Detección de peatones basado en elementos del cuerpo y acelerado con GPU con pruebas en ambiente real

por

Ing. Ángel Alejandro Gómez Casasola

Tesis sometida como requisito parcial para obtener el grado de

MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRÓNICA

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica.

Tonantzintla, Puebla

Febrero 2016

Director de tesis

Dr. Jorge Francisco Martínez Carballido

RESUMEN

De acuerdo a la Organización Mundial de la Salud (WHO por sus siglas en inglés), los accidentes de tránsito representan la novena causa de muerte a nivel mundial, y se pronostica que para el año 2030 será la quinta [1].

Para revertir este hecho, se han desarrollado sistemas que asisten al conductor (ADAS), así como vehículos autónomos. Ambas tecnologías necesitan detectar eficientemente a los peatones que corren el riesgo de ser atropellados para que puedan llevar a cabo una acción y así evitar el accidente.

El principal inconveniente de los algoritmos para la detección de peatones existentes es que son muy complejos, lo que genera tiempos de procesamiento prolongados. Debido a esto, tanto los ADAS como los vehículos autónomos sólo son efectivos a bajas velocidades, máximo 50Km/h.

En este trabajo de tesis se presenta un algoritmo para la detección de peatones más simple que los presentados hasta ahora, sin dejar de ser eficaz. Este algoritmo se basa en la detección individual de las partes más relevantes del cuerpo de una persona, como son las piernas, los brazos o la cabeza. Además, para reducir aún más los tiempos de procesamiento, el algoritmo es implementado en una GPU por medio de OpenCL, alcanzando hasta 350 FPS.

ABSTRACT

According to the World Health Organization (WHO), traffic accidents represent the ninth leading cause of death worldwide, and it is predicted that in 2030 will be the fifth [1].

To reverse this fact, it have been developed systems that assist the driver (ADAS) as well as autonomous vehicles. Both technologies need to efficiently detect pedestrians at risk of being hit so they can perform an action to avoid the accident.

The main drawback of algorithms for pedestrian detection existing is that they are very complex, which generates long processing times. Because of this, both the ADAS as autonomous vehicles are only effective at low speeds, maximum 50km/h.

In this thesis it is presented an algorithm for pedestrian detection simpler than that reported so far, while remaining effective. This algorithm is based on the individual detection of the most important parts of a person's body, such as legs, arms or head. Moreover, to further reduce processing times, the algorithm is implemented on a GPU using OpenCL, reaching up to 350 FPS.

AGRADECIMIENTOS

A mi madre, por siempre brindarme su apoyo sin importar la situación. Sé que siempre podré contar con ella.

A mi padre, por todo el esfuerzo y sacrificio que dedicó para mi formación como profesionista. Este trabajo de tesis no hubiera sido posible sin él.

A mi asesor, Dr. Jorge Martínez Carballido, por su excelente orientación, apoyo, disposición, y por todos los conocimientos que me brindó para la realización de esta tesis.

Al CONACyT (Consejo Nacional de Ciencia y Tecnología) por el apoyo económico durante estos dos años de arduo trabajo. Su colaboración engrandece a nuestro país.

DEDICATORIA

A Dana y Bijou,

por enseñarme lo que es la felicidad

y las cosas por las que vale la pena luchar en esta vida.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS Y TABLAS

CAPÍTULO 1

INTRODUCCIÓN

La invención de los automóviles ha sido un gran logro en la historia de la humanidad ya que han contribuido enormemente en muchos aspectos de la vida. Gracias a ellos las personas se pueden trasladar fácil, rápida y cómodamente a sus diferentes destinos, además de poder trasladar materiales y recursos de su necesidad.

Desafortunadamente, así como han beneficiado a la humanidad, también han sido los responsables de millones de muertes y lesiones alrededor del mundo.

Los accidentes de tránsito son un severo problema que ha ido creciendo al mismo ritmo que la industria automotriz.

Es de suma importancia desarrollar tecnologías que puedan colaborar a disminuir los accidentes de tránsito.

1.1 Motivación

La industria automotriz, así como la del transporte en general, crece día a día, por lo que es necesario el desarrollo de nuevas y mejores tecnologías que puedan asistir y mejorar a los conductores de estos vehículos, sobre todo en el aspecto de seguridad.

Las GPUs en los últimos años han tenido una excelente aceptación y desempeño en el ámbito científico y tecnológico debido a sus características y prestaciones, con lo cual resultan atractivas para el desarrollo e implementación de nuevas tecnologías.

OpenCL es un estándar que facilita enormemente el desarrollo en GPUs, además de otros procesadores. Ha tenido un rápido crecimiento desde su creación hasta el día de hoy, ofreciendo importantes ventajas frente a otros potentes estándares lo que lo coloca entre una de las mejores opciones para el desarrollo basado en GPUs. Una de sus principales ventajas es que es un estándar abierto,

libre de licencias, además de que es compatible con cualquier GPU sin importar el fabricante.

1.2 Justificación

Cada año mueren aproximadamente 1.2 millones de personas alrededor del mundo debido a los accidentes de tránsito [1]. De continuar esta situación así, para el año 2030 los accidentes de tránsito representarán la quinta causa de muerte a nivel mundial.

Se han desarrollado diversas tecnologías para prevenir los accidentes de tránsito, especialmente los accidentes de vehículo contra persona, pero presentan importantes limitaciones.

Es imprescindible el mejoramiento de estas tecnologías para en un futuro erradicar los accidentes de tránsito.

1.3 Problema

Los algoritmos para la detección de peatones conocidos son muy complejos, por lo que requieren tiempos de procesamiento prolongados lo que limita considerablemente la velocidad de respuesta de las tecnologías que los utilizan. Es esta misma limitante lo que obliga a estas tecnologías valerse de un mayor número de sensores para mejorar el nivel de detección de los peatones, lo que eleva el costo de los automóviles que las implementan.

1.4 Objetivo

Diseñar un algoritmo sencillo y eficaz para la detección de peatones que sea capaz de procesar más de 135 imágenes por segundo (FPS). Además, dicho algoritmo será acelerado por medio del paralelismo que ofrecen las GPUs y la herramienta de desarrollo OpenCL. Los resultados obtenidos deberán demostrar que se puede detectar un peatón desde un automóvil que se mueva hasta 80Km/h con suficiente antelación para que éste (o la persona que lo conduce) pueda realizar una acción con el fin de no dañar al peatón.

1.5 Hipótesis

Para el desarrollo del algoritmo se tomó como referencia el funcionamiento del ojo humano, el cual analiza primeramente los contornos de los objetos (silueta) para extraer las características más sobresalientes, después se centra en analizar los

detalles y, finalmente, compara toda la información obtenida con la base de datos del cerebro (recuerdos) para obtener una concordancia y reconocer el objeto [2].

El algoritmo primeramente extraerá los bordes de la imagen y sobre ellos se enfocará en buscar las características más relevantes y constantes de una persona, como son las piernas, la cabeza, y los brazos.

Una vez detectadas esas partes, hará una comparación de proporciones dependiendo de la profundidad en donde se haya hecho la detección. Estas proporciones son tomadas de una persona promedio a diferentes profundidades, por lo que se mantienen constantes.

Las coincidencias tanto de proporciones como de partes detectadas determinarán la presencia de un peatón o no.

El algoritmo dará prioridad a las áreas por donde pasará el automóvil para detectar lo antes posible a los peatones que corren el riesgo de ser atropellados.

1.6 Alcance

En esta investigación se desarrolló y probó el algoritmo con una base de imágenes tomadas a plena luz del día, sin lluvia y sin viento excesivo en un ambiente no controlado en la zona centro de la Ciudad de México. Se considera a los peatones como personas de complexiones y estaturas promedio, tez morena clara, carentes de alguna anomalía física como la falta de una pierna, etc., y con una vestimenta promedio. Los peatones estarán cruzando las calles completamente de perfil y exclusivamente caminando, esto es, sin bicicletas o algún otro objeto de transporte. La resolución de imagen con la que se trabajó es VGA, 640x480 pixeles.

A pesar de que OpenCL es un estándar para la programación de diferentes tipos de procesadores (programación heterogénea), en este trabajo se utilizará únicamente para la programación de GPUs, en su versión 1.2.

Toda condición ajena a estas especificaciones así como la implementación del algoritmo en un automóvil (o algún otro vehículo) en tiempo en línea quedan fuera de este trabajo de tesis, pero contempladas como trabajos a futuro.

1.7 Estructura de la tesis

En el capítulo 2 se presentan los conceptos fundamentales relacionados con este trabajo y cómo han ido evolucionando. Además, se hace un compendio de los trabajos e investigaciones más importantes que se han desarrollado del tema hasta la fecha (estado del arte).

El capítulo 3 explica detalladamente el funcionamiento del algoritmo diseñado y su implementación en una GPU por medio de OpenCL.

En el capítulo 4 se muestran los resultados obtenidos y se hace un análisis de éstos para determinar la eficacia y competencia del algoritmo.

Finalmente, en el capítulo 5 se presentan las conclusiones en base a los resultados obtenidos y se plantean posibles trabajos subsecuentes de esta investigación.

CAPÍTULO 2

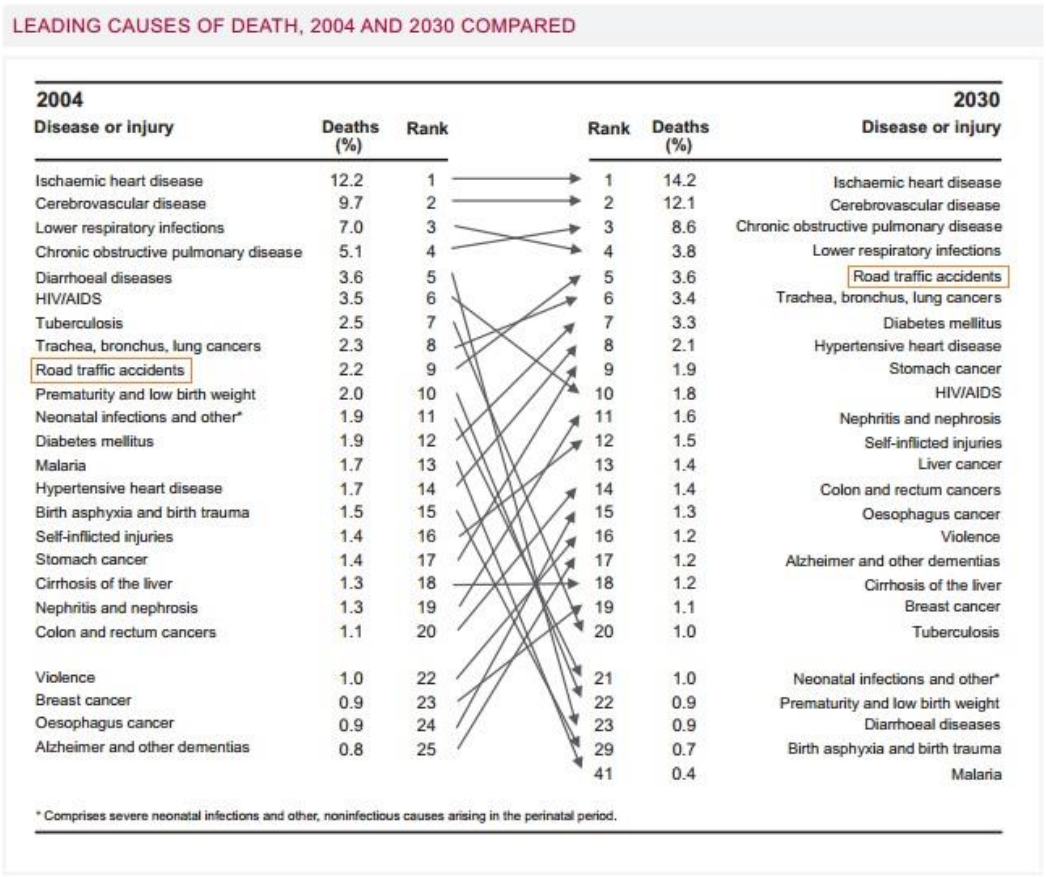
ANTECEDENTES

Desde el comienzo de la industria automotriz, a principios del siglo XX, los automóviles han pasado a formar parte en la vida cotidiana de las personas, usándolos para el transporte personal así como para el transporte de mercancías. Con el paso de los años han sido mejorados para ofrecer mayor rapidez, confort y seguridad.

2.1 Problemática automotriz y soluciones propuestas

Como muchas otras tecnologías, los automóviles desde su comienzo han cargado con un indeseable lado obscuro: los accidentes de tránsito. Es prácticamente inevitable que las rutas de los automóviles se crucen en algún momento con las de las personas que se trasladan caminando, los peatones. La primera muerte por un vehículo motorizado fue registrada en Irlanda el 31 de agosto de 1869 [3], por un vehículo experimental a vapor cuando todavía no iniciaba la producción de automóviles en masa. Aunque el número de fatalidades fue bajo en un comienzo, fueron incrementando exponencialmente a través de los años debido a la popularización del automóvil.

Hoy en día, de acuerdo con las estadísticas de la Organización Mundial de la Salud, o WHO por sus siglas en inglés, los accidentes de tránsito representan la novena causa de muerte a nivel mundial y se pronostica que para el año 2030 será la quinta [1]. Cada año casi 1.2 millones de personas mueren en accidentes de tránsito de los cuales dos terceras partes son peatones, mientras que el número de lesionados aumenta a 50 millones [1]. Además, de acuerdo con el incremento en la producción de automóviles en los países de bajos y medianos ingresos, este número se espera que aumente considerablemente.



LEADING CAUSES OF DEATH, 2004 AND 2030 COMPARED

2004 Disease or injury	Deaths (%)	Rank	Rank	Deaths (%)	2030 Disease or injury
Ischaemic heart disease	12.2	1	1	14.2	Ischaemic heart disease
Cerebrovascular disease	9.7	2	2	12.1	Cerebrovascular disease
Lower respiratory infections	7.0	3	3	8.6	Chronic obstructive pulmonary disease
Chronic obstructive pulmonary disease	5.1	4	4	3.8	Lower respiratory infections
Diarrhoeal diseases	3.6	5	5	3.6	Road traffic accidents
HIV/AIDS	3.5	6	6	3.4	Trachea, bronchus, lung cancers
Tuberculosis	2.5	7	7	3.3	Diabetes mellitus
Trachea, bronchus, lung cancers	2.3	8	8	2.1	Hypertensive heart disease
Road traffic accidents	2.2	9	9	1.9	Stomach cancer
Prematurity and low birth weight	2.0	10	10	1.8	HIV/AIDS
Neonatal infections and other*	1.9	11	11	1.6	Nephritis and nephrosis
Diabetes mellitus	1.9	12	12	1.5	Self-inflicted injuries
Malaria	1.7	13	13	1.4	Liver cancer
Hypertensive heart disease	1.7	14	14	1.4	Colon and rectum cancers
Birth asphyxia and birth trauma	1.5	15	15	1.3	Oesophagus cancer
Self-inflicted injuries	1.4	16	16	1.2	Violence
Stomach cancer	1.4	17	17	1.2	Alzheimer and other dementias
Cirrhosis of the liver	1.3	18	18	1.2	Cirrhosis of the liver
Nephritis and nephrosis	1.3	19	19	1.1	Breast cancer
Colon and rectum cancers	1.1	20	20	1.0	Tuberculosis
Violence	1.0	22	21	1.0	Neonatal infections and other*
Breast cancer	0.9	23	22	0.9	Prematurity and low birth weight
Oesophagus cancer	0.9	24	23	0.9	Diarrhoeal diseases
Alzheimer and other dementias	0.8	25	29	0.7	Birth asphyxia and birth trauma
			41	0.4	Malaria

* Comprises severe neonatal infections and other, noninfectious causes arising in the perinatal period.

Figura 2.1: Principales causas de muerte a nivel mundial y pronosticaciones para el año 2030 [1]

Con el avance de la tecnología han surgido múltiples sistemas implementados en los automóviles para mejorar la seguridad tanto del conductor (y los pasajeros) como la de las personas en las calles. Las bolsas de aire, los sistemas de frenos antibloqueo (ABS), la distribución electrónica de frenado (EBD), y el control de estabilidad (ESC), fueron de los primeros sistemas que se desarrollaron para mejorar la seguridad, pero enfocados principalmente a la del conductor y las personas a bordo del automóvil.

Gracias a los avances en el área computacional, la industria automotriz pudo valerse de tecnologías como la visión por computadora y la inteligencia artificial para desarrollar sistemas de seguridad un poco más orientada a los peatones. Ejemplos de estos sistemas son los sistemas avanzados de asistencia al conductor, los sistemas de protección a los peatones y los vehículos autónomos.

Sin embargo, a pesar de que estas tres tecnologías han sido eficaces, sufren de la misma limitante: sólo son efectivas a una velocidad del automóvil de 40Km/h como máximo.

Un peatón impactado por un automóvil a 40Km/h tiene más del 60% de probabilidades de sobrevivir, además de que es menos probable un accidente de tránsito a esa velocidad, mientras que a 80Km/h (o más) prácticamente significaría la muerte del peatón [4]. Es por esto que estas tres últimas tecnologías no han sido del todo útiles para evitar accidentes mortales.

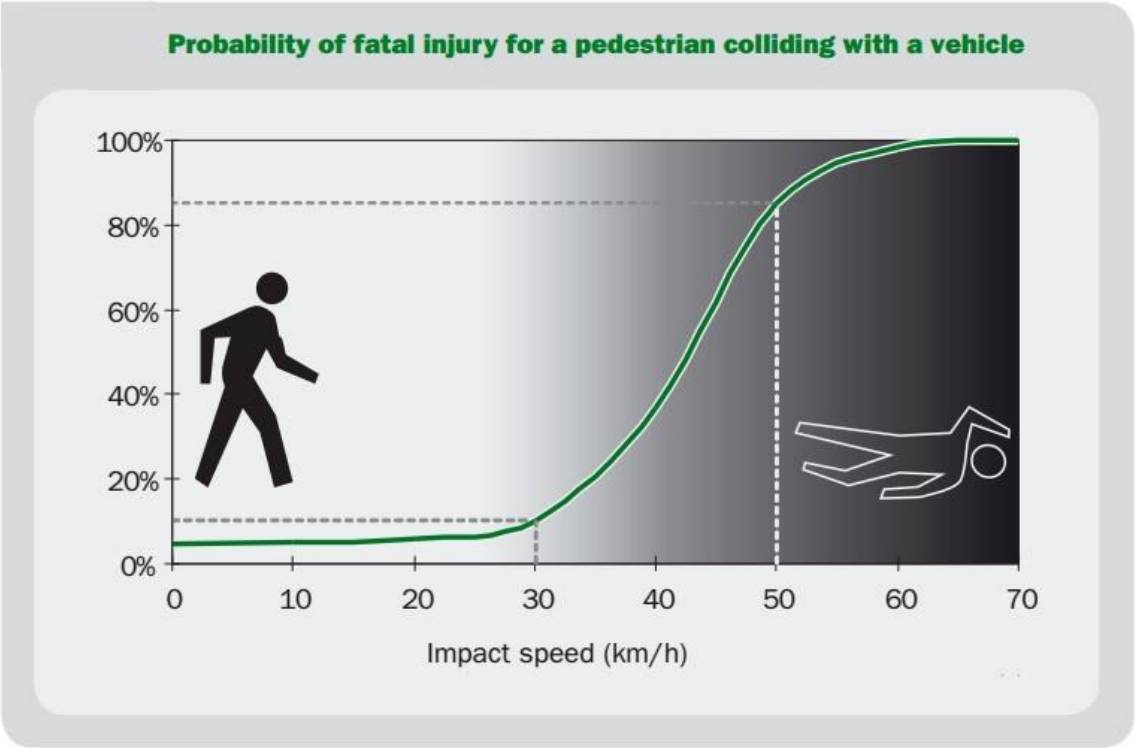


Figura 2.2: Probabilidad de muerte de una persona ante el impacto de un vehículo [4]

Se hablará un poco más a detalle de estas tecnologías para comprender su funcionamiento y analizar sus limitantes.

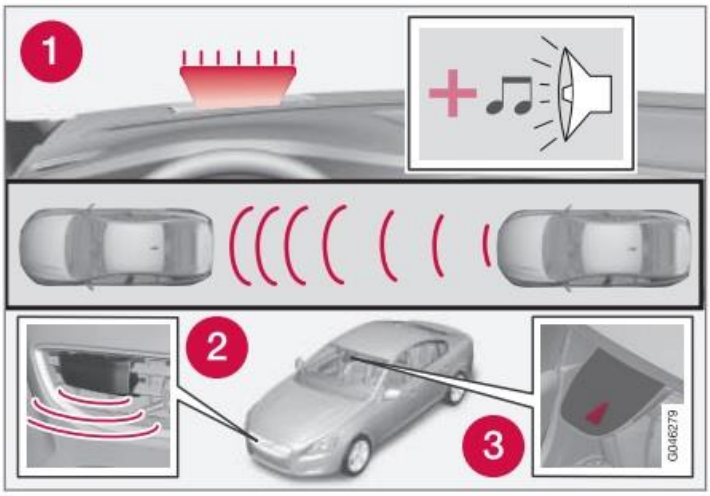
2.1.1 Sistemas avanzados de asistencia al conductor (ADAS)

Las personas que van conduciendo un automóvil son las principales responsables de su propia seguridad, de la de los pasajeros que lo acompañan así como la de los peatones en las calles. Basta con una simple mirada al teléfono, o al asiento trasero en donde van los niños, alguna escena en el camino que llame la atención, o simplemente el cansancio para ocasionar un accidente de tránsito. Es por esta razón que se desarrollaron los sistemas avanzados de asistencia al conductor, o ADAS (Advanced Driver Assistance Systems) por sus siglas en inglés, los cuales son sistemas que básicamente alertan al conductor distraído en caso de que el sistema detecte un posible accidente. Esta alerta se realiza principalmente de forma

acústica, con suficiente antelación para que el conductor pueda reaccionar y tomar medidas.

