ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

PROYECTO DE GRADUACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTOR: ROSERO PRADO MILTON FABRICIO

TEMA: DESARROLLO DE UN SISTEMA PARA RECONOCIMIENTO DE VEHÍCULOS QUE CIRCULAN A EXCESO DE VELOCIDAD DENTRO DEL CAMPUS ESPE-SANGOLQUÍ MEDIANTE CAPTURA Y PROCESAMIENTO DE IMÁGENES EN BASE AL SISTEMA EMBEBIDO ODROID U3

DIRECTOR: ING. DANNY SOTOMAYOR

CODIRECTOR: ING. RAÚL HARO

SANGOLQUÍ, 2015

Certificado de tutoría

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICADO

Ing. Danny Sotomayor

Ing. Raúl Haro

CERTIFICAN

Que el trabajo titulado "DESARROLLO DE UN SISTEMA PARA RECONOCIMIENTO DE VEHÍCULOS QUE CIRCULAN A EXCESO DE VELOCIDAD DENTRO DEL CAMPUS ESPE-SANGOLQUÍ MEDIANTE CAPTURA Y PROCESAMIENTO DE IMÁGENES EN BASE AL SISTEMA EMBEBIDO ODROID U3", realizado por Rosero Prado Milton Fabricio, ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la Universidad de las Fuerzas Armadas - ESPE en su reglamento.

Debido a que se trata de un trabajo de investigación recomiendan su publicación.

Sangolquí, 29 de julio del 2015.

Ing. Danny Sotomayor

DIRECTOR

Ing. Raúl Haro

CODIRECTOR

Declaración de Responsabilidad

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

MILTON FABRICIO ROSERO PRADO

DECLARO QUE:

El proyecto de grado denominado "DESARROLLO DE UN SISTEMA PARA RECONOCIMIENTO DE VEHÍCULOS QUE CIRCULAN A EXCESO DE VELOCIDAD DENTRO DEL CAMPUS ESPE-SANGOLQUÍ MEDIANTE CAPTURA Y PROCESAMIENTO DE IMÁGENES EN BASE AL SISTEMA EMBEBIDO ODROID U3", ha sido desarrollado en base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie, de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Consecuentemente este trabajo es de mi autoría, en virtud de ello me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Sangolquí, 29 de julio del 2015.

Milton Fabricio Rosero Prado

Autorización de publicación

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

MILTON FABRICIO ROSERO PRADO

Autorizo a la Universidad de las Fuerzas Armadas - ESPE a publicar en la biblioteca virtual de la institución el presente trabajo "DESARROLLO DE UN SISTEMA PARA RECONOCIMIENTO DE VEHÍCULOS QUE CIRCULAN A EXCESO DE VELOCIDAD DENTRO DEL CAMPUS ESPE-SANGOLQUÍ MEDIANTE CAPTURA Y PROCESAMIENTO DE IMÁGENES EN BASE AL SISTEMA EMBEBIDO ODROID U3", cuyo contenido, ideas y criterios son de mi exclusiva autoría y responsabilidad.

Sangolquí, 29 de julio del 2015.

Milton Fabricio Rosero Prado

DEDICATORIA

Dedico esta tesis a todos quienes han formado parte de mi vida, aportando con sabiduría y experiencia a mi formación humana.

Milton Fabricio Rosero Prado

AGRADECIMIENTO

Agradezco a toda mi familia por su apoyo incondicional, por su ejemplo de perseverancia y de fuerza inquebrantable, por su preocupación constante y sus muestras de afecto desinteresado.

Agradezco a mis padres Milton y Mercy por haberme guiado en todo momento a través de caminos llenos de felicidad, armonía, sabiduría, comprensión, y compromiso.

A mis hermanos Pablo y Pame por su alegría, por su amistad, por su apoyo y su ejemplo de esfuerzo diario.

A todos los maestros que me han inspirado a ver más allá de las cosas y a descubrir las maravillas de la ciencia.

A mis amigos, que me han brindado su apoyo incondicional, con quienes hemos compartido experiencias inolvidables.

Milton Fabricio Rosero Prado

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En el presente proyecto se desarrolló un sistema para reconocimiento e identificación vehicular mediante procesamiento de imágenes en base al sistema embebido Odroid U3. Esta tarjeta embebida posee altas prestaciones de hardware que permiten realizar operaciones que requieren un elevado número de recursos como es el procesamiento de imágenes. El proceso general del sistema consta de tres etapas principales: detección del vehículo y determinación de su velocidad, identificación del vehículo mediante el reconocimiento de su número de placa, y el envío de la información obtenida hacia una base de datos en un servidor de una red local. Se desarrollaron algortimos con optimización de procesos y selección de regiones de interés, con el objetivo de obtener menores tiempos de procesamiento que los algoritmos existentes. Para ello se emplearon las librerías de *OpenCV*, y se utilizó *C++* como lenguaje de desarrollo. Se realizaron pruebas de funcionamiento para verificar su correcto desempeño. Con los datos obtenidos de las pruebas se calcularon los porcentajes de error para cada etapa y se midieron los tiempos de procesamiento requeridos. También se evaluaron los algoritmos en un computador de propósito general con el objetivo de comparar la eficiencia de procesamiento de la tarjeta embebida. Como resultados se obtuvo que la medición de velocidad presenta un error que está en el margen aceptable para equipos de medición de velocidad vehicular como son los fotoradares; la identificación del número de placa presenta un error aceptable tomando en cuenta las consideraciones del algoritmo desarrollado; y al analizar los tiempos de procesamiento requeridos por todas la etapas del sistema, se determinó que sí es factible realizar un procesamiento en tiempo real en la tarjeta embebida Odroid U3.

Palabras clave:

- PDI
- *ODROID*

- *OPENCV*
- VEHÍCULOS
- VELOCIDAD
- *APNR*

ABSTRACT

In this work, we developed a system for vehicle recognition and identification with image processing based on the embedded system ODROID U3. This embedded card has high performance hardware for performing operations that require a large number of resources such as image processing. The overall process of the system consists of three main stages: detection and determination of vehicle speed, vehicle identification by recognizing his plate number, and sending the information obtained to a database on a server in a local network. We developed algortimos with optimization processes and selection of regions of interest, in order to obtain shorter processing times than the existing algorithms. To do that, the *OpenCV* image processing libraries were used, and also *C++* as development language. Test runs were conducted to verify proper performance. With the data from the testing, error rates were calculated for each stage and the required processing times were measured. Algorithms were also evaluated on a general purpose computer in order to compare the efficiency of the embedded card processing time. As a result it was found that the speed measurement has an error that is in the acceptable range for vehicle speed measurement equipment, such as speed photo-radar; the number plate identification has an acceptable error considering the developed algorithm; and when the processing time required by all stages of the system was analized, we determined that it is feasible to perform a real-time processing in the embedded ODROID U3 card.

Key words:

- PDI
- *ODROID*
- *OPENCV*
- VEHICLES

- SPEED
- *APNR*

CAPÍTULO 1

INTRODUCCIÓN

Desde la antigüedad, la movilidad y el transporte han sido estructuras indispensables para el crecimiento y desarrollo de las civilizaciones. En los primeros sistemas de transporte terrestre se utilizaban animales para trasladar personas y mercancías a sitios lejanos. En las década de 1880 aparecieron los primeros automóviles, mas las personas se resistían a utilizar aquellos "carruajes sin caballos". Gracias a los rápidos avances técnicos y tecnológicos, en 1903 los automóviles estaban ya ampliamente difundidos, alcanzando velocidades mayores a los 110 Km/h; sin embargo, eran costosos y se averiaban constantemente. Desde entonces se han mejorado y abaratado debido a su producción en masa. Hoy son el medio de transporte cotidiano para millones de personas en todo el mundo, y por ello, obedecen a normas y leyes enfocadas hacia una adecuada circulación y prevención de accidentes de tránsito.

En enero del 2014 la Agencia Nacional de Tránsito registró que el exceso de velocidad es la segunda causa de muertes previsibles en accidentes de tránsito en el Ecuador [1], esto se corrobora con el informe de la Organización Mundial de la Salud, donde se muestra que el Ecuador es el segundo país en Sudamérica con el mayor índice de muertes por accidentes de tránsito. Sin embargo, la mayoría de conductores consideran que conducir a 120 Km/h es normal, ya que las mejoras aerodinámicas de los vehículos modernos proporcionan una mayor estabilidad. Pero lo que muchos

no reconocen es que a partir de 90 Km/h un vehículo es cada vez menos gobernable, aumentando así el peligro de sufrir un accidente.

Ante estas circunstancias han surgido diferentes técnicas de prevención y control vehicular para detectar infractores en señales de tránsito y exceso de velocidad. Entre los métodos empleados se encuentran: el detector por radar [2]; mediante sensor láser visible activo [3]; sensores magneto-resistivos instalados en la calzada [4]; cortinas fotoeléctricas a los costados de la carretera [5]; y mediante sistemas basados en CCTV [6]. Estos métodos de control se complementan con el proceso de identificación del infractor, acción necesaria para notificarle las advertencia o sanciones correspondientes. Esta actividad se la realiza en un proceso posterior donde los operadores inspeccionan las cámaras de seguridad.

Existen sistemas complementarios que se enfocan en la detección, conteo y clasificación de vehículos [7] [8] [9], mas no en infracciones de tránsito. Uno de ellos es el *Sistema de Lectura Automática de Matrículas (SLAM)* [10], que se basa en ANPR *(Automatic Number Plate Recognition)* [11]. Son varias las aplicaciones que un sistema ANPR puede brindar, entre ellas está el registro automático de vehículos a la entrada y salida de parqueaderos; acceso controlado para vehículos autorizados; el proveer información sobre el flujo vehicular; paso automático de vehículos por peajes; entre otros.

Muchas veces el proceso de detección de infracciones de tránsito queda inconclusa debido a que los sistemas actuales como radares y equipos láser son relativamente costosos (75000 dólares aproximadamente) por lo que sólo se tienen pocos equipos; a esto se suma la aparición de inhibidores de radares [12] que dificultan y hasta imposibilitan la detección de vehículos a exceso de velocidad. Además, no se cuenta con sistemas complementarios como APNR, y en vez de esto se emplea agentes de tránsito que se ven obligados a identificar visualmente al infractor: *"El equipo se coloca en un deter-*

minado lugar con dos operadores, quienes a su vez avisan, por medio de una radio, a otro grupo de policías que se colocan a un kilómetro más allá, sobre las características del carro infractor", según lo expresa Carlos Rivero, Jefe de Operaciones de Control Vehicular de Manabí, en una entrevista para el periódico El Diario.

Sistemas de detección vehicular mencionados anteriormente como la CORTINA FOTOELÉCTRICA, específicamente el modelo MAPS CF-324 [13] presentan un consumo de hasta 60W, requieren una tensión de alimentación de 220VAC, y tienen un peso de 50Kg/columna (100 Kg total), además su precio es relativamente alto y no posee sistema APNR. Por estas características resulta poco accesible para la mayoría de instituciones públicas o privadas, universidades, supermercados, parqueaderos y demás entidades.

Todos los métodos de detección de infracciones antes mencionados requieren sistemas complementarios que permitan identificar a los infractores, tarea realizada generalmente de manera no automatizada, es decir, se necesita personal adicional para realizar esta tarea. Además, es poco eficiente ya que la capacidad de concentración del ser humano disminuye en jornadas prolongadas, reflejándose en errores de registro y datos equivocados. Adicionalmente, se puede destacar que este tipo de control manual no se lo hace permanentemente, es decir, las 24 horas y 7 días a la semana; sino que se lo realiza en determinados días y durante ciertas horas, o en su lugar, con turnos rotativos que implican mayor número de personal.

Con lo antes mencionado, se puede concluir que un sistema automático de detección de infracciones de tránsito debe constar de tres partes fundamentales: 1) un sistema de detección de infracciones, 2) un sistema de identificación del infractor, y 3) un sistema de envío de esta información hacia una central. Con la información recibida del infractor se pueden entonces emitir las advertencias, multas o sanciones correspondientes.

En el presente proyecto se realizan estas tres tareas en un único sistema, mediante procesamiento digital de imágenes como alternativa a los métodos existentes, y que además será un sistema embebido, es decir, un circuito/placa electrónica de tamaño, peso y dimensiones reducidas diseñado para realizar una o algunas pocas funciones dedicadas [14], con bajo consumo de energía (procesadores hasta 2000 MHz, consumo de 0.5 mW/MHz), precio relativamente bajo, y que realiza tareas de procesamiento en tiempo real.

La función del sistema desarrollado es determinar la velocidad y el número de placa de los vehículos que circulan dentro del campus Espe-Sangolquí, e identificar aquellos con exceso de velocidad. Si un vehículo es considerado como infractor, su información es enviada hacia una base de datos en un servidor local.

Para el procesamiento y control general se decidió utilizar el sistema embebido *ODROID U3*, pues se requiere que el proyecto sea portable, replicable, de bajo costo y de bajo consumo de recursos. Las características que destacan al sistema elegido sobre otros sistemas similares como *Arduino*, *Raspberry Pi*, *BeagleBone Black*, entre otros, son su procesador *ARM Exynos4412 Prime Quad-Core 1.7Ghz* y su memoria RAM *2Gbyte LPDDR2 880Mega Data Rate*, características que hacen del *Odriod U3* un sistema apto para realizar procesamiento de imágenes requiriendo poco consumo de recursos gracias a la tecnología ARM [15] y LPDDR [16]. Utiliza para su funcionamiento hardware y software de código abierto, lo cual permite que el proyecto sea replicable, y sus dimensiones físicas 83 x 48 mm y 48g de peso le añaden portabilidad. Adicionalmente, posee módulos Ethernet y Wireless LAN que le proporcionan conectividad y comunicación remota.

Para la implementación del proyecto, inicialmente se analizaron las características del entorno de despliegue que permitieron determinar los siguientes parámetros: 1) ubicación física del sistema dentro del campus, área de cobertura, calidad y reso-

lución de la imagen; 2) las regiones de interés en la imagen para su segmentación y procesamiento; y 3) el tratamiento específico que se hará a cada región. Debido a que la capacidad de procesamiento en sistemas embebidos es limitada en comparación a equipos de uso general, estos parámetros son específicos para la aplicación propuesta con el objetivo de obtener un desempeño óptimo del sistema, pudiéndose adaptarlos a distintos escenarios.

De acuerdo a los parámetros determinados se desarrollaron los algoritmos de detección, reconocimiento e identificación de vehículos, y se los evaluaron en primera instancia empleando *Matlab*, ya que este software posee métodos que facilitan una rápida implementación de técnicas complejas en procesamiento de imágenes. Posteriormente, se implementaron los algoritmos en la tarjeta embebida utilizando software de desarrollo compatible con GNU/Linux: *OpenCV*, *Tesseract OCR* y *Qt-Creator*.

La información de velocidad, número de placa, fecha, hora y fotografía del auto infractor son almacenados en una base de datos de una red local, para lo cual se utilizó *MySQL* como aplicación de servidor que se encarga de gestionar la información enviada desde el terminal cliente *Odriod U3*.

Como resultados finales del proyecto se obtuvieron los aplicativos en *Opencv* y en *Qt-Creator*, completamente funcionales en la tarjeta embebida. El despliegue del sistema para el presente proyecto de tesis se realizó en un sitio estratégico con el objetivo de validar su correcto funcionamiento. Posteriormente, dependiendo de los requerimientos institucionales, se podrá realizar la planificación para desplegar varios puntos de control vehicular.

La finalidad de este proyecto es desarrollar un sistema para detección vehicular e identificación de placas que se ejecute desde un sistema embebido, y evaluar su desempeño en tiempo real. En base a ello, en el capítulo 2 se describe y analiza el estado del arte de conceptos que abarcan al sistema en general: técnicas de monitorización

e identificación vehicular, sistemas de reconocimiento automático de matrículas, y reconocimiento óptico de caracteres. En el capítulo 3 se describen los materiales y métodos de hardware y software utilizados, las características del sistema embebido *ODROID U3*, y de los programas empleados para el desarrollo del proyecto. El capítulo 4 describe el proceso general de funcionamiento del sistema propuesto, detallando los procedimientos y algoritmos implementados. El capítulo 5 muestra el análisis de los resultados obtenidos en las pruebas de funcionamiento; y finalmente en el capítulo 6 se detallan las conclusiones, recomendaciones y trabajos futuros.

CAPÍTULO 2

ESTUDIO DEL ESTADO DEL ARTE

2.1 Técnicas de Monitorización e Identificación Vehicular

La monitorización y los sistemas de identificación vehicular han surgido con la necesidad de prevenir accidente de tránsito, y también para optimizar el uso de infraestructuras relacionadas a los sistemas viales [17].

Con el tiempo se han creado técnicas utilizadas para la identificación vehicular [18], entre las cuales tenemos las siguientes:

- **Radio Frecuencia**: Es un sistema implementado en cada uno de los vehículos, éste tiene como puntos positivos el ser económico y flexible, y como punto negativo que debe encontrarse dentro del rango de cobertura de una antena para ser identificado.
- **Satelital**: La monitorización de vehículos se realiza a través de satélites, utilizando equipos de GPS. Estos sistemas brindan la posición exacta de un vehículo en particular pero con la desventaja de que se requieren antenas terrenas para poder realizar la transmisión de una posición, lo que hace que el sistema

sea costoso.

- **Óptica**: Esta técnica se basa en el reconocimiento de los caracteres alfanuméricos en las placas de autos, teniendo como desventaja que puede identificar un vehículo a la vez, y necesita un horizonte visual.

- **Láser**: Se basa en la lectura de códigos implementado en cada placa de un vehículo, considerando que si se desea aplicar esta técnica se requiere un rediseño de placas, que se adapten a los códigos de barras.

Actualmente la alternativa utilizada con mayor frecuencia para la monitorización vehicular es por medio de la placa [19], con la finalidad de contar con un método de localización de un automóvil en movimiento, buscando incrementar los índices de control de diferentes organizaciones.

Las placas de los vehículos se usan como registros para la circulación legal en todo el mundo, donde cada gobierno emite sus propias placas con características particulares y diseños propios, tomando en consideración que todas ellas tienen una forma rectangular, con sus esquinas redondeadas y deben incluir el identificador de la ciudad con sus respectivos identificadores propios de números y letras [20].

En el Ecuador, se tiene un diseño único para todo el país, las letras y números tienen un relieve de 2mm, su color es negro mate, y se encuentran sobre un fondo reflectivo que indica el tipo de servicio del vehículo. Cada placa lleva en la parte superior la palabra Ecuador y contiene 3 letras y de 3 a 4 dígitos (figura 1), la primera letra corresponde a la provincia en la cuál el carro ha sido matriculado, los demás caracteres son identificadores que han sido asignados para el vehículo [21].

Figura 1: Formato de Placa Vehicular en el Ecuador [21].

Es por ello que varios sistemas han optado por la identificación vehicular con una placa, como por ejemplo el acceso a los estacionamientos, en donde a través de este identificador se puede calcular el tiempo que un automóvil se ha encontrado dentro del mismo, y así realizar el cálculo de cuanto debe pagar cada usuario. También ha sido utilizada esta técnica para el control de acceso, un sistema implementado generalmente por empresas las cuales restringen el paso a vehículos autorizados [20].

2.1.1 Reconocimiento Óptico de Caracteres (OCR)

El reconocimiento óptico de caracteres se basa en una simulación de la habilidad humana para el reconocimiento automático de patrones entre los distintos caracteres alfanuméricos haciendo uso de distintos modelos tanto físicos como matemáticos.

Uno de los primeros trabajos para el reconocimiento de caracteres se realizó en 1929, donde Gustav Tauschek obtuvo una patente sobre OCR en Alemania, a lo que le precedió Handel en 1933 logrando obtener otra patente en Estados Unidos. Uno de los primeros sistemas comerciales aplicando OCR fue utilizado por una compañía de California, *Standard Oil Company*, la cual la utilizaba para leer las impresiones de tarjetas de crédito para propósitos de facturación.

Este tipo de técnica se utilizó por primera vez en identificación por número de placa en Gran Bretaña en el año 1979, en la cuál se realizaron pruebas en carreteras y túneles con la finalidad de reducir al mínimo errores del sistema con ello se llegó a

optimizar los sistemas ópticos de caracteres.

El reconocimiento óptico de caracteres, cuenta con distintas metodologías para lograr la identificación de patrones [21], entre los más destacados tenemos a los siguientes:

- Redes Neuronales: Se basa en solucionar problemas no por la secuencia de pasos específicos sino como la evolución de un sistema que aprende y genera soluciones, es decir cuenta con algún tipo de inteligencia.
- Método Lógico: Su solución se basa en utilizar conjuntos lógicos, combinacionales y secuenciales.
- Método Evolutivo: Brinda una estructura de datos donde se toma en consideración todas las posibles soluciones a un sistema.
- Método Probabilista: Hace uso de métricas como la varianza, covarianza, distribución entre otros, para solucionar un problema basado en la teoría de la probabilidad.
- Método Geométrico: Agrupa resultados, basándose en la cercanía de una serie de vectores acordes a un criterio en específico.

Además de las metodologías OCR cuenta con diversos software libres que han sido utilizados para el reconocimiento de patrones, como Cuneiformes, Tesseract, Ocrad, GOCR o también se puede llegar a encontrar en el mercado software con licencia como, ExperVision, Microsoft Office Document Imaging, OmniPage, SmartScore entre otros.

2.1.1.1 Esquema General del OCR

Cada método posee un esquema genérico que se basa en la comparación del carácter de entrada, el mismo que ha sido obtenido de una imagen inicial, la cual ha sido sometida a un proceso de filtración, con un alfabeto específico, obteniendo así el reconocimiento de cada uno de estos símbolos, la figura 2 muestra en síntesis como trabaja un OCR, frente a una imagen de entrada.

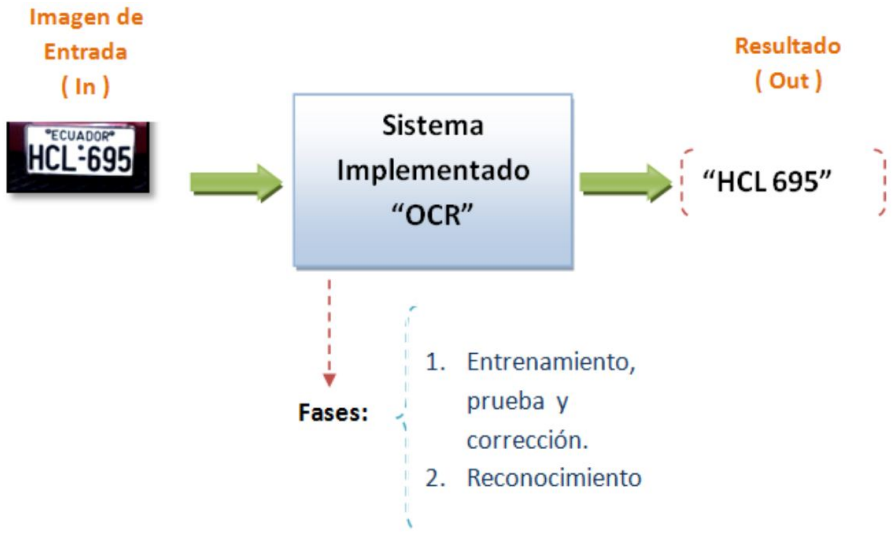


Figura 2: Esquema General del OCR y sus Fases [21].

2.1.1.2 Etapas Fundamentales del OCR

Un sistema de este tipo cuenta con dos etapas fundamentales, entrenamiento y reconocimiento [21].

- **Entrenamiento**

  En esta etapa se debe determinar el conjunto de caracteres que serán utilizados y con ello diseñar una imagen con texto que contenga un conjunto de ejemplos, tomando en consideración aspectos importantes como:

- Se debe contar con al menos 5 muestras para caracteres especiales.
- Cuando se tiene caracteres frecuentes, se debe tomar muchas más muestras.
- Los datos de entrenamiento deben ser agrupados por tipo de letra.

En base a ello, se puede realizar diferentes tipos de entrenamiento como, Distinción entre caracteres alfabéticos y numéricos, optimizar el reconocimiento cuando se tiene similitud de caracteres, generar patrones para caracteres especiales, identificación de caracteres segados, tomando en cuenta que una fotografía no nos brindará siempre una toma perpendicular como se muestra en la figura 3.

Figura 3: Caracteres Sesgados obtenidos de una toma fotográfica no perpendicular [21].

- **Reconocimiento**

  Una vez identificados los caracteres obtenidos en la imagen se entra en etapa de reconocimiento en la cual se compara con el alfabeto específico y se generan los valores obtenidos al usuario.

2.1.2 Sistemas de Reconocimiento Automático de Matrículas (ANPR)

Los sistemas de reconocimiento automático de placas vehiculares son aplicaciones desarrolladas a través de visión de computadora, sus componentes son parte de hardware y software entrenados para poder realizar la lectura de una placa.

Los sistemas ANPR se dieron a conocer en el año 1976, donde la policía británica comenzó a desarrollar los primeros prototipos que empezaron a funcionar en 1989 en industria, y posteriormente en la identificación de autos robados [19]. Los actuales sistemas de ANPR pueden escanear placas vehiculares con un rango de distancias entre 5 y 50 metros según el modelo de la cámara que se encuentre utilizando, incluso se pueden llegar a registrar imágenes de las placas de los vehículos que circulan a grandes velocidades.

La figura 4 esquematiza el proceso básico en el cual una cámara captura la imagen de una placa vehicular, luego de ello basándose en técnicas de procesamiento de imágenes se realiza la segmentación y la binarización de la captura, para determinar el número de placa por medio de un algoritmo basado en reconocimiento óptico de caracteres.

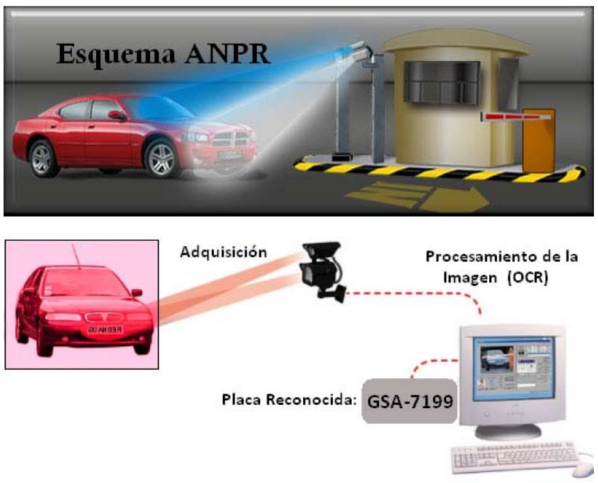


Figura 4: Sistemas de Reconocimiento Automático de Matrículas (ANPR) [21].

Los sistemas ANPR cuentan con diversas aplicaciones [20] entre las más destacadas tenemos:

- Estaciones de servicios, para el control de conductores y el pago que deben cancelar.
- Sistemas de Gestión de tráfico para determinar cuanto un vehículo se tarda al pasar en un periodo de congestión o sin ello.
- Control de fraude en autopistas, con el fin de determinar si un vehículo ha sido sustraído o no tiene derecho de circulación.

2.1.2.1 Componentes de un Sistema ANPR

Cada uno de estos sistemas basan su funcionamiento en el complemento tanto de herramientas de software como de hardware [22].

