Universidad Tecnológica Nacional

Proyecto Final

Investigación e implementación de algoritmos de visión asistida aplicados a estimación de flujo vehicular y velocidad de automóviles

---

Autores:

- Comas, Sergio Johann
- Taborda, Félix Matías

Director:

- Burgos, Sergio Enrique

Proyecto final presentado para cumplimentar los requisitos académicos para acceder al título de Ingeniero Electrónica

en la

Facultad Regional Paraná

Marzo de 2019

Agradecimientos:

En primer lugar, a nuestras familias por el afecto y apoyo incondicional en todo momento de nuestras carreras. A todo el personal docente y no docente por todo el afecto y conocimientos transmitidos. Y por último a nuestros amigos y compañeros de facultad con los que transitamos este camino de estudio.

Comas, Sergio Johann

Taborda, Félix Matías

Universidad Tecnológica Nacional

Abstract

Facultad Regional Paraná

Ingeniero en Electrónica

Investigación e implementación de algoritmos de visión asistida aplicados a estimación de flujo vehicular y velocidad de automóviles

Comas, Sergio Johann
Taborda, Félix Matías

Abstract:

Motivated by the fact that in the subfluvial tunnel information on traffic parameters cannot be accessed from the command room, different alternatives were assessed to obtain such information. As in the room only video information from the security cameras is available, it was concluded that artificial vision techniques are the most appropriate to obtain this information inside the tunnel.

Different algorithms were assessed to identify the most efficient one for this purpose and then it was implemented in a general purpose PC. The software tools used were GCC, QT and OpenCV. With these, it was possible to estimate the speed of the vehicles and classify them by means of different image filters.

After obtaining a functional version of our software, we investigated the feasibility of running it in a low-resource device using different development kits, such as RaspBerry Pi 3, Zybo Zynq-7000 and BeagleBone Black. Tests were carried out on these devices but

results were not satisfactory under the tunnel conditions. Therefore, a neural network solution was proposed that makes it possible to make the estimates. In the RaspBerry Pi 3, however, the algorithm performed very well when used under good light conditions and the right camera positions.

Keywords:
Area - Blur – contours – detection – Embedded System - GCC – Lucas Kanade – Moment – Morphology – OpenCv – Optical Flow – Qt

Resumen:
Motivados por la problemática del túnel subfluvial en la cual la información sobre los parámetros de tráfico no puede ser vista desde la sala de comandos, se comienza a investigar sobre distintas alternativas de medición de estos parámetros, ya que solo se poseía la información del video de las cámaras de seguridad. En base a esto se concluye que uno de los métodos más conveniente dentro del túnel para obtener esta información son las técnicas de visión artificial.
Se realizó una investigación sobre los distintos algoritmos para poder determinar cuál era el más eficiente para realizar dicha tarea y luego se procedió a implementar estos en una PC de propósito general. Las herramientas de software utilizadas fueron GCC, QT y OpenCV. Con estos se logró llegar, mediante distintos filtrados de la imagen, a estimar la velocidad de los vehículos y a poder hacer una clasificación de los mismos.
Luego de tener una versión funcional de nuestro software se investigó sobre la viabilidad de portar el mismo a un dispositivo de bajos recursos, utilizando para esto distintos kits de desarrollo, como fueron la RaspBerry Pi 3, Zybo Zynq-7000 y BeagleBone Black. Se realizaron pruebas en estos dispositivos, obteniendo resultados poco satisfactorios para las condiciones del túnel, y se propuso una solución mediante redes neuronales que hace posible la realización de las estimaciones. A pesar de esto, el algoritmo en la RaspBerry Pi 3 se comportó de manera excelente en buenas condiciones de luz y posiciones de cámara.

Palabras Clave:
Área - Blur – Contornos – Detección – GCC – Lucas Kanade – Momento - Morfología – OpenCv – Flujo Óptico - Qt – Sistema Embebido

Reconocimientos:

Se presta especial reconocimiento al Ingeniero Sergio Burgos, por su apoyo y contribución en el presente trabajo, y a nuestra querida facultad Universidad Tecnológica Nacional – Regional Paraná.

Índice:

Lista de Figuras:

Lista de Tablas:

Dedicado a:

Nuestras familias y seres queridos que nos acompañaron en este camino, nos apoyaron y alentaron en todo momento.

Capítulo 1: Introducción

La visión artificial o computarizada se define como una disciplina científica que desarrolla métodos para capturar, procesar, analizar y comprender imágenes del mundo real “con el fin de producir información numérica o simbólica para que pueda ser tratada por una computadora”. La intención es que “las computadoras puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación”. Esta disciplina permite, por ejemplo, detectar, segmentar, ubicar o reconocer determinados objetos y seguirlos en una secuencia de imágenes.

Durante los últimos años, el incremento del poder de la computación y el precio más accesible de las cámaras hizo que la visión computarizada saliera de los laboratorios para apuntalar diferentes aplicaciones en el mundo real. Entre ellas se encuentra la video vigilancia, conteo de vehículos, control de la velocidad, extracción de parámetros fundamentales del tráfico, clasificación de vehículos, detección de incidentes y anomalías, aplicación de la señalización circunstancial adaptativa, y servicios de información sobre el estado del tráfico.

El objetivo de este trabajo consiste en estudiar y proponer técnicas de estimación de velocidad y análisis de tráfico mediante el uso de visión artificial. Dichas técnicas proponen partir de una solución implementada en una computadora de propósito general, luego portarla a un dispositivo de bajos recursos, y finalmente optimizar las tareas que demandan mayor procesamiento.

Este proyecto va dirigido a entes de control de tránsito privados y públicos, organismos encargados de la medición y recolección de datos de tráfico, etc.

Actualmente existen empresas que comercializan productos basados en técnicas de visión artificial, para el análisis de diferentes patrones y comportamiento de objetos, como lectores de patentes, detectores de vehículos en banquina, seguimiento de tráfico, reconocimiento de datos biométricos, etc.

La ventaja que tienen este tipo de sistema es que puede ser utilizado en base a instalaciones de monitoreo de video ya existentes, evitando tener que agregar un sistema adicional para la determinación de los parámetros antes mencionados.

Desde el punto de vista tecnológico, es un sector cuyo desarrollo está íntimamente ligado a la informática y a la capacidad computacional que existe en un momento determinado. Debido a esta característica, es muy probable que aún quede por explotar el mayor potencial que puedan alcanzar sus tecnologías.

Desde el punto de vista cultural, todavía existe un claro desconocimiento, por parte de los usuarios finales, de las capacidades y limitaciones de un sistema de visión artificial. Esto genera cierta desconfianza a la hora de optar por una tecnología de este sector frente a sus productos sustitutivos.

Para finalizar hay que tener en cuenta que, aunque se han implantado muchos sistemas basados en visión artificial con resultados positivos, hay muchos escenarios en los que esta tecnología no está a la altura de las circunstancias, lo que puede suponer un riesgo si las tecnologías alternativas tienen una evolución más rápida.

Capítulo 2: Desarrollo

2.1 Descripción

Nuestro proyecto se puede dividir en varias etapas. Se comienza obteniendo la imagen por medio de una cámara o filmación para luego ser digitalizada y procesada. Dentro de este procesamiento se puede destacar como primer paso un segmentado de imagen, el cual separa el área de interés descartándose el resto de la información. Una vez extraída el área de relevancia, se aplican técnicas de procesamiento para la obtención de características tales como área, perímetro, excentricidad, momentos de inercia, esqueletos, etc. Y por último se realiza el reconocimiento e interpretación de los parámetros deseados.

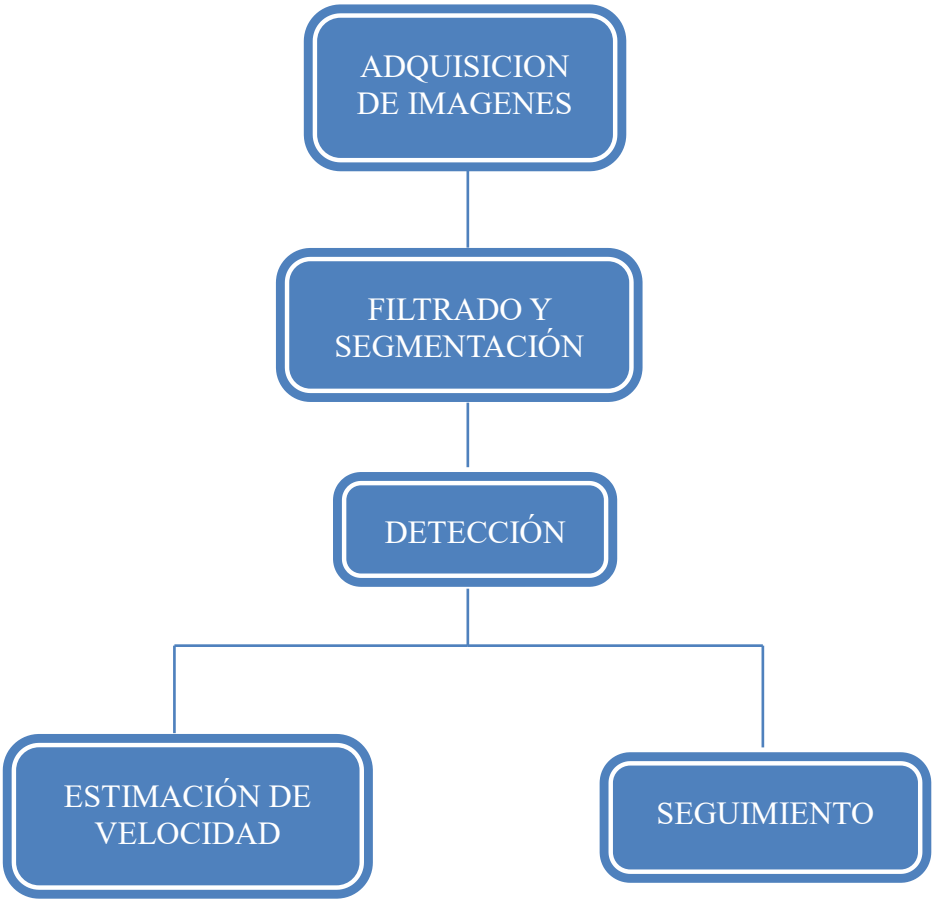


Diagrama de bloques general

2.1.1 Adquisición de imágenes

Las imágenes pueden llegar a la computadora de dos formas diferentes. La primera es a través de una placa digitalizadora, la cual convierte la señal de la cámara analógica, en datos binarios, y a través de un protocolo preestablecido, es llevado a la memoria de la PC, para su procesado.

La segunda forma es a través de un video, previamente digitalizado y almacenado en un formato preestablecido. Este es el caso usado en ese trabajo, ya que mejora el tiempo de desarrollo al no tener que estar en el lugar físico donde se encuentran las cámaras. El formato del video es avi, con resolución 860 x 480.