El principal problema de estos sistemas radica en cómo detectar el posible accidente. Principalmente se utilizan sistemas de visión por computadora empleando cámaras CCD, ya sea en la parte frontal del vehículo, en la trasera o ambas, junto con algoritmos para la detección de algún patrón que pueda generar un accidente, como los vehículos alrededor, obstáculos, el camino que debe seguir el automóvil, los peatones, etc. [5]. También se usan diferentes tipos de sensores como los radares igualmente acompañados por algoritmos para detectar estos patrones [6], sensores de proximidad o incluso sensores de ritmo cardiaco instalados en el volante del automóvil para detectar si el conductor se adormece mientras maneja y alertarlo, entre otros.

Se han desarrollado ADAS más avanzados que en vez de únicamente alertar al conductor, además toman el control del automóvil realizando la acción de frenado [7].



Function overview[18].

1. Audio-visual warning signal in the event of a collision risk.
2. Radar sensor[19]
3. Camera sensor

Figura 2.3: ADAS implementado en el Volvo® modelo S80 2016 [8]

Estos sistemas se han desempeñado relativamente bien alertando al conductor en caso de posibles colisiones contra otros vehículos, en caso de que el automóvil se salga de la ruta (subirse a la acera) o si el conductor presenta cansancio o

somnolencia; situaciones que mejoran principalmente la seguridad del conductor aunque ligeramente también la de los peatones.

Sin embargo, para la detección de peatones que corren el riesgo de ser atropellados no se ha podido desarrollar un sistema eficaz a más de 40Km/h [9]. Esto es debido a que los algoritmos para la detección de peatones conocidos son muy complejos por lo que los procesadores no alcanzan a procesar el algoritmo con suficiente antelación para alertar al conductor a mayores velocidades, la cual es la principal limitante de estos sistemas.

2.1.2 Sistemas de protección a los peatones (PPS)

Los ADAS introdujeron una ligera mejora para la seguridad de los peatones, sin embargo, esto no fue suficiente. Es por esto que se desarrollaron los sistemas de protección a los peatones, o PPS (Pedestrian Protection System) por sus siglas en inglés, los cuales, como su nombre lo indica, son sistemas que están enfocados puramente a la protección de los peatones.

El funcionamiento de estos sistemas consiste en detectar a los peatones que corren el riesgo de ser atropellados para luego realizar las acciones necesarias para evitar el atropello, o en el peor de los casos, aligerarlo lo más posible.

Las acciones que pueden realizar estos sistemas incluyen alertar al conductor, por lo que un PPS se podría considerar como un ADAS enfocado sólo a los peatones. También pueden tomar control del automóvil para realizar acciones evasivas o de frenado.

En el caso de que el atropello sea inevitable, pueden activar mecanismos en el automóvil para aligerar el impacto contra el peatón, como son bolsas de aire en la parte frontal para que la cabeza del peatón no golpeé directamente contra el cristal.

Figura 2.4: PPS implementado en la Land Rover® Discovery 2015 [10]

La manera en cómo estos sistemas detectan a los peatones que pueden ser atropellados es la misma que con los ADAS, por medio de cámaras y algoritmos para la detección de peatones, por lo que esta tecnología sufre exactamente la misma limitante. Al trabajar con algoritmos complejos, sólo son efectivos a bajas velocidades.

2.1.3 Vehículos autónomos

Como se mencionó anteriormente, el conductor de un automóvil puede ser el responsable de accidentes de tránsito debido a sus limitaciones humanas como el cansancio, la distracción o malos reflejos. Es por esta razón que se empezaron a desarrollar automóviles que no necesitan de un conductor humano, esto es, que son operados completamente por máquinas y un sistema electrónico computarizado, conocidos como vehículos autónomos.

Las ventajas de los vehículos autónomos son, principalmente, la seguridad, ya que el sistema del vehículo se encargaría de llevar a los pasajeros a su destino atendiendo todos los señalamientos y reglas de tránsito así como ir monitoreando el no lastimar a nadie en las calles, todo esto sin sufrir distracciones o cansancio [11].

Además, el vehículo una vez haya dejado a los pasajeros en su destino, podría regresar a su propia cochera de forma autónoma con el fin de reducir el problema de estacionamiento en las calles, entre muchas otras ventajas.

Para realizar tales acciones, los vehículos autónomos se valen de múltiples dispositivos, como láseres, radares, sensores de proximidad, sistemas de posicionamiento global (GPS), cámaras infrarrojas, cámaras CCD; todo esto controlado por un complejo sistema computarizado [12].

E. D. Dickmanns *et al.* [13] presentaron uno de los primeros sistemas de conducción autónoma que era capaz de conducir hasta una velocidad de 96Km/h en ambiente controlado; esto es, sin imprevistos en obstáculos o peatones.

Sin embargo, para que el vehículo autónomo pueda conducirse en vías públicas necesita poder detectar obstáculos y peatones para no causar accidentes. Es para esto que utiliza las cámaras, junto con algoritmos para la detección de peatones, al igual que los ADAS y los PPS, por lo que estos vehículos padecen la misma limitante, no son efectivos a más de 40Km/h.

Es por esta razón que actualmente no hay vehículos autónomos de forma comercial, únicamente prototipos.

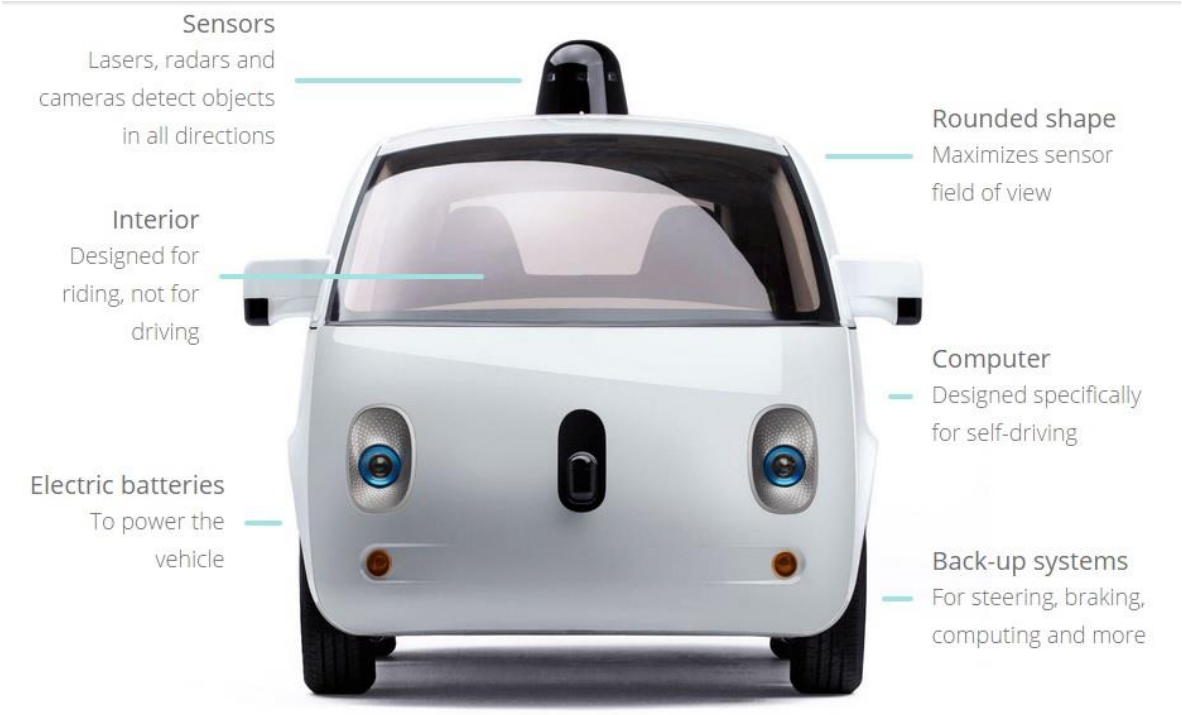


Figura 2.5: Vehículo autónomo desarrollado por Google® [11]

2.2 Algoritmos para la detección de peatones

Como se mencionó en la sección anterior, la principal limitante de las tecnologías desarrolladas para prevenir los accidentes de tránsito, específicamente accidentes con los peatones, radica en los algoritmos empleados, por lo que es imprescindible el desarrollo de mejores algoritmos para la detección de peatones con el fin de reducir estos accidentes.

En esta sección se mencionan brevemente algunos conceptos así como los algoritmos más conocidos y empleados para la detección de peatones con el fin de comprender por qué es complicado su procesamiento.

2.2.1 Arquitectura general

Básicamente, un algoritmo completo para la detección de peatones se puede dividir en 4 partes o módulos, cada uno de ellos con algoritmos propios encargados del procesamiento de dicho módulo.

- **Preprocesamiento:** toma los datos de entrada (imágenes) y los prepara para su posterior procesamiento. En este módulo se emplean algoritmos para filtrar ciertas características de interés de la imagen, como algún canal de color (rojo, verde, azul, etc.) y, principalmente, algoritmos para la detección de bordes, como son el algoritmo Canny o Sobel, entre otros.

- **Generación de candidatos:** extrae regiones de interés (candidatos) de la imagen para ser enviados al módulo de clasificación evitando las regiones donde no aparecen peatones.
- **Clasificación:** recibe una lista de candidatos para ser clasificados como peatón o no peatón.
- **Verificación y refinamiento:** verifica y refina los candidatos clasificados como peatones, referidos como detecciones, y descarta los falsos positivos.

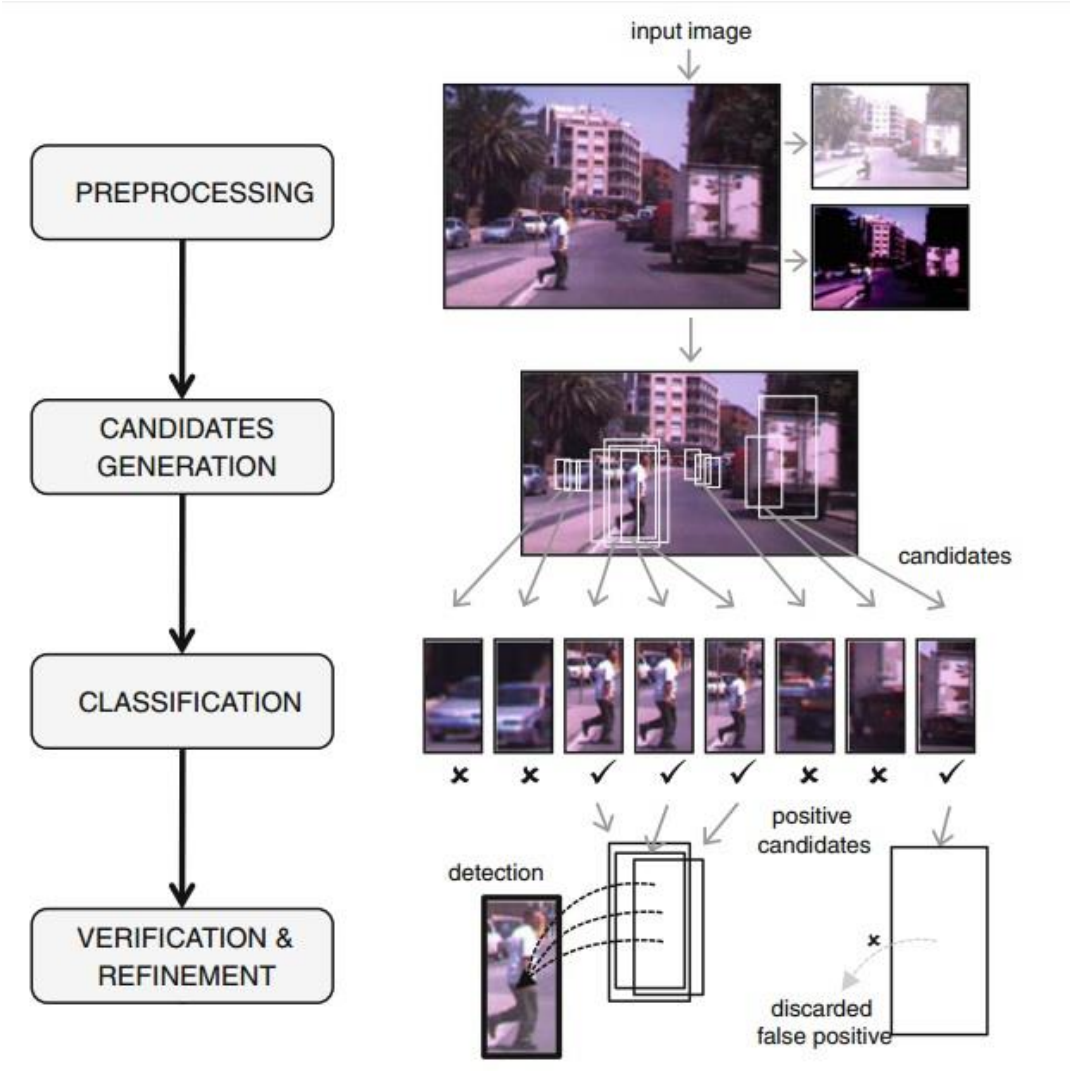


Figura 2.6: Arquitectura general de un algoritmo para la detección de peatones [9]

Algunos algoritmos omiten el último módulo, con el fin de reducir la cantidad de procesamiento, confiando plenamente en los candidatos arrojados por el clasificador, el cual es la parte más compleja del algoritmo.

2.2.2 Máquinas de soporte vectorial (SVM)

Las máquinas de soporte vectorial, o SVM por sus siglas en inglés, son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios AT&T®.

Los SVM son potentes clasificadores empleados en múltiples aplicaciones. Para el caso de la detección de peatones, se emplean ampliamente junto con algoritmos que extraen características de una imagen para que luego el SVM las clasifique como peatón o no peatón.

Dado un conjunto de puntos (o características), en el que cada uno de ellos pertenece a dos categorías (peatón y no peatón), el SVM construye un modelo capaz de predecir si un punto nuevo pertenece a una categoría o a la otra.

En otras palabras, primeramente hay que «entrenar» al SVM con una vasta base de datos (aprendizaje supervisado) conformada por características extraídas de imágenes en las cuales en unas aparece un peatón y en otras no. Con esta información, el SVM construirá hiperplanos (o funciones) en los cuales quedarán ordenadas las características en su respectiva categoría.

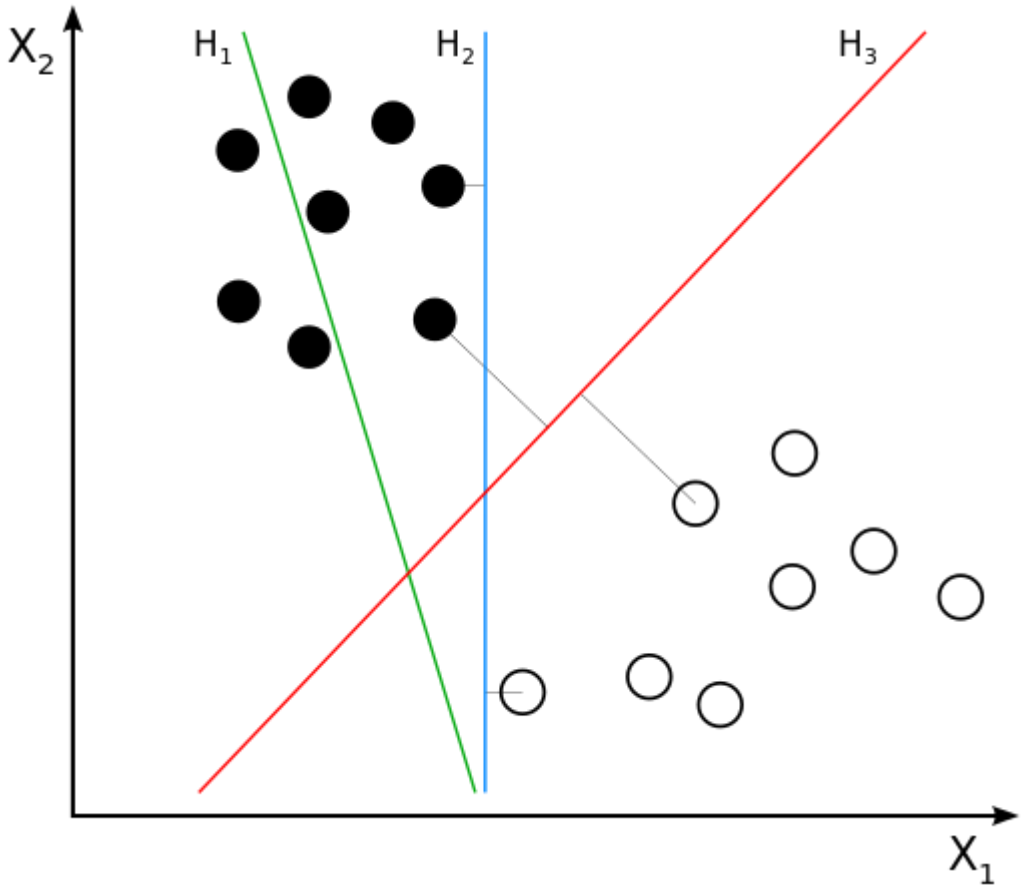


Figura 2.7: SVM clasificando los datos en 3 posibles hiperplanos

En la figura 2.7, se pueden considerar los puntos negros como las características en donde no aparece un peatón, y los blancos donde sí. De los tres hiperplanos arrojados por el SVM, el hiperplano 3 (H_3, rojo) sería el que mejor clasifica las características.

Con este clasificador construido, o entrenado, al introducir una nueva imagen se podría saber con facilidad si hay o no un peatón.

Los SVM pueden calcular hiperplanos unidimensionales (líneas rectas), como es el caso de la figura 2.7, bidimensionales, o de cualquier cantidad de dimensiones, aumentando así la precisión de la clasificación, pero a la vez también aumentando la complejidad del algoritmo y dependiendo del número de características consideradas.

Las desventajas de este algoritmo es que requiere efectuar muchos cálculos matemáticos para poder determinar los hiperplanos, además, se requiere de una vasta base de datos, por lo general de cientos de imágenes, para poder construir un SVM robusto.

Otra desventaja es que los SVM por sí solos no son algoritmos para la detección de peatones, únicamente son clasificadores, por lo que se necesita de otros algoritmos para la extracción de las características, lo que implica un mayor procesamiento.

La ventaja es que el entrenamiento sólo se realiza una vez, y haciéndolo con una vasta base de datos se obtiene un potente clasificador.

2.2.3 Características Haar

M. Oren *et al.* [14] propusieron un algoritmo para la extracción de características por medio de las wavelets de Haar.

Este algoritmo implementa tres tipos de wavelets de Haar (filtros), vertical, horizontal y diagonal, para eliminar los detalles de una imagen y dejar sólo las características relevantes (bordes).

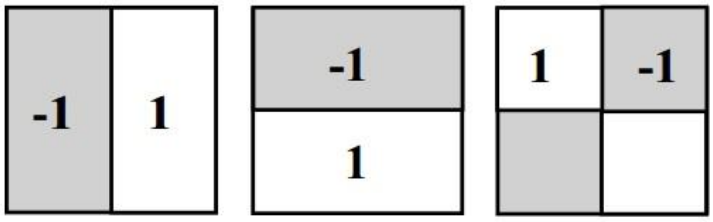


Figura 2.8: Tres tipos diferentes de wavelets de Haar que utiliza el algoritmo [14]

En imágenes a color, el algoritmo se debe implementar por separado a cada canal (R, G y B).

Se debe fijar una escala (coeficientes), especificado en [14], para definir la cantidad de detalle que se busca obtener (o suprimir).

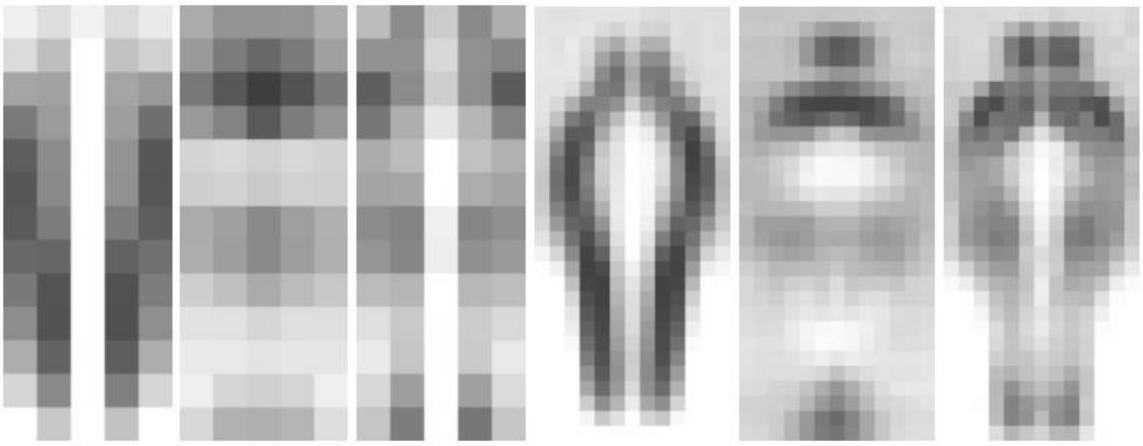


Figura 2.9: Imagen representada por características Haar utilizando dos escalas diferentes [14]

En la figura 2.9 se muestran los resultados de la imagen de una persona filtrada con los tres tipos diferentes de wavelets. Las primeras tres, de izquierda a derecha, con una escala de 32x32 pixeles, y las últimas tres con una de 16x16 pixeles (mayor detalle).