- **Componentes de Software**

  Se cuenta con herramientas y librerías de desarrollo, como Captura de video, procesamiento de imágenes y técnicas de OCR.

- **Componentes de Hardware**

  Se relaciona con el sistema de iluminación, cámaras, digitalizadores de imagen y un computador.

2.1.2.2 Funcionamiento de un Sistema ANPR

Los algoritmos fundamentales que un software basado en ANPR necesita para poder identificar una placa vehicular son [20]:

- **Localización de la Placa Vehicular:** Se refiere a poder aislar la imagen de la placa de una imagen general del vehículo.
- **Orientación y Tamaño de la matrícula:** Compensa los ángulos que hacen que las letras dentro de la placa parezcan torcidas o sesgadas.
- **Normalización:** Se encarga de ajustar el brillo y contraste de la imagen.
- **Segmentación de Caracteres:** Separa los diferentes caracteres encontrados en la placa vehicular.
- **Reconocimiento Óptico de Caracteres:** Los compara con un alfabeto específico, este punto se encuentra basado en las técnicas OCR, las mismas que ya fueron explicadas anteriormente.
- **Análisis Sintáctico y Geométrico:** Comprueba que los caracteres encontrados pertenezcan a la placa vehicular.

En base a estos puntos podemos determinar que un sistema ANPR, se compone de tres etapas fundamentales:

- Toma de imagen y reconocimiento de la placa en la imagen.
- Pre procesamiento de la placa.
- Reconocimiento de caracteres

2.2 Procesamiento Digital de Imágenes y Visión Artificial

2.2.1 Introducción al Procesamiento Digital de Imágenes

Basándose en que una señal de video es una secuencia de imágenes estáticas se determina que es posible realizar un análisis de una señal de video al procesar una cierta cantidad de imágenes estáticas en un intervalo de tiempo. Por ellos las imágenes pueden ser digitalmente analizadas ya sea por técnicas de procesamiento digital de imágenes (DIP) o con teoría de conjuntos, denominada morfología matemática.

El procesamiento digital de imágenes es un conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información relevante [23]. Es un campo de investigación abierto. El constante progreso en esta área no ha sido por si mismo, sino en conjunto con otras áreas con las cuales esta relacionada como las matemáticas, la computación, y el conocimiento cada vez mayor de ciertos órganos del cuerpo humano que intervienen en la percepción y en la manipulación de las imágenes. Esta temática ha surgido con la necesidad del hombre por imitar y usar ciertas características del ser humano como apoyo en la solución de problemas. El avance del Procesamiento Digital de Imágenes se ve reflejado en la medicina, la astronomía, geología, microscopía entre otras áreas que involucran el tratamiento de imágenes [24].

2.2.1.1 Imagen Digital

Ciertamente una imagen es la representación plana de un objeto real en las tres dimensiones, pero una imagen digital es un conjunto finito de elementos, es decir una imagen en dos dimensiones, donde se encuentra representada por una función f(X,Y)

donde X, Y representan las coordenadas de un plano y representan la intensidad de gris en la imagen, cuando X, Y son discretos y finitos se puede considerar a una imagen digital (figura 5).

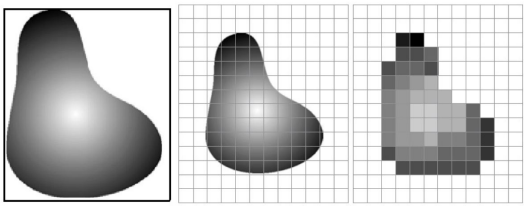


Figura 5: Representación de una imagen digital [25].

El manejo de las imágenes digitales se ha convertido en las últimas décadas en un tema de interés en distintas áreas de las ciencias naturales, las ciencias médicas y las aplicaciones tecnológicas entre otras. El crecimiento en el procesamiento de las computadoras, las capacidades de almacenamiento y los nuevos sistemas de captura e impresión de bajo costo han facilitado el desarrollo de ésta revolución tecnológica [25].

2.2.1.2 Etapas de Procesamiento

Dentro del proceso utilizado para el procesamiento digital de imágenes, tenemos 5 fases como se observa en la figura 6.

- **Captura:** Representa al diseño de las propiedades de la captura como el tipo de cámara, distancia al objeto, píxeles entre otros.
- **Pre- Procesamiento:** Se enfoca en disminuir al mínimo el entorno que no es de interés para el objeto que se desea captar y procesar.
- **Segmentación:** Se encarga de reconocer y extraer cada uno de los objetos presentes en la imagen.

- **Extracción de Características:** Esta etapa se encarga de seleccionar y extraer ciertas características que son relevantes para la identificación de los objetos deseados.

- **Identificación de Objetos:** Hace uso de un modelo de toma de decisión para decidir a qué categoría pertenece cada objeto, se utiliza entrenamiento para poder establecer clasificadores.

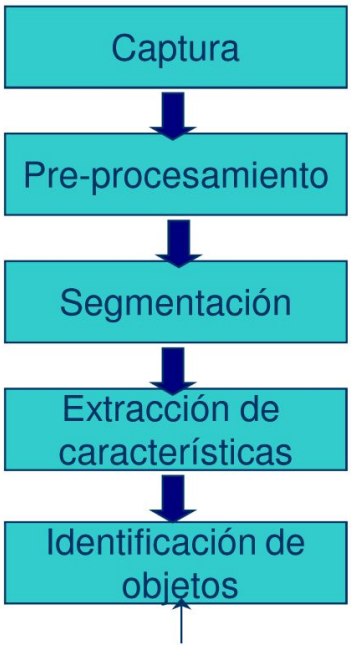


Figura 6: Etapas para el Procesamiento Digital de Imágenes [25].

2.2.2 Visión Artificial

Es un campo también conocido como visión por computador, y es parte de una disciplina de la inteligencia artificial y el propósito general de este campo es programar un computador para que pueda entender una escena o las características de una imagen digital. Sus aplicaciones son amplias, puede ser integrado en la industria, medicina, seguridad robótica e incluso en el entretenimiento [23].

La visión artificial se compone de un conjunto de procesos destinados a realizar el análisis de imágenes. Estos procesos son: captación de imágenes, memorización de la información, procesado e interpretación de los resultados [26].

2.2.2.1 Componentes básicos en el Procesamiento Digital de Imágenes

Existen seis componentes fundamentales en visión artificial [26] para representar y procesar una imagen: digitalización, píxel, nivel de grises, imagen binaria, escena y ventana.

- **Digitalización:** Se genera una función en la medida de brillo o muestreos realizados a intervalos de tiempo espaciados regularmente.
- **Píxel:** Representa a cada elemento de una cuadrícula que es considerada como la imagen digital.
- **Nivel de Grises:** Cuando una imagen es digitalizada, la intensidad del brillo en la escena original correspondiente a cada punto es cuantificada, dando lugar a un numero denominado nivel de grises.
- **Imagen Binaria:** Solo cuenta con dos niveles de gris, negro y blanco, cada pixel se convierte en dichos colores en función de un nivel umbral o conocido como binario.
- **Escena:** Es un área de memoria donde se guardan todos los parámetros referentes a la inspección de un objeto en particular.
- **Ventana:** Es el área específica de la imagen recogida que se quiere inspeccionar

2.2.2.2 Diagrama de Bloques

La visión artificial sigue un esquema de bloques desde que se capta la imagen hasta que se la muestra al usuario en un dispositivo, como se muestra en la figura 7.

- **Módulo de Digitalización:** Se encarga de convertir una señal analógica brindada por la cámara a una señal digital.
- **Memoria de Imagen:** Almacena la señal que ha sido digitalizada.
- **Módulo de Visualización:** Convierte la señal digital que se encuentra en la memoria en una señal de video analógica, la misma que será visualizada en el monitor.
- **Procesador de Imagen:** Se encarga de procesar e interpretar las imágenes captadas por la cámara.
- **Módulo de Entradas y Salidas:** Realiza la gestión de entrada y salida que actúan sobre los dispositivos externos.
- **Comunicación:** Representan la comunicación por la cual se encuentran conectados los dispositivos.

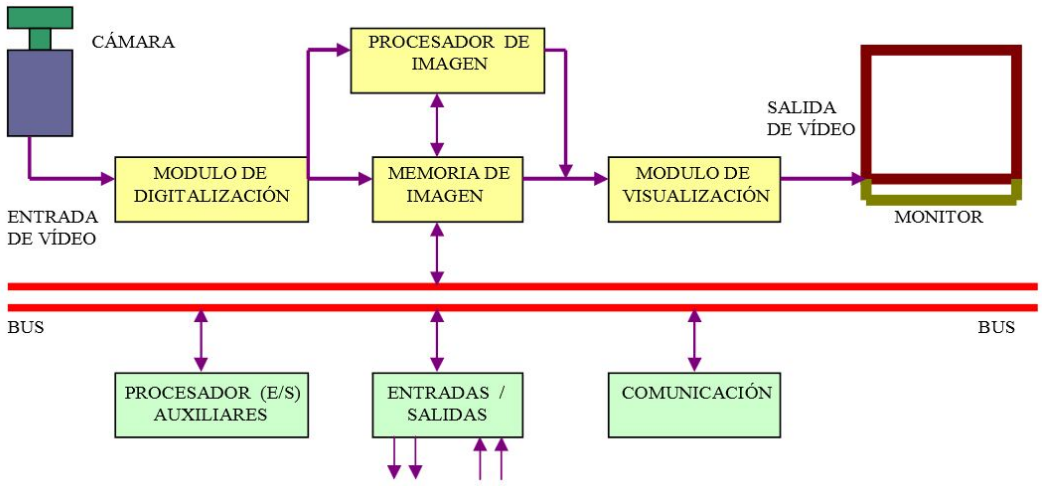


Figura 7: Diagrama de Bloques en Visión Artificial [26].

2.2.3 Técnicas de Procesamiento Digital de Imágenes

A continuación se describen las principales técnicas de procesamiento digital de imágenes utilizadas en el presente proyecto.

2.2.3.1 Filtrado de Imágenes

El filtrado lineal o no lineal de imágenes 2D son técnicas que se aplican sobre las imágenes donde se modifica el valor de cada uno de sus pixeles (x, y) en la imagen de origen, considerando y utilizando los pixeles que lo rodean (vecindad) para calcular una respuesta. En caso de un filtro lineal, es una suma ponderada de los valores de pixel. En caso de operaciones morfológicas, son los valores mínimos o máximos. La respuesta calculada se almacena en la imagen de destino en la misma ubicación (x, y). Esto significa que la imagen de salida será del mismo tamaño que la imagen de entrada, exceptuando algunos filtros especiales.

Una imagen se puede filtrar en el dominio del espacio, trabajando directamente

sobre los píxeles de la imagen, o en el dominio de la frecuencia, donde las operaciones se llevan a cabo en la transformada de Fourier de la imagen.

FILTRO GAUSSIANO

Un filtro gaussiano da menos peso a los pixeles a medida que estos están más alejados del centro de la máscara. Se usa generalmente para emborronar imágenes y eliminar ruido. Consiste en que cada píxel de la imagen se reemplaza por el valor calculado en la fórmula de la figura 8, utilizando los píxeles vecinos. Se puede operar mediante convolución con una máscara determinada, la cual modela la función gaussiana (figura 8).

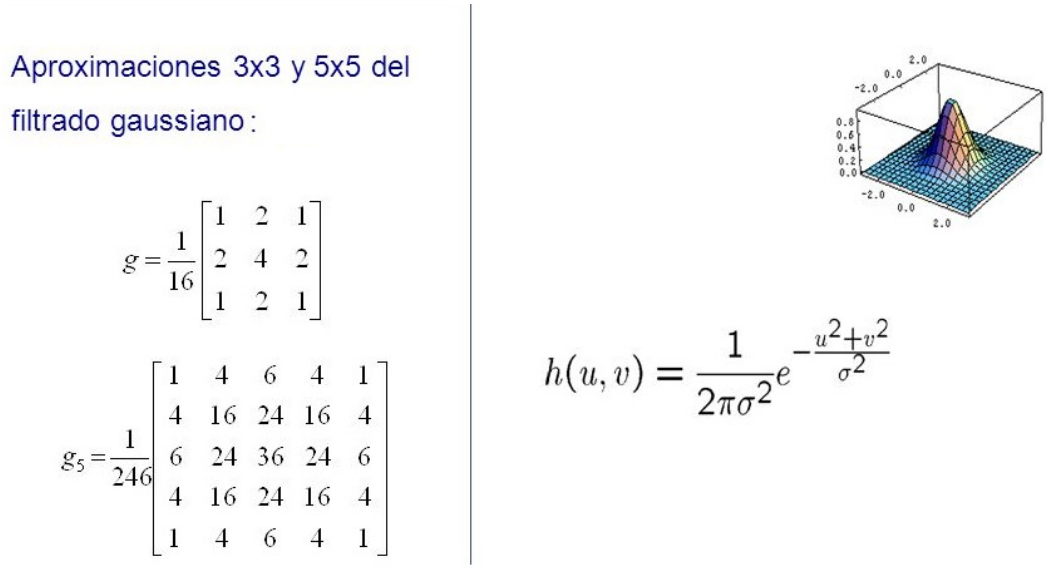


Figura 8: Fórmula y aproximaciones de 3x3 y 5x5 para el filtro gaussiano.

En la figura 9 se aprecia el resultado de eliminar el ruido en una imagen, empleando el filtro gaussiano.

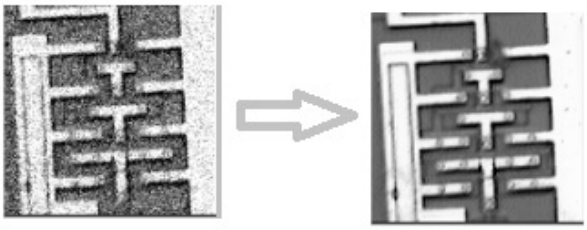


Figura 9: Eliminación de ruido con filtro gaussiano.

2.2.3.2 Detección de Bordes

Los bordes en una imagen digital se pueden definir como transiciones entre dos regiones de color significativamente distintos. Suministran una valiosa información sobre las fronteras de los objetos y puede ser utilizada para segmentar la imagen, reconocer objetos, entre otros. La mayoría de las técnicas para detectar bordes emplean operadores locales basados en distintas aproximaciones discretas de la primera y segunda derivada de los niveles de grises de la imagen.

La derivada de una señal continua proporciona las variaciones locales con respecto a la variable, de forma que el valor de la derivada es mayor cuanto más rápidas son estas variaciones.

En el caso de funciones bidimensionales f(x,y), como lo son las imágenes, la derivada es un vector que apunta en la dirección de la máxima variación de f(x,y) y cuyo módulo es proporcional a dicha variación. Este vector se denomina gradiente y se define (figura 10):

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} \qquad \text{Mag}[\nabla f(x,y)] = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

$$\theta = \arctan \frac{\frac{\partial f(x,y)}{\partial x}}{\frac{\partial f(x,y)}{\partial y}}$$

Figura 10: Fórmula del vector gradiente en imágenes.

En el caso bidimensional discreto, las distintas aproximaciones del operador gradiente se basan en diferencias entre los niveles de grises de la imagen. Las derivadas parciales en los ejes X e Y pueden aproximarse por la diferencia de pixeles adyacentes de la misma fila o columna (figura 11):

Eje X:

$$\frac{\partial f(x,y)}{\partial x} \approx \nabla_x f(x,y) = f(x,y) - f(x-1,y)$$

-1	1

Eje Y:

$$\frac{\partial f(x,y)}{\partial y} \approx \nabla_y f(x,y) = f(x,y) - f(x,y-1)$$

-1
1

Figura 11: Discretización del vector gradiente ejes X e Y.

Se puede determinar en imágenes individuales los bordes horizontales y verticales, los cuales se pueden sumar para obtener los bordes de toda la imagen en conjunto (figura 12)

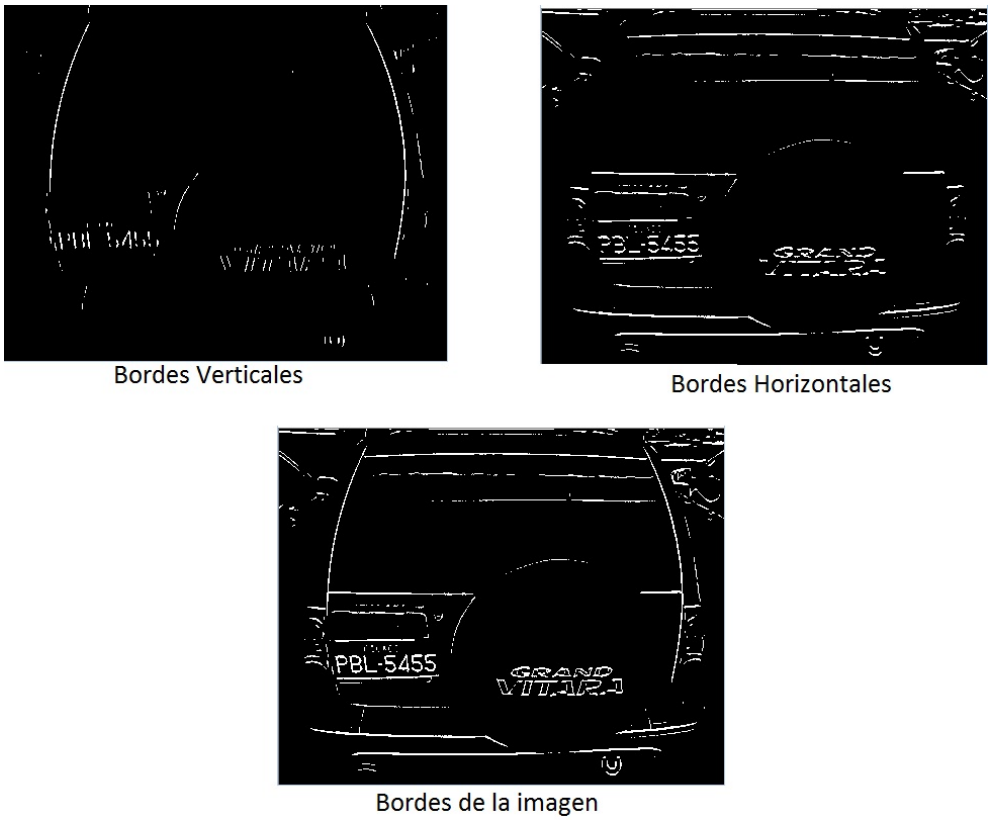
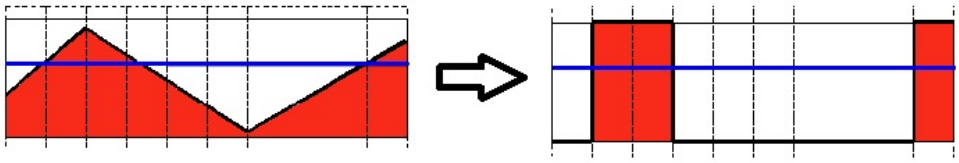


Figura 12: Detección de bordes horizontales y verticales.

2.2.3.3 Umbralización de Imágenes (Thresholding)

Es una técnica de segmentación de imágenes que tiene por objetivo separar grupos de pixeles con características similares. Esta separación es basada en la variación de intensidad entre los pixeles que corresponden a objetos, de los pixeles que corresponden al fondo de la imagen. Consiste en convertir una imagen en escala de grises en otra imagen que posee un número de tonos de grises reducido, siendo generalmente solo dos tonos (binarización) que corresponden a blanco y negro. Para esto, se realiza la comparación de la intensidad de cada pixel con respecto a un valor umbral (threshold). Si el valor de un pixel es menor al umbral, éste tomará el valor de 0 (negro) que corresponde al fondo de la imagen; y si el valor del pixel es mayor al umbral, tomará el valor de 1 (blanco) correspondiente a un objeto (figura 13).



$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Figura 13: Representación de la Umbralización de imágenes.

El valor del umbral puede ser especificado directamente o se lo puede determinar utilizando el algoritmo de Otsu, el cual encuentra un valor óptimo analizando la intensidad de cada pixel en toda la imagen. El método de Otsu utiliza técnicas estadísticas para resolver el problema. En concreto, se utiliza la variancia, que es una medida de la dispersión de valores, en este caso se trata de la dispersión de los niveles de gris. Se calcula el valor umbral de forma que la dispersión dentro de cada segmento sea lo más pequeña posible, pero al mismo tiempo la dispersión sea lo más alta posible entre segmentos diferentes. Para ello se calcula el cociente entre ambas variancias y se busca un valor umbral para el que este cociente sea máximo.

Es posible también aplicar una **umbralización por regiones**, la cual consiste en dividir la imagen en regiones más pequeñas y en cada una realizar el proceso de umbralización, independiente del resto de la imagen.

2.2.3.4 Proyección horizontal y vertical de una imagen

Consiste en realizar la sumatoria de cada valor de los pixeles para cada fila o columna de la imagen, obteniendo un solo vector de resultados. Generalmente se aplica la proyección horizontal cuando en la imagen existe una gran cantidad de segmentos verticales; y la proyección vertical cuando la mayoría de segmentos son horizontales.

Del vector resultante se puede extraer características como crestas y valles, los cuales determinan respectivamente las zonas de mayor concurrencia de objetos, y las ubicaciones donde no se encuentra la presencia de objetos en la imagen. En la figura 14, la proyección horizontal muestra la región que corresponde a los caracteres de la placa en conjunto, que permite ubicar una zona de interés; mientras que en la proyección vertical se presentan crestas y valles, las cuales corresponden e identifican a caracteres individuales y a los espacios entre estos, respectivamente.

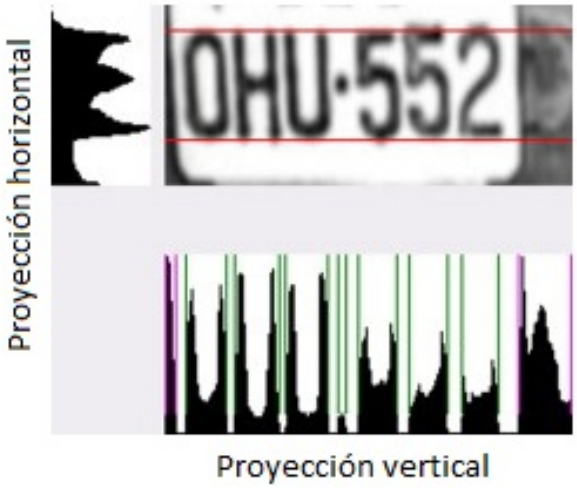


Figura 14: Proyección horizontal y vertical de una imagen.

2.2.3.5 Match Template (Comparación de Plantillas)

Es una técnica que se utiliza para buscar áreas de una imagen que son similares o presentan coincidencias con otra imagen de muestra o plantilla. Para su ejecución se necesitan dos componentes principales (figura 15):

- **Imagen fuente:** La imagen en la cual se espera encontrar una coincidencia con la imagen modelo.
- **Imagen modelo:** La imagen de muestra con la cual se compara la imagen fuente,

el objetivo es encontrar la zona de coincidencia más alta.

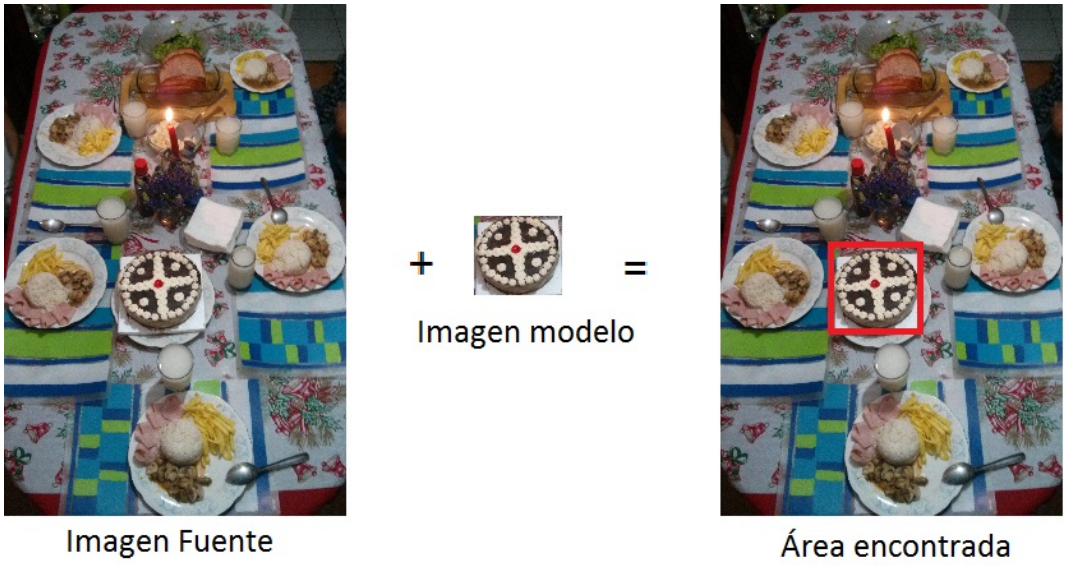


Figura 15: Ejemplo de Match Templating.

Para identificar el área de coincidencia, se compara la imagen de la plantilla con la imagen de origen, **deslizándola** sobre ella (figura 16):

Figura 16: Ejemplo de deslizamiento en Match Templating.

Por **deslizar**, se refiere a mover la imagen modelo un pixel a la vez, de izquierda a derecha, y de arriba hacia abajo. En cada ubicación se calcula una métrica, la cual representa cuan similar es la plantilla en esa área particular de la imagen fuente. Dicha métrica se guarda en una matriz, en la cual, después de terminar la comparación sobre toda la imagen fuente, se buscan los valores más altos, los que representarán la zona correspondiente a la imagen modelo buscada.

La métrica que se empleó en el proyecto fue la de **Correlación Normalizada en Imágenes**.

La correlación estadística determina la relación o dependencia que existe entre las dos variables que intervienen en una distribución bidimensional. Es decir, determinar si los cambios en una de las variables influyen en los cambios de la otra. En caso de que suceda, diremos que las variables están correlacionadas o que hay correlación entre ellas.

La correlación en imágenes se basa en la maximización de un coeficiente de correlación que se determina mediante el análisis del subconjunto de pixeles entre las dos imágenes. Su fórmula se muestra en la figura 17.

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Figura 17: Fórmula correlación normalizada.

Donde:

- **R**, es la matriz donde se guardan las métricas calculadas de toda la imagen.
- **T(x',y')**, es la intensidad del pixel en la imagen base a partir de la cual se miden

los desplazamientos la imagen modelo.

- **I(x,y)**, es la intensidad del pixel sobre el que se realiza la métrica en la imagen fuente.

2.3 Procesadores ARM de Bajo Consumo

Acorn RISC Machine, Advanced RISC Machine (ARM) es una arquitectura de procesadores RISC de 32 bits desarrollada por *ARM Limited*, que es ampliamente empleada en sistemas integrados. Actualmente, la familia de procesadores ARM está presente en el 75% de los sistemas integrados de CPU RISC de 32 bits, haciéndola una de las arquitecturas más prometedoras de 32 bits en el mundo [27].

