



**ESPE**  
UNIVERSIDAD DE LAS FUERZAS ARMADAS  
INNOVACIÓN PARA LA EXCELENCIA

**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y  
TELECOMUNICACIONES**

**CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN  
Y CONTROL**

**TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL  
TÍTULO DE INGENIERO EN ELECTRÓNICA, AUTOMATIZACIÓN Y  
CONTROL**

**TEMA: IDENTIFICACIÓN EN TIEMPO REAL DE PÉRDIDA DE VÍA**

**AUTOR: VILLEGAS PICO, MARTÍN BELISARIO**

**DIRECTOR: ING. AGUILAR CASTILLO, WILBERT GEOVANNY, PHD**

**SANGOLQUÍ**

**2019**



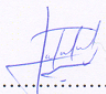
DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y  
TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA,  
AUTOMATIZACIÓN Y CONTROL

CERTIFICACIÓN

Certifico que el trabajo de titulación "*IDENTIFICACIÓN EN TIEMPO REAL DE PÉRDIDA DE VÍA*" fue realizado por el señor *Villegas Pico, Martín Belisario* el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustenten públicamente.

Sangolquí, 16 de diciembre del 2019

  
.....  
Ing. Wilbert G. Aguilar, PhD  
CI. 0703844696



**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y  
TELECOMUNICACIONES**

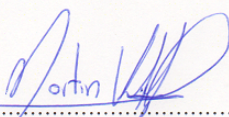
**CARRERA DE INGENIERÍA EN ELECTRÓNICA,  
AUTOMATIZACIÓN Y CONTROL**

**AUTORÍA DE RESPONSABILIDAD**

Yo, *Villegas Pico, Martín Belisario*, declaro que el contenido, ideas y criterios del trabajo de titulación: "*Identificación en tiempo real de pérdida de vía*" es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, 16 de diciembre del 2019



.....  
**Villegas Pico, Martín Belisario**  
CI. 1803727104




DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y  
TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA,  
AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

*Yo, Villegas Pico, Martín Belisario, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: "Identificación en tiempo real de pérdida de vía" en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.*

Sangolquí, 16 de diciembre del 2019

  
.....  
Villegas Pico, Martín Belisario  
CI. 1803727104

# *Dedicatoria*

*Dedicado a Dios,  
a mis padres  
y hermano.*

## *Agradecimiento*

Con la realización de este trabajo, quiero expresar mis más sinceros sentimientos de gratitud, cariño y respeto a Dios, a mi padre, a mi madre y a mi hermano quienes siempre han estado apoyándome en cada etapa de mi vida. También agradezco a la Universidad de las Fuerzas Armadas ESPE, a sus docentes y a todas las personas con quienes he podido compartir conocimientos y experiencias que han servido para crecer personal y profesionalmente para construir el camino hacia mis objetivos.

De todo corazón,  
Martín Villegas Pico

---

# Índice de Contenidos

<b>CERTIFICADO DEL DIRECTOR . . . . .</b>	<b>i</b>
<b>AUTORÍA DE RESPONSABILIDAD . . . . .</b>	<b>ii</b>
<b>AUTORIZACIÓN . . . . .</b>	<b>iii</b>
<b>Dedicatoria . . . . .</b>	<b>iv</b>
<b>Agradecimiento . . . . .</b>	<b>v</b>
<b>Índice de Contenidos . . . . .</b>	<b>vi</b>
<b>Índice de Figuras . . . . .</b>	<b>xi</b>
<b>Índice de Tablas . . . . .</b>	<b>xiv</b>
<b>Resumen . . . . .</b>	<b>xvi</b>
<b>Abstract . . . . .</b>	<b>xvii</b>

---

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1.	Antecedentes	1
1.2.	Justificación e Importancia	2
1.3.	Alcance del proyecto	3
1.4.	Objetivos	3
1.4.1	General	3
1.4.2	Específicos	3
<b>2</b>	<b>Estado del Arte</b>	<b>4</b>
2.1.	Visión artificial y detección de líneas de carretera	4
2.2.	Sensores para visión artificial	7
2.3.	Tiempo de respuesta del conductor	8
<b>3</b>	<b>Detección de líneas y pérdida de vía</b>	<b>10</b>
3.1.	Arquitectura de la red neuronal convolucional VGG16	11
3.1.1	Filtro convolucional	11
3.1.2	Max Pooling	12
3.1.3	Unidad lineal rectificada (ReLU)	13
3.1.4	Función exponencial normalizada (Softmax)	14
3.1.5	Sumario	14
3.2.	Detector de bordes Canny	15
3.3.	Transformada de Hough	18



---

3.4. Filtro Butterworth . . . . .	19
3.4.1 Diseño del filtro digital . . . . .	21
<b>4 Descripción del experimento . . . . .</b>	<b>22</b>
4.1. Definiciones preliminares . . . . .	23
4.1.1 Nvidia CUDA . . . . .	23
4.1.2 Nvidia cuDNN . . . . .	24
4.1.3 TensorFlow . . . . .	25
4.1.4 OpenCV . . . . .	26
4.2. Requerimientos de hardware y software . . . . .	26
4.2.1 Hardware . . . . .	26
4.2.2 Sistema operativo . . . . .	27
4.2.3 Tensorflow . . . . .	27
4.2.3.1 Requerimientos previa la instalación de Tensorflow . . . . .	27
4.2.3.2 Re-compilación e instalación . . . . .	29
4.3. Implementación de la RCN . . . . .	30
4.3.1 Mapas de segmentación binaria . . . . .	30
4.4. Segmentación binaria con una red VGG16 en Tensorflow . . . . .	36
4.4.1 Aspectos más importantes de la arquitectura desarrollada . . . . .	36
4.4.2 Desarrollo de la arquitectura . . . . .	36
4.4.2.1 Codificador . . . . .	36

---

4.4.2.2	Decodificador . . . . .	39
4.4.3	Generación de modelos matemáticos de la líneas de carretera . . . . .	41
4.4.4	Filtración en tiempo real . . . . .	46
4.4.4.1	Diseño del filtro pasabajos Butterworth de segundo orden . . . . .	46
4.4.5	Detección de pérdida de vía . . . . .	48
<b>5</b>	<b>Análisis de resultados . . . . .</b>	<b>51</b>
5.1.	Métricas de evaluación . . . . .	51
5.1.1	Función de pérdida por entropía cruzada . . . . .	51
5.1.2	Sensibilidad y especificidad de un sistema . . . . .	52
5.2.	Evaluación del sistema . . . . .	53
5.2.1	Descripción de escenarios de evaluación . . . . .	53
5.2.2	Eficiencia del aprendizaje de la red neuronal . . . . .	53
5.2.3	Desempeño del filtro y detección de pérdida de vía . . . . .	59
5.2.3.1	Comparativa de señales . . . . .	60
5.2.3.2	Comparación cuantitativa de $m_i + m_d$ . . . . .	64
5.2.3.3	Falsos positivos, negativos y sensibilidad del sistema . . . . .	69
5.3.	Tiempo de procesamiento . . . . .	69
<b>6</b>	<b>Conclusiones y trabajo futuro . . . . .</b>	<b>75</b>
6.1.	Conclusiones . . . . .	75
6.2.	Trabajo futuro . . . . .	76

---

**Referencias . . . . . 77**

# Índice de Figuras

<i>Figura 1</i>	Segmentación de imágenes . . . . .	5
<i>Figura 2</i>	Tiempos de respuesta de un conductor ante diversos estímulos . . . . .	9
<i>Figura 3</i>	Gráfica de la función de activación ReLU. . . . .	13
<i>Figura 4</i>	Esquemático de la arquitectura VGG16 . . . . .	14
<i>Figura 5</i>	Gradiente para la eliminación de no-máximos. . . . .	17
<i>Figura 6</i>	Umbral de histéresis. . . . .	17
<i>Figura 7</i>	Representación de un punto de una recta en coordenadas polares. . . . .	18
<i>Figura 8</i>	Curvas sinusoides generadas a partir de pares ordenados de una recta . . . .	19
<i>Figura 9</i>	Comparativa de la respuesta de un filtro Butterworth (rojo) con respecto a otros . . . . .	20
<i>Figura 10</i>	Diagrama de bloques funcional del proyecto. . . . .	22
<i>Figura 11</i>	Imagen de carretera con la cual se generará una máscara binaria . . . . .	31
<i>Figura 12</i>	Configuración del color de fondo y frente en GIMP . . . . .	31
<i>Figura 13</i>	Configuración de propiedades para la creación de una nueva capa . . . . .	32
<i>Figura 14</i>	Opciones de visibilidad y bloqueo de las capas previo al subrayado de las líneas. . . . .	33
<i>Figura 15</i>	Configuración de la herramienta lápiz . . . . .	33
<i>Figura 16</i>	Subrayado de las líneas de carretera . . . . .	34
<i>Figura 17</i>	Opciones de visibilidad y bloqueo de las capas previo a la exportación de la máscara . . . . .	34
<i>Figura 18</i>	Configuración de parámetros para la exportación de la máscara . . . . .	35
<i>Figura 19</i>	Mapa de segmentación de las líneas de carretera. . . . .	35
<i>Figura 20</i>	Secuencia de realización de convoluciones y reducciones en el codificador. .	39
<i>Figura 21</i>	Abajo: Máscara generada por la red neuronal de la imagen de la carretera superior . . . . .	41
<i>Figura 22</i>	Detección de bordes mediante el algoritmo de Canny . . . . .	42
<i>Figura 23</i>	Líneas detectadas mediante la transformada de Hough . . . . .	43
<i>Figura 24</i>	Sistema de referencia de una imagen . . . . .	44

<i>Figura 25</i>	Lineas de carretera en el sistema de referencia . . . . .	44
<i>Figura 26</i>	Diagrama de flujo del algoritmo de clasificación de líneas . . . . .	45
<i>Figura 27</i>	Líneas de carretera detectadas y clasificadas . . . . .	46
<i>Figura 28</i>	Comportamiento de la pendiente de la línea derecha respecto al tiempo sin filtración de ruido . . . . .	47
<i>Figura 29</i>	Comportamiento del filtro Butterworth pasabajos . . . . .	48
<i>Figura 30</i>	Los datos de $m_i + m_d$ obedecen a una distribución de Gauss. La región en azul representa que el auto esta centrado en la vía. . . . .	50
<i>Figura 31</i>	Variación del error a lo largo del entrenamiento de la red neuronal. . . . .	54
<i>Figura 32</i>	Variación de la exactitud del sistema. . . . .	54
<i>Figura 33</i>	Resultados generados por la red neuronal. Cada fila es un escenario diferente. La segunda columna es la máscara generada por la red neuronal, la tercera es el resultado final (en calibración) y la cuarta es el sistema detectando cuando el vehículo pierde la trayectoria correcta . . . . .	55
<i>Figura 34</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 1A . . .	60
<i>Figura 35</i>	Señal filtrada del comportamiento de las pendientes del escenario 1A . . .	60
<i>Figura 36</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 1B . . .	60
<i>Figura 37</i>	Señal filtrada del comportamiento de las pendientes del escenario 1B . . .	60
<i>Figura 38</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 2A . . .	61
<i>Figura 39</i>	Señal filtrada del comportamiento de las pendientes del escenario 2A . . .	61
<i>Figura 40</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 2B . . .	61
<i>Figura 41</i>	Señal filtrada del comportamiento de las pendientes del escenario 2B . . .	61
<i>Figura 42</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 2C . . .	62
<i>Figura 43</i>	Señal filtrada del comportamiento de las pendientes del escenario 2C . . .	62
<i>Figura 44</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 3A . . .	62
<i>Figura 45</i>	Señal filtrada del comportamiento de las pendientes del escenario 3A . . .	62
<i>Figura 46</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 3B . . .	63
<i>Figura 47</i>	Señal filtrada del comportamiento de las pendientes del escenario 3B . . .	63
<i>Figura 48</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 4B . . .	63
<i>Figura 49</i>	Señal filtrada del comportamiento de las pendientes del escenario 4B . . .	63
<i>Figura 50</i>	Señal sin filtrar del comportamiento de las pendientes del escenario 5A . . .	64
<i>Figura 51</i>	Señal filtrada del comportamiento de las pendientes del escenario 5A . . .	64
<i>Figura 52</i>	Tiempo de procesamiento por cuadro del escenario 1A . . . . .	70
<i>Figura 53</i>	Tiempo de procesamiento por cuadro del escenario 1B . . . . .	70
<i>Figura 54</i>	Tiempo de procesamiento por cuadro del escenario 2A . . . . .	71
<i>Figura 55</i>	Tiempo de procesamiento por cuadro del escenario 2B . . . . .	71
<i>Figura 56</i>	Tiempo de procesamiento por cuadro del escenario 2C . . . . .	72
<i>Figura 57</i>	Tiempo de procesamiento por cuadro del escenario 3A . . . . .	72

---

<i>Figura 58</i>	Tiempo de procesamiento por cuadro del escenario 3B . . . . .	73
<i>Figura 59</i>	Tiempo de procesamiento por cuadro del escenario 4B . . . . .	73
<i>Figura 60</i>	Tiempo de procesamiento por cuadro del escenario 5A . . . . .	74

# Índice de Tablas

Tabla 1	<i>Sumario de la arquitectura VGG16</i> . . . . .	15
Tabla 2	<i>Comparativa del número de núcleos y precios de las CPUs y GPUs</i> . . . . .	23
Tabla 3	<i>Descripción de los principales componentes de hardware del ordenador donde se desarrolla este proyecto</i> . . . . .	27
Tabla 4	<i>Software preliminar requerido para instalar TensorFlow</i> . . . . .	29
Tabla 5	<i>Valores RGB asignados a la segmentación binaria</i> . . . . .	30
Tabla 6	<i>Comparación de los arreglos de la capa 12 y 13</i> . . . . .	38
Tabla 7	<i>Comportamiento de la pendientes en el caso ideal.</i> . . . . .	48
Tabla 8	<i>Comportamiento de la pendientes con compensación por variación de la posición de la cámara</i> . . . . .	49
Tabla 9	<i>Descripción de los escenarios de prueba</i> . . . . .	53
Tabla 10	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 1A</i> . . .	56
Tabla 11	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 1B</i> . . .	56
Tabla 12	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 2A</i> . . .	56
Tabla 13	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 2B</i> . . .	57
Tabla 14	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 2C</i> . . .	57
Tabla 15	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 3A</i> . . .	57
Tabla 16	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 3B</i> . . .	58
Tabla 17	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 4A</i> . . .	58
Tabla 18	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 4B</i> . . .	58
Tabla 19	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 5A</i> . . .	59
Tabla 20	<i>Eficiencia de la clasificación de las líneas de carretera del escenario 5B</i> . . .	59
Tabla 21	<i>Comparativa de pendientes del escenario 1A</i> . . . . .	64
Tabla 22	<i>Comparativa de pendientes del escenario 1B</i> . . . . .	65
Tabla 23	<i>Comparativa de pendientes del escenario 2A</i> . . . . .	65
Tabla 24	<i>Comparativa de pendientes del escenario 2B</i> . . . . .	65
Tabla 25	<i>Comparativa de pendientes del escenario 2C</i> . . . . .	66
Tabla 26	<i>Comparativa de pendientes del escenario 3A</i> . . . . .	66

---

Tabla 27	<i>Comparativa de pendientes del escenario 3B</i> . . . . .	66
Tabla 28	<i>Comparativa de pendientes del escenario 4A</i> . . . . .	67
Tabla 29	<i>Comparativa de pendientes del escenario 4B</i> . . . . .	67
Tabla 30	<i>Comparativa de pendientes del escenario 5A</i> . . . . .	67
Tabla 31	<i>Comparativa de pendientes del escenario 5B</i> . . . . .	68
Tabla 32	<i>Error cuadrático medio del las pendientes calculadas manualmente respecto a las calculadas por el sistema</i> . . . . .	68
Tabla 33	<i>Falsos positivos, negativos, sensibilidad y especificidad del sistema</i> . . . . .	69



## RESUMEN

En Ecuador, un gran porcentaje de accidentes de tránsito en la actualidad se producen debido a la pérdida de vía no intencionada de un vehículo debido a situaciones de distracción o cansancio de los conductores; siendo estos factores los causantes de alrededor del 25 % de los siniestros de tránsito totales ocurridos en el país. En este trabajo de investigación, se propone un prototipo de un sistema de alerta de pérdida de vía ejecutada en tiempo real; el cual proporciona al conductor una alerta cuando su vehículo está fuera de la trayectoria correcta en la carretera para que pueda reaccionar a tiempo y evite un accidente. Para lograr esto, mediante unidades de procesamiento de gráficos (GPUs), se usaron redes neuronales convolucionales para clasificar las líneas de carretera conjuntamente con varios métodos heurísticos de procesamiento de imágenes como la transformada de Hough y el detector de bordes de Canny. Para determinar cuando el vehículo entra en pérdida de vía, se ha realizado el análisis de las pendientes de las líneas de carretera para definir un umbral donde se considera que el vehículo sigue una trayectoria correcta. De esta forma, mediante alertas sonoras el conductor puede tomar medidas de emergencia necesarias lo más pronto posible para regresar a la trayectoria correcta y evitar una colisión. Con esto, se busca mitigar el número de siniestros de tránsito ocasionados por la pérdida de vía.

### **PALABRAS CLAVE:**

- **VEHÍCULOS AUTÓNOMOS**
- **MACHINE LEARNING**
- **REDES NEURONALES CONVOLUCIONALES**
- **DETECCIÓN EN TIEMPO REAL**

## ABSTRACT

In Ecuador, a large percentage of traffic accidents currently occur due to unintentional lane departure of a vehicle due to situations of distraction or fatigue of drivers; these factors have been the cause of around 25 % of the total traffic accidents occurred in the country. In this research work, a prototype of a real-time lane departure warning system is proposed; which provides the driver with an alert when his vehicle is off the right path on the road so he can react on time and avoid an accident. To achieve this, using graphic processing units (GPUs), convolutional neural networks were used to classify road lines in conjunction with various heuristic image processing methods such as the Hough transform and the Canny edge detector. To determine when lane departure occurs, the analysis of the slopes of the road lines has been carried out to define a threshold where the vehicle is considered to be following a correct path. In this way, through sound alerts the driver can take emergency measures as soon as possible to return to the correct path and avoid a collision. With this, we seek to mitigate the number of traffic accidents caused by lane departure due to distraction or fatigue.

### **KEYWORDS:**

- **AUTONOMOUS VEHICLES**
- **MACHINE LEARNING**
- **CONVOLUTIONAL NEURAL NETWORKS**
- **REAL TIME DETECTION**

# Capítulo 1

## Introducción

En Ecuador, un gran número de siniestros en las vías ocurren debido a la pérdida de vía del vehículo cuando el conductor se distrae o está cansado. El uso de la tecnología puede llegar a ser muy útil para prevenir accidentes de tránsito de este tipo. En efecto, en fracciones de segundo, se puede detectar cuando el vehículo ha comenzado a circular en una trayectoria peligrosa e informar inmediatamente al conductor para que entre en alerta y evite accidentarse.

### 1.1. Antecedentes

La percepción digital de la trayectoria de una carretera es crucial para desarrollar sistemas avanzados de asistencia al conductor. Este campo de investigación ha sido muy activo durante las últimas décadas lo cual ha generado un considerable progreso durante los últimos años (Bar Hillel y cols., 2014; Dickmanns y Mysliwetz, 1992; Zhang, 1991). Sin embargo, el principal problema en el desarrollo de estos sistemas es la limitación de los sistemas digitales para identificar adecuadamente la trayectoria de una carretera bajo condiciones de incertidumbre (Thorpe y cols., 1991). La percepción de la trayectoria de una carretera depende, primero, de la variabilidad del entorno de las vías; por ejemplo, las señalizaciones, los colores y las texturas, las cuales pueden variar en gran medida a lo largo de las autopistas (McCall y Trivedi, 2006). Otro factor, es el tipo de herramienta que se utiliza para adquirir los datos de las vías, siendo unas de las herramientas más utilizadas

las cámaras (Bar Hillel y cols., 2014; McCall y Trivedi, 2006). Mediante el procesamiento de imágenes de video de cámaras, además de datos de otros sensores, se han logrado sistemas capaces de detectar las señalizaciones de vías en tiempo real (Felisa y Zani, 2010; Huang y cols., 2009).