Una vez habiendo eliminado el detalle deseado, se deberán normalizar los pixeles de las imágenes obtenidas con las tres wavelets y concatenarlas para así obtener el vector de características.

Estas características se utilizarán para entrenar un SVM y construir el clasificador.

Las desventajas de este algoritmo es que genera mucha información (características) debido a que emplea tres wavelets diferentes para abarcar los detalles horizontales, verticales y diagonales de la imagen, además de la necesidad de un SVM.

2.2.4 Histograma de gradientes orientados (HOG)

El histograma de gradientes orientados, o HOG por sus siglas en inglés, es un algoritmo para la extracción de características basadas en la orientación de los gradientes de la imagen, diseñado por N. Dalal *et al.* [15].

A grandes rasgos, primeramente el algoritmo debe calcular los gradientes de cada uno de los pixeles de la imagen, así como su magnitud y dirección, mediante las siguientes ecuaciones [16]:

$$f_x(x,y) = f(x+1,y) - f(x-1,y)$$

$$f_y(x,y) = f(x,y+1) - f(x,y-1)$$

$$m(x,y) = \sqrt{{f_x}^2(x,y) + {f_y}^2(x,y)}$$

$$\theta(x,y) = arctan\left(\frac{f_y(x,y)}{f_x(x,y)}\right)$$

Después, se deberá dividir la imagen en bloques, y cada bloque en celdas. El tamaño de estos bloques y celdas dependerá del tamaño de la imagen, aunque el autor recomienda para la mayoría de los casos utilizar celdas de 8x8 pixeles y bloques de 4 celdas adyacentes.

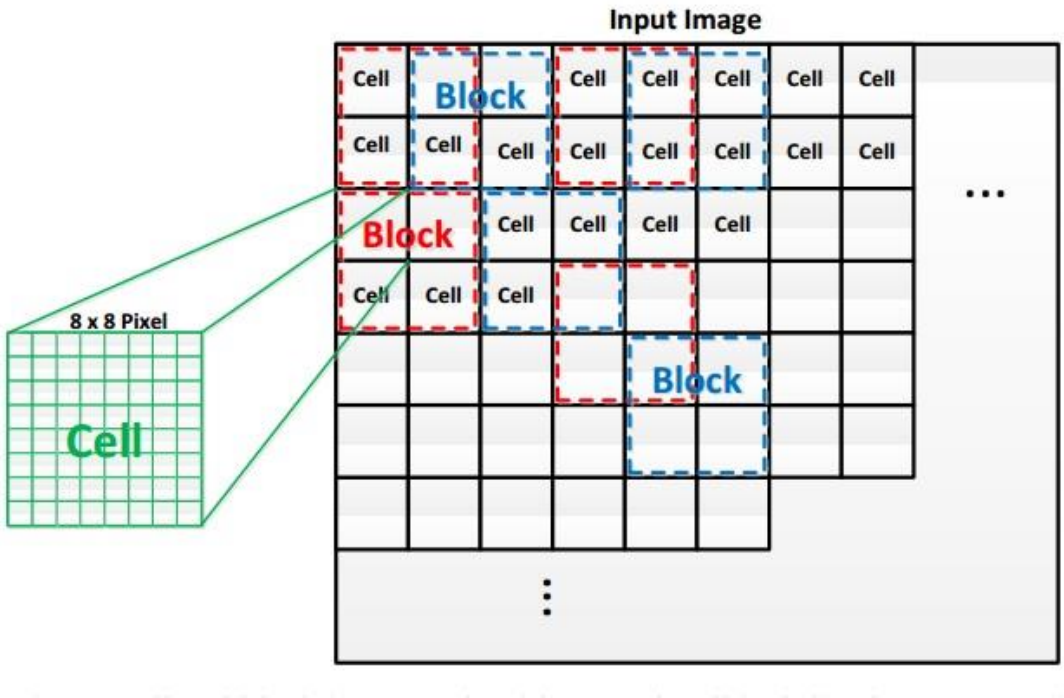


Figura 2.10: HOG, división de la imagen en bloques y celdas [16]

Una vez teniendo los bloques y celdas definidos, se generará un histograma de cada celda sumando las magnitudes de los gradientes que estén dentro de un mismo intervalo definido de orientación del gradiente. El autor recomienda trabajar con 9 intervalos de orientación como se muestra en la figura 2.11.

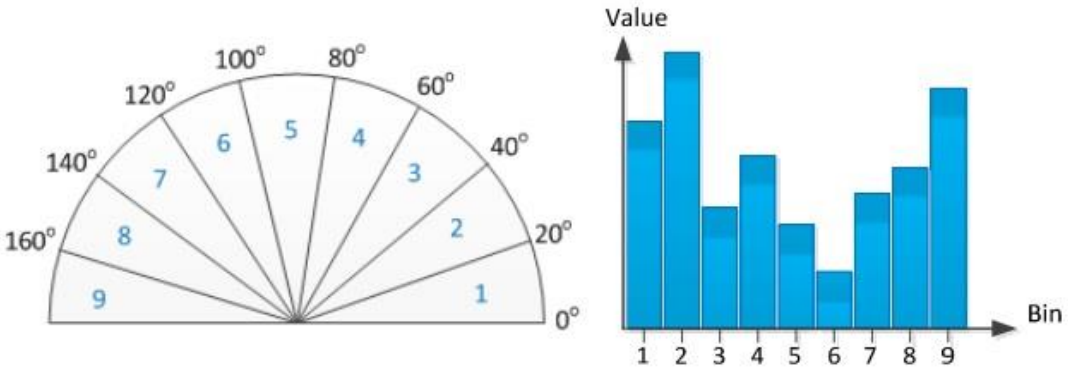


Figura 2.11: HOG, histograma de una celda [16]

Después, se deberán normalizar los histogramas obtenidos con cualquiera de los métodos L1-norm, L2-norm, L1-sqrt o L2-hys.

Finalmente, se acomodarán los histogramas de cada celda como se especifica en [15] para obtener así el vector de características.

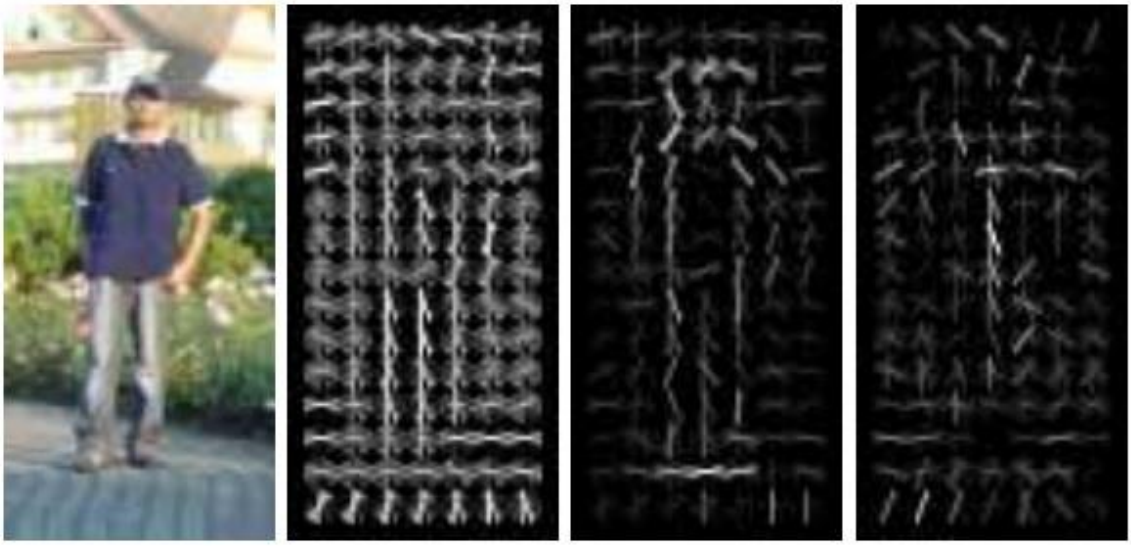


Figura 2.12: Imagen representada por características HOG utilizando tres tamaños de bloque diferentes [15]

Al igual que el algoritmo de Haar, este algoritmo necesitará de un SVM como clasificador para poder definir si hay o no un peatón en la escena.

El HOG, es el algoritmo más conocido y usado para la detección de peatones, y de otros objetos, debido a su efectividad.

Las desventajas de este algoritmo es que se necesitan procesar operaciones como raíces cuadradas y arcos tangentes en cada pixel de la imagen las cuales resultan laboriosas para los procesadores, además de que se necesita de un SVM.

2.2.5 Múltiples partes del cuerpo

La combinación de algoritmos para extraer características junto con algoritmos clasificadores genera un valioso modelo del peatón (o persona), discriminando los detalles irrelevantes como el fondo.

Sin embargo, al basar estos algoritmos en modelos de una persona «completa», se tiene la desventaja de que en la imagen deberá aparecer el peatón completo y bien definido para poder ser reconocido. Esto es que si en la imagen aparece alguna parte del peatón (pierna, brazo, etc.) borrosa, tapada o con poco contraste, éste no podrá ser reconocido. Algo bastante difícil de evitar en una ciudad promedio, llena de personas y objetos atravesándose, con cambios de clima e iluminación, además de que tanto el automóvil como el peatón están en movimiento.

Es por esto que se empezó a trabajar con estos algoritmos pero basándolos ahora en modelos de las partes principales del cuerpo de una persona, como la cabeza, las piernas, los brazos o el tronco.

El objetivo es detectar cuantas partes sea posible para así con ellas construir un modelo parcial de una persona. El algoritmo, en base a las partes detectadas, determinará si se tiene o no la suficiente información para concluir que hay un peatón en la escena.

Es muy común que las personas lleven algún objeto cargando, lo cual altera su forma de persona. Sin embargo, gracias a este enfoque, al detectar las piernas y la cabeza, se concluiría que en efecto es una persona.

A. Mohan *et al.* [17] hicieron una evolución del algoritmo Haar [14] orientándolo a cuatro partes diferentes del cuerpo. A. Shashua *et al.* [18] implementaron el algoritmo HOG con doce ventanas orientadas a diferentes partes del cuerpo. Ambos trabajos obtuvieron notables mejoras frente a sus predecesores.

En este trabajo de tesis se utiliza este mismo enfoque, basado en partes separadas del cuerpo, debido a sus ventajas frente a un enfoque basado en el cuerpo completo de una persona. Además, al trabajar cada parte del cuerpo por separado, el algoritmo se vuelve altamente paralelizable.

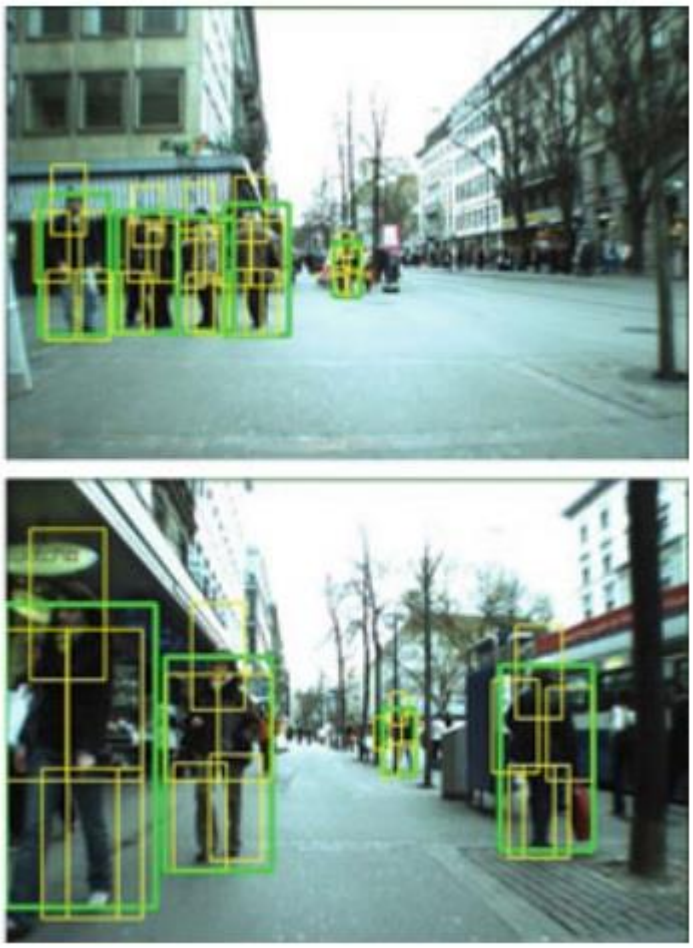


Figura 2.13: Algoritmos orientados a diferentes partes del cuerpo [9]

2.3 Unidad de procesamiento gráfico (GPU)

Una vez teniendo un algoritmo con el cual trabajar, se necesita de algún dispositivo que lo ejecute.

Una unidad de procesamiento gráfico, o GPU por sus siglas en inglés, es un coprocesador que interactúa con el procesador central, o CPU, de un ordenador (PC) encargándose del procesamiento gráfico y las operaciones de punto flotante (FLOPS). Esto con el fin de aligerar la carga de trabajo del CPU.

Actividades relacionadas con el procesamiento gráfico como la reproducción de video, el renderizado, los videojuegos, entre otras, las puede llevar a cabo la GPU, mientras el CPU puede dedicarse a otras tareas como la ejecución de los programas, con lo que se consigue una notable mejora en rendimiento del ordenador en comparación con los ordenadores que no tengan GPU.

Fueron desarrolladas a finales de la década de 1980 como descendientes de los chips gráficos monolíticos.

Los principales fabricantes de GPUs en la actualidad son AMD® y NVIDIA®.

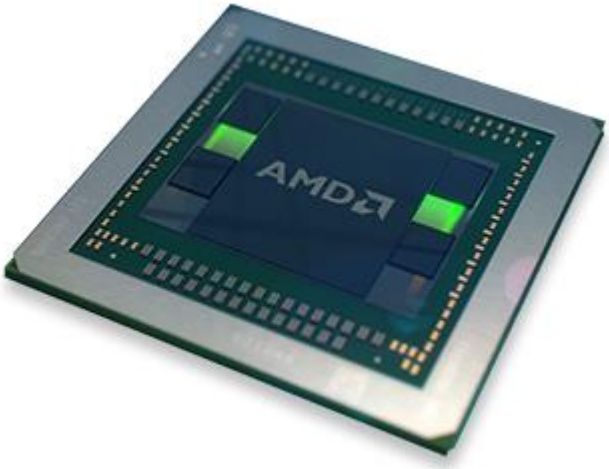


Figura 2.14: GPU fabricada por AMD® [19]

Las GPUs se comunican con el CPU por medio del protocolo Peripheral Component Interconnect Express, abreviado PCI-E o PCIe.

En cuanto a la arquitectura, las GPUs se caracterizan por ser altamente segmentadas, lo que significa que poseen una gran cantidad de unidades funcionales, mucho mayor a las de un CPU. Gracias a esto, las GPUs ofrecen un destacado nivel de paralelismo.

Como comparativa, el CPU Intel® Core i7 4770K puede realizar hasta 8 procesos en forma paralela (debido a sus 4 núcleos y 2 hilos de ejecución) [20], mientras que la GPU AMD® Radeon R9 290X puede realizar hasta 2816 procesos en forma paralela (debido a sus 44 unidades de cómputo y 64 sombreadores) [19]. Una considerable diferencia de paralelismo.

Gracias al alto nivel de paralelismo que ofrecen las GPUs, entre otras prestaciones, éstas se empezaron a utilizar en aplicaciones fuera del procesamiento gráfico, especialmente en el ámbito científico y de simulación.

Actualmente, las GPUs se utilizan en simulaciones Monte Carlo, predicción del tiempo, procesamiento de audio, Mecánica Cuántica, Astrofísica, visión por computadora, procesamiento de señales, Control, redes neuronales, entre muchas otras áreas y aplicaciones.

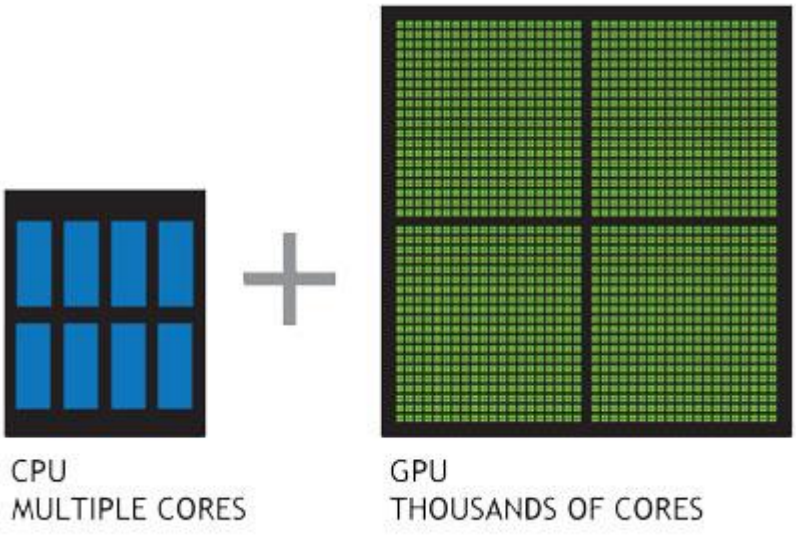


Figura 2.15: Comparativa entre el paralelismo de un CPU y una GPU [21]

Debido a esta gran aceptación y desempeño que han tenido las GPUs, surgió el término «unidad de procesamiento gráfico de propósito general», o GPGPU por sus siglas en inglés, lo que significa que hoy en día las GPUs se utilizan no sólo para el procesamiento gráfico, si no en muchas otras aplicaciones.

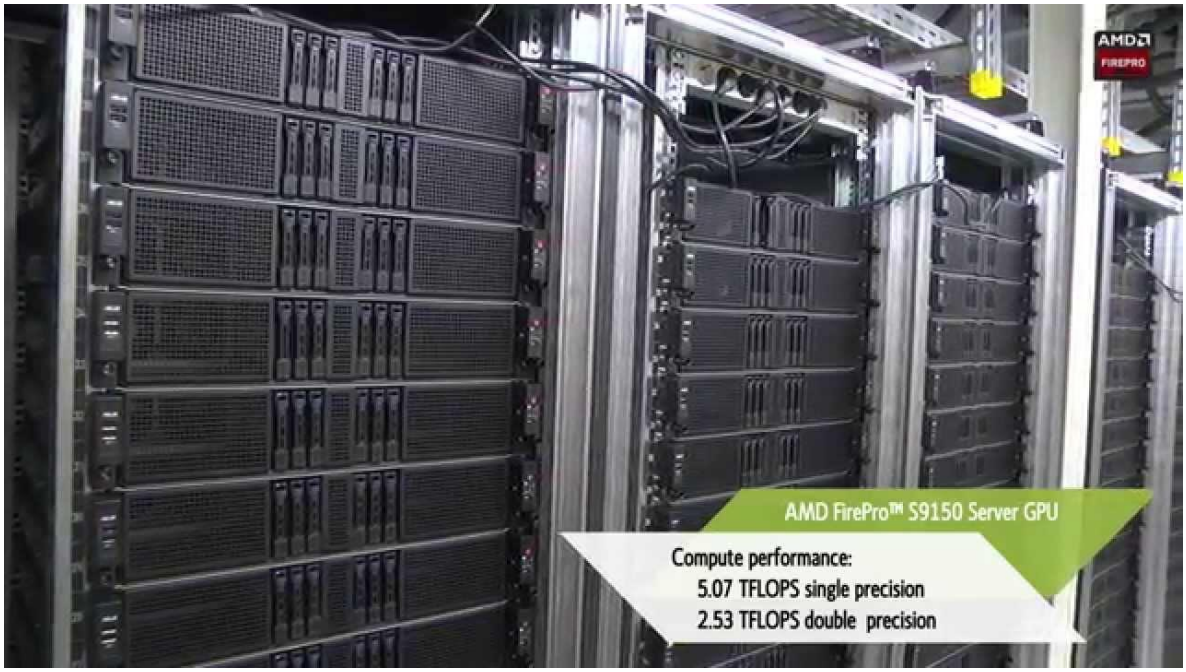


Figura 2.16: Supercomputadora Lattice-CSC conformada por 640 GPUs [22]

2.4 OpenCL

Existen múltiples herramientas para la interacción entre persona y CPU, herramientas con las cuales una persona puede indicarle (programar) a un CPU que realice cierta tarea. Estas herramientas son conocidas como lenguajes de programación, y ejemplos de ellos son C, Java, Python, entre otros.

Para el caso de las GPUs, existen herramientas similares que sirven para poder programar a estos dispositivos para que realicen una tarea específica. Ejemplos de estas herramientas son BrookGPU, Sh, CUDA y OpenCL, siendo estas últimas dos las más importantes y más usadas.

Open Computing Language, abreviado OpenCL, es un entorno de desarrollo que consta de una interfaz de programación de aplicaciones (API) y de un lenguaje de programación. Juntos permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en CPUs como en GPUs, además de otros procesadores como DSPs, FPGAs, y APUs, entre otros, lo que se conoce como programación heterogénea.

El lenguaje está basado en C99, eliminando cierta funcionalidad y extendiéndolo con operaciones vectoriales. Debido a esto, para poder utilizar esta herramienta no es necesario el conocimiento de un nuevo lenguaje de programación para las personas familiarizadas con C.

Apple® creó la especificación original y fue desarrollada en conjunto con AMD®, IBM®, Intel® y NVIDIA®, entre otros.

Apple® la propuso al Khronos Group para convertirla en un estándar abierto y libre de derechos. El 16 de junio de 2008 Khronos creó el “Compute Working Group” para llevar a cabo el proceso de estandarización. En 2013 se publicó la versión 2.0 del estándar.

Gracias al entorno de OpenCL, se pueden desarrollar programas que puedan ser ejecutados por diferentes procesadores, aprovechando los recursos que ofrece cada uno como el paralelismo, velocidad, memoria, etc., para poder realizar tareas que serían altamente demandantes o excesivas para un CPU.