2.3.1 Historia de los Procesadores ARM

El diseño de ARM se inició en 1983, como un proyecto de desarrollo de la empresa *Acorn Computers Ltd*, Roger Wilson y Steve Furber, comenzaron el desarrollo de una tecnología que se asemejaba a un *MOS Technology 6502* avanzado. Acorn tenía una larga línea de computadoras basadas en el 6502, por lo tanto, un chip que fuera similar podría representar una ventaja significativa a la compañía y en el mercado. Su trabajo finalizó en el año de 1985, y desde el siguiente año ya se presentó ARM2 el cual se consideró como la primera producción real de estos dispositivos [27].

El ARM2 fue considerado el microprocesador de 32 bits más simple en el mundo, con solo 30 mil transistores. Tampoco poseía caché, como la mayoría de las CPUs de esos días. Esta simplicidad les permitió un menor uso de energía. Su sucesor, el ARM3, fue producido con un caché de 4 KB y con mejor rendimiento es decir con un consumo menor.

En los años 80, Apple Computer comenzó a trabajar con Acorn en una nueva versión del núcleo de ARM, que se convertiría eventualmente en el ARM6. El primer modelo fue lanzado en 1991. En 1994, Acorn utilizó el ARM 610 como CPU principal en sus computadoras *Risc PC*. Luego de unos años DEC licenció la arquitectura y produjo el StrongARM. Este trabajo fue pasado luego a Intel como parte de una resolución judicial, e Intel tuvo la oportunidad de complementar su antigua línea i960 con el StrongARM. Luego Intel desarrolló su propia implementación de alto rendimiento conocido como XScale [27].

2.3.2 Evolución de los procesadores ARM de bajo consumo

El mercado siempre se ha centrado en dar énfasis a las velocidades y toda esa potencia que se está alcanzando con el desarrollo de los procesadores. Es conocido por todos que nos encontramos en una constante evolución, en donde los objetivos de casi todos los fabricantes ha sido seguir trabajando en aumentar velocidad y prestaciones a lo que vendría a ser el cerebro de los teléfonos y tablets.

Los fabricantes constantemente se encuentran buscando alternativas de optimización para el procesador, que sin lugar a dudas es el elemento central y siempre ha tenido una relevancia por encima del resto de componentes. Pero ciertos fabricantes han planteado nuevas propuestas de optimización en el desarrollo de sus productos, como el caso de ARM, que ha presentado entre una de sus líneas de desarrollo a los Procesadores de bajo consumo [28].

ARM ha planteado un concepto en el cual no busca centrarse en la velocidad, si no en el bajo consumo, ultra bajo. Este concepto se ha basado en una preparación para que sus procesadores sean un complemento para los dispositivos involucrados en el Internet de las cosas. Es por ese motivo, principalmente, por el que ARM trabaja en estos procesadores que deben tener un consumo mínimo. Así es posible que se puedan

abastecer de una fuente de energía que no dependa de una batería, su ventaja es que estas temáticas han sido poco exploradas y por ello se estudian muchas posibilidades, incluida la utilización de las ondas electromagnéticas para transmitir información [28].

2.3.3 Diseño y Tecnología de Procesadores ARM

El juego de instrucciones del ARM es similar al del MOS 6502, pero incluye características adicionales que le permiten conseguir un mejor rendimiento en su ejecución. La característica esencial en estos procesadores es el uso de los 4 bits superiores como código de condición, haciendo que cualquier instrucción pueda ser condicional. Este corte reduce el espacio para algunos desplazamientos en el acceso a la memoria, pero permite evitar perder ciclos de reloj al ejecutar pequeños partes de código con ejecución condicional [27].

El procesador ARM también tiene algunas características especiales como el direccionamiento relativo, y el pre y post incremento en el modo de direccionamiento. Además cuenta con dos modos de funcionamiento:

- **ARMI:** Las instrucciones ocupan 4 bytes, y se ejecutan de una manera más rápida y potente. Tiene implementada una tecnología que permite que ciertos tipos de arquitecturas ejecuten Java bytecode nativamente en el hardware.
- **THUMB:** Las instrucciones ocupan 2 bytes y su ventaja es el menor consumo de potencia. El rendimiento puede ser superior a un código de 32 bits en donde el puerto de memoria o ancho del bus de comunicaciones son menores a 32 bits.

ARM está usando el concepto llamado *Big.Little* (Grande.Pequeño) en el cual los núcleos de bajo consumo de energía se mezclan con núcleos de gran desempeño para proporcionar cómputo balanceado. Por ejemplo, un teléfono inteligente puede tener

núcleos de gran desempeño para manejar las aplicaciones demandantes, y núcleos de bajo consumo de energía para manejar tareas de bajo nivel como las llamadas telefónicas.

ARM domina en teléfonos inteligentes y tablets, pero apunta a poner su marca en el mercado de servidores dominado por los chips x86 de Intel y AMD. Existe un creciente interés en los servidores ARM como una forma energéticamente eficiente de manejar un gran número de solicitudes web como las búsquedas y las redes sociales. Dell y HP ya ofrecen prototipos de servidores ARM para evaluación a los clientes que buscan desplegar servidores ARM para recortar los costos en energía.

CAPÍTULO 3

MATERIALES Y MÉTODOS

En el presente proyecto se utiliza el sistema embebido ODROID U3 como unidad de procesamiento central, con el fin de determinar la velocidad de los vehículos, el número de placa, y trasmitir esta información hacia una base de datos en una red local. Para ello se emplea una cámara de video y una antena Wi-Fi como periféricos de adquisición y envío de información respectivamente. A continuación se expone el por qué se utilizó este sistema embebido y se detallan sus componentes y características:

3.1 *Hardware*

3.1.1 Parámetros de Selección del Sistema de Procesamiento Utilizado

En la actualidad existen diversas herramientas para realizar el procesamiento de datos computacionales, cada una de ellas posee características que la hacen idealmente funcional para ciertas actividades, y que cumplen eficazmente los requerimientos de los usuarios. Se puede diferenciar como los más destacados a los **Sistemas de Propósito General** y a los **Sistemas Embebidos**:

3.1.1.1 SISTEMAS DE PROPÓSITO GENERAL

- **COMPUTADORES PERSONALES**:

  Los computadores personales son en la actualidad una herramienta de uso común. Toman su nombre debido a que prácticamente se las puede utilizar para cualquier tarea y actividad computacional de la vida diaria de las personas, son empleados para realizar tareas desde escribir un documento de texto, hasta realizar complejas actividades industriales. No tienen un uso específico, por lo cual permiten la instalación y ejecución de programas que realizan actividades determinadas según el requerimiento del usuario.

- **SERVIDORES DEDICADOS**:

  Un servidor es un computador que presta servicios a otras computadoras a través de una red. Estos servicios pueden ser muy variados desde administrar bases de datos, usuarios y recursos en una red, datos, servicios como telefonía y voz sobre ip, alojamiento web, entre otros. [29] Los requerimientos de hardware para los servidores varían dependiendo del tipo de aplicación del servidor.

  La tarea de los servidores es la de brindar servicios dentro de una red a un gran número de usuarios, por lo que requieren características tales como conexiones de alta velocidad y altas prestaciones para todos los dispositivos de I/O. Generalmente se accede a los servidores a través de la red, pudiendo funcionar sin necesidad de un monitor u otros dispositivos de entrada. Muchos servidores no cuentan con una interfaz gráfica de usuario (GUI) puesto que esta funcionalidad consume recursos que pueden ser utilizados por otros procesos. De igual manera las interfaces de audio y USB también pueden ser omitidas.

Figura 18: Servidores DELL PowerEdge. [30]

3.1.1.2 SISTEMAS EMBEBIDOS

Es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas, [31] [32] frecuentemente en un sistema de computación en tiempo real. En comparación a lo que ocurre con los ordenadores de propósito general que están diseñados para cubrir diversas necesidades, los sistemas embebidos son diseñados para cubrir necesidades específicas. La mayoría de sus componentes se encuentran incluidos en una sola placa y muchas veces los dispositivos resultantes no tienen el aspecto de lo que comunmente se asocia a una computadora. Como ejemplo de sistemas embebidos podrían ser dispositivos como un taxímetro, un sistema de control de acceso, la electrónica que controla una máquina expendedora, el sistema de control de una fotocopiadora entre otras múltiples aplicaciones.

Por lo general los sistemas embebidos se pueden programar directamente en lenguaje de bajo nivel o ensamblador del microcontrolador o microprocesador incorporado sobre el mismo, o también pueden utilizarse lenguajes compilados como C o C++; en

algunos casos, cuando el tiempo de respuesta de la aplicación no es un factor crítico, también pueden usarse lenguajes interpretados como JAVA o PYTHON.

Componentes de un sistema embebido:

En la parte central se encuentra ***el microprocesador, microcontrolador, DSP, etc***. Es decir, la CPU o unidad que aporta capacidad de cómputo al sistema, pudiendo incluir memoria interna o externa, un micro con arquitectura específica según requisitos.

La comunicación adquiere gran importancia en los sistemas embebidos. Lo normal es que el sistema pueda comunicarse mediante interfaces estándar de cable o inalámbricas. Así un SI normalmente incorporará puertos de comunicaciones del tipo RS-232, RS-485, SPI, I2C, CAN, USB, IP, Wi-Fi, GSM, GPRS, DSRC, etc.

El subsistema de presentación suele ser una pantalla gráfica, táctil, LCD, alfanumérico, etc.

Actuadores son los posibles elementos electrónicos que el sistema se encarga de controlar. Puede ser un motor eléctrico, un conmutador tipo relé etc. El más habitual puede ser una salida de señal PWM para control de la velocidad en motores de corriente continua.

El ***módulo de Entradas y Salidas*** analógicas y digitales suele emplearse para digitalizar señales analógicas procedentes de sensores, activar diodos LED, reconocer el estado abierto cerrado de un conmutador o pulsador, etc.

El módulo de reloj es el encargado de generar las diferentes señales de reloj a partir de un único oscilador principal. El tipo de oscilador es importante por varios aspectos: por la frecuencia necesaria, por la estabilidad necesaria y por el consumo de corriente requerido. El oscilador con mejores características en cuanto a estabilidad y coste son los basados en resonador de cristal de cuarzo, mientras que los que requieren menor consumo son los RC. Mediante sistemas PLL se obtienen otras frecuencias con

la misma estabilidad que el oscilador patrón.

El módulo de energía se encarga de generar las diferentes tensiones y corrientes necesarias para alimentar los diferentes circuitos del SE. Usualmente se trabaja con un rango de posibles tensiones de entrada que mediante conversores ac/dc o dc/dc se obtienen las diferentes tensiones necesarias para alimentar los diversos componentes activos del circuito. El consumo de energía puede ser determinante en el desarrollo de algunos sistemas embebidos que necesariamente se alimentan con baterías, con lo que el tiempo de uso del SE suele ser la duración de la carga de las baterías.

Microprocesadores y sistemas embebidos

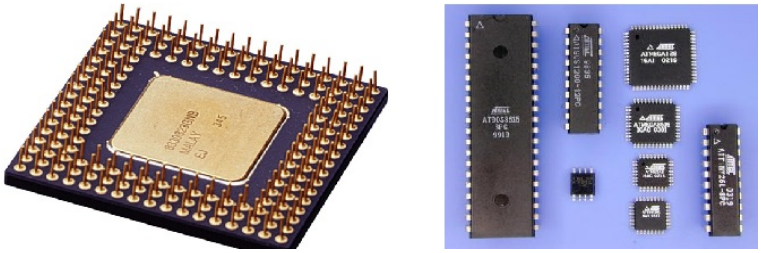


Figura 19: Microprocesadores AVR [33].

Un microprocesador es una implementación en forma de circuito integrado (IC) de la Unidad Central de Proceso CPU de una computadora. Frecuentemente se refiere a un microprocesador como simplemente "CPU" y la parte de un sistema que contiene al microprocesador se denomina subsistema de CPU.

Los subsistemas de entrada/salida y memoria pueden ser combinados con un subsistema de CPU para formar una computadora o sistema embebido completo. Estos subsistemas se interconectan mediante los buses de sistema (formados a su vez por el bus de control, el bus de direcciones y el bus de datos).

El subsistema de entrada acepta datos del exterior para ser procesados mientras que el subsistema de salida transfiere los resultados hacia el exterior. Lo más habitual

es que haya varios subsistemas de entrada y varios de salida. A estos subsistemas se les reconoce habitualmente como periféricos de E/S.

El subsistema de memoria almacena las instrucciones que controlan el funcionamiento del sistema. Estas instrucciones comprenden el programa que ejecuta el sistema. La memoria también almacena varios tipos de datos: datos de entrada que aún no han sido procesados, resultados intermedios del procesado y resultados finales en espera de salida al exterior.

Es importante darse cuenta de que los subsistemas estructuran a un sistema según funcionalidades. La subdivisión física de un sistema, en términos de circuitos integrados o placas de circuito impreso (PCB) puede y es normalmente diferente. Un solo circuito integrado (IC) puede proporcionar múltiples funciones, tales como memoria y entrada/salida.

Un microcontrolador (MCU) es un IC que incluye una CPU, memoria y circuitos de entradas y salidas (E/S) . Entre los subsistemas de E/S que incluyen los microcontroladores se encuentran los temporizadores, los convertidores analógico a digital (ADC) y digital a analógico (DAC) y los canales de comunicaciones serie. Estos subsistemas de E/S se suelen optimizar para aplicaciones específicas (audio, video, procesos industriales, comunicaciones, etc.).

En general, un SE (Sistema Electrónico) consiste en un sistema con microprocesador cuyo hardware y software están específicamente diseñados y optimizados para resolver un problema concreto eficientemente. Normalmente un SE interactúa continuamente con el entorno para vigilar o controlar algún proceso mediante una serie de sensores. Su hardware se diseña normalmente a nivel de chips, o de interconexión de PCB, buscando la mínima circuitería y el menor tamaño para una aplicación particular.

Otra alternativa consiste en el diseño a nivel de PCB consistente en el ensam-

blado de placas con microprocesadores comerciales que responden normalmente a un estándar como el PC-104 (placas de tamaño concreto que se interconectan entre sí apilándolas unas sobre otras, cada una de ellas con una funcionalidad específica dentro del objetivo global que tenga el SE). Esta última solución acelera el tiempo de diseño pero no optimiza ni el tamaño del sistema ni el número de componentes utilizados ni el coste unitario. En general, un sistema embebido simple contará con un microprocesador, memoria, unos pocos periféricos de E/S y un programa dedicado a una aplicación concreta almacenado permanentemente en la memoria. El término embebido o empotrado hace referencia al hecho de que el microcomputador está encerrado o instalado dentro de un sistema mayor y su existencia como microcomputador puede no ser aparente. Un usuario no técnico de un sistema embebido puede no ser consciente de que está usando un sistema computador. En algunos hogares las personas, que no tienen por qué ser usuarias de una computadora personal estándar (PC), utilizan del orden de diez o más sistemas embebidos cada día.

Las microcomputadoras en estos sistemas controlan electrodomésticos tales como: televisores, videos, lavadoras, alarmas, teléfonos inalámbricos, etc. Incluso una PC tiene sistemas embebidos en el monitor, impresora, y periféricos en general, adicionales a la CPU de la propia PC. Un automóvil puede tener hasta un centenar de microprocesadores y microcontroladores que controlan cosas como la ignición, transmisión, dirección asistida, frenos antibloqueo (ABS), control de la tracción, etc.

Los sistemas embebidos se caracterizan normalmente por la necesidad de dispositivos de E/S especiales. Cuando se opta por diseñar el sistema embebidos partiendo de una placa con microcomputador también es necesario comprar o diseñar placas de E/S adicionales para cumplir con los requisitos de la aplicación concreta.

Muchos sistemas embebidos son sistemas de tiempo real. Un sistema de tiempo real debe responder, dentro de un intervalo restringido de tiempo, a eventos externos

mediante la ejecución de la tarea asociada con cada evento. Los sistemas de tiempo real se pueden caracterizar como blandos o duros. Si un sistema de tiempo real blando no cumple con sus restricciones de tiempo, simplemente se degrada el rendimiento del sistema, pero si el sistema es de tiempo real duro y no cumple con sus restricciones de tiempo, el sistema fallará. Este fallo puede tener posiblemente consecuencias catastróficas.

Un sistema embebido complejo puede utilizar un sistema operativo como apoyo para la ejecución de sus programas, sobre todo cuando se requiere la ejecución simultánea de los mismos. Cuando se utiliza un sistema operativo lo más probable es que se tenga que tratar de un sistema operativo de tiempo real (RTOS), que es un sistema operativo diseñado y optimizado para manejar fuertes restricciones de tiempo asociadas con eventos en aplicaciones de tiempo real. En una aplicación de tiempo real compleja la utilización de un sistema operativo de tiempo real multitarea puede simplificar el desarrollo del software.

Arquitecturas de computadores más empleadas

Una PC embebida posee una arquitectura semejante a la de un PC. Brevemente éstos son los elementos básicos:

- **Microprocesador**

  Es el encargado de realizar las operaciones de cálculo principales del sistema. Ejecuta código para realizar una determinada tarea y dirige el funcionamiento de los demás elementos que le rodean.

- **Memoria**

  En ella se encuentra almacenado el código de los programas que el sistema puede ejecutar así como los datos. Su característica principal es que debe tener un acceso de lectura y escritura lo más rápido posible para que el microprocesador

no pierda tiempo en tareas que no son meramente de cálculo. Al ser volátil el sistema requiere de un soporte donde se almacenen los datos incluso sin disponer de alimentación o energía.

- **Caché**

  Memoria más rápida que la principal en la que se almacenan los datos y el código accedido últimamente. Dado que el sistema realiza microtareas, muchas veces repetitivas, la caché hace ahorrar tiempo ya que no hará falta ir a memoria principal si el dato o la instrucción ya se encuentra en la caché. Dado su alto precio tiene un tamaño muy inferior (8?512 KB) con respecto a la principal (8?256 MB). En el interior del chip del microprocesador se encuentra una pequeña caché (L1), pero normalmente se tiene una mayor en otro chip de la placa madre (L2).

- **Disco duro**

  En él la información no es volátil y además puede conseguir capacidades muy elevadas. A diferencia de la memoria que es de estado sólido éste suele ser magnético. Pero su excesivo tamaño a veces lo hace inviable para PC embebidas, con lo que se requieren soluciones como unidades de estado sólido. Otro problema que presentan los dispositivos magnéticos, a la hora de integrarlos en sistemas embebidos, es que llevan partes mecánicas móviles, lo que los hace inviables para entornos donde estos estarán expuestos a ciertas condiciones de vibración. Existen en el mercado varias soluciones de esta clase (DiskOnChip, CompactFlash, IDE Flash Drive, etc.) con capacidades suficientes para la mayoría de sistemas embebidos (desde 2 MB hasta más de 1 GB). El controlador del disco duro de PC estándar cumple con el estándar IDE y es un chip más de la placa madre.

- **Disco flexible**

  Su función es la de almacenamiento, pero con discos con capacidades mucho

más pequeñas y la ventaja de su portabilidad. Normalmente se encontraban en computadora personal estándar pero no así en una PC embebida. Llevan varios años en total desuso en PC comunes.

- **BIOS-ROM**

  BIOS (Basic Input & Output System, sistema básico de entrada y salida) es código que es necesario para inicializar la computadora y para poner en comunicación los distintos elementos de la placa madre. La ROM (Read Only Memory, memoria de sólo lectura no volátil) es un chip donde se encuentra el código BIOS.

- **CMOS-RAM**

  Es un chip de memoria de lectura y escritura alimentado con una pila donde se almacena el tipo y ubicación de los dispositivos conectados a la placa madre (disco duro, puertos de entrada y salida, etc.). Además contiene un reloj en permanente funcionamiento que ofrece al sistema la fecha y la hora.

- **Chipset**

  Chip que se encarga de controlar las interrupciones dirigidas al microprocesador, el acceso directo a memoria (DMA) y al bus ISA, además de ofrecer temporizadores, etc. Es frecuente encontrar la CMOS-RAM y el reloj de tiempo real en el interior del Chip Set.

- **Entradas al sistema**

  Pueden existir puertos para mouse, teclado, vídeo en formato digital, comunicaciones serie o paralelo, etc.

- **Salidas al sistema**

  Puertos de vídeo para monitor o televisión, pantallas de cristal líquido, altavoces, comunicaciones serie o paralelo, etc.

- **Ranuras de expansión para tarjetas de tareas específicas**

  Ranuras que pueden no venir incorporadas en la placa madre, como pueden ser más puertos de comunicaciones, acceso a red de computadoras vía LAN (Local Area Network, red de área local) o vía red telefónica: básica, RDSI (Red Digital de Servicios Integrados), ADSL (Asynchronous Digital Subscriber Loop, Lazo Digital Asíncrono del Abonado), Cablemódem, etc. Un PC estándar suele tener muchas más ranuras de expansión que una PC embebida. Las ranuras de expansión están asociadas a distintos tipos de bus: VESA, ISA, PCI, NLX (ISA + PCI), etc.

Actualmente existen en el mercado fabricantes que integran un microprocesador y los elementos controladores de los dispositivos fundamentales de entrada y salida en un mismo chip, pensando en las necesidades de los sistemas embebidos (bajo coste, pequeño tamaño, entradas y salidas específicas, etc.). Su capacidad de proceso suele ser inferior a los procesadores de propósito general pero cumplen con su cometido ya que los sistemas donde se ubican no requieren tanta potencia. Los principales fabricantes son STMicroelectronics (familia de chips STPC), AMD (familia Geode), Motorola (familia ColdFire) e Intel.

En cuanto a los sistemas operativos necesarios para que un sistema basado en microprocesador pueda funcionar y ejecutar programas suelen ser específicos para los sistemas embebidos. Así se encuentran los sistemas operativos de bajos requisitos de memoria, posibilidad de ejecución de aplicaciones de tiempo real, modulares (inclusión sólo de los elementos necesarios del sistema operativo para el sistema embebido concreto), etc. Los más conocidos en la actualidad son Windows CE, QNX y VxWorks de WindRiver.

Ventajas de un sistema embebido sobre los sistemas tradicionales

Los equipos industriales de medida y control tradicionales están basados en un microprocesador con un sistema operativo privativo o específico para la aplicación correspondiente. Dicha aplicación se programa en ensamblador para el microprocesador dado o en lenguaje C, realizando llamadas a las funciones básicas de ese sistema operativo que en ciertos casos ni siquiera llega a existir. Con los modernos sistemas PC embebida basados en microprocesadores i486 o i586 se llega a integrar el mundo del PC compatible con las aplicaciones industriales. Ello implica numerosas ventajas:

- Posibilidad de utilización de sistemas operativos potentes que ya realizan numerosas tareas: comunicaciones por redes de datos, soporte gráfico, concurrencia con lanzamiento de threads, etc. Estos sistemas operativos pueden ser los mismos que para PC compatibles (Linux, Windows, MS-DOS) con fuertes exigencias en hardware o bien ser una versión reducida de los mismos con características orientadas a los PC embebidos.

- Al utilizar dichos sistemas operativos se pueden encontrar fácilmente herramientas de desarrollo software potentes así como numerosos programadores que las dominan, dada la extensión mundial de las aplicaciones para PC compatibles.

- Reducción en el precio de los componentes hardware y software debido a la gran cantidad de PCs en el mundo.

Por lo anteriormente mencionado, se decidió utilizar un sistema embebido para el procesamiento y control general del presente trabajo, específicamente el *ODROID U3*, pues se requiere que el proyecto sea portable, replicable, de bajo costo y de bajo consumo de recursos. Las características que destacan al sistema elegido sobre otros sistemas similares como *Arduino*, *Raspberry Pi*, *BeagleBone Black*, entre otros, son su procesador *ARM Exynos4412 Prime Quad-Core 1.7Ghz* y su memoria RAM *2Gbyte LPDDR2 880Mega Data Rate*, características que hacen del *Odriod U3* un sistema

apto para realizar procesamiento de imágenes requiriendo poco consumo de recursos gracias a la tecnología ARM [15] y LPDDR [16]. Utiliza para su funcionamiento hardware y software de código abierto, lo cual permite que el proyecto sea replicable, y sus dimensiones físicas 83 x 48 mm y 48g de peso le añaden portabilidad. Adicionalmente, posee módulos Ethernet y Wireless LAN que le proporcionan conectividad y comunicación remota.

3.1.2 *Sistema Embebido ODROID U3*

Figura 20: Sistema embebido *ODROID U3*. [34]

Es una plataforma de desarrollo elaborada por la empresa Sur Coreana *HARDKERNEL* la cual incluye hardware y software que pueden ser adaptados a los requerimientos de sus usuarios. Se lo considera como un sistema embebido de propósito general debido a que sus funcionalidades pueden ser modificadas, presenta bajo consumo de energía, sus dimensiones son relativamente pequeñas (su tamaño se compara a un mouse de computadora), y presenta un rendimiento óptimo al realizar tareas en tiempo real con costo computacional elevado como el procesamiento de imágenes. En la tabla 1 se detallan sus especificaciones:

Tabla 1: Especificaciones del Sistema Embebido *ODROID U3.*

Procesador	**Samsung Exynos4412 Prime Cortex-A9 Quad Core 1.7Ghz with 1MB L2 cache**
Memoria	2048MB(2GB) LP-DDR2 880Mega data rate
Acelerador 3D	Mali-400 Quad Core 440MHz
Video	Soporta 1080p via cable HDMI (H.264+AAC formato basado en MP4)
Salida de Video	Conector micro HDMI
Audio	Salida de audio estándar de 3.5mm HDMI Digital
LAN	10/100Mbps Ethernet con conector RJ-45(soporta Auto-MDIX)
USB2.0 Host	3 puertos de alta velocidad conector estándar tipo A.
Dispositivo USB 2.0	ADB/Mass storage(Micro USB), Modo Host .
Display	Monitor HDMI
IO Port	GPIO, UART, I2C, SPI
Almacenamiento	MicroSD Card Slot Ranura para módulo eMMC
Alimentación	5V 2A Power
Software del Sistema	Linux : Xubuntu 13.10 o su ùltima versión Android : u-boot 2010.12, Kernel 3.0.x, Android 4.x Código fuente completo disponible.
Dimensiones del PCB	83 x 48 mm
Peso	48g incluído su estuche protector.

En la figura 21 se muestra la distrubución de los componentes de la placa del *ODROID U3* listados en la tabla 2.

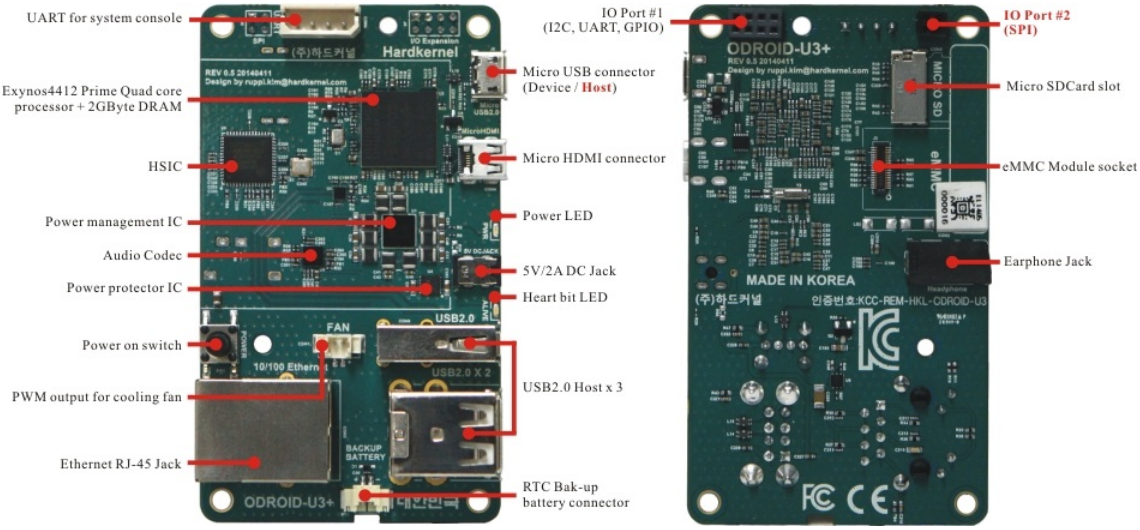


Figura 21: Distribución de componentes del *ODROID U3* [34].