## 1.2. Justificación e Importancia

En el Ecuador, un gran porcentaje de accidentes de tránsito ocurren debido a vehículos que pierden su trayectoria correcta en la calzada; y, cuyas consecuencias son colisiones con otros vehículos, caídas de vehículos al vacío, personas atropelladas, daños a bienes públicos y privados; todas poniendo en riesgo vidas humanas. En efecto, según la Agencia Nacional de Tránsito (ANT), a septiembre del 2018 se han registrado 273 siniestros de tránsito por conducir en malas condiciones físicas (sueño, cansancio, fatiga) y 4450 por conducir con desatención a las condiciones de tránsito; situaciones que, conjuntamente, representan el 25 % de los factores causantes de accidentes de tránsito a nivel nacional. Dichos siniestros suman un saldo de 728 fallecidos que representan alrededor del 45 % del total de personas que han perdido la vida en las vías (Agencia Nacional de Tránsito (ANT), 2018a,b). Aunque existen programas nacionales de concientización de responsabilidad en la vías, los números antes mencionados siguen siendo importantes y requieren más alternativas que reduzcan el número de accidentes provocados en la vías.

Hablando cuantitativamente sobre las consecuencias de la distracción, cuando un conductor está concentrado en el volante, tarda de 0.4 a 0.6 segundos en reaccionar; sin embargo, cuando está distraído puede tardar en reaccionar hasta 1.5 segundos; es decir, se puede llegar a duplicar el tiempo de reacción. (Žuraulis y cols., 2018). Por otro lado, cuando al conductor distraído se le proporciona algún tipo de estímulo; por ejemplo, un estímulo auditivo, el individuo entra en alerta más rápidamente, disminuyendo el tiempo de reacción y permitiendo aumentar las posibilidades de prevenir un accidente (van der Heiden y cols., 2018). Consecuentemente, este proyecto de investigación se desarrolla para emitir una alerta a los conductores cuando su vehículo haya comenzado a circular incorrectamente por la calzada debido a la pérdida de atención al volante, ya sea por situaciones físicas o distracciones. De esta forma, se busca disminuir los siniestros de tránsito; al mismo tiempo

que se busca reducir las pérdidas humanas y materiales.

### **1.3. Alcance del proyecto**

El proyecto tiene la finalidad de desarrollar un programa informático que se ejecute en tiempo real. El cual, en no más de 0.3 segundos detecte las líneas de carretera y alerte al conductor cuando su vehículo entre en pérdida de vía. El proyecto, inicialmente, está orientado a operar en vehículos livianos que circulen en carreteras rectas a una velocidad máxima de 50 *km/h*, que es el límite máximo de velocidad establecido por la ANT para vehículos livianos en zonas urbanas.

### **1.4. Objetivos**

#### **1.4.1. General**

Diseñar e implementar un sistema que alerte al conductor cuando su vehículo siga una trayectoria incorrecta en la vía.

#### **1.4.2. Específicos**

- Identificar las características de una carretera urbana como la estructura física, texturas y señalización horizontal.
- Experimentar con varios métodos de solución y elegir el más eficaz.
- Diseñar e implementar los algoritmos de detección y pérdida de vía.
- Implementar un prototipo funcional del sistema.

## Capítulo 2

### Estado del Arte

#### 2.1. Visión artificial y detección de líneas de carretera

La visión artificial es el conjunto de técnicas que permiten a las computadoras comprender el mundo real, generalmente, a través del procesamiento de imágenes y videos. Es así como la visión artificial está presente en varias actividades del día a día como la detección de texto, el reconocimiento facial, la identificación de enfermedades y la conducción autónoma de vehículos (Szeliski, 2010). Hablando de este último, con la visión artificial se puede procesar cada imagen de la carretera para detectar las líneas pintadas sobre ella para determinar trayectorias y límites viales (Y. Zhou y cols., 2006). Esta técnica es muy útil cuando existen problemáticas como sombras y cambios de luz que afectan las texturas de las vías y la presencia de otros vehículos u obstáculos que disminuyen la visibilidad; ya que se pueden atenuar y después detectar los patrones viales eficientemente (Bertozzi y Broggi, 1998). Por ejemplo, para este fin, la universidad The Carnegie Mellon ha desarrollado el sistema AURORA que identifica las líneas de carrera en cada imagen obtenida mediante una cámara a color montada en el vehículo en ángulo degradado (M. Chen y cols., 1995). Además de detectar las líneas y límites de una carretera, se puede determinar la posición relativa de un vehículo con respecto a la carretera con la finalidad de saber cuando el auto sigue una trayectoria peligrosa y hacerle saber al conductor para que tome precauciones. El algoritmo para calcular el tiempo de cruce de línea (TLC, por sus siglas en inglés) permite predecir

cuando el auto este a punto de salirse de los límites de la carretera analizando la dirección de la línea vial y la posición relativa del vehículo; y con esto, detectar los niveles de fatiga y desatención del conductor (Risack y cols., 2000).

Detectar cualquier objeto en una imagen, como la línea de una carretera, es un gran desafío debido a la gran cantidad de píxeles que se deben procesar y relacionar para identificar un patrón semejante al que se busca dentro de la imagen (Krizhevsky y cols., 2012). Una de las técnicas más útiles es la segmentación de imágenes que consiste en la clasificación de áreas u objetos dentro de la imagen basado en características que estos puedan compartir como colores de los píxeles, formas y texturas (Palomino y Concha, 2009). En 1 se ilustra un ejemplo de segmentación de imágenes.



**Figura 1.** Segmentación de imágenes: Las líneas horizontales de la carretera de la imagen superior se han segmentado y se ha generado la máscara de la imagen inferior

Algunos algoritmos para segmentación de imágenes usan escala de grises para clasificar las regiones de la imagen. Para lograrlo, se analizan los cambios en la intensidad del color monocromático, las continuidades del color, los patrones similares y las líneas de la imagen; sin embargo, esto solo sirve cuando la imagen esta en escala de grises (González y Woods, 1996). Para segmentar imágenes a color uno de los métodos más populares es la segmentación semántica que asigna a cada pixel una clase, que representa el objeto más probable al que pertenece (L.-C. Chen y cols., 2014). Otras técnicas crean regiones de píxeles y les otorgan etiquetas únicas para segmentar las imágenes (Dai y cols., 2015; Hariharan y cols., 2014). Aunque existe una gran cantidad de algoritmos que con los que se puede realizar segmentación de imágenes, ninguna logra realizarla a la perfección porque, en la mayoría, la entrada de datos debe ser de un tamaño específico, lo que generalmente resulta en la pérdida de resolución en la imagen y consecuentemente causando que a muchos objetos se los distinga con una clase que no les corresponde (Noh y cols., 2015). Una de las técnicas que mejor logra llevar a cabo la segmentación semántica es el aprendizaje profundo a través de redes neuronales artificiales, que mediante sus conexiones se pueden generar «píxeles inteligentes» que determinan a que clase de objeto pertenece una región en la imagen (Erhan y cols., 2014).

Las redes neuronales convolucionales (RNC) son un tipo especial de redes neuronales artificiales las cuales, a diferencia de estas últimas, donde cada neurona de una capa se interconecta con todas y cada una de las neuronas de las capas adyacentes; en las RNCs solo se conectan neuronas que comparten propiedades entre si. Para hacer esto se establecen convoluciones o «filtros» y agrupaciones de máximos que toman un conjunto de elementos y generan características únicas entre ellas que generan menos conexiones entre neuronas y menos parámetros lo que se traduce a menor costo computacional (Long y cols., 2015). Además, las RNCs tienen la ventaja de realizar dos principales tareas: la convolución y la desconvolución de una imagen. Por un lado, la convolución significa reducir los parámetros de procesamiento de la imagen para buscar relaciones y asignar clases a los diferentes objetos de la imagen. En el otro caso, la desconvolución, es el proceso contrario; es decir, a partir de las clases que se han generado se aumentan los parámetros para generar una imagen segmentada (Simonyan y Zisserman, 2014). En otras palabras, con las RNCs se puede insertar una imagen y obtener a la salida todas sus regiones clasificadas. Aunque todas tienen el mismo principio básico de funcionamiento, existen varias arquitecturas para implementar una RCN. Están principalmente varían en el número de capas de convolución y desconvolución



así como la dimensión del filtro de procesamiento de píxeles, siendo una de las más populares la VGG16 con 16 capas de convolución y un filtro de dimensión  $3 \times 3$ . Esta es una de las arquitecturas más usadas debido a su arquitectura uniforme que facilita su implementación y extracción de características de una imagen (Qassim y cols., 2018).

Las RNCs son altamente robustas detectando líneas de carretera incluso cuando las condiciones de luz y clima son adversas e incluso cuando las líneas que dividen las vías son borrosas; a diferencia de otros métodos, como la transformada de Hugh y el detector de bordes Canny que se usan para definir modelos matemáticos a partir de patrones en las imágenes, que tienden a fallar ante variaciones de la calzada e incluso pueden generar un mayor costo computacional (Neven y cols., 2018). Sin embargo, a pesar de tener ciertas limitaciones, no se deben dejar de lado estas técnicas heurísticas; en efecto, pueden ser herramientas muy útiles para el post-procesamiento de las imágenes y obtención de los modelos matemáticos de las líneas de carretera.

Para la segmentación semántica de una imagen, las RCNs son muy útiles porque son capaces de procesar imágenes con un tamaño donde, la resolución es adecuada ( $512 \times 256px$ ) y permite clasificar los objetos correctamente y además se pueden procesar todos los píxeles sin la necesidad que exista una conexión neuronal compleja por cada neurona.

## 2.2. Sensores para visión artificial

Para usar la visión artificial en las carreteras se necesitan de sensores que alimenten al sistema con datos de la carretera; específicamente imágenes. Los autos se pueden equipar con detectores ultrasónicos o radares láser pero estos dispositivos presentan principalmente dos inconvenientes: son muy costosos y son de difícil instalación (Tseng, 2004).

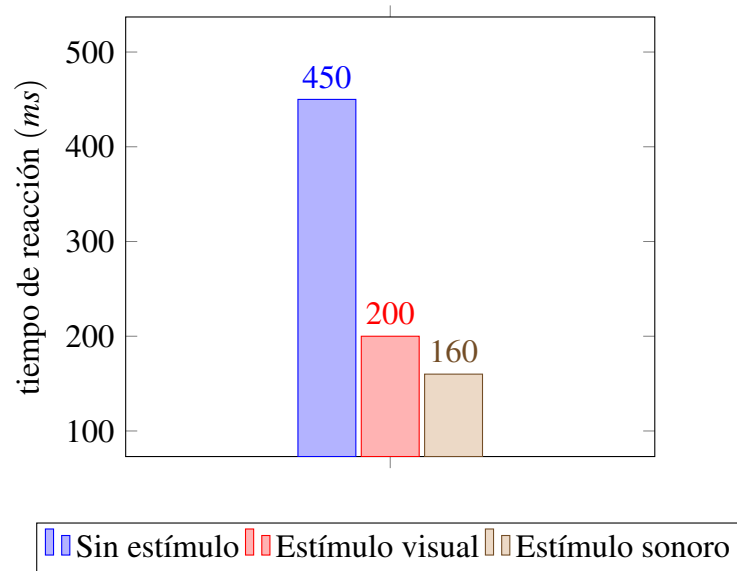
Entre los sensores más comunes para detectar líneas de carretera están los sensores LIDAR el cual a través de señales de radar que rebotan en cualquier obstáculo en la trayectoria del vehículo donde se ha instalado, se mapea la carretera en base a los datos de reflexión. Una de las desventajas de este sistema que su instalación y calibración deben ser sumamente precisas para que no existan errores de cálculo de distancia y de perspectiva. Además, el costo de este sistema es considerable

ya que en el mercado internacional, el precio mínimo de este equipo es de \$500 (Ogawa y Takagi, 2006; Takagi y cols., 2006).

Una herramienta alternativa son las cámaras que pueden ser instaladas fácilmente en el vehículo y son bastante accesibles económicamente. Existen dos tipos de cámaras: las monoculares y las binoculares; estas últimas son pueden ser muy útiles si se tiene como objetivo principal medir distancias en las carreteras; sin embargo, requieren de una adecuada y minuciosa calibración. (Goro y Onoguchi, 2018). Por otro lado, las cámaras monoculares son bastantes utilizadas en aplicaciones de detección de líneas de carretera ya que para esta aplicación no se requiere una calibración minuciosa y tampoco es necesario calcular distancias exactas. También, algoritmos de detección de líneas con el detector de bordes de Canny y la transformada de Hough son fácilmente aplicables en imágenes monoculares (and Ji-Hun Bae y Song, 2011; Huan y cols., 2008; X. Zhou y cols., 2003).

### **2.3. Tiempo de respuesta del conductor**

Varios estudios que se han realizado para determinar el tiempo de reacción de un conductor en base a cuanto demoran en pisar el pedal del freno previo a un estímulo de alerta concluyen que el tiempo de reacción es de entre 0.35 y 0.43 segundos. Estos tiempos se dilatan proporcionalmente a la edad de la persona; sin embargo, no hay variación en cuanto al género (Coley y cols., 2009; Warshawsky-Livne y Shinar, 2002). Cuando a un conductor se le provee algún tipo de estímulo ya sea visual, auditivo o táctil la reacción del conductor es mucho más rápida (Petermeijer y cols., 2017). Muchos investigadores han concluido que la reacción ante un estímulo sonoro es mucho más rápida que ante un estímulo visual; por ejemplo, el tiempo de respuesta ante un señal visual es de entre 180 – 200ms y ante una alerta sonora de 140 – 160ms, como se ilustra en la figura 2. Esto se debe, a que el estímulo sonoro llega mucho más rápido al cerebro (Galton, 1890).



**Figura 2.** Tiempos de respuesta de un conductor ante diversos estímulos

## Capítulo 3

# Detección de líneas y pérdida de vía

En este capítulo se explican todas la herramientas matemáticas y de procesamiento de imágenes que se han usado para desarrollar este proyecto, las mismas que se enlistan a continuación.

1. Arquitectura de la red neuronal convolucional VGG16
2. Detector de bordes de Canny
3. Transformada de Hough
4. Filtro Butterworth

## 3.1. Arquitectura de la red neuronal convolucional VGG16

Desarrollada por el Visual Geometry Group (VGG) de la universidad de Oxford, la arquitectura VGG16 es un modelo de red neuronal profunda de 16 capas para la detección de objetos en imágenes. El dato de entrada para este sistema es una imagen con formato RGB y dimensiones  $512 \times 256 \times 3$ . Donde a lo largo de las 12 primeras capas de predicción neuronal, se usan filtros o convoluciones con un campo receptivo pequeño de  $3 \times 3$  y función de activación ReLU y en las 4 capas restantes se usan capas completamente conectadas; de las cuales, las 3 primeras tienen función de activación ReLU y la última, función Softmax. Gráficamente, la arquitectura de esta red neuronal se ilustra en la figura 4. Además, la organización de la esta arquitectura se presenta en la tabla 1.

### 3.1.1. Filtro convolucional

La convolución es una herramienta matemática fundamental de una RCN que sirve para agrupar dos conjuntos de información. A través del filtro convolucional, o también denominado kernel, se crean mapas de características tales como el gradiente de píxeles, color, formas, discontinuidades, entre otros que permiten la clasificación de los objetos dentro de una imagen.

La convolución se realiza desplazando el kernel a través de la entrada de izquierda a derecha y de arriba hacia abajo, típicamente con paso de 1; y en cada desplazamiento, se efectúa la sumatoria del producto elemento a elemento de la entrada y el kernel. Matemáticamente, la convolución está definida por la ecuación 3.1, donde  $g(x,t)$  es el resultado de aplicar el kernel,  $f(x,y)$  es la imagen original y  $\omega$  es el kernel.

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^a \sum_{t=-b}^b \omega(s,t) f(x-s, y-t) \quad (3.1)$$

Los kernels usados por la red VGG16 en sus capas de convolución tienen dimensión  $3 \times 3$ . La razón de esta dimensión, es que, además de reducir el número de pesos de la red neuronal, permite lograr un filtro simétrico al tener un píxel central y un mismo número de píxeles alrededor de este

con el fin de prevenir errores de solapamiento y distorsión que ocasionan que la imagen se vuelva indistinguible y su reconstrucción sea problemática.

Debido a que la profundidad de las capas de una RCN no es unidimensional, el kernel se ajusta para que su tercera dimensión sea igual al número de capas. Por ejemplo, en la capa de entrada, la imagen esta compuesta por las 3 capas de color (R,G,B); entonces, el kernel tendrá dimensión  $3 \times 3 \times 3$ . Por otro lado, en la capa 1 donde existen 64 capas para extracción de características, el kernel es de  $3 \times 3 \times 64$  (referir a la tabla 1).

### 3.1.2. Max Pooling

Max Pooling es una operación matemática usada para reducir las dimensiones de un objeto multi-dimensional, como una imagen, tomando el número mayor en una región del objeto definida por una ventana de dimensión  $n$  que se desplaza de izquierda a derecha y de arriba hacia abajo con paso  $p$ . Es necesario saber que para obtener un resultado inverso aproximado de esta operación, denominada *unpooling* se debe guardar la posición del máximo en la matriz sin reducir.

En la ecuación 3.2 se ha reducido la matriz  $A$  a la mitad de sus dimensiones usando MaxPooling con una ventana de dimensión  $d$  y paso  $p$  igual 2.

Si:

$$A_{4 \times 4} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix}$$

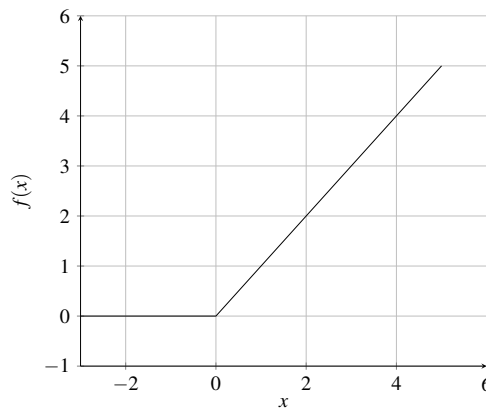
Entonces, si  $d = p = 2$ :

$$MaxPool_p^d(A) = \begin{pmatrix} \max \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} & \max \begin{pmatrix} a_{1,3} & a_{1,4} \\ a_{2,3} & a_{2,4} \end{pmatrix} \\ \max \begin{pmatrix} a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{pmatrix} & \max \begin{pmatrix} a_{3,3} & a_{3,4} \\ a_{4,3} & a_{4,4} \end{pmatrix} \end{pmatrix} \quad (3.2)$$

### 3.1.3. Unidad lineal rectificada (ReLU)

La unidad lineal rectificada (ReLU) es una de las funciones de activación más comúnmente usadas en RCNs. Matemáticamente, la función ReLU está definida por la ecuación 3.3. Esta función retorna el mismo valor de  $x$  cuando  $x$  es positiva y 0 cuando  $x$  es negativa. La gráfica de la función ReLU se ilustra en 3.

$$f(x) = \max(0, x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (3.3)$$



**Figura 3.** Gráfica de la función de activación ReLU.

#### Características

- Su cálculo requiere un costo computacional mínimo.
- Converge rápidamente debido a su comportamiento lineal que evita que su derivada se sature a 0 cuando  $x$  es muy grande.
- Siendo igual a cero cuando  $x$  es negativa, ReLU es una aproximación a una red neuronal biológica donde no todas las neuronas se activan al mismo tiempo.