Las condiciones primordiales que debe cumplir la señal de video para poder ser procesado son:

-Luminosidad: Debe haber luminosidad constante, ya que al disminuir esta, aumenta la perdida de color y no se pueden distinguir los objetos.

-El ángulo de la cámara: Esta debe estar en un ángulo, en la cual no se produzcan solapamientos en la imagen.

-Nitidez: Debe ser lo suficiente para identificar los contornos de los objetos.

Una vez que el programa extrae los datos del video o de la memoria en la que escribe la placa digitalizadora, el proceso siguiente es indiferente para cualquiera de los dos casos.

2.1.2 Análisis y selección de las funciones para el filtrado y segmentado de la imagen.

Para lograr un porcentaje de acierto elevado en la detección, clasificación y seguimiento, se usan distintas funciones que ayudan a extraer los datos de interés de las imágenes procesadas.

Los más usados, en el tratamiento de las imágenes son:

- cvtColor () – Conversión a escala de grises: antes de realizar ninguna operación con las imágenes, es conveniente convertir a escalas de grises. Resulta menos complejo y óptimo trabajar con este tipo de imágenes.

Figura 1 - Resultado de conversión a escalas de grises en una imagen [11]

- gaussianBlur () – Eliminación de ruido: Se minimizo el ruido provocado por la propia cámara y por la iluminación. Esto se hace a través de promediar cada pixel con sus vecinos. Se conoce comúnmente como suavizado.

Figura 2 - Resultado de aplicarle un Gaussian Blur a una imagen [11]

- medianBlur () - Es un filtro, el cual se basa en el promediado de los pixeles que se encuentran dentro del Kernel. El valor obtenido se coloca en el centro del mismo, reemplazando al valor anterior. El Kernel tiene que tener filas y columnas impares. Su función es eliminar el ruido sal y pimienta en las imágenes.

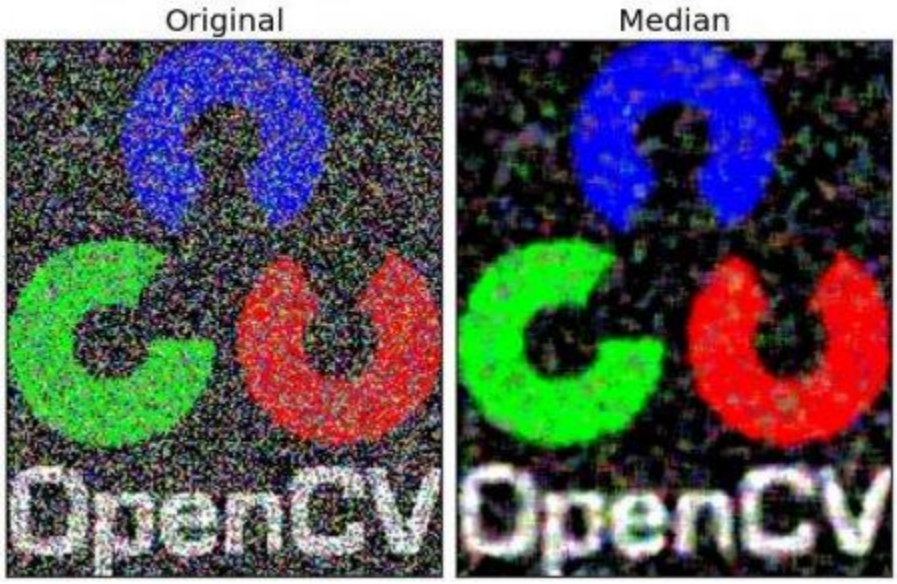


Figura 3 - Salida del algoritmo MedianBlur [11]

- bilateralFilter () - Es altamente efectivo para eliminar el ruido mientras mantiene los bordes. Es una funcion muy costosa en tiempo de procesado. Este filtro gaussiano es una función del espacio solo, es decir, los píxeles cercanos se consideran mientras se filtra. No considera si los píxeles tienen casi la misma intensidad. No considera si el píxel es un píxel de borde o no. Así que difumina también los bordes, lo que no queremos hacer.
- Blur () - Este tipo de filtrado tiene un efecto de desenfoque gaussiano en la imagen pasada como parámetro. Recibe como parámetro una matriz de N x M con el cual se convoluciona la imagen para producir dicho efecto.
- Absdiff () – Substracción entre el segundo y el primer frame: a la hora de restar una imagen con otra, se debe tener en cuenta que podemos obtener valores negativos. Para evitar esto lo que hicimos fue restar y obtener los valores absolutos de cada pixel.
- Threshold () – Aplicación de umbral: En esta parte del proceso lo que hacemos es quedarnos con aquellos pixeles que superen cierto umbral. El objetivo es binarizar la imagen, es decir, tener dos posibles valores. Todos aquellos que superen el umbral serán pixeles blancos. Esto nos servirá para seleccionar el objeto en movimiento.

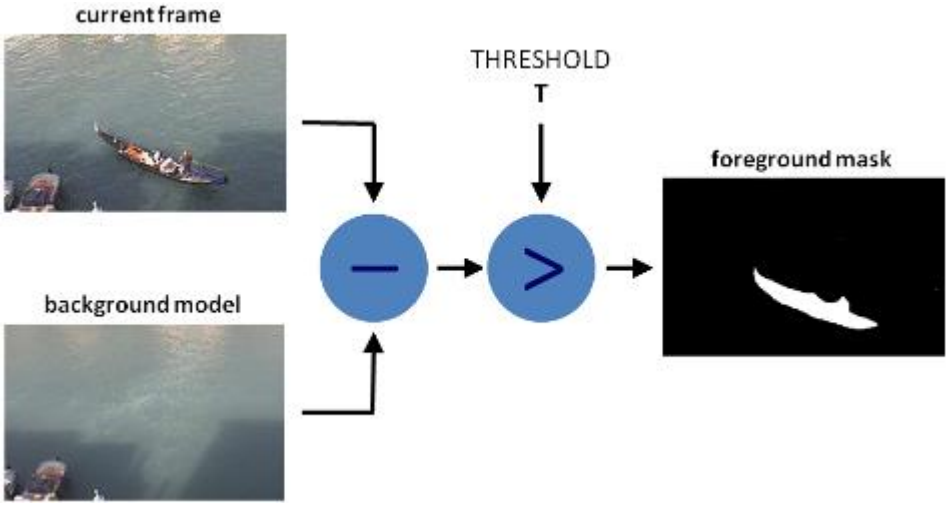


Figura 4 - Aplicación de la diferencia entre dos frame seguido de un umbral [9]

- Morfología:

Las operaciones morfológicas a imágenes se definen como procedimientos en los cuales cada nuevo pixel de la imagen resultante es obtenido de una operación no lineal entre un conjunto de puntos de la imagen original F[x,y] y un conjunto de puntos conocido con el nombre de elemento estructurante S[x,y]. Este elemento estructurante recorre toda la imagen para obtener todos los puntos de la nueva imagen, sin cambiar el tamaño de la imagen de salida.

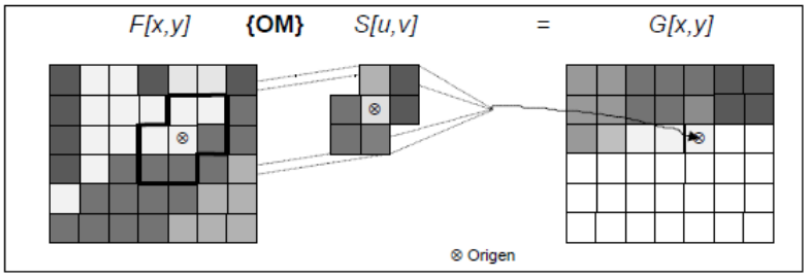


Figura 5 - Operación morfológica de un elemento estructurante S[u,v] sobre un área de la imagen F[x,y] para obtener una imagen G[x,y] [9].

Dependiendo de los elementos estructurantes y de las operaciones utilizadas, los filtros morfológicos, pueden detectar bordes en las imágenes, filtrar objetos de tamaños menores a uno determinado, suavizar fondos texturizados, detectar fallos en patrones de textura, etc.

En imágenes binarias las operaciones morfológicas son la dilatación y erosión, las cuales consiste en tomar el elemento estructurante (uno de cuyos puntos se considera el origen) y superponerlo a la imagen.

Si en el proceso de erosión el elemento estructurante está completamente en la imagen, el punto de esta se mantiene, de lo contrario pasa a ser del color del fondo. La erosión se aplica para reducir saliente de objetos, eliminar partes de tamaño menor que el elemento estructurante y eliminar conexiones insignificantes entre objetos mayores.

En el proceso de dilatación si parte del elemento estructurante es igual al contenido de la imagen, el punto de la imagen pasa a ser igual al punto origen del elemento estructurante, de otra forma pasa a ser del color del fondo. La dilatación se aplica para rellenar entrantes de objetos y unir pequeñas separaciones en las imágenes.

Figura 6 - Ejemplo de uso de la función Erosión y Dilation [8]

- fillPoly () - Con esta función se dibuja un polígono que contornea el área de interés, dejando el resto de color negro. Para esto se le pasa un vector de puntos, los cuales son los vértices del polígono.

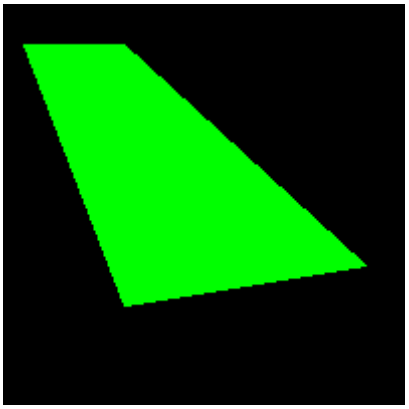


Figura 7 - Ilustración de figura realizada con fillPoly() [5]

- Line() – Esta función sirve para graficar líneas en la imagen. Variando el grosor de la línea se puede usar para tapar una zona de la imagen.

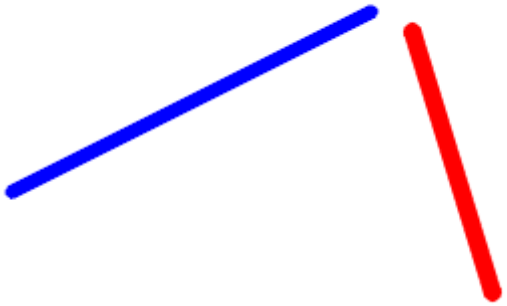


Figura 8 - Líneas de distintos colores creadas con la función line () [12]

2.1.3 Análisis y selección de técnicas de procesamiento para la detección del objeto de interés.

Una vez que la imagen tiene la información de interés resaltada, se pasa a la siguiente etapa, la cual se encarga de detectar los vehículos. Su funcionamiento se puede implementar de numerosas maneras, algunas con mayor costo computacional, pero más eficiente y otras más simples y rápidas, pero con la desventaja de cometer mucho error en la detección.