Tabla 2: Componentes del Sistema Embebido *ODROID U3.*

CPU	Procesador Quad-core Exynos4412 Prime Cortex-A9 1.7GHz 2Gbyte LPDDR2 880Mega Data Rate
PMIC	MAX77686 Power Management IC de MAXIM
HSIC USB 2.0 Hub	USB3503A Integrated USB 2.0-compatible hub / HSIC upstream port from SMSC/Microchip
HSIC Ethernet controller	Controlador Ethernet LAN9730HSIC USB 2.0 to 10/100 con HP Auto-MDIX de SMSC/Microchip
Audio CODEC	MAX98090 es un CODEC de MAXIM con destacadas características de alto desempeño.
Protection IC	Protección de sobre voltaje NCP372 , de sobre corriente y voltaje inverso IC de OnSemi.
USB Load switch	Protección NCP380 para suministro de energía USB de OnSemi.
HDMI conditioner	Transmisor HDMI IP4791CZ12 de NXP
HDMI connector	Etándar Micro-HDMI, soporta resoluciones de hasta 1920 x 1080
Connectivity	USB Host x 3, Device x 1, Ethernet RJ-45, Puerto de Audio
IO Ports	GPIO, UART, I2C, SPI
Storage Slot	Ranura Micro-SD, conector para módulo eMMC
DC Input	Entrada de 5V / 2A, diámetro interno 0.8mm, diámetro externo 2.5mm

En la figura 22 se muestra el diagrama de bloques del *ODROID U3*, que contiene un esquema conceptual de la distribución de sus componentes. En la parte central se encuentra la plataforma de procesamiento de datos y multimedia, a las cuales se conectan los demás componentes para formar la aplicación de procesamiento central.

A los extremos se encuentran los componentes de entrada y salida que proporcionan la interacción con el usuario.

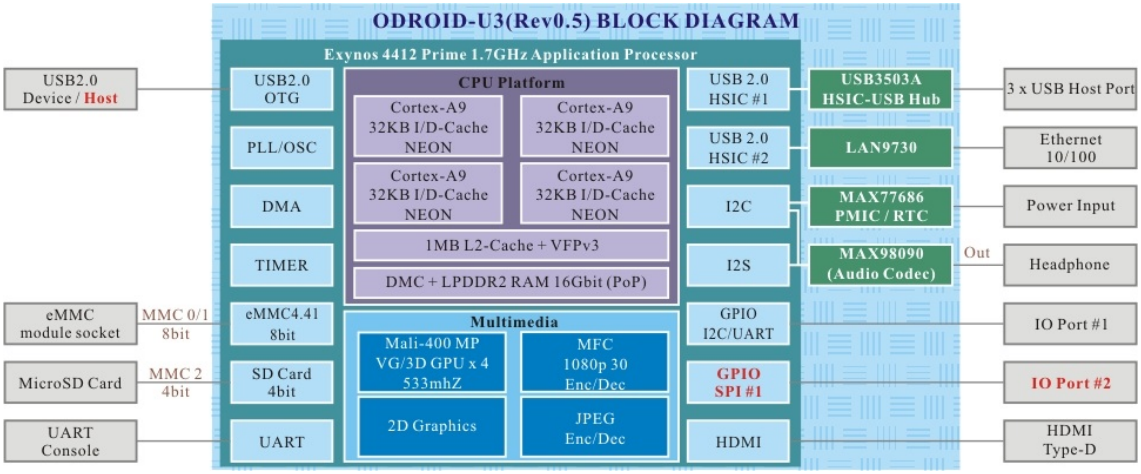


Figura 22: Diarama de bloques *ODROID U3* [34].

CÁMARA ODROID USB-CAM 720P

Figura 23: Cámara ODROID USB-CAM 720P.

Es la cámara utilizada para la adquisición de video y captura de imágenes en el presente proyecto (figura 23). Se eligió este modelo ya que es proporcionado por la misma empresa *HARDKERNEL* ofreciendo una compatibilidad completa con el *ODROID U3*. Su resolución de imagen es hasta 720P HD que es la resolución óptima para obtener los primeros resultados en trabajos de procesamiento de imágenes. Sus características

y componentes son los siguientes:

- Resolución real 720P HD, tamaño de salida de imagen 16:9
- Interface de alta velocidad USB2.0 plug-n-play (UVC)
- Sensor CMOS de 1.0 Megapixeles (resolución 1280 * 720 HD)
- Micrófono UAC(USB Audio Class) incluído.
- Soporta formatos de captura MPEG/MJPEG(Video)y BMP/JPEG(imágenes)
- Hasta 30 fps (imágenes por segundo)
- Controlador de imagen SONIX SN9C259 USB
- Sensor de imágen Novatek NY99140
- Alimentación DC 5V/500mA
- Ángulo de visión: 65 grados
- Campo de visión: 68 grados

3.2 *Software*

El funcionamiento del hardware detallado en la sección anterior requiere un sistema operativo apropiado para sus especificaciones, arquitectura y características. Además, también son necesarias aplicaciones que permitan la implementación de los algoritmos en forma de código para su posterior ejecución y desarrollo final del proyecto. A continuación se describen estos aspectos:

3.2.1 Sistemas Operativos Sobre Procesadores ARM

La familia ARM es la familia más avanzada de microcontroladores. Hay muchas razones por las que se desarrollan aplicaciones y proyectos basadas en sistemas embebidos, tales como la funcionalidad del dispositivo, un conjunto de periféricos y procesamiento de datos de alta velocidad. Para esto, requieren un sistema operativo que coordine todas las actividades que se realizan en el sistema, el cual debe ser diseñado para el conjunto de instrucciones que posee su arquitectura. En la actualidad el sistema operativo base compatible con procesadores ARM es ***Linux***, sobre el cual se han derivado varias distribuciones compatibles con el *ODROID U3*. *HARDKERNEL* recomienda el uso de las dos más conocidas que son ***ANDROID*** y ***UBUNTU***, aunque no se limita su uso sólamente a éstas.

3.2.1.1 ANDROID

Es un sistema operativo basado en el núcleo Linux. Fue diseñado para dispositivos móviles con pantalla táctil como teléfonos inteligentes o tablets, para relojes inteligentes, televisores, automóviles y sistemas embebidos de desarrollo. Todos estos dispositivos utilizan un procesador ARM. Fue desarrollado inicialmente por *Android Inc.*, una firma comprada por *Google* en 2005. La plataforma de hardware principal de Android es la arquitectura ARM, aunque también hay soporte para x86 en el proyecto *Android-x86*,[35] y Google TV utiliza una versión especial de Android x86. Android se desarrolla de forma abierta y se puede acceder tanto al código fuente [36] como a la lista de incidencias [37] donde se pueden ver problemas aún no resueltos y reportar problemas nuevos.

APLICACIONES

Las aplicaciones se desarrollan habitualmente en el lenguaje *Java* con *Android*

Software Development Kit (Android SDK), pero están disponibles otras herramientas de desarrollo, incluyendo un Kit de Desarrollo Nativo para aplicaciones o extensiones en C o C++, *Google App Inventor*, un entorno visual para programadores novatos y varios marcos de aplicaciones basadas en la web multiteléfono.

Google Play es la tienda en línea de software desarrollado por *Google* para dispositivos Android. Una aplicación llamada *"Play Store"* que se encuentra instalada en la mayoría de los dispositivos Android permite a los usuarios navegar y descargar aplicaciones publicadas por los desarrolladores. Android posee una gran comunidad de desarrolladores creando aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, se ha llegado ya al 1.000.000 de aplicaciones.

ARQUITECTURA

Los componentes principales del sistema operativo de Android son:

- **Aplicaciones:** las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- **Marco de trabajo de aplicaciones:** los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades. Este mecanismo permite que los componentes sean reemplazados por el usuario.
- **Bibliotecas:** Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios,

bibliotecas de gráficos, 3D y SQLite, entre otras.

- **Runtime de Android:** Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".

- **Núcleo Linux:** Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

CARACTERÍSTICAS

Las características y especificaciones actuales de Android son: [38]

- **Diseño de dispositivo**

  La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0 y diseño de teléfonos tradicionales.

- **Almacenamiento**

  SQLite, una base de datos liviana, que es usada para propósitos de almacenamiento de datos.

- **Conectividad**

  Android soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC, WiMAX, GPRS, UMTS y HSDPA+.

- **Mensajería**

  SMS y MMS son formas de mensajería, incluyendo mensajería de texto y ahora la Android Cloud to Device Messaging Framework (C2DM) es parte del servicio de Push Messaging de Android.

- **Navegador web**

  El navegador web incluido en Android está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome. El navegador por defecto de Ice Cream Sandwich obtiene una puntuación de 100/100 en el test Acid3.

- **Soporte de Java**

  Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik. Dalvik es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dipositivos móviles que funcionan con batería y que tienen memoria y procesador limitados. El soporte para J2ME puede ser agregado mediante aplicaciones de terceros como el J2ME MIDP Runner.

- **Soporte multimedia**

  Android soporta los siguientes formatos multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC,

HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

- **Soporte para streaming**

  Streaming RTP/RTSP (3GPP PSS, ISMA), descarga progresiva de HTML (HTML5 <video> tag). Adobe Flash Streaming (RTMP) es soportado mediante el Adobe Flash Player. Se planea el soporte de Microsoft Smooth Streaming con el port de Silverlight a Android. Adobe Flash HTTP Dynamic Streaming estará disponible mediante una actualización de Adobe Flash Player.

- **Soporte para hardware adicional**

  Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.

- **Entorno de desarrollo**

  Incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. Inicialmente el entorno de desarrollo integrado (IDE) utilizado era *Eclipse* con el plugin de *Herramientas de Desarrollo de Android* (ADT). Ahora se considera como entorno oficial *Android Studio*, descargable desde la página oficial de desarrolladores de Android.

- **Multitarea**

  Multitarea real de aplicaciones está disponible, es decir, las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj.

3.2.1.2 UBUNTU

Ubuntu es un sistema operativo basado en *GNU/Linux* que se distribuye como software libre [39], el cual incluye su propio entorno de escritorio denominado Unity.

Está orientado a los nuevos usuarios y a los de nivel medio, con un fuerte enfoque en la facilidad de uso y en mejorar su experiencia. Está compuesto de múltiple software normalmente distribuido bajo una licencia libre o de código abierto. Ubuntu es patrocinado por *Canonical*, una compañía británica que mantiene el software libre y gratuito, con esto la empresa es capaz de aprovechar los desarrolladores de la comunidad para mejorar los componentes de su sistema operativo. Extraoficialmente, la comunidad de desarrolladores proporciona soporte para otras derivaciones de Ubuntu, con otros entornos gráficos, como *Kubuntu*, *Xubuntu*, *Ubuntu MATE*, *Edubuntu*, *Ubuntu Studio*, *Mythbuntu*, *Ubuntu GNOME* y *Lubuntu*. [40]

Canonical, además de mantener Ubuntu, también provee de una versión orientada a servidores, Ubuntu Server; una versión para empresas, Ubuntu Business Desktop Remix; una para televisores, Ubuntu TV; otra versión para tablets, Ubuntu Tablet 11; y una para usar el escritorio desde teléfonos inteligentes, Ubuntu for Android.

CARACTERÍSTICAS

Ubuntu soporta oficialmente dos arquitecturas de hardware en computadoras personales y servidores: 32-bit (x86) y 64-bit (x86-64). Sin embargo, extraoficialmente, Ubuntu ha sido portado a más arquitecturas: **ARM**, PowerPC, SPARC e IA-64. A partir de la versión 9.04 se empezó a ofrecer soporte extraoficial para procesadores ARM. Al igual que la mayoría de los sistemas de escritorio basados en Linux, Ubuntu es capaz de actualizar a la vez todas las aplicaciones instaladas en la máquina a través de repositorios. Ubuntu está siendo traducido a más de 130 idiomas, y cada usuario puede colaborar voluntariamente, a través de Internet. Sus características más destacadas son: [41]

- **Ubuntu y la comunidad**

  Los usuarios pueden participar en el desarrollo de Ubuntu, escribiendo código,

solucionando bugs, probando versiones inestables del sistema, etc. Desde febrero de 2008 existe el sitio *Brainstorm* que permite a los usuarios proponer sus ideas y votar las del resto. También se informa de las ideas propuestas que se están desarrollando o están previstas.

- **Software incluido**

  Ubuntu posee una gran gama de aplicaciones para llevar a cabo tareas cotidianas, entretenimiento, desarrollo y aplicaciones para la configuración de todo el sistema. La interfaz predeterminada de Ubuntu es Unity y utiliza en conjunto las aplicaciones de GNOME. Existen otras versiones extraoficiales mantenidas por la comunidad, con diferentes escritorios, y pueden ser instalados independientemente del instalado por defecto en Ubuntu.

- **Aplicaciones de Ubuntu**

  Ubuntu es conocido por su facilidad de uso y las aplicaciones orientadas al usuario final. Las principales aplicaciones que trae Ubuntu por defecto son: navegador web *Mozilla Firefox*, cliente de mensajería instantánea *Empathy*, cliente de correo *Thunderbird*, reproductor multimedia *Totem*, reproductor de música *Rhythmbox*, gestor y editor de fotos *Shotwell*, administrador de archivos *Nautilus*, cliente de BitTorrent *Transmission*, cliente de escritorio remoto *Remmina*, grabador de discos *Brasero*, suite ofimática *LibreOffice*, lector de documentos PDF *Evince*, editor de texto *Gedit*, cliente para sincronizar y respaldar archivos en línea *Ubuntu One* (desarrollada por Canonical), y la tienda de aplicaciones para instalar/eliminar/comprar aplicaciones *Centro de software de Ubuntu* (también desarrollada por Canonical).

- **Seguridad y accesibilidad**

  El sistema incluye funciones avanzadas de seguridad y entre sus políticas se encuentra el no activar, de forma predeterminada, procesos latentes al momento

de instalarse. Por eso mismo, no hay un cortafuegos predeterminado, ya que supuestamente no existen servicios que puedan atentar a la seguridad del sistema. Para labores o tareas administrativas en la línea de comandos incluye una herramienta llamada *sudo* (de las siglas en inglés de *SwitchUser do*), con la que se evita el uso del usuario administrador. Posee accesibilidad e internacionalización, de modo que el sistema esté disponible para tanta gente como sea posible.

- **Organización del software**

  Ubuntu internamente divide todo el software en cuatro secciones, llamadas "componentes", para mostrar diferencias en licencias y la prioridad con la que se atienden los problemas que informen los usuarios. Estos componentes son: **main**, **restricted**, **universe** y **multiverse**. Por defecto se instalan paquetes de los componentes *main* y *restricted.* Los paquetes del componente *universe* de Ubuntu generalmente se basan en los paquetes de la rama inestable (Sid) y en el repositorio experimental de Debian.

  - **main:** Contiene solamente los paquetes que cumplen los requisitos de la licencia de Ubuntu, y para los que hay soporte disponible por parte de su equipo. Éste está pensado para que incluya todo lo necesario para la mayoría de los sistemas Linux de uso general. Los paquetes de este componente poseen ayuda técnica garantizada y mejoras de seguridad oportunas.
  - **restricted:** contiene paquetes soportados por los desarrolladores de Ubuntu debido a su importancia, pero que no está disponible bajo ningún tipo de licencia libre para incluir en main. En este lugar se incluyen los paquetes tales como los controladores propietarios de algunas tarjetas gráficas, como por ejemplo, los de ATI y NVIDIA. El nivel de la ayuda es más limitado que para main, puesto que los desarrolladores pueden no tener acceso al

código fuente.

- **universe:** contiene una amplia gama de programas, que pueden o no tener una licencia restringida, pero que no recibe apoyo por parte del equipo de Ubuntu sino por parte de la comunidad. Esto permite que los usuarios instalen toda clase de programas en el sistema guardándolos en un lugar aparte de los paquetes soportados: main y restricted.
- **multiverse:** contiene los paquetes sin soporte debido a que no cumplen los requisitos de software libre.

En este proyecto se utilizó Ubuntu como sistema operativo para el funcionamiento del *ODROID U3*, debido a que se requiere configurar aspectos puntuales de funcionamiento del sistema, para lo cual este sistema operativo contiene herramientas de revisión y control detallado. También debido a que se requieren programas adicionales, y todos éstos no están disponibles para Android.

3.2.2 OPENCV (OPEN SOURCE COMPUTER VISION LIBRARY)

Es un conjunto de librerías (biblioteca) que poseen una licencia BSD y por lo tanto es gratis, tanto para uso académico y comercial. Cuenta con interfaces de *C++*, *C*, *Python* y *Java*, y es compatible con *Windows*, *Linux*, *Mac OS*, *iOS* y *Android*. OpenCV está diseñado para una eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. Gracias a su optimización en C/C++, la biblioteca puede tomar ventaja de procesamiento multi-core. Ha sido habilitado con *OpenCL*, y puede tomar ventaja de la aceleración de hardware de la plataforma de computación heterogénea subyacente. Adoptado en todo el mundo, OpenCV tiene más de 47 mil personas en su comunidad de usuarios y el número estimado de descargas es superior a 9 millones.

En OpenCV se han desarrollado gran cantidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica.

OpenCV posee una estructura modular, lo cual significa que el paquete incluye librerías estáticas o compartidas. Los módulos principales son los siguientes: [42]

- **core:** Módulo compacto que define las estructuras de datos básicos, incluyendo arreglos multidimensionales de tipo *Mat* y funciones básicas utilizadas por todos los demás módulos.
- **imgproc:** Módulo de procesamiento de imagen que incluye imágenes no lineales y filtrado lineal, transformaciones geométricas (cambiar el tamaño, afín y la perspectiva de deformación, reasignación genérica basada en tablas), la conversión de espacios de color, histogramas, entre otros.
- **video:** Módulo de análisis de video que incluye Estimación de movimiento, sustracción de fondo, y algoritmos de seguimiento de objetos.
- **calib3d** Algoritmos básicos de vistas de geometría múltiple, la calibración individual y doble de cámaras, estimación deposición de objetos, algoritmos de correspondencia estéreo y elementos de reconstrucción 3D.
- **features2d** Detectores de rasgos sobresalientes, descriptores y comparadores descriptivos.
- **objdetect** Detección de objetos e instancias de clases predefinidas (por ejemplo, caras, ojos, las tazas, personas, vehículos, etc.).

- **highgui** Una interfaz fácil de usar para la captura de vídeo, imagen y codecs de vídeo, así como caacterísticas simples de interfaz de usuario.

- **gpu** Algoritmos acelerados por GPU para diferentes módulos OpenCV.

3.2.3 TESSERACT OCR

Tesseract es una librería para reconocimiento óptico de caracteres (OCR). Es considerado uno de los motor de OCR de código abierto más precisos disponible actualmente. Utilizando las librerías de procesamiento de imágenes de *Leptónica* [43], perimite leer una gran variedad de formatos de imagen y convertirlos a texto en más de 60 idiomas. Además permite ser entrenado para soportar lenguajes personalizados o específicos. Fue uno de los 3 mejores motores en la prueba de precisión 1995 UNLV. Entre 1995 y 2006 había poco trabajo hecho en él, pero desde entonces se ha mejorado ampliamente por *Google*.

Tesseract funciona en Linux, Windows (con VC++ Express o CygWin) y en Mac OSX. También puede ser compilado para otras plataformas, incluyendo Android y Iphone. Tesseract puede ser usada directamente como una aplicación, o también mediante su herramienta denominada *API* puede incluirse en proyectos de desarrolladores. Soporta una amplia variedad de lenguajes de programación. Tesseract no posee una interfaz gráfica de usuario, pero existen diversos proyectos de terceros que lo implementan.

3.2.4 QT CREATOR

Qt Creator es un completo IDE (entorno de desarrollo integrado) multiplataforma que se enfoca en el **Diseño de Interfaces Gráficas de Usuario** para proyectos C++ [44], que permiten una interacción del usuario con su aplicación. Está disponible para

los sistemas operativos Linux, Mac OSX y Windows. Qt Creator permite desarrollar aplicaciones e interfaces de usuario una sola vez y desplegarlas a través de varios sistemas operativos móviles y de escritorio. Proporciona las herramientas para llevar a cabo tareas a lo largo de todo el ciclo de vida de desarrollo de aplicaciones (figura 24), desde la creación de un proyecto hasta la implementación de la aplicación en las plataformas de destino.

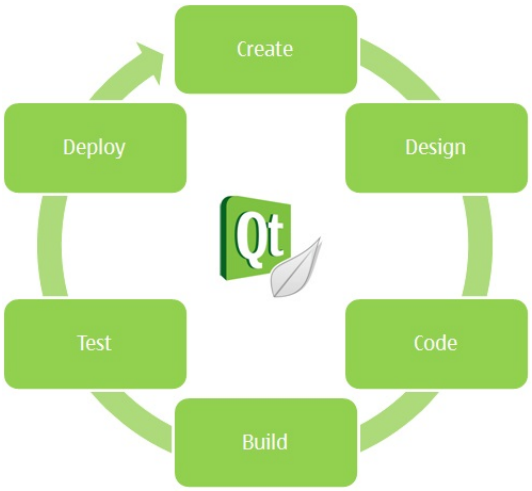


Figura 24: Ciclo de desarrollo de software en QT Creator. [44]

CAPÍTULO 4

DISEÑO E IMPLEMENTACIÓN

4.1 Entorno de Despliegue y Algoritmos

4.1.1 Proceso General del Sistema

Los componentes y el esquema de funcionamiento del sistema de detección e identificación vehicular se muestra en la figura 25. Como componente central se encuentra el ODROID U3, que es elemento de procesamiento y control; como elementos de entrada permanentes se encuentra la fuente de energía y la cámara para la adquisición de imágenes; como elemento de salida se encuentra la antena Wi-Fi para la conexión a una red local y envío de información mediante un access point; y como elemento de configuración se utiliza un computador, con el cual se conecta remotamente al ODROID para la puesta en marcha y configuraciones necesarias.

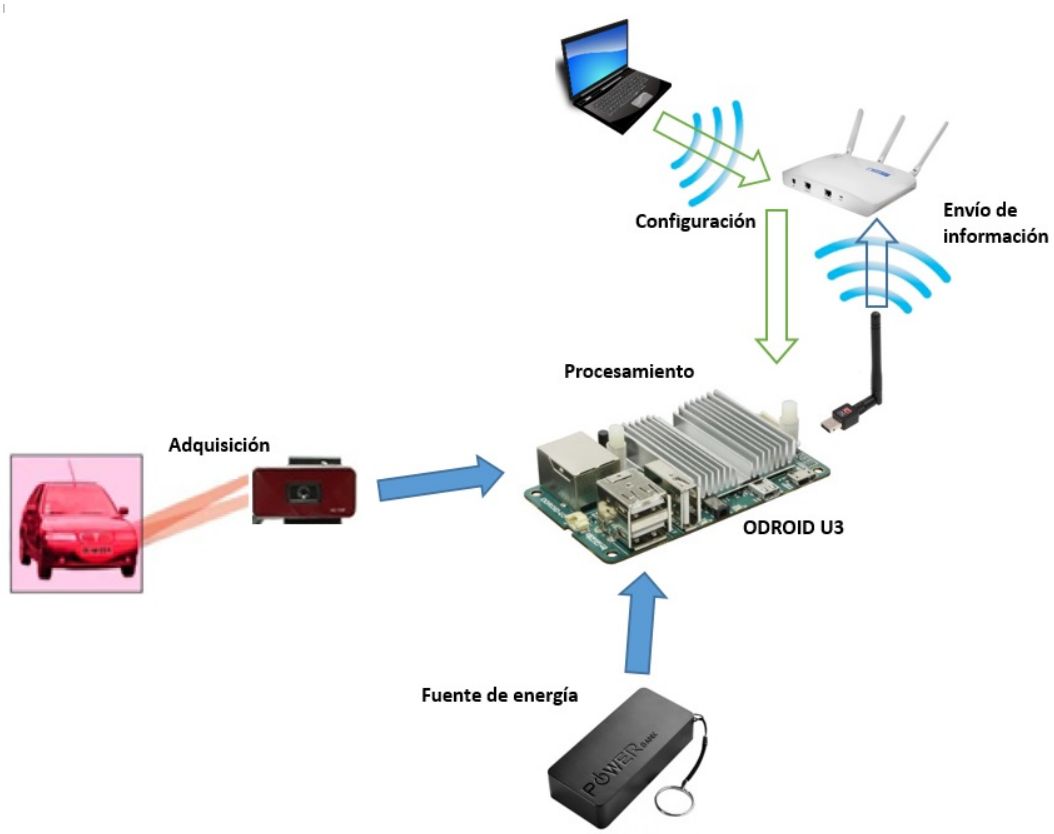


Figura 25: Componentes y esquema de funcionamiento.

El sistema está compuesto por subprocesos o módulos que deben seguir un determinado orden. Cada uno de estos pasos por sí solo es un proceso individual completo y complejo que proporciona información importante al sistema. En conjunto, le dan forma y unifican el proyecto como un solo sistema. En la figura 73 se muestra el proceso general desarrollado.

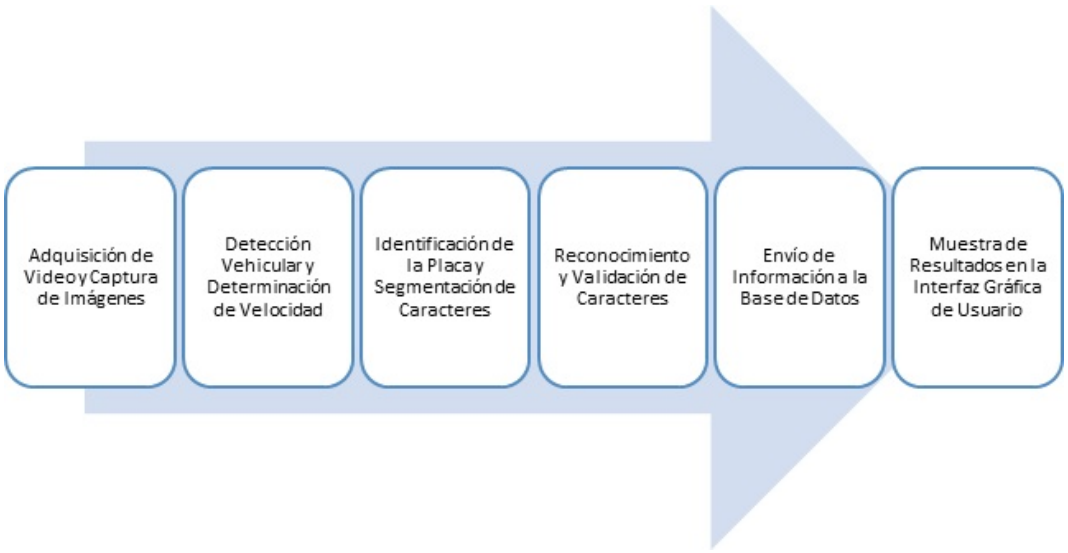


Figura 26: Proceso general del Sistema de Reconocimiento e Identificación Vehicular.