### 3.1.4. Función exponencial normalizada (Softmax)

La función exponencial normalizada comúnmente conocida como Softmax es una función de activación que se suele usar en la última capa de un clasificador con redes neuronales cuyo resultado será el evento más probable calculado por esta función. Matemáticamente, la función Softmax esta definida por la ecuación:

$$F(X_i) = \frac{\exp(X_i)}{\sum_{j=0}^k \exp(X_j)} \quad i = 0, 1, 2, \dots, k \quad (3.4)$$

Características

- Cada salida es una probabilidad en el rango de 0 a 1
- La suma de las probabilidades de las salidas es igual a 1

### 3.1.5. Sumario

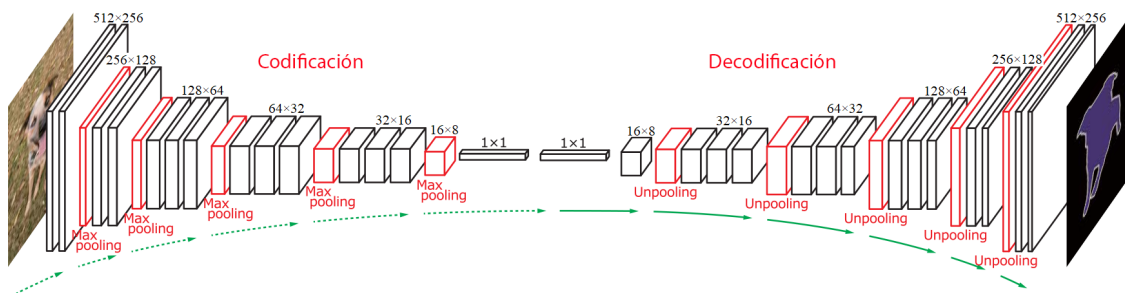


Figura 4. Esquemático de la arquitectura VGG16



**Tabla 1**

Sumario de la arquitectura VGG16

Capa	Descripción	Dimensión	Dim. del filtro	No. de filtros	Paso	Función
-	Imagen entrada	$512 \times 256 \times 3$	-	-	-	-
1,2	Convolución	$512 \times 256 \times 64$	$3 \times 3$	64	1	ReLU
-	Max Pooling	$256 \times 128 \times 64$	$2 \times 2$	-	2	-
3,4	Convolución	$256 \times 128 \times 128$	$3 \times 3$	128	1	ReLU
-	Max Pooling	$128 \times 64 \times 128$	$2 \times 2$	-	2	-
5,6	Convolución	$128 \times 64 \times 256$	$3 \times 3$	256	1	ReLU
-	Max Pooling	$64 \times 32 \times 256$	$2 \times 2$	-	2	-
7,8,9	Convolución	$64 \times 32 \times 512$	$3 \times 3$	512	1	ReLU
-	Max Pooling	$32 \times 16 \times 512$	$2 \times 2$	-	2	-
10,11,12	Convolución	$32 \times 16 \times 512$	$3 \times 3$	512	1	ReLU
-	Max Pooling	$16 \times 8 \times 512$	$2 \times 2$	-	2	-
13	Conexión densa	65536	1	-	-	ReLU
14	Conexión densa	4096	1	-	-	ReLU
15	Conexión densa	4096	1	-	-	ReLU
16	Conexión densa	1000	1	-	-	Softmax

## 3.2. Detector de bordes Canny

El detector de bordes Canny es un algoritmo multi-etapa para detectar un amplio rango de información estructural de una imagen, principalmente líneas y bordes desarrollado por John F. Canny. Es un algoritmo sencillo que usa pocos recursos computacionales y tiene gran eficiencia.

Las etapas de algoritmo son:

### 1. Filtro Gaussiano

La presencia de ruido en la imagen puede generar errores en la detección de los bordes como la presencia de falsos positivos o negativos. Para reducir el ruido, se realiza una operación de

convolución con un filtro Gaussiano de dimensiones  $(2k + 1) \times (2k + 1)$  que esta dado por la expresión matemática definida en 3.5

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i - (k + 1))^2 + (j - (k + 1))^2}{2\sigma^2}\right) ; \quad 1 \leq i, j \leq (2k + 1) \quad (3.5)$$

Cabe recalcar que la dimensión del filtro afecta al rendimiento de la detección. Es decir, mientras más grande sea la dimensión, menor será el campo receptivo del filtro. Entonces, por lo general, se usa un filtro de tamaño  $5 \times 5$ .

## 2. Intensidad del gradiente

Los bordes en una imagen pueden apuntar en diferentes direcciones: horizontal, vertical y diagonal. Por esta razón, se aplica el filtro de Sobel para obtener las derivadas o gradientes de intensidad de color horizontalmente ( $G_x$ ) y verticalmente ( $G_y$ ), para posteriormente calcular la magnitud ( $G$ ) del gradiente, que es perpendicular al borde, definida por la ecuación 3.6 y el ángulo  $\theta$  del gradiente con respecto a plano de la imagen definido por la ecuación 3.7

$$G = \sqrt{G_x^2 + G_y^2} \quad (3.6)$$

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad (3.7)$$

## 3. Eliminación de los valores no-máximos

Habiendo obtenido la magnitud y el ángulo del gradiente, se procesan cada uno de los píxeles para determinar si es un máximo en la región cercana al gradiente. Entonces, si el píxel no es un máximo, se considera que no pertenece al borde y se elimina (se asigna 0). En 5 se ilustra como el punto  $A$  pertenece a un borde a lo largo del eje vertical, y los puntos  $B$  y  $C$  están en dirección al gradiente; es decir; están perpendicularmente a este. Entonces, en base a  $B$  y  $C$  se analiza si  $A$  es un máximo local para continuar su procesamiento en la siguiente etapa.

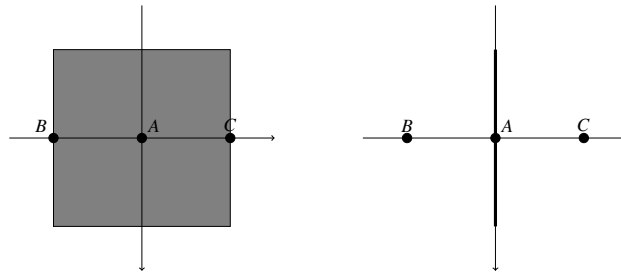


Figura 5. Gradiente para la eliminación de no-máximos.

#### 4. Umbral de histéresis

En esta última etapa se determina si una línea es un borde o no. Para lograrlo, se necesitan dos umbrales: un máximo y un mínimo, donde aquellos valores por encima del valor máximo se consideran como bordes verdaderos y aquellos por debajo del umbral mínimo se descartan. Para los valores que están entre los dos umbrales, es necesario determinar si tienen conexión con un borde verdadero, si este es el caso, se concluirá que son bordes verdaderos; caso contrario, también se descartarán. En 6 se ilustra como el segmento  $\overline{AB}$  es un borde verdadero y, por estar conectado a este,  $\overline{BC}$  también es un borde. Sin embargo, la curva  $\overline{AC}$ , al carecer de este tipo de conexión, es descartada como borde.

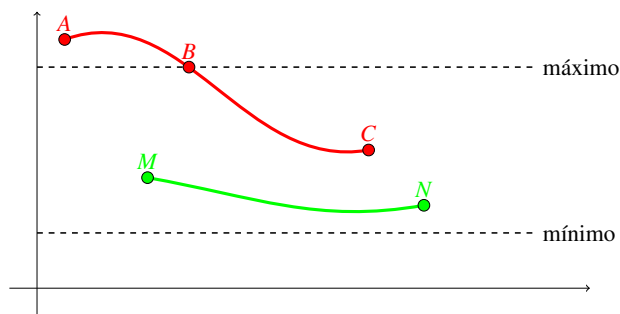


Figura 6. Umbral de histéresis. La curva verde no es un borde porque no tiene conexión alguna con un borde verdadero, como sucede con el segmento  $\overline{BC}$  de la curva roja.

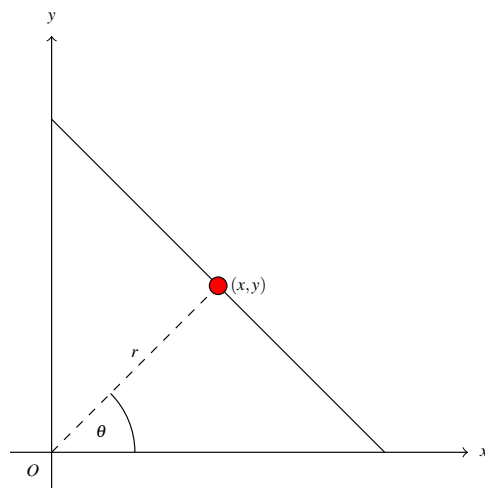
### 3.3. Transformada de Hough

La transformada de Hough es un algoritmo usado para la extracción de características de una figura en una imagen. Típicamente, se usa para la extracción de formas regulares como rectas, elipses y círculos. En el desarrollo de este proyecto, el uso de la transformada de Hough se centra en la identificación de rectas de la forma  $y = mx + b$

#### Principio de funcionamiento

La representación más simple de una recta es aquella que se expresa a través de los parámetros de la pendiente ( $m$ ) y el desplazamiento vertical ( $b$ ); sin embargo, cuando la recta es o esta cercana a ser vertical, el parámetro  $m$  se «desborda» alcanzando valores cercanos a  $-\infty$  y  $+\infty$ , lo que genera altos costos computacionales. Entonces, para prevenir el desbordamiento de parámetros se usan coordenadas polares, donde un punto cualquiera de una recta en coordenadas rectangulares puede ser expresado por medio de la ecuación 3.8, donde  $r$  es el vector desde el origen hacia el punto  $(x,y)$  y  $\theta$  es el ángulo que forma  $r$  con el eje  $x$ .

$$r = x \cos \theta + y \sin \theta \quad (3.8)$$



**Figura 7.** Representación de un punto de una recta en coordenadas polares.

En el sistema polar, por cada par ordenado  $(x, y)$ , se genera una familia de líneas definidas por 3.8, cuyos parámetros  $(\theta, r)$  al ser graficados como  $r = f(\theta)$  producen una curva sinusoidal. Si las curvas sinusoides de diferentes pares  $(x, y)$  se intersecan, dichos pares ordenados están contenidos dentro de una misma recta; y la coordenada  $(r, \theta)$ , donde se han intersecado estas curvas, representa un vector perpendicular a la recta. En 8, todas las curvas intersecan en el punto  $p$  porque todos los puntos, que han generado cada curva, pertenecen a la misma recta;  $p$  tiene parámetros  $(\theta, r)$  que generan un vector normal a la recta  $y = -x + 5$ .

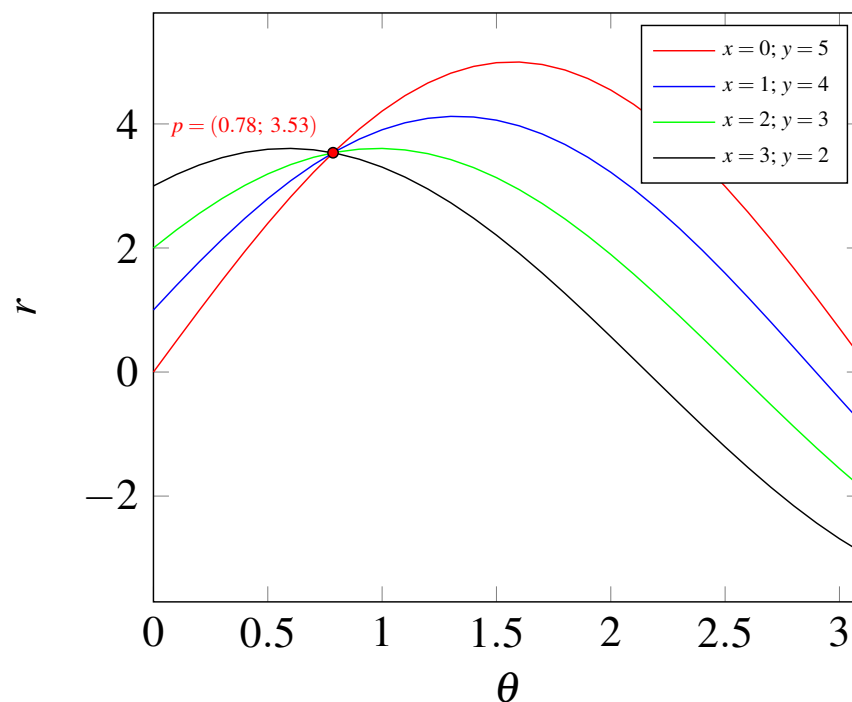
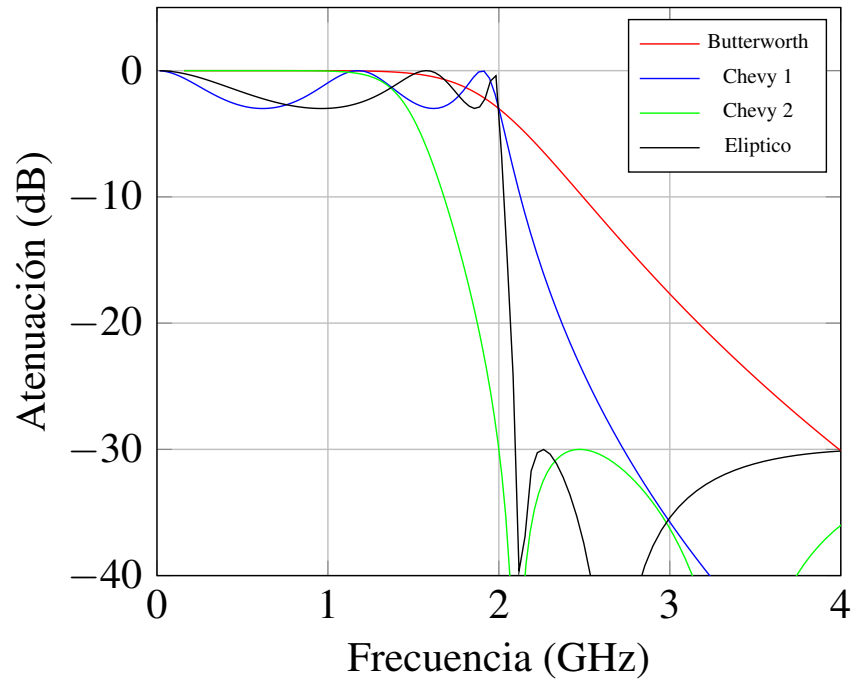


Figura 8. Curvas sinusoides generadas a partir de pares ordenados de una recta

### 3.4. Filtro Butterworth

El filtro Butterworth se usa para mantener respuesta constante y sin rizado en la banda de paso y de corte como se ilustra en la figura 9. Además, la salida del filtro solo se atenúa al llegar a la

frecuencia; de esta forma, el filtro Butterworth, a diferencia de otros filtros, es el que mejor logra alcanzar una respuesta plana.



**Figura 9.** Comparativa de la respuesta de un filtro Butterworth (rojo) con respecto a otros

Algunas de las características más sobresalientes de este filtro son:

1. Analizando las aproximaciones de Taylor para este filtro, se puede concluir que es el que más se acerca a ser un filtro ideal ya que no presenta atenuación en las bandas de paso ni de corte.
2. La frecuencia de corte para diseñar este filtro se selecciona en correspondencia con la ganancia de  $3dBs$

### 3.4.1. Diseño del filtro digital

El diseño del filtro digital Butterworth se lo realiza a partir de su forma análoga. De esta forma, la expresión matemática o función de transferencia de un filtro Butterworth análogo de segundo orden esta dado por:

$$H_a(s) = \frac{1}{s^2 + \sqrt{2}s + 1} \quad (3.9)$$

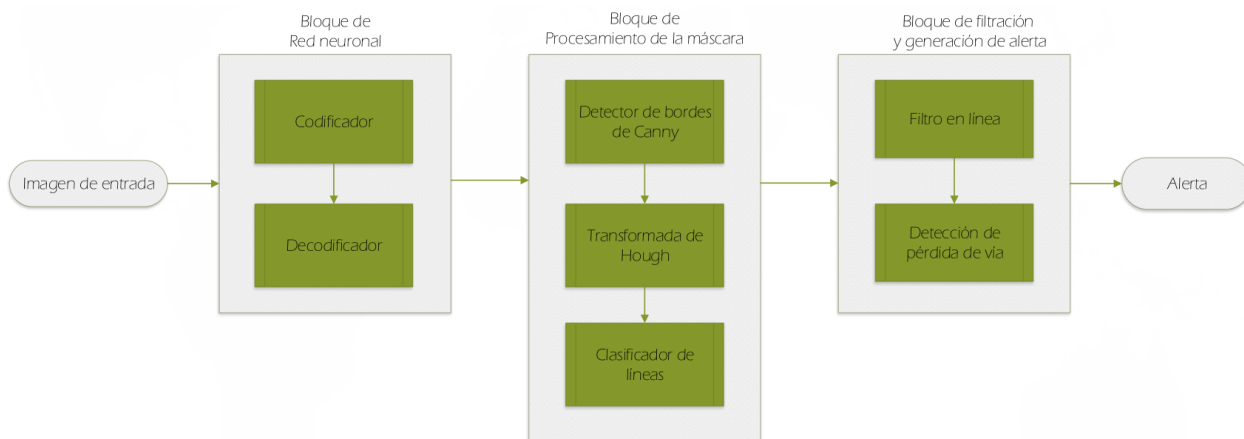
Para convertirlo a un filtro digital se debe pasar del dominio de  $s$  al dominio de  $z$  mediante la transformada bilinear donde  $s = c \frac{1-z^{-1}}{1+z^{-1}}$  y  $c = \cot(\omega_c T/2)$  siendo  $\omega_c$  la frecuencia de corte y  $T$  es el periodo de la señal. Entonces, la función de transferencia del filtro en su versión digital es:

$$H_d(z) = \frac{(1+z^{-1})^2}{c^2(1-z^{-1})^2 + c\sqrt{2}(1+z^{-2}) + 1} \quad (3.10)$$

## Capítulo 4

### Descripción del experimento

En este capítulo se describe el procedimiento de la implementación del sistema de alerta de pérdida de vía. La figura 10 muestra todos los bloques funcionales del proyecto desde que ingresa un cuadro o imagen del video hasta la generación de la alerta.



**Figura 10.** Diagrama de bloques funcional del proyecto.



## 4.1. Definiciones preliminares

### 4.1.1. Nvidia CUDA

En las inteligencias artificiales (IAs) se realizan millones de cálculos computacionales que involucran matrices, por lo que se necesitan unidades que puedan procesar estos cálculos en el menor tiempo posible. En un computador existen dos tipos de unidades de procesamiento: la unidad central de proceso (CPU) y la unidad de procesamiento de gráficos (GPU) ambas diseñadas para propósitos diferentes. Por un lado, la CPU es capaz de realizar diferentes tareas como interpretar la información generada por periféricos, manejo de memoria, etc; sin embargo, cada núcleo del CPU solo puede ejecutar una sola tarea a la vez. La GPU, por otro lado, está diseñada bajo la filosofía de procesar una inmensa cantidad de datos a la vez; por ejemplo, se pueden procesar múltiples píxeles al mismo tiempo para mostrar una imagen en pantalla. Si realizamos una comparación cuantitativa, una de las principales razones por las cuales las GPUs superan a las CPUs en procesamiento paralelo, es el número de procesos que cada uno puede ejecutar al mismo tiempo. En 2 se presenta una comparativa en el número de tareas en paralelo que pueden realizar diferentes CPUs y GPUs, además del costo de cada uno; entonces se puede concluir que en relación costo-beneficio las GPU son claramente la mejor opción para computaciones en paralelo.

#### Tabla 2

*Comparativa del número de núcleos y precios de las CPUs y GPUs basado en datos técnicos y precios obtenidos desde las páginas web oficiales de Intel, AMD y Nvidia*

Dispositivo	Tipo	Fecha	Procesos en paralelo	Precio
AMD Ryzen Threadripper 2990	CPU	Final 2018	64	\$1700
Intel Core i9-9900k	CPU	Final 2018	16	\$500
Nvidia Geforce GTX 770	GPU	Mediados 2013	960	\$128
Nvidia Geforce GTX 660	GPU	Mediados 2013	384	\$70

Siendo las aplicaciones de segmentación de imágenes con RNCs, una problemas de computación en paralelo en donde se generan un conjunto muy extenso se datos a ser procesados debido

al número de píxeles en la imagen se necesita una interfaz mediante la cual se pueda usar la GPU para el procesamiento.