OpenCL aprovecha el paralelismo de los dispositivos, como las GPUs, al repartir las tareas en las unidades de cómputo o núcleos de los dispositivos, que OpenCL denomina como work-item.

Las principales ventajas de OpenCL frente a CUDA y otros entornos de desarrollo para GPUs son:

- Es un estándar abierto, libre de licencias.

- Permite computación paralela en GPUs, así como también en CPUs, DSPs, FPGAs y otros procesadores independientemente del fabricante.
- Emplea un subconjunto del lenguaje C99.

Figura 2.17: Logotipo del estándar OpenCL [23]

La supercomputadora Lattice-CSC de la figura 2.16, la supercomputadora más eficiente del mundo energéticamente, es programada completamente por medio de OpenCL, en su versión 2.0 [22], lo que demuestra el potencial, importancia y aceptación que tiene este estándar.

2.5 Estado del arte

Múltiples trabajos y tecnologías sobre el tema se han desarrollado en estos últimos años, además de los ya presentados.

A continuación se hace un compendio de los desarrollos más importantes llevados a cabo hasta la fecha, para tener un panorama general de lo que ya se ha hecho y los retos que hay por delante.

- El Volvo® S80 2015 (entre otros modelos) implementa un sistema que alerta al conductor (ADAS) en caso de que el automóvil se salga del camino, o en caso de que se cambie de carril sin haber activado las direccionales antes. También alerta al conductor en caso de posible colisión contra un automóvil, contra un ciclista o contra un peatón, siempre y cuando éstos se encuentren exactamente delante del automóvil. Además, el sistema podrá frenar el automóvil en caso de que el conductor no reaccione ante estas advertencias. En el caso del ciclista y el peatón, el sistema sólo es efectivo hasta los 50Km/h, mientras que con el automóvil sólo mientras exista una diferencia de velocidades de 4 a 15Km/h [8].

- En su fase de desarrollo, el vehículo autónomo de Google® ha estado involucrado en 12 accidentes de tránsito. El vehículo no ha sido probado en condiciones de lluvia o nieve. Presenta dificultades al reconocer objetos que no representen un peligro para vehículo, como la basura, por lo que constantemente está girando innecesariamente para esquivar el objeto. El vehículo no es capaz de reconocer cuando un oficial le indica que se detenga. Google® asegura que todo esto será solucionado y que el vehículo estará a la venta al público en el año 2020 [24].
- R. Sun *et al.* [25] implementaron un algoritmo para la detección de peatones basado en HOG en una GPU AMD® y en una NVIDIA® por medio de OpenCL, obteniendo 36 cuadros por segundo (FPS) con una resolución de 768x576.
- R. Benenson *et al.* [26] mejoraron el algoritmo HOG y lo implementaron en una GPU NVIDIA® obteniendo 100 FPS en promedio, y hasta 135 FPS con algunas restricciones.
- S. Bauer *et al.* [27] desarrollaron un sistema de múltiples sensores para la detección de peatones junto con un FPGA y una GPU en un mismo sistema. Utilizaron el algoritmo HOG, el cual es procesado por el FPGA, y un SVM gaussiano procesado por la GPU. Trabajaron con imágenes de 320x240 pixeles de resolución a 25 FPS.
- M. Hahnle *et al.* [28] implementaron un algoritmo para la detección de peatones basado en HOG y SVM en un FPGA, alcanzando 64 FPS con imágenes HD, 1920x1080 pixeles.
- La NVIDIA® Tesla K80 tiene un desempeño de 2.91 TFLOPS en precisión doble, 8.74 TFLOPS en precisión simple y cuenta con 4992 núcleos CUDA, convirtiéndola en la GPU más potente comercialmente hasta la fecha [29].

CAPÍTULO 3

DESCRIPCIÓN GENERAL

Debido a los antecedentes y hechos presentados en el capítulo 2, fue que se decidió desarrollar un algoritmo simple para la detección de peatones con el fin de que las tecnologías que lo implementen (vehículos autónomos, etc.) tengan un tiempo de respuesta menor y puedan operar a mayores velocidades.

Las GPUs, debido a sus prestaciones, resultan sumamente atractivas como dispositivos de implementación de este algoritmo, mientras que OpenCL es el entorno ideal para hacerlo debido a sus ventajas frente a otros.

3.1 Visión humana

El sentido más importante que posee el humano, así como otras especies, es el de la vista. Tres cuartas partes de la capacidad de procesamiento del cerebro humano están dedicadas únicamente a la visión, lo que muestra que es un sistema sumamente complejo.

En primera instancia, el ojo utiliza las células llamadas bastones. Estas células son extremadamente sensibles a la luz lo que provoca que se saturen rápidamente y no sean capaces de detectar colores. Sin embargo, son excelentes para detectar los pequeños contrastes de luz que hay entre objetos, por lo que son capaces de enviar al cerebro una imagen en blanco y negro resaltando los contornos contrastantes de los objetos. Esta imagen es de suma importancia para el cerebro, ya que elimina los numerosos detalles y muestra únicamente la forma que tienen los objetos [2].

Después, el ojo utiliza las células llamadas conos las cuales son capaces de detectar colores. Algunos son sensibles a la luz roja, otros a la luz verde y otros a la luz azul, por lo que en conjunto pueden detectar millones de colores diferentes. Estas células envían al cerebro una imagen a color sumamente detallada y compleja [2].

Figura 3.1: Representación de una imagen captada por los bastones

El cerebro analiza las imágenes captadas por los bastones y los conos para extraer características particulares y compararlas con lo que conoce para así identificar los objetos.

Este sistema entre ojo y cerebro se tomó como principal referencia para el desarrollo de este algoritmo para la detección de peatones.

3.2 Funcionamiento general del algoritmo

El funcionamiento del algoritmo se basa en detectar tres partes imprescindibles del cuerpo de una persona: las piernas, los brazos y la cabeza.

Primeramente, a partir de una imagen a color capturada por una cámara, se deberán extraer los bordes de la imagen para eliminar el detalle y trabajar únicamente con los contornos de los objetos (funcionamiento de los bastones).

Las piernas y los brazos de una persona pueden ser consideradas como dos segmentos paralelos horizontales o verticales, con un rango de separación constante entre sí determinada por el ancho de una pierna o brazo promedio, y una longitud también constante determinada por el largo de una pierna o brazo promedio.

Con la imagen de bordes, se procederá a buscar estos patrones. En caso de encontrarse, se podrá concluir que se trata de un brazo o una pierna dependiendo de la altura en la que se haya encontrado (un brazo se encuentra más arriba que una pierna).

Gracias a esta interpretación se pueden detectar los brazos y piernas de una persona sin necesidad de una imagen detallada o a colores.

La detección de la cabeza no resulta tan simple por medio de los bordes. Es por esto que para la detección de la cabeza se utilizará la imagen a color (funcionamiento de los conos).

Debido a que la gran mayoría de personas llevan la cara descubierta al caminar por las ciudades, se buscará en la imagen una gama de colores que coincidan con el color de la piel. Esta búsqueda se hará únicamente a la altura donde se espera encontrar una cabeza. En las zonas donde se encuentre una gran concentración de estos colores y coincidan con el alto y ancho de una cara promedio, se podrá concluir que se trata de una cabeza.

De esta forma el algoritmo podrá detectar las piernas, los brazos y la cabeza de una persona.

Es evidente que en una imagen se podrán encontrar segmentos horizontales y verticales que no representen un brazo o una pierna, como por ejemplo un árbol o un poste, y concentraciones de color piel que no representen una cabeza, como el color de una casa. Es por esta razón que el algoritmo, finalmente, con la información que se recolectó, deberá determinar si algunas de las posibles piernas, brazos y cabezas detectadas están alineadas en un intervalo determinado que coincida con la fisionomía de una persona. Así, se concluirá si se ha detectado o no una persona.

Este algoritmo elimina la necesidad de cálculos matemáticos complejos y de algoritmos de aprendizaje.

3.3 Segmentación de profundidades

Para que el algoritmo funcione es necesario conocer previamente las proporciones de una persona promedio, como el largo y ancho de las piernas, brazos y cabeza, así como el color piel de referencia, el ancho y la altura de la persona. Estas proporciones van variando dependiendo qué tan cerca o lejos se encuentre una persona.

Por esta razón, el algoritmo se deberá ejecutar a diferentes profundidades para cubrir todo el campo visible o zonas por donde pasará el vehículo.

En este trabajo de tesis se aplicará el algoritmo en tres distancias diferentes en las cuales se conocen las proporciones de una persona a dichas distancias gracias a una base de imágenes que se tomaron en la Cuidad de México, mediante una cámara con resolución VGA. Esto se abarcará más a detalle en el capítulo 3.12.

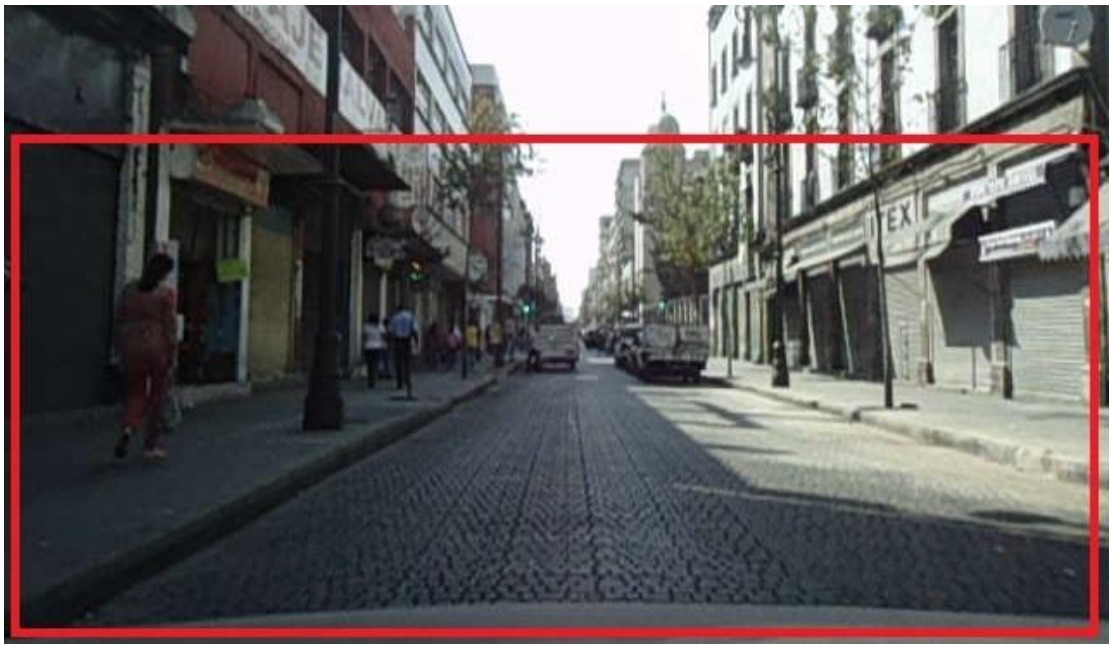


Figura 3.2: Profundidad de prioridad alta

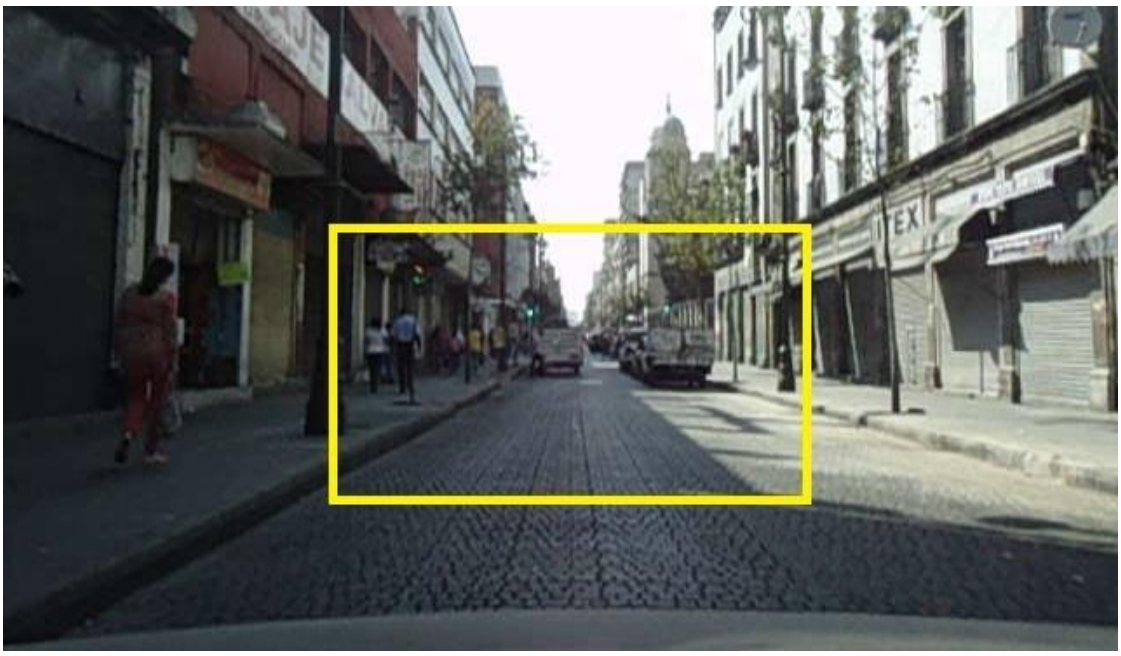


Figura 3.3: Profundidad de prioridad media

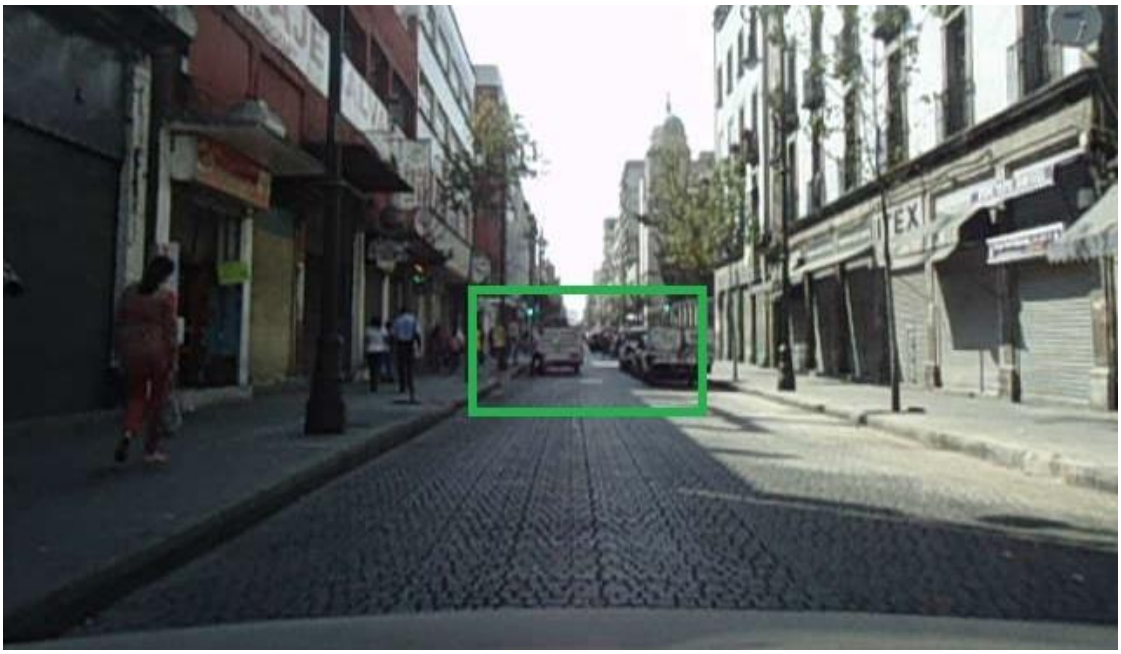


Figura 3.4: Profundidad de prioridad baja

La primera será la de prioridad alta, es decir, el área que se encuentra más cerca al vehículo y en la cual se corre el mayor riesgo de ser atropellado. La segunda es de prioridad media la cual se sitúa más atrás de la primera; sigue habiendo riesgo de atropello aunque en menor medida. La tercera es de prioridad baja y se encuentra alejada del vehículo; no hay tanto riesgo de atropello pero sirve como anticipación a un accidente. En las figuras 3.2, 3.3 y 3.4 se muestran estas áreas, respectivamente.

La zona de prioridad alta será la primera en analizarse para detectar el peatón lo antes posible y prevenir el accidente, seguida de la media y al final la baja. Después de analizar las tres zonas concluirá la ejecución del algoritmo para luego volverse a repetir en el siguiente cuadro (frame) o imagen enviada por la cámara.

Se utilizará la zona de prioridad alta en la mayoría de las imágenes de ejemplo de la descripción de este algoritmo ya que es la zona que se puede apreciar con mayor claridad.

3.4 Detección de bordes

Como se mencionó anteriormente, el primer paso en el algoritmo es extraer los bordes de la imagen.

Existen múltiples algoritmos para la extracción de bordes en una imagen. Algunos más complejos que otros, ofreciendo diferentes ventajas y desventajas.

En 1986, J. Canny [30] propuso un algoritmo el cual consistía en determinar la magnitud y dirección de los gradientes de las diferenciales de cada pixel en una imagen. Estos gradientes presentan mayores magnitudes y direcciones opuestas en condiciones de alto contraste, lo que facilita detectar las zonas contrastantes (bordes) y eliminar el resto.

Para obtener las gradientes de las diferenciales en los pixeles, existen múltiples operadores para ello, siendo el operador Sobel el más simple y utilizado.

A continuación se muestran los operadores Sobel (horizontal y vertical), así como las ecuaciones para calcular la magnitud y dirección de los gradientes.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G_{mag} = \sqrt{{G_x}^2 + {G_y}^2}$$

$$G_{dir} = arctan\left(\frac{G_y}{G_x}\right)$$

Para obtener mejores resultados, se le han hecho varias mejoras al algoritmo como aplicar un filtro gaussiano a la imagen antes de calcular los gradientes para reducir el ruido, aplicar el algoritmo de no-máxima supresión (non-maximum suppression) para que los bordes queden definidos por un solo pixel, así como también el algoritmo de doble umbral (double threshold) para determinar mediante dos constantes el rango de pixeles que serán eliminados y los que permanecerán como bordes.

Figura 3.5: Imagen obtenida con el algoritmo Canny

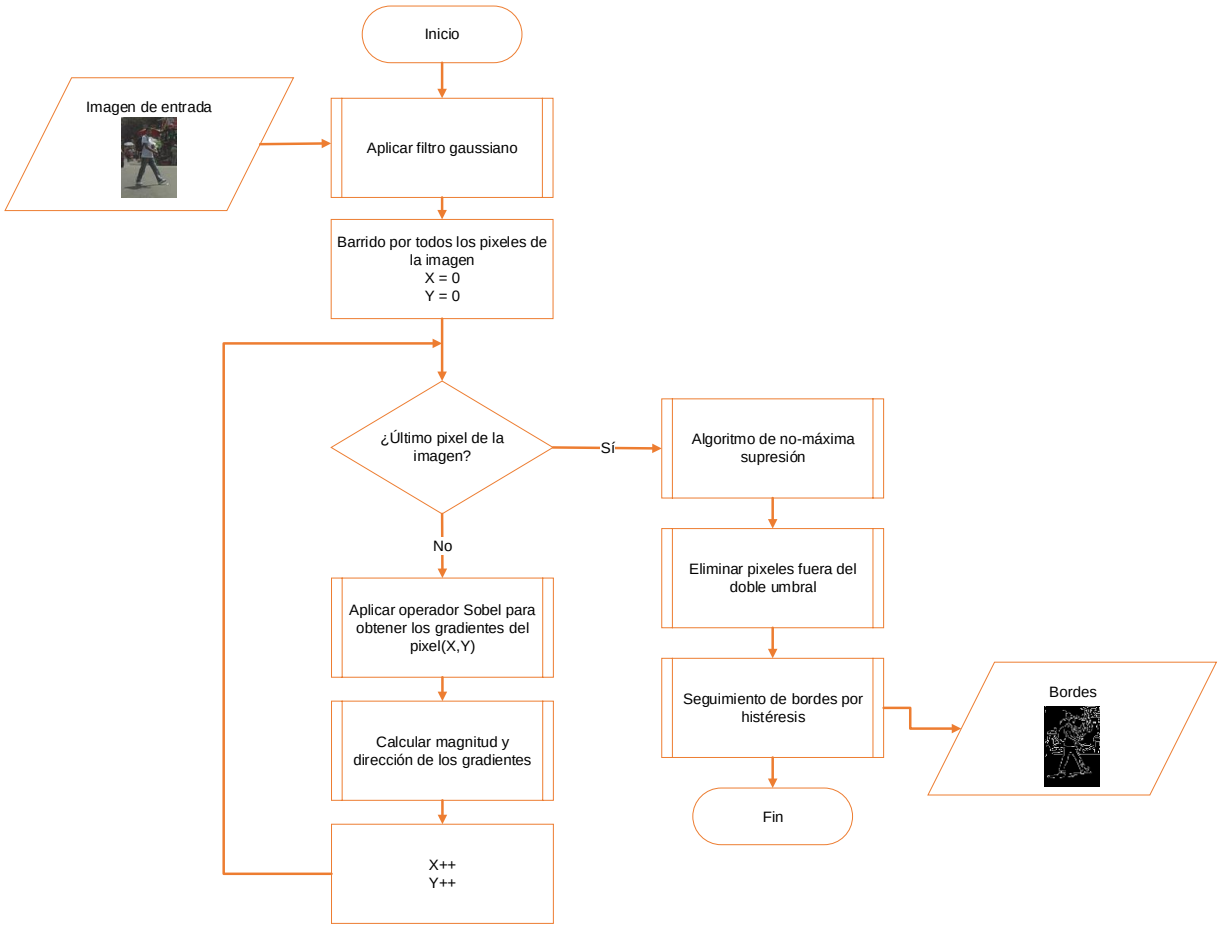


Diagrama 3.1: Algoritmo Canny para detección de bordes

MATLAB, la herramienta que se utilizó en los primeros prototipos de este algoritmo, emplea una función para la obtención de los bordes de una imagen por medio de diferentes algoritmos, los cuales son el algoritmo Sobel, Prewitt, laplaciano de Gauss (log), Roberts, Zero Cross y Canny.