El primer paso es la Adquisición de video y captura de imágenes, una mala adquisición haría que todos los procesos siguientes presenten errores, produciendo un completo mal funcionamiento. El siguiente paso es la Detección Vehicular y Determinación de Velocidad, donde a partir de la imagen capturada se identifica la presencia de vehículos, y con las continuas interacciones se determina su velocidad instantánea. Después de identificar un vehículo se localiza qué lugar de la imagen corresponde a una placa vehicular; si se encuentra una, se la segmenta en regiones más pequeñas correspondientes a los caracteres individuales. Luego se realiza el OCR y se valida que la secuencia obtenida corresponda efectivamente a una placa. A continuación se envía esta información obtenida, junto con la fecha y hora del sistema, hacia una base de datos en una red local. Finalmente se muestra toda la información, incluyendo las fotografías del vehículo y de la placa detectada, en la interfaz de usuario.

4.1.2 Análisis del Entorno de Despliegue y Regiones de Interés

Para iniciar la elaboración de los procesos que componen el algoritmo general, es necesario realizar previamente un análisis de las imágenes de video que se va a obtener y posteriormente procesar para determinar la velocidad y placa del vehículo. Para ello se analizaron los siguientes factores:

- **Resolución de la imagen a obtener**

  En procesamiento de imágenes la resolución de las mismas es un factor determinante en los resultados obtenidos, tanto en la exactitud y porcentaje de errores como en el tiempo de procesamiento. Entre mayor tamaño tenga una imagen procesada, mayor será el tiempo que se requiera para su análisis, pero a cambio se pueden encontrar detalles que en imágenes pequeñas no se podría.

  Debido a que en el presente proyecto se utiliza un sistema embebido con capacidades de procesamiento limitadas, el tamaño de la imagen no debe ser demasiado grande. La cámara ODROID USB-WEBCAM tiene una resolución de 720P (1280 x 720), pudiéndose configurarla para resoluciones menores.

- **Selección del sitio para la ubicación del sistema**

  Para una primera implementación se requiere ubicar sitios estratégicos que proporcionen una vista superior de los vehículos en circulación, de manera que se facilite el reconocimiento del movimiento de los mismos a lo largo de la carretera, y que minimice las posibles interferencias en la imagen. Se considera que el sistema está bajo condiciones ideales cuando cumple los siguientes parámetros:

  - **Altura de la cámara.** Se determinó que una altura adecuada que proporciona una vista ideal para la detección vehicular es de 4.50 metros.

- **Distribución de elementos en la imagen.** En la figura 27 se muestra una vista superior ideal para la captura de imágenes, en la cual casi la totalidad de la imagen muestra la carretera, se puede apreciar el suelo que separa los vehículos entre ellos, y se tiene una distribución similar entre carriles.
- **Iluminación ambiental.** Debido a que la cámara empleada no posee visión infrarroja para su funcionamiento durante horas de la noche, y a que no se posee iluminación externa adicional, el sistema solamente puede operar correctamente durante el día, donde existe una iluminación adecuada.
- **Condiciones climáticas.** Las pruebas de funcionamiento del sistema se realizaron en días soleados, nublados y parcialmente nublados, durante los cuales no existieron factores que interfieran su funcionamiento. No se tiene registro de su desempeño bajo condiciones de lluvia, granizo o neblina, factores que además de intervenir en las condiciones físicas del sistema, añaden ruido significativo a las imágenes capturadas por la cámara.

Figura 27: Vista superior desde 4.50 metros de altura, ideal para detección vehicular.

En la figura 28 se muestra una vista no ideal para identificación de vehículos en circulación, ya que:

- 1. La carretera no ocupa la mayor parte de la imagen

- 2. No se aprecia la separación entre vehiculos
- 3. La distribución de carriles no es similar ya que un carril ocupa más espacio en la imagen que el otro.

Figura 28: Vista no ideal para detección vehicular.

- **Regiones de interés en la imagen**

  Teniendo como prioridad el tiempo de procesamiento debido a que se requiere un desempeño adecuado en tiempo real, se decidió emplear técnicas de segmentación asistida y selección de regiones de interés en la imagen para reducir el tamaño total a procesar, y con esto los tiempos de procesamiento.

  En la figura 29 se muestra las regiones que pueden ser consideradas de interés, pues contienen ubicaciones estratégicas en la imagen por donde continuamente se localizarán los vehículos a lo largo de la carretera.

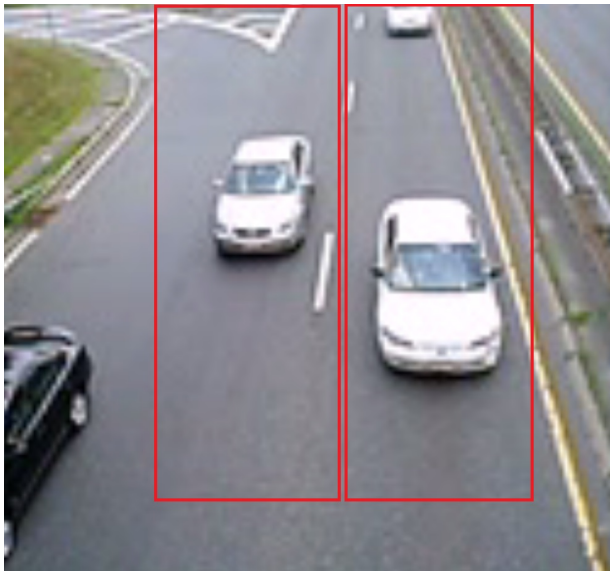


Figura 29: Regiones de interés en la imagen obtenida.

4.1.3 Implementación preliminar de algoritmos en Matlab

Con el objetivo de obtener una aproximación del funcionamiento del sistema, se implementaron algoritmos básicos de cada paso del proceso general utilizando *Matlab* en Windows, debido a que este software posee métodos que permiten una rápida implementación de técnicas complejas en procesamiento de imágenes. Al hacer esto se pudo evaluar distintos métodos para determinar los más adecuados para emplearse luego en el ODROID U3. No se realizó el desarrollo completo en esta plataforma debido a que no se puede ejecutar Matlab sobre sistemas embebidos, principalmente porque no es compatible con arquitecturas ARM; y también porque no se puede migrar con facilidad su código a lenguajes de programación compilados como C/C++.

El uso de Matlab sobre Windows permitió evaluar y valorar las técnicas utilizadas en el proceso general del sistema, solamente no se realizó el último paso que es la integración con la interfaz de usuario, puesto que esta no sería la plataforma de funcionamiento del proyecto final. Los resultados obtenidos de las pruebas en Matlab se

detallarán en el capítulo correspondiente a *Análisis de Resultados*.

4.2 Implementación de Algortimos en el *ODROID U3*

4.2.1 Instalación del Software Necesario

Los procesos de instalación a continuación descritos se realizaron sobre Ubuntu 14.04 para dispositivos ARM, pero deben funcionar de igual manera con otras distribuciones.

4.2.1.1 Instalación de OpenCV

Paquetes requeridos:

Previo a la instalación de OpenCV se necesitan paquetes complementarios, entre los que se incluyen los compiladores tanto para OpenCV como para los programas desarrollados con el mismo:

- GCC 4.4.x o posterior
- CMake 2.6 o posterior
- Git
- GTK+2.x o posterior, incluyendo las cabeceras (libgtk2.0-dev)
- pkg-config
- Python 2.6 y Numpy 1.5 o sus versiones posteriores con sus paquetes de desarrollo python-dev, python-numpy)

- Paquetes de desarrollo ffmpeg o libav: libavcodec-dev, libavformat-dev, libswscale-dev
- (opcional) libtbb2 libtbb-dev
- (opcional) libdc1394 2.x
- (opcional) libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev

Estos paquetes pueden ser instalados utilizando el terminal con los siguientes comandos:

Compilador:

```
sudo apt-get install build-essential
```

Requerido:

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

Opcional:

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

También se los puede instalar usando el *Gestor de Paquetes Synaptic de Ubuntu.*

El código fuente de OpenCV se lo puede descargar desde la página web:

`http://github.com/itseez/opencv`

o desde el terminal de Linux, mediante los comandos:

```
cd ~/<my_working _directory>
git clone https://github.com/Itseez/opencv.git
```

Generación de Librerías OpenCV Usando Cmake, Mediante Línea de Comandos

1.- Crear un directorio temporal, para fines explicativos se lo va a denominar como `cmake_binary_dir`, donde se alojarán todos los archivos generados.

```
cd ~/opencv
mkdir release
```

2.- Ingresar al directorio `cmake_binary_dir` y ejecutar la configuración en CMake:

```
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
```

3.- Ingresar a la carpeta temporal creada, y generar e instalar las librerías de OpenCV con los comandos:

```
make
sudo make install
sudo ldconfig
```

Instalación de OpenCV desde los Repositorios

Alternativamente, se puede instalar OpenCV mediante archivos precompilados desde los repositorios. Ubuntu 14.04 incluye repositorios de OpenCV, por lo que no es necesario añadirlos previamente. Esto se lo puede realizar con el siguiente comando:

```
sudo apt-get install opencv
```

Siguiendo estos pasos, OpenCV se instalará correctamente para su uso mediante programación en C/C++. Se puede tambien añadir compatibilidad para su programación con Python y Java. Debido a que este proyecto se lo realizó en C++ no se requirieron estos pasos adicionales.

4.2.1.2 Instalación de Tesseract

Primero se requiere instalar las librerías y herramientas para compilación:

```
sudo apt-get install libpng-dev libjpeg-dev libtiff-dev zlib1g-dev
sudo apt-get install gcc g++
sudo apt-get install autoconf automake libtool checkinstall
```

Nótese que `gcc` *y* `g++` *ya se instalaron previamente con OpenCV.*

Instalar *Leptonica* desde el código funete:

```
wget http://www.leptonica.org/source/leptonica-1.69.tar.gz
tar -zxvf leptonica-1.69.tar.gz
cd leptonica-1.69
./configure
make
sudo checkinstall
sudo ldconfig
```

Luego instalar *Tesseract OCR* desde el código fuente:

```
wget https://tesseract-ocr.googlecode.com/files/tesseract-ocr-3.02.02.tar.gz
tar -zxvf tesseract-ocr-3.02.02.tar.gz
cd tesseract-ocr
./autogen.sh
./configure
make (this may take a while)
sudo make install
sudo ldconfig
```

Finalmete, se debe instalar los idiomas que se desee reconocer. Para ello se debe colocar los archivos entrenados correspondientes (trained data) en la ubicación `/usr/local/share/tessdata`. Los archivos para varios idiomas están disponibles en la web de Tesseract, son los archivos con extensión `.traineddata`:

`https://code.google.com/p/tesseract-ocr/downloads/list`

ENTRENAMIENTO DE UN LENGUAJE PERSONALIZADO

Tesseract permite crear un lenguaje personalizado, es decir, con caracteres distintos a los que contienen sus archivos entrenados. En este proyecto se entrenó un lenguaje nuevo a partir de los distintos modelos y formas que presentan los caracteres de las placas vehiculares del Ecuador.

El proceso de entrenamiento y generación del archivo `.traineddata` se lo realizó en Windows, debido a que se encontraron herramientas que facilitaban este proceso, y además contaban con una interfaz gráfica de usuario. La API de entrenamiento que proporciona Tesseract y todos los pasos de configuración que se requieren, en Linux solamente se los puede realizar mediante línea de comandos.

El software que permite realizar facilmente este proceso es *Serak trainer for Tesseract*, proyecto que se lo encuentra en:

`https://code.google.com/p/serak-tesseract-trainer/`

Al ejecutar el programa se muestra la imagen de la figura 30. En la parte izquierda se puede ver una lista de pasos que se deben seguir para realizar correctamente el entrenamiento. Al seleccionar `File > New Project` se abrirá una ventana donde se pide ingresar la ubicación para guardar el proyecto.

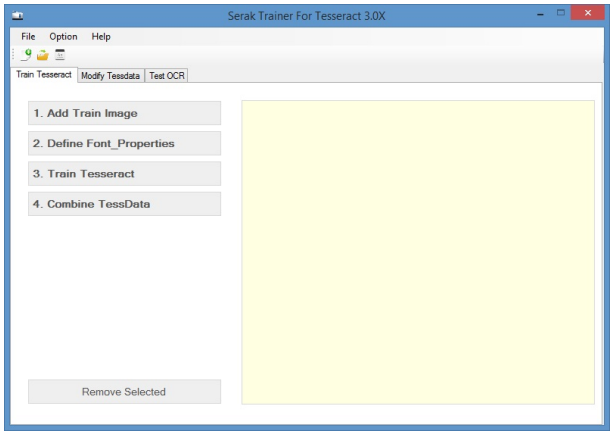


Figura 30: Serak Trainer for Tesseract.

El siguiente paso es la preparación de los archivos necesarios para el entrenamiento. Para esto se pueden utilizar las herramientas ***jTessBoxEditor*** y ***QT-BOX Editor***. Estos programas generan los llamados *Box files*, que son archivos que contienen las coordenadas de los cuadrados que encierran a los caracteres dentro de la imagen de entrenamiento (figura 31). ***jTessBoxEditor*** genera dos archivos con nombres que tienen el siguiente formato:

`lang.font_name.exp0.tiff`

`lang.fon_name.exp0.box`

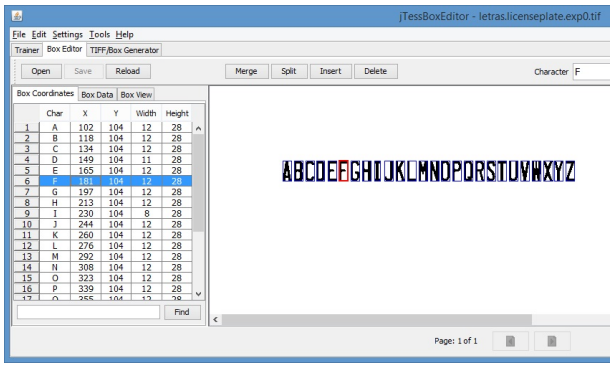


Figura 31: *Box-files* en *jTessBoxEditor*.

Se pueden descargar e instalar estos programas desde la siguientes direcciones web:

jTessBoxEditor: `http://vietocr.sourceforge.net/training.html`

QT-BOX Editor: `https://github.com/zdenop/qt-box-editor/downloads`

Seleccionando el paso ***1. Add Train Image***, se carga el archivo para el entrenamiento, aquí simplemente se ubica a la imagen `.tiff` generada en el paso anterior.

En el paso ***2. Define Font_Properties*** se indica las propiedades que tienen los caracteres a entrenarse, ya sea negrita, cursiva, etc. Seleccionar las que corresponda y

hacer clic en `Save changes and Close`.

A continuación se selecciona el paso ***3. Train Tesseract***, se mostrará una consola realizando el entrenamiento y una vez terminado aparecerá el mensaje *Training Complete*.

En el paso ***4. CombineTessData*** se mostrará nuevamente una consola seguida del mensaje *Creation of Tessdata is Succesfull*. Si no se presenta ningún mensaje de error, el proceso de entrenamiento ha finalizado correctamente.

En la ubicación donde se guardó el proyecto se creará una carpeta llamada `TessData` que contiene el resultado final del entrenamiento, que es el archivo `.trainedata`. Se debe copiar este archivo en la carpeta `/traindata` del directorio de instalación de Tesseract.

4.2.1.3 Instalación de Code::Blocks

Code::Blocks es un entorno de desarrollo integrado libre y multiplataforma para el desarrollo de programas en lenguaje C y C++. Está basado en la plataforma de interfaces gráficas *WxWidgets*, lo cual quiere decir que puede usarse libremente en diversos sistemas operativos.

Su instalación se la realizó desde la terminal, mediante los siguientes pasos:

- Abrir el terminal y escribir el siguiente comando:

```
sudo apt-get install codeblocks
```

- Nos pedirá ingresar la contraseña de super-usuario, la ingresamos.

  A continuación empezará la descarga de los paquetes necesarios para la instalación.

- Instalar el compilador.

```
sudo apt-get install g++
```

CREACIÓN DE UN PROYECTO EN CODE::BLOCKS

Para empezar la programación del código se requiere crear un proyecto, que a su vez permite la compilación directa desde el IDE, sin tener que utilizar la línea de comandos.

La primera vez que ejecutamos el IDE Code::Blocks veremos la imagen de la figura 32.

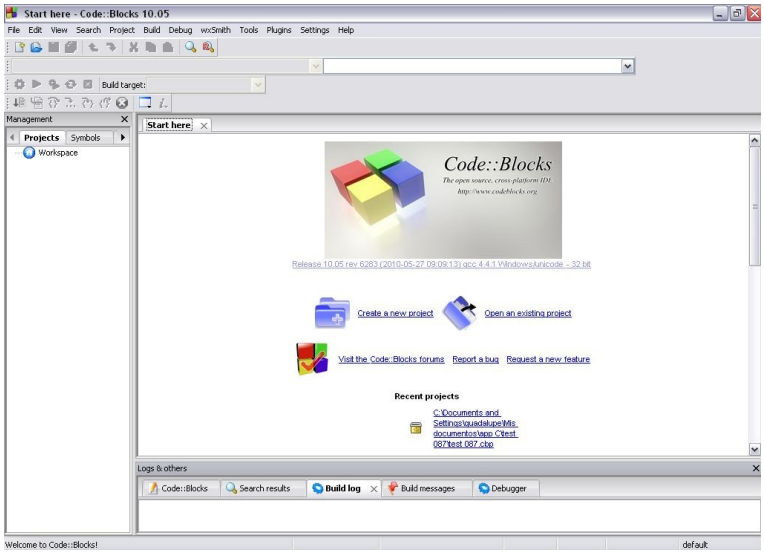


Figura 32: Pantalla de inicio de Code::Blocks.

Ahora, para crear un proyecto vamos a File > New > Project (figura 33).

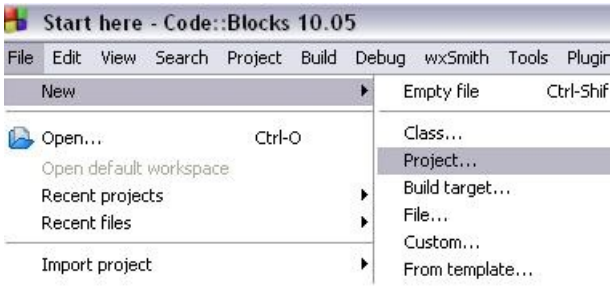


Figura 33: Creación de un proyecto en Code:Blocks.

A continuación aparecerá una ventana emergente como en la figura 34. Seleccionamos `Console application` y presionamos Go.

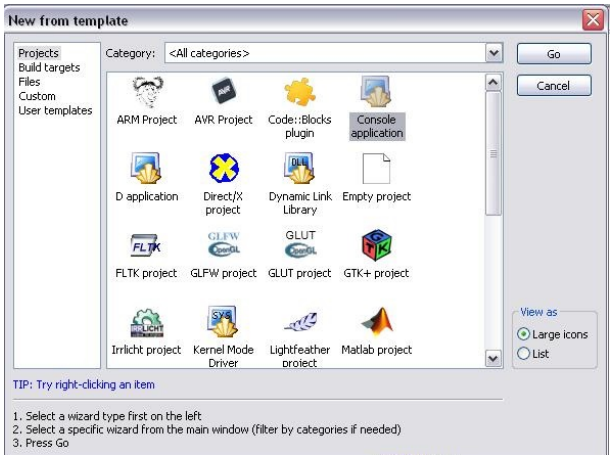


Figura 34: Selección de *Console Aplication* en Code:Blocks.

Presionamos `next` y nos aparecerá una ventana para elegir el lenguaje (figura 35).

Figura 35: Selección del lenguaje de programación en Code:Blocks.

Presionamos next e introducimos el título de la aplicación en `Project title`.

Nuevamente presionamos next. Aparecerá una ventana como en la figura 36. Aquí podemos configurar las opciones del espacio de trabajo y la ubucación donde se guardarán nuestros archivos. Presionamos `Finish`.

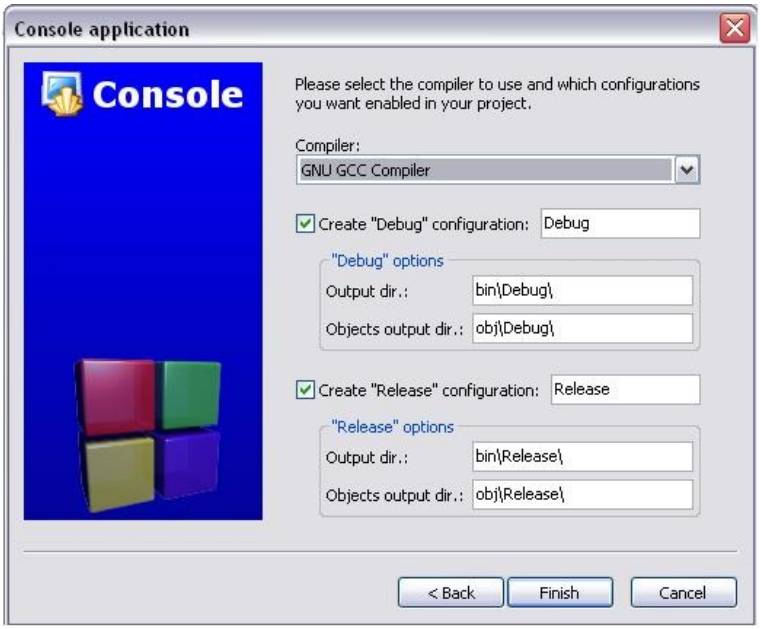


Figura 36: Configuración espacio de trabajo en Code:Blocks.

Abrimos la carpeta azul de la izquierda (figura 37) y seleccionamos `main.c` que será el archivo de codigo fuente de nuestra aplicación, para editarlo y posteriormente compilarlo. Para la compilación y ejecución del programa podemos presionar la tecla `F9`, también utilizando los íconos de la barra de herramientas (figura 37), mediante o el menu `Build` > `Build and Run`.

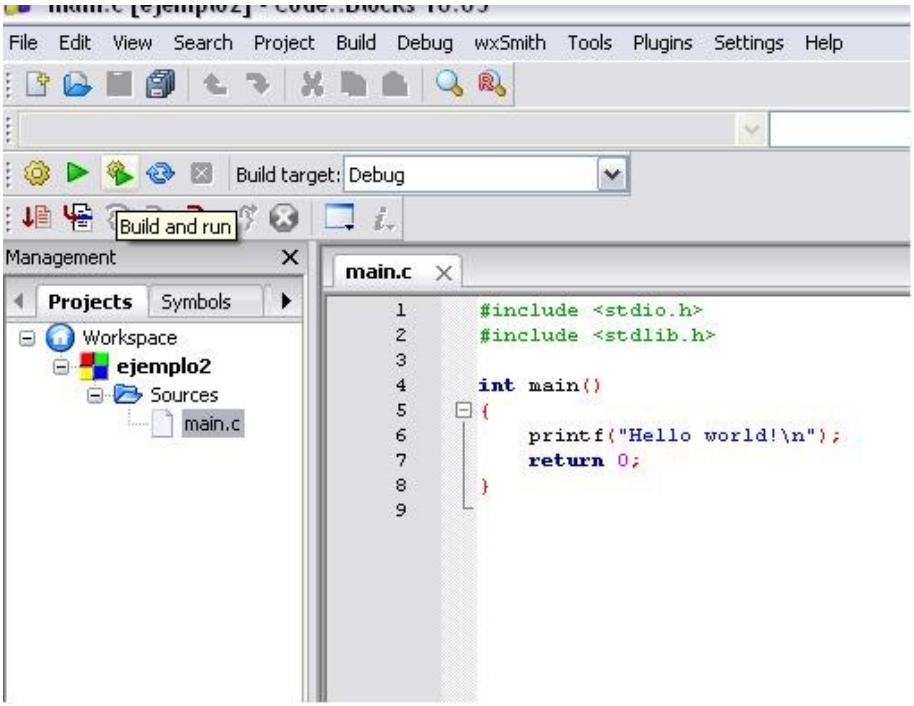


Figura 37: Compilar un proyecto en Code:Blocks.

4.2.1.4 Instalación de XAMPP

XAMPP es una distribución de Apache gratuita que contiene *MySQL*, *PHP* y *Perl*. En este proyecto se la utilizó para alojar la base de datos que guarda la información de los vehículos detectados. Este servidor estará en un computador bajo *Windows 8*, con lo cual se evidencia que el proceso de Tranmisión de Información en el sistema desarrollado es independiente de plataforma, y puede ser migrado hacia administraciones de bases de datos más complejas.

Pasos para su instalación:

- Ingresar a la página `http://www.apachefriends.org/en/xampp-windows.html` y descargar la versión adecuada para el sistema operativo.
- Al ejecutar el archivo descargado empezará el proceso de instalación, que sigue

los pasos de una instalación normal en Windows. En la ventana *Seleccionar Componentes* (figura 38) se debe seleccionar necesariamente los servidores `Apache` y `MySQL`, los demás servidores son opcionales.

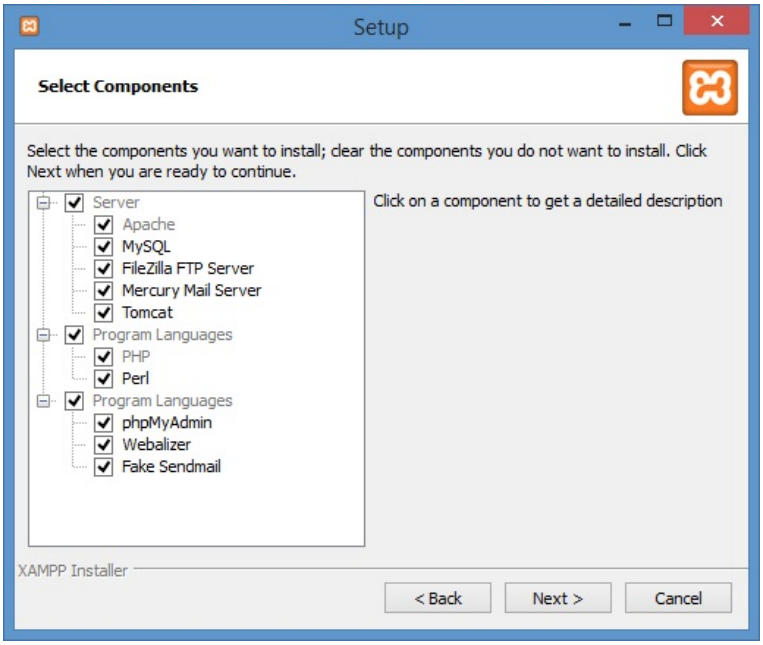


Figura 38: Instalación de componentes de XAMPP.