La arquitectura de computación unificada (CUDA, por sus siglas en inglés) de Nvidia es una plataforma y extensión de código que permite realizar operaciones en paralelo a través de la utilización de unidades de procesamiento de gráficos (GPU) incrementando drásticamente la capacidad de procesamiento de datos (NVIDIA, 2019b). Es muy útil para operaciones que requieran cálculos matriciales o tensores como las aplicaciones con IAs. Al ser un tipo de extensión de programación más no un lenguaje, CUDA puede ser implementado en lenguajes muy populares y robustos como Python y C. El uso de la plataforma CUDA tiene los siguientes objetivos:

- Proveer de una serie de extensiones a lenguajes de programación estándares que permiten la ejecución en paralelo del código
- Dar soporte a aplicaciones donde se requiera el uso tanto de la CPU como de la GPU. Donde las tareas seriales se ejecuten en la CPU y aquellas que necesiten de procesamiento paralelo se ejecuten en la GPU

Los requerimientos para poder ejecutar CUDA en un sistema informático son los siguientes:

1. Una GPU con capacidad mínima CUDA 3.0
2. Una versión soportada de Linux
3. Kit de desarrollo Nvidia CUDA

#### **4.1.2. Nvidia cuDNN**

La librería de redes neuronales profundas de Nvidia CUDA (cuDNN, por sus siglas en inglés) es una librería acelerada por GPU que provee implementaciones primitivas para rutinas de convolución, desconvolución, normalización, capas de activación y agrupación en redes neuronales. El

uso de esta librería evita, en su mayoría, la compleja implementación matemática de la red neuronal en bajo nivel y permite que el desarrollador tenga mayor facilidad al desarrollar aplicaciones en alto nivel (NVIDIA, 2019a,c).

NVIDIA cuDNN, mediante sus implementaciones, acelera el procesamiento de las aplicaciones con RNCs para el reconocimiento de objetos en imágenes donde es necesario el procesamiento de inmensas cantidades de datos y el cálculo de millones de operaciones matemáticas con gran eficiencia para lograr una mínima latencia para la ejecución en tiempo real. Variaciones de la transformada rápida de Fourier (FFT) son los principales métodos matemáticos usados en cuDNN para la reducción de la complejidad matemática a lo largo de las capas de procesamiento de una RNC (Lavin y Gray, 2016).

### 4.1.3. TensorFlow

Tensorflow es una librería de código abierto desarrollado por Google para el desarrollo de modelos de aprendizaje artificial. Esta librería puede ser usada desde Python donde el principal tipo de dato que se maneja es el tensor. Un tensor no es más que una serie de valores numéricos dispuestos en un arreglo matricial de  $n$  dimensiones, a lo que, en TensorFlow, se denomina rango del tensor. (Google Inc., 2019). En 4.1, la matriz  $A$  representa un tensor de rango 2.

$$A_{3 \times 3} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \quad (4.1)$$

La arquitectura de TensorFlow esta distribuida de la siguiente forma:

- Pre-procesamiento de datos: A través de esta etapa se transforman los datos de entrada en comprensibles para la aplicación, corrigiendo, errores, inconsistencias o vacíos en la información.
- Desarrollo del modelo: Se selecciona el modelo que mejor se adapte a la aplicación y se realizan las configuraciones pertinentes de los parámetros del procesamiento de datos como

computaciones con GPU/CPU, asignación de la memoria, cantidad de información procesada en paralelo.

- Entrenamiento: Se obtienen los pesos del modelo de red neuronal.

#### 4.1.4. OpenCV

OpenCV es una librería de código abierto de visión y aprendizaje artificial desarrollada para facilitar el desarrollo de aplicaciones que involucren percepción de objetos, generalmente en imágenes, por computadora. Posee alrededor de 2500 algoritmos desarrollados a partir de investigaciones científicas así como métodos de aprendizaje artificial para detectar objetos, patrones, bordes, clasificar imágenes, identificar alteraciones en la cámara, reducir y aumentar las dimensiones de una imagen y también cambiar sus propiedades de color (OpenCV, 2019). Para el desarrollo de esta aplicación, los principales algoritmos de OpenCV usados son el detector de bordes de Canny y la detección de líneas a partir de la transformada de Hough.

## 4.2. Requerimientos de hardware y software

### 4.2.1. Hardware

Para instalar TensorFlow con capacidad de procesamiento en paralelo mediante GPUs, se necesita, preferiblemente, una GPU Nvidia (GPUs de otros fabricantes poseen soporte para TensorFlow mínimo o nulo) con capacidad mínima CUDA 3.0. Aunque en este proyecto, la GPU es la encargada de realizar la mayor parte del trabajo en cuanto a procesamiento de información, es recomendando que el CPU, la RAM y el disco duro dispongan de una buena velocidad de procesamiento y lectura de datos para acelerar algunas de las tareas que no realiza la GPU y aumentar la velocidad de comunicación entre los dispositivos involucrados.

En la tabla 3 se detallan los principales componentes de hardware del ordenador que servirán para desarrollar el proyecto.

**Tabla 3**

*Descripción de los principales componentes de hardware del ordenador donde se desarrolla este proyecto*

Descripción	Tipo	Velocidad	Núcleos	Memoria
Intel Core i7-4700MQ	CPU	2.4-3.4GHz	8	-
Nvidia GeForce GTX770M	GPU	2.0GHz	960 (CUDA)	3GB
Corsair Vegeance GDDR3	RAM	1.6GHz	-	24GB
Toshiba THNSNH256GMCT	SSD	6Gbps	-	256GB

### 4.2.2. Sistema operativo

En cuanto a software; primero, el sistema operativo debe utilizar muy poca cantidad de recursos de hardware para aprovechar todas las capacidades del ordenador en el procesamiento de este proyecto de ejecución en tiempo real, también debe brindar al usuario herramientas para usar los componentes de hardware sin restricciones, como realizar computaciones mediante GPUs y cambiar el modo de ejecución de sentencias del CPU. Por estas razones, se ha decidido usar la distribución de código abierto de Linux: Ubuntu 16.04; considerándose que esta versión es la más estable hasta el momento. Cabe recalcar, para que el sistema operativo pueda aprovechar al máximo los beneficios de hardware del ordenador es necesario que Ubuntu sea instalado en una partición del disco duro más no como una máquina virtual.

### 4.2.3. Tensorflow

#### 4.2.3.1. Requerimientos previa la instalación de Tensorflow

##### 1. Nvidia CUDA Toolkit

Para que Tensorflow se comunique con la GPU se debe instalar la plataforma CUDA siguiendo el siguiente procedimiento:

- **Verificar si la tarjeta de video tiene capacidad CUDA.** Refiriéndose a la tabla 3, la GPU a usarse tiene capacidad CUDA 3.0

- **Verificar la compatibilidad con el sistema operativo.** Ubuntu 16.04 es compatible con CUDA.
- **Verificar la instalación de GCC.** GNU Compiler Collection (GCC) es una colección de compiladores para lenguajes como C/C++, Fortran entre otros. GCC está instalado en el ordenador.
- **Descargar Nvidia CUDA Toolkit.** Los instaladores están disponibles gratuitamente en línea en la página de desarrolladores de Nvidia. Para Ubuntu 16.04 y otras distribuciones de Linux, la página web proporciona los comandos para instalar CUDA desde el terminal.

## 2. Nvidia cuDNN

La instalación de Nvidia cuDNN que contiene sentencias de bajo nivel para procesar aplicaciones de redes neuronales aceleradas con GPUs debe seguir el siguiente procedimiento.

- a) Instalar Nvidia CUDA Toolkit
- b) Registrarse gratuitamente en Nvidia Developers
- c) Desde la página de Nvidia cuDNN, escoger el archivo instalador más reciente y adecuado según el sistema operativo en formato \*.tgz
- d) Descomprimir el archivo y copiar los archivos en la dirección donde se ha instalado Nvidia CUDA Toolkit

## 3. Bazel

Es una herramienta que permite la compilación de programas de software libre a partir del código fuente. En esta ocasión se usa para re-compilar TensorFlow. Para instalarlo de siguen los siguientes pasos:

- **Descargar el instalador.** Varias versiones del instalador están disponibles en el repositorio del proyecto Bazel en Github. Se recomienda descargar la versión 0.19.0 ya que esta no presenta problemas de compatibilidad y no causará errores durante la compilación.

- **Convertir el instalador en ejecutable.** Usando el Terminal, convertir el archivo descargado en ejecutable mediante el comando `chmod +x <nombre-del-archivo>.sh`
- **Instalar.** Para instalar Bazel, a partir del ejecutable creado en el paso anterior, ejecutar `./<nombre-del-archivo>.sh`

### Sumario de software preliminar previa la instalación de Tensorflow

Los componentes de software presentados en la tabla 4 son mandatorios para para la instalación de Tensorflow y el desarrollo de aplicaciones con redes neuronales aceleradas por GPUs.

**Tabla 4**

*Software preliminar requerido para instalar TensorFlow*

	Descripción	Tipo
1	Nvidia CUDA Toolkit	Plataforma de aceleración con GPUs
2	Nvidia cuDNN	Librería de primitivas para redes neuronales
3	Bazel	Compilador de software libre

#### 4.2.3.2. Re-compilación e instalación

Los instaladores de Tensorflow disponibles en línea solo son compatibles con versiones de Nvidia CUDA 3.5 y superiores. Por esta razón, para que pueda ejecutarse en este ordenador, se debe re-compilar el instalador de Tensorflow para permitir compatibilidad con Nvidia CUDA 3.0.

Para re-compilar Tensorflow realizamos el siguiente procedimiento:

1. Descargar la última versión del código fuente de Tensorflow que puede ser encontrado en el repositorio de Github. Se creará una carpeta con el nombre Tensorflow.
2. Abrir el terminal y cambiar el directorio raíz a la carpeta Tensorflow que se creó en el paso anterior

3. Iniciar la configuración pre-ensamble de Tensorflow. En este paso se selecciona CUDA 3.0 como mínimo compatible con Tensorflow GPU y también se cambia el modo de ejecución de sentencias del CPU a A
4. Ajustar los parámetros de configuración según el anexo 1.
5. Compilar TensorFlow con Bazel. En el ordenador especificado, este proceso se completó en aproximadamente 5 horas.
6. Instalar Tensorflow con el instalador generado en el paso anterior.
7. Configurar Tensorflow con Python
8. Verificar la instalación

## 4.3. Implementación de la RCN

### 4.3.1. Mapas de segmentación binaria

En esta etapa se clasifican las líneas de carretera de cualquier otro objeto en la imagen, asignando a los píxeles de las líneas con color blanco y todo lo demás con color negro. Numéricamente, esta clasificación está dada por valores RGB expresados en la tabla 5.

**Tabla 5**

*Valores RGB asignados a la segmentación binaria*

Color	R	G	B	Descripción
Blanco	255	255	255	Líneas de carretera
Negro	0	0	0	Cualquier objeto que no forme parte de la línea

Usando la herramienta para edición de imágenes gratuita GIMP se crean las máscaras de segmentación binaria a partir de las imágenes obtenidas de la carretera con el siguiente procedimiento:

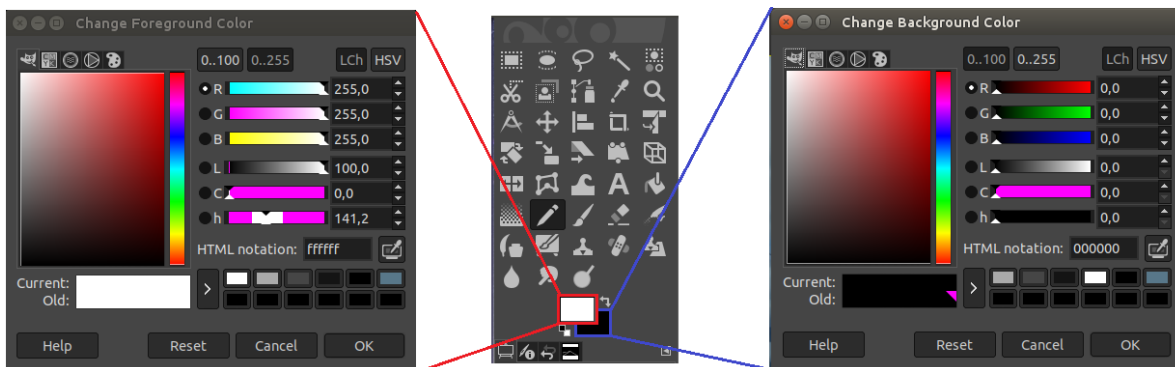


1. Inicializar GIMP y abrir la imagen de la carretera con la cual se creará la máscara binaria haciendo clic en *Abrir* en el menú *Archivo*. Para este ejemplo, se usará la imagen de la figura 11.



**Figura 11.** Imagen de carretera con la cual se generará una máscara binaria

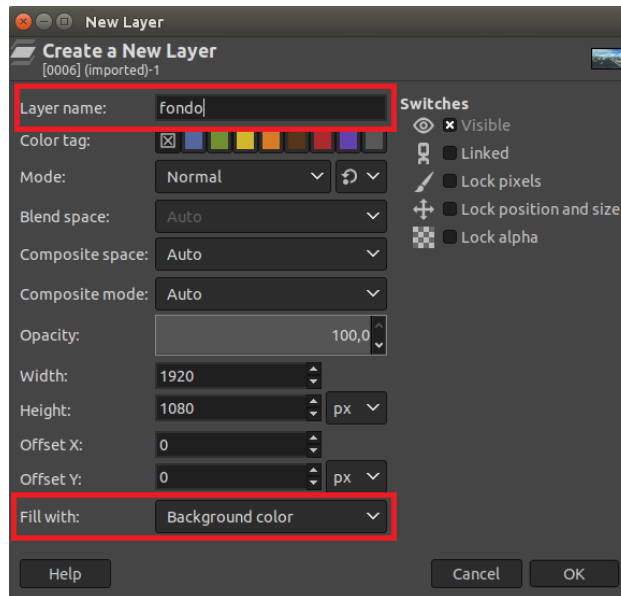
2. Al lado izquierdo, configurar el *Color de fondo* a negro; es decir, RGB (0,0,0); y el *Color de frente* a blanco; es decir, RGB (255,255,255); como se muestra en la figura 12.



**Figura 12.** Configuración del color de fondo y frente en GIMP

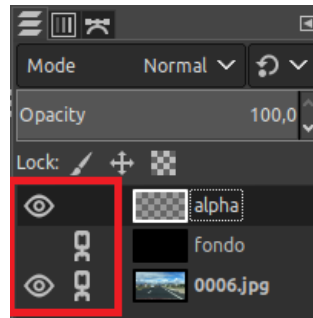
3. Presionar *Shift + Ctrl + N* para crear una nueva capa. Aparecerá un recuadro de configuraciones; en la última opción, *Rellenar con*, seleccionamos *Color de fondo*. Además, asignar

un nombre a la capa en la primera opción, en este caso *fondo*. Dejar los valores por defecto para las demás propiedades. Ver figura 13.



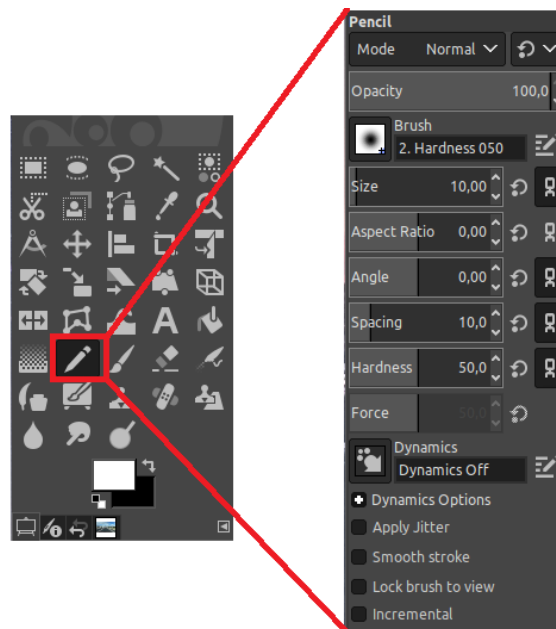
**Figura 13.** Configuración de propiedades para la creación de una nueva capa

4. Repetir el paso anterior para crear una nueva capa. En este caso, seleccionamos *Transparencia* en la opción de *Rellenar con*. El nombre de esta capa será *alpha*.
5. En el menú de capas, al lado inferior derecho, activar la visibilidad y bloquear la capa de la imagen original, activar la visibilidad de la capa *alpha* y bloquear la capa *fondo* como se muestra en la figura 14.



**Figura 14.** Opciones de visibilidad y bloqueo de las capas previo al subrayado de las líneas.

6. Al lado izquierdo, seleccionar la herramienta lápiz y configurar sus propiedades con los valores de la figura 15.



**Figura 15.** Configuración de la herramienta lápiz

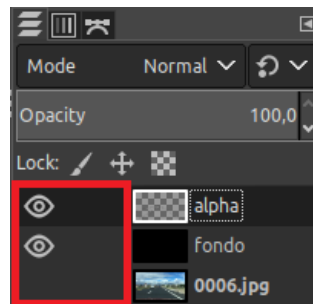
7. Verificar que la capa activa es *alpha*. Con la herramienta lápiz subrayar todas las líneas de la carretera. Para que la línea sea recta, hacer clic en el punto inicial de la línea y mantener

presionado *shift* hasta el punto final de la misma. En la figura 16 se muestra el resultado de este paso.



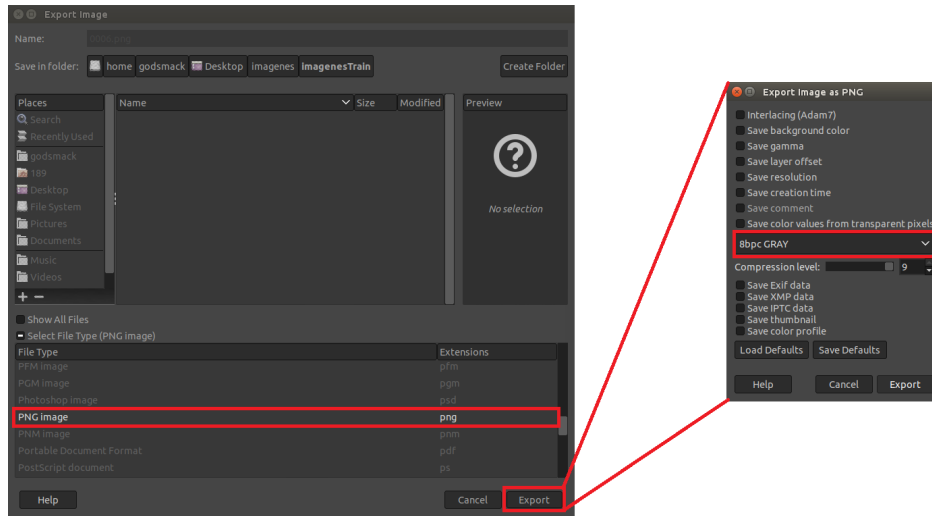
**Figura 16.** Subrayado de las líneas de carretera

8. Una vez subrayadas todas las líneas de carretera, en el menú de capas, desbloquear todas las capas, activar la visibilidad de las capas *alpha* y *fondo* y desactivar la visibilidad de la capa de la imagen original como se ilustra en la figura 17.



