tiempo_total = tiempo_final-tiempo_inicial
    verifcar_tiempo.append(tiempo_total)
    #print('########')
    #print(tiempo_total)

    #video_salida.write(frame)
    cv2.imshow('img', frame)

    #cv2.imshow('img', frame_gris)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```