Luego se pide indicar la ruta de instalación, dejar la ubicación predeterminada `C:/xampp` y se continuará normalmente la instalación. Una vez finalizada, se debe abrir el *Panel de Control de Xampp* (figura 39) desde el escritorio o los programas de inicio.

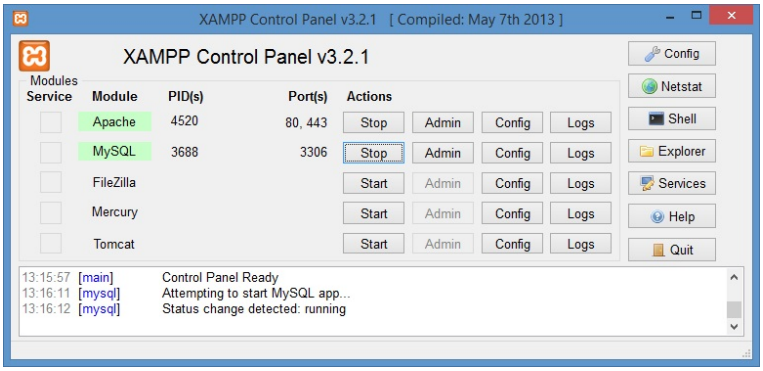


Figura 39: Panel de Control de XAMPP.

Luego se debe iniciar los servidores *Apache* y *MySQL* presionando en `Start` en la sección `Actions`. Si se inician correctamente, en la sección `Module` cambiará el color de fondo del nombre del servidor a verde, y en la parte de abajo también se indicará que el Servicio ha sido iniciado.

Para verificar que se puede acceder al servicio, en un navegador web se debe ingresar la dirección ip del computador donde se instaló Xampp, o desde el computador servidor se puede también ingresar las direcciones `127.0.0.1` ó `localhost`. Se abrirá en el navegador la página principal de Xampp (figura 40).

Figura 40: Página principal dell servidor XAMPP instalado.

4.2.2 Detección Vehicular

En la figura 41 se muestra el diagrama de flujo que se desarrolló para realizar la Detección vehicular y determinar la velocidad instantánea:

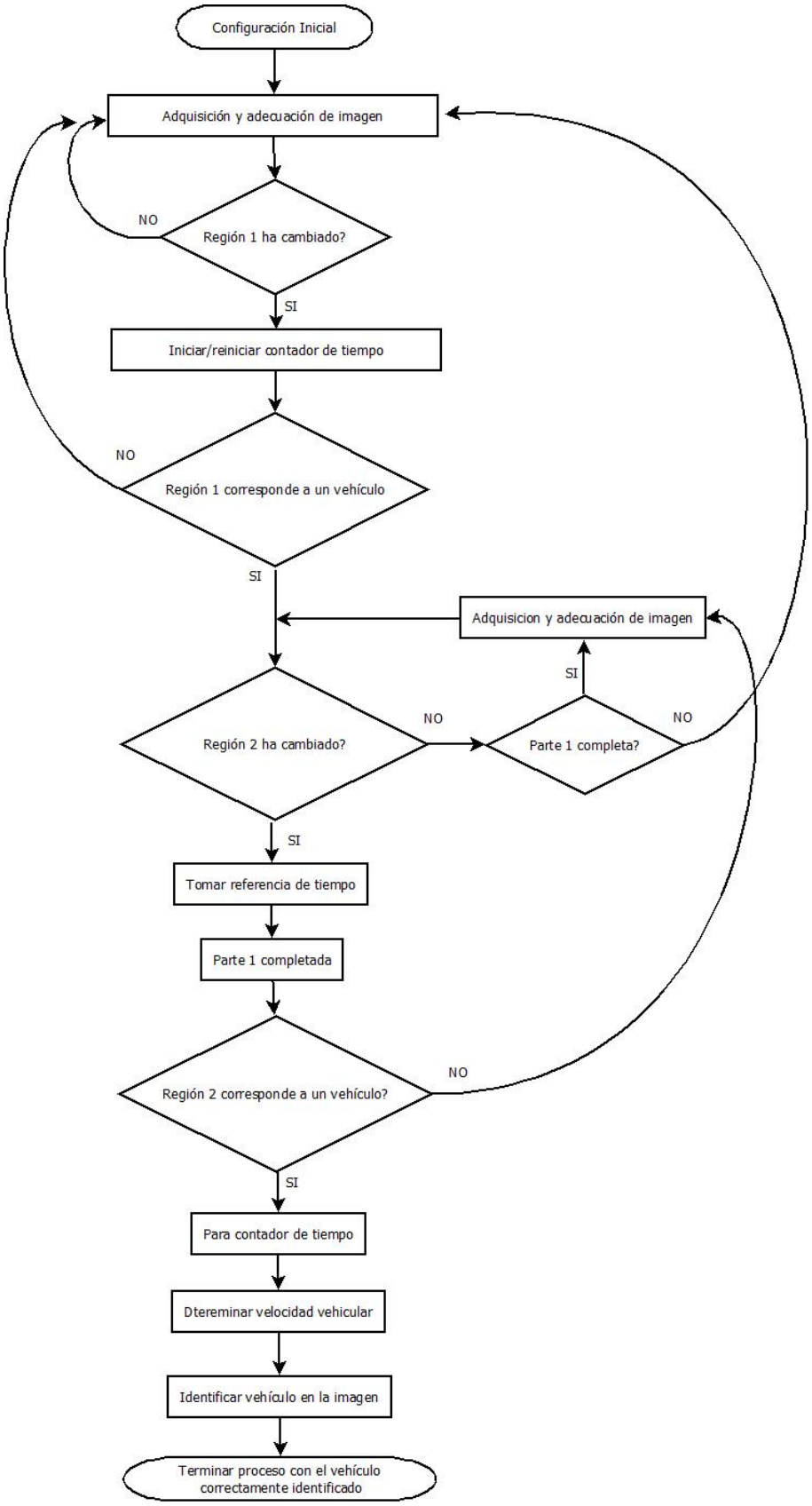


Figura 41: Diagrama de flujo del proceso de Detección vehicular y velocidad instantánea.

- **Configuración Inicial** Se cargan y configuran los parámetros que ingresa el usuario, como el número de carriles, velocidad máxima permitida, coordenadas de regiones de interés, necesarios para este proceso. En la sección 4.3 se detallan detenidamente estos parámetros.

- **Adquisición y adecuación de la imagen** Se obtiene la imagen desde la cámara de video, se aplica un filtro gaussiano uniforme de 3x3 para eliminar el ruido y resaltar las regiones de interés. Previo a la detección vehicular se obtienen las regiones de interés de fondo o de background, que corresponden a cuando no existen vehículos circulando. Esto se determina cuando al comparar diez imágenes consecutivas mediante la correlación, no se presentan valores menores a 0,98. Este proceso se repite constantemente con el objetivo de mantener el background actualizado, evitando que se produzcan falsas detecciones debido a cambios de iluminación a lo largo del día.

- **Región 1 ha cambiado?** Se verifica si la región de interés 1 ha cambiado con respecto a una imagen previa, lo que significa la presencia de un objeto en esa región. Se registra un cambio cuando la correlación entre la imagen de background y la nueva imagen presenta valores menores a 0,98 en la región de interés.

- **Iniciar/Reiniciar Contador de Tiempo** Se toma una referencia de tiempo inicial para posteriormente determinar el tiempo total y con esto, la velocidad instantánea. Esta referencia corresponde al número de ciclos que registra el procesador hasta ese momento.

- **Región 1 corresponde a un vehículo?** Se analiza si el objeto presente en la región 1 corresponde a patrones de un vehículo. Se considera como vehículo cuando el número de pixeles que cambia en la proyección vertical es mayor a 0,65 del ancho total del carril (figura 42). Caso contrario se considera que

el objeto no corresponde a un vehículo, pudiendo ser una persona, bicicleta, motocicleta, o la sombra de otro vehículo que circula por el carril contiguo.

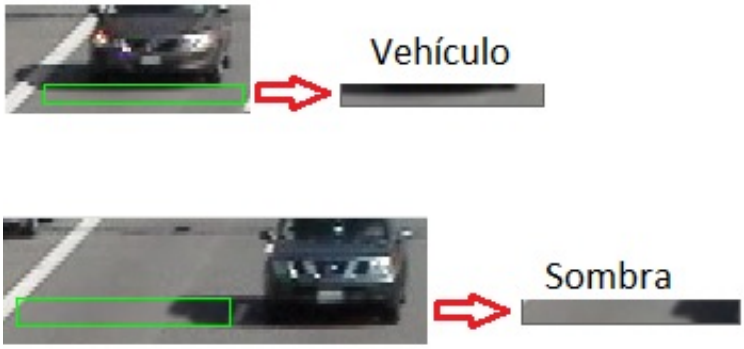


Figura 42: Cambios en las regiones de interés.

- **Región 2 ha cambiado?** Se verifica si la región de interés 2 ha cambiado con respecto a una imagen previa, lo que significa la presencia de un objeto en esa región. De igual manera, se registra un cambio cuando la correlación entre la imagen de background y la nueva imagen presenta valores menores a 0,98 en la región de interés.

- **Tomar referencia de tiempo** Se toma una segunda referencia de tiempo, que corresponde al tiempo trascurrido desde que el vehículo pasó por la región de interés 1 hasta la región de interés 2. El número de ciclos que registra el procesador será mayor a la primera referencia tomada.

- **Parte 1 completa** Cuando el vehículo ha llegado a la región de interés 2, se toma como terminada la primera parte de este proceso.

- **Región 2 corresponde a un vehículo?** Se analiza si el objeto presente en la región 2 corresponde a patrones de un vehículo, con los mismos parámetros que para la región de interés 1 previamente descritos. Si el análisis es positivo, se toma como válida la referencia de tiempo, obteniendo el tiempo total de recorrido al restar la referencia de tiempo 1 de la referencia de tiempo 2, y dividir este

valor para la frecuencia de operación del procesador. Conocida previamente la distancia entre las marcas y determinado el tiempo que se requiere para cruzarlas, se puede calcular la velocidad instantánea.

- **Identificar vehículo en la imagen** Se identifica la región que corresponde al vehículo, recortando la zona superior de la región de interés 2 (figura 43) y se la envía al siguiente proceso para la identificación de la ubicación de la placa.

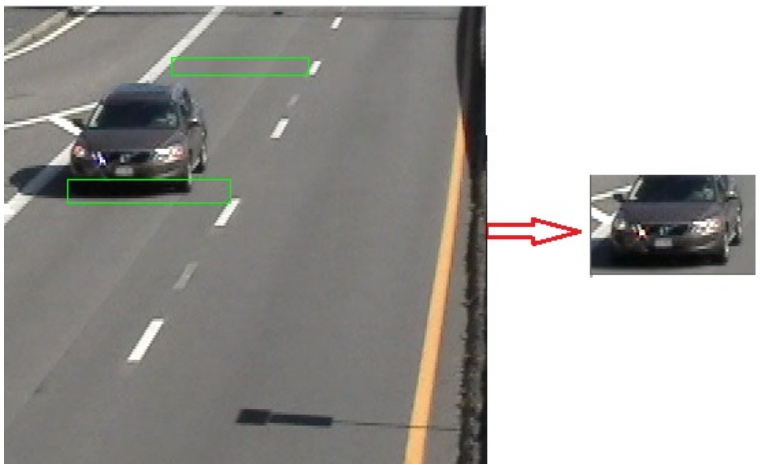


Figura 43: Imagen del vehículo detectado.

4.2.3 Identificación de la placa vehicular

En la figura 44 se muestra el diagrama de flujo que se desarrolló para realizar la Identificación de la zona de la placa:

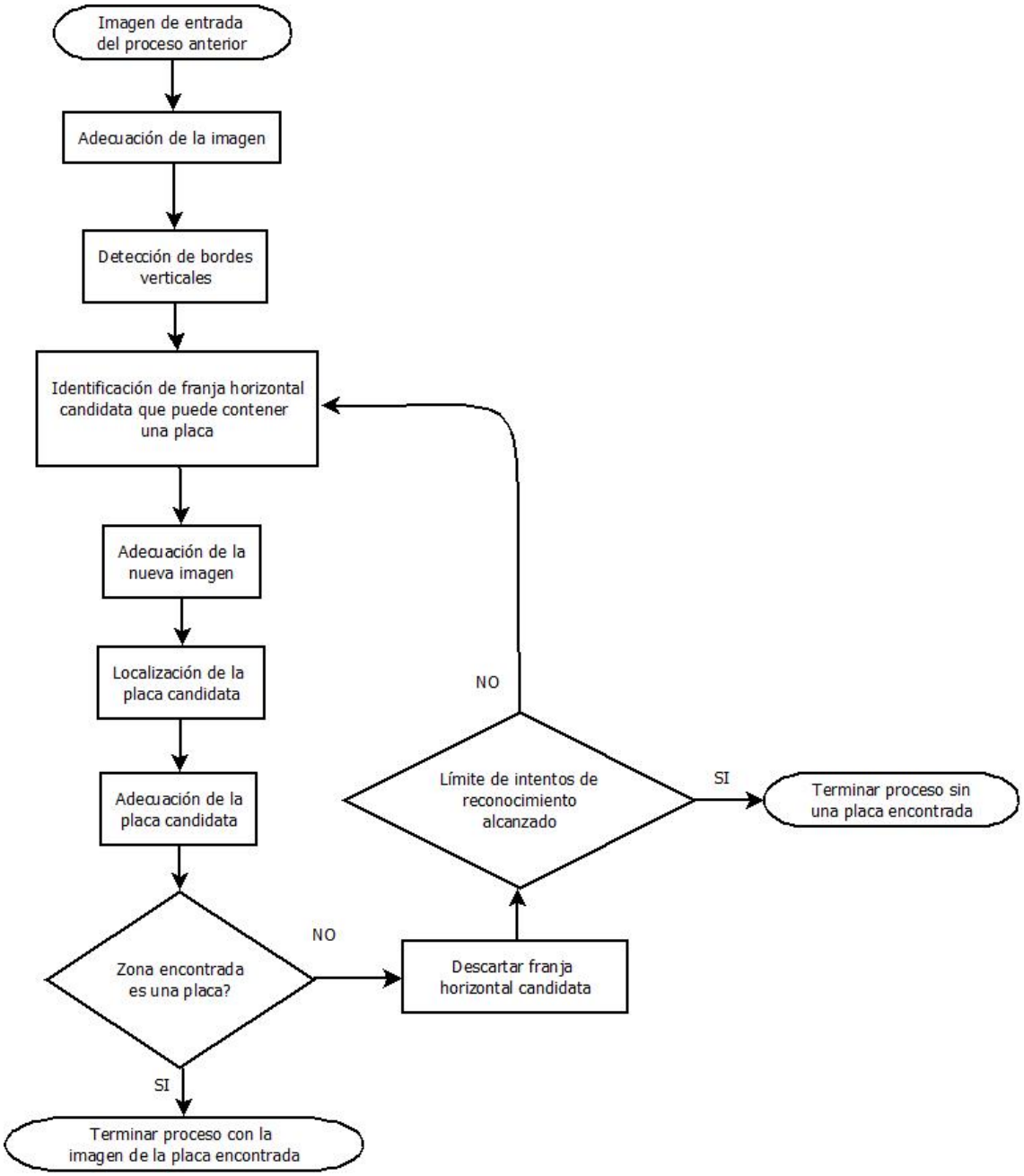


Figura 44: Diagrama de flujo del proceso de Identificación de la zona de la placa.

- **Adecuación de la imagen** Se aplica un filtro gaussiano uniforme de 3x3 para eliminar el ruido de toda la imagen, y otro filtro de media de 1x3 para enfatizar las regiones verticales y a la vez reducir las regiones horizontales pequeñas, con el objetivo de obtener posteriormente una correcta detección de bordes ver-

ticales.

- **Detección de bordes verticales** Se aplica a la imagen un filtro de sobel y se toma solamente los valores mayores a 0.3 veces el máximo valor en la imagen obtenida. El resultado se muestra en la figura 45, en la cual se aprecia que resalta la zona donde se ubica la placa.

Figura 45: Detección de bordes verticales.

- **Identificación de franja horizontal candidata que puede contener una placa** A partir de los bordes verticales se realiza la proyección horizontal, los picos que se presenten en la misma son regiones candidatas donde posiblemente se encuentre la placa. En la figura 46 se tienen dos filas candidatas que deberán ser analizadas para determinar cuál corresponde efectivamente a la placa.

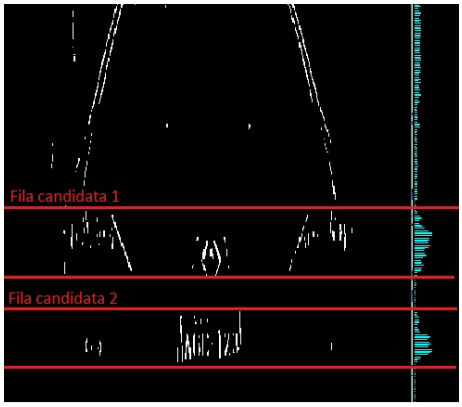


Figura 46: Proyección horizontal de los bordes verticales.

- **Adecuación de la nueva imagen** En la imagen de la fila candidata se realiza nuevamente una detección de bordes verticales pero con parámetros distintos al primer proceso, ya que se analiza una zona más pequeña y se requiere enfatizar los rasgos que corresponden a los caracteres de la placa. A continuación se obtiene la proyección vertical, a la cual se aplica un filtro gaussiano para obtener una distribución uniforme en la que se enfatiza la zona de la placa (figura 48). Con esto se obtiene específicamente la ubicación en la fila candidata donde se encuentra la placa.

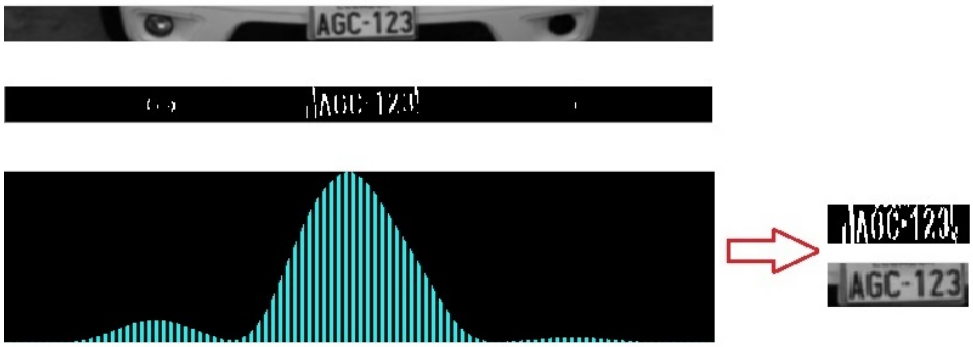


Figura 47: Proyección vertical de la fila candidata y ubicación de la placa.

- **Adecuación de la placa candidata** Se umbraliza y se binariza la imagen, para posteriormente eliminar las regiones adyacentes a los bordes ya que los caracteres se encontrarán en el centro, y también las zonas muy pequeñas como el guión u otras que se presentan debido a manchas en la placa que no corresponden a caracteres (figura **??**).

Figura 48: Umbralización, eliminación de bordes y de zonas pequeñas.

- **Zona encontrada es una placa?** Se evalúa si la zona final encontrada presenta características de una placa: determinando que el número de regiones que contiene, que representan los caracteres de la placa, no sea menor a cuatro ni mayor a siete; determinando en sus dimensiones, que el ancho sea mayor que el alto de la placa en no más de 10 veces (zona muy alargada) y no menos de 0.75 veces (zona tendiendo a cuadrado). Si el resultado es positivo se termina el proceso obteniendo la imagen de la placa; si es negativo, se descarta la franja analizada, y si aún no se ha alcanzado el límite de intentos de reconocimiento se repite el proceso con la siguiente fila candidata, caso contrario se lo termina sin haber encontrado una placa.

4.2.4 Reconocimiento de Caracteres de la Placa

En la figura 49 se muestra el diagrama de flujo que se desarrolló para realizar el Reconocimiento de caracteres de la Placa:

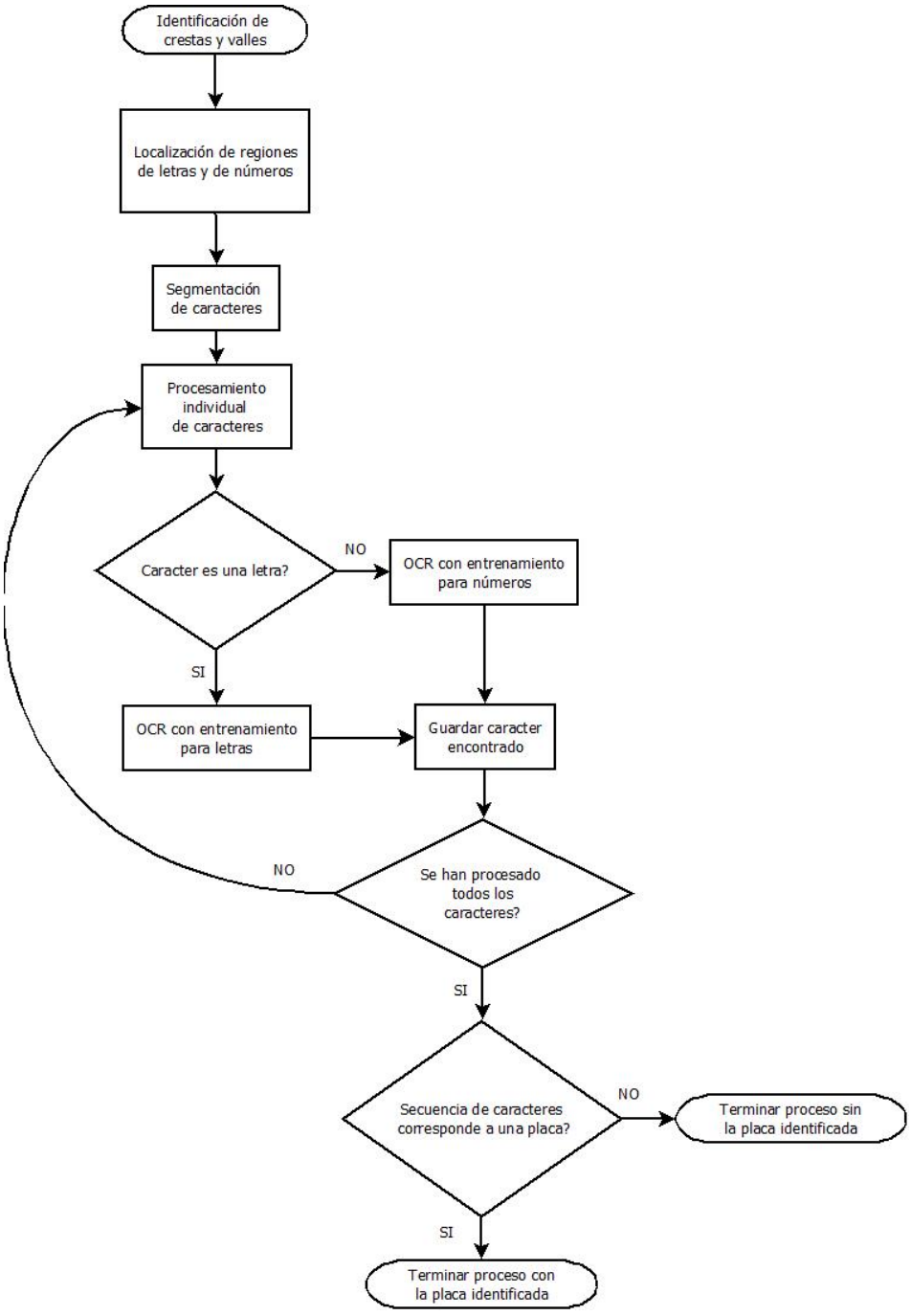


Figura 49: Diagrama de flujo del proceso de Reconocimiento de caracteres de la placa.

- **Identificación de crestas y valles** Se realiza la proyección vertical de la placa, en la cual las crestas o picos corresponden a los caracteres, y los espacios o sitios con poca concurrencia entre éstos representan los valles (figura 50).

Figura 50: Proyección vertical de la placa, identificación de crestas y valles.

- **Localización de regiones de letras y números** Las características de las placas de vehículos ecuatorianos permiten separar las letras de los números sin la necesidad de un primer proceso de OCR. En la figura 51 se aprecia que en el centro de la placa existe un valle que es mayor a los demás, que corresponde al guión eliminado previamente. La parte izquierda a este espacio corresponde solamente a letras, y la parte derecha solamente a números. Al realizar previamente esta división, se garantiza que en el proceso de OCR no habrán errores de reconocimiento de números por letras y viceversa (como por ejemplo el número 8 por la letra B, o el 5 por la S), por lo que no se requiere una posterior validación.

Figura 51: Identificación de zonas de letras y números en las placas ecuatorianas.

- **Segmentación de caracteres y procesamiento individual** Ubicando las zonas de los picos en la proyección vertical de la placa, se separan los caracteres y se busca la región que contiene a cada uno.

- **Caracter es una letra?** Se realiza el OCR correspondiente para letras o números

y se almacenan todos los caracteres identificados en una sola matriz, formando nuevamente la placa.

Una vez que se han procesado todos los caracteres se evalúa si la secuencia identificada corresponde a una placa, es decir, si posee tres letras y tres o cuatro números; si el resultado es positivo se termina el proceso con la placa identificada, caso contrario se termina con un resultado negativo.

4.2.5 Envío de Información a la Base de Datos

En la figura 52 se muestra el diagrama de flujo que se desarrolló para realizar el Envío de información a la base de datos:

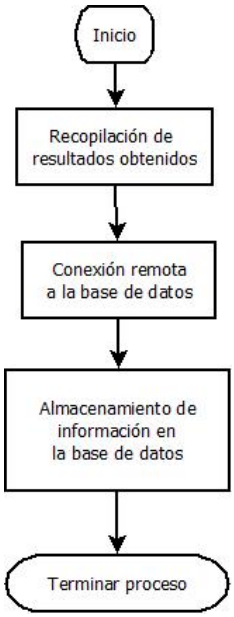


Figura 52: Diagrama de flujo del proceso de Envío de información a la base de datos.

Es preciso indicar que para este proceso se realizaron operaciones básicas de Recopilación de resultados, Conexión remota y posterior Almacenamiento de datos. Debido a que el tema central no se enfoca en parámetros de Bases de Datos, el objetivo

de este módulo es brindar al sistema comunicación con otros dispositivos, en este caso mediante una red local.

4.3 Interfaz Gráfica de Usuario (GUI)

Una Interfaz Gráfica permite interactuar a los usuarios con la aplicación en ejecución. Es una parte muy importante en sistemas que requieren constante entrada o salida de información visual hacia el usuario, ya que si esta interfaz falla, el proceso completo podría quedar inservible o presentar resultados erróneos.

El presente proyecto tiene un módulo de GUI que permite al usuario la visualización de resultados, sin embargo, no es un proceso vital para el funcionamiento del sistema. El GUI se lo utiliza puntualmente en la configuración inicial, y eventualmente para corregir posibles descalibraciones. Una vez configurado correctamente, el sistema puede realizar el reconocimiento e identificación de vehículos sin la necesidad de una interfaz gráfica.