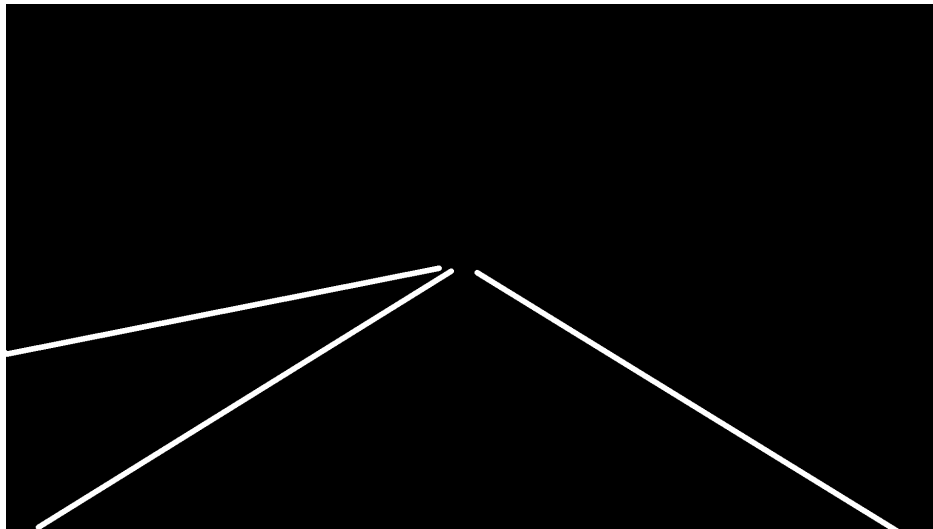
**Figura 17.** Opciones de visibilidad y bloqueo de las capas previo a la exportación de la máscara

9. Finalmente, para exportar la máscara binaria, hacer clic en la opción *Exportar* del menú *Archivo*. Seleccionar el formato de salida de la imagen como PNG, aparecerá una ventana emergente y cambiar solamente el formato de compresión a *8bpc GRAY*. Ver figura 18.



**Figura 18.** Configuración de parámetros para la exportación de la máscara

Entonces se obtiene el mapa de segmentación ilustrado en la figura 19 que servirá como ejemplo de entrenamiento para la red neuronal.



**Figura 19.** Mapa de segmentación de las líneas de carretera.

En la segmentación binaria, se identifican e incluso predicen los píxeles de las líneas de carretera incluso si existen obstáculos a lo largo de la trayectoria, como vehículos, y señalización vertical, o incluso si la línea está des pintada presentando discontinuidades. Con esto, se logra que la red sea robusta, incluso ante la presencia de perturbaciones.

## 4.4. Segmentación binaria con una red VGG16 en Tensorflow

### 4.4.1. Aspectos más importantes de la arquitectura desarrollada

- La segmentación binaria se realiza en base a los pesos de la red pre-entrenada VGG16 disponibles en línea.
- Las capas finales de la red VGG16, que son capas completamente conectadas se transforman en capas convolucionales usando filtros o kernels de dimensión  $1 \times 1$ .
- La generación de la máscara de segmentación o etapa de decodificación se realiza mediante filtros de convolución transpuestos.
- En cada etapa, la decodificación se realiza mediante los mapas de características de la etapa de codificación de la red VGG16.

### 4.4.2. Desarrollo de la arquitectura

La segmentación binaria está compuesta por dos sub-procesos: codificador y decodificador.

#### 4.4.2.1. Codificador

El codificador toma la imagen ingresada como un arreglo de dimensiones  $512 \times 256 \times 3$  cuyos píxeles deben estar normalizados en el rango  $0 - 1$ . Después, de la capa 1 a la 12, a lo largo de todo el arreglo, se ejecutan convoluciones con los kernels para obtener  $n$  mapas de características de

la imagen y cada mapa incrementará la tercera dimensión del arreglo. Por ejemplo, en la primera convolución donde se generan 64 mapas de características de la imagen original, la dimensión del arreglo resultante es  $512 \times 256 \times 64$ . Los filtros de convolución tienen dimensión  $3 \times 3$  y, inicialmente (previo al entrenamiento), sus elementos son los pesos ( $w$  y  $b$ ) definidos por el modelo pre-entrenado VGG16. Cabe recalcar, que todo el proceso es recursivo es decir, la capa 1 toma como entrada la imagen original, la capa 2, el resultado de la capa 1, la capa 3, el resultado de la capa 2 y así sucesivamente.

Matemáticamente, el resultado  $C$  de la convolución de una imagen  $I$  a través de un kernel  $K$  es:

$$R = I * K$$

En Tensorflow, para la implementación de esta tarea, se usa la función `conv2d()` que recibe como parámetros al arreglo (la imagen), la dimensión del kernel o filtro, las dimensiones de salida (número de mapas de características) y los pesos  $w$  y desviaciones  $b$ . Cabe recalcar que después de la convolución variara solo la tercera dimensión del arreglo, donde cada una representa una característica en particular de la imagen (Ver tabla 1).

---

```
tensorflow.conv2d(arreglo, dimensiones_salida, dimension_kernel, w, b)
```

---

Posteriormente, cada determinado número de convoluciones (según la tabla 1), se realiza el proceso de reducción de dimensiones de superficie del arreglo con Max-Pooling para lo que se usa la función `max_pool()` cuyos parámetros son el arreglo de entrada, la dimensión de la ventana y el paso del desplazamiento.

---

```
tensorflow.max_pool(arreglo, dimension_ventana, paso)
```

---

Se debe recordar que durante la etapa de convoluciones no todas la neuronas están conectadas entre si; es decir, no se procesan todos los píxeles si no los mas representativos de acuerdo a los mapas de características.

Para pasar de la capa 12 a la 13, primero, la capa 12 se transforma un arreglo unidimensional; es decir, el arreglo de la capa 12 se re-ordena de forma lineal. De esta forma, los arreglos de las capas 12 y 13 tienen el mismo numero de elementos pero diferente dimensión como se ilustra en la tabla 6

**Tabla 6***Comparación de los arreglos de la capa 12 y 13*

Capa	Dim. del arreglo	No. elementos
12	$16 \times 8 \times 512$	65536
13	$1 \times 65536$	65536

Posteriormente, se realiza una conexión densa es decir, se conecta todas la neuronas entre sí. Para esto se usa la función de Tensorflow *tf.layer.dense()* cuyos parámetros son el arreglo de entrada, el tipo de función de activación (Ver tabla 1), los pesos ( $w$ ) y las desviaciones iniciales ( $b$ ).

---

```
tf.layers.dense(arreglo, activacion='relu', w, b)
```

---

Solo en la capa 16 se usa la función de activación Softmax que es una distribución de probabilidad que asigna a cada píxel la clase más probable a la que pertenece; es decir, calcula si un píxel es parte de la línea de carretera o no.

Sumarizando, la secuencia de convoluciones y reducciones por Max-Pooling es la que se presenta en el diagrama de flujo de la figura 20.



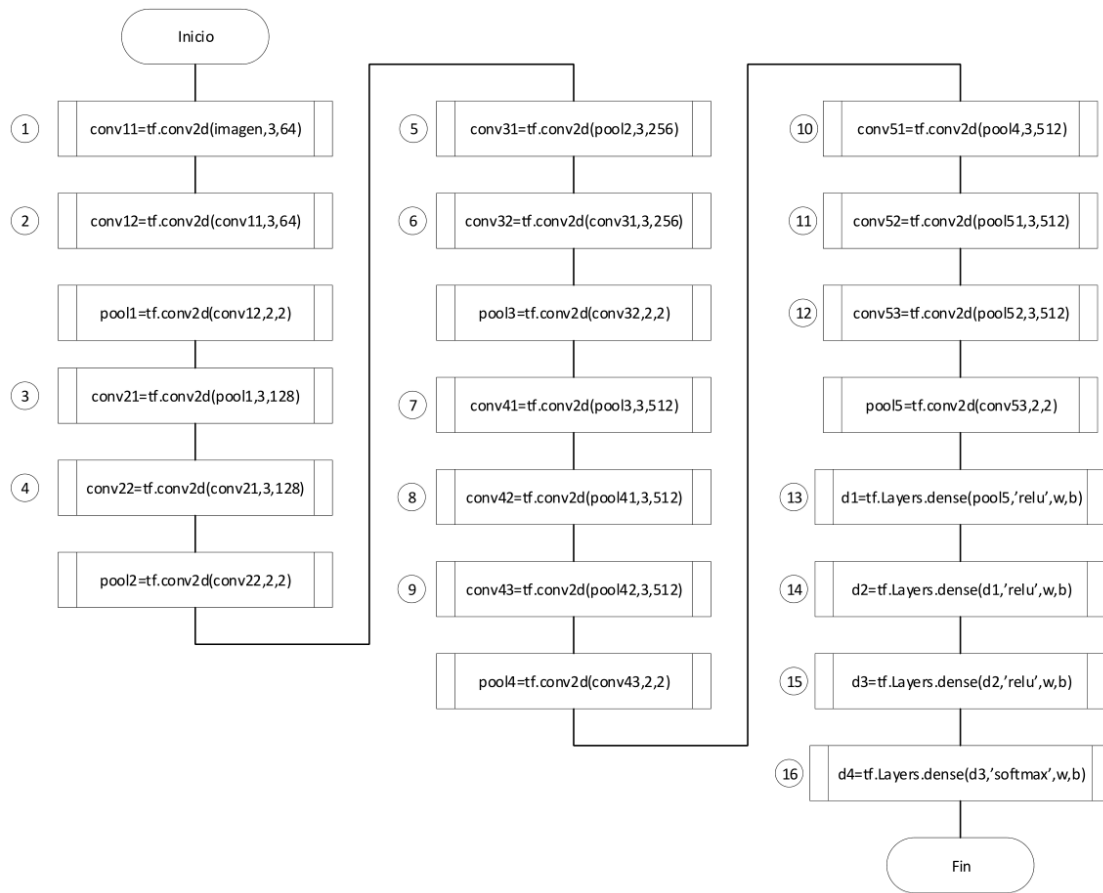


Figura 20. Secuencia de realización de convoluciones y reducciones en el codificador.

#### 4.4.2.2. Decodificador

El decodificador realiza, prácticamente, el mismo procedimiento del codificador pero en orden invertido. Entonces, el decodificador recoge el arreglo lineal de generado en el codificador y genera la máscara de segmentación. Para el efecto, se deben realizar dos operaciones: la des-convolución y la dilatación del arreglo en orden inverso al del codificador. La des-convolución es el resultado de realizar el mismo procedimiento de convolución del codificador pero con el inverso de los filtros; es decir, el resultado de una des-convolución  $D$  de un arreglo  $I$  cuyo kernel de codificación fue  $K$  es:

$$D = I * K^{-1}$$

Por su parte, la dilatación del arreglo se realiza mediante la operación un-pooling. Tensorflow integra la función *deconv2d()* que además de realizar el proceso de des-convolución permite dilatar el arreglo mediante un-pooling.

---

```
funcion decodificar(entrada, lista_descodificada):
    tensor_entrada = entrada[lista_descodificada[0]]['data']
    ganancia = tensorflow.conv2d(datos_tensor=tensor_entrada, canales_salida=64,
    kernel_dim=1)
    lista_descodificada = lista_descodificada[1:]
    Para i en rango(len(lista_descodificada)):
        deconv = tensorflow.deconv2d(datos_tensor=ganancia, canales_salida=64,
        kernel_dim=4, paso=2)
        tensor_entrada = entrada[lista_descodificada[i]]
        ganancia = tensorflow.conv2d(datos_tensor=tensor_entrada, canales_salida=64,
        kernel_dim=1)
        combinado = tensorflow.sumar(deconv, ganancia)
        ganancia = combinado

    deconv_final = self.deconv2d(datos_tensor=ganancia, canales_salida=64,
    kernel_dim=16, paso=8)
    ganancia_final = self.conv2d(datos_tensor=deconv_final, canales_salida=2,
    kernel_dim=1,)

    ret['ganancia'] = ganancia_final
    ret['deconv'] = deconv_final

    devolver ret
```

---

El resultado de procesar la imagen de una carretera con la re neuronal VGG16 se presenta en la figura 21.



*Figura 21.* Abajo: Máscara generada por la red neuronal de la imagen de la carretera superior

#### 4.4.3. Generación de modelos matemáticos de la líneas de carretera

La máscara generada por la RNC entrega como resultado las líneas de carretera marcadas en color blanco; sin embargo, no existen variables que permitan analizar la pérdida de vía. Para solucionar esto, se deben crear modelos matemáticos de las líneas del tipo  $y = mx + b$  a fin de analizar

el comportamiento de las pendientes  $m$  cuando el vehículo pierde vía y cuando no. Entonces, para generar los modelos matemáticos se ejecuta lo siguiente:

### 1. Detector de bordes de Canny

Las líneas generadas en la máscara (Figura 21) son gruesas y carecen de suavidad; con lo cual, procesadas directamente de esta forma generarían errores en los pasos posteriores. Entonces, se aplica el detector de bordes Canny para detectar solo los bordes de las líneas, los cuales son mas suaves y permiten aplicar la Transformada de Hough con mayor eficacia. El resultado de aplicar este algoritmo es el de la figura 22



**Figura 22.** Detección de bordes mediante el algoritmo de Canny

### 2. Transformada de Hough

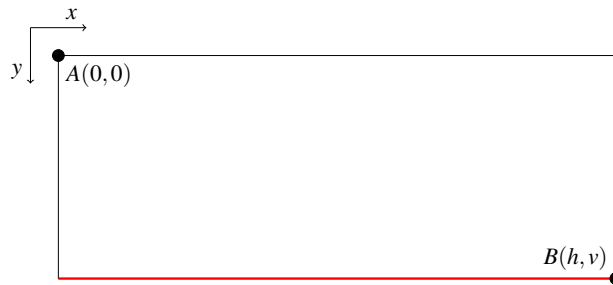
La transformada de Hough permite generar modelos del tipo  $y = mx + b$  de las líneas de una imagen. A través de la librería OpenCV, aplicamos este algoritmo con la función `houghP()` que entrega como resultado, por cada línea detectada en la imagen, los dos puntos de la recta que más se ajusten a la línea. Utilizando estos puntos se puede dibujar las líneas correspondientes a cada uno y el resultado se ilustra en la figura 23



**Figura 23.** Líneas detectadas mediante la transformada de Hough

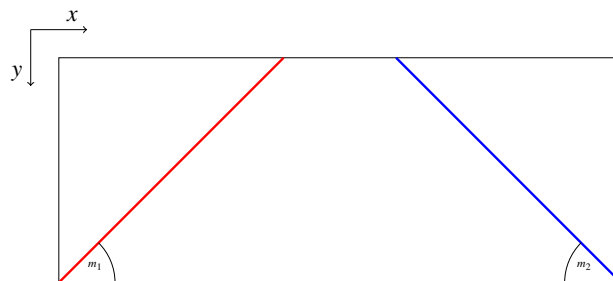
### 3. Clasificador de líneas

El detector de bordes de Canny tiende a generar más de un borde en una misma línea; por ejemplo, el borde exterior, interior y hasta bordes intermedios. Con lo cual, la transformada de Hough también genera más de una recta por línea de carretera como se aprecia en la figura 23. Lo ideal es que solo se genere una sola recta por cada línea de carretera, y a su vez que solo se muestren la líneas más cercanas a la perspectiva de la cámara del vehículo; es decir, una recta de límite izquierdo y una de límite derecho del carril. Para solucionar todo esto, se ha creado una función de clasificación de líneas, que compara todo el conjunto de rectas que se han generado en el paso anterior y se retornan tan solo las dos que más se ajusten a los límites del carril. Para realizar este algoritmo, primero, se debe tener en cuenta el sistema de referencia al que se somete una imagen, el cual se ilustra en la figura 24. Entonces, de acuerdo a la figura, si una línea está en correcta perspectiva, uno de sus puntos debe estar contenido en la recta del borde inferior de la imagen la cual se subraya en color rojo. Matemáticamente, siendo la recta  $f(x) = mx + b$ , si  $v = mx + b$  con  $0 < x < h$ , entonces  $f(x)$  está dentro de la perspectiva correcta.



**Figura 24.** Sistema de referencia de una imagen.  $h$  y  $v$  son las dimensiones horizontal y vertical respectivamente

Adicionalmente, después de detectada la primera línea, la segunda, además de estar en correcta perspectiva, debe tener una pendiente  $m$  diferente a la de la primera. En el caso ideal, como se ilustra en 25, ( $m_2 = -m_1$ ) con lo cual  $m_1 + m_2 = 0$ , sin embargo, en la vida real, debido a la variación del posicionamiento de la cámara esta condición casi nunca se cumple. Por lo que, se ha determinado que el umbral de  $m_1 + m_2 < 0.4$  es el más adecuado para decidir si la línea es diferente a la anterior.



**Figura 25.** Líneas de carretera en el sistema de referencia: En el caso ideal  $m_1 = -m_2$

El diagrama de flujo conjunto de estos dos algoritmos se presenta en la figura 26

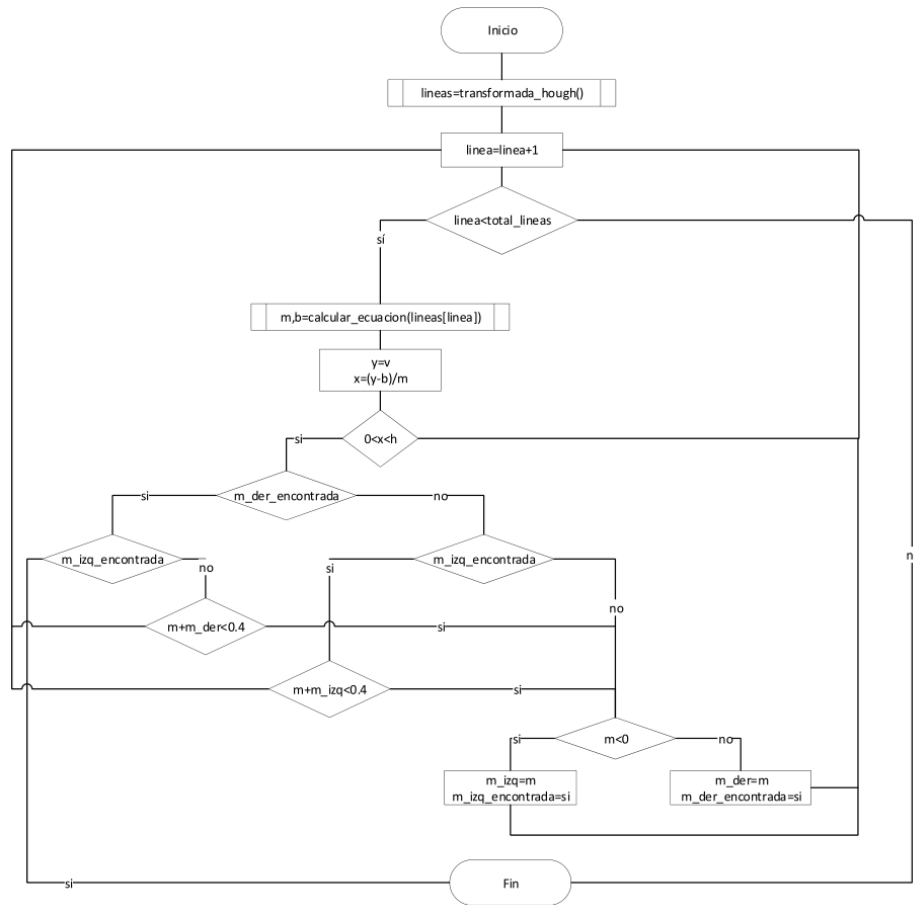


Figura 26. Diagrama de flujo del algoritmo de clasificación de líneas

Aplicando estos dos algoritmos de clasificación, la imagen resultante se ilustra en la figura 27



*Figura 27.* Líneas de carretera detectadas y clasificadas

#### 4.4.4. Filtración en tiempo real

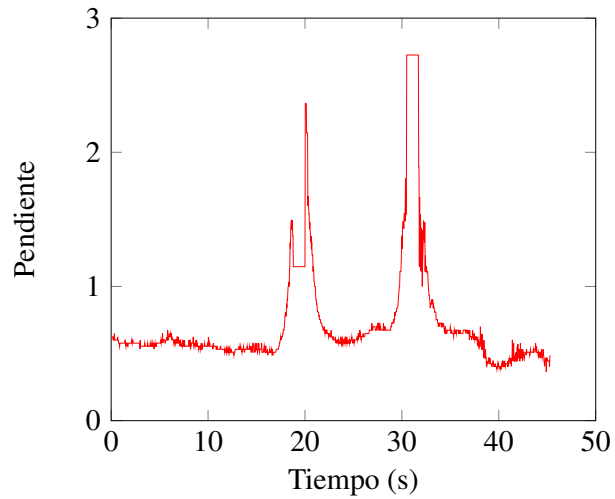
Si se analiza el comportamiento de las pendientes  $m$  de las líneas de carretera con respecto al tiempo ( $t$ ) se aprecia una cantidad considerable de ruido especialmente en situaciones de poca luz. La presencia de ruido provoca una mayor probabilidad de falsos positivos y negativos al momento de detectar la pérdida de vía. Para reducir el ruido es necesario implementar un filtro cuyo retraso sea mínimo, su procesamiento sea rápido y que pueda funcionar en tiempo real. Después de la experimentación con diferentes tipos de filtros, entre ellos filtros FIR e IIR, se llegó a la conclusión que el filtro pasabajos Butterworth de orden 2 es el que más se ajusta las necesidades de este proyecto ya que reduce notoriamente el ruido, tiene un retraso mínimo ( $< 30ms$ ) y funciona en tiempo real.