En la figura 3.6 se muestran las imágenes de los bordes obtenidos aplicando estos diferentes algoritmos a una misma imagen. De izquierda a derecha, se muestra primeramente con el algoritmo Canny, seguido por el laplaciano de Gauss, Prewitt, Roberts, Sobel y Zero Cross.

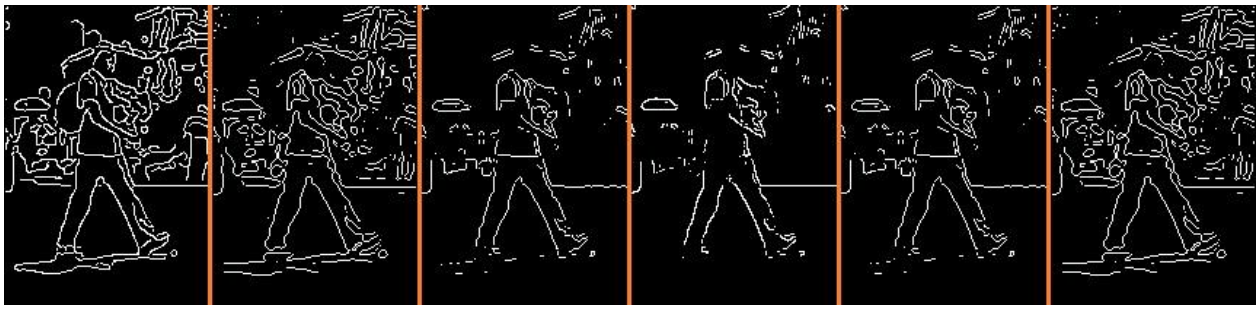


Figura 3.6 Detección de bordes por medio de diferentes algoritmos

Como se puede observar en la figura 3.6, con el algoritmo Canny es con el que se obtiene mejor definición de los bordes y segmentos más continuos. Por esta razón y debido a su relativa simplicidad fue que se decidió utilizar el algoritmo Canny como detector de bordes en este trabajo.

3.5 Segmentos horizontales y verticales

Anteriormente se mencionó que tanto las piernas como los brazos se pueden considerar como dos segmentos paralelos horizontales o verticales con un rango de separación constante.

Primeramente se hará la detección de todos los segmentos continuos, parcialmente horizontales o verticales, esto es, que tengan una continuidad de arriba hacia abajo o de izquierda a derecha, sin que tengan que llegar a ser exactamente horizontales o verticales.

Para lograr esto se deberá hacer un barrido por todos los pixeles de la imagen de bordes. Para el caso de extraer los segmentos horizontales, el barrido empezará de izquierda a derecha. Para el caso de los segmentos verticales, será de arriba a abajo.

Al encontrar un pixel en blanco (255 si la imagen es en enteros o 1 si la imagen está binarizada), a continuación se hará un nuevo barrido y se comprobará si existe otro pixel en blanco exactamente a su derecha, o a su derecha y un pixel arriba, o a su derecha y un pixel abajo, en el caso de los segmentos horizontales. Para los segmentos verticales se comprobará si existe un pixel en blanco exactamente abajo del pixel encontrado, o abajo y un pixel a la izquierda, o abajo y un pixel a la derecha.

Este proceso se volverá a repetir con los nuevos pixeles encontrados tantas veces dependiendo de la longitud de segmentos que queramos extraer. Si se llega a encontrar un pixel negro (0) se detendrá el nuevo barrido y se volverá a empezar con el siguiente pixel blanco encontrado.

En las figuras 3.7 y 3.8 se muestran ejemplos de este algoritmo descrito aplicado a una imagen de bordes.

En la figura 3.7 se muestra la extracción de los segmentos horizontales considerando segmentos de 30 pixeles como mínimo de largo. En la figura 3.8 se muestran los segmentos verticales de 40 pixeles como mínimo de largo.

Figura 3.7: Extracción de segmentos horizontales

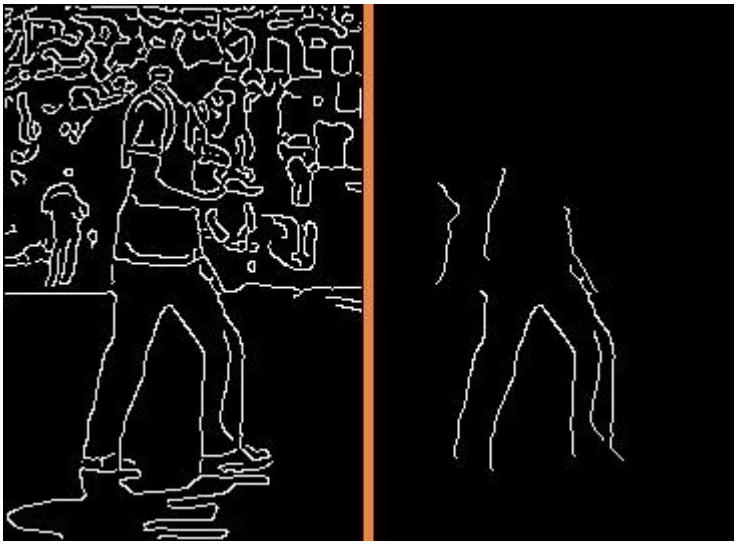


Figura 3.8: Extracción de segmentos verticales

Normalmente no se espera encontrar una pierna a la altura de la cabeza, o un brazo por donde está el suelo. Es por esto que para reducir tiempo de procesamiento y descartar información innecesaria, estos algoritmos deberán aplicarse únicamente a ciertas alturas donde se espere encontrar un brazo o una pierna.

En las figuras 3.9 y 3.10 se aplican estos algoritmos sólo en las zonas en donde se espera encontrar un brazo o una pierna, respectivamente.

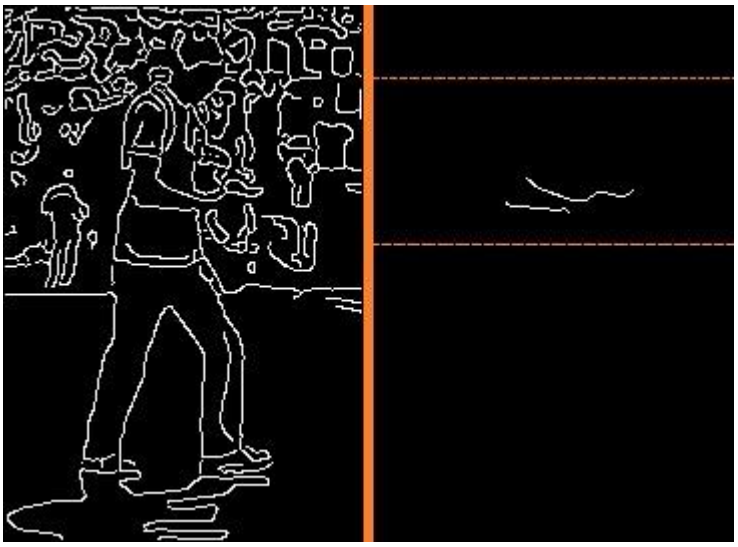


Figura 3.9: Algoritmo de segmentos horizontales aplicado únicamente en una región de interés

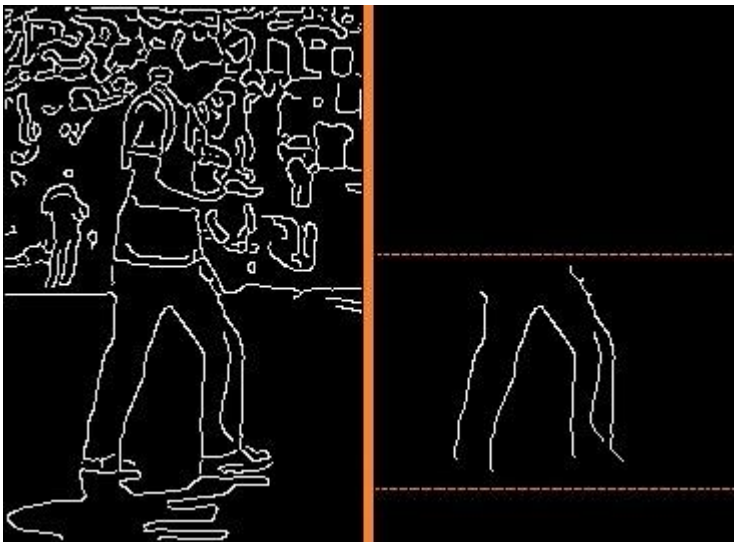


Figura 3.10: Algoritmo de segmentos verticales aplicado únicamente en una región de interés

Estos algoritmos de detección de segmentos horizontales y verticales serán la base para poder detectar un brazo o una pierna.

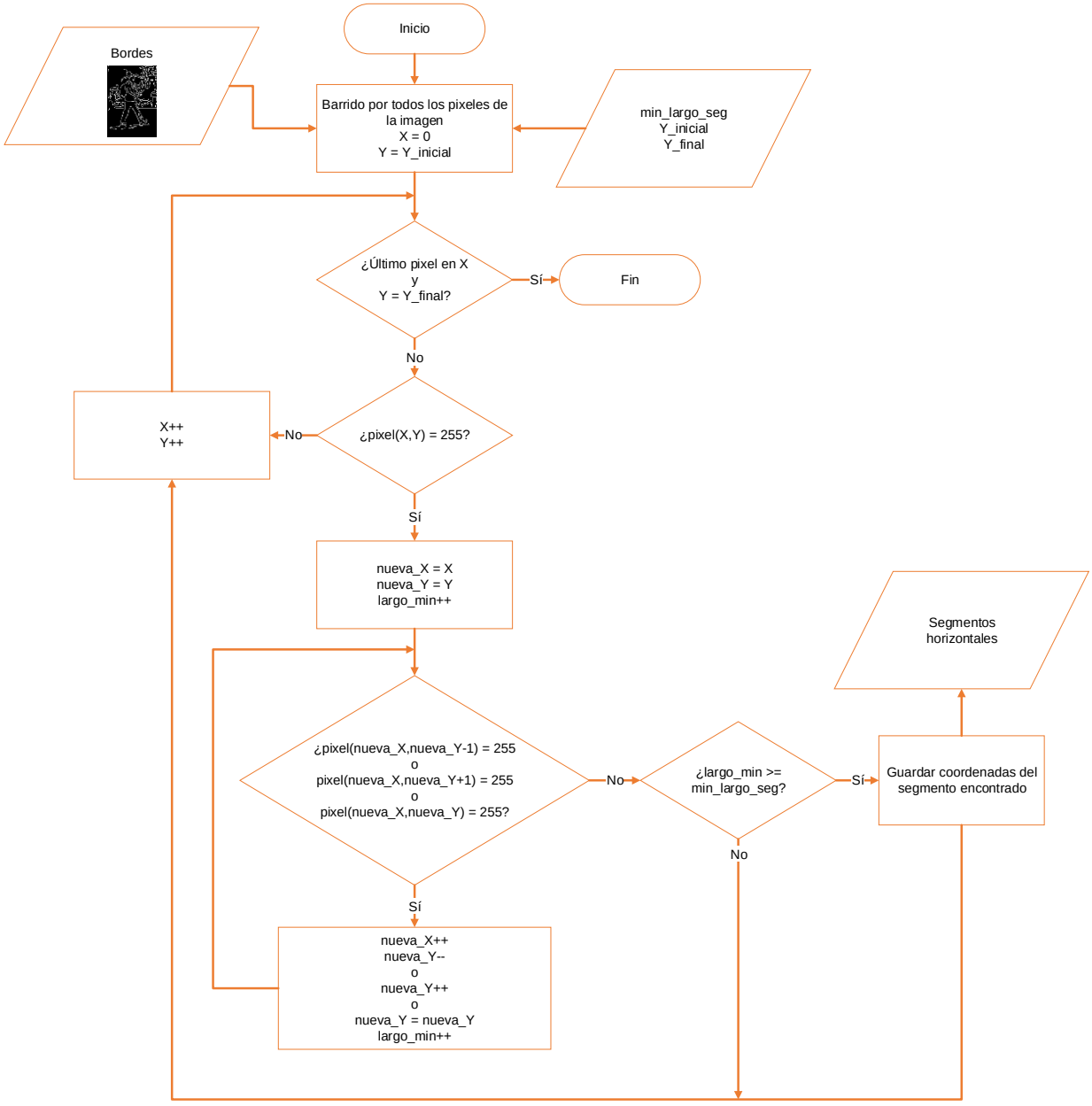


Diagrama 3.2: Detección de segmentos horizontales

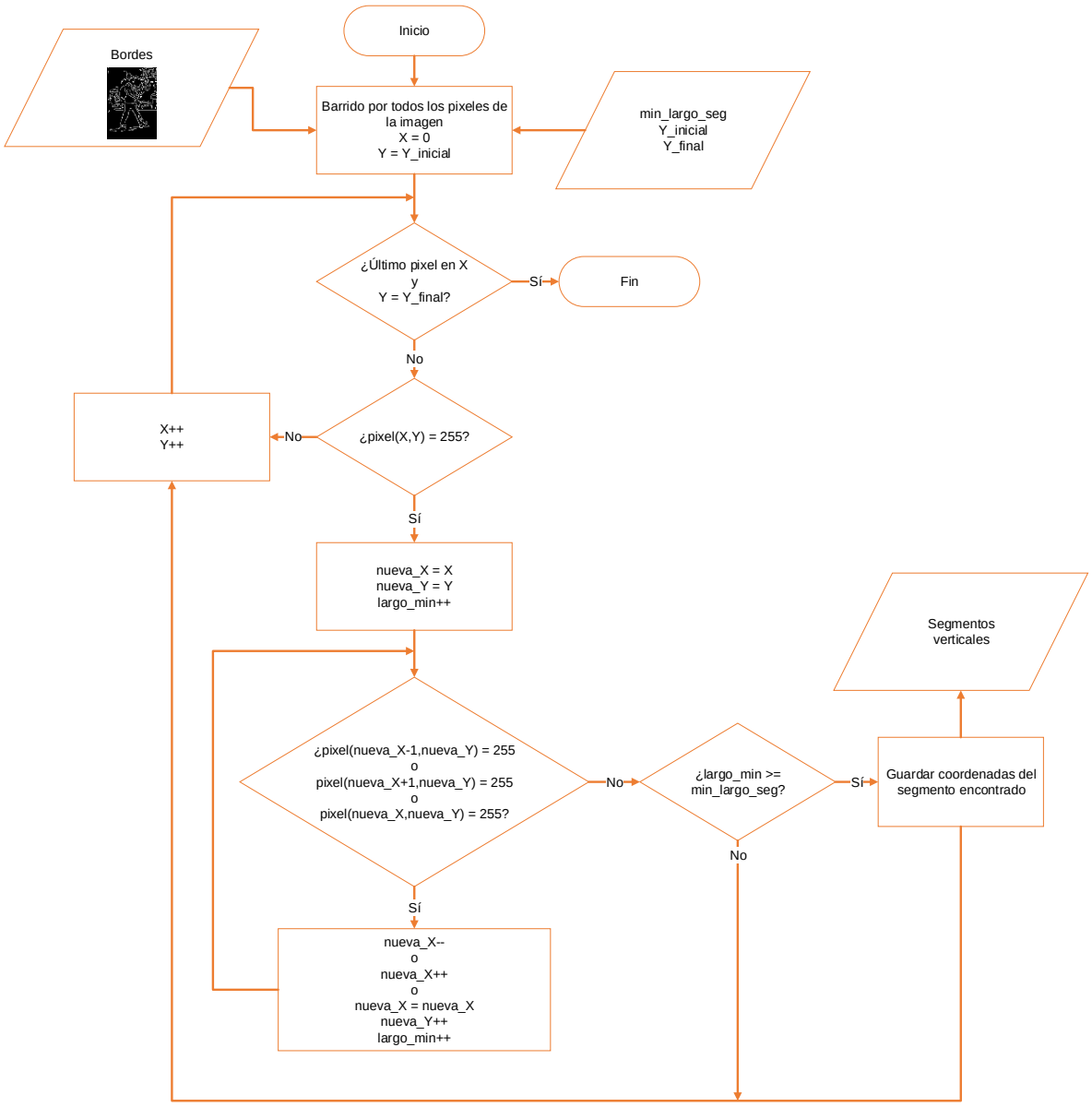


Diagrama 3.3: Detección de segmentos verticales

3.6 Detección de brazos

Una persona al ir caminando por las calles puede llevar los brazos caídos, esto es, de forma vertical. También puede ir sujetando algo, como un teléfono, por lo que para poder mirarlo su antebrazo quedará de forma horizontal. Debido a esto, para poder detectar un brazo se aplicarán los algoritmos para detectar segmentos tanto horizontales como verticales a la altura de los brazos.

Una vez teniendo todos los segmentos de interés, se comprobará si existen dos segmentos que tengan un rango de separación determinado por el ancho del brazo de una persona a la profundidad en la que se esté trabajando. Si un número de pixeles de cada segmento equivalente al largo del brazo de una persona se encuentran en este rango se concluirá que se tiene un posible brazo.

En la figura 3.11 se muestra un ejemplo del algoritmo de segmentos horizontales aplicado a una imagen en profundidad de prioridad media, y en la figura 3.12 el algoritmo de segmentos verticales en una de prioridad alta. Los puntos rojos indican el intervalo aceptable de separación entre segmento y segmento, mientras que la línea verde el largo mínimo de cada segmento que coincide con ese intervalo.

Figura 3.11: Posible brazo horizontal detectado

Figura 3.12: Posible brazo vertical detectado

Se almacenarán las coordenadas de cada pixel de estos segmentos detectados para su posterior análisis.

Diagrama 3.4: Detección de brazos horizontales

Diagrama 3.5: Detección de brazos verticales

3.7 Detección de piernas

Las piernas, a diferencia de los brazos, se encuentran de forma vertical la mayoría del tiempo que una persona va caminando por las calles. Es por esta razón que

para detectar las piernas únicamente se utilizará el algoritmo para detectar segmentos verticales.

Teniendo los segmentos de interés, se comprobará si existen dos segmentos con un rango de separación equivalente al ancho de una pierna promedio. Si un número de pixeles de cada segmento equivalente al largo de la pierna de una persona se encuentran en este rango se concluirá que se tiene una posible pierna.

En la figura 3.13 se muestra un ejemplo del algoritmo detectando las dos piernas de la persona. El segmento que se encuentra en medio de la pierna izquierda queda descartado ya que no cumple con el ancho de una pierna con respecto a los otros dos segmentos.

Figura 3.13: Posibles piernas detectadas

Los puntos rojos indican el intervalo aceptable de separación entre segmento y segmento, esto es, el ancho de una pierna promedio, mientras que la línea verde el largo mínimo de cada segmento que coincide con ese intervalo.

Figura 3.14: Una pierna detectada

En la figura 3.14 se muestra una sola pierna detectada ya que la pierna izquierda tuvo poco contraste con el fondo por lo que no apareció ese segmento en la detección de bordes.

Se almacenarán las coordenadas de cada pixel de estos segmentos detectados para su posterior análisis.

Diagrama 3.6: Detección de piernas

3.8 Marcaje de colores

Como se mencionó anteriormente, para la detección de la cabeza se utilizará la imagen a color y se buscarán en ella los pixeles que coincidan con el color de la piel. Se hará un barrido pixel por pixel y se comprobará si su color coincide con un rango de colores piel. Los pixeles de color que entren en este rango se marcarán en una segunda imagen en escala de grises, marcando el pixel encontrado de color blanco (255 o 1), mientras los pixeles que no coincida su color se dejarán de color negro (0).

Figura 3.15: Detección de colores piel

En la figura 3.15, en la parte derecha se muestran de color blanco los pixeles que coinciden con el color piel. Específicamente a esa profundidad se utilizó el color (90, 72, 67) como base y un intervalo de ±20% por canal. A profundidades más lejanas se tendrá que usar otro color como base ya que a mayores profundidades los colores tienden a oscurecerse debido a la menor resolución.

Al igual que con los segmentos horizontales y verticales, no tiene caso hacer la detección por toda la imagen, si no sólo a la altura por donde se espera encontrar una cabeza, como se muestra en la figura 3.16.

Figura 3.16: Algoritmo aplicado sólo en el área de interés

Esta imagen resultante a escala de grises se utilizará posteriormente para detectar la cabeza.