La interfaz gráfica se la realizó empleando QT creator. Su instalación se la realizó desde la línea de comandos:

```
sudo apt-get install qt-creator
```

Al haber instalado las dependencias para los programas anteriormente descritos (OpenCV, Tesseract, Code::Blocks) no se requieren más componentes adicionales.

CREACIÓN DE UN PROYECTO EN QT-CREATOR

- Seleccionar `File` > `New File or Project` (o presionar Ctrl+N).
- En `Projects` elegir `Applications` y seleccionar `Qt Widgets Application`, presionar `Choose...` (figura 53)

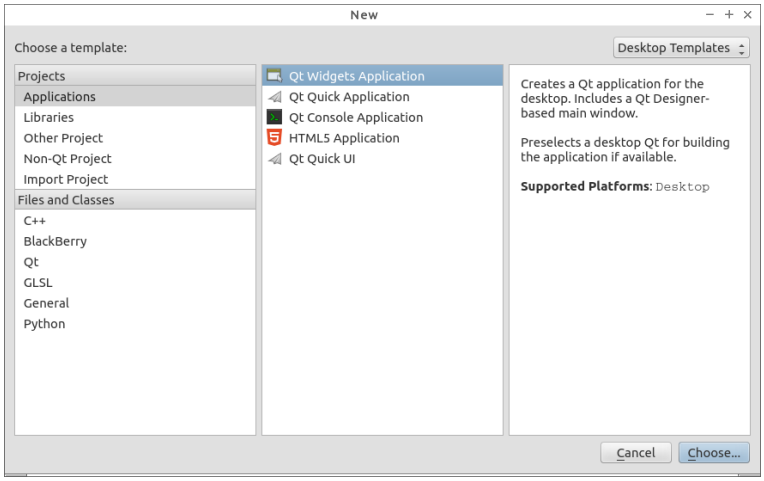


Figura 53: Creación de un proyecto en *QT-Creator*

- Ingresar el nombre del proyecto y seleccionar su ubicación, clic en `Next`
- Dejar como predeterminadas las demás opciones, hacer clic en `Next` hasta que se presente la pantalla principal para empezar a escribir el código (figura 54).

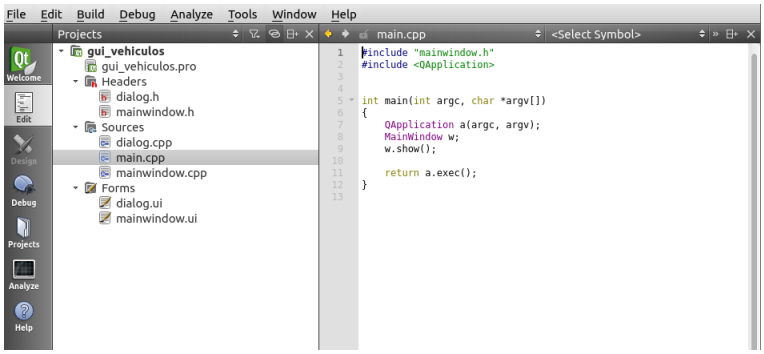


Figura 54: Pantalla principal de *QT-Creator*

VINCULAR OPENCV Y TESSERACT AL PROYECTO

En el explorador de carpetas del proyecto (figura 55), abrir el archivo `.PRO` y añadir las siguientes líneas antes de `SOURCES`:

```
INCLUDEPATH += /usr/local/include/opencv
LIBS += -L/usr/local/lib -lopencv_core -lopencv_imgcodecs -lopencv_highgui
```

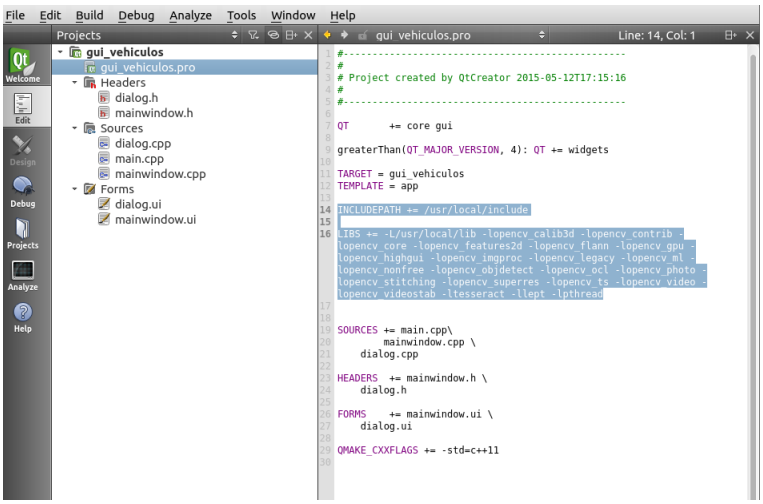


Figura 55: Explorador de carpetas de *QT-Creator*

Con esto Qt-Creator utiliza las librerías de OpenCV y Tesseract para compilar sus programas. Una vez guardados los cambios ya se puede escribir y compilar código en Qt. En este paso se unen los algoritmos de los procesos anteriormente descritos, y se los vincula para su funcionamiento mediante una interfaz de usuario.

DISEÑO DE LA INTERFAZ GRÁFICA DE USUARIO

Qt-Creator permite diseñar fácilmente GUIs, posee Widgets predeterminados de botones, etiquetas, cuadros de texto, menús interactivos, y practicamente cualquier componente que se requiera para realizar las interfaces más complejas. En la GUI del proyecto se utilizaron dos ventanas, una para configuración (figura 56) y otra para mostrar los resultados del reconocimiento (figura 57).

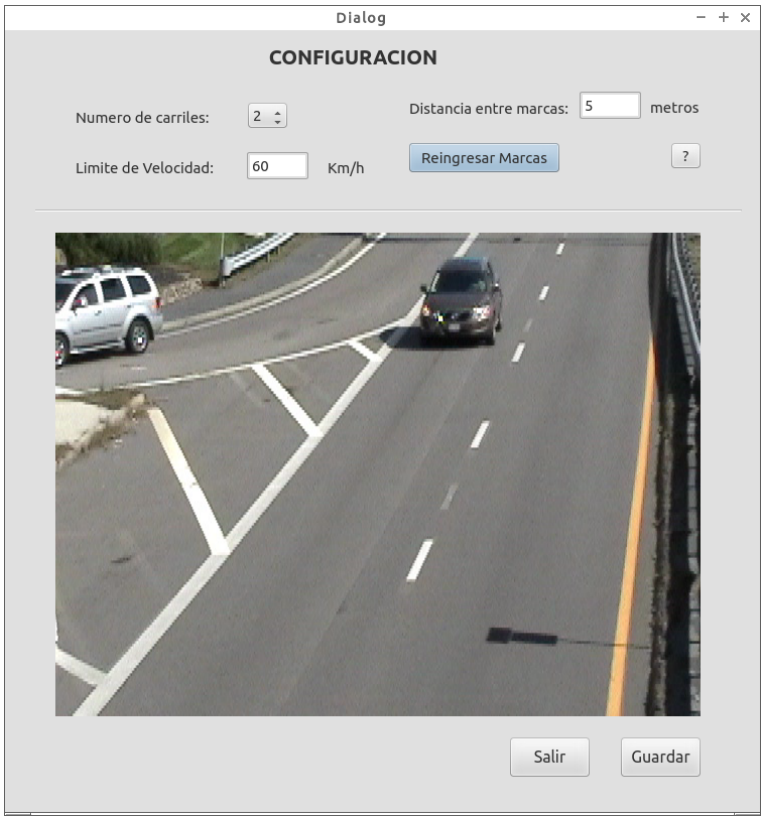


Figura 56: Ventana de configuración del GUI.

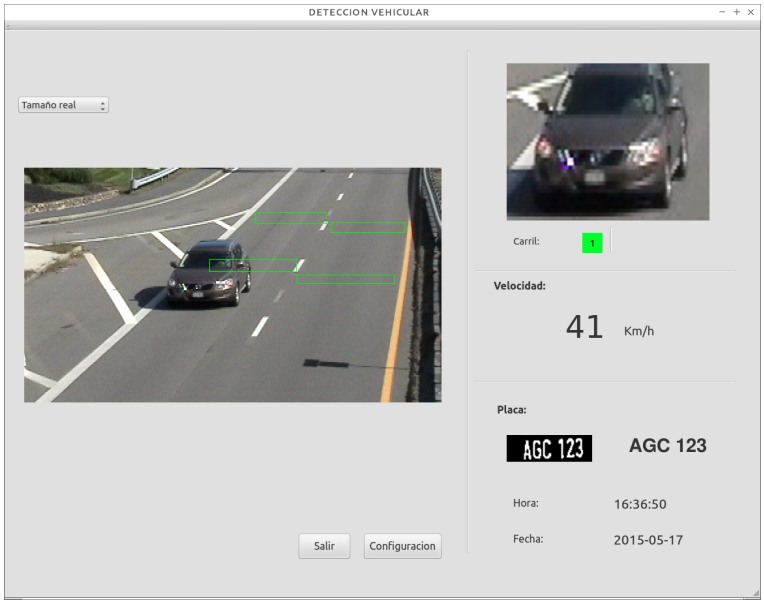


Figura 57: Ventana de presentación de resultados del GUI.

4.3.1 Manual de Usuario

Al iniciar la aplicación por primera vez se mostrará la ventana de CONFIGURACIÓN de la figura 56. Como no tiene ninguna configuración previa, el video se muestra sin nunguna marca de regiones de interés. Esta ventana muestra tres parámetros que pueden ser modificados por el usuario: el *número de carriles*, el *límite de velocidad* y la *distancia entre marcas*. Bajo de esta opción se encuentran dos botones: *Reingresar Marcas* y el botón de *Ayuda (?)*. Al seleccionar este último se mostrará una ventana (figura 58) con los pasos y el orden recomendado para la configuración del sistema.

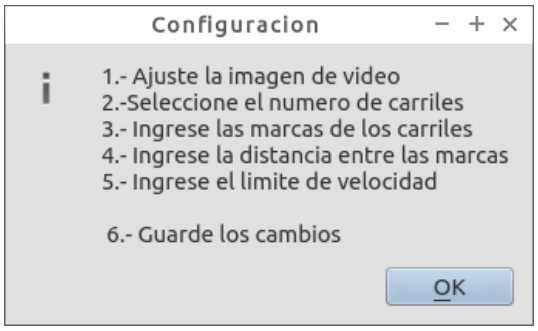


Figura 58: Pasos de configuración.

Siguiendo este orden, los pasos para la configuración son:

1. **Ajustar la imagen de video** hasta obtener una vista adecuada de la zona vehicular, y mantener la cámara en esa posición.

2. **Seleccionar el número de carriles** La aplicación fue diseñada para detectar vehículos solamente en 1 o 2 carriles, por lo que se presentarán solamente estas dos opciones. Al seleccionar un número de carril diferente al que se encuentre seleccionado, se abrirá automáticamente la ventana para seleccionar nuevas marcas en los carriles.

3. **Ingresar las marcas de los carriles** Al presionar el botón *Reingresar Marcas* se muestra un mensaje indicando que se debe realizar dos marcas por carril; dependiendo del número de carriles seleccionados en el paso anterior se podrán ingresar dos o cuatro marcas (1 o 2 carriles).

   Al presionar el botón *OK* se muestra una nueva ventana con la imagen actual de la cámara. Para realizar las marcas en el primer carril se utilizará el mouse haciendo clic y arrastrando a modo de selección, ubicando la primera región por donde pasarán los vehículos (figura 59). Una vez que se tenga correctamente señalada una marca, se debe presionar la letra *Q* para guardar la selección y poder indicar la segunda región por donde pasarán los vehículos. De la misma forma se debe presionar *Q* al finalizar la selección.

Figura 59: Ubicación de marcas en los carriles.

   Si se indicó que son dos carriles, se debe realizar la selección de dos nuevas marcas correspondientes al segundo carril. Al finalizar la ubicación de todas las marcas se mostrará un mensaje indicando que las nuevas ubicaciones fueron

guardadas correctamente.

4. **Ingrese la distancia entre las marcas** Este valor debe ser la distancia real de la carretera que existe entre la primera y segunda marca, la cual se debe conocer previamente. La distancia se medirá entre las líneas superiores de las marcas, como se indica en la figura 60.

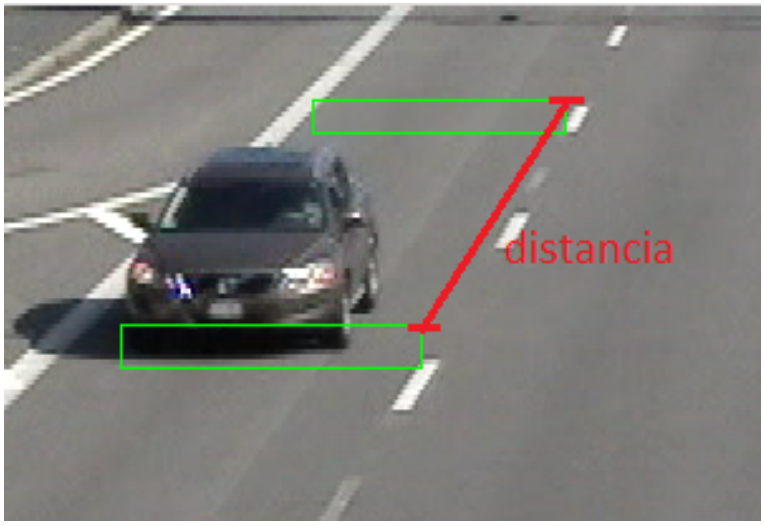


Figura 60: Medición correcta de la distancia entre marcas.

5. **Ingrese el límite de velocidad** Es la velocidad a partir de la cual el sistema identificará al vehículo como infractor.

6. **Guardar los cambios** Debe siempre guardarse los cambios antes de salir de la ventana de configuración para que estos tengan efecto. Después de ingresar nuevas marcas la interfaz guarda automáticamente los cambios. Esto en cambio no sucede cuando se cambie el valor de *distancia* o *velocidad máxima*.

   La información de configuración es guardada en un archivo de texto, y es cargada nuevamente cada vez que se vuelve a abrir la aplicación, evitando que se deba configurar nuevamente todo el sistema cada vez que se lo inicia.

La ventana de visualización de resultados se muestra en la figura 57. En la parte

izquierda se presenta el video con las respectivas marcas en los carriles, y sobre éste se encuentra un selector que tiene dos opciones: *Tamaño Real* y *Tamaño Ajustado(640x480)*. Mediante este selector se puede ajustar el tamaño de visualización de la imagen, si ésta es muy grande no se la podrá visualizar correctamente, y se deberá seleccionar la opción *Tamaño Ajustado*.

En la parte derecha se muestra la imagen del vehículo identificado y el número del carril por el que está circulando; la velocidad instantánea en Km/h; la imagen de la placa identificada; el número de placa reconocido; la hora y la fecha del reconocimiento.

Finalmente, el botón *Configuración* abre la ventana de configuración previamente detallada, y el botón *Salir* cierra la aplicación.

4.4 Implementación del Sistema para Pruebas de Funcionamiento

Para realizar las pruebas de funcionamiento se utilizó el soporte metálico de la figura 61. Este soporte consta de un tubo plegable que se sostiene sobre una base y alcanza alturas desde 1.70 hasta 4.50 metros, sobre el cual sobresale horizontalmente otro tubo de 1.50 metros, que en su extremo sostiene una caja metálica, dentro de la cual se coloca la tarjeta de procesamiento Odroid U3, la cámara web, la antena wifi y una batería de 5V/10400mAh (figura 62). Al ser el soporte desarmable, permite su movilidad para realizar pruebas de funcionamiento en distintos sitios de circulación vehicular.

Figura 61: Soporte metálico para pruebas de funcionamiento.

Figura 62: Caja metálica con los dispositivos para las pruebas de funcionamiento.

Debido a que el sistema se encuentra en exteriores y además se ubicará en alturas de hasta 4.5 metros, no se puede conectar una pantalla, mouse y teclado para la puesta en marcha y control del sistema. Para solucionar este inconveniente se accedió al Odroid U3 a través de otra computadora desde la red en la que se encuentra conectado, utilizando el protocolo Secure SHell, o intérprete de órdenes segura SSH [45], el cual permite manejar por completo un dispositivo mediante un intérprete de comandos, permitiendo también visualizar las aplicaciones que utilizan una interfaz gráfica. Todo

el tráfico de esta conexión es cifrado por el protocolo, lo cual hace que la comunicación sea segura pero a su vez se torna lenta para aplicaciones de tiempo real.

Para realizar esta conexión se ejecuta el siguiente comando en el terminal del equipo local, es decir, en la computadora desde la cual se conecta al Odroid U3:

```
ssh -X -p 1022 odroid@192.168.0.106
```

cuyos parámetros son los siguientes:

ssh, es el comando del protocolo de Secure Shell
-X, parámetro para visualizar aplicaciones con interfaz de usuario
-p 1022, parámetro que indica el puerto utilizado para la conexión
odroid, nombre del usuario en el equipo remoto
192.168.0.106, dirección IP del equipo remoto

Como respuesta al comando se presenta un mensaje pidiendo la contraseña del usuario, al ingresarla se muestra en el terminal un mensaje de haberse conectado correctamente y se podrán ingresar comandos que se ejecutan desde el Odroid U3. Antes de ubicar el soporte en una posición fija con los demás elementos, el Odroid U3 debe estar prendido y conectado a una red wifi, lo cual se configura previamente para que realice una conexión automática. Una vez realizada la conexión y haber accedido mediante SSH al Odroid U3 se puede controlar el sistema de reconocimiento vehicular utilizando la interfaz gráfica descrita en la sección anterior.

CAPÍTULO 5

ANÁLISIS DE RESULTADOS

El análisis de resultados está dividido en dos secciones: en la primera se analiza el funcionamiento en sí del sistema de detección e identificación vehicular; y en la segunda se analiza el tiempo de procesamiento de los algoritmos implementados.

5.1 Funcionamiento del Sistema de Detección e Identificación Vehicular

Para las pruebas de funcionamiento se colocaron los equipos en el soporte metálico a una altura de 4.50 metros, con el objetivo de obtener una vista superior de los vehículos en circulación, de manera que se facilite el reconocimiento del movimiento de los mismos a lo largo de la carretera, y que minimice las posibles interferencias en la imagen. Desde esta altura se obtuvo una imagen panorámica que se muestra en la Figura 63 de tamaño 1280x720 pixeles.

Figura 63: Imagen obtenida desde una altura de 4.50 metros.

Como configuración inicial, se ubicó y se tomó una distacia de 40 metros para colocar las marcas, se indicó la detección solamente en un carril ya que los vehículos pasaban por la mitad de la carretera, y se señaló como límite de velocidad 20 Km/h, que es el límite dentro de la Espe (Figura 64).

Figura 64: Configuración inicial para las pruebas de funcionamiento.

5.1.1 Detección Vehicular

En la etapa de detección vehicular se obtuvo que de 200 vehículos registrados manualmente que circularon por la zona de detección, 200 fueron registrados como autos por el sistema, obteniendo un 0% de error. Además, hubieron 2 bicicletas y 6 personas que atravesaron esta zona y no fueron registradas por el sistema, comprobando que la identificación funciona correctamente. Para comparar los resultados de la medición de

Tabla 3: Resultados del proceso de identificación vehicular

Velocidad Conocida (Km/h)	Velocidad Obtenida (Km/h)	Error (Km/h)	Error (%)
30	34.6	4.6	15.33
40	44.1	4.1	10.25
50	55.7	5.7	11.4

velocidad realizada por el sistema, y al no tener un equipo que registre resultados válidos de la velocidad de los vehículos como un sistema de radar, se realizó el siguiente procedimiento:

Se utilizó un vehículo con el cual se circuló a velocidades constantes de 30, 40 y 50 Km/h a través de la zona de detección, mientras el sistema funcionaba y realizaba el reconocimiento. Con este método se pudo comparar la velocidad obtenida por el sistema con la velocidad conocida del vehículo, obteniendo los resultados que se muestran en la tabla 3.

El porcentaje de error se calculó con la fórmula (5.1):

$$error = \frac{|Velocidad\ Obtenida - Velocidad\ Conocida|}{|velocidad\ Conocida|} * 100\% \tag{5.1}$$

En el Ecuador hasta el año 2015 no existe una ley de infracciones de velocidad que contemple el margen de error de los foto-radares, solamente se sanciona el exceso de velocidad y con eso se asume que estos equipos de medición son exactos. Por ello se tomó como referencia los umbrales de tolerancia de la Dirección General de Tráfico de España [46] que se aplican en las multas por excesos de velocidad detectados por radares. Este margen se sitúa en 7 km/h si el vehículo circula a menos de 100 km/h y del 7% si lo hace por encima de esa velocidad. Teniendo como referencia estos valores, el error máximo del sistema implementado es de 5.7 Km/h y estaría dentro del rango de tolerancia de los equipos de medición de velocidad vehicular.

5.1.2 Identificación Vehicular

La siguiente etapa es la identificación del número de placa del vehículo. A pesar de que las imágenes obtenidas son de 1280x720 pixeles y presentan una buena calidad de formas y colores en general, al estar los vehículos en movimiento no se obtienen imágenes claras de las zona de la placa, como se muestra en la Figura 65 por lo cual esta etapa no se realiza correctamente y se obtiene 100% de error.

Figura 65: Imagen del vehículo con zona de la placa no legible.

Para solucionar este inconveniente se realizaron otras pruebas de funcionamiento con la cámara ubicada a una altura de 1.70 metros, con lo cual se obtiene una imagen casi frontal de la placa del vehículo, pero se pierde considerablemente área de visión sobre la carretera. Desde esta ubicación se logró obtener imágenes similares a la que se muestra en la Figura 66, en la cual sí es posible localizar la placa para su identificación.

Tabla 4: Resultados de los procesos de reconocimiento vehicular

	Imágenes Analizadas por Proceso	Número de Aciertos	Error por Etapa	Error General	Porcentaje de aciertos por etapa	Porcentaje de aciertos General
Identificación Zona Placa	200	200	0%	0%	0%	100%
Segmentación de caracteres	200	190	5%	5%	95%	95%
OCR	190	172	9.47%	14%	90.53%	86%

Figura 66: Imagen del vehículo con zona de la placa legible.

Se analizaron 200 imágenes frontales de vehículos tomando en cuenta tres etapas principales de este proceso: la identificación de la zona de la placa, la segmentación de caracteres y el reconocimiento óptico de caracteres (OCR); obteniendo los resultados que se muestran en la tabla 4.

Los porcentajes de error se calcularon con las fórmulas (5.2) y (5.3):

$$error\ por\ etapa = \frac{|Imgenes\ analizadas\ por\ proceso - Nmero\ de\ Aciertos|}{|Imgenes\ analizadas\ por\ proceso|} * 100\% \tag{5.2}$$

$$error\ General = \frac{|Total\ Imgenes\ - Nmero\ de\ Aciertos|}{|Total\ Imgenes|} * 100\% \qquad (5.3)$$

El número de imágenes analizadas en cada proceso no es el mismo en todos ya que dependen de la efectividad del proceso anterior, es decir, si no se cumple satisfactoriamente la identificación en una etapa, no se continuará con la siguiente. En la primera etapa siempre se encuentra la zona que corresponde a la placa del vehículo, por lo tanto no presenta error. En la segunda etapa, el error del 5% en la segmentación de caracteres corresponde a vehículos con placas en mal estado, descoloridas, demasiado giradas o que partes del mismo vehículo obstruían su legibilidad; y el sistema de reconocimiento desarrollado no posee algoritmos morfológicos o de restauración de regiones para estos casos. Si se toma en cuenta solamente las placas en buen estado, el porcentaje de aciertos en la segmentación de caracteres sería del 100%.

La tercera etapa presenta un error del 9.47%, el cual se debe a que existen distintos tipos de letra utilizados en los caracteres de las placas del Ecuador, y no se pudo realizar el entrenamiento del lenguaje personalizado en Tesseract con todos ellos debido a que por seguridad la Agencia Nacional de Tránsito no divulga qué tipos de letra utiliza. Dicho entrenamiento se lo realizó empleando 15 muestras de cada caracter de fotografías de placas vehiculares previamente obtenidas, siendo en total 540 imágenes correspondientes a 10 números y 26 letras; sin embargo, no se puede aseverar que dentro de esas muestras se encuentran todos los tipos de letras de los vehículos en circulación. Este porcentaje de error se puede reducir con un correcto entrenamiento del lenguaje personalizado, si se dispone de los tipos de letras correspondientes.

5.1.3 Envío y Almacenamiento de la Información Obtenida

Después de haber identificado el número de placa de los vehículos, los datos obtenidos de velocidad, fecha, hora, número de placa, carril de circulación y fotografía

del vehículo, se envían a una base de datos en la red local. Para comprobar que este proceso se cumpla, se verificó que el número de placas en las que se realiza la segmentación de caracteres sea el mismo que el número de registros en la base de datos. Se revisó la base de datos en la computadora donde se instaló el servidor MySQL y se encontraron 190 registros (Figura 67), que corresponden a los 190 aciertos en la segmentación de caracteres. Este resultado incluye las 172 placas identificadas correctamente en el proceso de OCR y 18 erróneas.

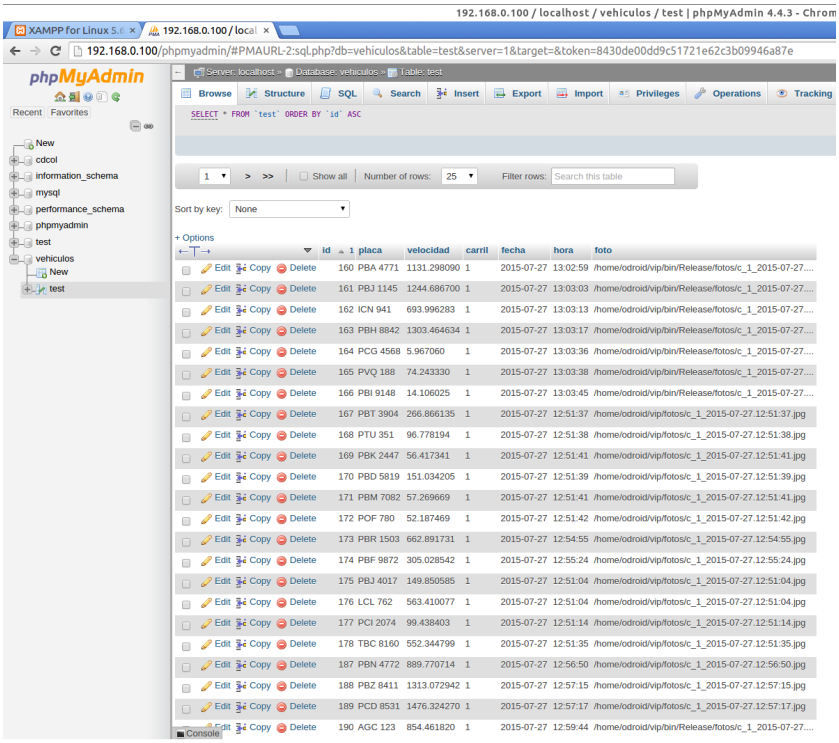


Figura 67: Registros almacenados en la base de datos con la información de los vehículos.