Blobs - Un blob es un conjunto de pixeles interconectados que comparten una característica en común, por ejemplo, el valor en la escala de grises. Para poder detectar esta particularidad, se usa la función SimpleBlobDetector. Esta es una función relativamente simple, la cual, a través de parámetros pasados como argumentos, se puede controlar la forma en la que realiza la detección.

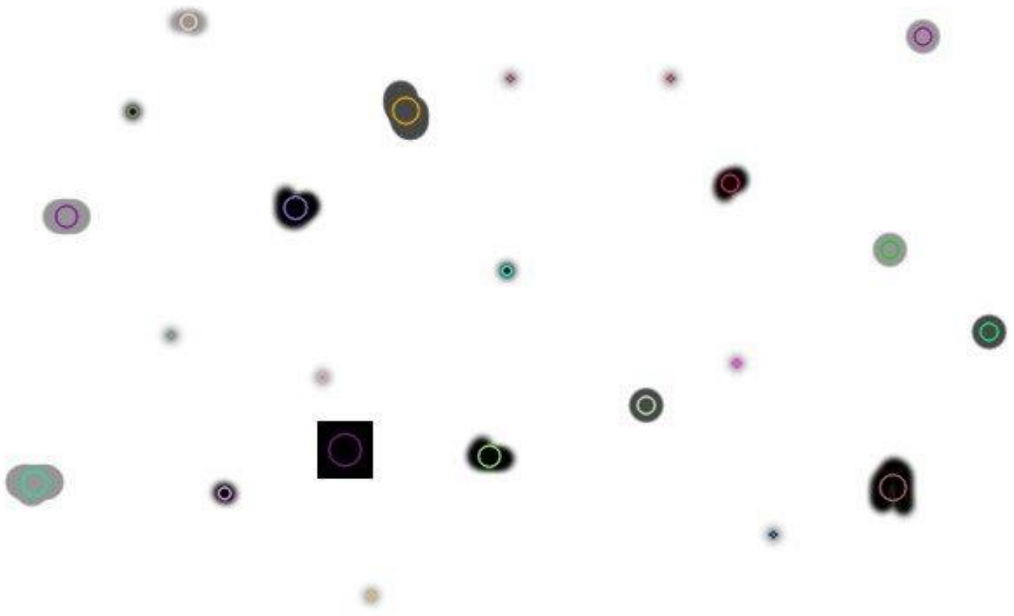


Figura 9 - Ejemplo de la salida de la detección de Blobs en la imagen. [13]

- findContours() - Este algoritmo busca los contornos en la imagen, tanto internos como externos y nos devuelve su ubicación en un vector, topología. Esta función tiene un consumo mínimo de procesador. Es muy usada para detectar formas, colores u objetos.

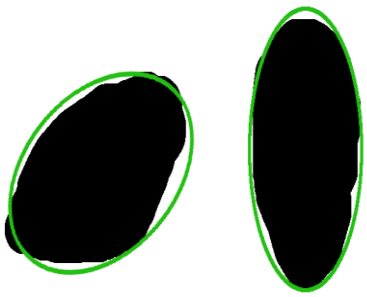


Figura 10 - Contorno detectado a través de la función findContours [6]

Detector de características Haar-like: Este tipo de detección de objetos, es más eficiente que los algoritmos vistos anteriormente. Se basa en un conjunto de clasificadores (funciones lógicas, generalmente lineales) en cascada, los cuales se encargan de detectar características para la que fueron entrenados. Si la detección es acertada, se pasa los datos validos al próximo bloque, así sucesivamente hasta llegar al último el cual confirma el dato buscado. En el caso contrario si no pasa el umbral de acierto de algún clasificador, se descarta la información y se prosigue con el siguiente dato. Individualmente no tienen la capacidad de realizar una detección completa, pero la suma de todos da una probabilidad de éxito elevada, dependiendo del entrenamiento del clasificador, esta tasa puede llegar a valores del 95-99%.

La entrada de los datos a procesar, se realizan extrayendo distintas ventanas con porciones de la imagen. Cada una de estas es enviada al clasificador.

En general, un número elevado de estas hipótesis (ventanas), no corresponde al dato buscado. Las cuáles serán descartadas mientras pasan de bloque en bloque por todo el clasificador.

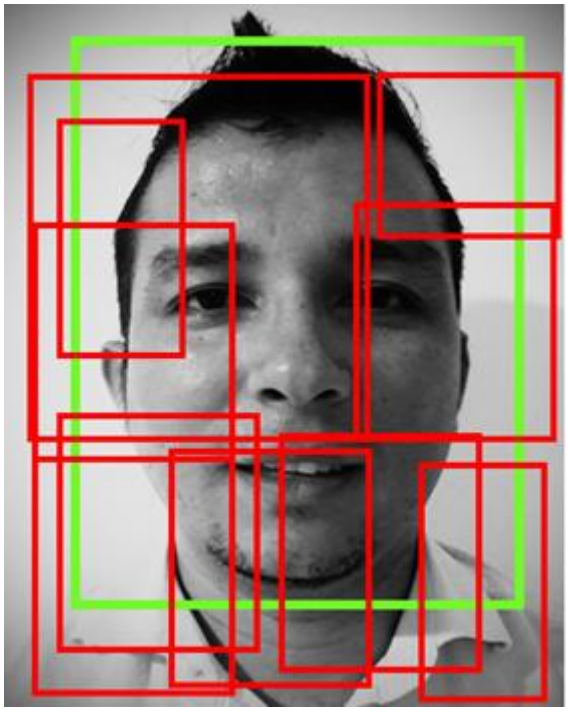


Figura 11- Ejemplo de una imagen que contiene un rostro, la cual se le han extraído distintas porciones para ser procesadas por el clasificador [3].

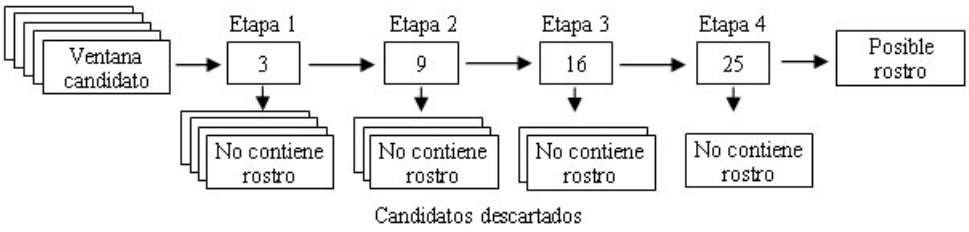


Figura 12 - Ilustración de funcionamiento de un clasificador cascada [4].

El caso con más información sobre la implementación de este tipo de algoritmo es la detección de rostros. Los clasificadores se encargan de detectar características como: ojos, nariz, boca, etc. Cada uno de estos datos se codifican como un dato a buscar.

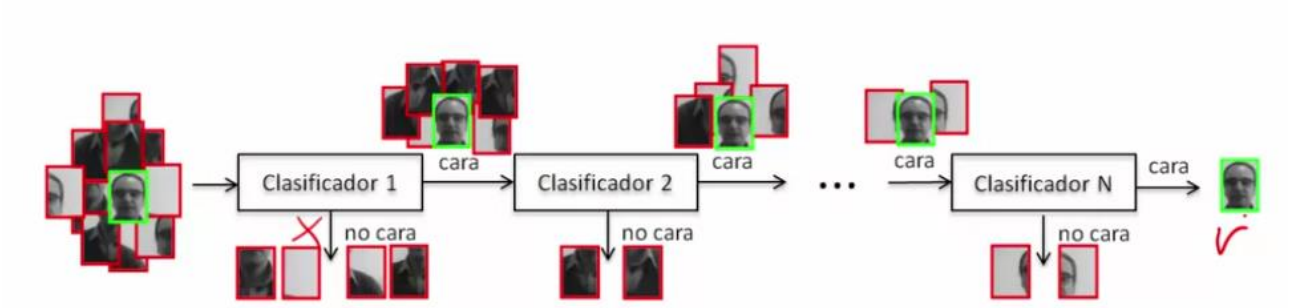


Figura 13 - Ejemplo de un clasificador procesando rostros [2].

Entrenamiento de un clasificador: Para lograr un correcto funcionamiento con una elevada tasa de acierto, se debe entrenar el algoritmo, con el fin de pueda reconocer el/los objetos deseado.

Este aprendizaje se realiza usando un programa que viene junto con la librería de implementación. Primero, se deben recolectar un número elevado de imágenes, donde se encuentre el objeto de interés. Luego se buscan otras fotos donde no este lo buscado. Una vez que se tiene lo anterior, se invoca al programa, se le pasan las imágenes y también se le dice el número de clasificadores que se van a entrenar. Después de elevadas iteraciones, produce al finalizar su ejecución, un archivo con la extensión “xml”. En este se encuentra la configuración del clasificador, ya entrenado, listo para su uso.

El siguiente paso es probar el algoritmo con el código entrenado, con el fin de determinar la tasa de éxito del entrenamiento, en el caso de que el nivel de error en la detección sea elevado, se deben cambiar los parámetros pasados al Programa de entrenamiento y

volver a realizar el aprendizaje, hasta lograr un correcto entrenamiento de la red de clasificadores.

- Redes Neuronales:

Las redes neuronales son un modelo computacional, el cual se basa en su homologo biológico. Está compuesta por un conjunto de unidades, llamadas neuronas, las cuales se interconectan entre si e intercambian información. La señal de entrada es procesada por este conjunto, la cual es sometida a unas series de operaciones matemáticas, obteniendo en su salida un resultado.