##### 4.4.4.1. Diseño del filtro pasabajos Butterworth de segundo orden

Teniendo en cuenta la función de transferencia de un filtro digital pasabajos Butterworth de segundo orden dado por la ecuación 3.10 podemos diseñar el filtro en base al comportamiento de las pendientes respecto al tiempo. Entonces, analizando la figura 28, la frecuencia de muestreo de la señal es  $f_s = 30$  debido a que en 1 segundo se procesan 30 imágenes. Por otro lado, para

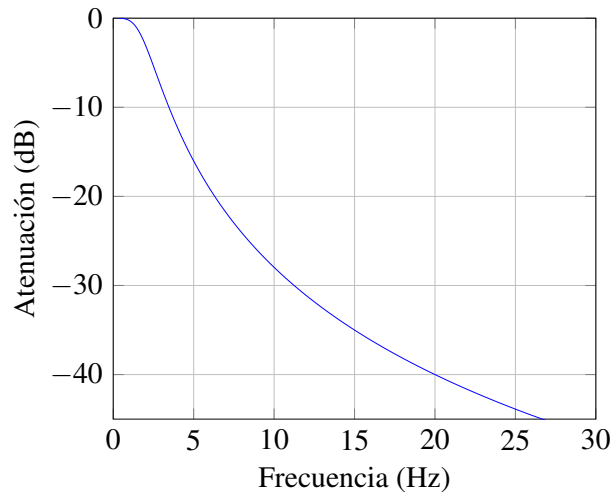


calcular la frecuencia de corte  $\omega_c$ , analizamos la frecuencia del ruido y se deduce que esta a la misma frecuencia de muestreo de la señal; es decir,  $\omega_c = 30$ .



**Figura 28.** Comportamiento de la pendiente de la línea derecha respecto al tiempo sin filtración de ruido

El comportamiento del filtro Butterworth pasabajos de orden 2 y con frecuencia de corte  $\omega = 30$  se ilustra en la figura 29 donde se puede apreciar que en la banda de paso no existe atenuación y en la banda de corte la atenuación es constante sin la presencia de saltos bruscos y ocurre a la misma frecuencia a la que ocurre el ruido.



**Figura 29.** Comportamiento del filtro Butterworth pasabajos

#### 4.4.5. Detección de pérdida de vía

Para la detección de pérdida de vía, analizamos el comportamiento de las pendientes. En el caso ideal, cuando la cámara está correctamente alineada con la carretera, ocurre lo que se detalla en la tabla 7.

**Tabla 7**

*Comportamiento de las pendientes en el caso ideal.*

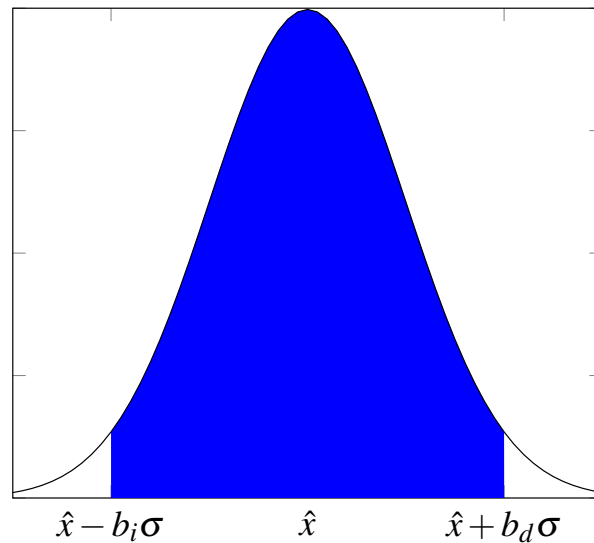
Caso	Comportamiento de pendientes
Centrado	$m1 + m2 = 0$
Desv. izquierda	$m1 + m2 < 0$
Desv. derecha	$m1 + m2 > 0$

Sin embargo, ya que es muy poco probable que la cámara esté perfectamente alineada con la carretera, se debe corregir ese error calculando un valor umbral o de compensación hacia la izquierda  $\delta_i$  y hacia la derecha  $\delta_d$  que determine cuando la pérdida de vía ocurre como se detalla en la tabla 8.

**Tabla 8***Comportamiento de las pendientes con compensación por variación de la posición de la cámara*

Caso	Comportamiento de pendientes
Centrado	$-\delta_i < m1 + m2 < \delta_d$
Desv. izquierda	$m1 + m2 < -\delta_i$
Desv. derecha	$m1 + m2 > \delta_d$

Para calcular  $\delta_i$  y  $\delta_d$  se requiere un tiempo de calibración  $T$  en el cual el conductor deberá conducir centrado a la carretera. En dicho periodo de tiempo se recolecta el valor de  $m_i + m_d$  en cada cuadro del vídeo. Cuando el tiempo de recolección de datos ha terminado, se realiza el análisis del comportamiento de los datos mediante un histograma y se concluye que los datos de  $m_i + m_d$  cuando el vehículo está centrado a la carretera tienen una distribución de Gauss como se ilustra en la figura 30. Con lo cual, se calcula la media  $\hat{x}$  y la desviación estándar  $\sigma$  de  $m_i + m_d$ . para determinar cuando el vehículo entra en pérdida de vía. Entonces,  $\delta_i = \hat{x} - b_i\sigma$  y  $\delta_d = \hat{x} + b_d\sigma$  donde a través de la experimentación se ha determinado que los valores  $b_i = 2$  y  $b_d = 3$  son los adecuados para detectar cuando el vehículo está en pérdida de vía. En 30, la región pintada de azul representa cuando el vehículo está centrado y en línea con la carretera cuando  $m_i + m_d$ , fuera de estos límites se interpreta como pérdida de vía hacia la izquierda o derecha respectivamente.



**Figura 30.** Los datos de  $m_i + m_d$  obedecen a una distribución de Gauss. La región en azul representa que el auto está centrado en la vía.

De esta forma, cuando las líneas que marcan la carretera (Ver Figura 27) están en color azul, significa que el sistema se está calibrando y por ende el sistema de detección de pérdida de vía no está funcionando aún. Cuando, ambas líneas están en color verde el sistema está funcionando y el vehículo circula correctamente; y cuando alguna línea se pinta en color rojo a manera de alerta, significa que existe pérdida de vía.

# Capítulo 5

## Análisis de resultados

En este capítulo mediante diversas métricas se evaluó el funcionamiento y robustez del sistema implementado en este proyecto.

### 5.1. Métricas de evaluación

#### 5.1.1. Función de pérdida por entropía cruzada

Para realizar la segmentación binaria se usa la función de pérdida de entropía cruzada. Esta función le asigna a cada píxel una probabilidad de pertenecer a una clase en particular; en este caso, existen dos clases: la línea de carretera que se representa por 1 y todo lo que no tiene que ver con la línea que se representa con 0. En otras palabras, la función de pérdida por entropía cruzada, en este trabajo, predice si un píxel es parte de la línea de carretera o no. Matemáticamente, la función de pérdida de entropía cruzada  $H_p(q)$  está definida por la ecuación 5.1

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \times \log(p(y_i)) + (1 - y_i) \times \log(1 - p(y_i)) \quad (5.1)$$

donde  $y$  es una de la dos clases; es decir, 1 para la línea de carretera y 0 para cualquier otro objeto que no sea la línea.  $p(y_i)$  es la probabilidad de que el píxel pertenezca a la línea de carretera y  $N$  es el número de píxeles de la imagen.

### 5.1.2. Sensibilidad y especificidad de un sistema

Para determinar la sensibilidad y especificidad de un sistema es necesario conocer los siguientes conceptos:

1. **Verdadero positivo (VP).**- Un verdadero positivo ocurre cuando el sistema emite una alerta cuando efectivamente el auto ha entrado en pérdida de vía.
2. **Verdadero negativo (VN).**- Un verdadero negativo se produce cuando el sistema no ha emitido ninguna alerta y el vehículo esta circulando correctamente en la vía.
3. **Falso positivo (FP).**- Un falso positivo ocurre cuando, el sistema emite una alerta a pesar de que el vehículo circula correctamente en la vía.
4. **Falso negativo (FN).**- Un falso negativo se da cuando el vehículo ha entrado en pérdida de vía y no se ha emitido ninguna alerta.

La sensibilidad  $S$  es la probabilidad de que el sistema detecte cuando el vehículo entre en pérdida de vía y esta dada por la ecuación 5.2.

$$S(\%) = \frac{VP}{VP + FN} \times 100\% \quad (5.2)$$

Por otro lado, la especificidad  $E$  es la probabilidad de que el sistema no emita ninguna alerta cuando el vehículo circula correctamente y esta dada por la ecuación 5.3.

$$E(\%) = \frac{VN}{VN + FP} \times 100\% \quad (5.3)$$

## 5.2. Evaluación del sistema

### 5.2.1. Descripción de escenarios de evaluación

Para determinar la eficiencia del proyecto, se ha probado el mismo en cinco carreteras diferentes, cada una, en condiciones de luz diurna, nocturna y un escenario con lluvia como se especifica en la tabla 9, dando como resultado 11 escenarios de prueba diferentes. De los cuales, se han tomado 5 imágenes de video aleatoriamente, por cada uno, para realizar las pruebas que se presentan a continuación

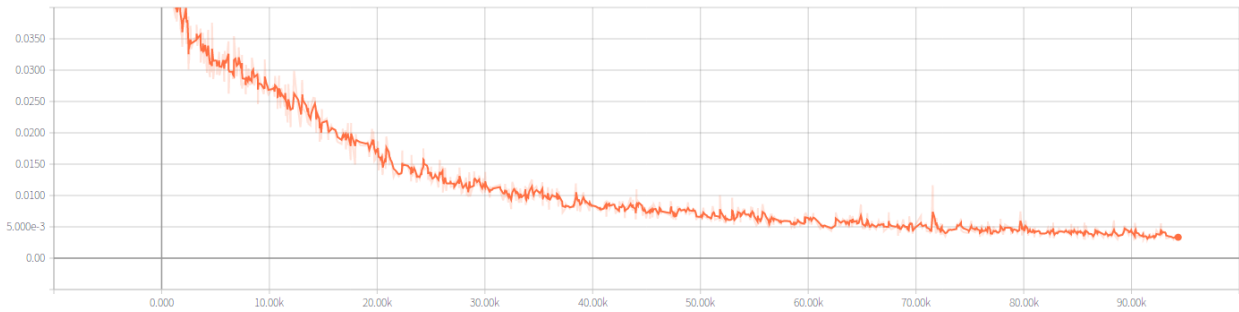
**Tabla 9**

*Descripción de los escenarios de prueba*

Descripción	Día (A)	Noche (B)	Lluvia (C)
1 Autopista de doble carril	×	×	
2 Autopista de un carril	×	×	×
3 Calle rural de un carril	×	×	
4 Avenida urbana de doble carril	×	×	
5 Avenida rural de un carril	×	×	

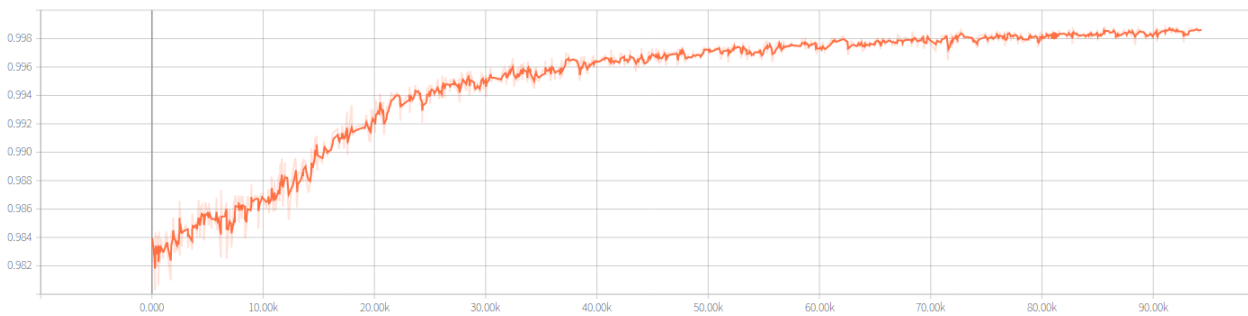
### 5.2.2. Eficiencia del aprendizaje de la red neuronal

Con la función de pérdida por entropía cruzada se realiza el análisis del error en cada época de entrenamiento de la red neuronal. En la figura 31 se ilustra el comportamiento del error a lo largo de 100,000 épocas de entrenamiento de la red neuronal. Al inicio del entrenamiento para este proyecto, se usó los pesos de red neuronal VGG16 disponibles en línea, los cuales generaban un error salida-entrada mayor a 3, al final del entrenamiento con los ejemplos generados específicamente para este proyecto. el error fue estable de alrededor de  $5 \times 10^{-3}$ .



**Figura 31.** Variación del error a lo largo del entrenamiento de la red neuronal.

A su vez, la exactitud del sistema se calcula comprando los píxeles del ejemplo de entrenamiento con respecto a los píxeles del resultado generado; con lo cual el comportamiento de la exactitud del sistema a lo largo del entrenamiento fue el que se ilustra en la 32



**Figura 32.** Variación de la exactitud del sistema.

En la figura 33, se ilustran los resultados obtenidos de clasificar los píxeles de la líneas de carretera mediante la red neuronal.





**Figura 33.** Resultados generados por la red neuronal. Cada fila es un escenario diferente. La segunda columna es la máscara generada por la red neuronal, la tercera es el resultado final (en calibración) y la cuarta es el sistema detectando cuando el vehículo pierde la trayectoria correcta

Para determinar la efectividad de la clasificación de los píxeles de las líneas numéricamente se ha comparado cada píxel de las máscara generada a mano con su respectivo píxel de la máscara generada por la red neuronal. Con lo cual se obtuvieron los resultados de las tablas 10-20, donde  $x_a$  es el numero de píxeles acertados o iguales entre las imágenes generadas a mano y mediante la red neuronal y  $x_b$  el numero de píxeles diferentes o erróneos. El error se ha calculado como  $err = x_b/N$ , siendo  $N$  el numero total de píxeles de la imagen.

**Tabla 10***Eficiencia de la clasificación de las líneas de carretera del escenario 1A*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2042245	31355	98.488
2	2044055	29545	98.575
3	2041729	31871	98.463
4	2041649	31951	98.459
5	2044747	28853	98.609

**Tabla 11***Eficiencia de la clasificación de las líneas de carretera del escenario 1B*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2037979	35621	98.282
2	2047339	26261	98.734
3	2044492	29108	98.596
4	2038994	34606	98.331
5	2043305	30295	98.539

**Tabla 12***Eficiencia de la clasificación de las líneas de carretera del escenario 2A*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2039933	33667	98.376
2	2039859	33741	98.373
3	2041004	32596	98.428
4	2043829	29771	98.564
5	2042037	31563	98.478

**Tabla 13***Eficiencia de la clasificación de las líneas de carretera del escenario 2B*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2038252	35348	98.295
2	2039912	33688	98.375
3	2041583	32017	98.456
4	2041346	32254	98.445
5	2044165	29435	98.580

**Tabla 14***Eficiencia de la clasificación de las líneas de carretera del escenario 2C*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2034846	38754	98.131
2	2041088	32512	98.432
3	2039891	33709	98.374
4	2046842	26758	98.710
5	2041708	31892	98.462

**Tabla 15***Eficiencia de la clasificación de las líneas de carretera del escenario 3A*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2046145	27455	98.676
2	2044772	28828	98.610
3	2042294	31306	98.490
4	2041913	31687	98.472
5	2044769	28831	98.610

**Tabla 16***Eficiencia de la clasificación de las líneas de carretera del escenario 3B*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2042527	31073	98.501
2	2046607	26993	98.698
3	2042062	31538	98.479
4	2034459	39141	98.112
5	2045566	28034	98.648

**Tabla 17***Eficiencia de la clasificación de las líneas de carretera del escenario 4A*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2038455	35145	98.305
2	2043720	29880	98.559
3	2037324	36276	98.251
4	2044269	29331	98.586
5	2032942	40658	98.039

**Tabla 18***Eficiencia de la clasificación de las líneas de carretera del escenario 4B*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2044565	29035	98.600
2	2046311	27289	98.684
3	2036844	36756	98.227
4	2039668	33932	98.364
5	2041245	32355	98.440

**Tabla 19***Eficiencia de la clasificación de las líneas de carretera del escenario 5A*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2037517	36083	98.260
2	2040693	32907	98.413
3	2034341	39259	98.107
4	2031941	41659	97.991
5	2041219	32381	98.438

**Tabla 20***Eficiencia de la clasificación de las líneas de carretera del escenario 5B*

Cuadro No.	$x_a$	$x_b$	% exactitud
1	2037740	32837	98.271
2	2037302	32073	98.250
3	2037295	32161	98.249
4	2033158	34247	98.050
5	2033235	31316	98.053

En general, la media de píxeles acertados y erróneos detectados por la red neuronal es 2041300 y 32300, respectivamente. Con lo cual se obtiene una media de exactitud de detección de las líneas del 98.442 %

### 5.2.3. Desempeño del filtro y detección de pérdida de vía

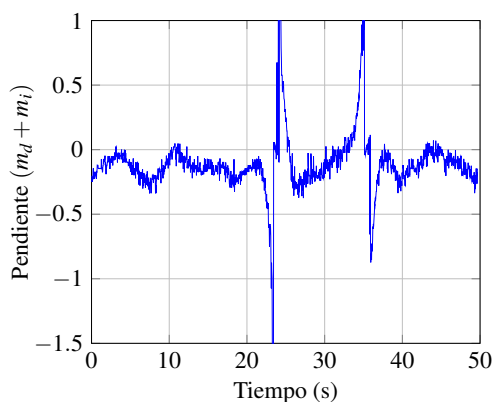
En esta sección se han realizado tres análisis diferentes para demostrar la efectividad del sistema:

1. Comparativa de la señal  $m_i + m_d$  respecto al tiempo antes y después de aplicar el filtro.
2. Comparativa del valor de  $m_i + m_d$  calculado a mano (M), calculado por el sistema sin filtro (N) y calculo por el sistema con filtro, en 5 imágenes aleatorias del video.

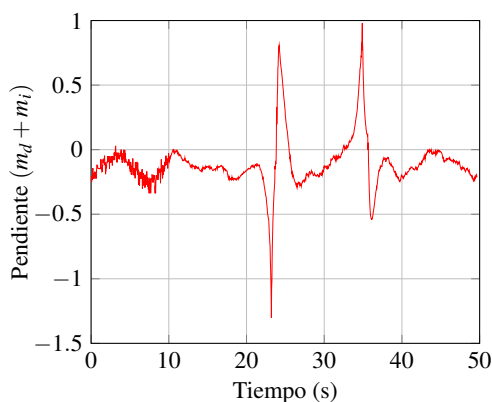
3. Número de falsos positivos y negativos.