5.2 Tiempo de procesamiento de los algoritmos implementados

Se analizó el tiempo que se requiere para la ejecución de los procesos principales del sistema, que son la detección vehicular, la identificación de la placa y el envío-

Tabla 5: Tiempos de procesamiento Detección Vehicular

Tamaño de la imagen (pixeles)	Tiempo Odroid (ms)	Tiempo PC (ms)
320x240	12	3
640x480	25	6
1280x720	34	8

almacenamiento de la información obtenida. El análisis se lo realizó en el Odroid U3 y en una computadora personal, con el objetivo de comparar la eficiencia de la tarjeta embebida respecto a un equipo de uso general.

Las características principales de la computadora utilizada son las siguientes:

- Procesador Intel Core i3-2330M (segunda generación) @2.20Ghz
- Memoria RAM 4 GB
- Sistema operativo de 64 bits Linux-Ubuntu 14.04

Se instalador exactamente los mismos programas, controladores y librerías que se utilizaron en el Odroid U3, descritas en la sección de Implementación. Para la medición del tiempo se utilizaron las funciones *getTickCount()* y *getTickFrequency()* de la librería `<time.h>` de C++.

5.2.1 Tiempos de Procesamiento en la Detección Vehicular

Se realizaron mediciones con tres de los tamaños a los que se puede configurar la captura de imágenes con la cámara empleada: 320x240, 640x480 y 1280x720. Se obtuvieron como promedios los resultados de la tabla 5.

Se puede establecer que existe una relación lineal entre el tamaño de la imagen y el tiempo que se requiere para su procesamiento (figura 68). Tanto en el Odroid U3

como en la PC, el tiempo empleado es prácticamente el doble y el triple, al igual que el tamaño de las imágenes respecto a la más pequeña. Se aprecia también que el Odroid U3 tarda 4.17 veces más en realizar el procesamiento que la PC.

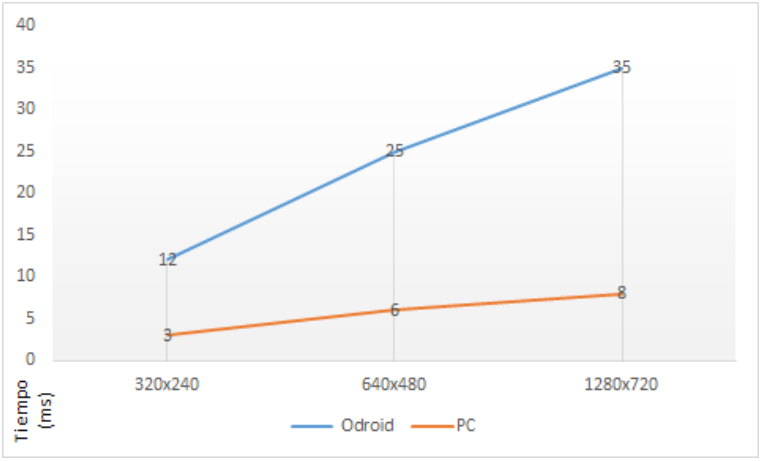


Figura 68: Comparación entre el tamaño de imagen y el tiempo de procesamiento para la detección vehicular.

5.2.2 Tiempos de Procesamiento en la Identificación Vehicular

En esta sección se analizó tanto el proceso general como los tres subprocesos de los cuales se conforma: localización de la placa, segmentación de caracteres, y OCR. Para el análisis del proceso general se utilizaron imágenes de distintos tamaños que resultaron del proceso de detección vehicular, y también imágenes de mayor tamaño adquiridas con cámaras distintas a la utilizada en el sistema, con el objetivo de evaluar el algoritmo desarrollado. Se eligieron diez tamaños de imágenes y se especificó la cantidad de pixeles que deben ser procesados. Los resultados obtenidos se los detalla en la tabla 6.

En las figuras 70 y 69 se aprecia que para este proceso, la relación entre el tiempo de procesamiento y el tamaño de la imagen no es completamente lineal, a pesar de que sí haya una tendencia ascendente. La relación promedio entre los tiempos obtenidos

Tabla 6: Tiempos de procesamiento general de Identificación Vehicular

Tamaño de la imagen (pixeles)	Total Pixeles	Tiempo Odroid (ms)	Tiempo PC (ms)	Relación Odroid/PC
259x194	50246	22	7	3.14
380x210	79800	27	8	3.38
320x270	86400	56	13	4.31
600x430	258000	57	12	4.75
640x480	307200	59	13	4.54
990x486	481140	99	20	4.95
1280x720	921600	96	19	5.05
1280x960	1228800	73	15	4.87
1920x1080	2073600	105	22	4.77
3264x2448	7990272	134	25	5.36
			Promedio:	**4.51**

en el Odroid y la PC es de 4.5, muy similar al del proceso anterior. En esta parte cabe recalcar que se realizó también mediciones de tiempo del algoritmo desarrollado en Matlab, cuyos resultados fueron superiores a los 200 ms para imágenes medias de 640x480, por lo que no se incluyeron estos resultados en las tablas y el análisis comparativo.

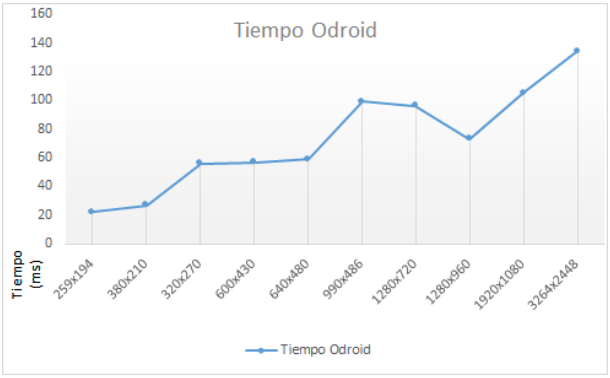


Figura 69: Comparación entre el tamaño de imagen y el tiempo de procesamiento en el Odroid para el proceso general identificación vehicular.

Tabla 7: Tiempos de procesamiento de los subprocesos de Identificación Vehicular en el Odroid

	259x194		640x480		3264x2448	
	Tiempo (ms)	%	Tiempo (ms)	%	Tiempo (ms)	%
Localización de la placa	5	22.72	38	64.4	120	89.55
Segmentación de caracteres	2	9.09	5	8.47	3	2.24
OCR	15	68.19	16	27.13	11	8.21
Total	22	100	59	100	134	100

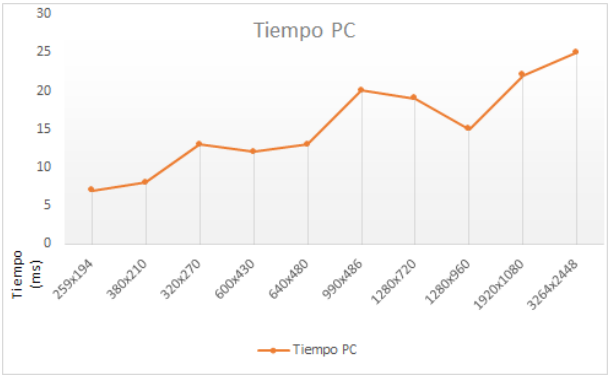


Figura 70: Comparación entre el tamaño de imagen y el tiempo de procesamiento en la PC para el proceso general identificación vehicular.

Para comprender por qué la relación no es completamente lineal, se analizaron los subprocesos que intervienen en la identificación vehicular, con tres tamaños de imágenes correspondientes al valor inferior (259x194), intermedio (640x480) y superior (3264x2448) de todos los que se encuentran en la tabla 6. Los resultados obtenidos se muestran en las tablas 7 y 8.

Las gráficas de las figuras 71 y 72 evidencian que tanto en el Odroid como en la PC, mientras los tiempos en el proceso de segmentación de caracteres y en el de OCR se mantienen prácticamente constantes a pesar de ser las imágenes de distintos tamaños, el tiempo en el proceso de localización de la placa se aumenta linealmente. Con esto se puede concluir que el tiempo requerido para la identificación vehicular depende principalmente de la localización de la placa, el cual a su vez depende del

Tabla 8: Tiempos de procesamiento de los subprocesos de Identificación Vehicular en la PC

	259x194		640x480		3264x2448	
	Tiempo (ms)	%	Tiempo (ms)	%	Tiempo (ms)	%
Localización de la placa	2	28.57	7.5	57.69	21	84
Segmentación de caracteres	1	14.28	1.5	11.54	0.6	2.4
OCR	4	57.15	4	30.77	3.4	13.6
Total	7	100	13	100	25	100

tamaño de la imagen.

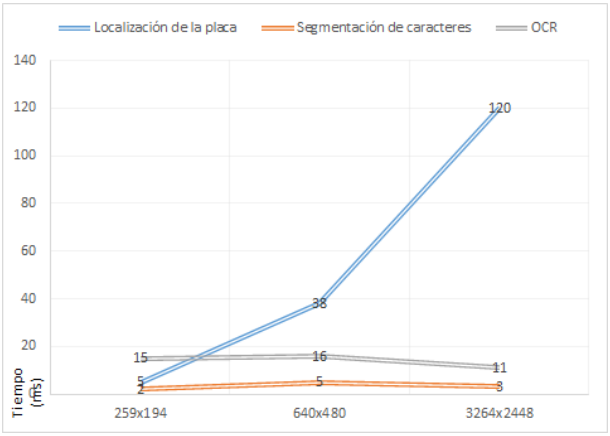


Figura 71: Comparación entre el tamaño de imagen y el tiempo de procesamiento en el Odroid para los subprocesos de la identificación vehicular.

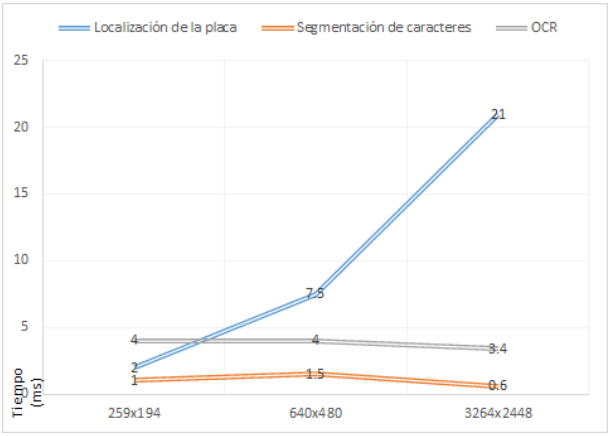


Figura 72: Comparación entre el tamaño de imagen y el tiempo de procesamiento en la PC para los subprocesos de la identificación vehicular.

5.2.3 Tiempos en el envío y almacenamiento de la información obtenida

Esta etapa incluye la conexión con el servidor, autenticación en la base de datos, envío de información y desconexión del servidor, procesos que se realizan empleando las librerías de MySQL para C++, e incluyendo en el código del programa las direcciones IP y las contraseñas necesarias para que durante el procesamiento no se requiera ninguna interacción con el usuario.

Se midieron los tiempos que todo este proceso necesita y se obtuvo que tanto para el Odroid como para la PC, se requieren en promedio **110 milisegundos**. Esto es debido a que ya no se utiliza procesamiento de imágenes sino distintos protocolos de comunicación. El tiempo requerido por este proceso haría que el sistema no se considere como de tiempo real, mas se debe tomar en cuenta que sólo se realiza el envío de información a la base de datos cuando se identifica correctamente un vehículo, teniendo como resultado un breve retardo durante este periodo.

5.3 Análisis de Capacidad de Procesamiento en Tiempo Real

Tomando como proceso principal la detección vehicular, pues es el que se ejecuta independientemente si existe o no la presencia de vehículos, se determinó en la sección anterior que se requiere en el Odroid un máximo de 34 milisegundos para imáges de tamaño 1280x720, lo cual resulta en 29 imágenes procesadas por segundo en el tamaño más grande que permite captura la cámara empleada. Cuando se detecta un vehículo, se debe añadir el tiempo de la identificación vehicular (96 ms) y el del envío de información a la base de datos (110 ms). En total, el proceso completo requiere 240 milisegundos, pudiéndose detectar hasta cuatro vehículos por segundo. Si se es-

tima que en la práctica los vehículos circulen con una diferencia de un segundo, se tendría 760 milisegundos en los que se continuaría realizando la detección vehicular, obteniéndose un promedio de 23 imágenes procesadas por segundo.

Realizando el mismo análisis para imágenes de 320x240 se obtiene un tiempo total de procesamiento de 149 milisegundos, con un máximo de 6 vehículos detectados por segundo, y con un promedio de 71 imágenes por segundo. Para imágenes de tamaño medio de 640x480 se obtiene un tiempo total de 194 milisegundos, con un máximo de 5 vehículos detectados, y con un promedio de 33 imágenes por segundo.

Es importante señalar que la cantidad de imágenes procesadas por segundo se ve directamente limitada por la cámara utilizada, la cual tiene una capacidad de captura máxima de 30 imágenes por segundo. Además que en condiciones de poca iluminación ambiental se obtienen solamente hasta 12 imágenes por segundo.

Con estos resultados se puede establecer que, sin tomar en cuenta las limitaciones de la cámara, sí es factible realizar el reconocimiento e identificación vehicular mediante procesamiento de imágenes en el sistema embebido Odroid U3.

Se debe indicar que como el Odroid U3 no está conectado a una pantalla el momento en que realiza el procesamiento, no se toma en cuenta el tiempo que se requiere para la visualización de un entorno gráfico. También se debe señalar que todos los análisis se realizaron con imágenes obtenidas durante el día, ya que la cámara no posee características para su funcionamiento con reducida iluminación ambiental.

CAPÍTULO 6

DISCUSIÓN

6.1 Conclusiones

El sistema de reconocimiento e identificación vehicular mediante procesamiento digital de imágenes en base al sistema embebido Odroid U3 desarrollado en el presente proyecto de grado presenta un porcentaje de aciertos del 100% para detección vehicular y 86% para identificación de la placa, bajo condiciones ideales, permitiendo además diferenciar vehículos de bicicletas y personas.

Se obtuvo un error promedio en la medición de velocidad de 4.8 Km/h. Después de realizar una investigación sobre leyes de tránsito en el Ecuador, no se logró encontrar una ley para infracciones de velocidad que contemple el margen de error de los equipos de medición. Por tal motivo se tomó como referencia los umbrales de tolerancia descritos por la Dirección General de Tráfico de España [46] que establecen márgenes de 7 Km/h si el vehículo circula a menos de 100 Km/h y del 7% si lo hace por encima de esta velocidad. Con lo cual se determinó que el sistema desarrollado se encuentra dentro de los márgenes de error para equipos de medición de velocidad vehicular.

El proceso de reconocimiento del número de placa se dividió en tres subprocesos

principales: la ubicación de la zona de la placa, con 100% de efectividad; la segmentación de caracteres, con el 95%; y el reconocimiento óptico de caracteres (OCR), con 90.53%. Como porcentaje de aciertos del proceso completo se obtuvo el 86%. Con este resultado se puede determinar que el sistema desarrollado presenta un 12% menos de efectividad en comparación a sistemas comerciales de reconocimiento automático de números de placa (APNR) cuyo porcentaje de aciertos es de 98% y 99%, debido principalmente a que poseen mejores prestaciones de hardware.

Los procesos de detección e identificación vehicular no pudieron ser realizados en conjunto desde una misma ubicación del sistema, debido a que la cámara web empleada posee características limitadas en resolución y para captura de imágenes en movimiento. Por ello, se necesitaron ubicaciones distintas para las pruebas de funcionamiento de cada proceso: a 4.50 metros de altura para la detección vehicular, y a 1.70 metros para la identificación de la placa.

El proceso completo, para imágenes de tamaño 1280x720, requiere 240 milisegundos en el Odroid U3, pudiéndose detectar hasta cuatro vehículos consecutivos por segundo. Este tiempo es 4.33 veces mayor que en el computador. Sin embargo, estimando que los vehículos circulen con una diferencia de 1 segundo, se pueden procesar 23 imágenes por segundo. Con estos resultados se puede establecer que, sin tomar en cuenta las limitaciones de la cámara, sí es factible la la aplicación del presente proyecto dentro de la Espe, en tiempo real.

6.2 Recomendaciones

Para mejorar el porcentaje de aciertos en el reconocimiento óptico de caracteres se recomienda realizar un entrenamiento con los tipos de letras específicos que se utilizan en las placas de vehículos del Ecuador, las cuales deben ser proporcionadas por la

Agencia Nacional de Tránsito.

Para realizar la detección e identificación vehicular desde una misma ubicación se recomienda el empleo de una cámara especializada para aplicaciones de reconocimiento vehicular, como es la cámara **KOMOTO NUMBER PLATE READCAM AVN-80RL25/AVP-80RL25** [47], que entre sus principales características presenta: Velocidades de captura desde 1/60 hasta 1/120,000 segundos; 80 leds infrarrojos para visión nocturna hasta 15 metros de alcance; sensor CCD 1/3" que proporciona una imagen de alta calidad; protección contra polvo y agua para su uso en exteriores.

Se recomienda colocar tres puntos de control estratégicos alrededor del campus Espe Sangolquí, los cuales se especifican en en anexo A. Dichos puntos se escogieron en base a: 1) la ubicación dentro del campus, se puede posicionar correctamente el sistema de detección vehicular debido a que se encuentran en tramos de rectas en la carretera; 2) conectividad, en estas ubicaciones se obtuvo una correcta conexión a la red Wi-Fi de la Espe, lo que permite efectuar el envío de información a la base de datos, y 3) velocidad alcanzada por los vehículos, en los sitios escogidos se observó que los vehículos circulan a más de 20 km/h, y que aumentan su velocidad en lugar de reducirla.

Se recomienda diseñar una carcasa de protección, como mínimo bajo las normas IP66 contra agua y polvo [48], que permita el adecuado uso del sistema en exteriores.

ANEXO A

Puntos de Control Vehicular

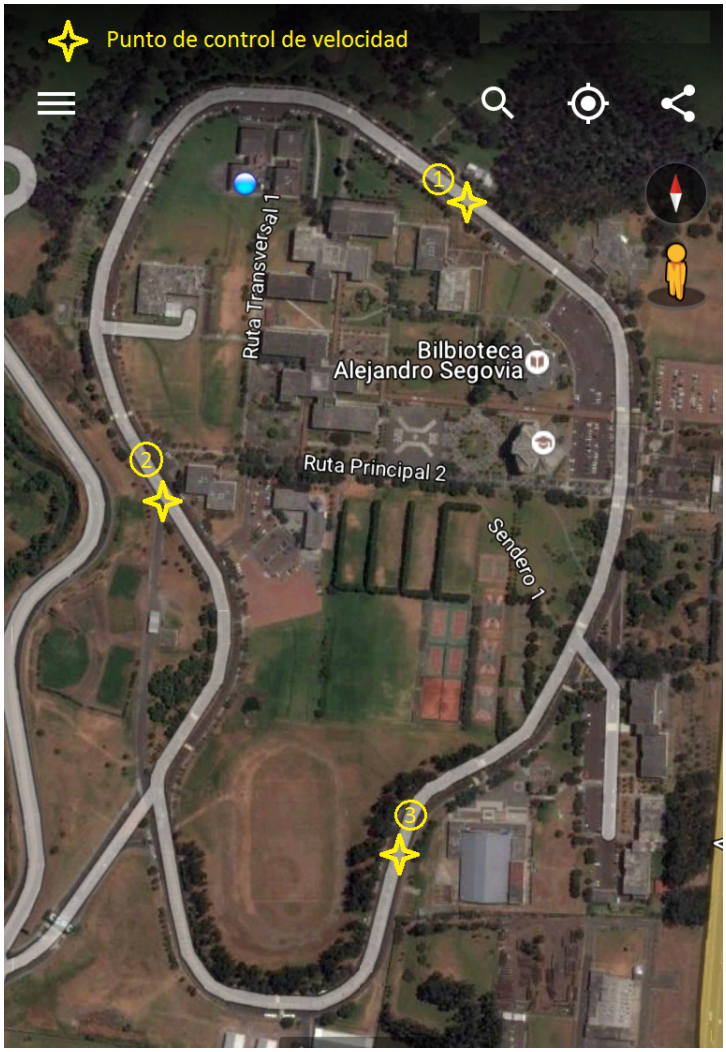


Figura 73: Puntos de control vehicular recomendados para el campus Espe Sangolquí.

BIBLIOGRAFÍA

[1] ANT, "Siniestros por causas probables a nivel nacional," 2014, extraído el 28 de abril del 2014.

[2] ANT2, "Fotoradares de última tecnología para el control de velocidades," 2013, extraído el 1 de mayo del 2014.

[3] INGETRA, "Sistemas de detección vehicular, en dos carriles, mediante sensor láser visible activo," 2012, extraído el 1 de mayo del 2014.

[4] SENSEFIELDS, "Circulaseguro seguridad vial y movilidad sostenible," 2009, extraído el 28 de abril del 2014.

[5] MAPS, "Detección y clasificación de vehículos mediante cortinas fotoeléctricas maps cfxx," extraído el 28 de abril del 2014.

[6] AMnetpro, "Sistema de detección electrónica de infracciones de tránsito," 2010, extraído el 30 de abril del 2014.

[7] A. Mahecha, J. Quiroga, and J. Sepúlveda, "Sistema de reconocimiento y lectura de placas de vehículos en movimiento," *XVI SIMPOSIO DE TRATAMIENTO DE SEnALES, IMAGENES Y VISIÓN ARTIFICIAL STSIVA*, 2008.

[8] A. Mahecha, J. Quiroga, and J. Sepúlveda, "Sistema de conteo, identificación y clasificación de vehículos en un peaje," *XIII SIMPOSIO DE TRATAMIENTO DE SEnALES, IMAGENES Y VISIÓN ARTIFICIAL STSIVA*, 2009.

[9] H. Rodríguez, R. Vera, and B. Vintimilla., "Detección y extracción de placas de vehículos en senales de video," *Revista Tecnológica ESPOLRTE, Vol. 25*, 2012.

[10] MAPS, "Sistema de lectura automática de matrículas," extraído el 28 de abril del 2014.

[11] C. Parsons, J. S. R. Wipond, and K. McArthur, "Anpr: Code and rhetorics of compliance," *European Journal of Law and Technology, Vol. 3*, 2012.

[12] ABC, "Detectores e inhibidores de radar: Nos pueden multar por utilizarlos?" extraído el 6 de mayo del 2014.

[13] MAPS, "Cortina fotoelectrica maps cf-324," extraído el 3 de mayo del 2014.

[14] M. Barr, "Embedded systems glossary," extraído el 6 de mayo del 2014.

[15] D. Millett, "Arm low-power processors and architectures," extraído el 6 de mayo del 2014.

[16] T. Instruments, "Lpddr," extraído el 28 de septiembre del 2014.

[17] O. Rosas and L. Alvarez, "Estimación de velocidades vehiculares en vías rápidas," *Congreso Anual de la AMCA*, 2004.

[18] H. H. Rivas, "La identificación vehicular por radiofrecuencia (rfid) en méxico, una realidad," *TecnoINTELECTO, Órgano de Divulgación Científica, ITCV*, 2007.

[19] V. Castro and G. Torres, "Prototipo de un sistema de ubicación y reconocimiento de placas vehiculares en automóviles en movimiento." *Avances en Informática y Sistema Computacionales Tomo I*, 2006.

[20] A. Ruiz, "Sistema de identificación de placas vehiculares con técnicas de visión computacional," Master's thesis, Instituto Politécnico Nacional, 2010.

[21] R. Gutierrez, M. Frydson, and B. Vintimilla, “Aplicación de visión por computador para el reconocimiento automático de placas vehiculares utilizando ocrs convencionales,” Master’s thesis, Escuela Superior Politécnica del Litoral, 2011.

[22] H. Rodriguez, R. Vera, and B. Vintimilla, “Detección y extracción de placas de vehículos en senales de video,” *Revista Tecnológica ESPOL RTE*, 2012.

[23] I. García, “Visión artificial y procesamiento digital de imágenes,” 2010, extraído Mayo 03, 2015.

[24] M. Ortiz, “Procesamiento digital de imágenes,” Master’s thesis, Universidad Autónoma de Puebla, 2013.

[25] R. Wainschenker, J. Massa, and P. Tristan, “Procesamiento digital de imágenes,” 2008, extraído Abril 13, 2015.

[26] C. I. P. "ETI", “Visión artificial,” extraído Mayo 02, 2015.

[27] ANTISACSOR, “Procesadores arm,” extraído Mayo 01, 2015.

[28] I. Callejas, “Procesadores de bajo consumo el futuro de los gadgets,” extraído Mayo 01, 2015.

[29] E. Comer and L. Stevens, *Client-Server Programming and Applications*, W. L. Department of Computer Sciences, Purdue University, Ed. Prentice Hall, 1993.

[30] DELL, “Servidores dell power edge,” 2015, extraído mayo 1, 2015.

[31] M. Barr, “Netrino technical library,” 2007, extraído mayo 1, 2015.

[32] S. Heath, *Embedded systems design*, E. series for design engineers, Ed. Newnes, 2003.

[33] E. Chan, “Simple avr programmers,” 2012, extraído mayo 1, 2015.

[34] HARDKERNEL, “Odroid u3 technical detail,” HARDKERNEL, Tech. Rep., 2014.

[35] Android, “Android-x86 - porting android to x86,” 2013, extraído mayo 5, 2015.

[36] Android, “Código fuente de android,” extraído mayo 5, 2015.

[37] Android, “Lista de incidencias de android,” extraído mayo 5, 2015.

[38] Android, “Android, the world’s most popular mobile platform,” 2011, extraído mayo 5, 2015.

[39] R. Grant, “Ubuntu for non-geeks,” 2010, 4th Edition.

[40] Ubuntu, “Canonical services,” 2012, http://www.ubuntu.com/management.

[41] Ubuntu, “Ubuntu design,” 2012, http://www.ubuntu.com/support/paid.

[42] opencv dev team, “Introduction to opencv,” 2011, http://docs.opencv.org/modules/core/doc/intro.html.

[43] D. Bloomberg, “Qt reference documentation,” 2015, http://leptonica.com/.

[44] Nokia, “Qt reference documentation,” 2008, http://docs.opencv.org/modules/core/doc/intro.html.

[45] E. C. Lonvick, “The secure shell (ssh) transport layer protocol,” Cisco Systems, Inc., Tech. Rep., 2006.

[46] M. Trillo, “Umbrales de tolerancia para radares en espaa,” 2015, extraído mayo 9, 2015.

[47] KOMOTO, “Number plate readcam avn-80rl25/avp-80rl25,” KOMOTO ENTERPRISE CO., LTD., Tech. Rep., 2010.

[48] I. E. Commission, “Degrees of protection provided by enclosures (ip code),” National Electrical Manufacturers Association, Tech. Rep., 2004.

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

ACTA DE ENTREGA

El presente proyecto fue entregado en el Departamento de Eléctrica y Electrónica, y reposa en los archivos desde:

Sangolquí, 17 / 08 / 2015.

Elaborado por:

Milton Fabricio Rosero Prado

Autoridad:

Msc. Ing. Darwin Alulema

DIRECTOR DE LA CARRERA DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE ING. ELECTRÓNICA EN TELECOMUNICACIONES