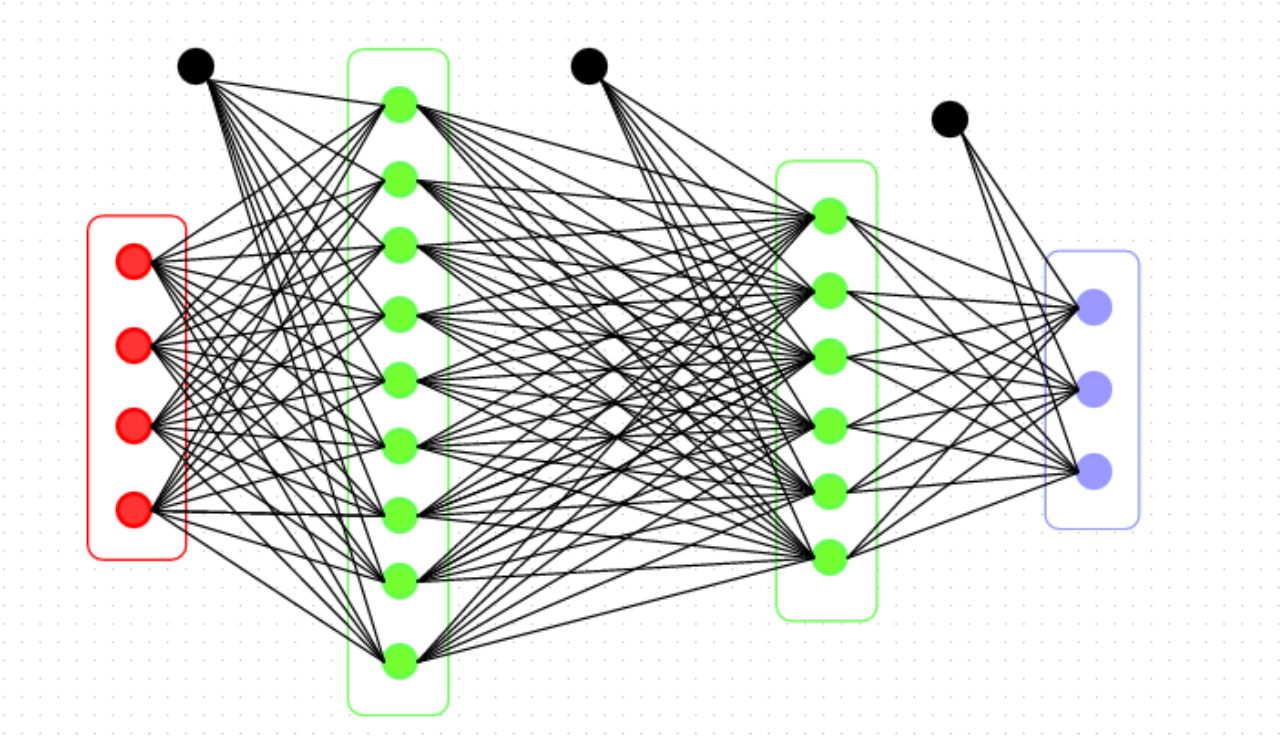


Figura 14 - Ilustración de una red neuronal, en rojo se observa la entrada, en verde la capa oculta y en azul la salida [14].

En el caso del procesamiento de imágenes, la forma más usada son las redes neuronales convolucionales (también denominadas CNN o ConvNet). Su funcionamiento consta de una capa que recibe y procesa la entrada (tomando como referencia una imagen en escala de grises, con 256 niveles u 8 bit de color, y de resolución 860 x 480). Este bloque de neuronas tiene la forma de una ventana (por ejemplo 24 de alto x 24 de ancho), la cual recorre la imagen. La red neuronal recibe la información correspondiente a los pixeles dentro de la ventana y a su posición con respecto a la imagen. Su funcionamiento es análogo a una convolución.

Cada una de las neuronas de entrada recibe la información de un pixel. Si luego de ser procesado supera un cierto umbral de activación, la información de dicha neurona pasa a la capa siguiente. Esto se repite hasta llegar a la última capa, en la cual se obtiene el resultado del procesamiento.

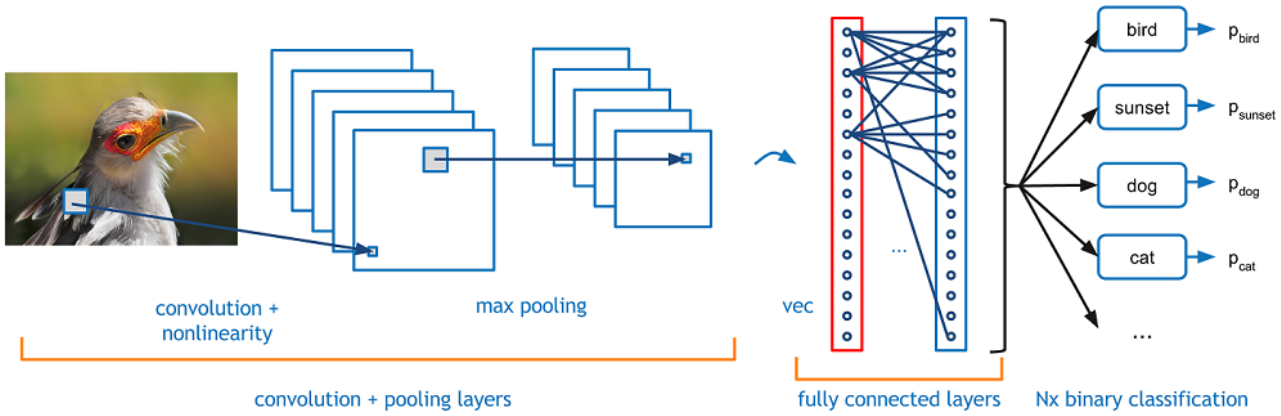


Figura 15 - Red neuronal convolucional, usada en la detección de animales [14].

La ventaja que posee frente a los algoritmos anteriores es que aprenden y se forman a si mismos, en vez de ser programados explícitamente. Su uso es destacable, en lugares donde los algoritmos convencionales, no logran una solución o una característica buscada.

Para su entrenamiento se requiere una gran cantidad de información, con la cual el sistema se nutre, hasta lograr una eficiencia elevada.

Existen librerías que ya contienen redes preentrenadas para la detección de objetos en las imágenes. Un ejemplo puede ser YOLO (You only look once), el cual viene en diferentes versiones dependiendo del nivel de optimización y el porcentaje de éxito. Lo anterior repercute directamente en la cantidad de cálculos requeridos para obtener una respuesta.

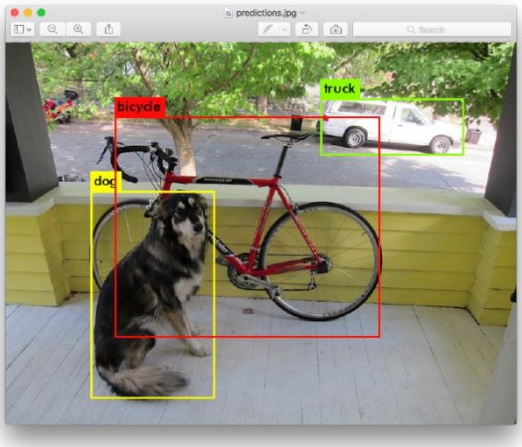


Figura 16 - Ejemplo de imagen procesada por YOLO [15].

2.1.4 Análisis y selección de las funciones para la estimación de velocidad.

Para la estimación de velocidad se analizó un algoritmo de fluido óptico. Esta técnica consiste en el seguimiento de un objeto dentro de una secuencia de imágenes.

Para lograr esto se estudiaron dos algoritmos, uno basado en Lucas-Kanade, y el otro basado en Gunnar Farneback.

El primero se basa en el seguimiento de puntos de interés, los cuales se pasan a la función y esta calcula su desplazamiento, si los hubo.
Un requisito para usar el algoritmo es que la imagen se tiene que pasar a escala de grises.

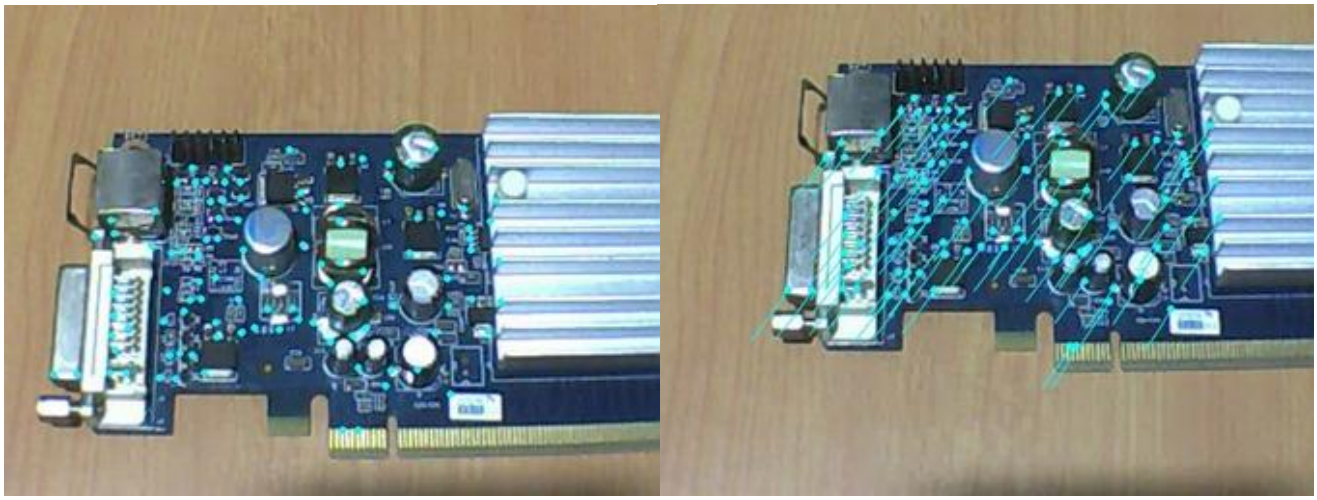


Figura 17 - Ejemplo de implementación del algoritmo de Lucas-Kanade [16]

El segundo a diferencia del anterior calcula el desplazamiento de todos los puntos de la imagen, esto implica mayor consumo de procesador, dado que nos devuelve una matriz donde se encuentran los vectores de los puntos desplazados.

Figura 18 - Ejemplo de implementación del algoritmo de Gunnar Franebeack [17]

2.1.5 Pruebas, experiencias.

Se realizaron pruebas utilizando un video proporcionado por el túnel y una PC de propósito general. En un inicio se comenzó a procesar las imágenes del video, transformándola en escala de grises y extrayéndoles el fondo, dado que el consumo de memoria y procesador disminuía al usar frame con estas características.

Después se comenzó a realizar los primeros filtrados de la imagen. Dado el efecto del ángulo de la cámara y el rebote de la luz de los vehículos que circulaban en dirección frontal a la misma, se buscaron combinaciones de estos, con el fin de minimizar dichas aberraciones. Una vez que se logró una mejora en la imagen, se pasó a probar umbrales, con el fin de binarizarla.

Cuando obtuvimos áreas blancas que representaban los vehículos, empezamos a probar algoritmos de detección.

El primero fue usando redes neuronales. Basándonos en ejemplos de uso, estas daban respuestas muy acertadas, el único problema que nos detuvo en su uso fue que el consumo de procesador era elevado, si se usaba una placa de video para el calculo de la misma, mejoraba su rendimiento, pero esto nos alejaba de la meta propuesta en esta tesis.

En segundo lugar, empezamos a probar algoritmos basados en clasificadores para la detección. Su entrenamiento era muy laborioso, dado que había que indicar imagen por imagen donde estaban los objetos de interés. Luego se tenía que entrenar la cascada de clasificadores y probarlos, evaluando el porcentaje de éxito. En nuestro caso era bajo, ya

que la calidad de las imágenes usadas del túnel no eran lo sufrientemente buenas para el entrenamiento.

Después probamos algoritmos de blob, el cual busca grupos de pixeles de iguales características. Las pruebas que hicimos no nos dieron buenos resultados, ya que la función requiere que se le pase información del objeto a detectar, como, por ejemplo, convexidad, umbral, área, tipo de figura geométrica, las cuales eran difícil de pasar dada la deformación del vehículo después del filtrado.