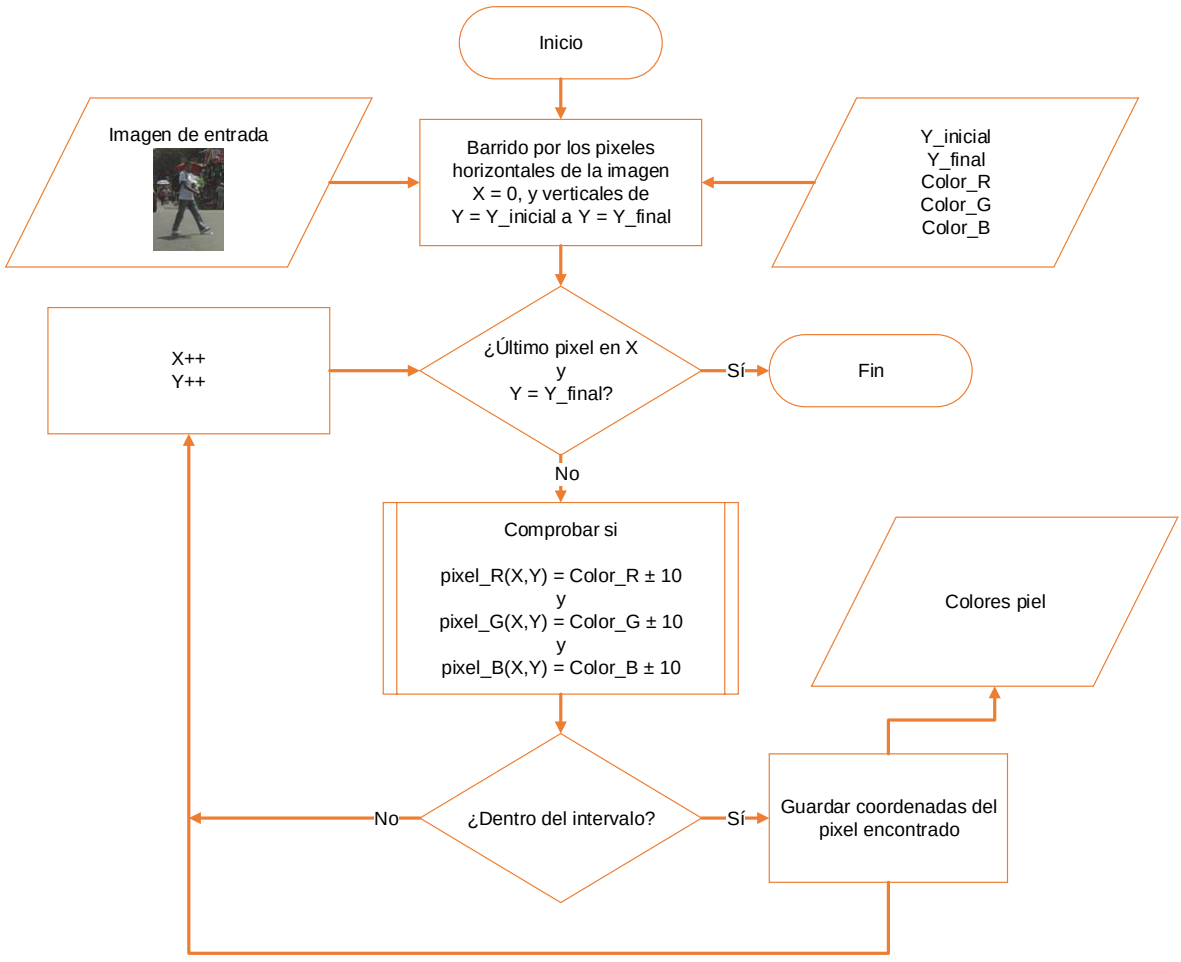


Diagrama 3.7: Detección de colores piel

3.9 Detección de cabezas

Teniendo la imagen con los pixeles de interés marcados, se procederá a hacer la detección de la cabeza. Para ello se hará un barrido por el área de interés y se hará un conteo de pixeles blancos de tal forma que en un área considerada como el ancho por alto de una cabeza promedio se obtenga un número determinado de pixeles blancos. Las áreas que contengan como mínimo ese número de pixeles serán consideradas como posibles cabezas.

En la figura 3.17 se muestra un ejemplo de una cabeza detectada. A esa profundidad se calculó que una cabeza deberá de ocupar un área de 40 x 50 pixeles, y por lo menos deberá haber 500 pixeles color piel (o blancos en la imagen a escala de grises) en esa área para considerarse como cabeza. Estas condiciones se cumplen en el recuadro.

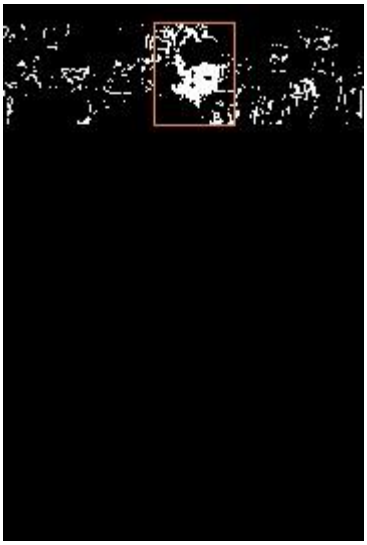


Figura 3.17: Cabeza detectada

Se almacenarán las coordenadas de los bordes de las posibles cabezas detectadas para su posterior análisis.

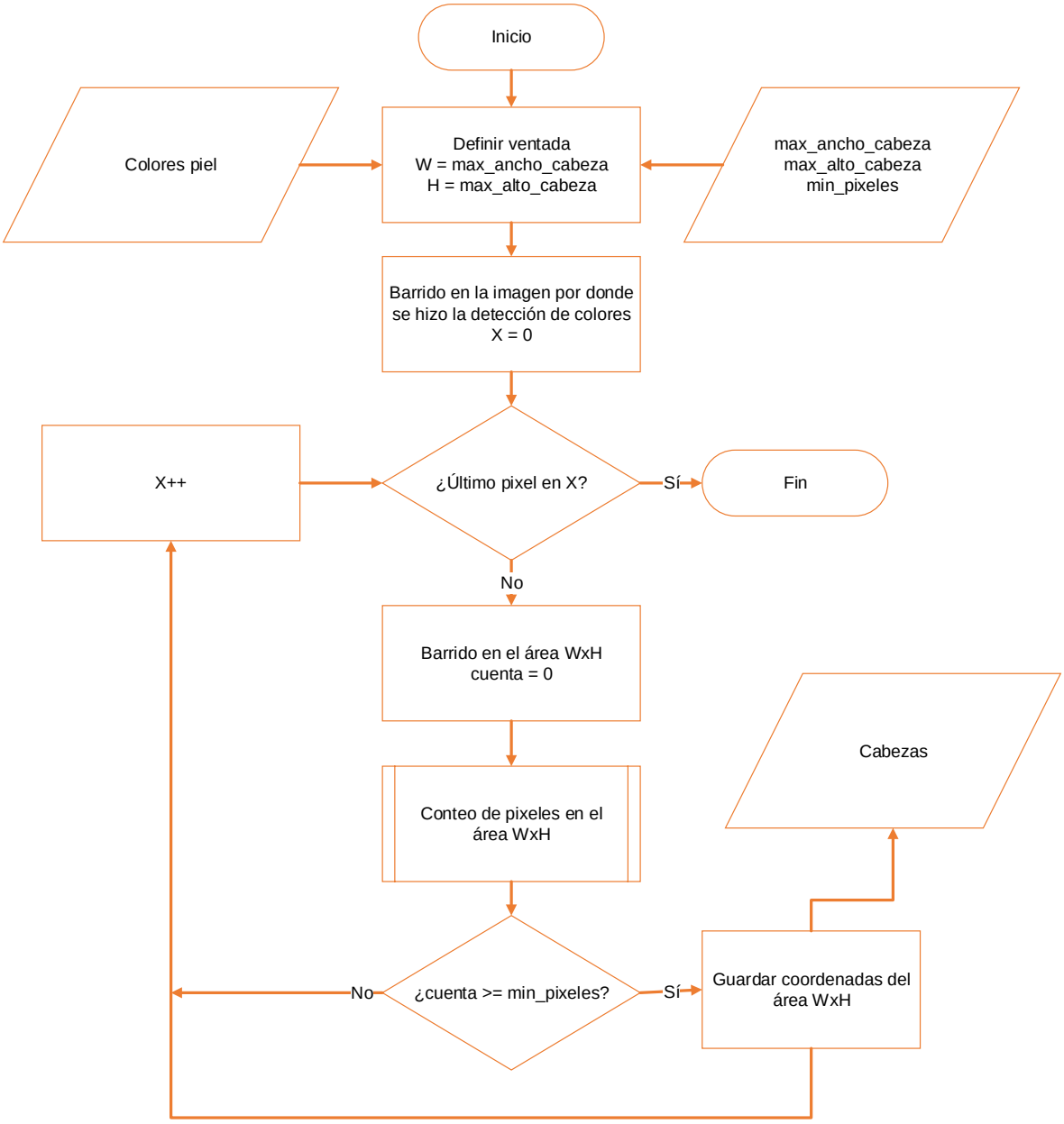


Diagrama 3.8: Detección de cabezas

3.10 Reconocimiento de peatones

Finalmente, al algoritmo trabajará con la información recolectada anteriormente para determinar la presencia de un peatón. Esta información son las piernas, los brazos y las cabezas detectadas.

Para este fin, el algoritmo verificará si existe un conjunto de cabeza, brazos y piernas que se encuentren ubicadas dentro de un intervalo de separación aceptable que concuerde con la fisionomía de un ser humano.

Este intervalo, al igual que las proporciones de una persona usadas anteriormente, es constante dependiendo de la profundidad en la que se esté trabajando y fue definido a partir de las imágenes recolectadas como base de datos.

En otras palabras, si el algoritmo verifica que se detectó una cabeza y está ubicada en medio de dos piernas también detectadas y éstas a su vez separadas 30cm (lo equivalente en pixeles) aproximadamente una de la otra, se podrá concluir que se trata de un ser humano (peatón) ya que concuerda con la fisionomía de uno. Por lo contrario, si el algoritmo verifica que se detectaron dos piernas, pero a 3m de separación una de la otra, y una cabeza a 1m de separación de la segunda pierna, concluirá que no hay presencia de peatón ya que ninguna persona tiene el cuerpo de esa forma.

El la figura 3.18 se muestra un ejemplo de este procedimiento. Al aplicar el algoritmo en la imagen mostrada (imagen superior), se detectaron 3 cabezas marcadas con verde, 4 brazos marcados con azul y 2 piernas marcadas en amarillo. Para efectos de visualización, los recuadros de colores se colocaron manualmente en la imagen inferior, mientras que en la imagen de en medio se juntaron las tres imágenes que arrojan los algoritmos para la detección de piernas, brazos y cabeza. Con esta información el algoritmo determinará la presencia de un peatón a partir de lo explicado anteriormente.

Las cabezas detectadas a los extremos junto con los brazos que se encuentran debajo de ellas coinciden con la fisionomía de una persona (que en realidad no son cabezas ni brazos) y podrían representar un peatón. Sin embargo, el brazo que se encuentra más a la izquierda queda totalmente descartado ya que no podría coincidir con la fisionomía de una persona junto con la cabeza que se encuentra más cercana (están muy separadas).

La cabeza que se encuentra en medio coincide perfectamente con el brazo que se encuentra debajo y las dos piernas más abajo. Por esta razón se concluye que hay un peatón ahí (recuadro anaranjado) y se descartan las dos opciones anteriores ya que con ésta se tiene más certeza debido a las piernas.

En caso de que las otras dos opciones realmente hubieran sido un peatón, el algoritmo le seguirá dando prioridad al que está en medio debido a que es el que corre más peligro de ser atropellado. Además, ya sea que exista un sólo peatón o varios en la escena, el automóvil deberá de frenarse de igual manera para prevenir el accidente, por lo que no será necesario detectar a todos.

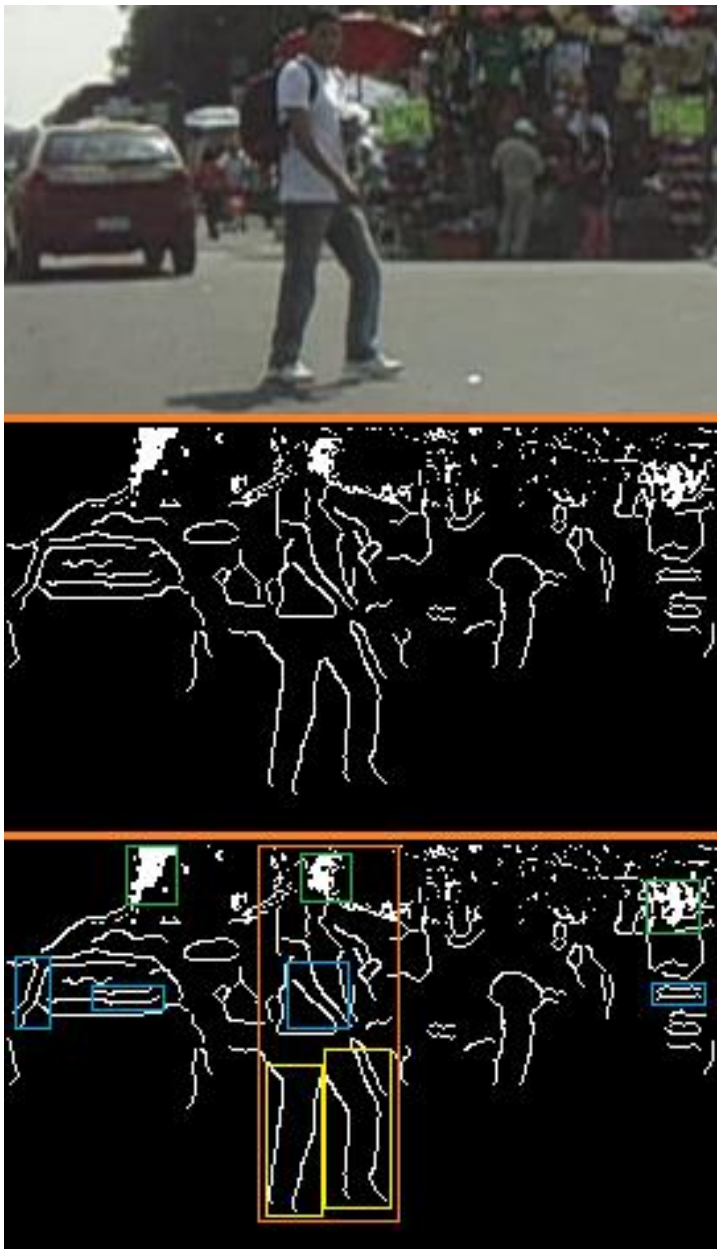


Figura 3.18: Peatón detectado

Debido a que no siempre se logrará hacer la detección de las tres partes del cuerpo (cabeza, brazo y piernas) por problemas de contraste, etc., el algoritmo emplea tres diferentes niveles de certidumbre dependiendo de lo que se haya detectado, los cuales son «peatón», «posible peatón» y «no peatón».

Para este criterio no se considera la presencia de los dos brazos en una persona, sino sólo uno, debido a que mayoritariamente las personas aparecerán de perfil en las imágenes ya que ésta es la visualización que se tiene desde el vehículo hacia la

persona que cruza la calle, por lo que uno de los brazos no será detectable ya que estará siendo tapado por el cuerpo.

En la tabla 3.1 se muestra lo que determina el algoritmo en base a las partes detectadas.

CABEZA	BRAZO	UNA PIERNA	DOS PIERNAS	CONCLUSIÓN
●	●		●	Peatón
●	●	●		Peatón
●			●	Peatón
●		●		Peatón
	●		●	Posible peatón
	●	●		Posible peatón
●	●			No peatón
●				No peatón
	●			No peatón
		●		No peatón
			●	No peatón
				No peatón

Tabla 3.1: Determinación de la presencia de un peatón

Como se puede observar, las piernas aportan la mayor contribución de certidumbre a la hora de determinar si se trata de un peatón o no, seguido de la cabeza y al final los brazos. Esto es debido a que los brazos son altamente confundibles con algún otro objeto o figura, mientras que el color de la piel, en algunos casos, podría coincidir con algún otro color de la escena y ser representado como una cabeza, como sucedió en la figura 3.18. Sin embargo, las piernas al estar situadas abajo hacen un buen contraste con el pavimento o algún otro fondo, la mayoría de las veces, además que muy pocas cosas situadas a esa altura tienen esa forma, por lo que es difícil confundirlas.

La forma en la que se podrían utilizar estos tres niveles de certidumbre sería, en el caso de que el vehículo detecte un peatón, éste frenaría automáticamente, mientras que cuando detecte un posible peatón únicamente mandaría una alerta al conductor para que éste compruebe si en verdad se trata o no de un peatón y de ser así pueda frenar el vehículo manualmente.

En la figura 3.19 no fue posible detectar algún brazo del peatón, sin embargo por su cabeza y sus piernas que fueron correctamente detectadas se concluye que es un peatón, según la tabla 3.1.

Figura 3.19: Peatón detectado a partir de la cabeza y dos piernas

En la figura 3.20 sólo fue posible detectar una pierna, ya que la otra apareció con un mal contraste debido a la sombra del vehículo de atrás. Sin embargo, con sólo detectar su cabeza y una de sus piernas fue suficiente para concluir que se trataba de un peatón.

El brazo que se detectó en esa imagen coincidió estar en una posición que coincidía con la fisionomía del peatón, el cual verdaderamente no es un brazo del peatón. Sin embargo, para el algoritmo no fue necesario analizarlo ya que con una sola pierna y la cabeza detectadas se podía concluir la presencia del peatón.

Figura 3.20: Peatón detectado a partir de la cabeza y una pierna

En la figura 3.21 se muestra una imagen en la cual no aparece ningún peatón. Como se puede observar, no se detecta ninguna cabeza o pierna, pero sí varios brazos ya que como se comentó anteriormente, éstos son los que tienden más a parecerse a otras cosas.

El algoritmo concluye correctamente que no hay ningún peatón.

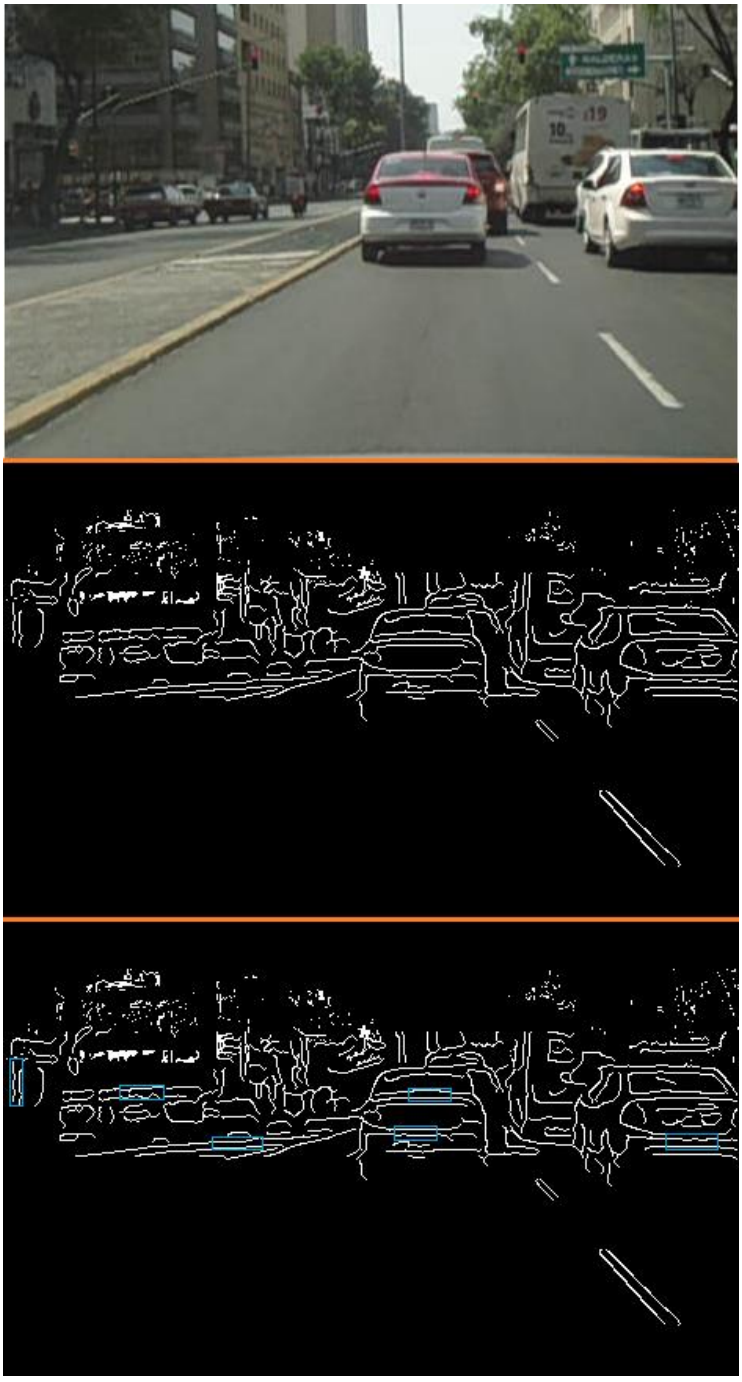


Figura 3.21: No peatón en la escena

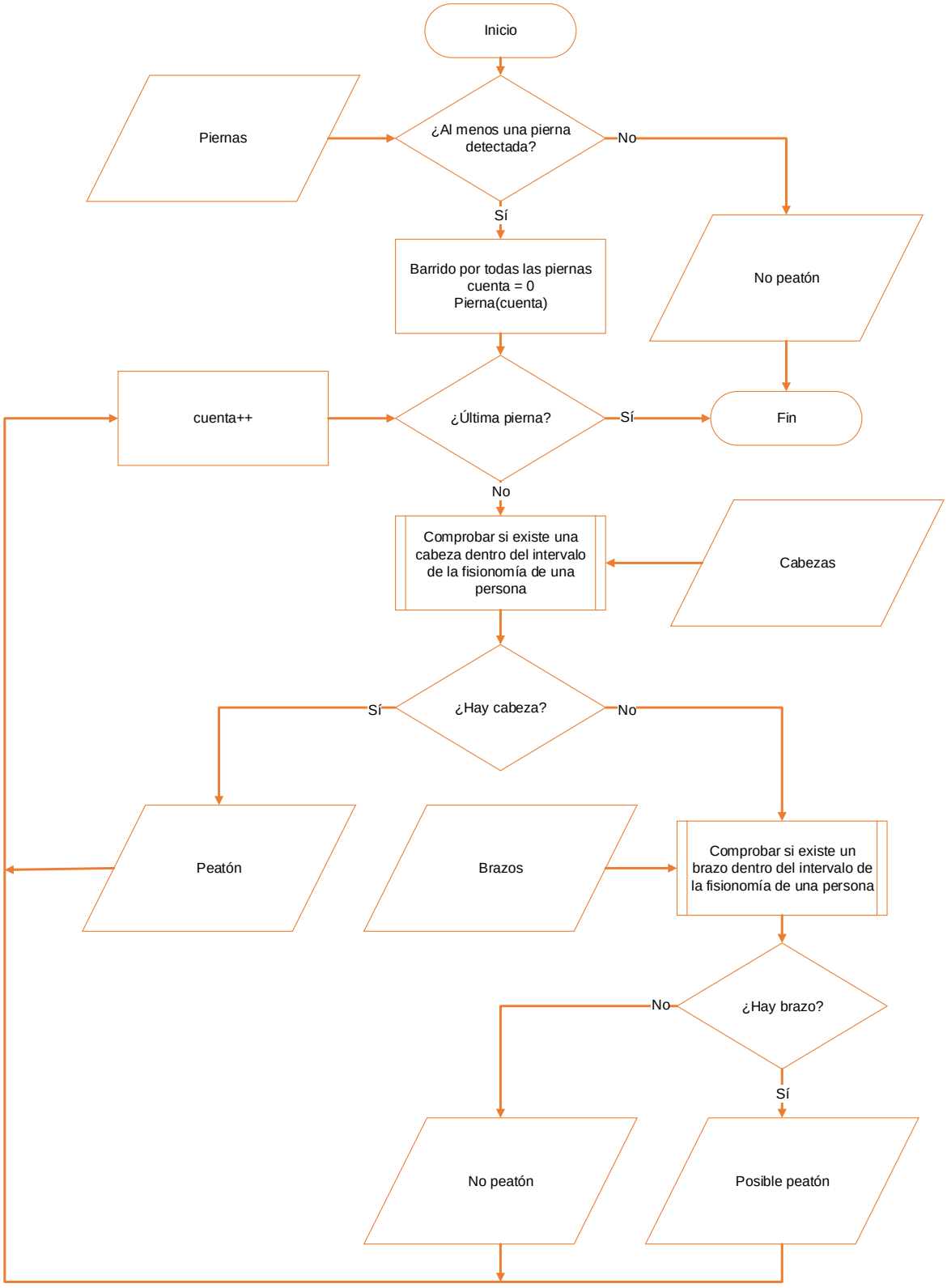


Diagrama 3.9: Detección de peatones

3.11 Implementación en GPU por medio de OpenCL

Como se mencionó anteriormente, este algoritmo diseñado será acelerado mediante una GPU como dispositivo de implementación, y mediante OpenCL como herramienta de programación.