### 5.2.3.1. Comparativa de señales

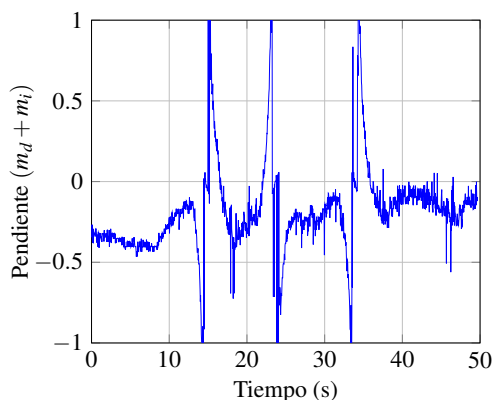
En esta prueba se realiza se compara de la señal  $m_i + m_d$  respecto al tiempo, antes y después de aplicar el filtro en cada uno de los 11 escenarios. Donde se observa que el filtro es capaz de eliminar en tiempo real la mayoría de ruido de la señal.



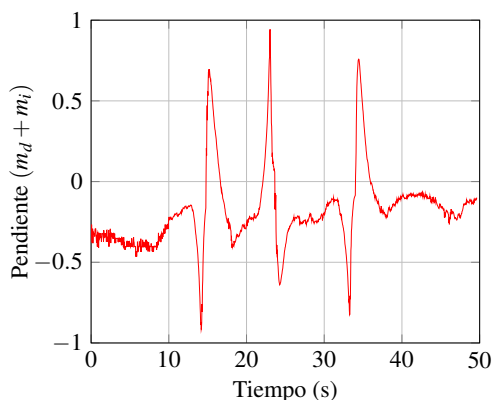
**Figura 34.** Señal sin filtrar del comportamiento de las pendientes del escenario 1A



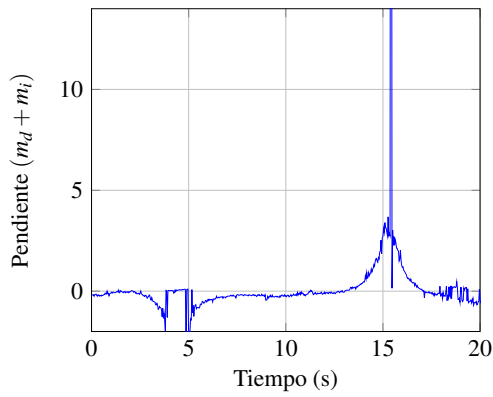
**Figura 35.** Señal filtrada del comportamiento de las pendientes del escenario 1A



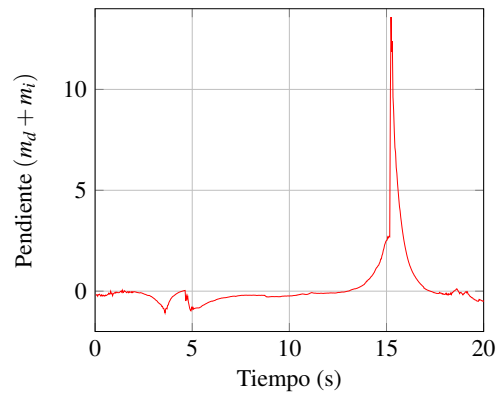
**Figura 36.** Señal sin filtrar del comportamiento de las pendientes del escenario 1B



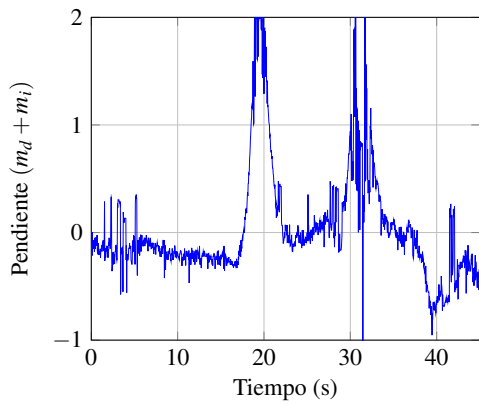
**Figura 37.** Señal filtrada del comportamiento de las pendientes del escenario 1B



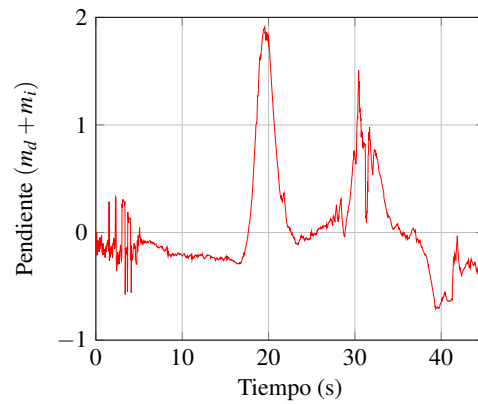
**Figura 38.** Señal sin filtrar del comportamiento de las pendientes del escenario 2A



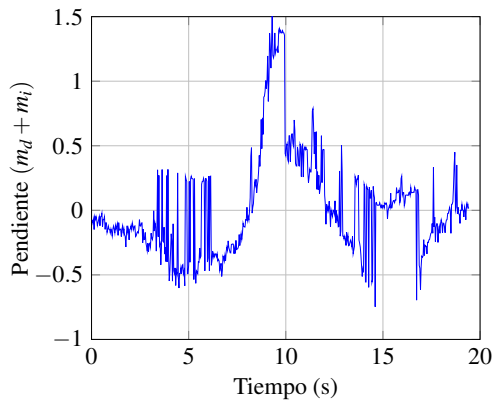
**Figura 39.** Señal filtrada del comportamiento de las pendientes del escenario 2A



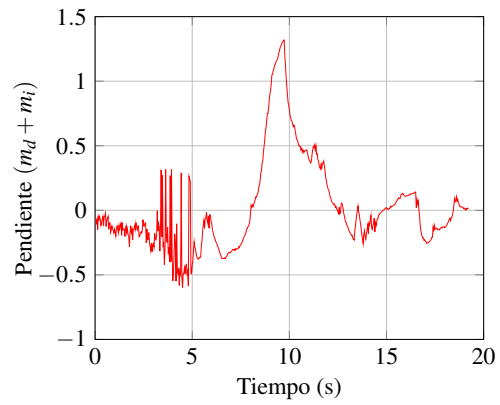
**Figura 40.** Señal sin filtrar del comportamiento de las pendientes del escenario 2B



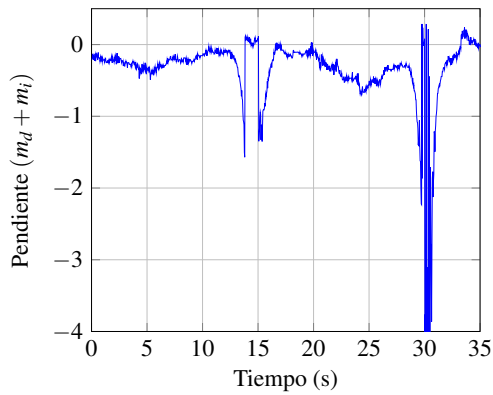
**Figura 41.** Señal filtrada del comportamiento de las pendientes del escenario 2B



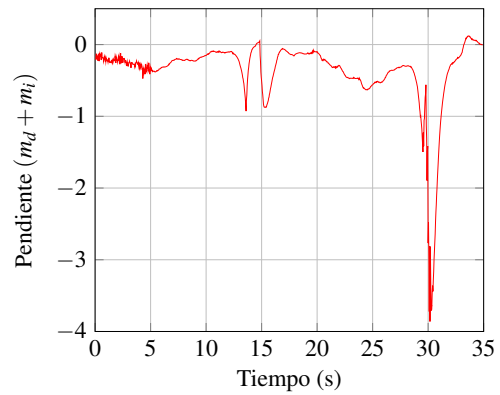
**Figura 42.** Señal sin filtrar del comportamiento de las pendientes del escenario 2C



**Figura 43.** Señal filtrada del comportamiento de las pendientes del escenario 2C

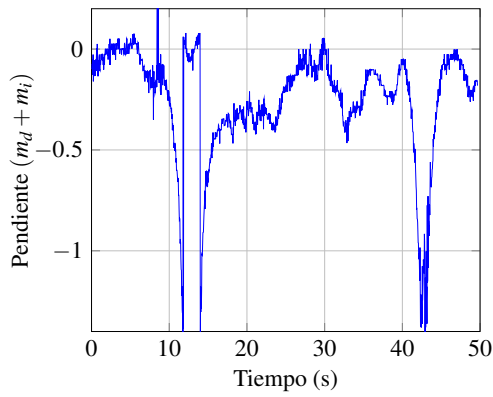


**Figura 44.** Señal sin filtrar del comportamiento de las pendientes del escenario 3A

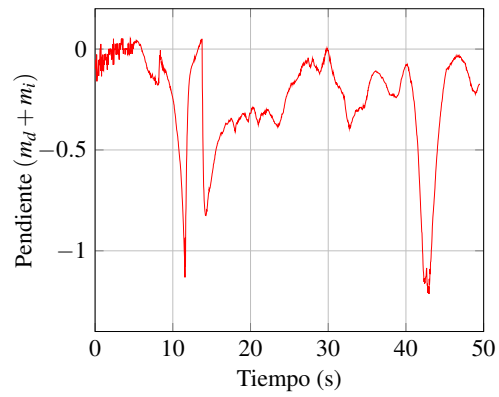


**Figura 45.** Señal filtrada del comportamiento de las pendientes del escenario 3A

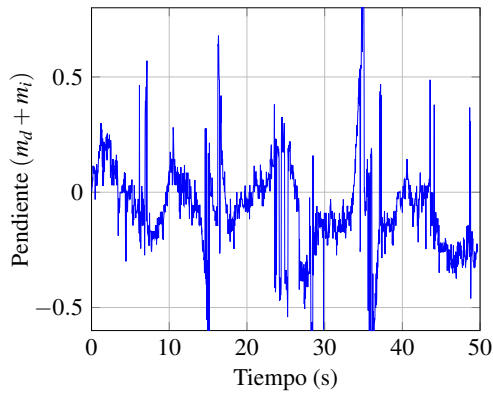




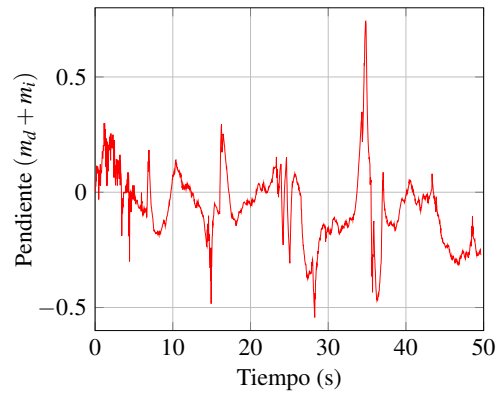
**Figura 46.** Señal sin filtrar del comportamiento de las pendientes del escenario 3B



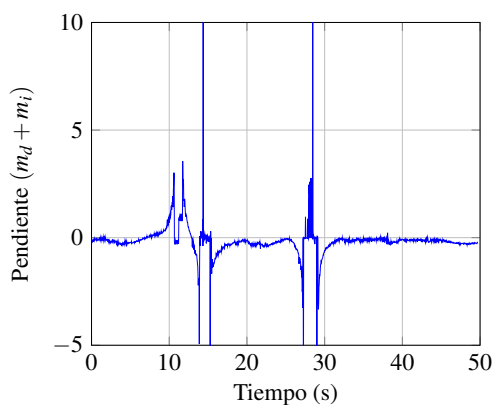
**Figura 47.** Señal filtrada del comportamiento de las pendientes del escenario 3B



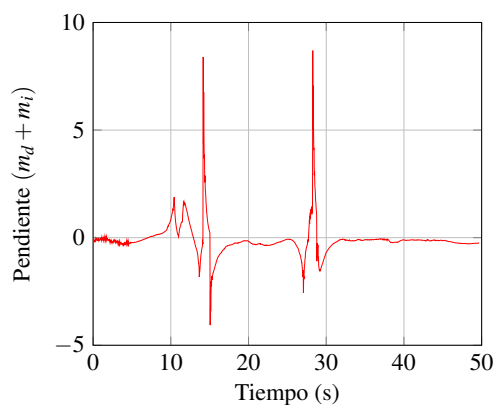
**Figura 48.** Señal sin filtrar del comportamiento de las pendientes del escenario 4B



**Figura 49.** Señal filtrada del comportamiento de las pendientes del escenario 4B



**Figura 50.** Señal sin filtrar del comportamiento de las pendientes del escenario 5A



**Figura 51.** Señal filtrada del comportamiento de las pendientes del escenario 5A

### 5.2.3.2. Comparación cuantitativa de $m_i + m_d$

En esta prueba se compara el valor Comparativa del valor de  $m_i + m_d$  calculado a mano (M), calculado por el sistema sin filtro (SF) y calculado por el sistema con filtro (F), en 5 imágenes aleatorias del video.

**Tabla 21**  
*Comparativa de pendientes del escenario 1A*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
681	0.446	-0.950	0.444	-0.898	0.454	-0.950
702	0.369	-0.893	0.344	-2.900	0.353	-0.896
716	0.792	-0.462	1.000	-0.324	0.776	-0.445
737	0.965	-0.425	0.869	-0.423	0.984	-0.414
738	0.971	-0.409	0.898	-0.404	0.967	-0.418

**Tabla 22***Comparativa de pendientes del escenario 1B*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
408	0.471	-0.847	0.465	-0.901	0.472	-0.856
411	0.466	-0.877	0.486	-0.900	0.457	-0.889
432	0.338	-0.868	0.344	-1.324	0.351	-0.868
452	1.039	-0.397	0.286	-0.363	1.026	-0.407
546	0.460	-0.876	0.554	-0.900	0.464	-0.869

**Tabla 23***Comparativa de pendientes del escenario 2A*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
86	0.557	-0.808	0.554	-0.901	0.567	-0.825
117	0.388	-0.761	0.363	-1.667	0.370	-0.746
155	0.396	-1.234	0.403	-0.324	0.402	-1.246
188	0.484	-0.908	0.508	-0.897	0.489	-0.902
271	0.569	-0.824	0.577	-1.000	0.555	-0.831

**Tabla 24***Comparativa de pendientes del escenario 2B*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
160	0.579	-0.700	0.601	-1.000	0.596	-0.690
258	0.580	-0.795	0.554	-0.901	0.565	-0.778
463	0.544	-0.772	0.576	-0.932	0.540	-0.783
639	0.869	-0.484	0.809	-0.464	0.887	-0.502
910	1.683	-0.605	1.536	-1.000	1.680	-0.611

**Tabla 25***Comparativa de pendientes del escenario 2C*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
152	0.505	-0.806	0.530	-0.932	0.504	-0.810
175	0.545	-0.562	0.531	-1.000	0.528	-0.574
303	0.985	-0.265	0.625	-0.230	0.972	-0.275
313	0.832	-0.293	0.929	-0.229	0.817	-0.284
506	0.458	-0.683	0.508	-0.933	0.462	-0.676

**Tabla 26***Comparativa de pendientes del escenario 3A*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
171	0.540	-0.887	0.531	-0.901	0.535	-0.893
395	0.566	-0.912	0.555	-0.873	0.562	-0.912
486	0.593	-0.917	0.602	-0.868	0.593	-0.928
620	0.670	-0.863	0.625	-0.933	0.662	-0.873
952	0.568	-0.987	0.577	-0.901	0.579	-0.985

**Tabla 27***Comparativa de pendientes del escenario 3B*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
239	0.664	-0.829	0.650	-1.000	0.660	-0.832
354	0.421	-0.952	0.424	-0.363	0.436	-0.958
419	0.434	-1.090	0.425	-0.363	0.430	-1.110
568	0.562	-0.884	0.554	-0.901	0.553	-0.876
990	0.522	-0.889	0.509	-0.966	0.516	-0.879

**Tabla 28***Comparativa de pendientes del escenario 4A*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
135	0.731	-0.566	0.752	-0.578	0.712	-0.555
331	0.922	-0.758	0.815	-0.784	0.910	-0.751
354	0.636	-0.611	0.652	-0.615	0.623	-0.627
411	0.508	-0.118	0.528	-0.985	0.502	-0.125
v 589	0.929	-0.818	0.921	-0.814	0.936	-0.820

**Tabla 29***Comparativa de pendientes del escenario 4B*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
304	0.841	-0.780	0.900	-0.725	0.847	-0.767
441	0.634	-0.834	0.624	-1.000	0.622	-0.846
514	0.826	-0.805	0.928	-0.839	0.833	-0.812
694	0.850	-0.721	0.932	-0.726	0.845	-0.739
801	0.771	-0.978	0.781	-0.841	0.767	-0.962

**Tabla 30***Comparativa de pendientes del escenario 5A*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
290	0.924	-0.406	0.899	-0.444	0.915	-0.388
377	0.906	-0.471	0.809	-0.487	0.925	-0.462
822	0.413	-0.964	0.268	-0.305	0.406	-0.956
826	0.324	-0.706	1.272	-0.305	0.332	-0.699
874	0.615	-2.130	0.268	-0.304	0.607	-2.131

**Tabla 31***Comparativa de pendientes del escenario 5B*

No. cuadro	Manual		Sin filtrar		Filtrado	
	$m_i$	$m_d$	$m_i$	$m_d$	$m_i$	$m_d$
254	0.603	-0.873	0.623	-0.952	0.605	-0.853
312	0.749	-0.834	0.715	-0.815	0.733	-0.823
326	0.734	-0.899	0.721	-0.825	0.745	-0.912
344	0.930	-0.722	0.892	-0.625	0.950	-0.714
745	0.795	-0.921	0.752	-0.845	0.812	-0.936

Si los cálculos manuales de  $m_i$  y  $m_d$  son los correctos; se calcula el error medio cuadrático (MSRE) entre las pendientes calculadas manualmente (M) y respectivamente, las pendientes calculadas por el sistema filtradas (F) y sin filtrar (SF). Con o cual se obtiene los resultados de la tabla 32, donde se observa que el MSRE es menor respecto a las pendientes calculadas por el sistema aplicando el filtro en tiempo real.

**Tabla 32***Error cuadrático medio del las pendientes calculadas manualmente respecto a las calculadas por el sistema*

Descripción	MSRE
$m_i$ (M) vs $m_i$ (SF)	0.472
$m_i$ (M) vs $m_i$ (F)	0.010
$m_d$ (M) vs $m_d$ (SF)	0.182
$m_d$ (M) vs $m_d$ (F)	0.011

### 5.2.3.3. Falsos positivos, negativos y sensibilidad del sistema

**Tabla 33**

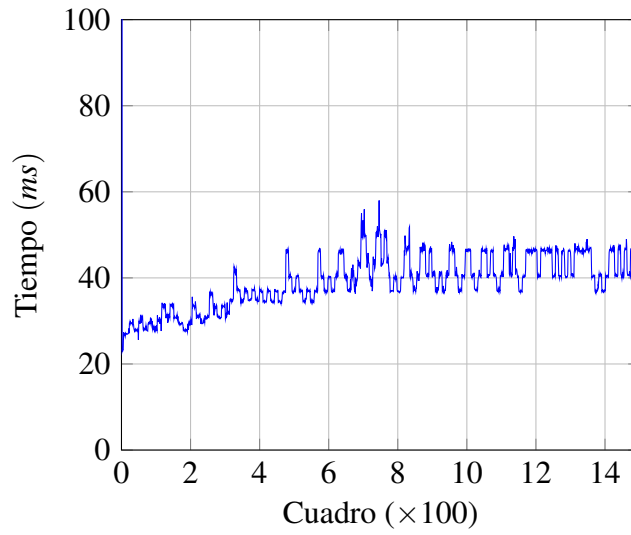
*Falsos positivos, negativos, sensibilidad y especificidad del sistema*

Escenario	VP	VN	FP	FN	% Sensibilidad	% Especificidad
1A	143	2116	26	14	91.083	98.786
1B	263	1832	79	25	91.319	95.866
2A	128	494	7	30	81.013	98.603
2B	260	1099	10	14	94.891	99.098
2C	134	449	10	21	86.452	97.821
3A	102	951	18	10	91.071	98.142
3B	90	1401	36	12	88.235	97.495
4A	136	513	20	19	87.742	96.248
4B	144	1946	14	17	89.441	99.286
5A	219	1698	36	18	92.405	97.924
5B	206	1022	38	22	90.351	96.415
<b>Total</b>	<b>1825</b>	<b>13521</b>	<b>294</b>	<b>202</b>	<b>90.035</b>	<b>97.872</b>

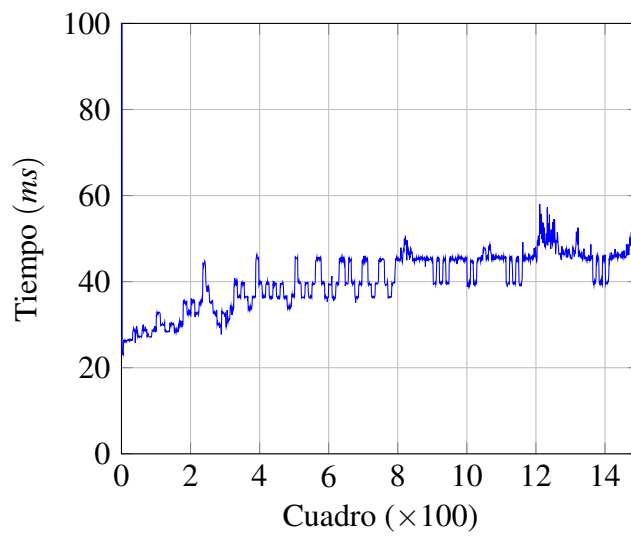
En conclusión, la sensibilidad media del sistema es del 90.035% y la especificidad media es del 97.872%.