Por último, usamos la función de detención de contornos. Esta no es tan precisa como las primeras, pero con una buena calibración de las áreas a detectar, nos daba la información que necesitábamos. A demás nos proporcionaba las coordenadas del objeto detectado, las cuales nos sirven para realizar el seguimiento.

Finalmente acoplamos al código las funciones de seguimiento y cálculo de velocidad. Esta se estima a través de los frame que tarda un vehículo al cruzar por unas marcas colocadas por software.

Luego de tener el código funcional en la PC, se empezó a portar el código en plataformas de bajo recursos. Comenzando por la BeagleBone, la cual tiene un CPU ARM de 1Ghz con 512MB de RAM. Al instalar las librerías requeridas se tuvo problemas con el tiempo de compilado de la misma, para llegar a la optimización requerida.

Dado que la plataforma anterior no pudo correr el código sin latencia, se optó por un sistema de mayor poder de cálculo, la Raspberry Pi 3, esta posee un CPU ARM con 4 núcleos, de 1.2Ghz cada uno, con 1 GB de RAM. En este caso se pudo compilar sin problemas la librería optimizada para el trabajo en esa arquitectura.

El código quedo funcional en la última placa, procesando un video filmado en la Avenida Almafuerte de la ciudad de Paraná.

2.1.6 Problemas y soluciones implementadas

El primer problema que se presentó al correr los algoritmos fue que la ubicación de la cámara en el túnel no era la óptima, ya que las luces de los vehículos que circulaban en sentido frontal a la misma producían una deformación del objeto pos-filtrado, aumentando su área y segmentándola. Si aumentábamos los umbrales de filtrado, estas aberraciones disminuían, pero a su vez dificultaba la detección de vehículos de menor porte, como ser los moto-vehículos.

A demás la nitidez de la cámara fue un factor que dificulto aún más el procesado y la detección de objetos (ya que no se podía distinguir una rueda de la sombra de un vehículo), además las lentes se encuentran cubiertas de polvo y humo. Sumado a esto si teníamos dos vehículos muy próximos entre sí, producían una deformación de nuestro objeto de interés haciendo que esto se interprete como un vehículo de gran porte.

Se logro disminuir el efecto de estos problemas, agregando e intercalando filtros, umbrales en serie y usando funciones morfológicas, para maximizar el área de interés y minimizar el ruido.

El tiempo demandado en la calibración de los filtros, además el agregado de operaciones morfológicas con su respectiva configuración fue una etapa la cual se repitió reiteradas veces, ya que al necesitar una modificación en la formación de áreas para poder distinguir los vehículos o resaltar alguna característica, requería comenzar el calibrado nuevamente.

El filtrado adicional que requería el video usado, produjo un incremento considerable en el consumo del CPU, esto repercutió directamente en el momento de portar el código al sistema embebido. La solución que se encontró fue probar con otro video, el cual fue filmado con un mejor ángulo e iluminación. Esto repercutió directamente en el filtrado, disminuyendo su exigencia. Logrando cumplir con la meta propuesta.

Otro problema es que los vehículos de mayor porte que circulaban en sentido de orientación de la cámara obstaculizaban el carril contrario, haciendo imposible la detección y medición de parámetros de tráfico en dicho carril. Para este problema no se encontró una solución ya que la posición de las cámaras dentro del túnel no puede ser modificada.

Figura 19 - Camión obstruyendo completamente el carril izquierdo

2.1.7 Análisis del funcionamiento

Para poder ejecutar el código, se debe instalar antes de comenzar, la librería que nos da soporte a la manipulación de imágenes, detección, seguimiento, etc.

La PC usada tiene instalado el Sistema Operativo Ubuntu 18.04.
Para poder compilar las funciones que nos dan soporte a nuestro software, se deben tener instalados los siguientes paquetes:

- cmake gcc g++ git libjpeg-dev libpng-dev libtiff5-dev libavcodec-dev libavformat-dev libswscale-dev pkg-config libgtk2.0-dev libopenblas-dev libatlas-base-dev liblapack-dev libeigen3-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev sphinx-common libtbb-dev yasm libopencore-amrnb-dev libopencore-amrwb-dev libopenexr-dev libgstreamer-plugins-base1.0-dev libavcodec-dev libavutil-dev libavfilter-dev libavformat-dev libavresample-dev ffmpeg

Luego de tener listo lo anterior, se deben descargar la librería con las funciones extras:

- Descargar opencv: https://github.com/opencv/opencv/archive/3.4.1.zip
- Descargar contrib: https://github.com/opencv/opencv_contrib/archive/3.4.1.zip

Una vez finalizado lo anterior, se procede a la compilación de la librería.

Dentro de la carpeta de opencv, se crea una carpeta “release”, y dentro se ejecuta el comando cmake, el cual crea la estructura y secuencia de compilación dependiendo de nuestro hardware. Luego de lo anterior ya estamos en condiciones de compilar todo.

- make -j4 && sudo make install

A partir de este punto ya tenemos instalada la librería funcional en el sistema.
Procedemos a explicar el código implementado.

Adquisición de las imágenes:

La imagen se obtiene de una fuente de video, la cual puede provenir de una grabación o de una placa digitalizadora indiferentemente.

Para esto se usó la función open proporcionada por la librería de opencv. Esta nos devuelve un puntero al área de memoria a leer.

file.open(“dirección del video/ruta dispositivo”);

Para la extracción de frame, se usó la función read, también proporcionada por la librería anterior. Cada vez que se invoca, esta nos entrega un nuevo fotograma.

file.read(Mat &frame);

Filtrado:

La imagen extraída(frame), viene a color, con una resolución preestablecida. Para mejorar el rendimiento en el procesado y en algunos casos, por exigencia de las funciones implementadas, se tiene que pasar a escala de grises, generalmente 8 bit por pixel (256

tonalidades). A demás, para minimizar el ruido en la imagen y resaltar la información de interés, como primer paso se optó por la extracción de fondo.

Para esto hicimos uso de dos funciones:

cvtColor (InputArray src, OutputArray dst, int code, int dstCn =0);

Con esta función convertimos la imagen a escala de grises, enviando como parámetros la imagen de origen. Luego en outputarray obtenemos la salida convertida y en int code colocamos un parámetro que corresponde a la conversión de RGB a escala de grises (CV_BGR2GRAY). Este parámetro internamente hace una especie de promedio, ponderando cada color, con lo cual, deja en limpio la intensidad de dicho punto. Este se termina colocando en la ubicación correspondiente al dato procesado.

absdiff(InputArray src1, InputArray src2, OutputArray dst)

Se sustrae el fondo mediante el cálculo de la diferencia absoluta entre dos matrices (en este caso entre dos frame).

Figura 20 - Imagen en color

Figura 21 - Imagen convertida a escala de grises

Figura 22 - Imagen luego de una sustracción de fondo utilizando absdiff.

Luego de esto se aplica un filtro blur para eliminar posibles ruidos en la imagen y se umbraliza para binarizar la misma.

Paso siguiente se le aplica morfología de cerrado para minimizar los huecos que quedan dentro de las áreas que producen los objetos.

Como las funciones anteriores son muy abrasivas con los objetos de interés, se aplica un desenfoque gaussiano y otro umbralizado para aumentar el área de interés del objeto tratado.

Figura 23 - Imagen luego del desenfoque gaussiano y el umbralizado

Segmentación:

Fillpoly- Con esta función se dibuja un polígono que contornea el área que no es de interés, dejando el resto visible para ser procesado. Para esto se le pasa un vector de puntos, los cuales son los vértices del polígono. Esta función simplifica notablemente el procesamiento eliminando partes de la imagen que no son necesarias.

fillPoly(Mat& img, Point**pts, const int* npts, int ncontours, const Scalar& color, int line Type=8, int shift=0, Point offset=Point())

Figura 24 - Se observa el funcionamiento de fillPoly, función utilizada para separar los carriles.

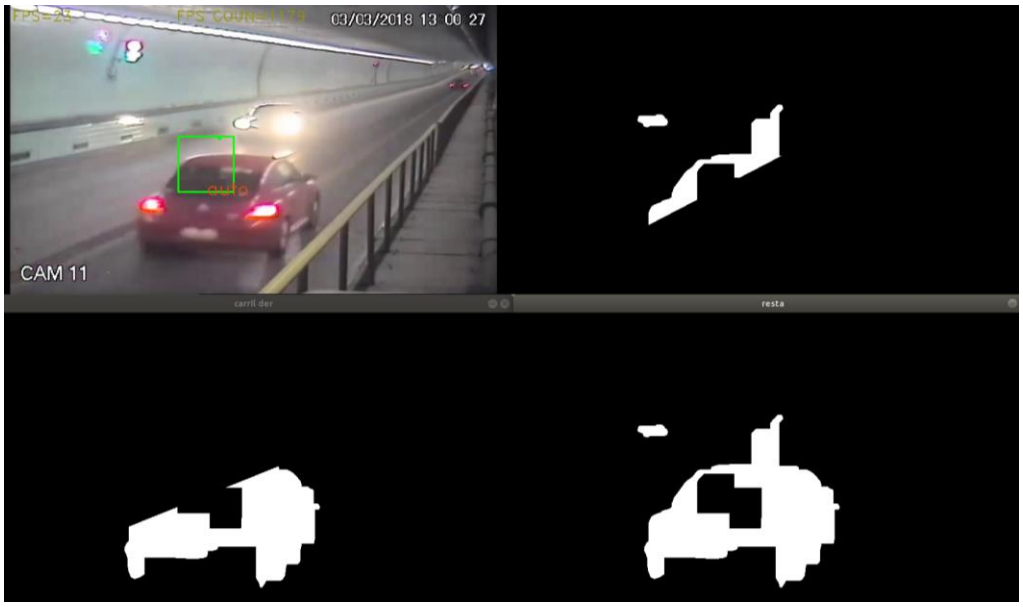


Figura 25 - Imagen procesada y como se ve cuando se extrae el carril no usado.

- Detección del Objeto:

Para la detección de los vehículos fue utilizada la detección de contornos como la manera más eficiente de optimizar los recursos de hardware. A través de la siguiente función se buscan todos los contornos en la imagen pasada como argumento. Esta los devuelve a través de un vector, el cual es procesado por otra función que nos devuelve su posición y área.

findCountours() – Es una función que permite encontrar los n contornos externos e internos de una imagen binaria. Con esta función obtenemos el contorno de los vehículos.

Una vez que se obtiene el área, esta se compara con una tabla y se verifica si corresponde a un camión, auto o moto.

En el caso que las áreas detectadas no correspondan a ninguno de los anteriores, se toma como no valida y se pide un nuevo frame.

Moments() – Esta función toma los parámetros que nos devuelve la función anterior y obtiene el área y centro de masa del objeto detectado.

En el caso de ser afirmativa la búsqueda, se pasa a la siguiente etapa. En esta se le pasa el área detectada, coordenadas e identificación del vehículo.