A continuación se hará una breve descripción de cómo es que opera OpenCL y las ventajas que ofrece, después se describirá brevemente el funcionamiento y segmentación de las GPUs en base a OpenCL, y finalmente se describirá cómo fue implementado el algoritmo en la GPU.

3.11.1 Funcionamiento de OpenCL

OpenCL es una herramienta para la programación heterogénea, esto es, programación que involucra diferentes dispositivos como GPUs, FPGAs, DSPs, etc., todos ellos en conjunto destinados a una misma aplicación. Para el caso de este trabajo de tesis, únicamente será una GPU.

OpenCL es una herramienta relativamente sencilla de utilizar, ya que está basada en el lenguaje C, sin embargo, es necesario conocer varios conceptos para poder entender su funcionamiento.

Primeramente, OpenCL necesita de un «host», esto es, un dispositivo que se encargará de organizar y asignar tareas a todos los demás dispositivos involucrados. En la mayoría de los casos, incluyendo este trabajo de tesis, el host será el CPU.

Los dispositivos que llevarán a cabo las tareas, GPUs, FPGAs, etc., son conocidos como «devices», o dispositivos en español.

Para que todos los dispositivos involucrados puedan relacionarse en una misma aplicación bajo las órdenes del mismo host, es necesario definir lo que OpenCL llama «context».

La programación del host se realiza exactamente igual que cualquier programa en C, pero la programación de los dispositivos se realiza en un archivo separado con extensión .cl. La programación en este archivo se realiza mediante el lenguaje propio de OpenCL, el cual como ya se ha mencionado, está basado en C99. OpenCL llama a estos programas como «programs».

En cada uno de estos programas se definirán las funciones que queremos que los dispositivos realicen. Estas funciones son conocidas como «kernels».

En tiempo de ejecución, los archivos .cl, o programs, serán compilados y se crearán los diferentes kernels. Éstos serán enviados por el host hacia los dispositivos para que realicen dicha tarea.

Para lograr esta comunicación directa entre el host y cada uno de los dispositivos, es necesario definir lo que OpenCL llama «command queue», o cola de comandos en español, para cada uno de los dispositivos.

Es necesario disponer y tener instalados los controladores, o drivers, de OpenCL para cada dispositivo para poder realizar la compilación de los programs y poder utilizar los dispositivos.

Este es el funcionamiento básico de OpenCL, el cual se ilustra en la figura 3.22.

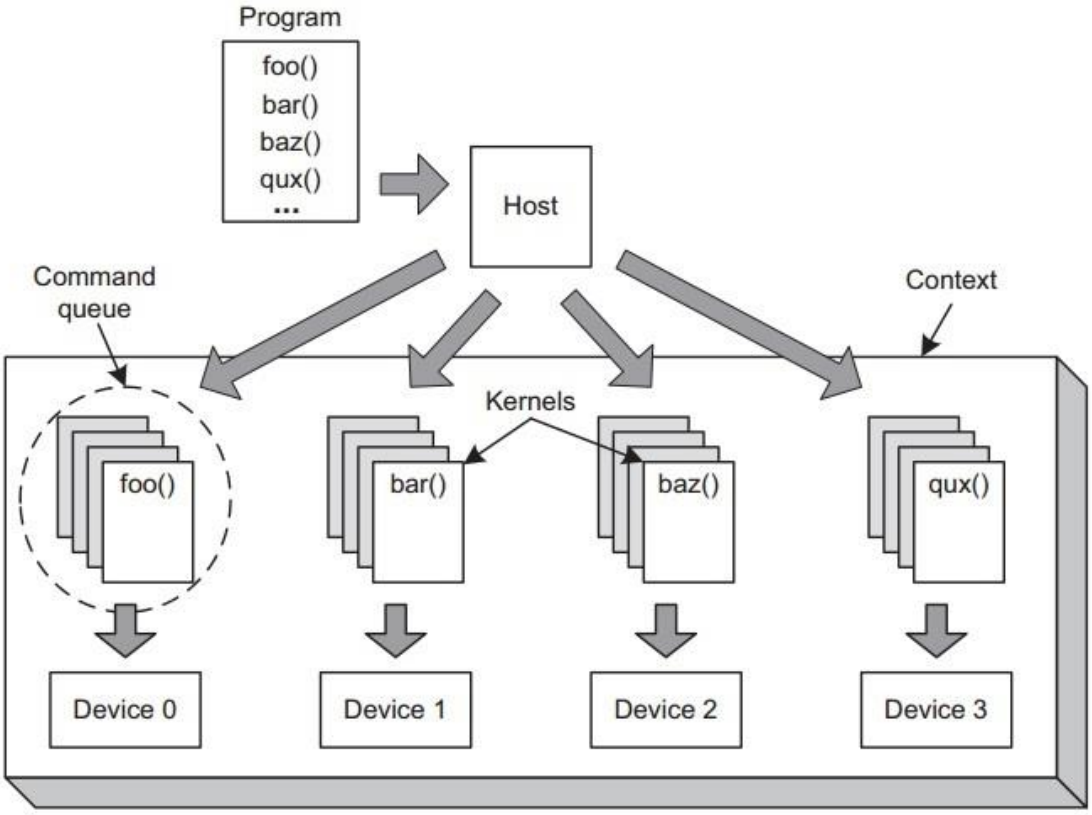


Figura 3.22: Funcionamiento de OpenCL [31]

3.11.2 Segmentación de la GPU

Las GPUs son dispositivos con un alto nivel de paralelismo, por lo que es importante conocer cómo es que OpenCL puede dividir o segmentar estos dispositivos para lograr una eficiente ejecución en paralelo de las tareas.

Los elementos dentro de la GPU que llevarán a cabo las tareas, o kernels, asignados por el host son conocidos como «work-items».

Es posible crear grupos formados por varios work-items cada uno, con el fin de que cada grupo realice una tarea diferente. Estos grupos son conocidos como «work-groups».

El número de work-groups que podemos crear en una GPU es determinado por el número de unidades de cómputo que físicamente tenga cada GPU. Es posible crear un número de work-groups mayor al número de unidades de cómputo, sin embargo estos work-groups no se ejecutarán completamente en paralelo. Se ejecutarán en paralelo la cantidad de work-groups equivalente a las unidades de cómputo, y al terminar su ejecución se ejecutarán los restantes.

Lo mismo aplica para el número de work-items que podremos crear. Cada GPU soporta cierto número de work-items por work-group, y en caso de definir más, éstos no serán ejecutados completamente en paralelo.

OpenCL dispone de tres tipos de memoria en la GPU. La primera es la memoria global, o «global memory» la cual es la de mayor capacidad de almacenamiento de las tres, sin embargo también es la más lenta a la hora de utilizarla. Cualquier work-item definido, independientemente del work-group al que pertenezca, puede utilizar esta memoria. Esta memoria es la única de las tres que puede ser utilizada por el host, por lo tanto esta memoria es el único medio por donde se pueden transferir datos desde el host al GPU y viceversa.

El siguiente tipo de memoria es la memoria local, o «local memory». Esta memoria tiene una capacidad de almacenamiento media al igual que una velocidad de lectura y escritura media. Esta memoria únicamente puede ser utilizada por los work-items dentro de un mismo work-group, por lo que únicamente podremos disponer de una cantidad de memorias locales diferentes equivalente al número de unidades de cómputo de la GPU.

El último tipo de memoria es la memoria privada, o «private memory». Como se podrá suponer, esta memoria es la más rápida de las tres, hasta 100 veces más rápida que la memoria global, sin embargo es la más pequeña en cuanto almacenamiento. Cada work-item definido cuenta con su propia memoria privada y únicamente dicho work-item podrá acceder a ella.

OpenCL dispone de funciones con las cuales podremos conocer el número de unidades de cómputo, los work-items máximos por work-group, así como las capacidades y velocidades de cada tipo de memoria, dependiendo de la GPU (o algún otro dispositivo) que estemos utilizando.

Todo lo mencionado anteriormente sobre la segmentación de la GPU se ilustra en la figura 3.23.

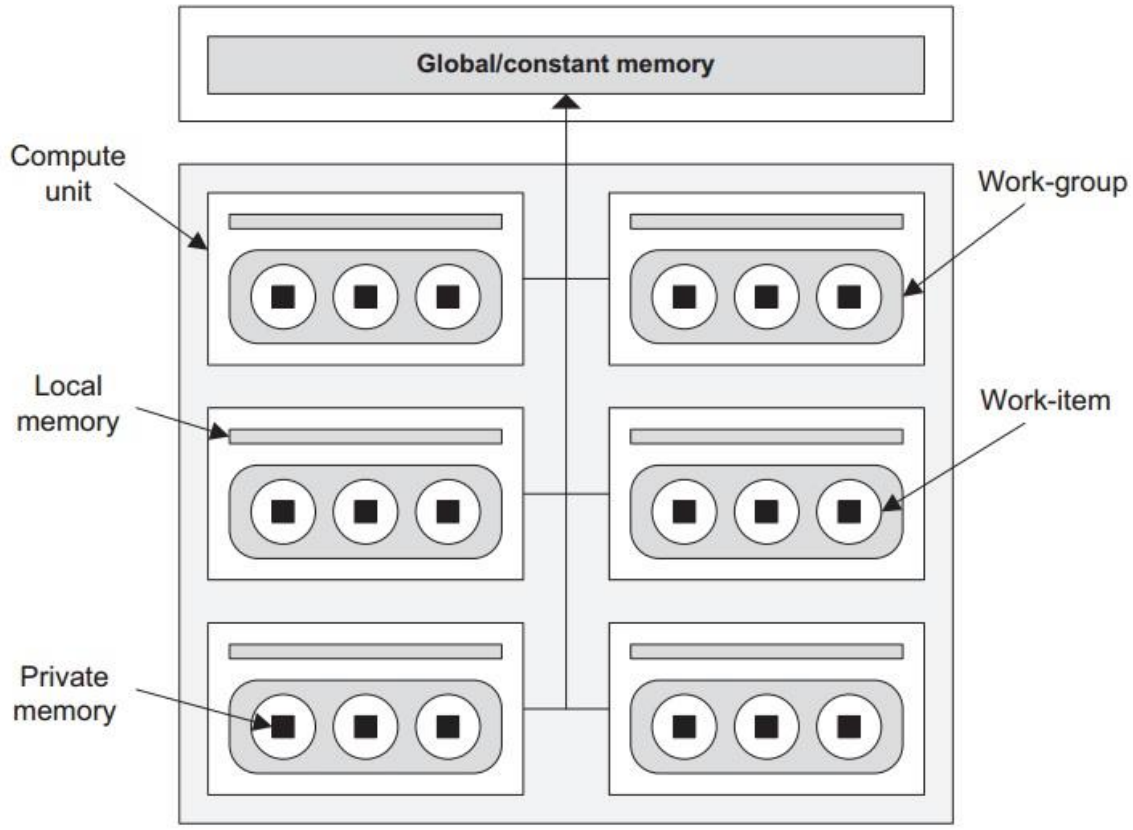


Figura 3.23: Segmentación de la GPU [31]

3.11.3 Paralelización del algoritmo

Existen múltiples maneras de paralelizar el algoritmo ya antes descrito. Sin embargo, debido a que todas las GPUs son de diferentes características, se optó por un enfoque generalizado, el cual consiste en que cada unidad de procesamiento disponible (work-item) analizará sólo un pixel de la imagen a la vez. Esto quiere decir que si analizamos una imagen VGA, 640x480 pixeles, para obtener los segmentos verticales, por ejemplo, se necesitarán 307,200 work-items.

En caso de que la GPU físicamente no cuente con ese número de work-items, OpenCL se encargará de ejecutar el mayor número posible en paralelo, y al terminar ejecutará el resto. Esta es una de las principales ventajas de OpenCL, ya que nos permite escribir programas generalizados sin tener que preocuparnos de los recursos con los que cuenta tal GPU, ya que eso lo manejará OpenCL automáticamente.

Primeramente, el host se encargará de ejecutar el algoritmo Canny para la detección de bordes, por medio de una biblioteca y enviará la imagen de bordes resultante a la GPU. Esto se mencionará más a detalle en el capítulo 4.2.

Una vez con la imagen de bordes, ya en la GPU, se ejecutará el algoritmo para el marcaje de colores para la detección de las cabezas, utilizando el enfoque mencionado anteriormente. Cada work-item se encargará de verificar si su pixel correspondiente está dentro del rango de colores piel. Este algoritmo arrojará una imagen como la de la figura 3.16 para su posterior análisis.

Al finalizar este algoritmo, se ejecutará la detección de cabezas utilizando la imagen que generó el algoritmo anterior. En esta parte se formarán work-groups del tamaño de una cabeza y se hará un barrido a lo largo de la imagen para que cada work-item vaya contando los colores piel y concluir si existe una cabeza o no.

Una vez concluido este algoritmo, se tendrán almacenadas las coordenadas de las cabezas detectadas.

Después, se ejecutará el algoritmo para la detección de los segmentos verticales, y al finalizar, el de los segmentos horizontales. Estos algoritmos obtendrán imágenes como las mostradas en las figuras 3.9 y 3.10.

Al terminar, con las imágenes obtenidas se ejecutará el algoritmo para la detección de piernas y después para la detección de brazos. Una vez finalizados, se tendrán almacenadas las coordenadas de las piernas y brazos detectados.

Finalmente se ejecutará el algoritmo para la detección de peatones. En este caso cada work-item no analizará un pixel, si no alguna parte del cuerpo detectada y verificará si otro work-item con alguna otra parte coincide con sus coordenadas.

Como se mencionó anteriormente, la imagen a profundidad de prioridad alta será la primera en analizarse, así que al terminar ésta, se repetirá el algoritmo para la de prioridad media y después para la de prioridad baja. Una vez finalizada esta última, concluirá el algoritmo y se volverá a ejecutar para el siguiente «frame».

3.12 Imágenes utilizadas y cálculo de proporciones

Para el desarrollo y prueba de este trabajo de tesis, fue necesario definir una base de imágenes con la cual trabajar. Para ello, se grabó una secuencia de video de 20 minutos en resolución VGA (640x480 pixeles) a una velocidad de 30 FPS, en un ambiente no controlado de constante paso peatonal en la zona centro de la Cuidad de México, a plena luz del día, sin lluvia y sin viento excesivo.

La secuencia de video fue fragmentada en sus respectivos cuadros, respetando la velocidad de 30 FPS, mediante el software FFmpeg [32], obteniendo un total de 36,000 imágenes.

De todas estas imágenes, fueron seleccionadas 30 en las cuales aparece claramente uno o más peatones atravesándose por el camino donde pasará el

vehículo por cada una de las tres profundidades a trabajar, con lo que se obtiene un total de 90 imágenes como base de trabajo.

Analizando cada una de estas imágenes, se observaron las zonas en donde sería posible detectar un peatón y se ajustaron los tamaños de las imágenes a cada profundidad para que el algoritmo no perdiera tiempo analizando zonas donde no sería posible detectar un peatón, como por ejemplo al nivel de los techos de las casas y más arriba.

En la tabla 3.2 se muestran los tamaños en pixeles a los que fueron reajustadas las imágenes dependiendo de su profundidad.

Prioridad de la profundidad	Tamaño (pixeles)
Alta	625x285
Media	270x155
Baja	130x80

Tabla 3.2: Tamaño de imagen de acuerdo a la profundidad del peatón

En cada una de las 30 imágenes para cada profundidad, se midió el alto y ancho de los peatones presentes, el largo y ancho de sus piernas y brazos, y se registró el color de su cara para después obtener el promedio de todos estos valores.

En la tabla 3.3 se muestran las proporciones o constantes que utiliza el algoritmo para cada profundidad, medida en pixeles.

Prioridad de la profundidad	Alto del peatón	Ancho del peatón	Largo pierna	Ancho pierna	Largo brazo	Ancho brazo	Color de la cara (R,G,B)
Alta	145	50	40	24	15	9	(110,85,74)
Media	110	40	30	10	10	6	(90,72,67)
Baja	45	10	8	5	6	4	(76,54,53)

Tabla 3.3: Proporciones del peatón a diferentes profundidades (pixeles)

Debido a que estas proporciones no serán exactas para todos los peatones ya que las personas no son exactamente iguales entre sí, el algoritmo empleará una tolerancia de ±20% para cada valor.

Finalmente, como procedimiento de caracterización, se aplicó el algoritmo terminado a las 90 imágenes de prueba y se calculó el mínimo número de características (pierna, brazo o cabeza) y cuáles necesita detectar el algoritmo para determinar correctamente si existe o no un peatón en la escena, lo cual se especifica en la tabla 3.1.

CAPÍTULO 4

RESULTADOS

A lo largo del desarrollo de este algoritmo para la detección de peatones se utilizaron diferentes herramientas de desarrollo para facilitar las tareas que se fueran requiriendo. A continuación se describe el uso de estas herramientas y los resultados obtenidos con cada una de ellas.

4.1 Implementación en MATLAB (CPU)

MATLAB [33] es una herramienta de software matemático, la cual cuenta con un lenguaje de programación propio (lenguaje M), que está orientada a cálculos matemáticos, graficación en 2D y 3D, simulación, procesamiento de imágenes, etc.

MATLAB fue el primer entorno de desarrollo que se utilizó para el desarrollo de este algoritmo debido a que es una herramienta de programación que proporciona múltiples funciones y bibliotecas predefinidas de fácil acceso, como para el manejo y procesamiento de imágenes, lo que facilita mucho el diseño de programas prototipo.

Con la función «imread» se puede acceder a cualquier imagen guardada en el ordenador y extraer sus pixeles en una variable para su posterior procesamiento. La función «edge» aplica el algoritmo Canny (entre otros) para obtener los bordes de una imagen sin que el usuario tenga que programar algún cálculo. Con la función «imwrite» se guarda la imagen procesada para poder visualizar el resultado. Para medir el tiempo que tarda en ejecutarse un programa, MATLAB brinda las funciones «tic» y «toc».

Estas funciones fueron de gran ayuda para realizar las primeras pruebas con este algoritmo para la detección de peatones.

Los resultados obtenidos del algoritmo implementado en MATLAB se muestran en las tablas 4.1, 4.2, 4.3 y 4.4.

4.2 Implementación en C (CPU)

MATLAB es una herramienta que facilita mucho el trabajo de programación, sobre todo cuando se trabaja con imágenes. Sin embargo, al ser un entorno que cuenta con una vasta cantidad de funciones y bibliotecas predefinidas, su tiempo de ejecución es más prolongado que con un lenguaje de programación puro.

Por esta razón, el siguiente paso fue implementar el algoritmo en lenguaje C para reducir el tiempo de ejecución.

En lenguaje C no existen funciones propias para acceder a los pixeles de una imagen almacenada en el ordenador, o para obtener los bordes de una imagen por medio del algoritmo Canny. Estas funciones deben ser diseñadas manualmente.

GraphicsMagick [34] es una biblioteca de código abierto escrita en C para el procesamiento de imágenes. Cuenta con muchas funciones, entre las que están la lectura y escritura de imágenes del ordenador. Con esta biblioteca se puede acceder a los pixeles de una imagen fácil y eficientemente evitando así el complejo diseño de programas para la descompresión de los diferentes formatos de imágenes.