## 5.3. Tiempo de procesamiento

Se ha generado una gráfica del tiempo que tarda en procesarse cada cuadro del vídeo por cada caso con una tarjeta gráfica nVidia GTX1660Ti.

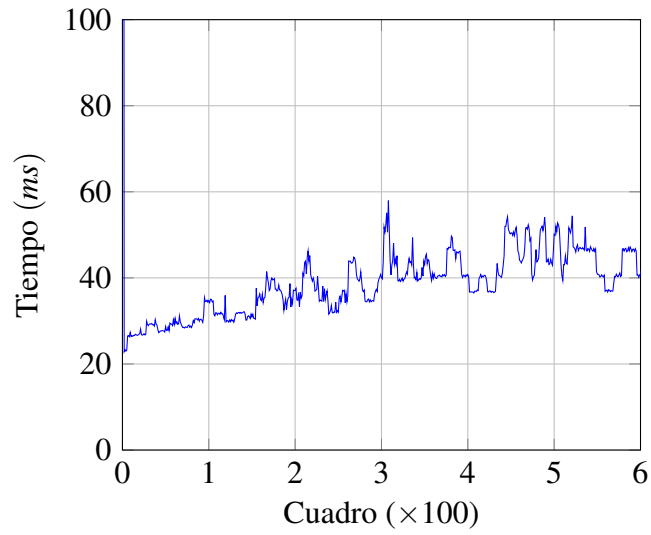


**Figura 52.** Tiempo de procesamiento por cuadro del escenario 1A

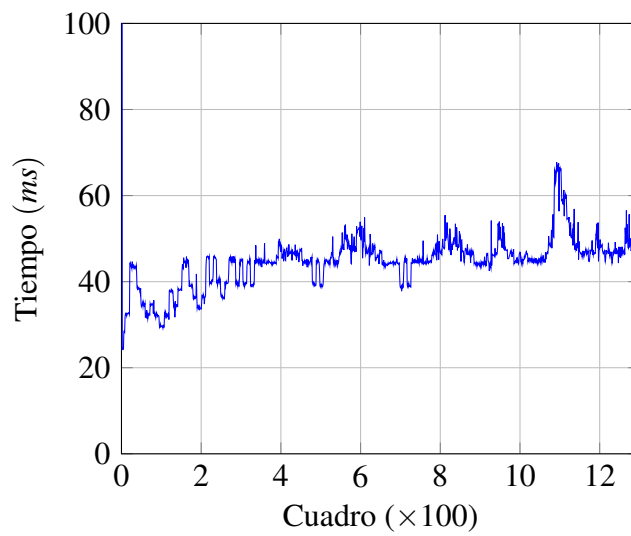


**Figura 53.** Tiempo de procesamiento por cuadro del escenario 1B

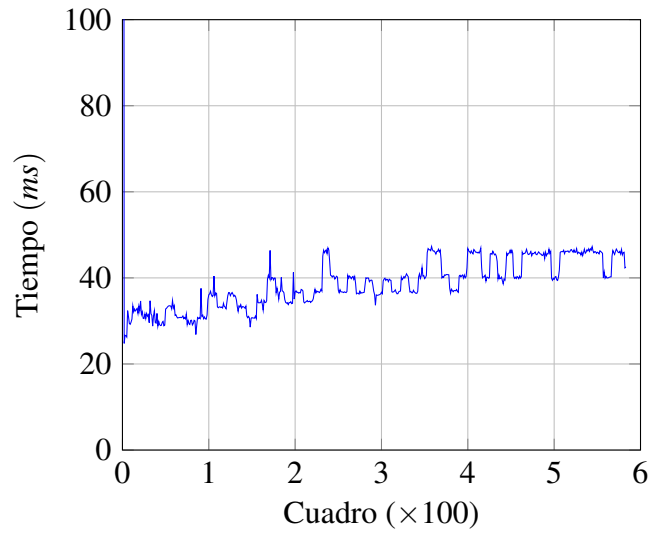




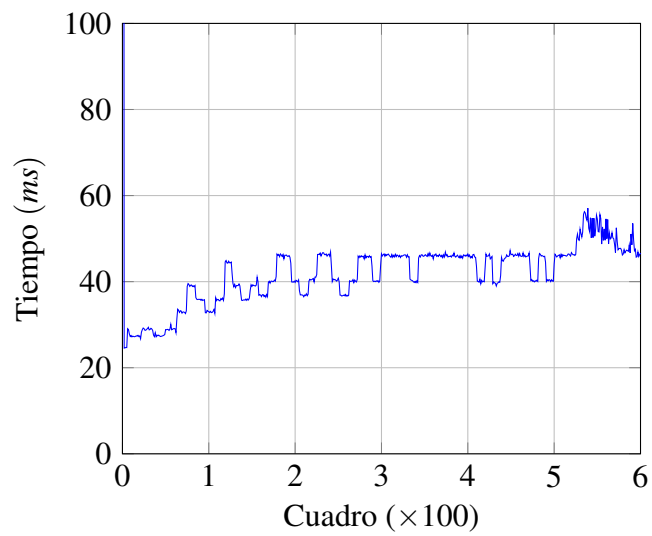
**Figura 54.** Tiempo de procesamiento por cuadro del escenario 2A



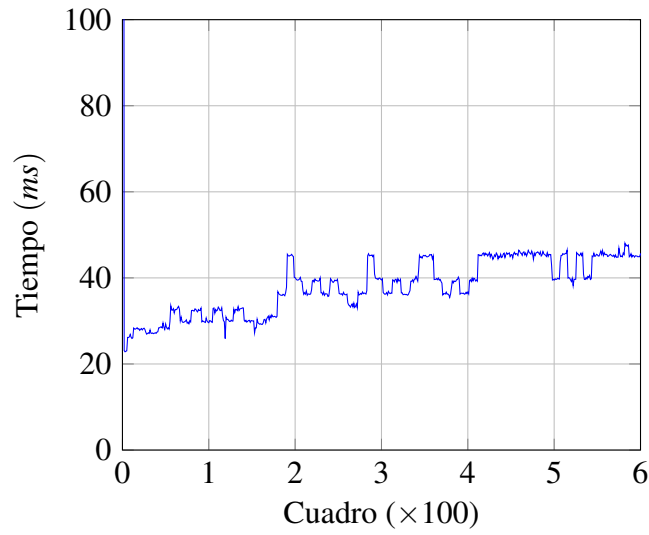
**Figura 55.** Tiempo de procesamiento por cuadro del escenario 2B



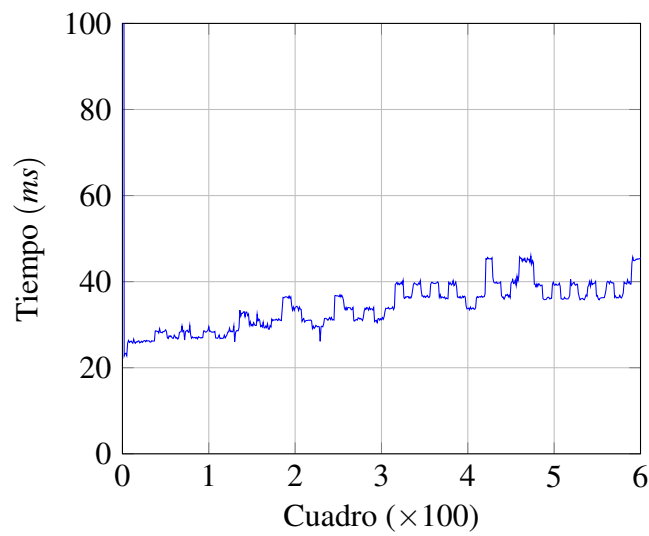
**Figura 56.** Tiempo de procesamiento por cuadro del escenario 2C



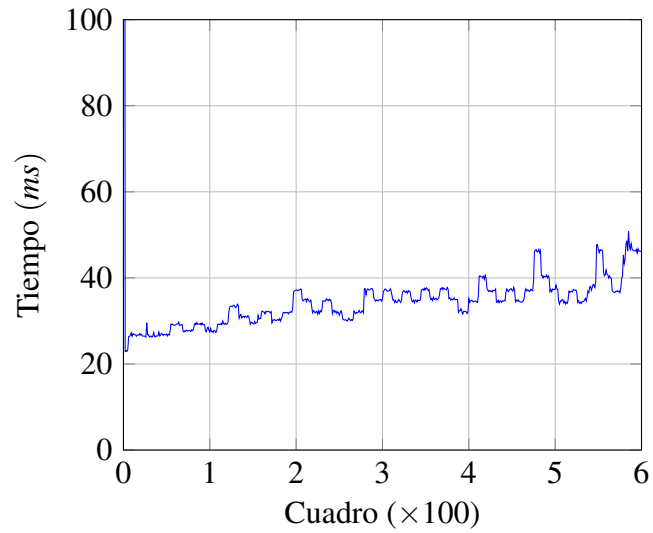
**Figura 57.** Tiempo de procesamiento por cuadro del escenario 3A



**Figura 58.** Tiempo de procesamiento por cuadro del escenario 3B



**Figura 59.** Tiempo de procesamiento por cuadro del escenario 4B



**Figura 60.** Tiempo de procesamiento por cuadro del escenario 5A

En conclusión, el tiempo de procesamiento por cuadro es siempre menor a  $70ms$ . Con lo cual, es sistema es lo suficientemente rápido para alertar a un conductor cuyo tiempo de reacción es de alrededor de  $250ms$ .

## Capítulo 6

# Conclusiones y trabajo futuro

### 6.1. Conclusiones

El uso de redes neuronales artificiales facilitó el funcionamiento del sistema en varios escenarios; es decir, en diferentes tipos de carretera, clima y condiciones de luz.

El tiempo de procesamiento del sistema es de alrededor 40ms; con lo cual es lo suficientemente rápido para alertar al conductor.

A través del uso del filtro en tiempo real se redujo gran cantidad del ruido del comportamiento del vehículo en la calzada; con lo cual se redujeron los falsos positivos y negativos.

El prototipo del sistema de detección de pérdida de vía desarrollado en este proyecto es robusto pues tiene una sensibilidad del 90.035 % y una especificidad del 97.872 %.

## **6.2. Trabajo futuro**

El proyecto desarrollado constituye un prototipo funcional que necesita ser transferido a dispositivos móviles para la distribución al público general. Para lo cual, las librerías de Tensorflow y OpenCV están disponibles para la creación de aplicaciones de Apple iOS y Android, que son los sistemas operativos móviles más populares en la actualidad.

En este proyecto, se realizó el modelado matemático de las líneas de carretera usando un modelo lineal; con lo cual, el mismo necesita ser actualizado a un modelo de mayor orden para un mejor funcionamiento ante situaciones de curvas.

---

## Referencias

- Agencia Nacional de Transito (ANT). (2018a). *Fallecidos Septiembre 2018*. Descargado de <https://www.ant.gob.ec/index.php/descargable/file/5873-fallecidos-septiembre-2018>
- Agencia Nacional de Transito (ANT). (2018b). *Siniestros Septiembre 2018*. Descargado de <https://www.ant.gob.ec/index.php/descargable/file/5872-siniestros-septiembre-2018>
- and Ji-Hun Bae, y Song, J. (2011, noviembre). Monocular vision-based lane detection using segmented regions from edge information. En *Proc. 8th int. conf. ubiquitous robots and ambient intelligence (urai)* (pp. 499–502). doi: 10.1109/URAI.2011.6145871
- Bar Hillel, A., Lerner, R., Levi, D., y Raz, G. (2014, abril). Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, 25(3), 727. Descargado de <http://dx.doi.org/10.1007/s00138-011-0404-2> doi: 10.1007/s00138-011-0404-2
- Bertozzi, M., y Broggi, A. (1998). Gold: A parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE transactions on image processing*, 7(1), 62–81.
- Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., y Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.
- Chen, M., Jochem, T., y Pomerleau, D. (1995). Aurora: A vision-based roadway departure warning system. En *Iros (1)* (pp. 243–248).

- Coley, G., Wesley, A., Reed, N., y Parry, I. (2009). Driver reaction times to familiar, but unexpected events. *TRL Published Project Report*.
- Dai, J., He, K., y Sun, J. (2015). Convolutional feature masking for joint object and stuff segmentation. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3992–4000).
- Dickmanns, E. D., y Mysliwetz, B. D. (1992, febrero). Recursive 3-D road and relative ego-state recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 199–213. doi: 10.1109/34.121789
- Erhan, D., Szegedy, C., Toshev, A., y Anguelov, D. (2014). Scalable object detection using deep neural networks. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2147–2154).
- Felisa, M., y Zani, P. (2010, junio). Robust monocular lane detection in urban environments. En *Proc. ieee intelligent vehicles symp* (pp. 591–596). doi: 10.1109/IVS.2010.5548028
- Galton, F. (1890). Exhibition of instruments (1) for testing perception of differences of tint, and (2) for determining reaction-time. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 19, 27–29.
- González, R. C., y Woods, R. E. (1996). *Tratamiento digital de imágenes* (Vol. 3). Addison-Wesley New York.
- Google Inc. (2019). *Why tensorflow*. Descargado de <https://www.tensorflow.org/about>
- Goro, K., y Onoguchi, K. (2018). Road boundary detection using in-vehicle monocular camera. En *Icpram* (pp. 379–387).
- Hariharan, B., Arbeláez, P., Girshick, R., y Malik, J. (2014). Simultaneous detection and segmentation. En *European conference on computer vision* (pp. 297–312).
- Huan, S., Jianguo, M., y Shunming, L. (2008). Monocular camera machine vision lane recognition algorithm and realization on arm system [j]. *Journal of Nanjing University of Aeronautics & Astronautics*, 2.



- Huang, A. S., Moore, D., Antone, M., Olson, E., y Teller, S. (2009, abril). Finding multiple lanes in urban road networks with vision and lidar. *Autonomous Robots*, 26(2-3), 103. Descargado de <http://dx.doi.org/10.1007/s10514-009-9113-3> doi: 10.1007/s10514-009-9113-3
- Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. En *Advances in neural information processing systems* (pp. 1097–1105).
- Lavin, A., y Gray, S. (2016). Fast algorithms for convolutional neural networks. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4013–4021).
- Long, J., Shelhamer, E., y Darrell, T. (2015). Fully convolutional networks for semantic segmentation. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3431–3440).
- McCall, J. C., y Trivedi, M. M. (2006, marzo). Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1), 20–37. doi: 10.1109/TITS.2006.869595
- Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., y Van Gool, L. (2018). Towards end-to-end lane detection: an instance segmentation approach. En *2018 ieee intelligent vehicles symposium (iv)* (pp. 286–291).
- Noh, H., Hong, S., y Han, B. (2015). Learning deconvolution network for semantic segmentation. En *Proceedings of the ieee international conference on computer vision* (pp. 1520–1528).
- NVIDIA. (2019a). *Deep learning sdk documentation*. Descargado de <https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html>
- NVIDIA. (2019b, agosto). Nvidia cuda installation guide for linux [Manual de software informático].
- NVIDIA. (2019c). *Nvidia cudnn*. Descargado de <https://developer.nvidia.com/cudnn>
- Ogawa, T., y Takagi, K. (2006, junio). Lane recognition using on-vehicle LIDAR. En *Proc. ieee intelligent vehicles symp* (pp. 540–545). doi: 10.1109/IVS.2006.1689684
- OpenCV. (2019). *Opencv about*. Descargado de <https://opencv.org/about/>

- Palomino, N. L. S., y Concha, U. N. R. (2009). Técnicas de segmentación en procesamiento digital de imágenes. *Revista de investigación de Sistemas e Informática*, 6(2), 9–16.
- Petermeijer, S., Doubek, F., y de Winter, J. (2017, octubre). Driver response times to auditory, visual, and tactile take-over requests: A simulator study with 101 participants. En *Proc. and cybernetics (smc) 2017 ieee int. conf. systems, man* (pp. 1505–1510). doi: 10.1109/SMC.2017.8122827
- Qassim, H., Verma, A., y Feinzimer, D. (2018). Compressed residual-vgg16 cnn model for big data places image recognition. En *2018 ieee 8th annual computing and communication workshop and conference (ccwc)* (pp. 169–175).
- Risack, R., Mohler, N., y Enkelmann, W. (2000, octubre). A video-based lane keeping assistant. En *Proc. ieee intelligent vehicles symp. 2000 (cat. no.00th8511)* (pp. 356–361). doi: 10.1109/IVS.2000.898369
- Simonyan, K., y Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.
- Takagi, K., Morikawa, K., Ogawa, T., y Saburi, M. (2006, junio). Road environment recognition using on-vehicle LIDAR. En *Proc. ieee intelligent vehicles symp* (pp. 120–125). doi: 10.1109/IVS.2006.1689615
- Thorpe, C., Herbert, M., Kanade, T., y Shafer, S. (1991, agosto). Toward autonomous driving: the cmu navlab. i. perception. *IEEE Expert*, 6(4), 31–42. doi: 10.1109/64.85919
- Tseng, D.-C. (2004, julio 20). *Monocular computer vision aided road vehicle driving for safety*. Google Patents. (US Patent 6,765,480)
- van der Heiden, R. M., Janssen, C. P., Donker, S. F., y Merckx, C. L. (2018). Visual in-car warnings: How fast do drivers respond? *Transportation Research Part F: Traffic Psychology and Behaviour*.

- 
- Warshawsky-Livne, L., y Shinar, D. (2002). Effects of uncertainty, transmission type, driver age and gender on brake reaction and movement time. *Journal of safety research*, 33(1), 117–128.
- Zhang, W.-B. (1991, octubre). A roadway information system for vehicle guidance/control. En *Proc. vehicle navigation and information systems conf* (Vol. 2, pp. 1111–1116). doi: 10.1109/VNIS.1991.205857
- Zhou, X., Huang, X.-y., y Li, Y. (2003). Lane keeping and distance measurement based on monocular vision [j]. *Journal of Image and Graphics*, 5.
- Zhou, Y., Xu, R., Hu, X., y Ye, Q. (2006). A robust lane detection and tracking method based on computer vision. *Measurement science and technology*, 17(4), 736.
- Žuraulis, V., Nagurnas, S., Pečeliūnas, R., Pumputis, V., y Skačkauskas, P. (2018). The analysis of drivers' reaction time using cell phone in the case of vehicle stabilization task. *International journal of occupational medicine and environmental health*, 1–16.