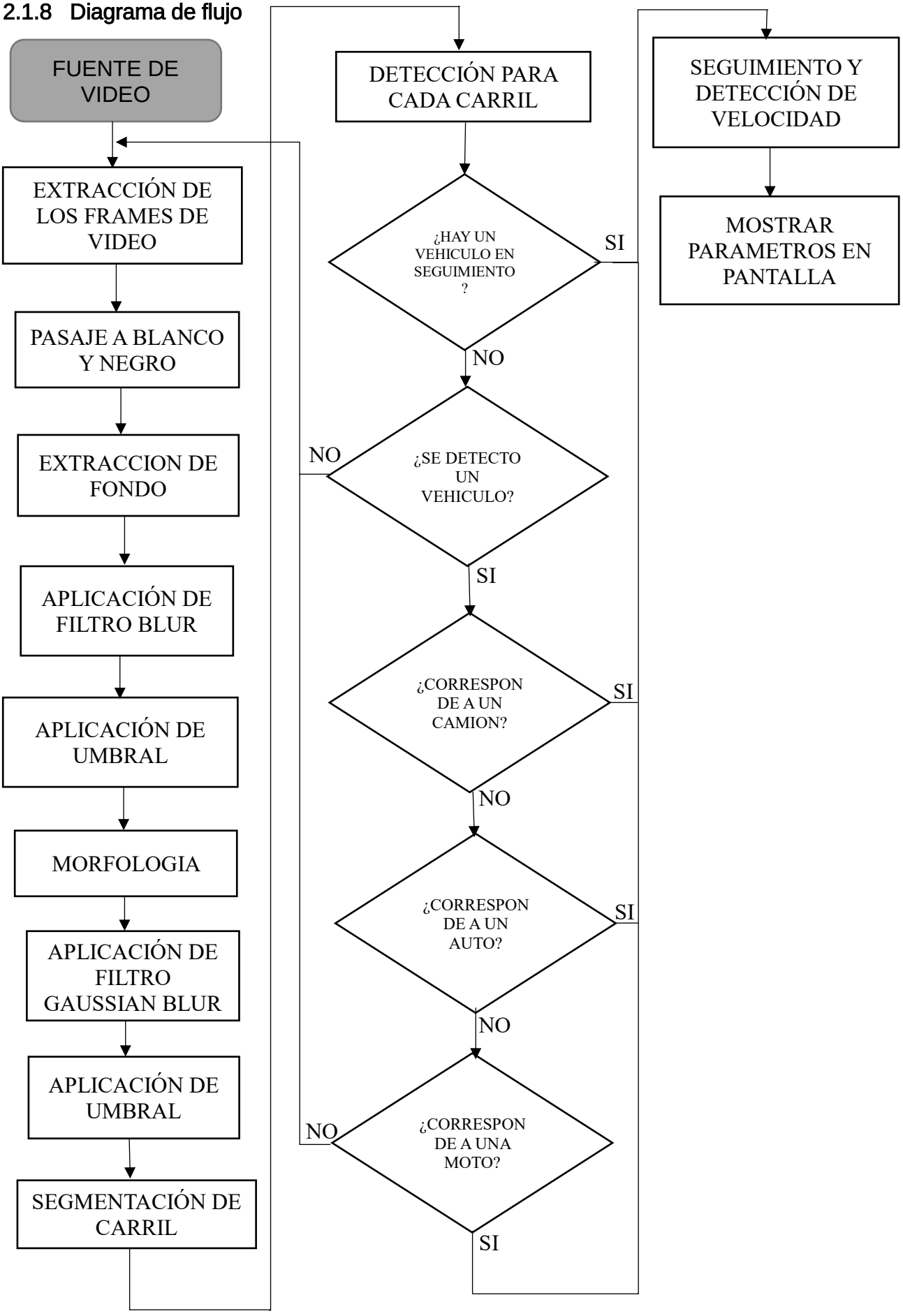
Estimación de velocidad:
Como último paso, se utilizó un algoritmo de fluido óptico basado en la función de Lucas-Kanade, el cual era el que mejor cumplía con los requisitos de tiempo en el sistema embebido.

Este es implementado por la función “cv::calcOpticalFlowLK()” a la cual se le pasa la imagen actual y la anterior en escala de grises, y se le proporciona una serie de puntos para los cuales se intentara calcular el flujo. La función devuelve las nuevas posiciones de dichos puntos junto con un código que indica si ha sido posible detectar el flujo (1 en el caso de éxito o 0 en caso contrario).

Usando este seguimiento, a través de un área previamente medida, se cuentan los frames que tarda el vehículo en recorrerla. Esto nos proporciona el tiempo en recorrer esta distancia y, a través de una simple formula se calcula la velocidad.

En el caso del túnel se usó como parámetro de referencia las rejillas de ventilación, las cuales tienen una separación de aproximadamente 3 [m], y en la avenida Almafuerte se usaron puntos de referencias de 1[m] sobre el asfalto.

2.1.8 Diagrama de flujo



2.2 Diseño Completo

2.2.1 Diseño Final

Luego de realizar todas las pruebas en PC y trasladar las mismas a un hardware de bajos recursos se observó que los resultados no eran favorables, por lo que se optó por probar otras fuentes de video en mejores condiciones obteniendo excelentes resultados. Se adapto el algoritmo para un óptimo funcionamiento en dicho hardware, y se testeo su funcionamiento.

Figura 26 - Hardware de bajos recursos (Raspberry Pi 3)

Las pruebas fueron realizadas en una RaspBerry Pi 3 cuyas características de hardware superaban a otras placas. Se agrego un disipador de calor sobre el microprocesador para que este no sufra un excesivo calentamiento y mejorar así el procesamiento de los datos.

Figura 27 - Disipador de calor sobre el procesador de la RaspBerry Pi 3

Se decidió mostrar los parámetros medidos en pantalla, y para esto se le anexo a la RaspBerry Pi un display lcd con una resolución de 800x480, la cual va conectada mediante la interfaz hdmi.

Figura 28 - Display LCD con resolución de 800 x 480 pixeles

Se utilizo la distro RaspBian Jessie (basado en Debian) como sistema operativo en la RaspBerry Pi 3, y la misma fue instalada en un pendrive sandisk de 16GB. Esto mejoro notablemente la fluidez del SO y del algoritmo implementado, ya que la velocidad del pendrive es superior al de la tarjeta microSD donde se monta dicho SO.

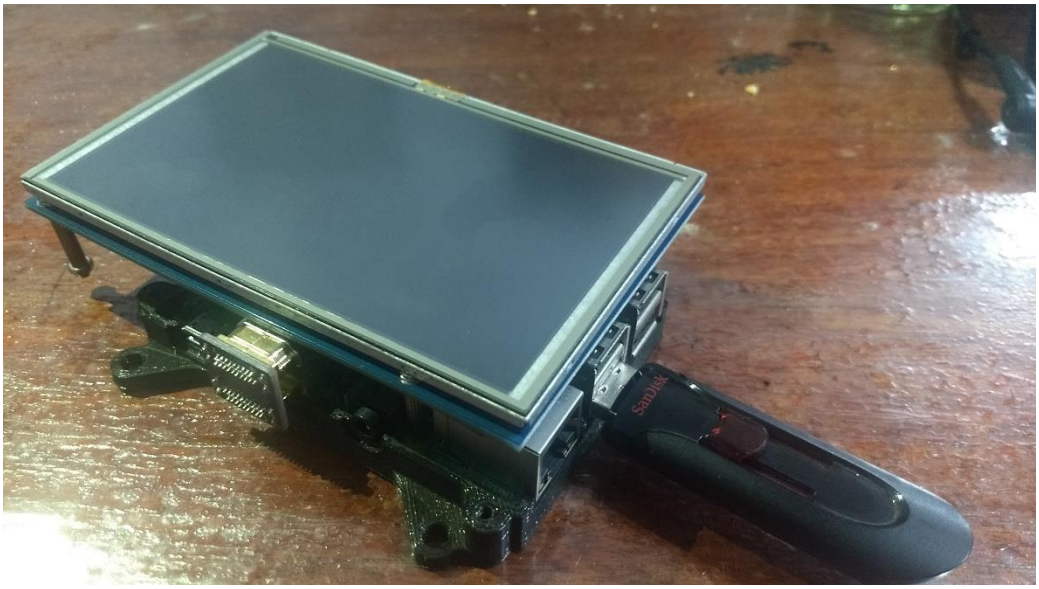


Figura 29 - Display LCD conectado a la Raspberry Pi 3 mediante la interfaz HDMI

Figura 30 - SO Raspbian Jessie iniciando en la RaspBerry Pi 3.

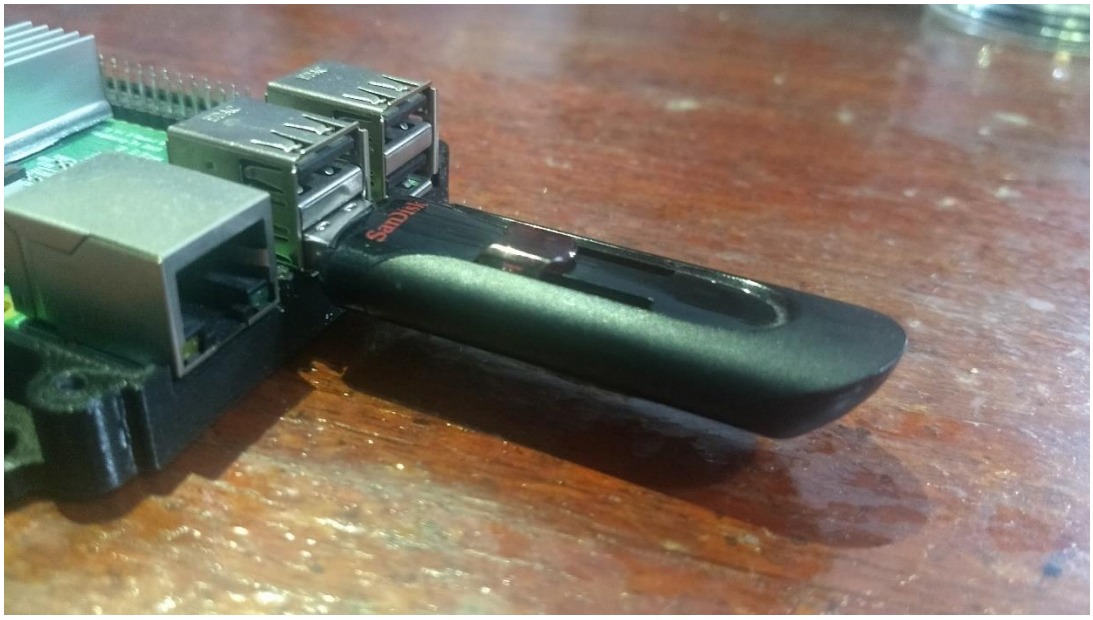


Figura 31 - Pen drive SanDisk de 16GB utilizado para albergar el Sistema Operativo RaspBian Jessie

- Calculo de error en conteo

Se realizaron cálculos de error en el conteo de vehículos en un carril tomando como total de la muestra el promedio obtenido del conteo manual. El porcentaje de error se calculo utilizando la siguiente expresión:

$$\frac{|C_M - C_A|}{C_M} \times 100\%$$

Donde:

CM = Conteo manual

CA = Conteo automatico

Conteo:

Fuente de video	Manual	Automático	Error (%)
Calle Almafuerte	58	52	10,3
Túnel Subfluvial	72	60	16,66

Tabla 1 - Porcentaje de error por video en el conteo.