OpenVX [35] es una biblioteca potente de código abierto para aplicaciones de visión por computadora desarrollada por el Khronos Group, los mismos desarrolladores de OpenCL. Esta biblioteca cuenta con una función, «vxuCannyEdgeDetector», para obtener los bordes de una imagen por medio del algoritmo Canny, de la misma forma que lo hace MATLAB.

Gracias a las bibliotecas GraphicsMagick y OpenVX se puede obtener la misma flexibilidad y simplicidad que con MATLAB, pero con un tiempo de ejecución propio del lenguaje C.

Con la ayuda de estas bibliotecas se desarrolló el algoritmo para la detección de peatones en lenguaje C.

El entorno de desarrollo (IDE) que se utilizó en este trabajo para la programación en C y en OpenCL fue Visual Studio [36].

Los resultados obtenidos del algoritmo implementado en lenguaje C se muestran en las tablas 4.1, 4.2, 4.3 y 4.4.

4.3 Implementación en OpenCL (CPU + GPU)

La última implementación de este algoritmo a lo largo de este trabajo de tesis fue en una GPU por medio de OpenCL para aprovechar el paralelismo que brindan estos dispositivos.

Como ya se había mencionado antes, OpenCL es un entorno de desarrollo que consta de una API y un lenguaje de programación que sirven para diseñar aplicaciones con diferentes niveles de paralelismo en dispositivos como GPUs, DSPs, FPGAs, coprocesadores, etc.

Al igual que la implementación en C, en esta última también se ocuparon las bibliotecas GraphicsMagick para el manejo de las imágenes y OpenVX para obtener los bordes de la imagen por medio del algoritmo Canny.

Esta última implementación se explicó a detalle en el capítulo 3.11, y los resultados obtenidos se muestran en las tablas 4.1, 4.2, 4.3 y 4.4.

4.4 Comparación y análisis de resultados

En el capítulo 3.3 se mencionó que es necesario establecer diferentes profundidades en una imagen para que el algoritmo opere en cada una de ellas con diferentes valores de las proporciones de una persona dependiendo si se encuentran cerca o lejos del vehículo.

En este trabajo se establecieron tres profundidades diferentes las cuales tienen una prioridad alta, media y baja. El algoritmo analizará primeramente la de prioridad alta, al terminar seguirá con la de prioridad media y por último con la de baja prioridad para así terminar con su ejecución y volver repetirse en el siguiente cuadro.

En cada una de las diferentes implementaciones que se hicieron a lo largo del desarrollo de este algoritmo (MATLAB, C y OpenCL) se probaron diferentes imágenes para medir los tiempos de ejecución de cada una y se obtuvo un promedio de estos tiempos para hacer una comparación.

En el desarrollo de este trabajo se disponía de dos computadoras de escritorio. La primera con un procesador Intel® Core i7 4770K a 3.5GHz, 8Gb de memoria RAM DDR3 a 1600MHz, con una GPU NVIDIA® GeForce GTX 750 a 1.2GHz, 1Gb de memoria RAM GDDR5 y 4096 procesos simultáneos. La segunda con un procesador Intel® Core i7 3820 a 3.6GHz, 8Gb de memoria RAM DDR3 a 1600MHz, con una GPU AMD® Radeon HD 7970 a 925MHz, 3Gb de memoria RAM GDDR5 y 8192 procesos simultáneos.

Los resultados de los tiempos de ejecución del algoritmo al analizar únicamente la profundidad de prioridad baja se muestran en la tabla 4.1, las de prioridad media en la tabla 4.2, las de alta prioridad en la tabla 4.3 y el tiempo total de ejecución del algoritmo, que sería la suma de las tres profundidades, en la tabla 4.4.

IDE	Lenguaje	Procesador CPU	GPU	Tiempo (s)
MATLAB	MATLAB	Intel® Core i7 4770K	N/A	0.56
Visual Studio	C	Intel® Core i7 4770K	N/A	0.054
Visual Studio	OpenCL	Intel® Core i7 4770K	Intel® HD Graphics 4600	0.0033
Visual Studio	OpenCL	Intel® Core i7 4770K	NVIDIA® GeForce GTX 750	0.00057
Visual Studio	OpenCL	Intel® Core i7 3820	AMD® Radeon HD 7970	0.00076

Tabla 4.1: Resultados en profundidad de prioridad baja, 130x80 pixeles

IDE	Lenguaje	Procesador CPU	GPU	Tiempo (s)
MATLAB	MATLAB	Intel® Core i7 4770K	N/A	1.2
Visual Studio	C	Intel® Core i7 4770K	N/A	0.19
Visual Studio	OpenCL	Intel® Core i7 4770K	Intel® HD Graphics 4600	0.0084
Visual Studio	OpenCL	Intel® Core i7 4770K	NVIDIA® GeForce GTX 750	0.0016
Visual Studio	OpenCL	Intel® Core i7 3820	AMD® Radeon HD 7970	0.00087

Tabla 4.2: Resultados en profundidad de prioridad media, 270x155 pixeles

IDE	Lenguaje	Procesador CPU	GPU	Tiempo (s)
MATLAB	MATLAB	Intel® Core i7 4770K	N/A	4.26
Visual Studio	C	Intel® Core i7 4770K	N/A	0.83
Visual Studio	OpenCL	Intel® Core i7 4770K	Intel® HD Graphics 4600	0.028
Visual Studio	OpenCL	Intel® Core i7 4770K	NVIDIA® GeForce GTX 750	0.0047
Visual Studio	OpenCL	Intel® Core i7 3820	AMD® Radeon HD 7970	0.0012

Tabla 4.3: Resultados en profundidad de prioridad alta, 625x285 pixeles

IDE	Lenguaje	Procesador CPU	GPU	Tiempo (s)	FPS
MATLAB	MATLAB	Intel® Core i7 4770K	N/A	6.02	0.17
Visual Studio	C	Intel® Core i7 4770K	N/A	1.074	0.93
Visual Studio	OpenCL	Intel® Core i7 4770K	Intel® HD Graphics 4600	0.0397	25.19
Visual Studio	OpenCL	Intel® Core i7 4770K	NVIDIA® GeForce GTX 750	0.00687	145.56
Visual Studio	OpenCL	Intel® Core i7 3820	AMD® Radeon HD 7970	0.00283	353.36

Tabla 4.4: Resultados totales de la ejecución del algoritmo

En la tabla 4.5 se muestran el promedio de los tiempos parciales de ejecución de cada uno de los subalgoritmos que utiliza el algoritmo para la detección de peatones los cuales son detección de posibles brazos, detección de posibles piernas, detección de posibles cabezas y finalmente detección del peatón, utilizando únicamente la implementación de OpenCL con la GPU AMD® Radeon HD 7970.

Además, en esta tabla también se muestra el porcentaje del tiempo total de ejecución del algoritmo que emplea cada uno de estos subalgoritmos, con lo que se puede observar que la detección de cabezas es el algoritmo que más tiempo necesita para ejecutarse de todos.

Algoritmo	Tiempo (s)	Porcentaje
Brazos	0.000184	6.51%
Piernas	0.000340	12.03%
Cabezas	0.002291	81.04%
Peatones	0.000012	0.42%

Tabla 4.5: Tiempos parciales por algoritmo

De las 90 imágenes de la base, se seleccionaron 30 para probar la eficacia del algoritmo.

En la tabla 4.6 se muestran las detecciones positivas que tuvo el algoritmo en las 30 imágenes analizadas, esto es, las veces que el algoritmo concluyó que había un peatón en la escena y verdaderamente lo había. También se muestran los falsos negativos los cuales indican la cantidad de veces que el algoritmo concluyó que no existía peatón en la escena y en realidad sí lo había, así como los falsos positivos que son las veces que el algoritmo detectó un peatón cuando en realidad no lo había.

IMÁGENES ANALIZADAS	POSITIVOS	FALSOS NEGATIVOS	FALSOS POSITIVOS
30	26	4	0

Tabla 4.6: Eficacia del algoritmo

En resumen, los resultados obtenidos son en promedio 145 FPS con la GPU NVIDIA® y 350 FPS con la de AMD®.

Se puede observar que en algunas ocasiones el algoritmo no detecta al peatón cuando en realidad lo hay. Esta situación es muy común en los algoritmos actuales para la detección de peatones [9].

Por otro lado, el algoritmo con las imágenes que se utilizaron, no cometió el error de detectar un peatón cuando en realidad no lo hay. Algo bastante favorable ya que de lo contrario se estarían realizando frenadas innecesarias en el vehículo que se implementara.

Los 26 positivos obtenidos de las 30 imágenes representan un 87% de eficacia del algoritmo, con un 13% de error.

Los resultados obtenidos en [26] sirven como buen punto de comparación ya que en dicha investigación también se trabajó con imágenes VGA, 640x480 pixeles.

Obtuvieron un máximo de 135 FPS utilizando una GPU NVIDIA® con un nivel de paralelismo y velocidad que se colocaría en medio de las dos utilizadas en esta investigación.

CAPÍTULO 5

CONCLUSIONES

Después del diseño e implementación de este algoritmo para la detección de peatones, se han obtenido resultados con alto rendimiento por lo que a continuación se enlistan las conclusiones a las que se ha llegado.

Cabe señalar que este algoritmo todavía no está del todo preparado para ser implementado en un automóvil en tiempo real ya que para dicho fin es necesario cubrir muchas más condiciones para poder operar de forma segura, como son condiciones de luz, clima, entorno, variantes en los peatones, etc.; condiciones que no están contempladas en este trabajo. Para esta etapa de desarrollo, el algoritmo presenta ciertas limitaciones que se enlistan a continuación.

Para eliminar estas limitaciones, mejorar los resultados obtenidos en este trabajo y lograr que este algoritmo sea robusto y confiable en la mayoría de las situaciones, se presenta también a continuación una lista de los trabajos a futuro que se pudieran realizar subsecuentes a este trabajo de tesis.

5.1 Conclusiones

- Se obtuvieron resultados muy consistentes. El tiempo de ejecución del algoritmo es de 5 a 10 más rápido al ejecutarse puramente en lenguaje C que en MATLAB, algo que era de esperarse debido a que MATLAB es un software interpretado que retardan la ejecución de los programas. Los tiempos obtenidos con las GPUs fueron mejorando acorde con sus características. La GPU HD Graphics 4600 es una GPU embebida junto con el procesador de Intel® por lo que ofrece una limitada velocidad y paralelismo. La GeForce GTX 750 de NVIDIA® es una GPU dedicada de pequeñas prestaciones, por lo que su velocidad y paralelismo son moderadas aunque mejor que la HD Graphics. Por último, la Radeon HD 7970 de AMD® es una potente GPU de gama alta que ofrece una buena

velocidad y un alto nivel de paralelismo, por lo que se esperaba que con esta GPU se iban a obtener los mejores resultados.

- La implementación de una GPU podría acelerar el tiempo de procesamiento de un algoritmo hasta 100 veces comparado con implementar el algoritmo únicamente con un CPU.
- Se obtuvieron mejores resultados en cuanto a FPS que los obtenidos en [25] y [26], incluso con una GPU de menores prestaciones.
- Este algoritmo se beneficia más del nivel de paralelismo que ofrece una GPU (cantidad de procesos simultáneos) que de su velocidad.
- El tiempo de ejecución prácticamente no aumenta al aumentar el nivel de definición del algoritmo Canny para extraer los bordes de la imagen ya que el algoritmo para la detección de peatones está programado para que cada unidad de procesamiento (work-item) de la GPU procese un solo pixel, por lo que las imágenes de la figura 5.1 tardarían prácticamente lo mismo en ser procesadas.

Figura 5.1: Diferentes definiciones en bordes de la misma imagen

- OpenCL brinda una enorme ayuda en la paralelización de algoritmos y multiplataformas.
- En este trabajo se pudo probar el algoritmo en GPUs de diferentes fabricantes como Intel®, NVIDIA® y AMD® gracias a OpenCL, algo que no hubiera sido posible con CUDA.

5.2 Limitaciones

- Mientras más alejada se encuentra la persona del vehículo resulta más complicada la detección de sus brazos.
- Al aplicar la detección de bordes, si el segmento de una pierna o un brazo se detecta discontinuo en alguna parte, como se muestra en la figura 5.2, se considerarán como dos segmentos diferentes y no se detectará dicha pierna o brazo debido a que no cumpliría con el mínimo de largo del segmento. Esta es la principal causa de la detección de los falsos negativos.

Figura 5.2: Segmento discontinuo

- Condiciones de poca luz o poco contraste aumentan proporcionalmente la limitación anterior como se muestra en la figura 5.3.

Figura 5.3: Poco contraste dificulta la detección de las piernas

- Si la persona gira la cabeza de tal forma que no se pueda apreciar la piel de su rostro, no se podrá hacer la detección de la cabeza.

5.3 Trabajos a futuro

- Implementar un algoritmo para la interpolación de los segmentos discontinuos, con esto se eliminaría la principal limitante de este algoritmo.

- Aumentar la resolución de las imágenes o de la cámara con la que se hará la captura para que las imágenes más alejadas tengan mejor definición. Incluso trabajar con dos cámaras, una de resolución VGA enfocada únicamente en la profundidad más cercana, y otra de mayor resolución enfocada a las profundidades más alejadas.
- Este algoritmo es altamente dependiente de algún algoritmo para la detección de bordes. En este trabajo se utilizó el algoritmo Canny, sin embargo éste no es el mejor. Probar con un mejor algoritmo para la detección de bordes podría llevar a mejores resultados.
- Trabajar con más de tres profundidades para abarcar un mayor rango de proporciones de personas.
- Utilizar una gama más amplia de colores piel para cubrir un mayor número de personas, ya que en este trabajo sólo se consideró la tez morena clara por ser la más abundante en México.
- Además de detectar la piel del rostro de las personas, también se podría intentar detectar el cabello para los casos en que la persona aparezca con la cabeza girada.
- Aplicar el mismo algoritmo basado en colores que se utilizó para la detección de las cabezas como soporte en la detección de los brazos, ya que muchas personas suelen vestir playeras de mangas cortas lo que facilitaría la detección de la piel de sus brazos como se muestra en la figura 3.15 y 3.19.
- En este trabajo se utilizó la versión 1.2 de OpenCL debido a que sólo se disponían de GPUs compatibles con esta versión. La versión 2.0 de OpenCL ofrece importantes mejoras por lo que implementar este algoritmo con dicha versión podría llevar a mejores resultados [37].
- Actualmente existen GPUs que ofrecen un mayor nivel de paralelismo, mayor velocidad y son de menor tamaño que las utilizadas en este trabajo. Con estas GPUs se pueden obtener mejores resultados y se facilitaría la implementación en un vehículo en tiempo real [38].
- Como se dijo anteriormente, OpenCL es compatible con múltiples dispositivos, incluyendo los procesadores que hoy en día se utilizan en los teléfonos inteligentes. Implementar este algoritmo en un teléfono inteligente facilitaría su implementación en un vehículo en tiempo real. Cualquier persona con un teléfono inteligente podría descargar la aplicación y disponer de un detector de peatones [39].

REFERENCIAS

[1] World Health Organization, «World health statistics 2008,» 2008.

[2] J. C. Russ, The Image Processing Handbook, CRC Press, 2011.

[3] I. Fallon y D. O'Neill, «The world's first automobile fatality,» *Accident analysis and prevention,* 2005.

[4] World Health Organization, «Speed managment,» 2008.

[5] C. J. Chen, H. Y. Peng, B. F. Wu y Y. H. Chen, «A real-time driving assistance and surveillance system,» *Journal of information science and engineering,* vol. 25, nº 5, 2009.

[6] W. S. Wijesoma, S. Kodagoda y A. P. Balasuriya, «Road-boundary detection and tracking using ladar sensing,» *IEEE transactions on robotics and automation,* vol. 20, nº 3, 2004.

[7] A. Doi, T. Butsuen, T. Niibe, T. Takagi, Y. Yamamoto y H. Seni, «Development of a rear-end collision avoidance system with automatic brake control,» *JSAE Review,* vol. 15, 1994.

[8] Volvo Car Corporation, «S80 owner's manual,» 2015.

[9] D. Gerónimo y A. M. López, Vision-based pedestrian protection systems for intelligent vehicles, Springer, 2013.

[10] Land Rover, «Discovery Sport 2015 specifications,» 2015. [En línea]. Available: http://www.landrover.com/vehicles/new-discovery-sport/specifications.html.

[11] Google, «Google Self-Driving Car Project,» [En línea]. Available: http://www.google.com/selfdrivingcar/.

[12] Google, «Google Self-Driving Car Project Navigating city streets,» [En línea]. Available: http://www.google.com/selfdrivingcar/how/.

[13] E. D. Dickmanns y A. Zapp, «A curvature-based scheme for improving road vehicle guidance by computer vision,» *SPIE.,* vol. 0727, 1987.

[14] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna y T. Poggio, «Pedestrian detection using wavelet templates,» *Computer vision and pattern recognition,* 1997.

[15] N. Dalal y B. Triggs, «Histograms of oriented gradients for human detection,» *Computer vision and pattern recognition,* vol. 1, 2005.

[16] M. Hemmati, M. Biglari-Abhari, S. Berber y S. Niar, «HOG feature extractor hardware accelerator for real-time pedestrian detection,» *Digital system design,* 2014.

[17] A. Mohan, C. Papageorgiou y T. Poggio, «Example-based object detection on images by components,» *IEEE transactions on pattern analysis and machine intelligence,* vol. 23, nº 4, 2001.

[18] A. Shashua, Y. Gdalyahu y G. Hayun, «Pedestrian detection for driving assistance systems: single-frame classification and system level performance,» *IEEE intelligent vehicles symposium,* 2004.

[19] AMD, «Tarjetas de gráficos AMD Radeon™ Serie R9,» [En línea]. Available: http://www.amd.com/es-xl/products/graphics/desktop/R9.

[20] Intel, «ARK Intel® Core™ i7-4770K Processor,» [En línea]. Available: http://ark.intel.com/es/products/75123/Intel-Core-i7-4770K-Processor-8M-Cache-up-to-3_90-GHz.

[21] NVIDIA, «QUÉ ES EL CÁLCULO ACELERADO EN LA GPU,» [En línea]. Available: http://www.nvidia.es/object/gpu-computing-es.html.

[22] D. Rohr, «The L-CSC cluster: An AMD-GPU-based cost- and power-efficient multi-GPU system for Lattice-QCD calculations at GSI,» 2014.

[23] Khronos Group, «The open standard for parallel programming of heterogeneous systems,» [En línea]. Available: https://www.khronos.org/opencl/.

[24] Google, «Google self-driving car project monthly report,» 2015.

[25] R. Sun, X. Wang y X. Ye, «Real-time pedestrian detection using OpenCL,» *International conference on audio, language and image processing (ICALIP),* 2014.

[26] R. Benenson, M. Mathias, R. Timofte y L. Van Gool, «Pedestrian detection at 100 frames per second,» *Computer vision and pattern recognition,* 2012.

[27] S. Bauer, S. Kohler, K. Doll y U. Brunsmann, «FPGA-GPU architecture for kernel SVM pedestrian detection,» *IEEE computer society conference on computer vision and pattern recognition workshops (CVPRW),* 2010.

[28] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann y K. Doll, «FPGA-based real-time pedestrian detection on high-resolution images,» *IEEE conference on computer vision and pattern recognition workshops (CVPRW),* 2013.

[29] NVIDIA, «TESLA GPU ACCELERATORS FOR SERVERS,» 2015. [En línea]. Available: http://www.nvidia.com/object/tesla-servers.html.

[30] J. Canny, «A computational approach to edge detection,» *Pattern analysis and machine intelligence,* 1986.

[31] M. Scarpino, OpenCL in Action: How to Accelerate Graphics and Computations, Manning, 2011.

[32] «FFmpeg,» [En línea]. Available: https://www.ffmpeg.org/.

[33] MathWorks, «MATLAB The Language of Technical Computing,» [En línea]. Available: http://www.mathworks.com/products/matlab/.

[34] GraphicsMagick, «GraphicsMagick Image Processing System,» [En línea]. Available: http://www.graphicsmagick.org/.

[35] Khronos Group, «OpenVX Portable, Power-efficient Vision Processing,» [En línea]. Available: https://www.khronos.org/openvx/.

[36] Microsoft, «Visual Studio,» [En línea]. Available: https://www.visualstudio.com/.

[37] D. Kaeli, P. Mistry y D. Schaa, Heterogeneous Computing with OpenCL 2.0, Morgan Kaufmann, 2015.

[38] ANANDTECH, «The AMD Radeon R9 Nano Review: The Power of Size,» [En línea]. Available: http://www.anandtech.com/show/9621/the-amd-radeon-r9-nano-review. [Último acceso: 2015].

[39] Qualcomm, «Adreno GPU SDK,» [En línea]. Available: https://developer.qualcomm.com/software/adreno-gpu-sdk/tools. [Último acceso: 2015].

[40] AMD, «HBM,» 2015. [En línea]. Available: http://www.amd.com/es-es/innovations/software-technologies/hbm.

[41] C.-L. Su, P.-Y. Chen, C.-C. Lan, L.-S. Huang y K.-H. Wu, «Overview and comparison of OpenCL and CUDA technology for GPGPU,» *Circuits and Systems (APCCAS),* 2012.

[42] J. Fang, A. Varbanescu y H. Sips, «A comprehensive performance comparison of CUDA and OpenCL,» *International conference on parallel processing (ICPP),* 2011.

[43] Khronos Group, «Khronos Releases OpenCL 2.1 Provisional Specification for Public Review,» 2015. [En línea]. Available: https://www.khronos.org/news/press/khronos-releases-opencl-2.1-provisional-specification-for-public-review.