2.2.2 Prestaciones

El algoritmo desarrollado tiene la particularidad que puede ser fácilmente portado a distintas plataformas de hardware manteniendo iguales parámetros y características de funcionamiento. Esto se debe a que fue optimizado para obtener el menor tiempo de procesamiento posible.

Para una mayor optimización y practicidad a la hora de visualizar los parámetros de tráfico se decidió mostrar los mismos por pantalla. De todas maneras, se puede tener acceso a un log, donde se puede visualizar y tener un control sobre los mismos de forma más detallada.

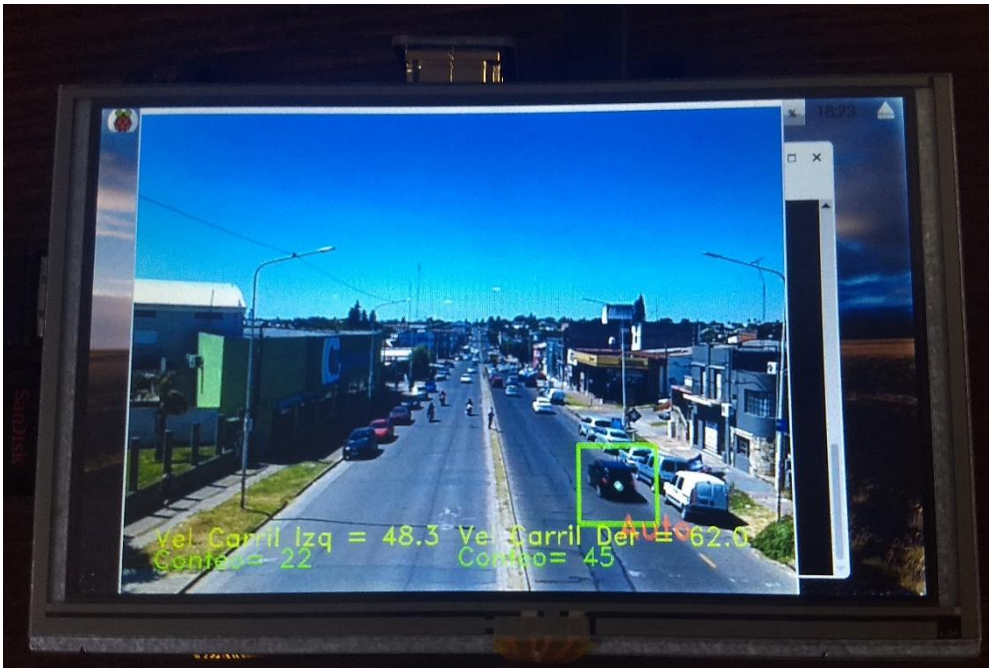


Figura 32 - Imagen procesando tráfico y mostrando datos por pantalla.

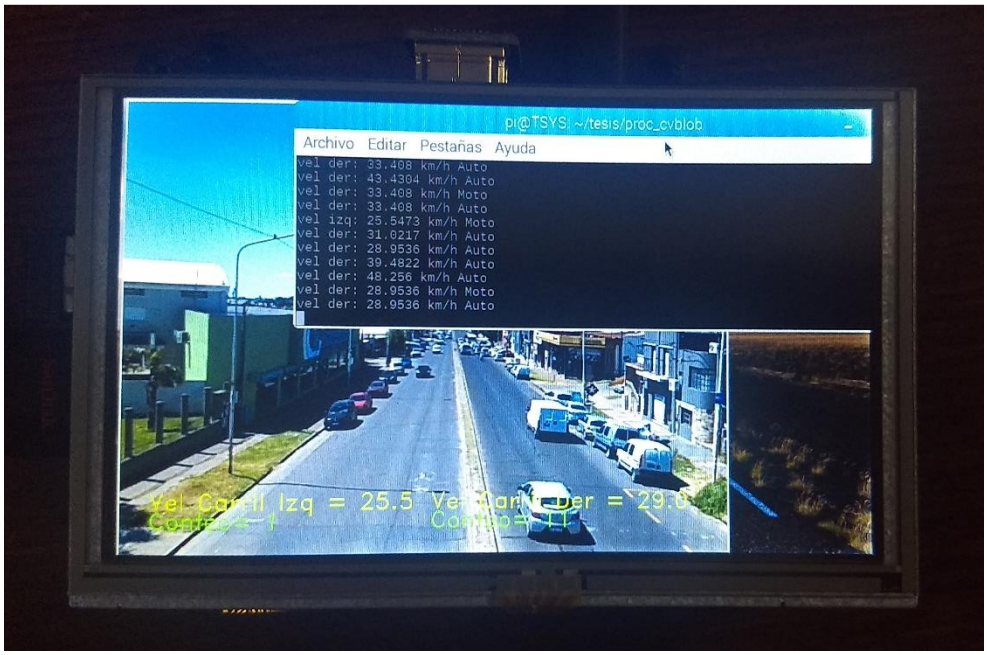


Figura 33 - Imagen visualizando log de velocidad en ambos carriles.

2.2.3 Fotos del hardware y capturas de pantalla del software.

Figura 34 - Hardware ejecutando algoritmo de estimación de parámetros de tráfico.

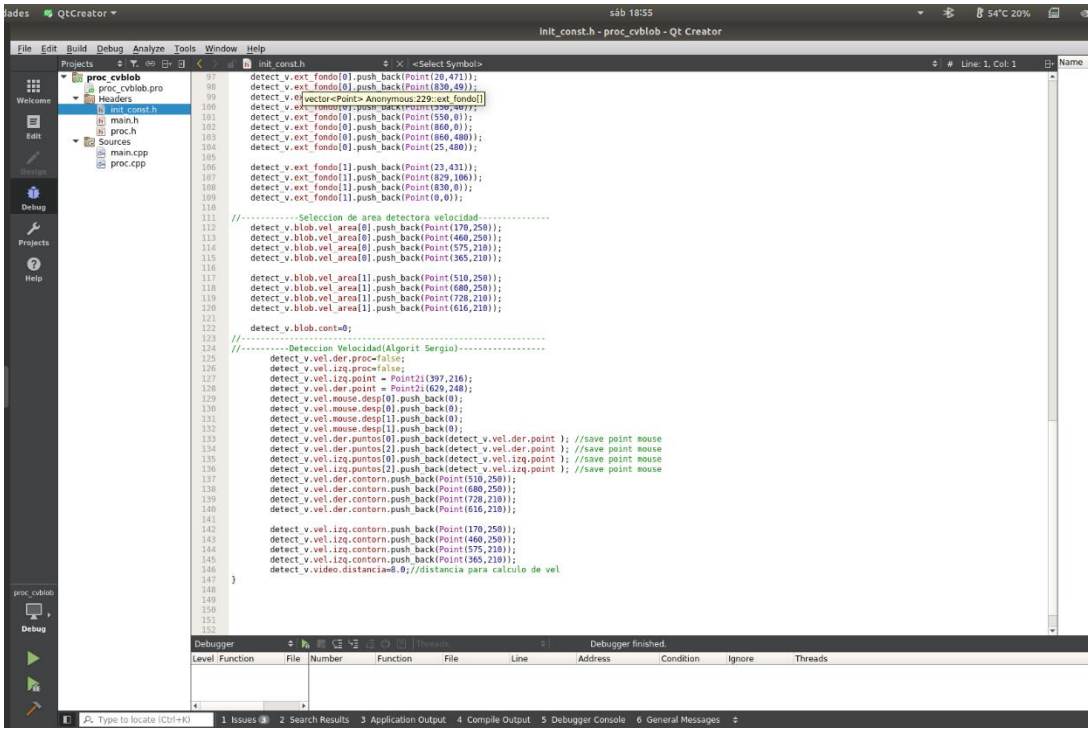


Figura 35 - Estructuras de configuración del algoritmo

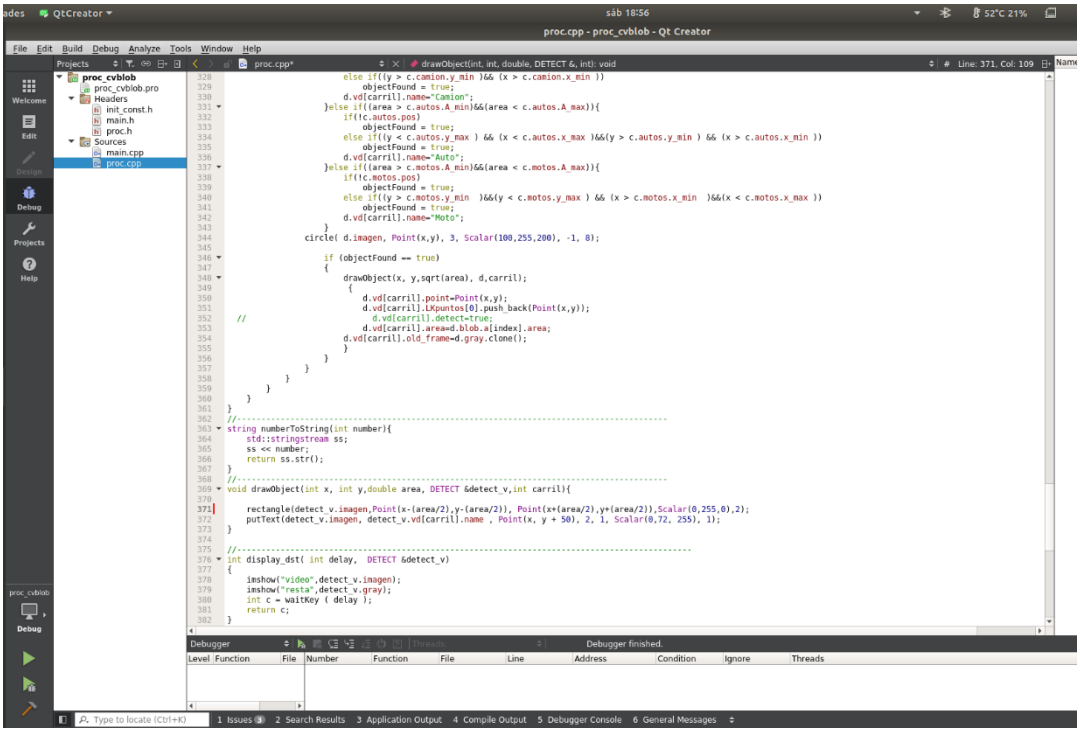


Figura 36 - Parte del código del conteo vehicular

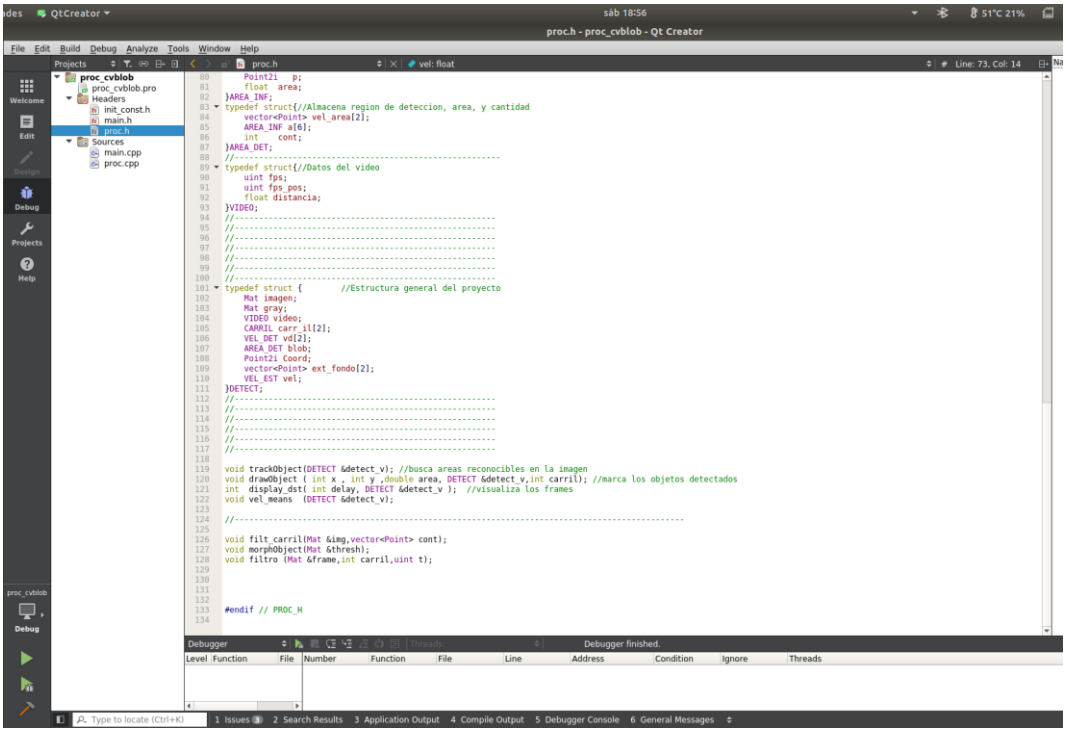


Figura 37 - Estructura de variables utilizadas en el algoritmo

Figura 38 - Imagen procesando el tráfico en el Túnel.

Capítulo 3: Resultados

En términos generales se puede decir que se obtuvieron resultados satisfactorios, tanto con la RaspBerry Pi 3(en condiciones más favorables que las del túnel), como en una PC de propósito general. Toda la información de los parámetros de tráfico es mostrada por pantalla para una fácil lectura de estos.

Se analizaron los resultados obtenidos en distintos fragmentos de video y en distintas horas del día encontrándose falsos positivos en el conteo vehicular. Esto se debe al solapamiento de objetos producto de la sombra de este y el ángulo de la cámara. A raíz de lo anterior se observó que el algoritmo procesa óptimamente los videos cuando el ángulo de la cámara es de 90 grados sobre la calle.

El algoritmo se desarrolló buscando la optimización del tiempo de ejecución, por lo tanto, se buscó la simplificación de los procesos con el fin de obtener los mejores tiempos de procesamiento.

Se pudo lograr el objetivo de estimar la velocidad de los vehículos dentro del túnel a pesar de la mala posición de la cámara. Esto solo fue viable en una PC de propósito general ya que la misma puede procesar los frame más rápido que la RaspBerry Pi 3. De esta manera se puede garantizar el procesamiento de los frame en tiempo real.

Capítulo 4: Análisis de Costos

Dado que se necesita un hardware mínimo para la realización de este prototipo se incluirá en el análisis el costo de una PC de propósito general. Una alternativa más económica sería la de utilizar una Raspberry Pi 3, pero nos encontraríamos con la limitación de que no sería apto para ser implementado en el túnel subfluvial. De todas maneras, será incluido en el análisis de costos.

A demás serán contempladas las horas dedicadas a la investigación y desarrollo del software.

Tabla 2 - Listado de Hardware utilizado:

Hardware	Precio
Motherboard MSI A68hm-e33 FM2+	$2300
CPU AMD A6 X2 7400k 3,9Ghz	$2189
Memoria RAM DDR3 Kingston 4GB	$2049
Gabinete rackeable Norco ATX	$1900
HDD 1TB sata 3 WD Blue	$2067,01
Fuente ATX Sentey metalblade 750W	$2880, 95
RaspBerry Pi 3	$2389
Dislpay LCD 800 x 480	$3132
Total:	**$18906,96**

Tabla 3 - Horas de trabajo:

Horas dedicadas	Precio Hora (pesos)	Precio
1900	100	$190000

Este proyecto va dirigido a entes de control de transito privados y públicos, organismos encargados de la medición y recolección de datos de tráfico, etc. Actualmente existen empresas que comercializan productos basados en técnicas de visión artificial, para el análisis de diferentes patrones y comportamiento de objetos, como lectores de patente, detectores de vehículos en banquina, seguimiento de tráfico, reconocimiento de datos biométricos, etc.

La ventaja que tienen este tipo de sistemas es que puede ser utilizado en base a instalaciones de monitoreo de video ya existentes, evitando tener que agregar un sistema adicional para la determinación de los parámetros antes mencionados, lo que hace que sea un producto muy rentable y de fácil amortización.

Capítulo 5: Discusión y Conclusión.

Para el realizado de las pruebas, se usaron los videos aportados por el Túnel Subfluvial, además se contó con videos filmados en la Avenida Almafuerte. Esto nos ayudó a encontrar diversos resultados, de los cuales en algunos casos se ha conseguido contajes muy aproximados a la realidad, y realizando clasificaciones bastante acertadas.

De otros, la dificultad del video (como es el caso del túnel subfluvial) ha hecho muy compleja esta tarea y se ha quedado lejos del propósito inicial utilizando un hardware de bajos recursos.

A pesar de esto existen soluciones para hardware de bajos recursos que se basan en redes neuronales autocontenidas, como es el caso del Intel Neuronal Compute Stick 2, que hace realizable este proyecto (en una RaspBerry Pi) con bajas probabilidades de error. Esto puede ser planteado como una mejora a lo tratado en este proyecto.

De todas maneras, utilizando una PC de propósito general se ha logrado, mediante la aplicación de diversos filtros, un funcionamiento aceptable del algoritmo utilizando las filmaciones del túnel.

La mayor dificultad que se ha encontrado en el desarrollo posiblemente sea la no estandarización de los videos, es decir, el hecho que el proyecto se tenga que amoldar a cualquier cámara de tráfico, y sobre todo a cualquier hora del día. Este problema se solucionaría utilizando algoritmos más eficientes, como lo son las redes neuronales. Esto conlleva a utilizar un hardware más potente o dedicado al cálculo vectorial, como pueden ser las placas de video con soporte CUDA.

Un gran número de empresas a nivel mundial se ha dedicado a proveer soluciones integrales a problemas de la industria utilizando como herramienta fundamental la visión artificial, esto la ha convertido en un área de gran desarrollo. Se han implementado ampliamente sistemas enfocados al estudio y caracterización del tráfico.

Desde el punto de vista tecnológico, es un sector cuyo desarrollo está íntimamente ligado a la informática y a la capacidad computacional que existe en un momento determinado. Debido a esta característica, es muy probable que aun quede por explotar el mayor potencial que puedan alcanzar sus tecnologías.

Desde el punto de vista cultural, todavía existe un claro desconocimiento, por parte de los usuarios finales, de las capacidades y limitaciones de un sistema de visión artificial. Esto genera cierta desconfianza a la hora de optar por una tecnología de este sector frente a sus productos sustitutivos.

Para finalizar hay que tener en cuenta que, aunque se han implantado muchos sistemas basados en visión artificial con resultados positivos, hay muchos escenarios en los que esta tecnología no está a la altura de las circunstancias, lo que puede suponer un riesgo si las tecnologías alternativas tienen una evolución más rápida.

Como un agregado, tomando los consejos del tribunal evaluador se deja como propuesta de anexo a futuro la detección de vehículos detenidos, para informar al operario que se produjo un accidente o una falla.

A demás la posibilidad de conectar una alarma visual y/o sonora con el fin de alertar a la central de control que un vehículo excedió los limites superiores o inferiores de velocidad permitidos.

Capítulo 6: Literatura Citada.

[1] https://docs.opencv.org/3.4/d0/de3/citelist.html#CITEREF_Suzuki85

[2] https://es.coursera.org/lecture/deteccion-objetos/l5-5-cascada-de-clasificadores-pRnHu

[3]https://es.wikipedia.org/wiki/Archivo:Ventanas_de_clasificaci%C3%B3n_en_una_imagen.png

[4] http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1815-59282012000200008

[5] https://stackoverflow.com/questions/32997224/draw-filled-shape-from-four-points

[6] https://stackoverflow.com/questions/21217506/robust-tracking-of-blobs

[7] http://www.maia.ub.es/~sergio/linked/vanessa07.pdf

[8] https://subscription.packtpub.com/book/application_development/9781785283932/2/ch02lvl1sec24/erosion-and-dilation

[9] https://www.behance.net/gallery/4057777/Vehicles-Detection-Tracking-and-Counting

[10]www.elai.upm.es/webantigua/spain/Asignaturas/Robotica/ApuntesVA/cap6VAProcMorf.pdf

[11] https://docs.opencv.org/3.4.1/index.html

[12] https://lumiguide.github.io/haskell-opencv/doc/OpenCV-ImgProc-Drawing.html

[13] https://www.learnopencv.com/blob-detection-using-opencv-python-c/

[14] https://medium.com/datos-y-ciencia/construye-tu-primer-clasificador-de-deep-learning-con-tensorflow-ejemplo-de-razas-de-perros-ed218bb4df89

[15] https://pjreddie.com/darknet/yolo/

[16] http://acodigo.blogspot.com/2017/07/flujo-optico-lucas-kanade-con-opencv.html

[17] http://acodigo.blogspot.com/2017/07/flujo-optico-gunnar-farneback.html

- https://www.tensorflow.org/api_docs/python/tf
- https://s3.amazonaws.com/academia.edu.documents/32708022/andrews_WVC2013.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1552357850&Signature=hvNIj5FaDyVLsMhnUuyauCvGslQ%3D&response-content-disposition=inline%3B%20filename%3DBGSLibrary_An_OpenCV_C_Background_Subtra.pdf
- Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library
- http://www.ijeee.net/uploadfile/2013/0702/20130702104409134.pdf